The Triumph and Tribulation of System Stabilization

Mohamed G. Gouda

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

Abstract. We give a concise outline of the theory of system stabilization. Our primary objective is to demonstrate the richness, depth, and ultimately the utility of this beautiful theory. Our secondary objective is to identify a number of problems that arise in the theory, and so highlight several research directions that can be pursued in the future. The stabilization of a system is defined as the ability of the system to converge to a closed (under execution) set of system states. We identify two forms of convergence (strong and weak) and two forms of closed sets of states (strong and weak), and so we end up with four forms of stabilization. The outlined theory is based on these four forms of stabilization.

Dedication. This paper is dedicated to the memory of my grandparents Asma (1897-1972) and Ahmed (1864-1967): on my mind and in my heart.

1 Introduction

The subject of this paper is the theory of system stabilization. This is a relatively new theory, at least in the realm of computing science. Hence, only few results have been attained in it. On the other hand, the evidence suggests that it is an intriguing theory, intellectually satisfying and widely applicable, and what more can a scientist ask for?

The theory of system stabilization deals with the ability of a system, starting from any state, to converge to a specific set of states that is closed under system execution. This theory is built around three important concepts: closure (under system execution), convergence, and stabilization. The main thrust of the theory is to answer two questions: how to verify the stabilization properties of a given system, and how to design a system such that it has a given set of stabilization properties. This theory can be used in explaining and investigating several important areas of computing systems: system initialization and reset, fault recovery, fault containment, and system adaptivity.

In this paper, we give an outline of the theory of system stabilization. The objective is to present the important subjects in the theory without addressing any of them in great detail. The presentation in the paper is intentionally concise so that the paper can fit in the available space. Our apologies are in order for not including proofs and examples.

2 Closure, Convergence, and Stabilization

A system is a pair (U, T), where U is a set of states, and T is a set of transitions: each transition in T is an ordered pair of states in U. For a transition (p, q) in T, state p is called the pre-state of the transition and state q is called the post-state of the transition.

A computation of a system (U, T) is a non-empty sequence (p.0, p.1, ...) such that the following three conditions hold.

i. Every element p.i in the sequence is a state in U.

ii. Every pair (p.i, p.(i+1)) in the sequence is a transition in T.

iii. Either the sequence is infinite, or it is finite and its last state is not a pre-state of any transition in T.

This abstract definition of a system applies to sequential, parallel, as well as distributed systems. In the case of a parallel or distributed system, however, a state of the system is a concatenation of the states of the system processes and the states of the communication channels between them, if any.

In a system (U, T), a subset P of U is strongly closed iff for every transition in T, if the pre-state of the transition is in P, then the post-state of the transition is in P.

In a system (U, T), a subset P of U is weakly closed iff every computation of the system, whose initial state is in P, has a non-empty suffix where each state is in P.

It follows from these definitions that every strongly closed subset is also weakly closed, but the reverse is not necessarily true. As an example, consider a system (U, T) where U = {p.0, p.1, p.2, p.3} and T = {(p.0, p.1), (p.1, p.1), (p.1, p.2), (p.2, p.3), (p.3, p.3)}. Each of the following subsets is strongly closed: the empty subset, {p.3}, {p.2, p.3}, {p.1, p.2, p.3}, and U. Thus, each of these subsets is also weakly closed. On the other hand, the subset {p.1, p.3} is weakly closed but not strongly closed.

One is tempted to think that each weakly closed subset P contains a strongly closed subset Q such that each system computation, whose initial state is in P, has a state in Q. If this was the case, the concept of weak closures would be redundant, and only strong closures would be needed. However, this is not the case. Consider the weakly closed subset {p.1, p.3} in the above example. Indeed, this subset contains a strongly closed subset, namely {p.3}. However, not every system computation whose initial state is in {p.1, p.3} has a state in {p.3}. In particular, the system computation (p.1, p.1, ...) does not have the state p.3. Thus, the weakly closed subset {p.1, p.3} cannot be replaced by the strongly closed subset {p.3}.

Let (U, T) be a system, P be a strongly closed subset of U, and Q be a (strongly or weakly) closed subset of U. Subset P strongly converges to Q iff every system computation, whose initial state is in P, has a state in Q.

Let (U, T) be a system, P be a strongly closed subset of U, and Q be a (strongly or weakly) closed subset of U. Subset P weakly converges to Q iff for

every state r that occurs in a system computation, whose initial state is in P, there is a system computation, whose initial state is r, and that computation has a state in Q.

Weak convergence is related to "probabilistic convergence" in the following sense. Consider a system (U, T), where the following three conditions hold.

i. P weakly converges to Q.

ii. The number of states in U is finite.

iii. For each state p in U, and each transition (p, q) in T, the probability, that transition (p, q) occurs in a computation whenever p occurs in that computation, is non-zero.

Then, with probability one, every computation of system (U, T), whose initial state is in P, has a state in Q.

This relationship between weak and probabilistic convergence can be proven as follows. Consider an arbitrary computation c of system (U, T). Because U is finite, at least one state r occurs infinitely many times in computation c. Because the probability that each transition, whose pre-state is r, occurs in c whenever r occurs in c is non-zero, then with probability one, each such transition occurs infinitely many times in c. Therefore, with probability one, each state that follows state r in system (U, T) occurs infinitely many times in c. This argument can be repeated a finite number of times to show that with probability one, each state that is reachable from state r in system (U, T) occurs infinitely many times in c. Because P weakly converges to Q, there is a state in Q that is reachable from r. With probability one, this state occurs (infinitely many times) in computation c.

It follows from this discussion that in systems with finite number of states, weak convergence can be easily converted to probabilistic convergence: Merely ensure that each transition has a non-zero probability of being executed, whenever the system reaches the pre-state of that transition. Henceforth, we ignore probabilistic convergence, and focus only on strong and weak convergence.

It is straightforward to show that if P strongly converges to Q then P weakly converges to Q, and that the reverse is not necessarily true. The definitions of strong and weak convergence yield four concepts.

i. Strong convergence to strong closure.

ii. Strong convergence to weak closure.

iii. Weak convergence to strong closure.

iv. Weak convergence to weak closure.

These four concepts are related as follows: (i) implies both (ii) and (iii), and each of (ii) and (iii) implies (iv).

Let P be a (strongly or weakly) closed subset of U in a system (U, T). System (U, T) strongly stabilizes to P iff U strongly converges to P. System (U, T) weakly stabilizes to P iff U weakly converges to P.

Because strong convergence implies weak convergence, it follows that strong stabilization implies weak stabilization. The definitions of strong and weak stabilization yield four concepts.

i. Strong stabilization to strong closure.

ii. Strong stabilization to weak closure.

iii. Weak stabilization to strong closure.

iv. Weak stabilization to weak closure.

These four concepts are related as follows: (i) implies both (ii) and (iii), and each of (ii) and (iii) implies (iv).

Having different degrees of stabilization is advantageous for two reasons. First, it allows us to compare and rate the stabilization properties of different systems. Second, because strong stabilization is costly or impossible to achieve in some cases, a weaker degree of stabilization can be sought in these cases.

Examples of systems that strongly stabilize to strong closures are given in [8], [9], [12], [13], [17], [18] [23], [29], and [30]. Examples of systems that strongly stabilize to weak closures are given in [11]. Examples of systems that strongly stabilize, with probability one, to strong closures are given in [7] and [15].

3 Laws of Stabilization

In this section, we state some laws concerning the closure, convergence, and stabilization properties of systems. These laws can be derived from the definitions in Section 2, and they can be used in reasoning about system stabilization. In these laws, let (U, T) be an arbitrary system, and let P, Q, R, and S be subsets of U.

i. Base:
U is strongly closed,
U strongly converges to U, and
(U, T) strongly stabilizes to U.

ii. Junctivity of Closure:
If both P and Q are strongly closed,
then both $P \cup Q$ and $P \cap Q$ are strongly closed.

If both P and Q are weakly closed,
then both $P \cup Q$ and $P \cap Q$ are weakly closed.

iii. Junctivity of Convergence:
If P strongly converges to Q,
and R strongly converges to S,
then $P \cup R$ strongly converges to $Q \cup S$,
and $P \cap R$ strongly converges to $Q \cap S$.

If P strongly converges to Q,
and R weakly converges to S,
then P∪R weakly converges to Q∪S,
and P∩R weakly converges to Q∩S.

If P weakly converges to Q,
and R weakly converges to S,
then P∪R weakly converges to Q∪S.

iv. Junctivity of Stabilization:
If (U, T) strongly stabilizes to P,
and (U, T) strongly stabilizes to Q,
then (U, T) strongly stabilizes to both P∪Q and P∩Q.

If (U, T) strongly stabilizes to P,
and (U, T) weakly stabilizes to Q,
then (U, T) weakly stabilizes to both P∪Q and P∩Q.

If (U, T) weakly stabilizes to P,
and (U, T) weakly stabilizes to Q,
then (U, T) weakly stabilizes to P∪Q.

v. From Closure to Convergence:
If P is strongly closed,
and Q is a weakly closed superset of P,
then P strongly converges to Q.

vi. From Convergence to Convergence
(Transitivity of Convergence):
If P strongly converges to Q,
and Q strongly converges to R,
then P strongly converges to R.

If P weakly converges to Q,
and Q weakly converges to R,
then P weakly converges to R.

vii. From Convergence to Stabilization:
If (U, T) strongly stabilizes to P,
and P strongly converges to Q,
then (U, T) strongly stabilizes to Q.

If (U, T) weakly stabilizes to P,
and P weakly converges to Q,
then (U, T) weakly stabilizes to Q.

The first law, base, demonstrates that the three properties of closure, convergence, and stabilization are meaningful. In particular, each of these properties applies to the set of all system states. The next three laws, junctivity, show how to use two instances of each of these properties to deduce two more instances of the same property. The last three laws, from-to, show how to use an instance of

closure, convergence, and stabilization to deduce a new instance of convergence, convergence, and stabilization, respectively.

4 Proof Obligations of Stabilization

In this section, we discuss the proof obligations needed for establishing the stabilization properties of systems. In what follows, let (U, T) be an arbitrary system, and let P, Q, Q.0, Q.1, ... be subsets of U. Also, let (D, $\ll$) denote a well-founded domain, where each decreasing, with respect to the less-than relation $\ll$, sequence of the elements in D is finite.

4.1 Proving Strong Closure

In order to establish that subset P is strongly closed in system (U, T), prove that for every state p in P, and every transition (p, q) in T, state q is in P.

4.2 Proving Weak Closure

In order to establish that subset P is weakly closed in system (U, T), exhibit an infinite sequence (Q.0, Q.1, ...) of subsets of U such that the following three assertions hold.

i. P is a subset of ($\cup$i, $0 \leq$ i, Q.i).

ii. For every k, ($\cup$i, $0 \leq$ i $<$ k, Q.i) is strongly closed.

iii. For every i, $0 \leq$ i, every computation of system (U, T), whose states are all in Q.i, has a non-empty suffix where every state is in P.

Two comments concerning this infinite sequence (Q.0, Q.1, ...) are in order. First, if there is an integer n such that for every i, i $\geq$ n, Q.i = Q.n, then the infinite sequence (Q.0, Q.1, ...) is in effect finite. Second, if for every i, i $\geq$ 0, Q.i = P, then P is strongly closed.

Assertion (i) can be proven using predicate calculus. Assertion (ii) can be proven as discussed in Section 4.1. Assertion (iii) can be proven for every i, $0 \leq$ i, as follows. Exhibit a well-founded domain (D, $\ll$) and a total function f : Q.i $\rightarrow$ D, such that for every transition (p, q) in T, where p and q are in Q.i,

(if q is not in P then f.q $\ll$ f.p) and
(if p is in P then q is in P)

4.3 Proving Strong Convergence

In order to establish that subset P strongly converges to Q in system (U, T), prove the following three assertions.

i. P is strongly closed.

ii. Q is (strongly or weakly) closed.

iii. Every computation of system (U, T), whose initial state is in P, has a state in Q.

Assertions (i) and (ii) can be proven as discussed in Sections 4.1 and 4.2. Assertion (iii) can be proven by exhibiting a well-founded domain (D, $\ll$) and a total function f : P $\rightarrow$ D, such that for every state p in P, and every transition (p, q) in T, if q is not in P then f.q $\ll$ f.p.

A modular proof of the property (P strongly converges to Q) in system (U, T) can be achieved by resorting to the transitivity of strong convergence. In particular, one can identify a finite sequence (Q.0, ... , Q.(n-1)) of subsets of U such that the following three conditions hold.

i. P = Q.0.

ii. Q = Q.(n-1).

iii. For every i, $0 \leq i < n-1$, Q.i strongly converges to Q.(i+1).

We refer to the sequence (Q.0, ... , Q.(n-1)) as a strong convergence stair.

4.4 Proving Weak Convergence

In order to establish that subset P weakly converges to Q in system (U, T) prove the following three assertions.

i. P is strongly closed.

ii. Q is (strongly or weakly) closed.

iii. For every state r that occurs in a system computation, whose initial state is in P, there is a system computation, whose initial state is r, and that computation has a state in Q.

Assertions (i) and (ii) can be proven as discussed in Sections 4.1 and 4.2. Assertion (iii) can be proven by exhibiting an infinite sequence (Q.0, Q.1, ...) of subsets of U such that the following three conditions hold.

a. P is a subset of ($\cup$i, $0 \leq$ i, Q.i).

b. Q.0 is a subset of Q.

c. For every Q.j, there is Q.i, $0 \leq i < j$, such that for every state p in Q.j, there is a transition (p, q) in T, where q is in Q.i.

A modular proof of the property (P weakly converges to Q) in system (U, T) can be achieved by resorting to the transitivity of weak convergence. In particular, one can identify a finite sequence (Q.0, ... , Q.(n-1)) of subsets of U such that the following three conditions hold.

i. P = Q.0.

ii. Q = Q.(n-1).

iii. For every i, $0 \leq i < n-1$, Q.i weakly converges to Q.(i+1).

We refer to the sequence (Q.0, ... , Q.(n-1)) as a weak convergence stair.

5 System Composition Made Easy

In this section, we show that multiple systems can be combined into a single composite system that has the same stabilization properties of the constituent systems. Toward this goal, we need to change our view of a system from being a pair (U, T) of states and transitions, to being a pair (V, C) of variables and actions. The two views (U, T) and (V, C) of a system are related as follows.

i. There is one-to-one correspondence between the states in U and the assignments of values to the variables in V. We adopt the notation $p \leftrightarrow x$ to mean that state p in U corresponds to assigning value x to the V variables.

ii. There is one-to-one correspondence between the transitions in T and the executions of actions in C. In particular, a transition (p, q) in T corresponds to an execution of action c in C iff there are two values x and y of the V variables such that the following three conditions hold.

   a. $p \leftrightarrow x$.

   b. $q \leftrightarrow y$.

   c. Executing action c when the V variables have value x yields the V variables with value y.

Let (V, C) be a system, where V is a set of variables and C is a set of actions, and let P be a subset of the system states. Subset P can be represented by a first order predicate, also denoted P for convenience, over the V variables such that for every system state p and every value x of the V variables, where $p \leftrightarrow x$,

(state p is in subset P
iff
assigning value x to the V variables makes predicate P true)

A predicate that represents a subset of the states of a system is called a state predicate of the system.

A state predicate of a system is called strongly (or weakly) closed iff the corresponding subset of system states is strongly (or weakly, respectively) closed. Having extended the concept of closure to state predicates, we can also extend the concepts of convergence and stabilization to state predicates.

Let (V.0, C.0), ... , (V.(n-1), C.(n-1)) be n systems, where each V.i is a set of variables, and each C.i is a set of actions. These systems can be combined into a system (V, C) as follows.

$$\begin{aligned} V &= V.0 \cup \ldots \cup V.(n\text{-}1) \\ C &= C.0 \cup \ldots \cup C.(n\text{-}1) \end{aligned}$$

In this case, the systems (V.0, C.0), ... , (V.(n-1), C.(n-1)) are called the constituent systems of the composite system (V, C). In the remainder of this section, we state sufficient conditions for ensuring that if the constituent systems are all strongly (or weakly) stabilizing, then the composite system is strongly (or weakly, respectively) stabilizing. To state these conditions, we need to introduce three concepts: input and output variables of a system and composition graph of a composite system. We discuss these three concepts in order.

Let (V, C) be a system. A variable u in V is called an input variable iff no action in C writes variable u. A variable in V, that is not an input variable, is called an output variable.

Let (V, C) be a composite system whose constituent systems are (V.0, C.0), ... , (V.(n-1), C.(n-1)). System (V, C) can be represented by a directed graph G, called the composition graph of (V, C), with two types of nodes. A node of the first type, called a v-node, represents a variable in the set $V.0 \cup \ldots \cup V.(n\text{-}1)$. A node of the second type, called a c-node, represents one of the action sets C.0, ... , or C.(n-1). In G, there is a directed edge from a v-node representing a variable u to a c-node representing an action set C.i iff u is an input variable in system (V.i, C.i). Also in G, there is a directed edge from a c-node representing an action set C.i to a v-node representing a variable u iff u is an output variable in system (V.i, C.i).

Theorem of Hierarchical Composition:
Let (V, C) be a composite system whose constituent systems are (V.0, C.0), ... , (V.(n-1), C.(n-1)).

If each constituent system (V.i, C.i) stabilizes to predicate P.i,
and the composition graph G of (V, C) satisfies the two conditions:
i. each v-node has at most one incoming edge in G, and
ii. there are no directed cycles in G,
then system (V, C) stabilizes to the predicate $P.0 \wedge \ldots \wedge P.(n\text{-}1)$.

Conditions (i) and (ii) in this theorem are somewhat severe. To relax condition (i), we introduce the concept of an isolation node.

Consider a v-node u with two incoming edges from two c-nodes C.i and C.j. Node u is called an isolation node iff for each value combination of the input variables of C.i and C.j, one of the following two conditions hold.

i. For each value of the output variables of C.i, executing each action in C.i keeps the variables of C.j unchanged.

ii. For each value of the output variables of C.j, executing each action in C.j keeps the variables of C.i unchanged.

In this definition, node u is called an isolation node because the stabilization of system (V.i, C.i) does not interfere with the stabilization of system (V.j, C.j),

or vice versa. We leave it for the reader to extend the definition of an isolation node to a v-node with more than two incoming edges.

Theorem of Acyclic Composition:
Let (V, C) be a composite system whose constituent systems are (V.0, C.0), ... , (V.(n-1), C.(n-1)).

If each constituent system (V.i, C.i) stabilizes to predicate P.i,
and the composition graph G of (V, C) satisfies the two conditions:
  i. each v-node with two or more incoming edges in G is an isolation node, and
  ii. there are no directed cycles in G,
then system (V, C) stabilizes to the predicate P.0 $\wedge$ ... $\wedge$ P.(n-1).

To relax condition (ii), we introduce the concept of a separation node. Consider a v-node u with one incoming edge from c-node C.i and one outgoing edge to c-node C.j. Node u is called a separation node iff for every value of variable u, one of the following two conditions hold.

i. For each value of the variables of C.i, other than u, executing each action in C.i keeps variable u unchanged.

ii. For each value of the variables of C.j, other than u, executing each action in C.j keeps the output variables of C.j unchanged.

In this definition, node u is called a separation node iff the value of variable u remains unchanged or system (V.j, C.j) has stabilized, in other words, the stabilization of system (V.i, C.i) does not affect the stabilization of system (V.j, C.j). We leave it for the reader to extend the definition of a separation node to a v-node with multiple incoming and outgoing edges.

Theorem of General Composition:
Let (V, C) be a composite system whose constituent systems are (V.0, C.0), ... , (V.(n-1), C.(n-1)).

If each constituent system (V.i, C.i) stabilizes to predicate P.i,
and the composition graph G of (V, C) satisfies the two conditions:
  i. each v-node with two or more incoming edges in G is an isolation node, and
  ii. each directed cycle in G has at least one separation node,
then system (V, C) stabilizes to the predicate P.0 $\wedge$ ... $\wedge$ P.(n-1).

A preliminary version of this method of system composition is presented in [19]. Other methods for composing stabilizing systems are presented in [24] and [26].

6 Applications of System Stabilization

Stabilization properties can be used in studying several areas of computing systems: initialization and reset, fault recovery, fault containment, and system adap-

tivity. Next, we discuss each of these four areas in turn.

6.1 Initialization and Reset

The stabilization properties of a system can be used to simplify the initialization and reset procedures for that system. This is beneficial especially for parallel and distributed systems whose initialization and reset procedures are usually very complicated.

Consider a system (V, C) whose initial or reset state is required to satisfy some state predicate Q. If a state predicate P strongly converges to Q in this system, then an initialization or reset procedure for the system is as follows. First, the system is initialized or reset to any state that satisfies predicate P. Second, the system is left to execute for some time until its current state satisfies Q. Note that if the system strongly stabilizes to Q, then the first step in this procedure is not needed.

Stabilizing systems for distributed reset are presented in [3] and [6].

6.2 Fault Recovery

Let (V, C) be a system, and let P and Q be two state predicates of that system. System (V, C) recovers from P to Q iff the following three conditions hold.

i. Each state of (V, C), that satisfies Q, satisfies P.

ii. Both P and Q are strongly closed in (V, C).

iii. P strongly converges to Q in (V, C).

In this definition of fault recovery, faults are not mentioned explicitly. Rather, the definition implies that each fault occurrence may change the state of the system from one that satisfies predicate Q to one that satisfies predicate P. Further occurrences of faults keep the system in states that satisfy predicate P. The ability of the system to recover from these fault occurrences is expressed by saying that P strongly converges to Q. In other words, when faults cease to occur for some time, the system returns to states that satisfy predicate Q during that period. Predicate P represents the system invariant when faults do occur, and predicate Q represents the system invariant when faults cease to occur.

One may feel uncomfortable about a definition of fault recovery that does not explicitly mention faults. If so, one can introduce faults to the above definition as follows. First, define a set F of faults, where each fault is an action that reads and writes the V variables in system (V, C). Second, add the requirement, that P is strongly closed in the fault system (V, F), to the above definition. Note that the fault system (V, F) is the same as the original system (V, C) except that set C of system actions is replaced by set F of fault actions. Third, the resulting definition is identified as system (V, C) recovers from F via P to Q.

This augmented definition of fault recovery is based on the assumption that the only effect of a fault on a system (V, C) is to change the current state of

(V, C). This assumption can always be realized by adding auxiliary variables to set V and augmenting some actions in set C. For example, to represent a fault that causes an action c in C to failstop and never be executed again, an auxiliary variable, named up, is added to set V and action c is augmented so that it cannot change the value of any variable when up = false. In this case, the failstop fault can be represented by the fault action up := false. Other types of faults such as Byzantine faults, stuck-at faults, and timing faults, can all be represented in the same way.

Consider a system (V, C) that recovers from P to Q. If P = true, then the fault recovery is called stabilizing, else it is called non-stabilizing. If P = S, then the fault recovery is called masking, otherwise it is called non-masking.

In choosing a state predicate P so that a system (V, C) recovers from P to Q, we are faced with two contradictory objectives. The first objective is to choose P as weak as possible (i. e. close to true) so that system (V, C) can recover from most faults. The second objective is to choose P as strong as possible (i. e. close to Q) so that during recovery the system is guaranteed to remain in states close to those satisfying Q. Fortunately, it is possible to achieve both these objectives. First, define two state predicates P.0 and P.1 of system (V, C), where P.0 is weak (i. e. close to true) and P.1 is strong (i. e. close to Q). Second, ensure that system (V, C) recovers from P.0 to Q and from P.1 to Q.

The above definition of fault recovery admits the following three laws.

i. Union of Fault Recovery:
   If system (V, C) recovers from P to Q,
   and system (V, C) recovers from R to Q,
   then system (V, C) recovers from P $\vee$ R to Q.

ii. Intersection of Fault Recovery:
   If system (V, C) recovers from P to Q,
   and system (V, C) recovers from P to R,
   then system (V, C) recovers from P to Q $\wedge$ R.

iii. Transitivity of Fault Recovery:
   If system (V, C) recovers from P to Q,
   and system (V, C) recovers from Q to R,
   then system (V, C) recovers from P to R,

For more details and many examples about this novel view of fault recovery, the reader is referred to [2], [4], and [14].

6.3 Fault Containment

Let (V, C) be a system and P and Q be two state predicates of (V, C). Also, let ef be a function that maps each state of (V, C) that satisfies predicate P, to a non-negative integer. System (V, C) contains P to Q for ef iff the following three conditions hold.

i. (V, C) recovers from P to Q.

ii. For every pair of states p and q that satisfy P, if there is an action in C whose execution at state p yields a state q, then ef.p $\geq$ ef.q.

iii. For every state p that satisfies S, ef.p = 0.

Function ef in this definition is called an effect of fault function. One example of an effect of fault function for a mutual exclusion system is as follows.

ef.p = max(0, x.p - 1)

where x.p is the number of processes in their critical sections at state p. Effect of fault functions satisfy the following law.

Effect of Fault Functions:
If system (V, C) contains P to Q for ef,
and system (V, C) contains P to Q for eg,
and x, y, and z are non-negative integers,
then system (V, C) contains P to Q for x*ef + y*eg + z*ef*eg.

6.4 System Adaptivity

An adaptive system is one whose computation adapts to the current state of its environment. As shown below, system adaptivity can be explained formally as a form of strong convergence. Before we define system adaptivity, we need to introduce the concept of a fixed point.

Let (V, C) be a system and Q be a strongly closed predicate of (V, C). Also, let U be a subset of V. Predicate Q is a fixed point for U iff starting at any state that satisfies Q, executing any action in C does not change the values of the variables in U. Note that if U is empty, then any strongly closed predicate is a fixed point for U.

Let (V, C) be a system and P and Q be two strongly closed predicates of (V, C), and let U be a subset of V. System (V, C) is adaptive from P to Q for U iff the following three conditions hold.

i. P is a boolean expression that involves only input variables in V.

ii. P strongly converges to Q.

iii. Q is a fixed point for U.

Consider a system (V, C) that is adaptive from P to Q for U. Each change in the state of the environment of (V, C) causes a corresponding change in the values of the input variables of (V, C). If the new values of the input variables make predicate P true, then the system eventually reaches states that satisfy predicate Q. System (V, C) stays within those states that satisfy Q at least until a later change in the state of the environment causes P to become false. While system (V, C) is within states that satisfy Q, the variables in set U have fixed values.

This concept of system adaptivity can be generalized as follows. Let (V, C) be a system, and let (P.0, ... , P.(r-1)) and (Q.0, ... , Q(r-1)) be two r-tuples of

strongly closed predicates of (V, C). Also, let (U.0, ... , U.(r-1)) be an r-tuple of subsets of V. System (V, C) is adaptive from (P.0, ... , P.(r-1)) to (Q.0, ... , Q.(r-1)) for (U.0, ... , U.(r-1)) iff the following three conditions hold.

i. Every P.i is a boolean expression that involves only input variables in V.

ii. Every P.i strongly converges to Q.i.

iii. Every Q.i is a fixed point for U.i.

An example of an adaptive system is an air-conditioner. When the measured temperature (of the environment) is more than the required temperature, the state of the air-conditioner becomes on and stays on until the measured temperature is less than or equal the required temperature. In this case, the state of the air-conditioner becomes off and stays off until the measured temperature is more than the required temperature, and the cycle repeats.

7 Tribulations of System Stabilization

The above theory of system stabilization comes with several problems. In this section, we briefly discuss three of these problems. The first problem of the theory of stabilization is that some well-known models of parallel and distributed systems cannot be used to represent stabilizing systems [20]. One example of such models is Petrinets.

An informal explanation of why effective Petrinets cannot be stabilizing is as follows. A stabilizing Petrinet N should be able to converge from an arbitrary state where there is a large number of tokens in each place to a state where the total number of tokens in the net is small. Thus, transition firings in N should consume more tokens than they produce. Unfortunately, this means that N can reach a state where there are not enough tokens to enable any transition in N. In other words, N can reach a deadlock state. (One way to avoid this deadlock state is to provide N with a special transition that can fire and produce tokens when it detects that N has no tokens.)

Other models that cannot be used to represent stabilizing systems are systems of CSP and finite state automata that communicate messages over unbounded channels. These are all well-known models of parallel, concurrent, and distributed systems, and the fact that they cannot represent stabilizing systems is disturbing at best.

The second problem of the theory of stabilization is that some stabilization properties of a system are fragile and can disappear if the system is transformed using some seemingly harmless system transformations [20]. It follows that if a system is designed to be stabilizing, then many natural implementations of that design may not be stabilizing. This observation suggests that implementation of stabilizing systems should proceed with care to ensure that the stabilization properties, achieved in the design, are preserved in the implementation.

The third problem of the theory of stabilization is the great difficulty that one encounters in verifying the stabilization properties of systems. It has been my experience that the ratio of the time to design a stabilizing system to the time to verify its stabilization properties is about one to ten. This is unacceptably low ratio considering that the ratio for verifying standard safety and progress properties is about one to three. Therefore, better proof systems are needed for verifying stabilization properties.

8 What to Do in the Next Ten Years

The theory of system stabilization is relatively new, and so the area has many issues and problems that merit further consideration and research. Some of these problems are mentioned in this section.

In Section 2, we introduced two forms of closure and two forms of convergence, and ended up with four forms of stabilization. Are there other meaningful forms of closure, convergence, and stabilization? To add some structure to this, otherwise open ended, problem, I propose that any new forms of closure, convergence, and stabilization should satisfy the laws in Section 3.

In Section 4, we discussed proof obligations for verifying the stabilization properties of systems. Then in Section 8, we complained that it is very hard to carry out such verifications. Is it possible to come up with stronger proof obligations that are easier to check? These stronger proof obligations would be useful even if they are only applicable to special classes of stabilization properties or special classes of systems. Also, what tools for automated verification of system stabilization?

In Section 5, we presented a method for combining systems while preserving their stabilization properties. This method is provably sound, but its effectiveness is yet to be investigated. In particular, the following questions needs to be addressed. What types of systems can be composed using this method? Are there other methods for composing other types of systems?

In Section 6, we discussed four applications of the theory of system stabilization. In all these applications, we used only strong closure and strong convergence. Can weak closure or weak convergence be used in these applications? Are there other applications of the theory of stabilization? Early efforts to answer this last question indicate that system diagnostics, system learning, network protocols [5], [21], [22], [28], and hardware design [1] can all be viewed as applications of the theory of stabilization.

Finally, we have ignored in the current paper the whole area of stabilization complexity. Early efforts in this area are reported in [10], [16], [25] and [27]. Nonetheless, this area remains largely uncharted, and is now ripe for thorough investigation.

Acknowledgments: I started to work on system stabilization in the spring of 1986. Since then, I was fortunate enough to have nineteen excellent coauthors who helped me a great deal in exploring this magnificent area. Those coauthors are M. Abadir, A. Arora, P. Attie, G. Brown, J. Burns, J. Cobb, J. Couvreur, S.

Dolev, M. Evangelist, N. Francez, F. Haddix, T. Herman, R. Howell, R. Miller, N. Multari, L. Rosier, M. Schneider, G. Varghese, and C. Wu. To each of these coauthors: Thank you for your help. I am also thankful to J. Cobb for reading earlier drafts of this paper and suggesting several improvements. My grandparents, who never had a formal education, were very excited when I told them I was planning to study engineering and science. At the time, I did not know what their excitement was all about. Ten years after they passed away, I started to understand. This paper is dedicated to their memory.

References

1. M. Abadir, M. G. Gouda, "The Stabilizing Computer," *Proceedings of the International Conference on Parallel and Distributed Systems*, Taiwan, pp. 90-96, 1992.

2. A. Arora, M. G. Gouda, "Closure and Convergence: A Foundation for Fault-Tolerant Computing," *IEEE Transactions on Software Engineering*, special issue on Software Reliability, Vol. 19, No. 3, pp. 1015-1027, November 1993.

3. A. Arora, M. G. Gouda, "Distributed Reset," *IEEE Transactions on Computers*, Vol. 43, No. 9, pp. 1026-1038, 1994.

4. A. Arora, M. G. Gouda, G. Varghese, "Constraint Satisfaction as a Basis for Designing Nonmasking Fault-Tolerance," in *Specification of Parallel Algorithms*, edited by G. E. Blelloch, K. M. Chandy, S. Jagannathan, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 18, 1994.

5. A. Arora, M. G. Gouda, T. Herman "Composite Routing Protocols," *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing*, December 1990.

6. B. Awerbuch, R. Ostrovsky, "Memory-efficient and Self-Stabilizing Network Reset," *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, 1994.

7. J. Beauquier, S. Cordier, S. Delaet, "Optimum Probabilistic Self-Stabilization on Uniform Rings," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

8. B. Bourgon, A. K. Datta, "A Self-Stabilizing Distributed Heap Maintenance Protocol," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

9. G. M. Brown, M. G. Gouda, C. L. Wu, "Token Systems that Self-Stabilize," *IEEE Transactions on Computers*, Vol. 38, No. 6, pp. 845–852, June 1989.

10. J. E. Burns, M. G. Gouda, R. E. Miller, "On Relaxing Interleaving Assumptions," *Proceedings of the MCC Workshop on Self-Stabilization*, Austin, Texas, 1989.

11. J. E. Burns, M. G. Gouda, R. E. Miller, "Stabilization and Pseudostabilization," *Distributed Computing*, special issue on Self-Stabilization, Vol. 7, No. 1, pp. 35-42, November 1993.

12. J. Couvreur, M. G. Gouda, N. Francez, "Asynchronous Unison," *Proceedings of the 12th International Conference on Distributed Computing Systems*, Tokyo, pp. 486-493, 1992.

13. E. W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control," *Communications of the ACM*, Vol. 17, No. 11, pp. 643-644, 1974.

14. S. Dolev, T. Herman, "SuperStabilizing Protocols for Dynamic Distributed Systems," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

15. S. Dolev, J. Welch, "Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

16. M. Evangelist, M. G. Gouda, "Convergence/Response Tradeoffs in Concurrent Systems," *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing*, December 1990.

17. S. Ghosh, A. Gupta, M. H. Karaata, S. V. Pemmaraju "Self-Stabilizing Dynamic Programming Algorithms on Trees," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

18. M. G. Gouda, "Stabilizing Observers", *Information Processing Letters*, Vol. 57, pp. 99-103, 1994.

19. M. G. Gouda, T. Herman, "Adaptive Programming," *IEEE Transactions on Software Engineering*, Vol. 17, No. 9, pp. 911–921, September 1991.

20. M. G. Gouda, R. R. Howell, L. E. Rosier, "The Instability of Self-Stabilization," *Acta Informatica*, Vol. 27, pp. 697–724, 1990.

21. M. G. Gouda, N. Multari, "Stabilizing Communication Protocols," *IEEE Transactions on Computing*, special issue on Protocol Engineering, Vol. 40, No. 4, pp. 448–458, April 1991.

22. M. G. Gouda, M. Schneider, "Maximum Flow Routing," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

23. C. Johnen, J. Beauquier, "Space-Efficient Distributed Self-Stabilizing Depth-First Token Circulation," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

24. S. Katz, K. J. Perry, "Self-Stabilizing Extensions for Message-Passing Systems," *Distributed Computing*, Vol. 7, pp. 17-26, 1993.

25. C. Lin, J. Simon, "Possibility and Impossibility Results for Self-Stabilizing Phase Clocks on Synchronous Rings," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

26. M. Schneider, "Self-Stabilization," *ACM Computing Surveys*, Vol. 25, No. 1, March 1993.

27. S. K. Shukla, D. J. Rosenkrantz, S. S. Ravi, "Observations on Self-Stabilizing Graph Algorithms for Anonymous Networks," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

28. J. Spinelli, R. G. Gallager, "Event Driven Topology Without Sequence Numbers," *IEEE Transactions on Communications*, Vol. 37, No. 5, pp. 468-474, 1989.

29. M. S. Tsai, S. T. Huang, "Self-Stabilizing Ring Orientation Protocols," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.

30. I. Yen, F. B. Bastani, "A Highly Safe Self-Stabilizing Mutual Exclusion Algorithm," *Proceedings of the Second Workshop on Self-Stabilizing Systems*, Technical Report, Department of Computer Science, University of Las Vegas, Las Vegas, Nevada, May 1995.