Predicting the Large-Scale Evolution of Tag Systems

Carlos Martin

School of Engineering and Applied Science
Columbia University
New York, New York

We present a method for predicting the large-scale evolution of a tag system from its production rules. The tag system's evolution is divided into stages called "epochs" in which the tag system evolves monotonously. The distribution of strings of symbols in the queue at the beginning of an epoch determines the large-scale behavior of the tag system during that epoch, including its growth rate. To predict the tag system's large-scale properties over multiple epochs, we show how to predict the next epoch's initial queue contents from the current epoch's initial queue contents. We compare the values predicted by this method to simulations and find that great prediction accuracy is retained over several epochs.

1. Introduction

1.1 Definition of a Tag System

A tag system is a model of computation comprised of a finite state machine and a queue. The queue contains symbols belonging to some alphabet Σ. The finite state machine specifies a production function that maps strings of n symbols (elements of Σ^n) to strings of arbitrary length (elements of Σ^*).

In each step of a computation, n symbols are removed from the front of the queue and the corresponding string from the production function is added to the end of the same queue. This process is repeated until some halting condition is satisfied, such as there being fewer than n symbols in the queue.

Tag systems were created by the mathematician and logician Emil Leon Post, who is best known for his work in computability theory, as an example of a Post canonical system that is deterministic or *monogenic*, meaning that at most one string can be produced from any given string in one step [1, 2].

The emergence of complex behavior in very simple systems has been well documented and explored in detail in Stephen Wolfram's *A New Kind of Science* [3]. Wolfram's Principle of Computational Equivalence states that almost all processes that are not obviously sim-

ple can be viewed as computations of equivalent sophistication. Even small systems can, in principle, compute the same things as any computer, given an appropriate translation of inputs and outputs.

It is therefore not entirely surprising that despite the simplicity of their specification, tag systems have been shown to be capable of universal computation. This result was proven by Wang [4] and by Cocke and Minsky [5] through the construction of a two-tag system that can simulate a universal Turing machine. The search for other small universal tag systems remains open [6, 7].

Another interesting property of tag systems is their connection to problems in number theory. De Mol showed in [6] that the Collatz problem, a well-known unsolved problem in number theory, can be reduced to a small tag system. Furthermore, Conway proved in [8] that a generalization of the Collatz problem is algorithmically undecidable. Other undecidable variants of the Collatz problem are explored in [9]. These examples support the principle of computational irreducibility described by Wolfram in [3], which states that one cannot, in general, shortcut computations performed by sufficiently powerful automata.

Tag systems fall under this category. Although it is generally impossible to predict the exact behavior of tag systems, our study of their large-scale behavior could provide a useful heuristic to guide systematic searches for small tag system rules that can perform interesting computations, including ones connected to problems in number theory, like those of the Collatz problem.

Consider a simple two-tag system with the following production rules:

$$aa \to aab, \;\; ab \to ab, \;\; ba \to b, \;\; bb \to ba. \tag{1}$$

Figure 1 illustrates the evolution of this tag system when initialized with a queue containing 10 symbols (where *a* symbols are colored gray and *b* symbols are colored black).

Figure 1.

For this initial configuration, the tag system eventually reaches a state where it only has one symbol in the queue and thus terminates. Figure 2 illustrates the same tag system under a different initial configuration. For this configuration, the tag system becomes periodic.
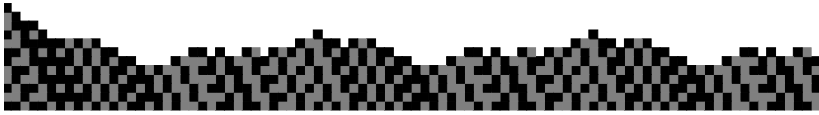
Figure 2.

Now consider a tag system with a different set of production rules:

$$aa \rightarrow aba, \ ab \rightarrow aa, \ ba \rightarrow bbb, \ bb \rightarrow ba. \tag{2}$$

Figure 3 illustrates the evolution of this tag system when initialized with a queue containing a random string of 10 symbols.
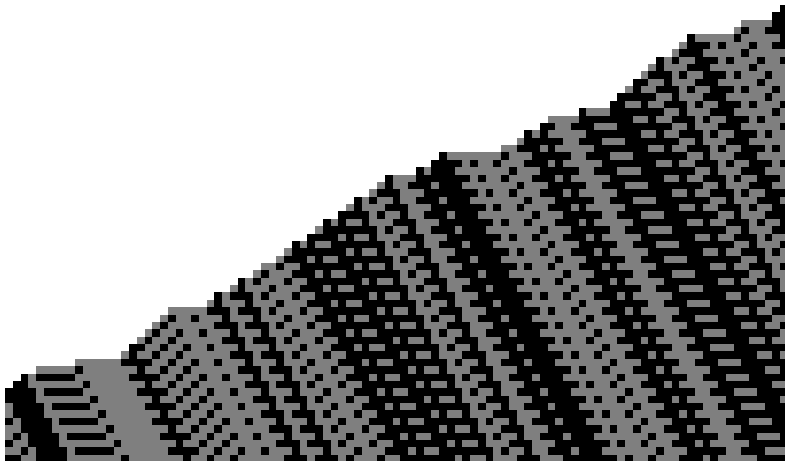

Figure 3.

For this initial configuration, the length of the queue grows without bound. Furthermore, the queue of a tag system with these rules cannot contract, because all productions have two or more symbols. Correspondingly, if all productions had two or fewer symbols, the queue of the tag system could never grow.

▌ 1.2 Examples of Large-Scale Evolution

We can study the properties and behavior of tag systems like these on a much larger scale, revealing interesting statistical properties. The following images illustrate the evolution of various two-tag systems for 5000 steps starting on a random initial state of 1000 symbols. Different random initial states all tend to produce roughly the same shape

for each tag system, indicating that their large-scale behavior is determined primarily by their production rules (Figure 4).

The tag system shown in Figure 5 appears to initially grow at a rate of about one symbol every four steps, before it begins to level off and asymptotically approach a growth rate of zero. Furthermore, we can observe that the density of b symbols in the queue increases sharply beyond this point, while the density of a symbols decreases.

Figure 4. $aa \rightarrow aab, \ ab \rightarrow ab, \ ba \rightarrow b, \ bb \rightarrow ba$.

Figure 5. $aa \rightarrow bb, \ ab \rightarrow bb, \ ba \rightarrow aaa, \ bb \rightarrow bb$.

The tag system shown in Figure 6 undergoes gradually decreasing growth without a sharp phase transition, coupled with a gradually increasing density of b symbols in the queue.
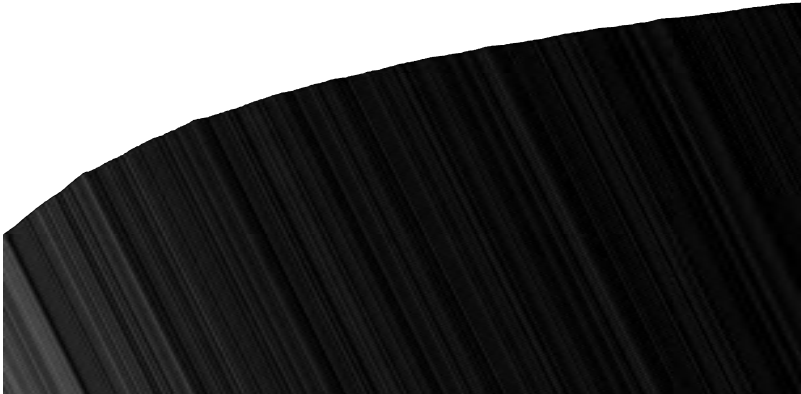

Figure 6. $aa \rightarrow bab, \ ab \rightarrow bbb, \ ba \rightarrow aab, \ bb \rightarrow bb$.

The tag system shown in Figure 7 is initially roughly constant in length, before transitioning to a long-term linear growth rate of approximately one symbol every five steps.

The tag system shown in Figure 8 has a phase of contraction before entering a phase where it remains constant in length and dominated by *a* symbols.
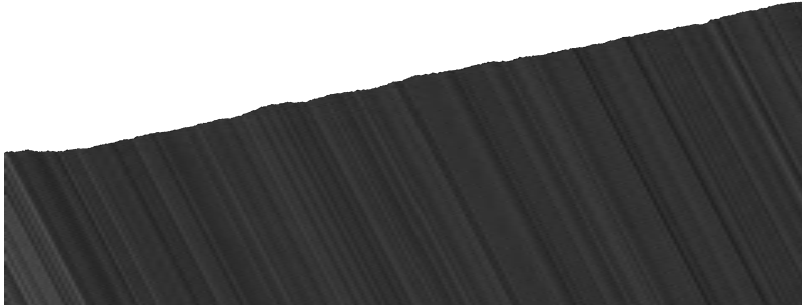

Figure 7. $aa \rightarrow b$, $ab \rightarrow b$, $ba \rightarrow aab$, $bb \rightarrow abb$.

Figure 8. $aa \rightarrow aa$, $ab \rightarrow ba$, $ba \rightarrow \varepsilon$, $bb \rightarrow ab$.

The tag system shown in Figure 9 enters a phase where it becomes entirely dominated by *b* symbols and remains constant in length, before entering another phase where it sharply contracts at a rate of approximately one symbol per step, until a single symbol remains.

Figure 9. $aa \rightarrow bbb$, $ab \rightarrow ab$, $ba \rightarrow bb$, $bb \rightarrow b$.

The tag system shown in Figure 10 at first contracts at a rate of approximately one symbol every two steps. It then transitions to a phase where it gradually starts approaching a growth rate of one symbol per

step, while becoming increasingly dominated by *a* symbols. Its shape is reminiscent of an hourglass.

The tag system shown in Figure 11 exhibits a particularly interesting behavior. The tag system repeatedly alternates between two phases. In the first phase, the tag system becomes dominated by *b* symbols while remaining roughly constant in length. In the second phase, the densities of *a* and *b* symbols become roughly equal while the tag system contracts at a rate of approximately one symbol every two steps.
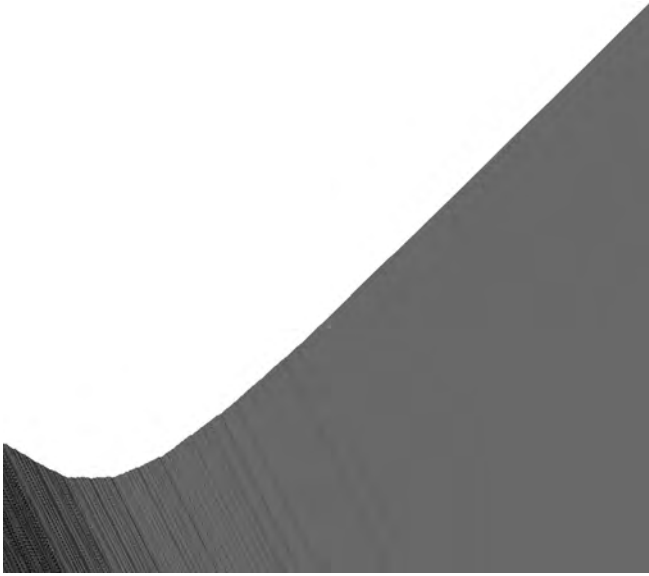

Figure 10. *aa → aaa*, *ab → b*, *ba → a*, *bb → b*.
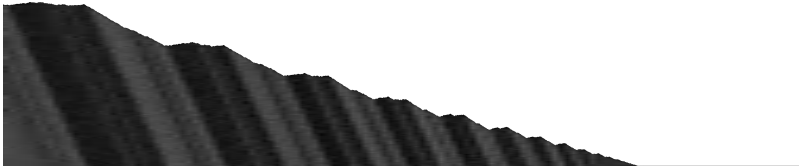
Figure 11. *aa → bbb*, *ab → ab*, *ba → bb*, *bb → a*.

▎ 1.3 Epochs and Phase Transitions

There is a systematic way to study the large-scale growth patterns and behavior of these tag systems, even if they are nonlinear (as demonstrated by some of the previous examples). We will use this approach to formulate an algorithmic procedure for predicting these large-scale properties.

The first insight needed to understand the large-scale evolution of these tag systems is that the history or evolution of the tag system can be divided naturally into distinct stages, which we will refer to as *epochs*.

The beginning of a new epoch occurs when all the symbols belonging to the previous epoch have been consumed by the tag system (i.e., removed from the queue). Figure 12 illustrates the evolution of a tag system where symbols in the queue are colored according to which epoch they were produced in.

Figure 13 illustrates the same evolution, while highlighting the beginning of each epoch with a vertical line.

Notice that the length of the queue tends to change in a relatively linear manner within each epoch. The same can be said of the previous examples of two-tag systems. Figure 14 illustrates the evolution of a tag system with decelerating growth.
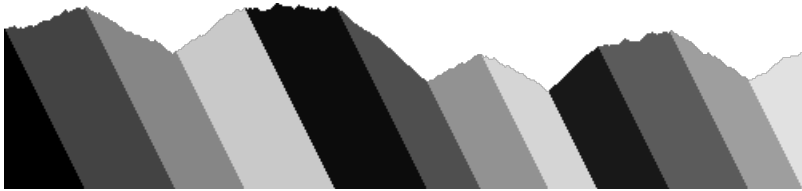

Figure 12.

Figure 13.

Figure 14.

Figure 15 illustrates a tag system that exhibits a strong phase transition from contraction to growth on the second epoch.
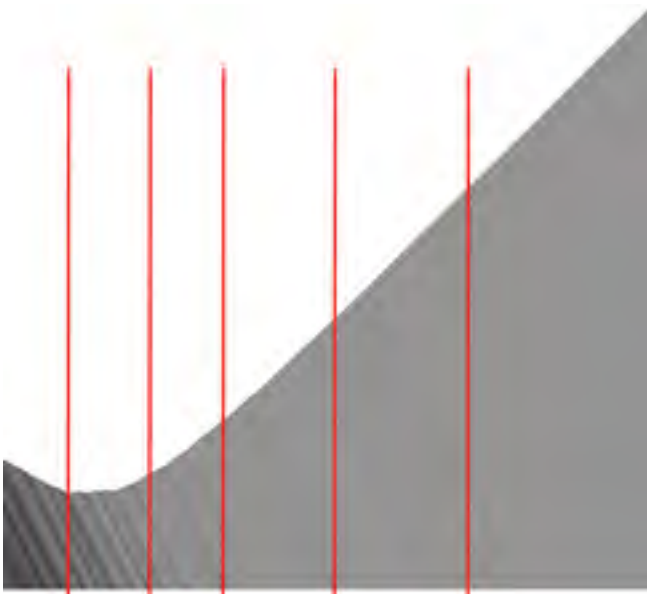

Figure 15.

Figure 16 illustrates a tag system that repeatedly transitions between a phase of constant length and a phase of constant contraction.
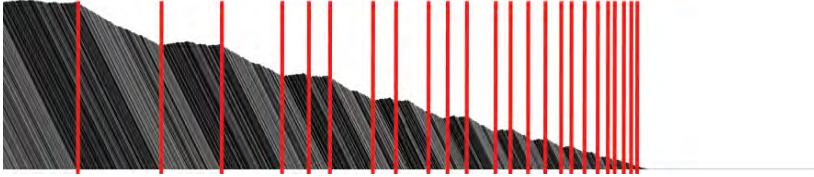
Figure 16.

The remainder of this paper will make the assumption that the distributions of symbols and strings of symbols on the queue are stationary along its length, meaning they do not change significantly from the beginning to the end of the queue. If a distribution is stationary at the beginning of an epoch, it will likely remain stationary at the beginning of the next epoch, since the latter is generated from the former by the same set of production rules. The stationarity assumption means that the large-scale properties of a tag system tend to change linearly within epochs.

The second key insight needed to predict the large-scale evolution of a tag system is the following: the distribution of symbols on the queue for the beginning of an epoch is entirely a function of the distribution of symbols on the queue for the beginning of the previous epoch.

In particular, the distribution of *tuples* of symbols on the queue for the beginning of an epoch can be used to determine the distribution of *strings* of symbols for the next epoch (by analyzing the production rules of the tag system). From this string distribution, one can determine the distribution of *tuples* of symbols, which can, in turn, be used to determine the string distribution for the *third* epoch, and so on. This procedure follows.

```
word_probs[0] = initial_word_probs
for epoch in range (epochs):
    prod_probs[epoch + 1] =
get_prod_probs(word_probs[epoch])
    word_probs[epoch + 1] =
get_word_probs(prod_probs[epoch + 1])
```

where `word_probs` is the distribution of n-tuples on the tag system queue and `prod_probs` is the distribution indicating the probability of different string productions being generated at the beginning of an epoch.

In the next section, we will describe how to find the production distribution from the tuple distribution and, with more difficulty, the tuple distribution from the production distribution.

2. Derivation

2.1 Generating a Production

Let p be the contents of the queue at the beginning of an epoch:

$$p = q_1 \cdot q_2 \cdot q_3 \cdot \cdots \cdot q_m, \tag{3}$$

where $q_i \in \Sigma^n$ and $\cdot$ denotes concatenation. The production rules constitute a function f that maps strings of length n to strings of arbitrary length:

$$f : \Sigma^n \to \Sigma^*. \tag{4}$$

Hence the contents of the queue at the beginning of the next epoch are

$$\begin{aligned} p &= f(q_1) \cdot f(q_2) \cdot f(q_3) \cdot \cdots \cdot f(q_m) \\ &= r_1 \cdot r_2 \cdot r_3 \cdot \cdots \cdot r_m, \end{aligned} \tag{5}$$

where $r_i \in \Sigma^*$. The concatenation of these productions, in turn, determines the productions that are generated in the epoch after the next one. This process is illustrated in Figure 17.
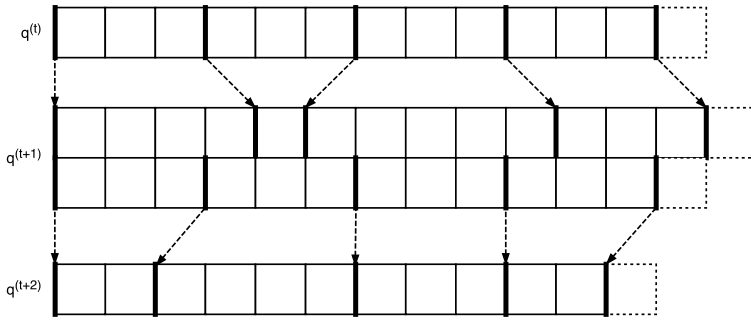
Figure 17.

The probability that $r_i = s$ for some $s \in \Sigma^*$ is

$$\begin{aligned} P(r_i = s) &= P(f(q_i) = s) = P\left(\bigcup_{t \in \Sigma^n} q_i = t \cap f(q_i) = s\right) = \\ &\sum_{t \in \Sigma^n} P(q_i = t \cap f(q_i) = s) = \\ &\sum_{t \in \Sigma^n} P(q_i = t) P(f(q_i) = s \mid q_i = t) = \\ &\sum_{t \in \Sigma^n} P(q_i = t) P(f(t) = s) = \sum_{t \in \Sigma^n} P(q_i = t)[f(t) = s], \end{aligned} \tag{6}$$

since $P(f(t) = s) \in \{0, 1\}$ if the tag system is deterministic. Consequently, given the distribution of strings of length n at the begin-

ning of an epoch, one can determine the distribution of strings produced during that epoch. Because the queue at the beginning of the next epoch consists of the concatenation of these productions, one can theoretically determine the distribution of strings of length n at the beginning of the next epoch as well.

▌ 2.2 Selecting a Production Instance

Let i be a position selected uniformly at random from p. Recall the definition of a uniform distribution:

$$X \sim \mathcal{U}(S) \Longleftrightarrow (\forall\; R \subset S)\left(P(X \in R) = \frac{\mu(R)}{\mu(S)}\right), \tag{7}$$

where X is the random variable, S is the sample space, and μ is a measure. In our case, $X = i$ and $S = [0, p)$:

$$P(i \in R) = \frac{\mu(R)}{\mu([0, p))}. \tag{8}$$

Let $i \circ j$ be the statement that i belongs to r_j:

$$\begin{aligned}
i \circ j &\Longleftrightarrow |r_1 \cdots \cdots r_{j-1}| \le i < |r_1 \cdots \cdots r_{j-1} \cdot r_j| \\
&\Longleftrightarrow i \in \big[|r_1 \cdots \cdots r_{j-1}|, |r_1 \cdots \cdots r_{j-1} \cdot r_j|\big).
\end{aligned} \tag{9}$$

The probability of this is

$$\begin{aligned}
P(i \circ j) &= P\big(i \in \big[|r_1 \cdots \cdots r_{j-1}|, |r_1 \cdots \cdots r_{j-1} \cdot r_j|\big)\big) = \\
&\frac{\mu\big(\big[|r_1 \cdots \cdots r_{j-1}|, |r_1 \cdots \cdots r_{j-1} \cdot r_j|\big)\big)}{\mu([0, p))} = \\
&\frac{|r_1 \cdots \cdots r_{j-1} \cdot r_j| - |r_1 \cdots \cdots r_{j-1}|}{|p|} = \frac{|r_j|}{|p|}.
\end{aligned} \tag{10}$$

Therefore, the probability of selecting a position that belongs to r_j is proportional to the length of r_j (Figure 18). In other words, longer production instances are more likely to contain the position that was selected at random.
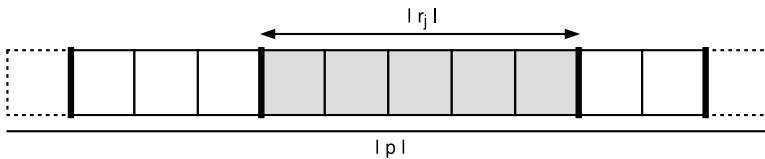

Figure 18.

▍ 2.3 Selecting a Production Type

Let $i \lhd s$ be the statement that i belongs to some r_j equal to s:

$$i \lhd s \Longleftrightarrow \bigcup_{j \in [1,m]} i \circ j \cap r_j = s. \tag{11}$$

The probability of this is

$$
\begin{aligned}
\mathrm{P}(i \lhd s) &= \mathrm{P}\!\left(\bigcup_{j \in [1,m]} i \circ j \cap r_j = s\right) = \\
&\sum_{j \in [1,m]} \mathrm{P}\!\left(i \circ j \cap r_j = s\right) = \sum_{j \in [1,m]} \mathrm{P}(r_j = s)\mathrm{P}(i \circ j \mid r_j = s) = \\
&\sum_{j \in [1,m]} \mathrm{P}(r_j = s)\frac{|s|}{|p|} = \frac{|s|}{|p|}\sum_{j \in [1,m]} \mathrm{P}(r_j = s) = \\
&\mathrm{P}(r_j = s)\frac{|s|}{|p|}\sum_{j \in [1,m]} 1 = \mathrm{P}(r_j = s)\frac{|s|}{|p|}m.
\end{aligned}
\tag{12}
$$

$\mathrm{P}(r_j = s)$ is factored from the sum because it is independent of j. Intuitively, the production type s is expected to appear $\mathrm{P}(r_j = s)m$ times in p, where m is the total number of productions. Hence symbols belonging to s are expected to appear $\mathrm{P}(r_j = s)m|s|$ times. The probability that a randomly selected symbol belongs to s is the ratio of this quantity to the total number of symbols, which is the length of p. The expected length of p can be found as follows:

$$
\begin{aligned}
\mathrm{E}[|p|] &= \mathrm{E}[|r_1 \cdot r_2 \cdot r_3 \cdots|] = \mathrm{E}\!\left[\sum_{j \in [1,m]} |r_j|\right] = \\
&\sum_{j \in [1,m]} \mathrm{E}[|r_j|] = \mathrm{E}[|r_j|]\sum_{j \in [1,m]} 1 = \mathrm{E}[|r_j|]m,
\end{aligned}
\tag{13}
$$

where $\mathrm{E}[|r_j|]$ is factored from the sum because it is independent of j. Recall that the expected value of a function f of a random variable X is

$$\mathrm{E}[f(X)] = \sum_{x \in \Omega} \mathrm{P}(X = x)f(x), \tag{14}$$

where Ω is the sample space. Hence

$$\mathrm{E}[|r_j|] = \sum_{s' \in \Sigma^*} \mathrm{P}(r_j = s')|s'|, \tag{15}$$

and the expected length of p is

$$\mathrm{E}[|p|] = m \sum_{s' \in \Sigma^*} \mathrm{P}(r_j = s')|s'|. \tag{16}$$

Therefore

$$P(i \triangleleft s) = P(r_j = s)\frac{|s|}{|p|}m =$$
$$\frac{P(r_j = s)|s|m}{m\sum_{s' \in \Sigma^*} P(r_j = s')|s'|} = \frac{P(r_j = s)|s|}{\sum_{s' \in \Sigma^*} P(r_j = s')|s'|}. \tag{17}$$

In other words, the probability of selecting a position belonging to a production type is dependent on both the length of the production and the probability of that production being generated.

▎ 2.4 Selecting a String Given a Production

For any string s, let $s_{a:b}$ be the substring of s starting at position a (inclusive) and ending at position b (exclusive):

$$s_{a:b} = s_a s_{a+1} s_{a+2} \cdots s_{b-2} s_{b-1}. \tag{18}$$

For all $r \in \Sigma^n$, one can determine the probability that $q_{i:i+|r|} = r$ as follows:

$$P(q_{i:i+|r|} = r) = P\left(\bigcup_{s \in \Sigma^*} q_{i:i+|r|} = r \cap i \triangleleft s\right) =$$
$$\sum_{s \in \Sigma^*} P(q_{i:i+|r|} = r \cap i \triangleleft s) = \tag{19}$$
$$\sum_{s \in \Sigma^*} P(i \triangleleft s)P(q_{i:i+|r|} = r \mid i \triangleleft s).$$

The conditional probability $P(q_{i:i+|r|} = r \mid i \triangleleft s)$ depends on which position j in s has been selected, and can be determined as follows:

$$P(q_{i:i+|r|} = r \mid i \triangleleft s) = P\left(\bigcup_{j \in [0,|s|)} q_{i:i+|r|} = r \cap j\right) =$$
$$\sum_{j \in [0,|s|)} P(q_{i:i+|r|} = r \cap j) = \sum_{j \in [0,|s|)} P(j)P(q_{i:i+|r|} = r \mid j). \tag{20}$$

Because i is selected uniformly at random from p, j is also selected uniformly at random from s:

$$P(q_{i:i+|r|} = r \mid i \triangleleft s) = \sum_{j \in [0,|s|)} P(j)P(q_{i:i+|r|} = r \mid j) =$$
$$\sum_{j \in [0,|s|)} \frac{1}{|s|} P(q_{i:i+|r|} = r \mid j) = \frac{1}{|s|} \sum_{j \in [0,|s|)} P(q_{i:i+|r|} = r \mid j). \tag{21}$$

To determine $P(q_{i:i+|r|} = r \mid j)$, first consider whether $j + |r| < |s|$ or $j + |r| \geq |s|$ (Figure 19). If $j + |r| < |s|$, check whether $s_{j:j+|r|} = r$.
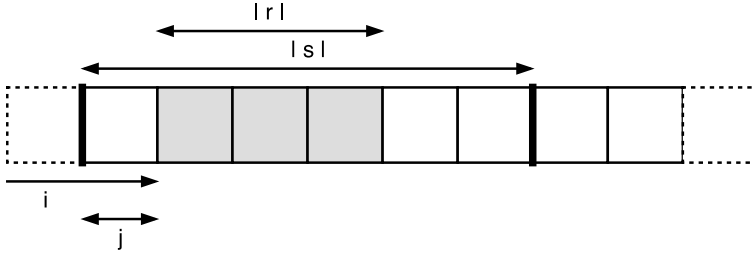
Figure 19.

If $j + |r| \geq |s|$, check whether $s_{j:|s|} = r_{0:|s|-j}$ *and* whether the rest of p begins with the remaining substring of r (or $p_{i-j+|s|:i+|r|} = r_{|s|-j:|r|}$) (Figure 20).
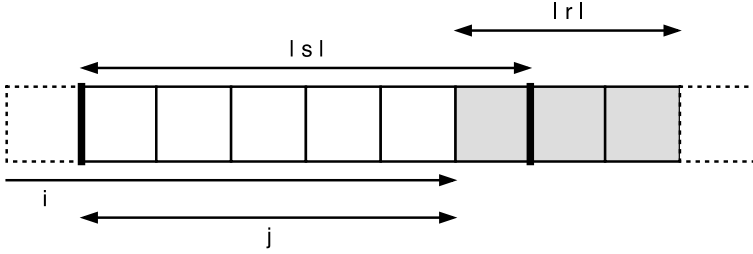
Figure 20.

In summary:

$$p_{i:i+|r|} = r \Leftrightarrow$$
$$\begin{cases} s_{j:j+|r|} = r & j + |r| < |s| \\ s_{j:|s|} = r_{0:|s|-j} \cap p_{i-j+|s|:i+|r|} = r_{|s|-j:|r|} & j + |r| \geq |s|. \end{cases} \tag{22}$$

Therefore

$$\mathrm{P}(p_{i:i+|r|} = r) \Leftrightarrow$$
$$\begin{cases} 1 & j + |r| < |s| \cap s_{j:j+|r|} = r \\ \mathrm{P}(p_{i-j+|s|:i+|r|} = r_{|s|-j:|r|}) & j + |r| \geq |s| \cap s_{j:|s|} = r_{0:|s|-j}. \end{cases} \tag{23}$$

▌ 2.5 Production Sequence Beginning with a String

Let $p' = p_{i-j+|s|:|p|}$ denote the rest of p, that is, the substring of p that consists of the concatenation of all the production instances following the production instance that contained the selected position i:

$$p' = r_j \cdot r_{j+1} \cdot r_{j+2} \cdots \tag{24}$$

The probability that p' begins with a string t is

$$
\begin{aligned}
P\big(p'_{0:|t|} = t\big) &= P\Big(p'_{0:|t|} = t \cap \bigcup_{s \in \Sigma^*} r_j = s\Big) = \\
&\quad P\Big(\bigcup_{s \in \Sigma^*} p'_{0:|t|} = t \cap r_j = s\Big) = \\
&\quad \sum_{s \in \Sigma^*} P\big(p'_{0:|t|} = t \cap r_j = s\big) = \\
&\quad \sum_{s \in \Sigma^*} P\big(r_j = s\big) P\big(p'_{0:|t|} = t \mid r_j = s\big).
\end{aligned} \tag{25}
$$

To determine $P\big(p'_{0:|t|} = t \mid r_j = s\big)$, first consider whether $|t| \le |s|$ or $|t| > |s|$. If $|t| \le |s|$, then the sequence begins with t if the first $|t|$ symbols of s match t. Alternatively, if $|t| > |s|$, then the sequence begins with t if the first $|s|$ symbols of t match s *and* the rest of the sequence begins with the remaining symbols of t. This procedure is illustrated in Figure 21.
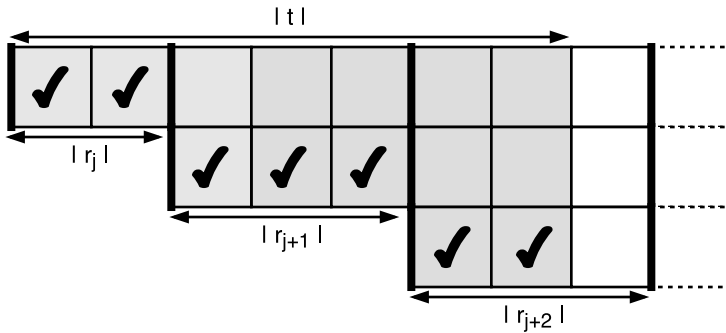
Figure 21.

This means the conditional probability can be expressed as

$$
P\big(p'_{0:|t|} = t \mid r_j = s\big) =
\begin{cases}
1 & |t| \le |s| \cap s_{0:|t|} = t \\
P\big(p'_{|s|:|t|} = t_{|s|:|t|}\big) & |t| > |s| \cap s = t_{0:|s|} \\
0 & \text{otherwise.}
\end{cases} \tag{26}
$$

Because $r_j = s$, it is the case that

$$
\begin{aligned}
p'_{|s|:|t|} &= \big(r_j \cdot r_{j+1} \cdot r_{j+2} \cdot \ldots\big)_{|s|:|t|} = \big(r_j \cdot r_{j+1} \cdot r_{j+2} \cdot \ldots\big)_{|r_j|:|t|} = \\
&\quad \big(r_{j+1} \cdot r_{j+2} \cdot \ldots\big)_{0:|t|-|r_j|} = \big(r_{j+1} \cdot r_{j+2} \cdot \ldots\big)_{0:|t|-|s|}.
\end{aligned} \tag{27}
$$

But if $P(r_j = s)$ is the same as $P(r_{j+1} = s)$ for all j (under the assumption that the distribution of strings on the queue is stationary), then

$$P(p'_{|s|:|t|} = t_{|s|:|t|}) = P((r_{j+1} \cdot r_{j+2} \cdots)_{0:|t|-|s|} = t_{|s|:|t|}) = \\ P((r_j \cdot r_{j+1} \cdots)_{0:|t|-|s|} = t_{|s|:|t|}) = P(p'_{0:|t|-|s|} = t_{|s|:|t|}). \tag{28}$$

In other words, the probability that the rest of the sequence begins with the rest of t is the same as the probability that the original sequence begins with the rest of t (under the assumption of stationarity). This result provides us with a recursive formula for calculating the probability that a string formed by a sequence of concatenated productions begins with a particular string:

$$P(p'_{0:|t|} = t) = \sum_{s \in \Sigma^*} P(r_j = s) P(p'_{0:|t|} = t \mid r_j = s) = \\ \sum_{s \in \Sigma^*} P(r_j = s) \begin{cases} 1 & |t| \leq |s| \cap s_{0:|t|} = t \\ P(p'_{|s|:|t|} = t_{|s|:|t|}) & |t| > |s| \cap s = t_{0:|s|} = \\ 0 & \text{otherwise} \end{cases} \\ \sum_{s \in \Sigma^*} P(r_j = s) \begin{cases} 1 & |t| \leq |s| \cap s_{0:|t|} = t \\ P(p'_{0:|t|-|s|} = t_{|s|:|t|}) & |t| > |s| \cap s = t_{0:|s|} \\ 0 & \text{otherwise.} \end{cases} \tag{29}$$

In the next subsection, we will present the simplified version of this procedure for a two-tag system together with an example of its application.

2.6 Case for a Two-Tag System

Consider the probability that a pair of adjacent symbols randomly selected from p corresponds to π, where $\pi = \pi_0 \pi_1 \in \Sigma^2$. This probability is

$$P(p_i p_{i+1} = \pi) = P\left(p_i p_{i+1} = \pi \cap \bigcup_{s \in \Sigma^*} i \lhd s\right) = \\ P\left(\bigcup_{s \in \Sigma^*} p_i p_{i+1} = \pi \cap i \lhd s\right) = \\ \sum_{s \in \Sigma^*} P(p_i p_{i+1} = \pi \cap i \lhd s) = \\ \sum_{s \in \Sigma^*} P(i \lhd s) P(p_i p_{i+1} = \pi \mid i \lhd s). \tag{30}$$

In the previous section it was shown that

$$P(i \lhd s) = \frac{P(r_j = s)|s|}{\sum_{s' \in \Sigma^*} P(r_j = s')|s'|}. \tag{31}$$

Hence what remains to be determined is $P(p_i p_{i+1} = \pi \mid i \lhd s)$, or the probability of selecting a particular adjacent pair of symbols π, given

that the first element of that pair belongs to some production equal to s.

The first element of the pair could be found in any position in s. Because i is distributed randomly and uniformly over p, the position of the selected element is also distributed randomly and uniformly over s.

For example, if $s = bbb$, then $\underline{b}bb$, $b\underline{b}b$, or $bb\underline{b}$, where the underline indicates which position has been selected. Since i is uniformly distributed over every symbol position in p, each symbol position in s is equally likely to be selected by i, given that $i \lhd s$.

Consider $s = bbb$ once again. Since every symbol in s is b, then

$$P(p_i = b \mid i \lhd bbb) = 1. \tag{32}$$

What is the probability that $p_{i+1} = b$? We know that

$$P(p_{i+1} = b \mid \underline{b}bb) = P(p_{i+1} = b \mid b\underline{b}b) = 1, \tag{33}$$

since the symbol following the first and second positions in bbb is always b. But what about $P(p_{i+1} = b \mid bb\underline{b})$, namely, the probability that b follows the *last* symbol in s? Clearly, in this case, p_{i+1} will be a symbol belonging to the next production, not the current one.

We can determine $P(p_{i+1} = b \mid bb\underline{b})$ by considering the *first* symbol of every subsequent production, weighed by the probability of that production.

For example, suppose that

$$\begin{aligned}
P(r_j = bbb) &= 1/4, \\
P(r_j = ab) &= 1/4, \\
P(r_j = bb) &= 1/4, \\
P(r_j = a) &= 1/4.
\end{aligned} \tag{34}$$

We then know that

$$\begin{aligned}
P(i \lhd bbb) &= 3/8, \\
P(i \lhd ab) &= 2/8, \\
P(i \lhd bb) &= 2/8, \\
P(i \lhd a) &= 1/8.
\end{aligned} \tag{35}$$

Hence

$$\begin{aligned}
P(p_{i+1} = b \mid bb\underline{b}) &= \sum_{s \in \Sigma^*} [s_0 = b] P(r_j = s) = \\
P(r_j = bbb) &+ P(r_j = bb) = 1/4 + 1/4 = 1/2.
\end{aligned} \tag{36}$$

We can then calculate $P(p_{i+1} = b \mid i \lhd bbb)$ as

$$
\begin{aligned}
P&(p_{i+1} = b \mid i \lhd bbb) = \\
&P(p_{i+1} = b \cap (\underline{bb}b \cup b\underline{b}b \cup bb\underline{b})) = \\
&1/3 + 1/3 + 1/3 \cdot P(p_{i+1} = b \mid bb\underline{b}) = \\
&1/3 + 1/3 + 1/6 = 5/6.
\end{aligned}
\tag{37}
$$

Therefore $P(p_i p_{i+1} = bb \mid i \lhd bbb) = 5/6$. In general,

$$
P(p_i = \pi_0 \mid i \lhd s) = \frac{1}{|s|} \sum_{k \in [0, |s|)} [s_k = \pi_0]
\tag{38}
$$

and

$$
\begin{aligned}
P&(p_i p_{i+1} = \pi \mid i \lhd s) = \\
&\frac{1}{|s|} \left([s_{|s|-1} = \pi_0] B_{\pi_1} + \sum_{k \in [0, |s|-1)} [s_k s_{k+1} = \pi] \right),
\end{aligned}
\tag{39}
$$

where B_{π_1} is the probability that the next production begins with π_1:

$$
\begin{aligned}
B_\beta &= P((r_j)_0 = \pi_1) = P\left((r_j)_0 = \pi_1 \cap \bigcup_{s' \in \Sigma^*} r_j = s'\right) = \\
&P\left(\bigcup_{s' \in \Sigma^*} (r_j)_0 = \pi_1 \cap r_j = s'\right) = \\
&\sum_{s' \in \Sigma^*} P((r_j)_0 = \pi_1 \cap r_j = s') = \\
&\sum_{s' \in \Sigma^*} P(r_j = s') P((r_j)_0 = \pi_1 \mid r_j = s') = \\
&\sum_{s' \in \Sigma^*} P(r_j = s') [s'_0 = \pi_1].
\end{aligned}
\tag{40}
$$

It is possible to determine some properties of the tag system during a particular epoch based on the initial pair probability distribution of that epoch. For example, we can determine the expected queue growth per step by calculating the expected length of the production distribution:

$$
E[|r_j|] = \sum_{s' \in \Sigma^*} P(r_j = s') |s'|
\tag{41}
$$

and subtracting n, since n symbols are deleted at each step.

We can also determine the density of different symbols at the beginning of this epoch from the pair distribution by counting the occurrences of those symbols in each production and weighing the counts by the probability of that production being generated, then summing over all productions.

2.7 Expected Length of the Queue

It is possible to estimate the length of the tag system's queue at any step along its evolution using the methods we have developed.

For an n-tag system, the duration or number of steps in an epoch is the length of the queue at the beginning of that epoch divided by n, since n symbols are removed from the front of the queue in each step of the computation.

The expected change in queue length during an epoch is the expected growth per step during that epoch times the number of steps. Hence the expected length of the queue at the beginning of the next epoch is given by

$$
\begin{aligned}
\text{next length} &= \text{current length} + \text{length change} \\
&= \text{current length} + \text{length change per step} \times \text{steps} \\
&= \text{current length} + \text{length change per step} \times \\
&\qquad \frac{\text{current length}}{n} \\
&= \text{current length} \times \left(1 + \frac{\text{length change per step}}{n}\right).
\end{aligned}
\tag{42}
$$

For example, the predicted queue length at the beginning of every epoch for a particular set of rules, starting with 100 symbols in the queue, is given in Table 1.

$aa \to aaa,\ ab \to b,\ ba \to a,\ bb \to b$							
Epoch	0	1	2	3	4	5	6
Growth	−0.500	0.000	0.444	0.750	0.904	0.966	0.989
Length	100.00	75.00	75.00	91.65	126.02	182.98	271.36

Table 1.

3. Algorithm

3.1 Two-Tag System Simulator

The following program, written in the Python 3 programming language, simulates a two-tag system with the set of production rules

$$
aa \to bbb,\ ab \to ab,\ ba \to bb,\ bb \to a
\tag{43}
$$

for 10 epochs, starting with a random configuration of 10 000 symbols. The program prints the density of each symbol in the queue as well as the length of the queue at the beginning of each epoch.

```python
import random

# Returns the production to be appended to the queue
def get_prod(word):
```

```
  if word == (0, 0):
    return (1, 1, 1)
  if word == (0, 1):
    return (0, 1)
  if word == (1, 0):
    return (1, 1)
  if word == (1, 1):
    return (0,)

# Creates a queue with a random configuration of specified
length
def create_queue(length):
  return tuple(random.choice([0, 1]) for n in
range(length))

# Updates the queue by deleting two symbols from the front
and
# appending a corresponding production to the back
def update(queue):
  return queue[2:] + get_prod(queue[:2])

# Update the queue until all original symbols have been
deleted
def update_epoch(queue):
  for n in range(int(len(queue)/2)):
    queue = update(queue)
  return queue

epochs = 10
queue = create_queue(10000)

for epoch in range(epochs):
  print('Epoch ' + str(epoch))
  print('Length: ' + str(len(queue)))
  print('Density of a symbols: ' +
str(queue.count(0)/len(queue)))
  print('Density of b symbols: ' +
str(queue.count(1)/len(queue)))
  print('')
  queue = update_epoch(queue)
```

An example output for this particular set of rules follows.

```
Epoch 0
Length: 10000
Density of a symbols: 0.5006
Density of b symbols: 0.4994

Epoch 1
Length: 10006
Density of a symbols: 0.24475314811113333
Density of b symbols: 0.7552468518888666

Epoch 2
Length: 7452
Density of a symbols: 0.5017444981213097
Density of b symbols: 0.4982555018786903
```

```
Epoch 3
Length: 7465
Density of a symbols: 0.2505023442732753
Density of b symbols: 0.7494976557267247


Epoch 4
Length: 5603
Density of a symbols: 0.5059789398536498
Density of b symbols: 0.49402106014635017


Epoch 5
Length: 5637
Density of a symbols: 0.24942345219088166
Density of b symbols: 0.7505765478091183


Epoch 6
Length: 4224
Density of a symbols: 0.506155303030303
Density of b symbols: 0.49384469696969696


Epoch 7
Length: 4250
Density of a symbols: 0.24776470588235294
Density of b symbols: 0.7522352941176471


Epoch 8
Length: 3178
Density of a symbols: 0.4977973568281938
Density of b symbols: 0.5022026431718062


Epoch 9
Length: 3171
Density of a symbols: 0.25449385052034057
Density of b symbols: 0.7455061494796594
```

The following Python code

```python
import matplotlib.pyplot as plt
plt.title('Tag system simulations')
plt.xlabel('Step')
plt.ylabel('Queue length')
for trial in range(100):
  lengths = []
  queue = create_queue(10000)
  while len(queue) >= 2:
    lengths.append(len(queue))
    queue = update(queue)
  plt.plot(lengths, ',', color='.75')
plt.show()
```

plots the queue length at every step for 100 tag systems with the same rule set but with different initial conditions. The plot is shown in Figure 22.
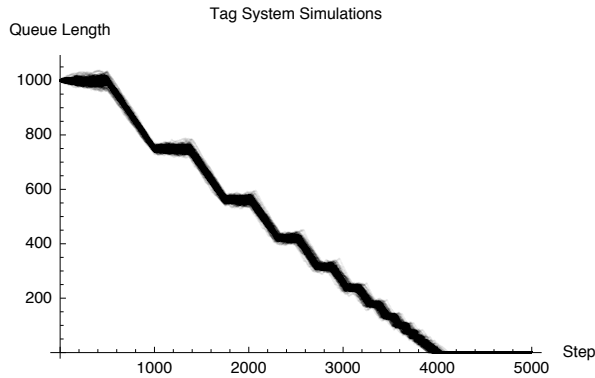
Figure 22.

▌ 3.2 Two-Tag System Predictor

The following diagram illustrates how the algorithm works: we find the selected pair for each position in every production and, in the case of the last position, consider every second production that could be appended.

ε 1
└─ bbb α
  ├─ <u>b</u>bb α
  ├─ b<u>b</u>b α
  ├─ bb<u>b</u> α
  │ ├─ bb<u>b bbb</u> $\alpha \times 1/4 = \alpha/4$
  │ ├─ bb<u>b ab</u> $\alpha \times 1/4 = \alpha/4$
  │ ├─ bb<u>b bb</u> $\alpha \times 1/4 = \alpha/4$
  │ └─ bb<u>b a</u> $\alpha \times 1/4 = \alpha/4$
  ├─ ab β
  │ ├─ <u>a</u>b β
  │ ├─ a<u>b</u> β
  │ │ ├─ a<u>b bbb</u> $\beta \times 1/4 = \beta/4$
  │ │ ├─ a<u>b ab</u> $\beta \times 1/4 = \beta/4$
  │ │ ├─ a<u>b bb</u> $\beta \times 1/4 = \beta/4$
  │ │ └─ a<u>b a</u> $\beta \times 1/4 = \beta/4$
  ├─ bb γ
  │ ├─ <u>b</u>b γ
  │ ├─ b<u>b</u> γ
  │ │ ├─ b<u>b bbb</u> $\gamma \times 1/4 = \gamma/4$
  │ │ ├─ b<u>b ab</u> $\gamma \times 1/4 = \gamma/4$
  │ │ ├─ b<u>b bb</u> $\gamma \times 1/4 = \gamma/4$
  │ │ └─ b<u>b a</u> $\gamma \times 1/4 = \gamma/4$
  └─ a δ
     └─ <u>a</u> δ
        ├─ <u>a bbb</u> $\delta \times 1/4 = \delta/4$
        ├─ <u>a ab</u> $\delta \times 1/4 = \delta/4$
        ├─ <u>a bb</u> $\delta \times 1/4 = \delta/4$
        └─ <u>a a</u> $\delta \times 1/4 = \delta/4$
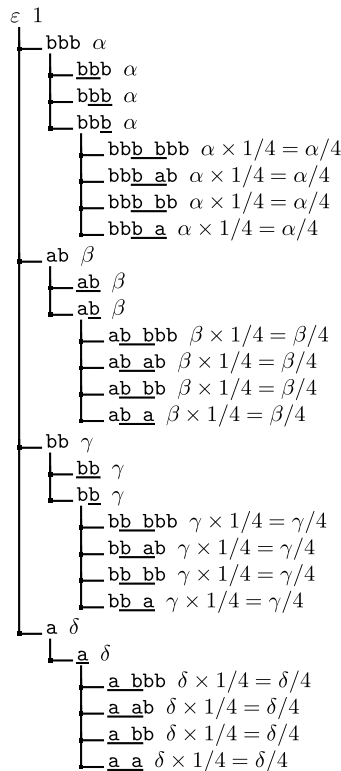
where $\alpha + \beta + \gamma + \delta = 1$. Adding up the quantities for each pair yields

$$
\begin{aligned}
\underline{aa}: \quad & \delta/4 + \delta/4 = \delta/2, \\
\underline{ab}: \quad & \beta + \delta/4 + \delta/4 = \beta + \delta/2, \\
\underline{ba}: \quad & \alpha/4 + \alpha/4 + \beta/4 + \beta/4 + \gamma/4 + \gamma/4 = \\
& \alpha/2 + \beta/2 + \gamma/2, \\
\underline{bb}: \quad & \alpha + \alpha + \alpha/4 + \alpha/4 + \beta/4 + \beta/4 + \gamma + \gamma/4 + \\
& \gamma/4 = 5\alpha/2 + \beta/2 + 3\gamma/2, \\
\text{total}: \quad & \delta/2 + \beta + \delta/2 + \alpha/2 + \beta/2 + \gamma/2 + 5\alpha/2 + \\
& \beta/2 + 3\gamma/2 = 3\alpha + 2\beta + 2\gamma + \delta.
\end{aligned}
\tag{44}
$$

Dividing the quantity of each pair by the total yields

$$
\begin{aligned}
\mathrm{P}(\underline{aa}) &= \frac{\delta/2}{3\alpha + 2\beta + 2\gamma + \delta} = \frac{\delta}{6\alpha + 4\beta + 4\gamma + 2\delta}, \\
\mathrm{P}(\underline{ab}) &= \frac{\beta + \delta/2}{3\alpha + 2\beta + 2\gamma + \delta} = \frac{2\beta + \delta}{6\alpha + 4\beta + 4\gamma + 2\delta}, \\
\mathrm{P}(\underline{ba}) &= \frac{\alpha/2 + \beta/2 + \gamma/2}{3\alpha + 2\beta + 2\gamma + \delta} = \frac{\alpha + \beta + \gamma}{6\alpha + 4\beta + 4\gamma + 2\delta}, \\
\mathrm{P}(\underline{bb}) &= \frac{5\alpha/2 + \beta/2 + 3\gamma/2}{3\alpha + 2\beta + 2\gamma + \delta} = \frac{5\alpha + \beta + 3\gamma}{6\alpha + 4\beta + 4\gamma + 2\delta}.
\end{aligned}
\tag{45}
$$

For instance, letting $\alpha = \beta = \gamma = \delta = 1/4$ yields

$$
\begin{aligned}
& 6\alpha + 4\beta + 4\gamma + 2\delta = 4, \\
& \mathrm{P}(\underline{aa}) = 1/16 = 0.0625, \\
& \mathrm{P}(\underline{ab}) = 3/16 = 0.1875, \\
& \mathrm{P}(\underline{ba}) = 3/16 = 0.1875, \\
& \mathrm{P}(\underline{bb}) = 9/16 = 0.5625,
\end{aligned}
\tag{46}
$$

which are the probabilities of randomly selecting each pair from the queue, allowing us to predict the productions that will be generated during this epoch. The following program implements this algorithm, using it to predict the large-scale properties of the specified rule set of a two-tag system.

```
# Returns the production to be appended to the queue
def get_prod(word):
  if word == (0, 0):
    return (1, 1, 1)
  if word == (0, 1):
    return (0, 1)
  if word == (1, 0):
```

```
      return (1, 1)
    if word == (1, 1):
      return (0,)

  # Returns the production distribution from the word
  distribution
  def get_prod_probs(word_probs, get_prod):
    prod_probs = {}
    for word in word_probs:
      prod = get_prod(word)
      if prod not in prod_probs: prod_probs[prod] = 0
      prod_probs[prod] += word_probs[word]
    return prod_probs

  # Normalizes a distribution into a probability distribution
  def normalize(probs):
    total = sum(probs.values())
    for event in probs:
      probs[event] /= total
    return probs

  # Returns the word distribution from the production
  distribution
  def get_word_probs(prod_probs):
    word_probs = {}
    for prod in prod_probs:
      if len(prod) == 0: continue
      for pos in range(len(prod) - 1):
        word = prod[pos:pos + 2]
        if word not in word_probs: word_probs[word] = 0
        word_probs[word] += prod_probs[prod]
      # Consider every production that could follow the
  current one
      for prod2 in prod_probs:
        if len(prod2) == 0: continue
        word = (prod[-1], prod2[0])
        if word not in word_probs: word_probs[word] = 0
        word_probs[word] += prod_probs[prod] *
  prod_probs[prod2]
    return normalize(word_probs)

  # Returns the expected density of symbols on the queue
  def get_densities(word_probs):
    densities = {}
    for word in word_probs:
      for symbol in word:
        if symbol not in densities: densities[symbol] = 0
        densities[symbol] += word_probs[word]
    return normalize(densities)

  def get_growth(word_probs, get_prod):
    expected_length = 0
    for word in word_probs:
      expected_length += len(get_prod(word)) *
```

```
word_probs[word]
  return expected_length – 2

initial_word_probs = {}
initial_word_probs[(0, 0)] = .25
initial_word_probs[(0, 1)] = .25
initial_word_probs[(1, 0)] = .25
initial_word_probs[(1, 1)] = .25

word_probs = {}
prod_probs = {}

word_probs[0] = initial_word_probs
epochs = 10

for epoch in range(epochs):
  prod_probs[epoch + 1] =
get_prod_probs(word_probs[epoch], get_prod)
  word_probs[epoch + 1] = get_word_probs(prod_probs[epoch
+ 1])

length = 10000
for epoch in range(epochs):
  print('Epoch ' + str(epoch))
  densities = get_densities(word_probs[epoch])
  growth = get_growth(word_probs[epoch], get_prod)
  print('Length: ' + str(length))
  print('Density of a symbols: ' + str(densities[0]))
  print('Density of b symbols: ' + str(densities[1]))
  print('')
  length *= (1 + growth/2)
```

Notice that the lengths of the string productions being considered inside the `get_prod_probs` function do not appear explicitly because the factors of |*s*| in the probability of that production being selected from the queue and of selecting a particular position within that production cancel each other.

The output for this particular set of rules follows.

```
Epoch 0
Length: 10000
Density of a symbols: 0.5
Density of b symbols: 0.5

Epoch 1
Length: 10000.0
Density of a symbols: 0.25
Density of b symbols: 0.75

Epoch 2
Length: 7500.0
Density of a symbols: 0.5
Density of b symbols: 0.5
```

```
Epoch 3
Length: 7500.0
Density of a symbols: 0.25
Density of b symbols: 0.75

Epoch 4
Length: 5625.0
Density of a symbols: 0.5
Density of b symbols: 0.5

Epoch 5
Length: 5625.0
Density of a symbols: 0.25
Density of b symbols: 0.75

Epoch 6
Length: 4218.75
Density of a symbols: 0.5
Density of b symbols: 0.5

Epoch 7
Length: 4218.75
Density of a symbols: 0.25
Density of b symbols: 0.75

Epoch 8
Length: 3164.0625
Density of a symbols: 0.5
Density of b symbols: 0.5

Epoch 9
Length: 3164.0625
Density of a symbols: 0.25
Density of b symbols: 0.75
```

Notice the similarity of these values to those of the output produced by the tag system simulator, and the similarity of the queue lengths that were predicted by the algorithm to those of the sharp changepoints in the plot.

4. Results

The following tables show the expected growth of the queue per step along with the density of a and b symbols at the beginning of each epoch for different tag system rule sets. These tables also show the expected length of the queue at the beginning of each epoch, assuming an initial reference length of 100.

$aa \to aab,\ ab \to ab,\ ba \to b,\ bb \to ba$							
Epoch	0	1	2	3	4	5	6
a density	0.500	0.500	0.467	0.455	0.450	0.449	0.448
b density	0.500	0.500	0.533	0.545	0.550	0.551	0.552
Growth	0.000	−0.125	−0.167	−0.181	−0.186	−0.188	−0.189
Length	100.00	100.00	93.75	85.92	78.15	70.88	64.22

aa → bb, ab → bb, ba → aaa, bb → bb							
Epoch	0	1	2	3	4	5	6
a density	0.500	0.333	0.120	0.054	0.026	0.013	0.006
b density	0.500	0.667	0.880	0.946	0.974	0.987	0.994
Growth	0.250	0.083	0.037	0.017	0.008	0.004	0.002
Length	100.00	112.50	117.17	119.34	120.35	120.83	121.07

aa → bab, ab → bbb, ba → aab, bb → bb							
Epoch	0	1	2	3	4	5	6
a density	0.500	0.273	0.185	0.129	0.095	0.071	0.054
b density	0.500	0.727	0.815	0.871	0.905	0.929	0.946
Growth	0.750	0.455	0.296	0.210	0.153	0.115	0.088
Length	100.00	137.50	168.78	193.76	214.11	230.49	243.74

aa → b, ab → b, ba → aab, bb → abb							
Epoch	0	1	2	3	4	5	6
a density	0.500	0.375	0.389	0.400	0.398	0.397	0.397
b density	0.500	0.625	0.611	0.600	0.602	0.603	0.603
Growth	0.000	0.250	0.222	0.200	0.205	0.206	0.206
Length	100.00	100.00	112.50	124.99	137.49	151.58	167.19

aa → aa, ab → ba, ba → ε, bb → ab							
Epoch	0	1	2	3	4	5	6
a density	0.500	0.667	0.767	0.863	0.929	0.965	0.982
b density	0.500	0.333	0.233	0.137	0.071	0.035	0.018
Growth	−0.500	−0.571	−0.500	−0.306	−0.152	−0.073	−0.036
Length	100.00	75..00	53.59	40.19	34.04	31.45	30.31

aa → bbb, ab → ab, ba → bb, bb → b							
Epoch	0	1	2	3	4	5	6
a density	0.500	0.125	0.100	0.083	0.071	0.062	0.056
b density	0.500	0.875	0.900	0.917	0.929	0.938	0.944
Growth	0.000	−0.750	−0.800	−0.833	−0.857	−0.875	−0.889
Length	100.00	100.00	62.50	37.50	21.88	12.51	7.034

aa → aaa, ab → b, ba → a, bb → b							
Epoch	0	1	2	3	4	5	6
a density	0.500	0.667	0.833	0.932	0.975	0.991	0.997
b density	0.500	0.333	0.167	0.068	0.025	0.009	0.003
Growth	−0.500	0.000	0.444	0.750	0.904	0.966	0.989
Length	100.00	75.00	75.00	91.65	126.02	182.98	271.36

$aa \to bbb$, $ab \to ab$, $ba \to bb$, $bb \to a$							
Epoch	0	1	2	3	4	5	6
a density	0.500	0.250	0.500	0.250	0.500	0.250	0.500
b density	0.500	0.750	0.500	0.750	0.500	0.750	0.500
Growth	0.000	−0.500	0.000	−0.500	0.000	−0.500	0.000
Length	100.00	100.00	75.00	75.00	56.25	56.25	42.19

The following table compares the values for these properties that were predicted using our methods to values that were obtained from simulations by averaging the results of 10 000 trials over six epochs.

$aa \to aaa$, $ab \to b$, $ba \to a$, $bb \to b$							
Epoch	0	1	2	3	4	5	6
Density of *a* in Queue							
Predicted	0.500	0.667	0.833	0.932	0.975	0.991	0.997
Measured	0.500	0.662	0.826	0.925	0.972	0.990	0.997
Error	0.000	0.005	0.007	0.007	0.003	0.001	0.000
Density of *b* in Queue							
Predicted	0.500	0.333	0.167	0.068	0.025	0.009	0.003
Measured	0.500	0.338	0.174	0.075	0.028	0.010	0.003
Error	0.000	0.005	0.007	0.007	0.003	0.001	0.000
Length of Queue							
Predicted	100.00	75.00	75.00	91.65	126.02	182.98	271.36
Measured	100.00	74.98	74.98	92.24	126.80	183.74	272.16
Error	0.00	0.02	0.02	0.59	0.78	0.76	0.80

A similar table can be created for the other rule sets. Other rule sets demonstrate a similar relative error despite the difference in production rules, corroborating the general accuracy and precision of the method.

5. Conclusion

In this paper, we have developed a method for predicting the large-scale properties of *n*-tag systems directly from their production rules. From the distribution of tuples of *n* symbols on the queue at the beginning of an epoch, one can predict the distribution of productions that are generated during that epoch. In turn, from this distribution, one can predict the distribution of tuples of symbols for the next epoch. This process can be repeated for any number of epochs.

From the tuple distribution of an epoch, one can determine various properties of the tag system's evolution within that epoch. For example, one can determine the density of particular symbols on the queue, the growth of the queue, or the length of the queue at any step by linear interpolation between the expected length of the beginning of the current epoch and that of the next epoch.

We have compared the property values predicted using our methods to those measured by performing multiple simulations under random initial configurations. These predictions retain great accuracy even after several epochs.

Investigating other properties of the symbol distribution on the queue and of the production rules themselves could yield further insight into the large-scale behavior of tag systems. In particular, we would like to find "shortcuts" for determining whether a particular set of rules creates sharp phase transitions between epochs, as opposed to more gradual changes, or determining whether a particular set of rules creates periodic behavior in the large-scale properties of the tag system, as shown in one of the examples. One might also want to determine whether and when the tag system reaches an equilibrium distribution by examining the production rules of the tag system directly.

References

[1] M. L. Minsky, "Recursive Unsolvability of Post's Problem of 'Tag' and Other Topics in Theory of Turing Machines," *Annals of Mathematics*, **74**(3), 1961 pp. 437–455. doi:10.2307/1970290.

[2] E. L. Post, "Formal Reductions of the General Combinatorial Decision Problem," *American Journal of Mathematics*, **65**(2), 1943 pp. 197–215. doi:10.2307/2371809.

[3] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[4] H. Wang, "Tag Systems and Lag Systems," *Mathematische Annalen*, **152**(1), 1963 pp. 65–74. doi:10.1007/BF01343730.

[5] J. Cocke and M. Minsky, "Universality of Tag Systems with P = 2," *Journal of the ACM*, **11**(1), 1964 pp. 15–20. doi:10.1145/321203.321206.

[6] L. De Mol, "Tag Systems and Collatz-Like Functions," *Theoretical Computer Science*, **390**(1), 2008 pp. 92–101. doi:10.1016/j.tcs.2007.10.020.

[7] Y. Rogozhin, "Small Universal Turing Machines," *Theoretical Computer Science*, **168**(2), 1996 pp. 215–240. doi:10.1016/S0304-3975(96)00077-1.

[8] J. H. Conway, "Unpredictable Iterations," in *Proceedings of the 1972 Number Theory Conference*, Boulder, CO, Boulder: University of Colorado, 1972 pp. 49–52.

[9] E. Lehtonen, "Two Undecidable Variants of Collatz's Problems," *Theoretical Computer Science*, **407**(1–3), 2008 pp. 596–600. doi:10.1016/j.tcs.2008.08.029.