PROBABILISTIC SELF-STABILIZATION

Ted Herman

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188

TR-89-34 November 1989

Abstract

A probabilistic self-stabilizing algorithm for a ring of identical processes is presented; the number of processes in the ring is odd, the processes operate synchronously, and communication is unidirectional in the ring. The normal function of the algorithm is to circulate a single token in the ring. If the initial state of the ring is abnormal, i.e. the number of tokens differs from one, then execution of the algorithm results probabilistically in convergence to a normal state with one token.

Keywords: Distributed Computing, Probabilistic Algorithms, Self-Stabilization, Uniform Rings.

0 Introduction

A self-stabilizing algorithm for a ring of identical processes is required; the algorithm is to circulate exactly one token in the ring: if, in an initial state of ring, there are numerous tokens, then the algorithm is required to reduce the number of tokens until there is exactly one token. Problems in a ring of identical processes have been previously considered, for example electing a unique leader in a ring of indistinguishable processes [7]. It is well known that for such problems, no deterministic solution is possible [1]. Typically, probablistic methods (using randomization) are proposed as solutions to these problems. A probabilistic technique for self-stabilization has been proposed in [6] for an asynchronous ring of processes. The original paper on self-stabilization [4] is also based on asynchronous rings. Unlike previous works, the setting for this paper is a ring of processes that operate synchronously.

Probabilistic algorithms have the drawback that there is no bound on the number of steps required to terminate. Instead, analysis derives the expected number of steps required for termination. Probabilistic algorithms do have positive aspects. They are usually more efficient, in some respect, than corresponding deterministic algorithms. Many probabilistic algorithms are also notable for the simplicity of their construction. The algorithm presented in this paper is no exception — it is an extremely simple program. It also has lower space complexity than the minimum possible for a deterministic algorithm.

Self-stabilizing systems are interesting because they prevail over some chaotic conditions. A self-stabilizing system is one that establishes legitimacy starting from an arbitrary, chaotic initial state; this behavior is non-trivial when system control is distributed rather than centralized. Such behavior is desirable in real systems that suffer from transient power fluctuations, or distributed systems that may be locally, dynamically reconfigured from time to time. After a dose of chaos, a self-stabilizing system recovers and operates normally. An intriguing property of a probabilistic self-stabilizing algorithm is that it is designed to overcome chaos, but uses randomness to do so.

The remainder of the paper consists of three sections. The first section presents the algorithm and proves correctness. Results about the expected behavior of the algorithm appear in the second section. The third section contains concluding remarks and open questions.

1 The Algorithm

Let n be the number of processes in the ring. We require that n be odd to obtain an algorithm that is self-stabilizing to a single-token state. If n is even, then the algorithm self-stabilizes to a state without tokens. The case of even n turns out to be useful for analysis presented in the second section, so lemmas and theorems in this section are stated for any finite, non-zero n.

The state of each process is a single bit. The state of the ring can be represented by a vector of n bits. Suppose x is such a vector; we index x by subscripts to refer to individual elements of the vector. Indexing is defined for any integer subscript by residue modulo n: x_i denotes the element x_k where $k = i \bmod n$.

Each process uses a random bit, denoted γ_i for process i. We assume that any interrogation of γ_i satisfies $\Pr(\gamma_i = 1) = r_i$ and $\Pr(\gamma_i = 0) = (1 - r_i)$ where r_i is some fixed number in the range $0 < r_i < 1$.

The basic step of the algorithm is procedure f, which inputs a ring state and outputs a ring state. To describe f, let x be an input ring state and let y be the output ring state obtained by some execution of f. For $0 \le i < n$, ring state y satisfies

$$y_i = \begin{cases} x_{i-1} & \text{if} \quad x_i \neq x_{i-1} \\ \gamma_i & \text{if} \quad x_i = x_{i-1} \end{cases}$$

Let $y = f.x$ denote that y is an output of a computation of procedure f with input x. We caution the reader that this notation can be deceptive. For instance, from $y = f.x$ and $z = f.x$ it does not follow that $y = z$ because f is not a deterministic function. Our convention is that $f.x$ refers to one given computation of f within the scope of a definition or lemma. We use the notation $f^k.x$ to denote k successive computations of f, i.e. $f^k.x = f^{k-1}.(f.x)$.

The algorithm consists of the computation $f^k.x$ where k is unbounded. One possible implementation is the iteration of the (parallel) assignment '$x := f.x$'; other methods of computing are also feasible, including non-deterministic order in computing the elements of $f^k.x$. We define a "token" to be any consecutive pair (x_{i-1}, x_i) of equal bits in the ring. Given any ring state x, if n is odd, then execution of the algorithm results probabilistically in a ring state with one token.

The rest of this section is devoted to proving the correctness of the algorithm. Correctness is shown by fulfilling three obligations. First, the safety of the algorithm is demonstrated: safety is the property that execution of the algorithm does not increase the number of tokens in the ring. Second, we show that the algorithm progresses in its normal state: progress is the property that a single token circulates in the ring. The third obligation is to show convergence to a normal state: convergence is the property that the algorithm reduces the number of tokens in a ring with more than one token. The progress and convergence properties are probabilistic, whereas the safety property is not a probabilistic property of the algorithm.

Prior to presenting the correctness arguments, some notation is introduced. Our presentation of the algorithm (above) expresses a ring state as a vector of bits. Alternatively, the state of the ring can be denoted by a string expression. For instance, 1^n denotes the ring state in which $x_i = 1$ for all i. As a notational convenience, we let variables a and b stand for complementary bits in an expression. For instance, arguments about the expression $a^p b^q$ apply to either $0^p 1^q$ or $1^p 0^q$. Let c be an anonymous bit; we regard c as a placeholder in an expression. For instance, acc represents aaa, aba, abb, and aab; the expression c^n represents all ring states. The following notation is used to associate an expression with indexed bits in a vector.

$$x[j : p \leq j < p + 3] \;=\; aba$$

means that $x_p = a$, $x_{p+1} = b$, and $x_{p+2} = a$. (Note that x_p and x_{p+2} are the same bit if $n = 2$.)

The function T reports the presence of a token at a specified process:

$$T.x.i \equiv \; (x_i = x_{i-1}).$$

Given a ring state x we say there is a token at process i if $T.x.i$ holds. The number of tokens in the ring is denoted by $S.x$:

$$S.x = (\mathbf{N}i : \; 0 \leq i < n : \; T.x.i).$$

Obviously, the number of tokens is constrained by the ring size: $0 \leq S.x \leq n$. The following lemma shows that if n is odd, the number of tokens is odd.

Lemma 0 $\quad S.x \bmod 2 = n \bmod 2.$

Proof [due to J. Misra]. Define function σ_i to be the integer subtraction $\sigma_i = x_i - x_{i-1}$. Let K be the number of instances where $\sigma_i = 0$ in the ring (i.e., $K = S.x$), let L be the number of instances where $\sigma_i = 1$ and let M be the number of instances where $\sigma_i = -1$. Thus, $K + L + M = n$. Observe that

$$\sum_{i=0}^{n-1} \sigma_i \;=\; 0 \;=\; (K \times 0) \;+\; (L \times 1) \;+\; (M \times (-1)).$$

Therefore $L = M$, so $n = K + 2L$. This proves the lemma since $(n - 2L)$ is even (odd) iff n is even (odd). $\square$

As a consequence of Lemma 0, any procedure that changes the number of tokens will change the number by some multiple of two. It is also clear that, to require a ring state with exactly one token, it is necessary that the ring have an odd number of processes.

The following lemma is a first step in the proof of safety. It demonstrates a local property suggesting that tokens do not spontaneously appear in a computation of $f.x$.

Lemma 1 $\quad T.(f.x).i \;\Rightarrow\; T.x.i \;\vee\; T.x.(i-1)$.

Proof of the contrapositive, by unwinding definitions. Let $y = f.x$. The obligation is to show $\neg T.x.i \;\wedge\; \neg T.x.(i-1) \;\Rightarrow\; \neg T.y.i$. From the antecedent we may write $x[j : i-2 \le j \le i] \;=\; bab$ to represent all states where $\neg T.x.i$ and $\neg T.x.(i-1)$ hold. By the definition of f, $y_i = a$ and $y_{i-1} = b$, therefore $T.y.i$ is *false*, which proves the lemma. $\square$

When there is a single token in the ring and $n > 1$, it is easy to see (the reader is invited to construct examples) that by application of procedure f, the token either 'moves' from some process j to process $(j+1)$ or 'stays' at process j. When there are numerous tokens in a ring it is not obvious how they 'move' by application of f. We introduce a function to describe token movement: $M.x.y.i$ is defined for ring states x and y (typically $y = f.x$) and a process i:

$$M.x.y.i = \begin{cases} \textit{false} & \text{if } \neg T.x.i \\ \textit{false} & \text{if } 0 = i \bmod n \;\wedge\; S.x = n \;\wedge\; \gamma_0 = 0 \\ \textit{true} & \text{if } 0 = i \bmod n \;\wedge\; S.x = n \;\wedge\; \gamma_0 = 1 \end{cases}$$

for the remaining cases, $T.x.i \;\wedge\; (0 \ne i \bmod n \;\vee\; S.x \ne n)$:

$$M.x.y.i = \begin{cases} \textit{true} & \text{if } \neg T.x.(i+1) \ \wedge \ T.y.(i+1) \\ \textit{false} & \text{if } \neg T.x.(i+1) \ \wedge \ \neg T.y.(i+1) \\ M.x.y.(i+1) & \text{if } T.x.(i+1) \ \wedge \ T.y.(i+1) \\ \neg M.x.y.(i+1) & \text{if } T.x.(i+1) \ \wedge \ \neg T.y.(i+1) \end{cases}$$

Given a ring state x with a token at process i and a computation $f.x$, we say that the token at process i moves iff $M.x.(f.x).i$ holds. It is straightforward to verify that the case enumeration in the definition of M is complete, that cases are disjoint, and that M is well-founded.

The next result is concerned with a fine point in the definition of M; it turns out to be useful to prove the an important lemma that follows.

Lemma 2 $\neg T.x.(i-1) \ \wedge \ T.x.i \ \wedge \ T.(f.x).i \ \Rightarrow \ \neg M.x.(f.x).i.$

Proof by induction. Let $y = f.x$. We set up the induction by defining a string expression for the lemma's antecendent; induction is over the length of the string expression. Our application of induction is based on the following observation.

Consider the expression aac^p. In this expression, there is a token at the second bit and possibly there are tokens within the string c^p. By removing the second bit of this expression we obtain a new expression ac^p. Observe that tokens occur within string c^p, in exactly the same places, in either aac^p or ac^p because a precedes c^p in both expressions. The same type of operation can also be applied to the expression $abac^p$: by removing the second and third bits we obtain the expression ac^p, which preserves the token occurrence within string c^p.

From the antecedent of the lemma, it follows that $S.x \neq n$. Therefore the definition of M is completely determined by the token placement within x and y. If there is a token at some process j in both x and y, then (by the argument of the preceding paragraph) we may remove process j from both x and y, and remaining token occurrences are unaffected; moreover, by the recurrence in the definition of M (in the case where $T.x.j$ and $T.y.j$ are *true*), the evaluation of M is unchanged for processes remaining after the removal of process j. This removal of a process from both x and y obtains shorter string expressions, which is our technique for induction.

A string expression representing the antecedent is

$$x[j : (i-2) \leq j < (i+k)] \quad = \quad ba^k b$$

for some $k \geq 2$ (in this expression $T.x.(i-1)$ is *false* and $T.x.i$ is *true*, corresponding to the antecedent). By the definition of f and from $T.y.i$ (the antecedent) the string expression for y is

$$y[j : (i-2) \leq j < (i+k)] \quad = \quad cb^2c^{k-2}a.$$

For convenience, we write $(ba^kb, cb^2c^{k-2}a)$ to depict the antecedent. The proof of the lemma is by induction on k.

Basis. $k = 2$. The antecedent is (ba^2b, cb^2a). In this expression $T.y.(i+1)$ is *false* and $\neg T.x.(i+1)$ is *false*, so by definition $M.x.y.i$ is *false*.
Induction. $k > 2$. Consider the following three cases.
Case $T.y.(i+1)$, which has the form $(ba^kb, cb^3c^{k-3}a)$. In this case we remove process $(i+1)$ from x and y to obtain $(ba^{k-1}b, cb^2c^{k-3}a)$. This smaller form preserves the evaluation of $M.x.y.i$, therefore by the inductive hypothesis $M.x.y.i$ is *false*.
Case $\neg T.y.(i+1) \;\wedge\; T.y.(i+2)$, which has the form $(ba^kb, cb^2a^2c^{k-4}a)$. In this case we remove process $(i+2)$ from x and y and appeal to the inductive hypothesis as in the previous case.
Case $\neg T.y.(i+1) \;\wedge\; \neg T.y.(i+2)$, which has the form $(ba^kb, cb^2abc^{k-4}a)$. In this case we remove processes $(i+1)$ and $(i+2)$ from x and y and appeal to the inductive hypothesis. □

Lemma 3 $\quad T.(f.x).i \;\Rightarrow\; \neg M.x.(f.x).i \;\vee\; M.x.(f.x).(i-1)$

Proof. The conclusion is trivially obtained if $n = 1$; henceforth we assume $n > 1$. Suppose $T.(f.x).i \equiv true$ and consider two cases for $T.x.i$:

Case $\neg T.x.i$. In this case Lemma 1 rules out the possibility that $T.x.(i-1)$ is *false*, therefore $T.x.(i-1)$ is *true* and $M.x.(f.x).(i-1) \equiv true$ for this case.

Case $T.x.i$. Consider two subcases for $T.x.(i-1)$. If $T.x.(i-1)$ is *false* then by Lemma 2, $M.x.(f.x).i \equiv false$. If $T.x.(i-1)$ is *true* then by M's definition, $M.x.(f.x).i \equiv false$ or $M.x.(f.x).(i-1) \equiv true$. $\square$

Let $\mathcal{M}$ be a map from the tokens of x to positions of $f.x$:

$$\mathcal{M}.x.(f.x).i = \begin{cases} i & \text{if} \quad \neg M.x.(f.x).i \\ (i+1) & \text{if} \quad M.x.(f.x).i \end{cases}$$

If we combine Lemma 1 and Lemma 3, the following property of $\mathcal{M}$ is apparent: if $T.(f.x).i$ is *true*, then

$$(T.x.i \;\wedge\; \mathcal{M}.x.(f.x).i = i) \quad \text{or} \quad (T.x.(i-1) \;\wedge\; \mathcal{M}.x.(f.x).(i-1) = i).$$

In other words, each token in $f.x$ is mapped by $\mathcal{M}$ from some token in x. This result finds significance in proving safety and convergence properties of the algorithm.

Theorem 0 (Safety) $\quad S.x \geq S.(f.x)$.

Proof by contradiction. Suppose $S.x < S.(f.x)$. By Lemma 3 and the definition of $\mathcal{M}$, each token in $f.x$ corresponds to some token in x. Therefore some token in x is mapped by $\mathcal{M}$ to two tokens of $f.x$ which contradicts the functionality of M. $\square$

The Safety Theorem and Lemma 0 demonstrate that if there is one token in ring state x, then all subsequently computed states $f^k.x$ also have exactly one token. We expect that the algorithm should also progress in such a normal state, that is, the token should circulate in the ring. The following lemma and theorem show that computation of f probabilistically circulates a token in the ring.

Lemma 4 $T.x.i \;\Rightarrow\; (\exists t:\; 0 < t < 1:\; \Pr(M.x.(f.x).i) = t)$.

Proof. Consider some computation $y = f.x$; it is straightforward to show that there is another computation $z = f.x$ that differs from y only in the choice of γ_i, such that $M.x.y.i \not\equiv M.x.z.i$. The probability r_i therefore satisfies the conclusion. □

Theorem 1 (Progress)

$$S.x = 1 \;\wedge\; T.x.i \;\Rightarrow\; \Pr(\exists k ::\; T.(f^k.x).(i+1)\;) = 1.$$

Proof. It suffices to show

$$\lim_{k\to\infty} \Pr(\;\forall j : 0 \le j < k:\; T.(f^j.x).(i+1) \equiv \mathit{false}\;) = 0.$$

By Lemma 4 the probability that the token does not move in one computation $f.x$ is fixed and less than one; therefore the limit probability that the token does not move in k consecutive computations of f tends to zero as k exceeds any bound. □

Convergence is the remaining issue in the proof of correctness, that is to show with probability one, from any initial state, execution of the algorithm eventually arrives at a state with at most one token. To show convergence properties we introduce a function D to partition the state-space of processes. D is defined as the minimum distance between two tokens in the ring:

$$\begin{aligned} D.x \;=\; & (\mathbf{min}\, k : 0 < k < n \;\wedge \\ & (\exists i ::\; T.x.i \;\wedge\; T.x.(i+k) \;\wedge \\ & (\forall j:\; 0 < j < k:\; \neg T.x.(i+j))\;) :\; k). \end{aligned}$$

Observe that $D.x$ is at most $n/2$. The following two lemmas show that the distance between tokens decreases probabilistically.

Lemma 5 $T.x.i \;\wedge\; T.x.j \;\wedge\; (i \bmod n) \neq (j \bmod n)$
$\Rightarrow\; (\exists t : 0 < t < 1:\; \Pr(M.x.(f.x).i \not\equiv M.x.(f.x).j) = t)$.

The proof is similar to the proof of Lemma 4. □

Lemma 6 $D.x = p \ \wedge \ p > 1 \ \Rightarrow \ \Pr(\exists k :: \ D.(f^k.x) < D.x) = 1$

Proof by induction on p. Observe that $D.x > 1$ implies (by Lemma 5) that the probability of $D.(f.x) < D.x$ is non-zero.

Basis $p = n/2$. By the same argument given in the proof of the Progress Theorem, the probability of $(\forall k :: \ D.(f^k.x) = n/2)$ tends to zero as k tends to infinity.

Induction $1 < p < n/2$. Suppose we have some computation that satisfies $(\forall k :: \ D.(f^k.x) \geq D.x)$. By the inductive hypothesis, for every i such that $D.(f^i.x) > D.x$,

$$\Pr(\exists j : j > i : \ D.(f^j.x) < D.(f^i.x) \ ) = 1.$$

Consequently, with probability one, there are infinitely many m such that $D^m.x = D.x$. The number of states is finite, so some state z satisfying $D.z = D.x$ is visited infinitely often. By Lemma 5 the probability of $D.(f.z) < D.z$ is non-zero, therefore the limit probability of the computation we have supposed is zero. □

The convergence obligation is fulfilled by the following two results.

Lemma 7
$T.x.(i-1) \ \wedge \ T.x.i \ \wedge \ M.x.(f.x).(i-1) \ \wedge \ \neg M.x.(f.x).i \ \Rightarrow \ S.(f.x) < S.x$

Proof by contradiction, using a similar argument to that of the proof of the Safety Theorem. □

Theorem 2 (Convergence) $S.x > 1 \ \Rightarrow \ \Pr(\exists k :: \ S.(f^k.x) < S.x) = 1$

Proof. Lemma 6 can be applied repetitively to conclude, with probability one, $D.(f^m.x) = 1$ for some m. Observe that $D.(f^m.x) = 1$ implies (Lemmas 7 and 5) there is a non-zero probability that $S.(f^{m+1}.x) < S.(f^m.x)$. By a similar argument to that given in the proof of Lemma 6 it can be shown that the limit probability of $S.(f^k.x) \geq S.x$ is zero. $\square$

2 Analysis

Two of the correctness issues resolved in the previous section have parallels in this section. Corresponding to the issue of progress, we calculate the expected time for a single token to circulate in a ring; corresponding to the issue of convergence, we obtain a bound for the expected time for the algorithm to minimize the number of tokens in the ring. By 'expected time' we mean the average number k for which $f^k.x$ establishes progress or convergence. For convergence, the initial state x should be the case for which the expected convergence is maximum (the worst case). In order to calculate expected time bounds for the algorithm we assume that all random bit generators behave like fair coin tosses; that is, $r_i = 1/2$ for each process i.

Lemma 8 For x a state of a ring with exactly one token, the expected minimum value of k such that $f^k.x$ advances the token exactly n times is $2n$.

Proof. We are interested in calculating the probability for each number k such that $f^k.x$ has the following behavior: in the computation $f^{k-1}.x$ the token does not move, whereas the token moves in the computation $f.(f^{k-1}.x)$. There are two possible outcomes, equally likely, of a computation of $f.x$: either the token will move or the token will stay. The probability that the token will move by a computation of $f.x$ is $1/2$. The probability that the token will stay by $f.x$ but move by $f.(f.x)$ is $1/4$. Thus the probability that the token first moves in the kth iteration of f is 2^{-k}. The expected value for the number of iterations of f to move a token is therefore

$$\sum_{i=0}^{\infty} i2^{-i}.$$

To compute this sum we use the technique of transform analysis. The sum is equivalent to $P(1)$ where

$$P(z) = \sum_{i=0}^{\infty} i2^{-i}z^{i}.$$

To obtain a closed form for $P(z)$ let $R(z)$ be defined as

$$R(z) = \sum_{i=0}^{\infty} 2^{-i}z^{i}.$$

Then $P(z) = zR'(z)$. The closed form for $R(z)$ is $(1 - z/2)^{-1}$, so

$$zR'(z) = (z/2)(1 - z/2)^{-2}.$$

Hence the value for $P(1)$ is 2, which is the expected number of iterations of f to move a token. The expected number of iterations to rotate the token around the ring is therefore $2n$. $\square$

To analyze the expected time for convergence we consider first a simpler problem. In a ring with exactly two tokens, what is the expected time to reach a state without tokens? Our analysis is based on the observation that a ring state x with exactly two tokens has four equally likely outcomes from the computation $f.x$. Either both tokens move, both tokens stay, the first moves while the second stays, or the first token stays and the second moves.

The distance partition D introduced for the proof of convergence provides a useful framework for calculating the probabilities for outcomes of $f.x$. Let $m = n/2$ and observe that $0 < D.x \le m$ for a ring with two tokens. Let A be the $m \times m$ matrix of transitional probabilities between classes of D. An element a_{ij} in the matrix is the probability that $D.(f.x) = j$ given that $D.x = i$. The matrix is

$$A = \left|\begin{array}{cccccc} 1/2 & 1/4 & & & & \\ 1/4 & 1/2 & 1/4 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \\ & & & 1/4 & 1/2 & 1/4 \\ & & & & 1/2 & 1/2 \end{array}\right|$$

Explanation for this matrix follows. The rows corresponding to x satisfying $1 < D.x < m$ have a regular pattern: consider the four possible outcomes from $f.x$. If both tokens move or if both tokens stay, then $D.x = D.(f.x)$; the probability of this event is therefore $1/2$. If one token moves and the other stays then either

$$D.(f.x) = D.x + 1 \quad \text{or} \quad D.(f.x) = D.x - 1.$$

These events occur with probability $1/4$. The exceptional cases in the matrix are $D.x = 1$ and $D.x = m$. If $D.x = 1$ then convergence ($D.(f.x) = 0$) is a possible outcome, with probability $1/4$. This event is excluded from the matrix A, which is defined only for ring states with two tokens. The other exception is $D.x = m$, for which $D.(f.x) = m$ with probability $1/2$ and $D.(f.x) = m - 1$ with probability $1/2$.

Lemma 9 For x a state of a ring with exactly two tokens, the expected minimum value of k such that $f^k.x$ has no tokens is at most $n^2/2$.

Proof. Let e_i denote the $1 \times m$ vector equal to the i^{th} row of the $m \times m$ identity matrix I. Let $j = D.x$. The vector e_j represents states satisfying $D.x = j$ (that is, the initial probability $\Pr(D.x = j) = 1$). The vector $e_j A$ contains the probabilities for $D.(f.x)$, and $e_j A^k$ contains the probabilities for $D.(f^k.x)$.

The probability that $D.(f^k.x) = 1$ is

$$C.k = e_j A^k e_1^T.$$

The probability that $f^{k+1}.x$ has zero tokens is computed from $C.k$ by $C.k/4$ because convergence is one of the four equally likely outcomes of f applied to a state with adjacent tokens. The expected value of k such that $D.(f^k.x) = 1$ is

$$E(kC.k) = \sum_{i=0}^{\infty} ie_j A^i e_1^T \quad = \quad e_j(\sum_{i=0}^{\infty} iA^i)e_1^T.$$

The closed form for the summation can be obtained by transform analysis [5]. The expected value for is given by

$$E(kC.k) = e_j(I-A)^{-1}A(I-A)^{-1}e_1^T.$$

The matrix $(I-A)^{-1}$ has a regular structure:

$$B = (I-A)^{-1} = \left| \begin{array}{cccccccc} 4 & 4 & 4 & 4 & . & . & 4 & 2 \\ 4 & 8 & 8 & 8 & . & . & 8 & 4 \\ 4 & 8 & 12 & 12 & . & . & 12 & 6 \\ 4 & 8 & 12 & 16 & . & . & 16 & 8 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ 4 & 8 & 12 & 16 & . & . & 4(m-1) & 2m \end{array} \right|$$

The matrix BAB also has a regular structure, but it is not necessary to compute it entirely; only the first column is of interest:

$$E(kC.k) = e_j(BABe_1^T).$$

Let $h = BABe_1^T$ (the first column of BAB). By straightforward calculation, the elements of the $m \times 1$ vector h are

$$h_i = 16mi - 8i^2 - 4.$$

Since $E(kC.k) = e_j h = h_j$, we are interested in

$$(\mathbf{max}\ j : 0 < j \leq m :\ \ h_j)$$

to obtain the worst case for an initial state x. In the range $0 < i \leq m$, the formula for h_i is an increasing function of i, so the worst case for x is $D.x = m$ — when the tokens are initially as far apart as possible. Evaluation of $E(kC.k)$ for this case is

$$E(kC.k) = h_m = 8m^2 - 4.$$

This result is an upper bound for the expected time to reach a state satisfying $D.(f^k.x) = 1$; the expected time for convergence is bounded by

$$E(1 + kC.k/4) = 1 + E(kC.k)/4 = n^2/2. \quad \square$$

Lemma 10 The expected minimum value of k such that $S.(f^k.x) < 2$ is at most $n^2 \lceil \log n \rceil /2$.

Proof. We define three intermediate procedures that are based on procedure f. The last of these procedures halves the number of tokens in expected time $n^2/2$; repeated application of this procedure models the behavior of f, and thereby we obtain the bound $n^2 \lceil \log n \rceil /2$ for convergence of f.

We define $\mathcal{F}$ to be a procedure for a ring with two or zero tokens. To compute $\mathcal{F}.x$, $y = f.x$ is computed first. If $S.y = 0$ then y is the output of $\mathcal{F}.x$. If $S.y > 0$ then some two-token state z satisfying $D.z = D.y$ is chosen arbitrarily to be the output of $\mathcal{F}.x$. Observe that Lemma 9 holds for $\mathcal{F}$: the expected minimum value of k such that $\mathcal{F}^k.x$ has no tokens is at most $n^2/2$.

We define $\mathcal{G}$ to be a procedure for a ring with two or zero tokens. To compute $\mathcal{G}.x$, $y = f.x$ is computed first. If $S.y = 0$ then y is the output of $\mathcal{G}.x$. If $S.y > 0$ then some two-token state z satisfying $D.z \leq D.y$ is chosen arbitrarily to be the output of $\mathcal{G}.x$. Observe that, with probability one, there exists k so that $D.(\mathcal{G}.x) = D.(\mathcal{F}^k.x)$. Therefore $\mathcal{G}$ converges at least as quickly as $\mathcal{F}$; the expected minimum value of k such that $\mathcal{G}^k.x$ has no tokens is bounded by $n^2/2$.

We define $\mathcal{H}$ to be the following procedure, defined for any state x such that $S.x \geq 2$. The computation of $\mathcal{H}$ consists of an initial step followed by an iteration.

The initial step is a coloring of two of the tokens of x: two tokens are selected, one is colored yellow and the other is colored green, so that $1 + \lfloor S.x/2 \rfloor$ tokens are on the (clockwise) path from green to yellow, and $1 + \lceil S.x/2 \rceil$ tokens are on the (clockwise) path from yellow to green. Some choice of two tokens to color green and yellow is clearly possible for any x with at least two tokens; there may be many possible choices for the two tokens, and any choice is satisfactory for the initial step.

The iterated step consists of an application of procedure f followed by a coloring of tokens, described below. The iteration terminates when the ring does not have a green token and a yellow token.

To describe the iterated step, let i be the process holding the green token, let j be the process holding the yellow token, let $y = f.x$, let $i' = \mathcal{M}.x.y.i$, and let $j' = \mathcal{M}.x.y.j$. Initially, all tokens in y are uncolored; then tokens are colored as follows. Let v denote the shortest path between i' and j', that is v is either the (clockwise) path from i' to j' or the (clockwise) path from j' to i'; in case both paths are the same length, one is chosen arbitrarily to be v. If there are at least two tokens in v, then the token closest to i' in v is colored green (if there is a token at i' then it is colored green) and the token closest to j' in v is colored yellow.

We claim that the expected number of iterations before $\mathcal{H}$ terminates is at most $n^2/2$. To see this, observe that for all but the final step of $\mathcal{H}$, the step either emulates f for the colored tokens, or obtains some state having two colored tokens with reduced distance. The colored tokens therefore describe a computation of $\mathcal{G}$ (except that $\mathcal{H}$ may terminate before $\mathcal{G}$ would terminate). Thus the expected time for convergence of $\mathcal{H}$ is at most $n^2/2$.

Let z be the final state from a terminating computation of $\mathcal{H}$ with input state x. In initial state x there are at least $1 + \lfloor S.x/2 \rfloor$ tokens on any path from one colored token to the other. In final state z there are no colored tokens, so there is a path between i' and j' with at most one token. Therefore

$$S.z \;\leq\; S.x - (1 + \lfloor S.x/2 \rfloor) + 1 \;=\; \lceil S.x/2 \rceil .$$

To complete the proof, consider the computation $\mathcal{H}^p.x$. We seek the minimum p so that the ring state has fewer than two tokens. The previous paragraph shows that $S.(\mathcal{H}.x) \leq \lceil S.x/2 \rceil$. Consequently $\lceil \log n \rceil$ is a bound on p. Each application of $\mathcal{H}$ terminates in at most $n^2/2$ expected iterations of f, thereby the expected number of iterations of f to reach a state with fewer than two tokens is at most $n^2 \lceil \log n \rceil / 2$. $\square$

3 Conclusion

It is difficult to compare the algorithm presented here to other work because the traditional model of computation for self-stabilization is an asynchronous network. Some of the asynchronous algorithms do admit the possibility of synchronous execution. For instance, the original unidirectional algorithm [4] can be used in a synchronous setting [3], but a distinguished process is required; moreover $O(\log n)$ bits per process are needed to represent a process state (the paper [2] cites $O(\log n)$ as a lower bound on any deterministic, unidirectional algorithm with indistinguishable processes).

In [2] there is an algorithm for uniform rings, which are rings of indistinguishable processes in an asynchronous model. That algorithm is deterministic and self-stabilizes for the case of prime n, however the algorithm fails if neighboring processes execute simultaneously and is unsuitable for the synchronous model. The probabilistic self-stabilizing algorithm in [6] is bidirectional; it is unclear if the algorithm admits synchronous execution.

Some interesting questions arise in connection with the algorithm presented here. Does there exist a similar probabilistic algorithm, self-stabilizing to a single token, for the case of even n? Is it possible to devise a deterministic pattern for the inputs γ_i so that the same program can be used to achieve self-stabilization, but converge faster? Can the bound $n^2\lceil\log n\rceil/2$ be lowered?

Acknowledgement. The author is grateful for criticisms from Edsger W. Dijkstra and Jayadev Misra.

References

[1] D. Angluin, "Local and Global Properties in Networks of Processes," 12th Annual ACM Symposium on Theory of Computing, pp. 82-93, April 1980.

[2] James E. Burns and Jan Pachl, "Uniform Self-Stabilizing Rings," *ACM Transactions on Programming Languages and Systems 11*, 2(April 1989), pp. 330-344.

[3] J. E. Burns, M. G. Gouda, and R. E. Miller, "On Relaxing Interleaving Assumptions," Technical Report GIT-ICS-87/36, Georgia Institute of Technology, Atlanta, Georgia, August 1988.

[4] Edsger W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control," *CACM 17*, 11(1974), pp. 643-644.

[5] Ronald A. Howard, *Dynamic Probabilistic Systems Volume I: Markov Models,* John Wiley & Sons, Inc., New York, 1971, pp. 53-54.

[6] Amos Israeli and Marc Jalfon, "Self-Stabilizing Ring Orientation," Department of Electrical Engineering, Technion—Israel, September 25, 1989.

[7] Alon Itai and Michael Rodeh, "Symmetry Breaking in Distributive Networks," 22nd Annual Symposium on Foundations of Computer Science, pp. 150-158, 1981.