CONVERGENCE/RESPONSE TRADEOFFS IN CONCURRENT SYSTEMS

Mohamed G. Gouda[1] and Michael Evangelist[2]

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

TR-88-39 October 1988

Abstract

A *self-stabilizing* system is one which if started at any unsafe state, is guaranteed to converge to a safe state within a finite number of state transitions. The *convergence span* of such a system is defined as the maximum number of critical transitions that can be executed before the system reaches a safe state. In this paper, we discuss the tradeoff between the convergence span of a self-stabilizing system and its response span. In particular, we argue that the convergence span can be reduced by some factor by increasing the response span by the same factor, and vice versa. Our discussion is centered on a class of self-stabilizing systems for detecting termination on a uni-directional ring. The discussion leads to a family of systems whose convergence span is O(n/k), and whose response span is O(nk), where n is the number of processes in the system, and k is a parameter whose value can be chosen arbitrarily from the domain 1 ... n.

---

[1]Department of Computer Sciences, University of Texas at Austin, TX 78712-1188. This work was supported in part by Office of Naval Research Contract N00014-86-K-0763, and in part by a contract with MCC.
[2] MCC, 3500 W. Balcones Center Dr., Austin, TX 78759.

1. Introduction

A system is called *self-stabilizing* iff starting at any unsafe state, the system is guaranteed to reach a "safe" state within a finite number of "critical" state transitions. Thus, to check whether a given system is self-stabilizing, one needs first to identify which system states are safe and which state transitions are critical. This identification depends on the intended functionality of the system. Consider for example a mutual exclusion system. A safe state for this system can be defined as any state s where at most one process is in its critical section at any state reachable from s. In other words, a safe state is one whose descendent states satisfy the safety requirement of mutual exclusion. The critical transitions for the system can be defined as those in which some process enters its critical section.

This definition of self-stabilizing systems is a generalization over the previous definition in which all state transitions are implicitly assumed critical [3]. The generalization allows a self-stabilizing system to remain within unsafe states indefinitely, provided that none of its critical transitions is executed infinitely often. For example, if none of the processes in the above mutual exclusion system ever attempts to enter its critical section, then the system can remain within unsafe states indefinitely. On one hand, this is unavoidable: a system converges to a safe state only if it is executed and so the mutual exclusion system can converge to a safe state only if its processes attempt to enter their critical sections. On the other hand, this is harmless: if no process attempts to enter its critical section (infinitely often), then the possibility of error occurrence, i.e. two processes being in their critical sections simultaneously, will not arise (infinitely often).

The maximum number of critical transitions that can be executed by a system before reaching a safe state is called the system's *convergence span*. A finite (infinite) convergence span for a system indicates that the system is (is not) self-stabilizing. A small (large) convergence span for a self-stabilizing system indicates that the system can commit a small (large) number of errors during its convergence, an *error* is defined in this context as an execution of a critical transition at an unsafe state.

The convergence span of a system is related (as shown shortly) to another quantity called the system's response span. The *response span* of a system is defined as the maximum number of transitions that can be executed by the system starting from some initial state and ending in some target state. The choice of initial and target states for defining the response span of a given system depends on the

particular system. For instance, an initial state for a mutual exclusion system can be any state in which no process is in its critical section and at least one process is trying to enter its critical section; a target state for the system can be any state in which at least one process is in its critical section.

The objective of this paper is to illustrate that the convergence span of a self-stabilizing system can be *inversely-proportional* to the system's response span. Thus, a price for reducing the convergence span of a system is a proportional increase in its response span. An inverse relationship between the convergence and response of a self-stabilizing system may not be very surprising. Indeed, if delaying the system's response can be tolerated, then more "checks" can be added into the system to reduce the maximum number of errors that it can commit as it converges from unsafe to safe states. What may be surprising is that the relationship is proportional, i.e. reducing the convergence span by some factor increases the response span by the *same* factor.

In order to make these concepts concrete, we discuss in this paper the convergence/response tradeoffs in a class of self-stabilizing systems for detecting termination. Each system has n processes $P_0, \ldots, P_{n-1}$ that execute an infinite stream of jobs, one after another. When the processes finish executing one job, P_0 detects the termination of execution and starts the execution of the next job, and the cycle repeats. The processes in each system are arranged on a uni-directional ring, where every process P_i reads the state of its left neighbor $P_{i-1 \bmod n}$, and uses these readings to update its own state. (Having the processes arranged on such a ring makes the task of achieving self-stabilization nontrivial and interesting.)

The rest of the paper is organized as follows. In Section 2, we discuss a simple termination-detection system that uses a binary-valued token. A version of this system was introduced earlier by Dijkstra, Feijen, and van Gastern [4], and independently by Gouda [6]. Our objective of presenting this system here is two-fold. First, we use the system as a vehicle to introduce our notation and to set the stage for subsequent discussions. Second, we show that this system, like most early termination-detection systems, e.g. [2, 5], is not self stabilizing. Two self-stabilizing variations of this system are discussed in Sections 3 and 4. The first variation, in Section 3, uses an n-valued token; it exhibits slow convergence and fast response. The second variation, in Section 4, uses a binary-valued token, and an n-valued counter in process P_0; it exhibits fast convergence and slow response. In Section 5, we discuss how to combine these two systems into one that exhibits flexible convergence and response.

Specifically, the convergence (response) span of the system can be reduced by any factor provided that its response (convergence) span increases by the same factor. Concluding remarks are in Section 6.

2. The Original System: Unstable

The basic idea of the original system is straightforward. The system consists of n processes $P_0, \ldots, P_{n-1}$ arranged on a uni-directional ring. Whenever process P_0 becomes idle, it sends a token to its right neighbor P_1. Whenever a process P_i, $i \neq 0$, becomes idle, then receives a token from its left neighbor $P_{i-1 \bmod n}$, it propagates the token to its right neighbor $P_{i+1 \bmod n}$. If the token returns to P_0 and P_0 is still idle, P_0 detects termination. If the token returns to P_0 and P_0 is busy, termination is not detected at this time, but the procedure is repeated when P_0 becomes idle at a later time. Notice that after an idle P_0 sends a token, it can become busy before it receives the token back; but if this happens, P_0 cannot become idle again until after it receives the token.

Formally, a *state of a process* P_i is a pair (s_i, v_i), where $s_i \in \{\text{idle, busy}\}$ and $v_i \in \{0, 1\}$.

A *system state* is a string of n pairs $(s_0, v_0)\ (s_1, v_1) \ldots (s_{n-1}, v_{n-1})$, where each (s_i, v_i) is a state of process P_i.

The possible activities of each process P_i are defined by a number of state transitions; each of which has the form:

(present state of $P_{i-1 \bmod n}$) (present state of P_i) $\rightarrow$ (next state of P_i)

This form suggests that the processes are arranged on uni-directional ring, where each process can read its own state and the state of its left neighbor, and use the information to update its state.

The *state transitions* of P_0 are as follows.

t_0. *Start termination detection of current job*: (s, v) (busy, v) $\rightarrow$ (idle, v+1 mod 2)
t_1. *If no termination, continue current job*: (busy, u) (idle, v) $\rightarrow$ (busy, v)
t_2. *If termination, start a new job*: (s, v) (idle, v) $\rightarrow$ (busy, v)

The *state transitions* of P_i, i = 1, ..., n–1, are as follows.

t_3. *Finish anytime*:	(s, u) (busy, v)	→ (idle, u)
t_4. *Help neighbor*:	(busy, u) (idle, v)	→ (busy, v)
t_5. *Propagate token*:	(s, u) (idle, v)	→ (idle, u) provided u≠v

These transitions can be explained informally as follows. In transition t_0, process P_0 starts the termination detection when its state (s_0, v_0) satisfies the condition s_0 = busy $\wedge$ $v_0 = v_{n-1}$. (In fact, all the v_i's are guaranteed to be equal at this instant, as shown shortly.) The termination detection is started by making s_0 = idle and $v_0 = v_{n-1} + 1 \bmod 2$. This "new" value of v_0 is propagated to the v_i's of other processes (transition t_5), after each of them finishes execution and becomes idle (transition t_3). Note that the propagation of this new value simulates the circulation of a token. If the new value of v_0 reaches v_{n-1} while s_0 remains idle, process P_0 detects the termination of the current job and starts the next job (transition t_2). If, on the other hand, the new value reaches v_{n-1} after s_0 becomes busy (transition t_1), termination is not detected at this time, and another termination detection is tried at some later time.

Formally, the system has the usual *interleaving semantics* where exactly one state transition, selected arbitrarily from those that are currently being enabled, is executed at a time.

We are now ready to define the concept of a safe system state, then use this concept to state and verify three useful properties of the system: liveness, safety, and integrity. Informally, liveness means that at each system state, safe or not, at least one state transition is enabled and can be executed. Safety means that any system state that is reachable from a safe state is safe. In other words, once the system is in a safe state, its progress can only be through safe states. Integrity means that as the system progresses through safe states, it will detect termination when and only when termination does occur. Later in the section, we show that the system is unstable, i.e. if it starts at some unsafe state, it can progress indefinitely through unsafe states.

A system state $(s_0, v_0) \ldots (s_{n-1}, v_{n-1})$ is called *safe* iff it satisfies the following two conditions:

0. $(s_0 = \text{idle}) \Rightarrow$
   $(\exists m : 0 \le m < n : (\forall i : 0 \le i \le m : s_i = \text{idle} \wedge v_i = v_0) \wedge (\forall i : m < i < n : v_i = v_0 - 1 \bmod 2))$
1. $(s_0 = \text{busy}) \Rightarrow$
   $(\exists m : 0 \le m < n : (\forall i : 0 \le i \le m : v_i = v_0) \wedge (\forall i : m < i < n : v_i = v_0 - 1 \bmod 2))$

Theorem 1: (*Liveness of Original System*) At least one state transition is enabled at each system state, whether safe or not.

Proof: Let $(s_0, v_0) \ldots (s_{n-1}, v_{n-1})$ be any system state. If there is some i, i = 1, …, n–1, such that s_i = busy, then transition t_3 of P_i is enabled at that state. Moreover, if there is some i, i = 1, …, n–1, such that s_i = idle and $v_i \neq v_{i-1}$ then transition t_5 of P_i is enabled at that state. Therefore, we need only to consider states that satisfy

$$(\forall i : 1 \le i < n : s_i = \text{idle} \wedge v_i = v_0).$$

For any such state, if s_0 = idle then transition t_2 is enabled at that state. Otherwise, s_0 = busy and transition t_0 is enabled at that state. ❑

Theorem 2: (*Safety of Original System*) Any system state that is reachable from a safe state is safe.

Proof: It is sufficient to show that if S is a safe state of the system, and if a state S' follows S over some state transition t, then S' is safe. The proof is by considering the six cases $t = t_0, \ldots, t = t_5$. ❑

Theorem 3: (*Integrity of Original System*) Starting at any *safe* state,

a. transition t_2 is executed, only when current job is completed, and
b. when current job is completed, t_2 is eventually executed.

Proof:

(*Part a*) If t_2 is executed at a safe state, then at that state: s_0 = idle and $v_0 = v_{n-1}$. Therefore, this state, being safe, is in the form: (idle, v_0)… (idle, v_0), i.e. current job is completed at that state.

(*Part b*) If current job is completed at a safe state $(s_0, v_0) \ldots (s_{n-1}, v_{n-1})$, then $(\forall i : 0 \leq i < n : s_i = \text{idle})$. Since this state is safe, it satisfies the following condition.

$$(\exists m : 0 \leq m < n : (\forall i : 0 \leq i \leq m : v_i = v_0) \wedge (\forall i : m < i < n : v_i = v_{0-1} \bmod 2))$$

The only transition that can be executed starting at this state is t_5. This transition can be executed at most n–m–1 times yielding at the end a state in the form (idle, v_0) ... (idle, v_0) at which only t_2 can be executed. ❑

From Theorem 1, the system is guaranteed to progress indefinitely regardless of whether its starting state is safe. From Theorems 2 and 3, if the system starts at a safe state, it will progress indefinitely through safe states, and will continue to detect termination when and only when termination does occur. Unfortunately, as we show next, if the system starts at an unsafe state, it can progress indefinitely through unsafe states while executing the critical transition t_2 infinitely many times. (Notice that we have identified t_2 as the critical transition of the system because it is the transition that performs the system's main function, namely detecting the termination of the current job and starting the next job.)

To show this *instability*, assume that a system with four processes is at the unsafe state

(idle, 0) (busy, 1) (idle, 0) (idle, 0).

Starting from this state, the following scenario keeps the system within unsafe states indefinitely:

After executing t_2, the state becomes: (busy, 0) (busy, 1) (idle, 0) (idle, 0).
After executing t_5, the state becomes: (busy, 0) (busy, 1) (idle, 1) (idle, 0).
After executing t_3, the state becomes: (busy, 0) (idle, 0) (idle, 1) (idle, 0).
After executing t_4, the state becomes: (busy, 0) (busy, 0) (idle, 1) (idle, 0).
After executing t_0, the state becomes: (idle, 1) (busy, 0) (idle, 1) (idle, 0).
After executing t_5, the state becomes: (idle, 1) (busy, 0) (idle, 1) (idle, 1).

This last state is the "mirror" of the initial state with each "0" being replaced by "1", and vice versa. By repeating the same sequence of transitions over and over, the system detects termination infinitely many times, even though termination never occurs.

In the next section, we present a *self-stabilizing* version of this system, i.e. one that if started at any unsafe state is guaranteed to converge to a safe state within a finite number of its critical transitions.

3. A Slow Convergence/Fast Response System

The system we consider in this section is identical to the previous system except that the value of each v_i ranges from 0 to n–1 instead of from 0 to 1. (Recall that n is the number of processes in the system.) The system transitions remain the same as before except for t_0. For convenience, we list here all the state transitions of the system.

The *state transitions* of process P_0 are as follows:

t_0.	*Start termination detection*:	(s, v) (busy, v)	→	(idle, v+1 mod n)
t_1.	*Continue current job*:	(busy, u) (idle, v)	→	(busy, v)
t_2.	*Start new job*:	(s, v) (idle, v)	→	(busy, v)

The *state transitions* of each P_i, i = 1, ..., n–1, are as follows:

t_3.	*Finish anytime*:	(s, u) (busy, v)	→	(idle, u)
t_4.	*Help neighbor*:	(busy, u) (idle, v)	→	(busy, v)
t_5.	*Propagate token*:	(s, v) (idle, v)	→	(idle, u) provided $u \neq v$

A system state $(s_0, v_0) \ldots (s_{n-1}, v_{n-1})$ is called *safe* iff it satisfies the following two conditions:

0. $(s_0 = \text{idle}) \Rightarrow$
   $(\exists m : 0 \leq m < n : (\forall i : 0 \leq i \leq m : s_i = \text{idle} \wedge v_i = v_0) \wedge (\forall i : m < i < n : v_i = v_0 - 1 \bmod n))$
1. $(s_0 = \text{busy}) \Rightarrow$
   $(\exists m : 0 \leq m < n : (\forall i : 0 \leq i \leq m : v_i = v_0) \wedge (\forall i : m < i < n : v_i = v_0 - 1 \bmod n))$

Based on this definition, the properties of liveness, safety, and integrity can be stated and verified for this system in the same way that these properties have been stated and verified for the original system. (See Theorems 1, 2, and 3.) We now concentrate on establishing that the current system is self-stabilizing and that its convergence span is O(n). But first we need to identify the critical transition(s) of

the system. The function of the system is to detect termination of the current job before starting the execution of the next job. Thus, because t_2 is the transition where termination of the current job is detected and execution of the next job is started, it is reasonable to identify t_2 as the system's critical transition. Based on this identification, we can now state and verify two theorems concerning self-stabilization and convergence span of the system.

Theorem 4: (*Self-Stabilization of Second System*) Starting at any unsafe state, the system will reach a safe state before executing t_2 for the $(n+1)^{th}$ time.

Proof: Assume that the system starts at an unsafe state

$$(s_0, v_0) \ldots (s_{n-1}, v_{n-1})$$

There are two cases to consider:

Case 0 (for each i, $1 \le i < n$, $v_i \neq v_0$): In this case, neither t_0 nor t_2 can be executed at the starting state. Executing any of the other transitions yields states in the form:

$$(r_0, v_0) \ldots (r_{m-1}, v_0)\ (r_m, v_m) \ldots (r_{n-1}, v_{n-1})$$

where (i) for each i, $m \le i < n$, $v_i \neq v_0$, and
(ii) if $r_0 = \text{idle}$ then $r_0 = r_1 = \ldots = r_{m-1} = \text{idle}$.

So long as $m < n$, neither transition t_0 nor t_2 can be executed. If t_2 is eventually executed for the first time, then prior to its execution, the system must have reached a state in the following form (this is the same as the previous form except that $m = n$ and $r_0 = \text{idle}$):

$$(\text{idle}, v_0)\ (\text{idle}, v_0) \ldots (\text{idle}, v_0)$$

Such a state is safe. In other words, the system will reach a safe state before the first execution of t_2.

Case 1 (for some i, $1 \le i < n$, $v_i = v_0$): Because each of the n v_i's has a value from 0 to n–1, and because at least two of them have equal values, there is some integer m, $0 \le m < n$, such that for each i, $0 \le i < n$, $v_i \neq m$. Of all the state transitions, only t_0 can increase the set of values of the v_i's. Because each execution of t_0 increments v_0 by

one, and because each two successive executions of t_2 must be separated by one execution of t_0, then before n executions of t_2, the system will reach a state where $v_0 = m$, and for every i, $1 \le i < m$, $v_i \neq m$. The argument of Case 0 (above) can now be applied to show that the system will reach a safe state before the next execution of t_2. ❑

Theorem 5: (*Convergence Span of Second System*) The convergence span of the system is O(n).

Proof: From Theorem 4, the convergence span is at most O(n); thus it remains to show that the convergence span of the system is at least O(n). This is done by exhibiting a computation of the system along which the critical transition t_2 is executed (n–1) times before the system reaches a safe state. Assume that the system starts at an unsafe state:

$$(s_0, v_0)\ (s_1, v_1)\ \ldots\ (s_{n-1}, v_{n-1})$$

where $s_1 = \text{busy}$,
for each i, $i \neq 1$, $s_i = \text{idle}$,
$v_{n-1} = v_0$, and
for each i, $2 \le i \le n-1$, $v_{n-i} = (i-1) + v_{n-1}$

At that state, the sequence of transitions $t_2.t_0.t_5$ can be executed leading the system to the unsafe state:

$s_1 = \text{busy}$,
for each i, $i \neq 1$, $s_i = \text{idle}$,
$v_{n-2} = v_{n-1} = v_0$, and
for each i, $3 \le i \le n-1$, $v_{n-i} = (i-2) + v_{n-1}$

At that state, the sequence of transitions $t_2.t_0.t_5.t_5$ can be executed leading the system to the unsafe state:

$s_1 = \text{busy}$,
for each i, $i \neq 1$, $s_i = \text{idle}$,
$v_{n-3} = v_{n-2} = v_{n-1} = v_0$, and
for each i, $4 \le i \le n-1$, $v_{n-i} = (i-3) + v_{n-1}$

This can be repeated a total of n–2 times leading the system to the unsafe state:

s_1 = busy,
for each i, $i \neq 1$, s_i = idle, and
$v_1 = v_2 = \ldots = v_{n-1} = v_0$.

At that state, t_2 can be executed for the $(n-1)^{th}$ time. This computation establishes that the convergence span is at least O(n), and the theorem follows. ❑

The responsiveness of this system can be measured by the maximum number of transitions which can be executed from the time a job is terminated until the time its termination is detected. Formally, the *response span* of the system can be defined as the maximum number of state transitions in a computation that (i) starts from a safe state where each s_i = idle, (ii) ends with a safe state of the form (idle, v) ... (idle, v), and (iii) does not reach a state of the form (idle, v) ... (idle, v) in the middle. Based on this definition, the response span for the system is O(n), similar to its convergence span.

In the next section, we present another termination-detection system whose convergence span is O(1), and whose response span is $O(n^2)$. The existence of these two systems demonstrates the potential tradeoff between convergence and response in self-stabilizing systems.

4. A Fast Convergence/Slow Response System

The idea for the third system is simple. Process P_0 circulates a binary token to detect termination, as in the original system. However, unlike the original system, P_0 sends the binary token and receives it n successive times before termination is finally detected. To keep track of the number of times the token has been circulated so far, P_0 is provided with a local n-valued counter named z. Therefore, a *state* of P_0 is a triple (s, v, z), and a *state of each other* P_i is a pair (s, v), where $s \in$ {idle, busy}, $v \in$ {0, 1}, and $z \in \{0, 1, \ldots, n-1\}$. A *system state* is a string

$$(s_0, v_0, z)\ (s_1, v_1) \ldots (s_{n-1}, v_{n-1})$$

where (s_0, v_0, z) is a state of P_0, and for each i, $i = 1, \ldots, n-1$, the pair (s_i, v_i) is a state of P_i.

The *state transitions* of P_0 are as follows:

t_0.	*Start detection*:	(s, v) (busy, v, z)	→	(idle, v+1 mod 2, 0)
t_{0a}.	*Continue detection*:	(s, v) (idle, v, z)	→	(idle, v+1 mod 2, z+1 mod n) provided z < n–1
t_1.	*Continue current job*:	(busy, u) (idle, v, z)	→	(busy, v, z)
t_2.	*Start new job*:	(s, v) (idle, v, n–1)	→	(busy, v, n–1)

The *state transitions* of each P_i, i = 1, ..., n–1, are as follows:

t_3.	*Finish anytime*:	(s, u) (busy, v)	→	(idle, u)
t_4.	*Help neighbor*:	(busy, u) (idle, v)	→	(busy, v)
t_5.	*Propagate token*:	(s, u) (idle, v)	→	(idle, u) provided u≠v

Notice that process P_1 needs only to read the first two components "s" and "v" of the state of process P_0. The third component "z" is read and written by P_0 only.

In order to verify the correctness of this system, we first define the concept of a safe state of the system. This concept can then be used in stating and verifying the five system properties: liveness, safety, integrity, self-stabilization, and convergence span. Proofs for the first three properties are similar to our proofs of similar properties in the original system; see Theorems 1, 2, and 3 above. We concentrate on proving self-stabilization and convergence span.

A system state $(s_0, v_0, z)\ (s_1, v_1)\ \ldots\ (s_{n-1}, v_{n-1})$ is called *safe* iff it satisfies the following condition:

$(s_0 = \text{idle}) \Rightarrow$
$(\exists m : 0 < m \leq n :$
$(\forall i : 0 \leq i < m : s_i = \text{idle}) \wedge$
$(m \neq n \Rightarrow s_m = \text{busy}) \wedge$
$((m \neq n \wedge v_0 \neq v_{n-1}) \Rightarrow (\text{number of k's where } m < k < n \text{ and } v_k \neq v_{k-1}) \leq n - z - 1) \wedge$
$((m \neq n \wedge v_0 = v_{n-1}) \Rightarrow (\text{number of k's where } m < k < n \text{ and } v_k \neq v_{k-1}) < n - z - 1)$
$)$

Some explanation of this definition is in order. Any system state where s_0 = busy is trivially safe. When s_0 is changed from busy to idle, by transition t_0, the resulting

system state is still safe because t_0 makes s_0 = idle and z = 0, and so the resulting system state satisfies

$$(\exists m : 0 < m \leq n : (\forall i : 0 \leq i < m : s_i = \text{idle}) \wedge (m \neq n \Rightarrow s_m = \text{busy}) \wedge ((\text{number of k's where } m < k < n) < n - z - 1))$$

which indicates that the state is safe. Notice that when a safe state satisfies the condition $s_0 = \text{idle} \wedge v_0 = v_{n-1} \wedge z = n - 1$, then

$$(\forall i : 0 \leq i < n : s_i = \text{idle})$$

at that state. Therefore, the condition $s_0 = \text{idle} \wedge v_0 = v_{n-1} \wedge z = n - 1$ can be used to detect termination: it is in fact the "guard" for the termination detection transition t_2.

Based on this definition of safe states, and assuming that transition t_2 is the only *critical* transition in the system, the system properties of self-stabilization and convergence span can be stated and verified as follows.

Theorem 6: (*Self-Stabilization of Third System*) Starting at any unsafe state, the system will reach a safe state before executing t_2 for the second time.

Proof: The state of process P_0 following the first execution of transition t_2 is in the form (busy, v, n–1). This state will persist until transition t_0 is executed, and so long as this state persists, no further execution of t_2 is possible. An execution of t_0 will lead the system to a state in the form:

$$(\text{idle}, v_0, 0)\ (s_1, v_1) \ldots (s_{n-1}, v_{n-1}).$$

It is straightforward to show that this state is safe. ❑

Theorem 7: (*Convergence Span of Third System*) The convergence span of the system is O(1).

Proof: From Theorem 6, the convergence span is at most one; this is sufficient to establish the theorem. ❑

The *response span* for this system can be defined as the maximum number of state transitions in a computation that (i) starts with a safe state where each s_i = idle, (ii) ends with a safe state (idle, v, n–1) (idle, v) ... (idle, v), and (iii) does not reach a state (idle, v, n–1) (idle, v) ... (idle, v) in the middle. Based on this definition, the response span for the system is $O(n^2)$.

5. The As-You-Like-It System

The convergence span for our second system is O(n), the same as its response span, while the convergence span for our third system is O(1) and its response span is $O(n^2)$. Thus, in going from the second system to the third, the convergence span was reduced by some factor, n, and the response span was increased by the same factor. These observations suggest the existence of a family of termination-detection systems: the convergence span for each member in the family is O(n/k), and its response span is O(nk), where k = 1, ..., n. If such a family exists, our second and third systems represent the two extreme members in the family with k = 1 and k = n, respectively. In this section, we show that such a family of systems does exist; in particular we define and prove the correctness of a typical, i.e., the k^{th}, member in one such family.

This system is a combination of our second and third systems. Like the second system, it circulates an n-valued token; and like the third system, the token is circulated a number of times (specifically k times) before termination can be detected. Thus if k = 1, the system becomes very similar to the second system, and if k = n, the system becomes very similar to the third system.

As before, we consider a system of n processes $P_0, \ldots, P_{n-1}$. A *state of* P_0 is a triple (s, v, z), and a *state of every other* P_i is a pair (s, v), where s ε {idle, busy}, v ε {0, 1, ..., n–1}, z ε {0, 1, ..., k–1}, and k is an integer whose value is at least one, and at most n.

The *state transitions* of P_0 are as follows:

t_0.	*Start detection*:	(s, v) (busy, v, z)	→	(idle, v+1 mod n, 0)
t_{0a}.	*Continue detection*:	(s, v) (idle, v, z)	→	(idle, v+1 mod n, z + 1 mod k) provided z < k–1
t_1.	*Continue current job*:	(busy, u) (idle, v, z)	→	(busy, v, z)
t_2.	*Start new job*:	(s, v) (idle, v, k–1)	→	(busy, v, k–1)

The *state transitions* of each P_i, i = 1, ..., n–1, are as follows:

t_3.	*Finish anytime*:	(s, u) (busy, v)	→	(idle, u)
t_4.	*Help neighbor*:	(busy, u) (idle, v)	→	(busy, v)
t_5.	*Propagate token*:	(s, u) (idle, v)	→	(idle, u) provided u≠v

Notice that as in the third system, process P_1 does not need to read the third component of the state of P_0.

A system state $(s_0, v_0, z)\ (s_1, v_1)\ \ldots\ (s_{n-1}, v_{n-1})$ is called *safe* iff it satisfies the same two conditions for a safe state in the second system. Thus, whether a system state is safe does not depend on the value of z.

Based on this definition, the system properties of liveness, safety, and integrity can be stated and verified in the same way they have been stated and verified for the original system. Next, we state and verify the system properties of self-stabilization and convergence span under the assumption that t_2 is the only critical transition in the system.

Theorem 8: (*Self-Stabilization of Fourth System*) Starting at any unsafe state, the system will reach a safe state before executing t_2 for the $((n/k) + 1)^{th}$ time.

Proof: Assume that the system starts at an unsafe state $(s_0, v_0, z)\ (s_1, v_1)\ \ldots\ (s_{n-1}, v_{n-1})$. There are two cases to consider.

Case 0 (for each i, $1 \le i < n$, $v_i \neq v_0$): In this case, none of the transitions t_0, t_{0a}, and t_2 can be executed at the starting state. Executing any of the other transitions yields states in the form:

$$(r_0, v_0, z)\ (r_1, v_0)\ \ldots\ (r_{m-1}, v_0)\ (r_m, v_m)\ \ldots\ (r_{n-1}, v_{n-1})$$

where (a) for each i, $m \le i < n$, $v_i \neq v_0$, and
(b) if r_0 = idle then $r_0 = r_1 = \ldots = r_{m-1}$ = idle.

So long as m < n, then none of the transitions t_0, t_{0a}, and t_2 can be executed. If t_2 is eventually executed for the first time, then prior to its execution the system must

have reached a state in the form (this form is the same as the previous one except that m = n):

(idle, v_0, k–1) (idle, v_0) ... (idle, v_0)

such a state is safe. Therefore, the system will reach a safe state before the first execution of t_2.

Case 1 (for some i, $1 \le i < n$, $v_i = v_0$): In this case, there is some integer m, $0 \le m < n$, such that for each i, $0 \le i < n$, $m \ne v_i$. Because each execution of t_0 or t_{0a} increments v_0 by one, and because each two successive executions of t_2 must be separated by at least one execution of t_0 and (k–1) executions of t_{0a}, then before (n/k) executions of t_2, the system will reach a state where (a) $v_0 = m$, and (b) for each i, $1 \le i < n$, $v_i \ne m$. The argument of Case 0 can now be applied to show that the system will reach a safe state before the next execution of t_2. ❑

The following theorem can be proved using an argument similar to that of Theorem 5.

Theorem 9: (*Convergence Span of Fourth System*) The convergence span of the system is O(n/k).

The *response span* for this system can be defined as the maximum number of state transitions in a system computation that (i) starts with a safe state where each s_i = idle, (ii) ends with a safe state of the form (idle, v, k–1) (idle, v) ... (idle, v), and (iii) does not have a state of the form (idle, v, k–1) (idle, v) ... (idle, v) in the middle. Based on this definition, the response span for the system is O(nk).

6. Concluding Remarks

The objective of this paper is to point out the tradeoffs between the convergence span of a self-stabilizing system and its response span: either span can be reduced by some factor on the expense that the other span is increased by the same factor. We have demonstrated these tradeoffs in a class of termination-detection systems, but we firmly believe that similar tradeoffs are available in many other classes of systems.

The tradeoffs between the convergence and response spans can be roughly explained as follows: Assume that at most m transitions, for some m, can be executed before a self-stabilizing system reaches a safe state. Also assume that at least r non-critical transitions are to be executed between every pair of successive executions of critical transitions. Thus the convergence span of the system is at most m/r, and its response span is at least r (assuming that at least one critical transition is executed in going from the initial state to the target state of the response span). Now, if r can be increased by some factor k while m remains unchanged, the convergence span of the system is reduced by a factor k, and the response span is increased by the same factor. This is the basic idea that we have used in developing the as-you-like-it system: for that system $m = n^2$, $r = nk$ where k is a parameter that can take any value from 1 to n.

Our generalization of the notion of self-stabilization is noteworthy. Indeed, without this generalization, none of the systems that we have discussed in this paper would have been regarded as self-stabilizing, as they should be. This generalization suggests the following condition on the critical transitions of a self-stabilizing system. Starting from every state of the system, there is a computation where some critical transition is executed. This condition prevents the undesirable situation of a self-stabilizing system being within its unsafe states indefinitely, but as it cannot execute any critical transitions, it does not have to converge to a safe state to be self-stabilizing. It is straightforward to show that in each of our self-stabilizing systems, the critical transition t_2 satisfies this condition.

The class of termination-detection systems that we have discussed in this paper is interesting in its own right. The important features of these systems can be summarized by the following table:

System	Convergence Span	Response Span	Method of Termination-Detection
original	∞	n	two-valued token
second	n	n	n-valued token
third	1	n^2	two-valued token + n-valued counter in P_0
as-you-like-it	n/k	nk	n-valued token + k-valued counter in P_0, k can take any value from 1 to n

Finally, we observe that the original system, which is not self-stabilizing, is comparable in many ways with each of the other systems which are self-stabilizing. This observation suggests that systems can usually be made self-stabilizing without complicating them in a major way. This conclusion is in complete agreement with our previous experience with self-stabilization [1, 7].

Acknowledgements: We have discussed this work with Fred Schneider at some point; his encouraging comments have convinced us to strive for the as-you-like-it system. We would also like to thank Jim Anderson and Paul Attie for reading an earlier version of the paper and suggesting a number of improvements in the presentation.

7. References

1. G. M. Brown, M. G. Gouda, and C. L. Wu, "Token Systems that Self-Stabilize," *IEEE Trans. on Computers*, to appear, 1988.

2. K. M. Chandy and J. Misra, *Parallel Program Design: A Foundation*, Addison-Wesley Publishing, 1988.

3. E. W. Dijkstra, "Self-Stabilizing Systems om Spite of Distributed Control," *CACM*, Vol. 17, No. 11, Nov. 1974, pp. 643-644.

4. E. W. Dijkstra, W. H. J. Feijen, and A. J. M. van Gastern, "Derivation of a Termination Detection Algorithm for Distributed Computations," *Information Processing Letters*, Vol. 16, 1983, pp. 217-219.

5. N. Francez, "Distributed Termination, *ACM TOPLAS*, Vol. 2, No. 1, 1980, pp. 42-55.

6. M. G. Gouda, "Distributed State Exploration for Protocol Validation," Tech. Report 185, Dept. of Computer Sciences, Univ. of Texas at Austin, 1981.

7. M. G. Gouda, "The Stabilizing Philosopher: Asymmetry by Memory and by Action," Tech. Report 87-12, Dept. of Computer Sciences, Univ. of Texas at Austin, 1987. Also, *Science of Computer Programming*, to appear.