Information Processing Letters 35 (1990) 63–67
North-Holland
29 June 1990

PROBABILISTIC SELF-STABILIZATION

Ted HERMAN

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

Communicated by J. Misra
Received 30 October 1989
Revised 21 February 1990

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

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. The solution presented in this paper is simple, inviting an informal example. Imagine seven boys, seated in a circle, each with a coin laying flat on one hand. In unison, all boys do the following. Each boy looks at the face of his own coin and that of the boy to his left in the circle; if the two coins show differing faces (head and tail) then he will turn his coin over; otherwise he will toss his coin to obtain a random face. This unison step is repeated ad infinitum. Regardless of the initial faces of the coins, after a finite number of steps (with probability one) only one boy tosses a coin in each step.

Problems in a ring of identical processes have been previously considered, for example electing a unique leader in a ring of indistinguishable processes [6]. It is well known that for such problems, no deterministic solution is possible [1]. Typically, probabilistic methods (using randomization) are proposed as solutions to these problems. The deterministic algorithm in [2], which self-stabilizes to a single token in a ring of identical processes, is based on a model that prohibits synchronous execution. A probabilistic technique for self-stabilization has been proposed in [5] for an asynchronous ring of processes. The original paper on self-stabilization [3] is also based on asynchronous rings. Unlike previous works, the setting for this paper is a ring of processes that operate synchronously.

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 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. To specify properties of a computation dependent on random bit(s) we employ the notation $\Pr(V) = v$ to mean that v is the probability of outcome V. We assume that any evaluation 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 $0 \leqslant i < n$, ring state y satisfies

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

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 can be deceptive notation. 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 nondeterministic 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.

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, we use string expressions to specify sets of ring states. For instance, 1^n denotes the set of ring states in which all bits are 1. As a notational convenience, 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 $1^p 0^q$. Consider the bits of some ring state x, from x_0 to x_{n-1}, arranged clockwise in a circle; a string expression for x is derived by listing (clockwise) all n bits in the circular arrangement, starting with an arbitrary bit. Thus, any rotation of a string expression specifies the same set of states: for example, $a^p b^q$ and $b^q a^p$ represent the same set of states. 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 the set of all ring states. For convenience, let c^0 denote an empty string.

The notation for a quantified expression is (*op*: *range*: *term*), where *op* specifies the operator and dummy variable for the quantification, *range* specifies the range of the dummy, and *term* is an expression that is some function or predicate dependent on the dummy. The operators used in this paper are $\forall$, $\exists$, **S** for summation, and **N** for counting the number of occurrences that a predicate term holds in the specified range. We use this notation frequently as argument in the probability (Pr) notation. In such instances, the quantified expression is a predicate that quantifies over states of an iterated computation.

Given a ring state x we say there is a *token* at process i if $x_i = x_{i-1}$; we say there is *shift* at process i if $x_i \neq x_{i-1}$. The number of tokens in the ring is denoted by $S.x$:

$$S.x = (\mathbf{N}i\colon 0 \leqslant i < n\colon x_i = x_{i-1}).$$

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

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

Proof. The lemma obviously holds in the case $S.x = n$. In the case $S.x < n$, ring state x is a member of the set of states given by a string expression consisting of alternating strings of "a" and "b" bits. That is, the string expression is of the form

$$a^{M.0} b^{M.1} \ldots a^{M.(k-2)} b^{M.(k-1)}$$

where M is a vector of k positive integers subject to the constraint

$$(1<k) \wedge (k \leqslant n) \wedge (\mathbf{S}j\colon 0 \leqslant j < k\colon M.j) = n.$$

In our choice of this string expression, we require that the initial term ($a^{M.0}$) and the final term ($b^{M.(k-1)}$) be different bits, i.e. $b \neq a$ (this requirement can be met provided $S.x < n$). As a consequence, k is even. Observe that there is a 1-1 correspondence between the shifts in x and the elements of vector M. Therefore x has k shifts. The lemma follows from $S.x = n - k$. □

As a consequence of Lemma 0, any procedure that changes the number by tokens will change the number of 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 theorem demonstrates that tokens do not increase in number by computation of f.

Theorem 0 (*Safety*). $S.x \geqslant S.(f.x)$.

Proof. The theorem trivially holds in the case $S.x = n$. In the case $S.x < n$, ring state x belongs to the set of states

$$a^{M.0}b^{M.1} \ldots a^{M.(k-2)}b^{M.(k-1)}$$

as described in the proof of Lemma 0. From the definition of f, and substituting c for each evaluation of a random bit, the state $f.x$ belongs to the set of states

$$bc^{M.0-1}ac^{M.1-1} \ldots bc^{M.(k-2)-1}ac^{M.(k-1)-1}.$$

(Note that we exploit the notation c^0 in this string expression; for example, if $M.0 = 1$ then the string $c^{M.0-1}$ is empty, thereby $bc^{M.0-1}a = ba$.) Observe that a string of the form "$bc^{M.i-1}a$" contains at least one shift, and similarly "$ac^{M.i-1}b$" contains at least one shift. Therefore there is at least one shift associated with each element of M. We conclude that the number of shifts in $f.x$ is at least as large as the number of shifts in x, which proves the theorem. □

The Safety Theorem and Lemma 0 demonstrate that if there is exactly 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 theorem shows that computation of f probabilistically circulates a token in the ring.

Theorem 1 (*Progress*)

$$S.x = 1 \wedge x_i = x_{i-1} \Rightarrow \Pr(\exists k\colon k>0\colon y = f^k.x \wedge y_i = y_{i+1}) = 1.$$

Proof. The theorem holds trivially in the case $n = 1$; henceforth suppose $n > 1$. Since the token is at process i in state x, the probability that the token is at process i in state $f.x$ is either $(1 - r_i)$ or r_i (the complementary outcome of $f.x$ places the token at process $i + 1$). Therefore it suffices to show

$$\Pr(\forall j\colon j>0\colon y = f^j.x \wedge y_{i-1} = y_i) = 0.$$

Let u be the maximum of r_i and $(1 - r_i)$. For any $m > 0$,

$$\Pr(\forall j\colon 0<j \leqslant m\colon y = f^j.x \wedge y_{i-1} = y_i) \leqslant u^m.$$

Hence the theorem follows from

$$\lim_{m \to \infty} u^m = 0. \quad \square$$

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 set of ring states with at least two tokens.

$D.x$ is defined as the minimum distance between two tokens in ring state x. If state x has fewer than two tokens then $D.x$ is undefined. If state x has a pair of adjacent tokens, then $D.x = 1$; that is, if x has a string expression of the form a^3c^{n-3} (or a^2 if $n = 2$) then $D.x = 1$. If $D.x \neq 1$ and x has a string expression of the form $a^2b^2c^{n-4}$ then $D.x = 2$. The expression $a^2ba^2c^{n-5}$ corresponds to $D.x = 3$, and so on. Observe that $D.x$ is at most $n/2$. The following lemma shows that the distance between tokens decreases probabilistically.

Lemma 1. $D.x = p \wedge p > 1 \;\Rightarrow\; \Pr(\exists k\colon\ k > 0\colon\ D.(f^k.x) < D.x) = 1$.

Proof. Observe, by expanding the definition of f under the condition $D.x > 1$, that $\Pr(D.(f.x) < D.x) > 0$ (for instance, if process i and process $i + k$ have tokens, and there are no tokens between i and $i + k$, then one possible outcome of computing f is that process $i + 1$ has a token and process $i + k$ has a token). We complete the proof by induction on p.

Basis: $p = n/2$. By a similar argument to that given in the proof of the Progress Theorem,

$$\Pr\big(\forall k\colon\ k > 0\colon\ D.(f^k.x) = n/2\big) = 0.$$

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

$$\Pr\big(\exists j\colon\ j > 0\colon\ D.\big(f^j.(f^i.x)\big) < D.\big(f^i.x\big)\big) = 1.$$

Consequently, with probability one, there are infinitely many m such that $D.(f^m.x) = D.x$. The number of states is finite, so some state z that satisfies $D.z = D.x$ is visited infinitely often in the supposed computation. The probability that $D.(f.z) < D.z$ is non-zero; consequently the probability that state z is visited infinitely often, without the outcome $D.(f.z) < D.z$, is zero. Therefore the probability of the computation we have supposed is zero. □

Theorem 2 (*Convergence*). $S.x > 1 \;\Rightarrow\; \Pr(\exists k\colon\ k > 0\colon\ S.(f^k.x) < S.x) = 1$.

Proof. Consider some state z satisfying $D.z = 1$. Recall the form "$bc^{M.i-1}a$" from the proof of the Safety Theorem. $D.z = 1$ implies there exists some i such that $M.i \geqslant 3$ in term "$bc^{M.i-1}a$" or a term "$ac^{M.i-1}b$"; a possible outcome of the computation $f.z$ is the assignment of values to the c-bits that increases the number of shifts in the string expression, thereby decreasing the number of tokens. Therefore $\Pr(S.(f.z) < S.z) > 0$. Lemma 1 can be applied repetitively to conclude, with probability one, $D.(f^m.x) = 1$ for some m; 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 1 it follows that

$$S.x > 1 \;\Rightarrow\; \Pr\big(\forall k\colon\ k > 0\colon\ S.(f^k.x) = S.x\big) = 0. \qquad \square$$

Acknowledgment

The author is grateful for criticisms from Edsger W. Dijkstra and Jayadev Misra. Suggestions from anonymous referees encouraged many improvements in the presentation.

References

[1] D. Angluin, Local and global properties in networks of processes, in: *Proc. 12th Ann. ACM Symp. on Theory of Computing* (1980) 82–93.

[2] J.E. Burns and J. Pachl, Uniform self-Stabilizing rings, *ACM Trans. on Programming Languages and Systems* **11** (2) (1989) 330–344.

[3] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, *Comm. ACM* **17** (11) (1974) 643–644.

[4] G.M. Brown, M.G. Gouda and C.L. Wu, Token systems that self-stabilize, *IEEE Trans. on Computers* **38** (6) (1989) 845–852.

[5] A. Israeli and M. Jalfon, Self-stabilizing ring orientation, Dept. of Electrical Engineering, Technion–Israel, 1989.

[6] A. Itai and M. Rodeh, Symmetry breaking in distributive networks, in: *Proc. 22nd Ann. Symp. on Foundations of Computer Science* (1981) 150–158.