Centrum voor Wiskunde en Informatica

REPORT_RAPPORT_

SEN

Software Engineering

Software ENgineering

A Note on _K_-state Self-Stabilization in a Ring with _K_=_N_

Wan Fokkink, Jaap-Henk Hoepman, Jun Pang

REPORT SEN-R0402 FEBRUARY 15, 2004

CWI is the National Research Institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organization for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

Copyright © 2004, Stichting Centrum voor Wiskunde en Informatica
P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

A Note on K-state Self-Stabilization in a Ring with $K=N$

ABSTRACT
We show that, contrary to common belief, Dijkstra's K-state mutual exclusion algorithm on a ring [1, 2] also stabilizes when the number K of states per process is one less than the number $N+1$ of processes in the ring. We formalize the algorithm and verify the proof in PVS, based on Qadeer and Shankar's work [8]. We show that $K=N$ is sharp by giving a counter-example for $K=N-1$.

2000 Mathematics Subject Classification: 65G20, 68M12, 68M14, 68W40
1998 ACM Computing Classification System: C.2.2, C.2.4, D.2.4, F.3.1
Keywords and Phrases: Distributed computing, fault tolerance, self-stabilization, formal verification, PVS
Note: This research is partly supported by the Dutch Technology Foundation STW under the project CES5008: Improving the quality of embedded systems using formal design and systematic testing.

A Note on K-state Self-Stabilization in a Ring with $K = N$ *

Wan Fokkink[1,3], Jaap-Henk Hoepman[2], and Jun Pang[1]

[1] CWI, Department of Software Engineering, PO Box 94079, 1090 GB Amsterdam, The Netherlands, {wan,pangjun}@cwi.nl
[2] University of Nijmegen, Department of Computer Science, PO BOX 9010, 6500 GL Nijmegen, The Netherlands, jhh@cs.kun.nl
[3] Vrije Universiteit Amsterdam, Department of Theoretical Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, wanf@cs.vu.nl

Abstract. We show that, contrary to common belief, Dijkstra's K-state mutual exclusion algorithm on a ring [1, 2] also stabilizes when the number K of states per process is one less than the number $N+1$ of processes in the ring. We formalize the algorithm and verify the proof in PVS, based on Qadeer and Shankar's work [8]. We show that $K = N$ is sharp by giving a counter-example for $K = N - 1$.

Keywords: Distributed computing, fault tolerance, self-stabilization, formal verification, PVS

ACM Computing Reviews Categories: C.2.2, C.2.4, D.2.4, F.3.1.

1 Introduction

In his seminal paper [1], Dijkstra introduced the notion of self-stabilization. A distributed system is said to be self-stabilizing if it satisfies the following two properties:

1. *convergence*: starting from an arbitrary state, the system is guaranteed to reach a stable state;
2. *closure*: once the system reaches a stable state, it cannot become unstable anymore.

A system with the property of self-stabilization can have the advantages of fault tolerance, robustness for dynamic topologies, and straightforward initialization.

Consider a system with a number of processes sharing a common resource (usually called critical section). Given an arbitrary initial state of the system, there might be more than one process enable to access the common resource. The problem of mutual exclusion is to guarantee that the common resource will not

* This research is partly supported by the Dutch Technology Foundation STW under the project CES5008: Improving the quality of embedded systems using formal design and systematic testing.

be accessed by more than one process simultaneously. Self-stabilizing algorithms for mutual exclusion make sure that each infinite run of the system reaches a stable state where exactly one process is enabled; and from then on, mutual exclusion of the common resource is guaranteed.

In [1], Dijkstra presented three elementary self-stabilizing algorithms for mutual exclusion on a ring network: an algorithm with K-state processes, an algorithm with four-state processes, and an algorithm with three-state processes. Regarding their correctness, he wrote:

- "For brevity's sake most of the heuristics that led me to find them, together with the proofs that they satisfy the requirements, have been omitted, [...]".

After more than ten years, Dijkstra [3] published a proof of self-stabilization of his algorithm with three-state processes, and acknowledged that the verification was actually not trivial.

In this paper, we focus on Dijkstra's algorithm with K-state processes. We consider a system of $N+1$ processes, numbered from 0 through N, arranged in a unidirectional ring. Each process p_i has a counter $v(i)$ that can hold a value from 0 to $K-1$. Each process can observe its own counter value and the counter value of its anti-clockwise neighbor. p_0 is a distinguished process that is enabled when $v(0) = v(N)$, and when enabled, it can increment its counter by 1 modulo K. Each process p_i for $i = 1, \ldots, N$ is enabled when $v(i) \neq v(i-1)$, and when enabled, it can update its counter value so that $v(i) = v(i-1)$. Thus the behavior of the system can be presented as follows:

Dijkstra's K-state algorithm for mutual exclusion. Let processes $p_0, \ldots, p_N$ form a unidirectional ring, where the counter for each process p_i holds a value $v(i) \in \{0, \ldots, K-1\}$.

- if $v(0) = v(N)$, then $v(0) := (v(0) + 1) \bmod K$;
- if $v(i) \neq v(i-1)$ for $i = 1, \ldots, N$, then $v(i) := v(i-1)$.

The system is said to be in a *stable* state if it contains exactly one enabled process, which can be interpreted as holding a token. This token can be passed along the ring network; a process can access the common resource only when it holds the token.

This algorithm has been proved correct by different proof methods for self-stabilization, e.g. [14, 11, 12]. It attracted much attention from the formal verification community. There are two distinct traditions in automatic verification: theorem proving and model checking. Merz [5] formalized the algorithm and proved it correct in Isabelle/HOL [6]. Qadeer and Shankar [8] applied PVS [7] to prove its correctness. Later on, Kulkarni et al. [4] also proved its correctness using PVS in a different fashion. Model checking techniques were applied to this algorithm in [10, 13]. Due to the state explosion problem, this approach has some restrictions: it cannot be directly used for any possible initial state, and/or it can only prove the algorithm correct with a limited number of processes and states.

However, all these proofs only showed correctness of the algorithm under a weaker condition, namely the algorithm is correct if $K > N$. This also happened in Schneider's survey paper on self-stabilization [9]. The only exception we could find is [4]. Although they proved the algorithm correct for $K > N$, almost at the end of the paper, they stated:

- "it is possible to prove stabilization when $K \geq N$– we will need to redo only the proofs that depend on this assumption, namely Lemmas 6.4, 6.6, 6.8."

However, the validity of this claim is not clear, especially their formulation of Lemma 6.4 is false when $K = N$.

Judging on the literature, it seems to be a common belief that Dijkstra's K-state mutual exclusion algorithm on a ring only stabilizes when $K > N$. But in fact, Dijkstra gave a note after presenting the solution with K-state machines in [1] as follows:

- "Note 1. [...] the relation $K \geq N$ is sufficient."

A brief informal proof sketch was given by himself in [2]. In addition, he said:

- "(and for smaller values of K counter examples kill the assumption of self-stabilization.)"

We note that, if $K = N$, there should be at least three processes in the ring; namely, if $K = N = 1$, then clearly p_0 is always enabled and p_1 is never enabled. If $K > N$, then the algorithm also works for a ring with two processes.

In this paper, we formally prove that if $N > 1$, then $K \geq N$ is sufficient for the stabilization of Dijkstra's K-state mutual exclusion algorithm. For the condition $K > N$, the proofs in $[14, 11, 8, 5, 4]$ used the classic pigeonhole principle. The proof for $K = N$ becomes considerably more complicated, since the pigeonhole principle cannot be simply applied for any state of the algorithm. This will be explained in detail in Section 3. Our proof, which is different from the proof sketch in [2], has been checked in PVS.

Outline of the paper. In Section 2, we show that Dijkstra's K-state mutual exclusion algorithm on a ring also stabilizes when the number of states per process is one less than the number of processes on the ring, namely $K \geq N$. We formalized the algorithm and checked our proof in PVS. Our verification in PVS is based on [8], we reused their formalization of the algorithm and most of their lemmas. We present the crucial lemmas of our PVS verification in Section 3. In Section 4, we show that $K \geq N$ is sharp by a counter-example, which was missing in [2]. We conclude this paper in Section 5.

2 Proof of Self-Stabilization

We give the proof that Dijkstra's K-state mutual exclusion algorithm on a ring stabilizes when $K \geq N$. First we prove the closure property for self-stabilization (see Proposition 1).

Lemma 1. *In each state of the algorithm, there is at least one enabled process.*

Proof. We distinguish two cases:

- for all $i \in \{1, \ldots, N\}$, $v(i) = v(0)$. In particular, $v(0) = v(N)$, which implies p_0 is enabled;
- otherwise, there exists a $j \in \{1, \ldots, N\}$ such that $v(j) \neq v(0)$, and for all $i \in \{1, \ldots, j-1\}$, $v(i) = v(0)$. Since $v(j) \neq v(j-1)$, p_j is enabled.

□

Lemma 1 implies that no run of the algorithm ever deadlocks, as in each state the enabled process(es) can "fire", meaning that the counter value is updated.

Proposition 1. *Once in a stable state, the system will remain in stable states.*

Proof. We assume p_i is the only enabled process in some stable state. It is easy to see that when p_i fires, it makes itself disabled, and it makes at most p_i's clockwise neighbor enabled. By Lemma 1, in each state of the algorithm, there exists at least one enabled process. Therefore, after the firing of p_i, the clockwise neighbor of p_i is the only enabled process, so the system remains in a stable state. □

We proceed to prove the convergence property for self-stabilization (see Theorem 1).

Lemma 2. *In each infinite run of the algorithm, p_0 fires infinitely often.*

Proof. Given a state, consider the sum over all elements $\{N-i \mid i \in \{1, \ldots, N\} \wedge p_i \textit{ is } \text{enabled}\}$. Clearly, when a nonzero process fires, this sum strictly decreases. Furthermore, for each state, this sum is at least 0. Hence, in each infinite run, p_0 must fire infinitely often. □

Definition 1. *The* legitimate states *are those states that satisfy $v(i) = x$ for all $i < j$ and $v(i) = (x-1) \bmod K$ for all $j \leq i \leq N$, for some choice of $x < K$ and $j \leq N$.*

Note that a legitimate state is stable, as only p_j is enabled.

Theorem 1. *Let $N > 1$. Even if $K = N$, Dijkstra's K-state mutual exclusion algorithm for $N+1$ processes stabilizes.*

Proof. By Lemma 1, no run of the algorithm ever deadlocks. By Lemma 2, in each infinite run of the algorithm p_0 fires infinitely often.

Let $N > 1$. We prove that each infinite run of the algorithm visits a legitimate state. Consider the case where p_0 fires for the first time. Then just before that, $v(0) = v(N) = y$ for some y, and the new value of $v(0)$ becomes $(y+1) \bmod K$. Now consider the case when p_0 fires again. Then just before that, $v(0) = v(N) = (y+1) \bmod K$. In order for p_N to change its counter value from y to $(y+1) \bmod K$, it must have copied $(y+1) \bmod K$ from its anti-clockwise neighbor p_{N-1}. This moment must have occurred after p_0 changed its counter value to $v(0) = (y+1) \bmod K$. But then, just after p_N copies $(y+1) \bmod K$ from p_{N-1},

we actually have $v(N-1) = v(N) = (y+1) \bmod K$. In other words, since $N > 1$ implies that $p_{N-1} \neq p_0$, two different nonzero processes hold the same counter value $(y+1) \bmod K$. Then the N nonzero processes hold at most $N-1$ different counter values from $\{0, \ldots, K-1\}$. When $K \geq N$ (so in particular when $K = N$), then at this point in time there is an $x < K$ that does not occur as the counter value of any nonzero process in the ring.

Since p_0 fires infinitely often, eventually $v(0)$ becomes x. The other processes merely copy counter values from their anti-clockwise neighbors, so at this point no other process holds x. The next time p_0 fires, $v(N) = v(0) = x$. The only way that p_N gets the counter value x is if all intermediate processes have copied x from p_0. We conclude that all processes have the counter value x, which is a legitimate state. □

Dijkstra [2] gave a specific scenario to show that the system will definitely reach a legitimate state, after p_0 has been enabled for N times. In most cases, a legitimate state can be detected earlier than in that scenario, as shown in the above proof.

3 Mechanical Verification in PVS

In [8], Qadeer and Shankar presented a detailed description of a mechanical verification in PVS of stabilization of Dijkstra's K-state mutual exclusion algorithm. Although they only checked the correctness of the algorithm under the condition $K > N$, their PVS formalism and proof could for a large part be reused,[4] which saved us much effort and gave us many insightful thoughts on the verification in PVS.

First, we present Qadeer and Shankar's claims to sketch their proof skeleton. Then we show the lemma that we had to adapt for our proof. The algorithm satisfies the following properties, for each state of the system, and each infinite run from this state:

- I. there is always at least one enabled process;
- II. the number of enabled processes never increases;
- III. the enabledness of each process is eventually toggled;
- IV. p_0 eventually takes on any counter value below K (follows by Property III);

These properties require no restriction on the relation between N and K. Property I corresponds to Lemma 1. Property II follows the fact that when a process fires, it makes itself disabled, and it makes at most its clockwise neighbor enabled. Property III is a more general version of Lemma 2. Qadeer and Shankar's PVS proof of these first four properties could be (more or less) reused by us directly.

[4] The URL `www.csl.sri.com/pvs/examples/self-stability` contains their PVS formalization and proofs.

– V. there is some value x below K such that $v(i) \neq x$ for all $i \in \{1, \ldots, N\}$ (follows by Property IV, and the proof of Theorem 1);
– VI. eventually $v(0) = x$, and $v(i) \neq x$ for all $i \in \{1, \ldots, N\}$; then p_0 is disabled until $v(i) = v(0)$ for all $i \in \{1, \ldots, N\}$ (follows by Property V);
– VII. the system is self-stabilizing (follows by properties VI, I, and II).

The proof of Property V uses the pigeonhole principle, which states that if each of $n+1$ pigeons is assigned to one of n pigeonholes, then some hole must contain at least two pigeons. This principle was also formulated and proved in [8].

Let $S(v)$ denote the set $\{x < K \mid \exists i \in \{1, \ldots, N\}(v(i) = x)\}$. The following lemma corresponds to Property V. It states that the nonzero processes do not contain all the possible counter values.

Lemma 3. *(Lemma 4.13 in [8]) If $K > N$, then $\exists x < K(x \notin S(v))$.*

Under the condition $K > N$, this can be informally proved as follows [8]: there are N nonzero processes, and hence at most N distinct counter values at these processes; if there are K $(K > N)$ possible counter values, then there must be some $x < K$ that is not the counter value at any nonzero process.

If we relax the condition to $K \geq N$, the above proof fails, because the pigeonhole principle does not apply when the number of pigeons equals the number of pigeonholes.

Starting from this point, we assume that $K \geq N$. We define $T(v)$ to denote the set $\{x < K \mid \exists i \in \{1, \ldots, N-1\}(v(i) = x)\}$. In the following lemma the pigeonhole principle does apply.

Lemma 4. $\exists x < K(x \notin T(v))$.

Proof. $T(v)$ contains at most $N - 1$ distinct counter values at processes from p_1 to p_{N-1}. If there are K $(K \geq N)$ possible counter values, then there must be some $x < K$ with $x \notin T(v)$. □

To check the proof of Lemma 4 in PVS, we could simply follow the PVS proof steps of Lemma 3 in [8]. Now we introduce an extra lemma.

Lemma 5. $v(N) \in T(v) \implies S(v) = T(v)$.

Proof. This is straightforward by the definitions of $S(v)$ and $T(v)$. □

In PVS, Lemma 5 could be proved by using existing PVS libraries for the finite cardinalities. Now we present the main lemma for our PVS proof, corresponding to Lemma 3 in [8] (Property VI).

Lemma 6. *Each infinite run of the algorithm eventually reaches a state where the nonzero processes do not contain all the possible counter values.*

Proof. We know from Property III that p_N will eventually fire. By the algorithm, we then have $v(N) = v(N-1)$, so that $v(N) \in T(v)$. By Lemma 5, $S(v) = T(v)$. By Lemma 4, we can find an $x < K$ with $x \notin T(v)$, so $x \notin S(v)$. □

After proving Lemma 6, and reusing (more or less) the lemmas and the PVS proof steps for properties VI and VII in [8], we could mechanically prove self-stabilization of Dijkstra's K-state algorithm in PVS.

4 $K = N$ is Sharp

In this section, we give a counter-example showing that a smaller value of K would kill self-stabilization. For example, in Fig. 1 (which assumes that $N \geq 3$), we have a system with $K = N - 1$, meaning that each process can have a counter value $\{0, \ldots, N-2\}$. Consider the initial state shown at the top left-hand side of Fig. 1, in which $p_0, \ldots, p_{N-2}$ hold counter values from 0 to $N-2$, p_{N-1} holds counter value 0, and p_N holds counter value 1. By the algorithm, $p_1, \ldots, p_N$ are enabled, so the number of enabled processes is N. (In Fig. 1, black processes are enabled.)

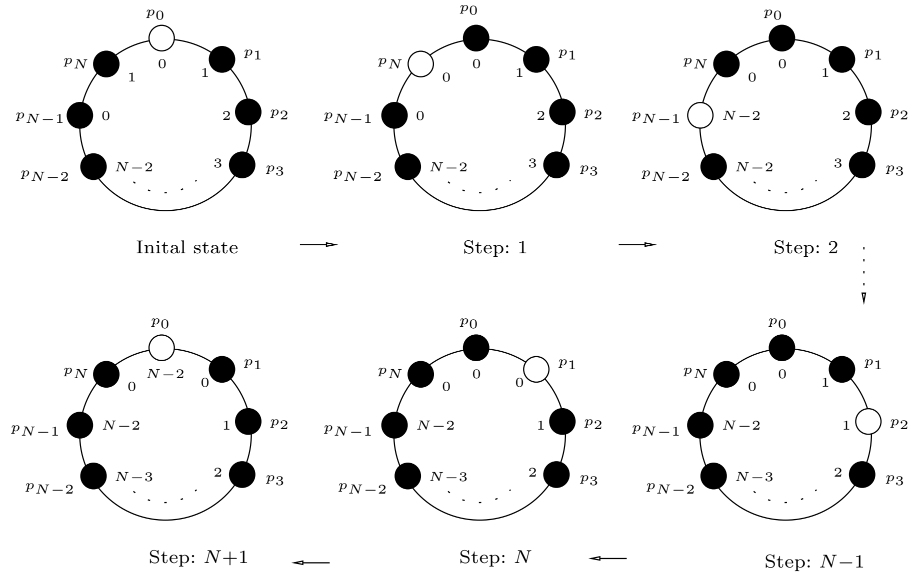

Fig. 1. A counter-example: a ring with $K = N - 1$

We have a run as follows:

Step 1: p_N fires and makes p_0 enabled;
Step 2: p_{N-1} fires and makes p_N enabled;
......
Step $N-1$: p_2 fires and makes p_3 enabled;
Step N: p_1 fires and makes p_2 enabled;
Step $N+1$: p_0 fires and makes p_1 enabled.

From the initial state, after the above $N+1$ steps (all processes have fired only once), the system ends in a state where the counter values of the processes are symmetric (modulo $N-1$) to the initial state, so it still has N enabled processes.

This scenario can be executed infinitely often without breaking the symmetry. So the system will never reach a legitimate state. Thus $K = N$ is sharp!

5 Conclusion

Judging on the literature on self-stabilization, it seems to be common belief that Dijkstra's K-state algorithm on a ring stabilizes when $K > N$. In this paper we show that, contrary to this common belief, the algorithm also stabilizes when the number of states per process is one less than the number of processes on the ring (namely $K = N$). Our proof was formalized and checked in PVS, based on [8]. We have given a counter-example showing that $K = N$ is indeed sharp.

One important fact (Lemma 6) used in our proof is that the nonzero processes do not contain all the possible counter values. By this observation, together with the fact that each process is infinitely often enabled, we can prove that each infinite run of the algorithm will reach a legitimate state. For the case $K > N$, this fact can be proved using the pigeonhole principle, as is done in [14, 11, 8, 5, 4]. For the case $K = N$ in this paper, we choose the moment that p_N is enabled and fires, which makes $v(N) = v(N - 1)$. After that we can apply the pigeonhole principle. Another important fact (Lemma 1) is that whenever the system reaches a stable state, it will remain in stable states. Thus we have proved the properties for self-stabilization.

Regarding the verification in PVS, we downloaded the PVS code and proof by Qadeer and Shankar. Following their proof steps in PVS, we simply added a new definition of $T(v)$, proved two new lemmas (Lemma 4 and Lemma 5), and adapted one lemma as Lemma 6. The whole verification did not take too much effort. First, we spent a few days to understand the formalism and proof in [8]. Since the PVS system, including PVS libraries, has been updated after 1998, the downloaded PVS proof could not be simply rerun. We made some adaptions to make their PVS proof work again. After that, when we had the idea to prove (as shown in Section 2) the algorithm correct under the condition $K = N$, the proof was completely checked in PVS within one day. The dump file containing our PVS formalization and proofs can be found at the URL `www.cwi.nl/~pangjun/stabilization/`.

References

1. E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, **17**(11): 643–644, 1974.
2. E.W. Dijkstra. Self-stabilization in spite of distributed control. In *Selected Writings on Computing: A Personal Perspective*, pp. 41–46, Springer-Verlag, 1982.
3. E.W. Dijkstra. A belated proof of self-stabilization. In *Distributed Computing*, **1**(1): 5-6, Springer-Verlag, 1986.
4. S.S. Kulkarni, J. Rushby, and N. Shankar. A case-study in component-based mechanical verification of fault-tolerant programs. In *Proc. 4th Workshop on Self-Stabilization*, pp. 33-40. IEEE Computer Society, 1999.

5. S. Merz. On the verification of a self-stabilizing algorithm. Technical Report, University of Munich, 1998.
6. T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic.* Springer-Verlag, 2002
7. S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In *Proc. 11th Conference on Automated Deduction*, LNCS 607, pp. 748–752. Springer-Verlag, 1992
8. S. Qadeer and N. Shankar. Verifying a self-stabilizing mutual exclusion algorithm. In *Proc. IFIP Working Conference on Programming Concept and Methods*, pp. 424-442, Chapman & Hall, 1998.
9. M. Schneider. Self-stabilization. *ACM Computing Surveys*, **25**(1): 45-67, 1993.
10. S.K. Shukla, D.J. Rosenkrantz, and S.S. Ravi. Simulation and validation tool for self-stabilizing protocols. In *Proc. 2nd SPIN Workshop*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science (**32**), 1996.
11. G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 1994.
12. O.E. Theel. Exploitation of Ljapunov theory for verifying self-stabilizing algorithms. In *Proc. 14th Conference on Distributed Computing*, LNCS 1914, pp. 209-222. Springer-Verlag, 2000.
13. T. Tsuchiya, S. Nagano, R.B. Paidi, and T. Kikuno. Symbolic model checking for self-stabilizing algorithms. *IEEE Transaction on Parallel and Distributed Systems*, **12**(1): 81-95, 2001.
14. G. Varghese. *Self-Stabilization by Local Checking and Corrections.* PhD thesis, MIT, 1992.