The Theory of Weak Stabilization*

Mohamed G. Gouda[1]

Department of Computer Sciences, The University of Texas at Austin
Austin, TX 78712-1188, U.S.A.
gouda@cs.utexas.edu

Abstract. We investigate a new property of computing systems called weak stabilization. Although this property is strictly weaker than the well-known property of stabilization, weak stabilization is superior to stabilization in several respects. In particular, adding delays to a system preserves the system property of weak stabilization, but does not necessarily preserve its stabilization property. Because most implementations are bound to add arbitrary delays to the systems being implemented, weakly stabilizing systems are much easier to implement than stabilizing systems. We also prove the following important result. A weakly stabilizing system that has a finite number of states is in fact stabilizing assuming that the system execution is strongly fair. Finally, we discuss an interesting method for composing several weakly stabilizing systems into a single weakly stabilizing system.

1 Introduction

There has been a growing interest in recent years to design and implement stabilizing computing systems. See for instance, [3], [4], [5], [6], and [7]. Unfortunately, stabilizing systems are difficult to implement in such a way as to preserve their stabilization properties. (This fact is often ignored in light of the immense intellectual pleasure that one can derive from designing such systems.) The main reason for this difficulty is that stabilization properties are extremely delay sensitive. The simple transformation of adding a small delay unit to a stabilizing system can yield this system non-stabilizing [1]. Because every system implementation is bound to add one or more delay units to the system being implemented, the implemented system often ends up being non-stabilizing.

This situation leaves the designers of stabilizing systems, who wish to implement their designs, with three options.

The first option is to identify all the delay units that may be added to the system during its implementation, then ensure that the system with the added delay units is still stabilizing. This option is not attractive because a system with many delay units has a relatively large state space, and the task of ensuring that such a system is stabilizing is usually hard.

The second option is to implement the system in a way that does not necessarily preserve its stabilization properties, then hope for the best, namely that

* This work is supported in part by DARPA contract F33615-01-C-1901.

A.K. Datta and T. Herman (Eds.): WSS 2001, LNCS 2194, pp. 114–123, 2001.

the implemented system will turn out to be "almost stabilizing". In this case, the system designer does not plan on validating this hope, and so the concept of "almost stabilization" is left vague and undefined.

The third option is to introduce a weaker version of the stabilization property, then show that this weak stabilization is delay insensitive. In this case, most sensible implementations of stabilizing (or even weak stabilizing) systems are guaranteed to be weakly stabilizing.

In this paper, we adopt this third option, and give a formal characterization of the weak stabilization property. In particular, we show that weak stabilization is delay insensitive and that it is a good approximation of the original stabilization property.

2 Stabilization and Weak Stabilization

A (computing) system is a nonempty set of variables, whose values are from predefined domains, and a nonempty set of actions that can be executed to update the values of the variables. Each action is of the form:

$\langle guard \rangle \rightarrow \langle assignment \rangle$

The $\langle guard \rangle$ is a Boolean expression over the system variables, and $\langle assignment \rangle$ is an assignment statement of the form:

$\langle variable \rangle := \langle expression \rangle$

The $\langle expression \rangle$ is an expression over the system variables and its value is from the domain of $\langle variable \rangle$.

For simplicity, we require that each variable of a system S be in the left-hand side of the assignment of at most one action of system S. In this case, the v action refers to the action where variable v is in the left-hand side of its assignment.

A state of a system S is a function that assigns a value to each variable of S. The value assigned to each variable is from the domain of that variable.

An action of a system S is enabled at a state p of S iff the guard of the action is true at state p. For simplicity, we assume that at least one action of a system S is enabled at each state of S.

A transition of a system S is a pair (p, q), where p and q are states of S and there is an action of S that is enabled at state p and executing this action starting at state p yields system S in state q.

A computation of a system S is an infinite sequence p.0, p.1, ... of S states such that every pair (p.i, p.(i+1)) of successive states in the sequence is a transition of S.

A state predicate of a system S is a function that has a Boolean value, true or false, at each state of S. Let true denote the state predicate whose value is true at each state of S, and false denote the state predicate whose value is false at each state of S.

A state p of a system S is a P state iff P is a state predicate of S whose value is true at state p.

Let P and Q be state predicates of a system S. Predicate P equals predicate Q, denoted P = Q, in system S iff P and Q have equal values at every state of S.

Predicate P implies predicate Q, denoted P ⇒ Q, in system S iff for every state p of system S, if P is true at p then Q is true at p.

A state predicate P of a system S is closed in S iff for each transition (p, q) of S, if p is a P state, then q is a P state.

A system S is stabilizing to a state predicate P iff the following two conditions hold. First, P is closed in S. Second, for every state p, every computation of S that starts at p has a P state.

A system S is weakly stabilizing to a state predicate P iff the following two conditions hold. First, P is closed in S. Second, for every state p, there is a computation of S that starts at p and has a P state.

Theorem 1. *If a system S is stabilizing to a state predicate P, then S is weakly stabilizing to P. The converse does not necessarily hold.*

Proof. Stabilization clearly implies weak stabilization. It remains to show that the converse does not necessarily hold. Consider a unidirectional token ring similar to that discussed in [2]. Assume that there are four or more actions in the ring and that the domain of values for each variable is 0..1. It is straightforward to show that this ring is weakly stabilizing but not stabilizing to an appropriate state predicate P.

3 Proof Obligations

In this section, we state proof obligations that can be used to prove that a system S is stabilizing, or weakly stabilizing, to a state predicate P. But first, we need to introduce the concepts of well-founded domain and ranking function.

A well-founded domain is a pair (D, >) where D is a set of elements and > is a total order relation over the elements of D such that each sequence of elements of D, that is decreasing with respect to the relation >, is finite.

A ranking function F of a system S is a function that assigns to each state p of S, a value F.p from a well-founded domain.

The following three theorems state proof obligations for stabilization and weak stabilization. Correctness of these theorems is straightforward.

Theorem 2. *(Closure of P)*

If *for every P state p of a system S and every transition (p, q) of S,*
q is a P state
then *P is closed in S.*

Theorem 3. *(Convergence to P)*

If *there is a ranking function F of a system S such that*
for every state p of S and every transition (p, q) of S
F.p > F.q or q is a P state
then *for every state p of S, every computation of S,*
that starts at p, has a P state.

Theorem 4. *(Weak Convergence to P)*

If *there is a ranking function F of a system S such that*
for every state p of S, there is a transition (p, q) of S where
F.p > F.q or q is a P state
then *for every state p of S, there is a computation of S*
that starts at p and has a P state.

Note that a ranking function for proving stabilization (or convergence) of a system S needs to be decreased by every action of S, whereas a ranking function for proving weak stabilization (or weak convergence) of S needs to be decreased by at least one action of S. Thus, a ranking function for proving stabilization of a system is stricter than one for proving weak stabilization of the same system.

4 Delay Insensitivity

In this section, we discuss how to transform a system by adding a delay to it, and show that in general such a transformation preserves weak stabilization of the system but not its stabilization.

Let v be a variable of a system S. System S can be transformed, by adding a delay to variable v, as follows.

i. Add to system S a new variable dv, whose domain of values is the same as that of v.
ii. Modify every action "g → s" of S by replacing every occurrence of v in the guard g and in the right-hand side of the assignment s by an occurrence of dv.
iii. Add the action "dv ≠ v → dv := v" to system S.

The resulting system after this transformation is denoted S<v>. The added variable dv in the transformed system can be thought of as a delayed version of variable v. The added dv-action in the transformed system can be thought of as the added delay to variable v.

Next, we discuss the effect of adding a delay to some variable of a system on the closed predicates of that system. Let P be a closed predicate of a system S. Is predicate P also closed in the transformed system S<v>? The answer to this question is "no" in general. This is because the added variable dv does not occur in P. Thus, the value of dv can be arbitrary at a P state of system S<v>. Starting from this state and executing an action where some variable that occurs in P is updated using variable dv can yield system S in a state where predicate P no longer holds. This shows that P is not closed in S<v>.

Though a predicate P that is closed in a system S is not necessarily closed in the transformed system S<v>, we show (in Theorem 5 below) that another predicate, related to P, is closed in S<v>. But first we need to introduce the concept of a state predicate being exclusive with respect to some variable in its system.

Let v be a variable of a system S, and P be a state predicate of S. Predicate P is v-exclusive in S iff for every P state p, if the v action in S, if any, is enabled at p, then no other action of S is enabled at p.

Theorem 5.

If *P is a closed state predicate of a system S, and*
P is v-exclusive in S,
then *the state predicate (P ∧ (dv = v ∨ G.v)) is closed in S<v>,*
where G.v is the negation of the disjunction of the guards of all actions, other than the dv action and the v action, in system S<v>.

Proof. Let p be a (P ∧ (dv = v ∨ G.v)) state of system S<v>, and assume that an execution of an action c of S<v> starting at state p yields the system in a state q. We need to show that q is a (P ∧ (dv = v ∨ G.v)) state. There are six cases to consider.

Case 1: *(p is a (P ∧ dv = v) state and c is the dv action)*

This case is not valid because the dv action is not enabled at p.

Case 2: *(p is a (P ∧ dv = v) state and c is the v action)*

In this case, P is true at state q because P is true at state p and P is closed in system S. Also, G.v is true at state p because p is a P state and P is v-exclusive in S. Predicate G.v remains true at state q because v, the only variable updated by action c, does not occur in G.v. Thus, q is a (P ∧ G.v) state.

Case 3: *(p is a (P ∧ dv = v) state and c is any other action)*

In this case, P is true at state q because P is true at state p and P is closed in system S. Also, dv = v at state q because dv = v at state p and neither dv nor v are updated by action c. Thus, q is a (P ∧ dv = v) state.

Case 4: *(p is a (P ∧ G.v) state and c is the dv action)*

In this case, P is true at state q because P is true at state p and dv, the only variable updated by action c, does not occur in P. Also, dv = v at state q because c is the dv-action. Thus, q is a (P ∧ dv = v) state.

Case 5: *(p is a (P ∧ G.v) state and c is the v action)*

In this case, P is true at state q because P is true at state p and P is closed in system S. Also, G.v is true at state p, and G.v remains true at state q because v, the only variable updated by action c, does not occur in G.v. Thus, q is a (P ∧ G.v) state.

Case 6: *(p is a (P ∧ G.v) state and c is any other action)*

This case is not valid because no action, other than the dv action and the v action, can be enabled at p where G.v is true.

Having established that the state predicate (P ∧ (dv = v ∨ G.v)) is closed in the transformed system S<v>, it seems reasonable to ask the following two questions.

Given that system S is stabilizing to P, is the transformed system S<v> stabilizing to (P ∧ (dv = v ∨ G.v))? Given that S is weakly stabilizing to P, is S<v> weakly stabilizing to (P ∧ (dv = v ∨ G.v))? The next two theorems answer these two questions with "not necessarily" and "yes" respectively.

Theorem 6.

If *P is a v-exclusive state predicate in a system S, and*
S is stabilizing to P,
then *S<v> is not necessarily stabilizing to (P ∧ (dv = v ∨ G.v)).*

Proof. We exhibit a system S that is stabilizing to a v-exclusive state predicate P, and show that the transformed system S<v> is not stabilizing to (P ∧ (dv = v ∨ G.v)). Consider a system S that has three binary variables, named "u", "v", and "out", and the following three actions:

$1: \quad u \neq v \rightarrow u := v$
$2: \quad v \neq u \rightarrow v := u$
$3: \quad u = v \rightarrow out := u$

It is straightforward to show that the state predicate P, defined as (u = v), is v-exclusive in S and that S is stabilizing to P.

Now consider the transformed system S<v>. This system has four binary variables, namely "u", "v", "out", and "dv", and the following four actions:

$1: \quad u \neq dv \rightarrow u := dv$
$2: \quad dv \neq u \rightarrow v := u$
$3: \quad u = dv \rightarrow out := u$
$4: \quad dv \neq v \rightarrow dv := v$

The predicate G.v, which is the negation of the disjunction of the guards of all actions other than the dv action and the v action in S<v>, is defined as follows.

$$\begin{aligned} G.v &= \neg\, (u \neq dv \ \vee \ u = dv) \\ &= \text{false} \end{aligned}$$

Thus, the state predicate (P ∧ (dv = v ∨ G.v)) is defined as (u = v ∧ dv = v).

To show that the transformed system S<v> is not stabilizing to (u = v ∧ dv = v), it is sufficient to exhibit an infinite computation of S<v> that does not have a state where (u = v ∧ dv = v) holds. Consider a state p of S<v> where (u = v ∧ u ≠ dv) holds. If system S<v> starts at state p and the four actions 1, then 3, then 4, then 2 are executed repeatedly in this order, then S<v> will never reach a state where (u = v ∧ dv = v) holds. Thus, S<v> is not stabilizing to (u = v ∧ dv = v).

Theorem 7.

If *P is a v-exclusive state predicate in a system S, and*
S is weakly stabilizing to P
then *S<v> is weakly stabilizing to (P ∧ (dv = v ∨ G.v)).*

Proof. By Theorem 5, the state predicate (P ∧ (dv = v ∨ G.v)) is closed in the transformed system S<v>. It remains to be shown that for every state p of S<v>, there is a computation of S<v> that starts at p and has a (P ∧ (dv = v ∨ G.v)) state. Let p be any state of system S<v>, and q be the corresponding state of system S. Thus, the value of each variable of system S at state q equals the value of the corresponding variable of system S<v> at state p. Because S is

weakly stabilizing to P, there is a computation x of S that starts at q and has a P state.

Now consider computation y of S<v> that is generated to track the generation of computation x as follows. First, computation y starts at state p. Second, the sequence of actions of system S<v> that is executed to generate the states in y is the same as the sequence of actions of system S that is executed to generate the states in x, with the following exception. If the values of variables dv and v are different in the last state generated in y, then the next executed action in y is the dv action. Then, the generation of computation y continues to track the generation of x. Because computation x eventually reaches a P state, computation y eventually reaches a (P $\wedge$ dv = v) state. This completes the proof that the transformed system S<v> is weakly stabilizing to (P $\wedge$ (dv = v $\vee$ G.v)).

5 Achieving Stabilization

In the last section, we established that adding delays to a system S preserves the weak stabilization of S, but does not necessarily preserve the stabilization of S. In this section, we establish that weak stabilization is a "good approximation" of stabilization. In particular, we show that under reasonable conditions, namely that the system has a finite number of states and its execution is strongly fair, weak stabilization of the system implies stabilization of the same system.

A computation of a system S is strongly fair iff for every transition (p, q) of system S, if state p occurs infinitely many times in the computation, then transition (p, q) occurs infinitely many times in the computation.

A system S is stabilizing to a state predicate P under strong fairness iff P is closed in S, and for every state p of S, every strongly fair computation of S, that starts at p, has a P state.

Theorem 8.

If *S is a system that has a finite number of states, and*
S is weakly stabilizing to a state predicate P,
then *S is stabilizing to P under strong fairness.*

Proof. Because S is weakly stabilizing to P, P is closed in S. It remains to be shown that for every state p of S, every strongly fair computation of S that starts at p has a P state. Let x be a strongly fair computation, of the form (p.0, p.1, ...), that starts at p (i. e. p = p.0). We need to show that computation x has a P state.

Because S has a finite number of states, at least one state q occurs infinitely many times in computation x. Because S is weakly stabilizing to P, there is a computation (q, q.1, q.2, ...) that starts at q and has a P state. Thus, some q.i in the computation (q, q.1, q.2, ...) is a P state. Because q occurs infinitely many times in the strongly fair computation x, both transition (q, q.1) and state q.1 occur infinitely many times in x. Similarly, because q.1 occurs infinitely many times in x, both transition (q.1, q.2) and state q.2 occur infinitely many times in x. This argument can be extended to show that a P state, namely state q.i,

occurs (infinitely many times) in computation x. This completes the proof that S is stabilizing to P under strong fairness.

6 Theorem of Weak Stabilization

The next three theorems state interesting properties of weak stabilization. Because similar properties hold for stabilization, these theorems serve as further evidence that weak stabilization is a "good approximation" of stabilization.

Theorem 9. *(Base Theorem)*
Each system is weakly stabilizing to the state predicate true.

Theorem 10. *(Theorem of Junction)*

If *S is weakly stabilizing to a state predicate P, and*
S is weakly stabilizing to a state predicate Q,
then *S is weakly stabilizing to (P ∨ Q), and*
S is weakly stabilizing to (P ∧ Q), provided that P ∧ Q ≠ false.

Theorem 11. *(Theorem of Weakening)*

If *S is weakly stabilizing to a state predicate P,*
Q is a closed state predicate in S, and
P ⇒ Q in S,
then *S is weakly stabilizing to Q.*

Proofs of these three theorems are rather straightforward. We present here the most interesting proof, namely the proof of the second part of Theorem 10.

Because both P and Q are closed in system S, then (P ∧ Q) is closed in S. It remains to show that for each state p of S, there is a computation that starts at p and has a (P ∧ Q) state. Because S is weakly stabilizing to P, there is a computation (p.0, p.1, ...) that starts at p (i.e. p = p.0) and has a P state p' (i.e. p' = p.i for some i). If p' is a Q state, then the required computation is (p.0, p.1, ...) itself. Otherwise, p' is not a Q state. But because S is also weakly stabilizing to Q, there is a computation (q.0, q.1, ...) that starts at p' (i.e. p' = q.0) and has a Q state q (i.e. q = q.j for some j). Because p' is a P state and P is closed in S, then every state in the computation (q.0, q.1, ...) is a P state. Thus, state q in the computation (q.0, q.1, ...) is a (P ∧ Q) state. In this case, the required computation is (p.0, ..., p.(i-1), p', q.1, ..., q.j, ...). Thus, S is weakly stabilizing to (P ∧ Q).

7 Composition of Weak Stabilization

In this section, we describe a method for composing several weakly stabilizing systems into a single weakly stabilizing system. Note that although there

are methods for composing several stabilizing systems into a single stabilizing system, the methods for composing weakly stabilizing systems seem richer than those for composing stabilizing systems. In particular, the method for composing weakly stabilizing systems described in this section cannot be used to compose stabilizing systems.

A state predicate P of a system S is called a fixed point in S iff every action "g → v := E" of S is such that g = false or v = E at every P state.

Let v and w be two variables of a system S. Variable v is an input of S iff v does not occur in the left-hand side of any assignment (in an action) in S. Variable w is an output of S iff w does not occur in the guard or in the right-hand side of any assignment (in an action) in S. We adopt the notation S[v, w] to denote a system S that has an input variable v and an output variable w.

Two systems S[v, w] and T[w, v] are compatible iff they have no common variables other than v and w.

Two compatible systems S[v, w] and T[w, v] can be composed into a single system, denoted S[v, w] ‖ T[w, v], as follows. First, the set of variables of the composed system is the union of the two sets of variables of systems S[v, w] and T[w, v]. Second, the set of actions of the composed system is the union of the two sets of actions of systems S[v, w] and T[w, v].

Theorem 12.

If *S[v, w] is weakly stabilizing to a fixed point P,*
T[w, v] is weakly stabilizing to a fixed point Q,
S[v, w] and T[w, v] are compatible,
there is one-to-one correspondence between the values of v and the values of w such that
P holds only at the corresponding value pairs, and
Q holds only at the corresponding value pairs,
then *S[v, w] ‖ T[w, v] is weakly stabilizing to (P ∧ Q).*

Proof. Because P is a fixed point in system S, each action "g → v := E" of S is such that g = false or v = E at every P state. Also, because Q is a fixed point in system T, each action "g → v := E" of T is such that g = false or v = E at every Q state. Thus, each action "g → v := E" of system S ‖ T is such that g = false or v = E at each (P ∧ Q) state, and the state predicate (P ∧ Q) is a fixed point in the system S ‖ T. It remains to show that for every state b of system S ‖ T, there is a computation of system S ‖ T that starts at state b and has a (P ∧ Q) state.

Let b be any state of system S ‖ T, and let b|S be the "projection" of state b on system S. Because system S is weakly stabilizing to P, there is a computation x of S that starts at state b|S and has a P state. Thus, there is a computation (b.0, b.1, ...) of system S ‖ T that starts at state b and has the same sequence of actions executed in computation x. Computation (b.0, b.1, ...) has a state c, i.e. b.i = c for some i, such that c|S is a P state that occurs in computation x. Because c|S is a P state, the values of the two variables v and w at state c constitute a corresponding value pair.

Because system T is weakly stabilizing to Q, there is a computation y of T that starts at state c|T and has a Q state. Thus, there is a computation (c.0, c.1, ...) of system S ‖ T that starts at state c and has the same sequence of actions executed in computation y. Computation (c.0, c.1, ...) has a state c.j such that state c.j|T is a Q state that occurs in computation y. Because c.j|T is a Q state, the values of the two variables v and w at state c.j constitute a corresponding value pair. Thus, the values of variables v and w at state c.j is the same as the values of their values at state c. Therefore, c.j is a (P ∧ Q) state. In other words, the computation (b.0, b.1, ..., b.i, c.1, ..., c.j, ...) starts at state b and has a (P ∧ Q) state. This completes our proof that S ‖ T is weakly stabilizing to (P ∧ Q).

8 Concluding Remarks

In this paper, we have introduced the property of weak stabilization and showed that this property is superior to the (strictly stronger) property of stabilization in many respects. First, we showed that the proof obligations for weak stabilization are less severe than those for stabilization. This suggests that verifying weak stabilization is easier than verifying stabilization. Second, we showed that adding delays to a system preserves the weak stabilization properties of that system but does not necessarily preserve the stabilization properties of the system. This suggests that weak stabilizing systems are easier to implement than stabilizing systems. Third, we showed that weakly stabilizing systems that have a finite numbers of states are in fact stabilizing under strong fairness, and showed that weak stabilization satisfies several interesting theorems that are also satisfied by stabilization. This suggests that weak stabilization is a "good approximation" of stabilization. Fourth, we described a method for combining weakly stabilizing systems and argued that this method cannot be used to compose stabilizing systems. This suggests that weak stabilizing systems are easier to compose than stabilizing systems.

References

1. Arora, A., Gouda, M.G.: Delay-Insensitive Stabilization. Proceedings of the Third Workshop on Self-Stabilizing Systems (1997) 95–109
2. Dijkstra, E.W.: Self-Stabilizing Systems in Spite of Distributed Control. Communications of the ACM, Vol. 17 (1974) 643–644
3. Dolev, S.: Self-Stabilization. 1st edn. MIT Press, Cambridge Massachusetts (2000)
4. Flatebo, M., Datta, A.K.: Self-Stabilization in Distributed Systems. In: Casavant, T.L., Singhal, M. (eds.): Readings in Distributed Computing Systems. Lecture Notes in Computer Science, Vol. 1281. Springer-Verlag, Berlin Heidelberg New York (1994) 100–114
5. Gouda, M.G.: The Triumph and Tribulation of System Stabilization. In: Helary, J.M., Raynal, M. (eds.): Proceedings of the International Workshop on Distributed Algorithms. Lecture Notes in Computer Science, Vol. 972. Springer-Verlag, Berlin Heidelberg (1995) 1–18
6. Herman, T.: A Comprehensive Bibliography on Self-Stabilization. `http://www.cs.uiowa.edu/ftp/selfstab/bibliography/2000` (2000)
7. Schneider, M.: Self-Stabilization. ACM Computing Surveys, Vol. 25 (1993) 45–67