2009 International Conference on Parallel and Distributed Computing, Applications and Technologies

Consistent Fixed Points and Negative Gain

H. B. Acharya
The University of Texas at Austin
acharya @ cs.utexas.edu

E. S. Elmallah
The University of Alberta
ehab @ cs.ualberta.ca

M. G. Gouda
The National Science Foundation

***Abstract*—We discuss the stabilization properties of networks that are composed of "displacement elements". Each displacement element is defined by an integer K, called the displacement of the element, an input variable x, and an output variable y, where the values of x and y are non-negative integers. An execution step of this element assigns to y the maximum of 0 and $K + x$. The objective of our discussion is to demonstrate that two principles play an important role in ensuring that a network N is stabilizing, i.e. starting from any global state, network N is guaranteed to reach a global fixed point. The first principle, named consistent fixed points, states that if a variable is written by two subnetworks of N, then the values of this variable, when these two subnetworks reach fixed points, are equal. The second principle, named negative gain, states that the sum of displacements along every directed loop in network N is negative.**

***Index Terms*—Stabilization, displacement networks, Loops, Negative gain, Consistent Fixed Points**

I. Introduction

In this paper, we refer to a network of communicating elements as stabilizing iff, starting from any global state, the network is guaranteed to reach a fixed point [4], [7], [5]. Although the theory of network stabilization is well established by now, very few general principles, which can be used to explain the stabilization of rich classes of networks, have been identified. To remedy this situation, we identify in this paper two general principles that can be used to design stabilizing networks. We also illustrate the utility of these two principles by showing that these principles can be used to design interesting classes of stabilizing displacement networks.

A displacement network consists of one or more displacement elements. Each element is defined by an integer K, called the displacement of the element, an input variable x, and an output variable y, where the values of x and y are non-negative integers. The execution step of this element assigns to y the maximum of 0 and $K + x$ (provided that this assignment changes the value of y). Note that there is an asymmetry between the elements that use a variable as an input and as an output; in general, an element cannot change the value of its own input variables. We will represent the variables by directed edges to show that information passes in one direction only. This means that even if element A can send information to B, B will not be able to send information to A unless there are loops. Our networks are thus unidirectional, and obey the theory developed by Tixeuil et al. in [11], [1], [2].

The two general principles that one needs to adopt while designing a stabilizing displacement network N are named consistent fixed points and negative gain. The principle of consistent fixed points states that, if a variable is written by two or more subnetworks of N, then the values of this variable written by these subnetworks, when these subnetworks reach fixed points, are equal. The principle of negative gain states that the sum of the displacements along every directed loop in network N is negative.

These two principles have counterparts in control theory. Specifically, the principle of consistent fixed points is analogous to the requirement that a control system be free from self-oscillations [10]. And the principle of negative gain is analogous to the requirement that the feedback loop of a control system be negative [3].

The remainder of this paper proceeds as follows: In Section II, we describe our model of displacement elements and networks. In Section III, we show that the principle of consistent fixed points is both necessary and sufficient for establishing the stabilization of acyclic networks. In Section IV, we prove that the principle of negative gain is both necessary and sufficient for establishing the stabilization of loop networks. Then in Section V, we show that the two principles together are necessary and sufficient to establish the stabilization of a class of "composite" networks, each of which consists of acyclic and loop subnetworks. Next, in Section VI, we show that the principle of negative gain is sufficient to establish the stabilization of classes of "bow" networks. Finally, we present our concluding remarks in Section VII.

II. Displacement Elements and Networks

A *displacement element*, or *element* for short, is defined by a triple

$$(K, x, y)$$

where

- K is an integer called the *displacement* of the element.
- x is a variable, whose value is a non-negative integer, called the *input variable* of the element.
- y is a variable, whose value is a non-negative integer, called the *output variable* of the element.

For convenience, Figure 1 shows a graphical representation of element (K, x, y). In this representation, the element is represented by a box, labeled K, that has one incoming edge, labeled x, and one outgoing edge, labeled y.

A *state* of element (K, x, y) is a pair (i, j) of non-negative integers, where i is the value of variable x at the state, and j is the value of variable y at the state. For example, $(2, 0)$ is

978-0-7695-3914-0/09 $26.00 © 2009 IEEE
DOI 10.1109/PDCAT.2009.85

Fig. 1. An element

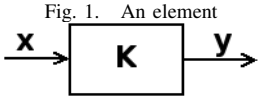
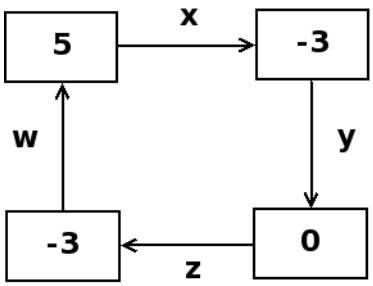


Fig. 2. A displacement network



a state of element $(-4, x, y)$, where 2 and 0 are the values of variables x and y, respectively, at state $(2, 0)$.

Note that if the two variables x and y for an element (K, x, y) are in fact the same variable, then the state of element (K, x, y) is a pair (i, i) of equal non-negative integers.

A state (i, j) of element (K, x, y) is called a *fixed point* of the element iff state (i, j) satisfies the predicate $(j = max(0, K + i))$. We refer to this predicate as the *fixed point predicate* of element (K, x, y).

For example, state $(2, 0)$ is a fixed point of element $(-4, x, y)$ since this state satisfies the predicate $(0 = max(0, -4+2))$. On the other hand, state $(2, 1)$ is not a fixed point of element $(-4, x, y)$ since this state does not satisfy the predicate $(1 = max(0, -4 + 2))$.

Let (i, j) be a state that is not a fixed point of element (K, x, y). It is straightforward to show that element (K, x, y) has exactly one fixed point of the form (i, j'). Element (K, x, y) is said to be *enabled for execution* at state (i, j), and its *execution* at state (i, j) yields the element in the fixed point (i, j').

A *displacement network*, or *network* for short, is a nonempty set of (displacement) elements.

As an example, Figure 2 shows a graphical representation of a network that consists of the four elements $(5, w, x)$, $(-3, x, y)$, $(0, y, z)$, and $(-3, z, w)$. Note that the elements in this network can be arranged on a directed loop where the output variable of each element is also the input variable of the "next" element on the loop.

A *global state* of a network is defined by a non-negative integer for every variable in every element in the network. For example, a global state of the network in Figure 2 can be defined by the predicate: $(w = 2 \wedge x = 0 \wedge y = 100 \wedge z = 2)$.

The global state of a network *includes* one state for each element in the network. For example, the global state $(w = 2 \wedge x = 0 \wedge y = 100 \wedge z = 2)$ for the network in Figure 2 includes the state $(2, 0)$ for element $(5, w, x)$ in the network.

A global state S of a network is called a *global fixed point* iff every state, included in S, of every element in the network is a fixed point of the element.

Let S be a global state of a network that has an element (K, x, y), and let S include the state (i, j) of element (K, x, y). Assume that (i, j) is not a fixed point of (K, x, y), whereas (i, j') is a fixed point of (K, x, y). In this case, element (K, x, y) is said to be *enabled for execution* at the global state S, and its *execution at* S yields the network in the global state S', where S' is the same as S except that the value of variable y is changed from j to j'. We refer to the pair (S, S') as a *transition* of the network caused by executing element (K, x, y).

A *computation* of a network is a sequence of global states of the network

$$S_1, S_2, \ldots$$

such that the following three conditions hold:

1) *Transition:*
   Every pair (S_i, S_{i+1}) of consecutive global states in the computation is a transition of the network caused by executing one element in the network.
2) *Maximality:*
   Either the computation is infinite, or it is finite and its last global state is a global fixed point of the network.
3) *Fairness:*
   If the computation has a global state S_i at which some element in the network is enabled for execution, then the computation has a subsequent global state $S_j, j > i$, at which the element is either executed or no longer enabled for execution.

A network is called *stabilizing* iff each computation of the network is finite.

III. Stabilization of Acyclic Networks

A network N is called *acyclic* iff N has no subset of elements that can be arranged on a directed loop where the output variable of each element in the subset is also the input variable of the next element on the loop. (Note that this definition of an acyclic network implies that an acyclic network can not have an element of the form (K, x, x).)

An input variable x of some element in an acyclic network N is called an *input variable of* N iff x is not the output variable of any element in N. Similarly, an output variable y of some element in an acyclic network N is called an *output variable of* N iff y is not the input variable of any element in N.

Let N be an acyclic network. A *chain* in N is a nonempty sequence, of elements in N, of the form:

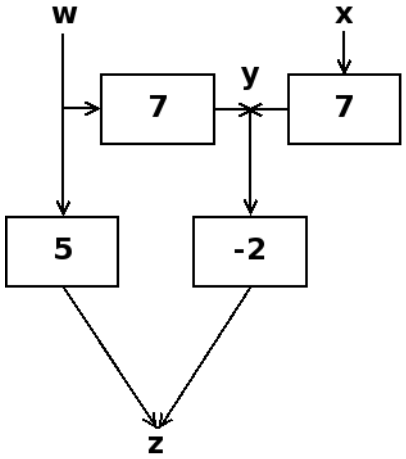
$$(K_1, x_1, x_2), (K_2, x_2, x_3), .., (K_r, x_r, x_{r+1})$$

where x_1 is an input variable of N. The *gain* of such a chain is the sum $(K_1 + K_2 + .. + K_r)$ of the displacements of the elements in the chain. This chain is said to *terminate* at variable x_{r+1}.

In an acyclic network N, two chains

$$(K_1, x_1, x_2), (K_2, x_2, x_3), .., (K_r, x_r, x_{r+1})$$
$$(L_1, y_1, y_2), (L_2, y_2, y_3), .., (L_s, y_s, y_{s+1})$$

Fig. 3. Acyclic Network



are called *consistent* iff the following two conditions hold:

- Variables x_1 and y_1 are the same input variable of N.
- The gains of the two chains are equal.

Theorem 1. *An acyclic network N is stabilizing iff every two chains, that terminate at the same variable in N, are consistent.*

Proof: A sequence composed of networks that are stabilizing is itself stabilizing. This is proved by mathematical induction. (It is trivially true for a sequence consisting of one network. In a chain of n networks, of which the first is N_1, a fair infinite computation ensures that each element in N_1 that is enabled for execution will eventually execute; as N_1 is stabilizing, it will reach a fixed point. From this time on, we remove N_1 and deal with the remaining chain which stabilizes as its input, the output of N_1, is stable. By inductive hypothesis, it stabilizes.)

Now we consider the case of networks that share an output variable. Each network is stabilizing and, by our two conditions, we ensure that they both converge to the same value for the output variable. Hence, after a finitely long computation, the final elements in the networks (the elements which actually output the shared output variable) cannot set it to different values and perpetually re-enable other final elements. The united network converges.

As an acyclic network is made up of individually stabilizing elements by these two operations - sequential composition and fan-in - any acyclic network satisfying both our conditions will converge.

■

As an example of this theorem, consider the acyclic network, shown in Figure 3, that consists of the following four elements:

$$N = \{(5, w, z), (7, w, y), (-2, y, z), (7, x, y)\}$$

Now, consider the following two chains in the network

$$\text{First chain} : (7, w, y)$$
$$\text{Second chain} : (7, x, y)$$

These two chains start with two distinct input variables, w and x, of the network. Thus, these two chains do not satisfy Condition 1 in Theorem 1, and so the network is not stabilizing.

The following theorem states a sufficient condition for combining two stabilizing acyclic networks into one stabilizing acyclic network.

Theorem 2. *Let N be a stabilizing network, and N' be a stabilizing acyclic network. If every variable that occurs in both N and N' is an input variable of N', then the composite network $(N \cup N')$ is stabilizing.*

Proof: (By contradiction) We show that, if we assume there is an infinitely long computation for $N \cup N'$, we can derive a logical fallacy.

In an infinitely long computation of $N \cup N'$, no element in N can remain enabled to execute and still not be chosen to execute (by our third condition). As N is given to be stabilizing, there cannot be an infinite sequence of executions of its elements; N will stabilize after a finite number of steps. Note that after this point, no elements in N can become enabled for execution: it is in a fixed point, and the execution of elements in N' does not affect the relationships between the input and output variables of elements in N.

After N reaches a fixed point, the input elements of N', including those that were variables in N, are constant. As N' is stabilizing, it will also stabilize after a finite number of steps. $N \cup N'$ stabilizes. This is incompatible with the assumption that there is an infinitely long computation for the combined network. Hence it is not possible for $N \cup N'$ to have an infinitely long computation, and it is proved to be stabilizing.

■

IV. Stabilization of Loop Networks

A network N, that has n elements, is called an *n-loop*, or *loop*, iff each variable in N occurs in at most two elements in N and the elements in N can be arranged on a directed loop where the output variable of each element is also the input variable of the next element on the loop.

An n-loop can be written as follows:

$$\{(K_1, x_1, x_2), (K_2, x_2, x_3), .., (K_n, x_n, x_1)\}$$

where the variables $x_1, x_2, .., x_n$ are distinct. The *gain* of this n-loop is the sum $(K_1 + K_2 + .. + K_n)$ of the displacements of the elements in the loop.

Figure 2 shows an example of a 4-loop network. The gain of this 4-loop is -1.

The following two theorems give necessary and sufficient conditions for 1-loop and 2-loop networks to be stabilizing.

Theorem 3. *A 1-loop is stabilizing iff its gain is at most 0.*

Proof: A 1-loop must have an element (K, x_1, x_1). If $K > 0$, the element has no fixed point where $x_1 = max(0, K + x_1)$; as there is no fixed point for the loop, it cannot stabilize. If $K < 0$, the system reaches the fixed point $(x_1 = 0)$. Any non negative value of x_1 is a fixed point if $K = 0$.

■

Theorem 4. *A 2-loop is stabilizing iff its gain is at most 0.*

Proof: Consider any computation of a 2-loop. The first element to execute, k, has displacement K; the other element, l, has displacement L. The value of the input variable of k immediately prior to its execution is x.

If the computation is infinitely long, the value at the input of k (also the output of l) forms the series $x, x + K + L, x + 2(K + L)$.. while the value of the output of k is $x + K, x + K + (K + L), x + K + 2(K + L)$..

If $K + L < 0$, we have two infinite decreasing sequences of whole numbers, which is impossible; hence the system converges.

If $K + L = 0$, the values are constant. Hence after the first execution, the system is in a fixed point.

If $K + L > 0$, we have an infinite execution hence the system is not stabilizing.

■

The next three theorems give necessary and sufficient conditions for an n-loop, where $n \geq 3$, to be stabilizing.

Theorem 5. *Let N be an n-loop where $n \geq 3$. If the gain of N is greater than 0, then N is not stabilizing.*

Proof: There exists no assignment of values to the variables that is a fixed point for a loop with positive gain.

■

Theorem 6. *Let N be an n-loop where $n \geq 3$. If the gain of N equals 0, then N is not stabilizing.*

Proof: Consider a 3-loop in which the elements are named k_1, k_2 and k_3 in the order in which they appear following the loop. Let the values at the inputs of k_1, k_2, k_3 be x_1, x_2, x_3. Now there is at least one infinitely long computation, which consists of the execution sequence k_1, k_3, k_2 repeated ad infinitum.

In the general case, consider that the value in any variable can, if it propagates once completely around the network without interference from other variables, establish a steady state. However, the different values do interfere with one another: in the above example, value x_1 tries to make variable x_2 converge to $x_1 + K_1$, while value x_3 tries to make it converge to $x_3 + K_3 + K_1$. As the gain of the loop is not negative, we cannot prove that a value will count down to zero and stop. Multiple values can travel round and round the network ad infinitum, provided we never allow a value to overtake another (in which case, the overtaken value is lost).

■

Theorem 7. *Let N be an n-loop where $n \geq 3$. If the gain of N is less than 0, then N is stabilizing.*

V. Stabilization of Composite Networks

In the previous two sections, we presented necessary and sufficient conditions for acyclic and loop networks to be stabilizing. In this section, we present sufficient conditions for combining stabilizing acyclic and loop networks into stabilizing composite networks. But first we need to state, in Theorems 8 and 9 below, sufficient conditions for ensuring that a stabilizing acyclic or loop network has a single global fixed point.

Theorem 8. *Let N be a stabilizing acyclic network. If the value of every input variable of N is specified, then the value of every non-input variable in N, when N is in a global fixed point, can be computed uniquely.*

Proof: (By contradiction) Suppose there exist two global fixed points of N for the same values of the input variables. For the fixed points to be distinct, at least one variable x must have different values in the two fixed points. Say x has the values x_1 and x_2. x cannot be an input variable: these are given to have the same values in both fixed points. We consider the element of which x is the output. This element has displacement k in both fixed points. Let the input variable of the element be x'. Now suppose x' has the same value x'_1 in both cases. If $x'_1 + K < 0$ then x_1, x_2 are both 0; otherwise, x_1, x_2 are both $x'_1 + K$. But $x_1 \neq x_2$. Hence x' has different values in the global fixed point - say x'_1, corresponding to x_1, and x'_2, corresponding to x_2. Applying this argument recursively, we eventually reach w, an input variable of N, which is the input to a chain whose output is x. w must also have different values in the two fixed points. This contradicts the given assertion that the input variables have the same values in both fixed points. Hence there can exist exactly one global fixed point for a given assignment of values to all input variables.

■

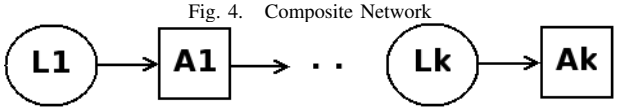
Theorem 9. *Let N be a stabilizing loop network. If the gain of N is negative, then the value of every variable in N, when N is in a global fixed point, can be computed uniquely.*

Proof: First we note that a fixed point of a stabilizing loop network has at least one variable set to 0. If this is not the case, let the displacements of the elements be $K_1, ..K_n$. Now we start from x_1, the input to element k_1. As no element is enabled, the output of k_1 is $x_1 + K_1$ etc. Proceeding in this fashion we get $x_1 = x_1 + \sum_{i=1}^{n} K_i$ which is impossible as the sum of displacements is negative, not zero.

Next, we consider the possibility that a loop network may have two fixed points but disjoint sets of variables are set to 0. Suppose in fixed point F_1, variable x is set to 0 and in F_2 it is not. In F_2 the next zero (proceeding along the ring) is y. But from F_2, the gain of the elements from x to y is negative; in F_1, as x is 0, y must be 0 also. Hence there is a shared zero element in both fixed points.

The remainder of the proof is straightforward: proceeding

Fig. 4. Composite Network



along the loop from a shared zero element, all the variables have the same value in both fixed points, as the displacements and the starting value are identical.

■

Next we describe how to combine k stabilizing acyclic networks and k stabilizing loop networks into one composite network.

A *composite network* C is defined as the union of $2k$ networks:

$$C = A_1 \cup .. \cup A_k \cup L_1 \cup .. \cup L_k$$

where the following five conditions hold:

1) Each A_i is a stabilizing acyclic network.
2) Each L_i is a (stabilizing) loop network with negative gain.
3) No variable occurs in two or more (acyclic or loop) networks except as specified by Conditions 4 and 5 below.
4) Each input variable in each A_i is also a variable in L_i.
5) Each non-input variable in each A_i can also be a variable in L_{i+1}.

A graphical representation of the composite network C is shown in Figure 4.

From Condition 2, each loop L_i in the composite network C has a negative gain. Thus, from Theorem 9, the value of each variable in L_i, when L_i is in a global fixed point, can be computed uniquely. Then, from Condition 4, the value of each input variable in each A_i can be computed uniquely. Then, from Theorem 8, the value of every non-input variable in A_i, when A_i is in a global fixed point, can be computed uniquely.

The composite network C is called *consistent* iff the following condition holds for every variable x that is both a non-input variable in A_i and a variable in L_{i+1}: the value of x when A_i is in a global fixed point equals the value of x when L_{i+1} is in a global fixed point.

Theorem 10. *A composite network is stabilizing iff it is consistent.*

VI. Stabilization of Bow Networks

A *bow network* N consists of $m+1$ elements as follows:

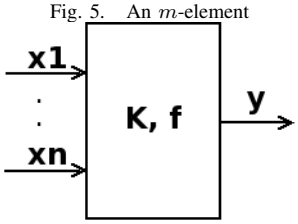
$$N = \{(K, x, y), (L_1, y, x), .., (L_m, y, x)\}$$

Note that this bow network has m directed loops, whose gains are $K + L_1$, $K + L_2$, .., and $K + L_m$.

Theorem 11. *A bow network is stabilizing iff the gain of each directed loop in the network is at most* 0.

In the remainder of this section, we extend the definition of an element from one that has single input variable to one that has m input variables. We refer to this extended element as an m-element, and we use it to define an extended bow network called an m-bow network. Finally, we present a theorem stating that, if the gain of every directed loop in an m-bow network is negative, then the network is stabilizing.

Fig. 5. An m-element



A function that takes as input a vector of one or more non-negative integers, and produces as output any one of the input integers, is called a *selector*. Examples of selectors are the functions *min*, *max*, and *median*. (Note that *mean* is not a selector.)

Let m be a positive integer. An m-element is defined as a tuple

$$(K, f, x_1, .., x_m, y)$$

where

- K is an integer called the *displacement* of the m-element.
- f is a *selector*.
- $x_1, .., x_m$ are variables, whose values are non-negative integers, called the *input variables* of the m-element.
- y is a variable, whose value is a non-negative integer, called the *output variable* of the m-element.

Note that in the special case where $m = 1$, the selector of an m-element degenerates into the identity function, and the m-element becomes a (displacement) element.

Figure 5 shows the m-element $(K, f, x_1, .., x_m, y)$. In this graphical representation, the m-element is represented by a box, labeled K, f, that has m incoming edges, labeled $x_1, .., x_m$, and one outgoing edge, labeled y.

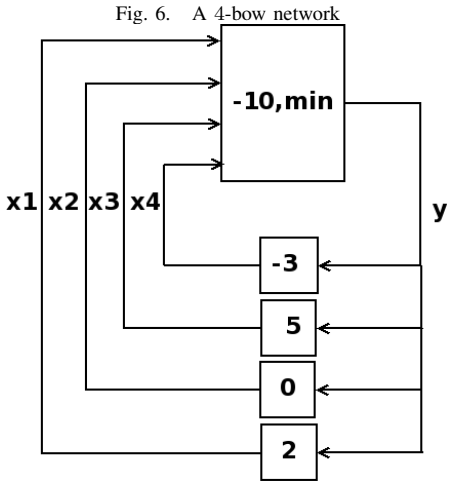
A *state* of an m-element $(K, f, x_1, .., x_m, y)$ is an $(m+1)$-tuple $(i_1, .., i_m, j)$ of non-negative integers, where each i_k is the value of the input variable x_k at the state, and j is the value of the output variable y at the state. For example, $(2, 0, 5, 0, 3)$ is a state of the 4-element $(-10, min, x_1, x_2, x_3, x_4, y)$.

A state $(i_1, .., i_m, j)$ of an m-element $(K, f, x_1, .., x_m, y)$ is called a *fixed point* of the m-element iff the state satisfies the predicate $(j = max(0, K + f(i_1, i_2, ..i_m)))$. We refer to this predicate as the *fixed point predicate* of the m- element.

For example, consider the 4-element $(-10, min, x_1, x_2, x_3, x_4, y)$ and the state $(2, 0, 5, 0, 0)$. This state satisfies the predicate $(0 = max(0, -10 + min(2, 0, 5, 0)))$, hence it is a fixed point of the 4-element.

Let $(i_1, ..i_m, j)$ be a state that is not a fixed point of the m-element $(K, f, x_1, .., x_m, y)$. It is straightforward to show that this m-element has exactly one fixed point, of the form $(i_1, .., i_m, j')$. The m-element is said to be *enabled for execution* at state $(i_1, .., i_m, j)$, and its *execution* at this state yields the element in the fixed point $(i_1, .., i_m, j')$.

Fig. 6. A 4-bow network



An m-*bow network* N is a set, containing one m-element and m displacement elements, of the form:

$$N = \{(K, f, x_1, .., x_m, y), \\ (L_1, y, x_1), \\ .. \\ (L_m, y, x_m)\}$$

Note that, in an m-bow network, the m-element $(K, ..)$ forms a *directed loop* with each element (L_k, y, x_k). The *gain* of each loop is the sum $(K + L_k)$ of the displacements of the two elements on the loop.

As an example, Figure 6 shows a graphical representation of a 4-bow network that consists of one 4-element $(-10, min, x_1, x_2, x_3, x_4, y)$ and four elements $(2, y, x_1)$, $(0, y, x_2)$, $(5, y, x_3)$, and $(-3, y, x_4)$. The gain of each directed loop in this network is negative.

A *global state* of the m-bow network is defined by a non-negative integer for each variable in the network. Thus, the global state of the m-bow network includes one state for each element in the network. For example, the global state $(x_1 = 2 \wedge x_2 = 0 \wedge x_3 = 5 \wedge x_4 = 0 \wedge y = 0)$ for the 4-bow network in Figure 6 includes:

- the state $(0, 2)$ for element $(2, y, x_1)$,
- the state $(0, 0)$ for element $(0, y, x_2)$,
- the state $(0, 5)$ for element $(5, y, x_3)$,
- the state $(0, 0)$ for element $(-3, y, x_4)$,
- the state $(2, 0, 5, 0, 0)$ for the 4-element $(-10, min, x_1, x_2, x_3, x_4, y)$.

The concepts of *global fixed point*, *transition*, and *computation* for an $m-$bow network are similar to their counterparts for a displacement network, as defined in Section II.

An m-bow network is called *stabilizing* iff each computation of the network is finite.

Theorem 12. *If the gain of each directed loop in an m-bow network N is negative, then N is stabilizing.*

The following theorem states a sufficient condition for combining a stabilizing m-bow network and a stabilizing acyclic network into one stabilizing network.

Theorem 13. *Let N be a stabilizing m-bow network, and N' be a stabilizing acyclic network. If every variable that occurs in both N and N' is an input variable of N', then the composite network $(N \cup N')$ is stabilizing.*

Proof: The proof is similar to that of Theorem 10. ■

VII. Concluding Remarks

We have shown in this paper that the two principles, of consistent fixed points and negative gain, are sufficient (and sometimes both necessary and sufficient) to establish stabilization of many classes of displacement networks.

Unfortunately, our model of displacement networks is so far rather limited. In this model, each element is defined by a triple (K, x, y), where K is an integer. The execution of this element can be represented by the action

$$\textbf{if } y \neq max(0, K + x) \textbf{ then } y := max(0, K + x)$$

It would be interesting to extend our model to allow each element to be defined as a triple (F, x, y) where F is a linear function, and the execution of the element can be represented by the action

$$\textbf{if } y \neq max(0, \lceil F(x) \rceil) \textbf{ then } y := max(0, \lceil F(x) \rceil)$$

The questions of how to state our two principles in this extended model, and whether the two principles are sufficient in this case to guarantee network stabilization, remain open. Considering that the stabilization of linear systems is an active area of research [6], [9], [8] we believe that this is a promising direction for further study.

References

[1] S. T. Bertrand Ducourthial. Self-stabilization with r-operators. *Distributed Computing*, 14(3):147–162, 2001.

[2] S. T. Bertrand Ducourthial. Self-stabilization with path algebra. *Theoretical Computer Science*, 293(1):219–236, 2003.

[3] R. W. Brockett and D. Liberzon. Quantized feedback stabilization of linear systems. *IEEE Trans. Automat. Control*, 45:1279–1289, 2000.

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

[5] S. Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.

[6] N. Elia and S. Mitter. Stabilization of linear systems with limited information. *Automatic Control, IEEE Transactions on*, 46(9):1384–1400, Sep 2001.

[7] M. G. Gouda. The triumph and tribulation of system stabilization. In *Proceedings of the 9th International Workshop on Distributed Algorithms*, pages 1–18, London, UK, 1995. Springer-Verlag.

[8] E. N. Hoch, D. Bickson, and D. Dolev. Self-stabilizing numerical iterative computation. In *Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 79–93, 2008.

[9] C. M. Özveren, A. S. Willsky, and P. J. Antsaklis. Stability and stabilizability of discrete event dynamic systems. *Journal of the ACM*, 38(3):729–751, 1991.

[10] A. Pourhiet, M. Corrège, and D. Caruana. Control of self-oscillating systems. *IEE Proceedings - Control Theory and Applications*, 150(6):599–610, 2003.

[11] S. T. Sajal Das, Ajoy Kumar Datta. Self-stabilizing algorithms in dag structured networks. *Parallel Processing Letters*, 9(4):563–574, 1999.