Supporting Increment and Decrement Operations in Balancing Networks*

William Aiello[1], Costas Busch[2], Maurice Herlihy[2], Marios Mavronicolas[3], Nir Shavit[4], and Dan Touitou[5]

[1] AT&T Labs, 180 Park Avenue, Florham Park, NJ 07932, USA.
aiello@research.att.com

[2] Department of Computer Science, Brown University, Providence, RI 02912, USA.
{cb, mph}@cs.brown.edu

[3] Department of Computer Science, University of Cyprus, Nicosia CY-1678, Cyprus, and Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, USA.§
mavronic@turing.cs.ucy.ac.cy

[4] Department of Computer Science, School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel.
shanir@math.tau.ac.il

[5] IDC Herzliya, Tel-Aviv, Israel.
danidin@math.tau.ac.il

Abstract. *Counting networks* are a class of distributed data structures that support highly concurrent implementations of shared *Fetch&Increment* counters. Applications of these counters include shared pools and stacks, load balancing, and software barriers [4, 12, 13, 18]. A limitation of counting networks is that the resulting shared counters can be incremented, but not decremented.

A recent result by Shavit and Touitou [18] showed that the subclass of tree-shaped counting networks can support, in addition, decrement operations. This paper generalizes their result, showing that *any* counting network can be extended to support atomic decrements in a simple and natural way. Moreover, it is shown that decrement operations can be supported in networks that provide weaker properties, such as K*-smoothing*. In general, we identify a broad class of properties, which we call *boundedness properties,* that are preserved by the introduction of decrements: if a balancing network satisfies a particular boundedness property for increments alone, then it continues to satisfy that property for both increments and decrements.

Our proofs are purely combinatorial and rely on the novel concept of a *fooling pair* of input vectors.

* This paper combines, unifies, and extends results appearing in preliminary form in [2] and [6].

§ Partially supported by funds for the promotion of research at University of Cyprus. Part of the work of this author was performed while at AT&T Labs – Research, Florham Park, NJ, as a visitor to the Special Year on Networks, DIMACS Center for Discrete Mathematics and Theoretical Computer Science, Piscataway, NJ.

1 Introduction

Counting networks were originally introduced by Aspnes, Herlihy, and Shavit [4] and subsequently extended [1, 10, 11]. They support highly concurrent implementations of shared *Fetch&Increment* counters, shared pools and stacks, load balancing modules, and software barriers [4, 12, 13, 18].

Counting networks are constructed from basic elements called *balancers*. A balancer can be thought of as a routing switch for elements called *tokens*. It has a collection of input wires and a collection of output wires, respectively called the balancer's *fan-in* and *fan-out*. Tokens arrive asynchronously on arbitrary input wires, and are routed to successive output wires in a "round-robin" fashion. If one thinks of a balancer as having a state 'toggle" variable tracking which output wire the next token should exit on, then a token traversal amounts to a *Fetch&Toggle* operation, retrieving the value of the output wire and changing the toggle state to point to the next wire. The distribution of tokens on the output wires of a balancer thus satisfies the *step property* [4]: if y_i tokens exit on output wire i, then $0 \leq y_i - y_j \leq 1$ for any $j > i$.

A *balancing network* is a network of balancers, constructed by connecting balancers' output wires with other balancers' input wires in an acyclic fashion, in a way similar to the way *comparator networks* are constructed from *comparators* [9, Chapter 28]. The network itself has a number of input and output wires. A token enters the network on an input wire, traverses a sequence of balancers, and exits on an output wire. A balancing network is a *K-smoothing network* [1, 4] if, when all tokens have exited the network, the difference between the maximum and minimum number of tokens that exit on any output wire is bounded by K, regardless of the distribution of input tokens. Smoothing networks can be used for distributed load balancing.

A 1-smoothing network is a *counting network* if it satisfies the same step property as a balancer: when all tokens have traversed the network, if y_i tokens exit on output wire i, then $0 \leq y_i - y_j \leq 1$ for any $j > i$. Counting networks can be used to implement *Fetch&Increment* counters: the l-th token to exit on the j-th output wire returns the value $j + (l-1)w_{\text{out}}$, where w_{out} is the network's fan-out.

A limitation of counting networks is that they support increments but not decrements. Many synchronization algorithms and tools require the ability to decrement shared objects.

Shavit and Touitou [18] devised the first counting network algorithm to support decrements for the class of networks that have the layout of a binary tree. They did so by introducing a new type of token for the decrement operation, which they named the *antitoken*.[1] Unlike a token, which traverses a balancer by fetching the toggle value and then advancing it, an antitoken sets the toggle back and then fetches it. Informally, an antitoken "cancels" the effect of the most recent token on the balancer's toggle state, and vice versa. They provide

[1] The name was actually suggested by Yehuda Afek (personal communication).

an operational proof that *counting trees* [19] count correctly when traversed by tokens and antitokens.

Shavit and Touitou [18] also introduced the notion of *elimination.* One can use a balancing network to implement a *pool*, a kind of concurrent stack. If a token representing an enqueue operation meets a token representing a dequeue operation in the network, then they can "cancel" one another immediately, without traversing the rest of the network.

It is natural to ask whether the same properties hold for *arbitrary* counting networks. More generally, what properties of balancing networks are preserved by the introduction of antitokens? In this paper, we give the first general answer to this question. We show the following results.

- If a balancing network is a counting network for tokens, then it is also a counting network for both tokens and antitokens. As a result *any* counting network can be extended to support a *Fetch&Decrement* operation.
- Any counting network, not just elimination trees, permits tokens and antitokens to eliminate one another.
- If a balancing network is a K-smoothing network when inputs are tokens, then it remains a K-smoothing network when inputs include both tokens and antitokens.
- We identify a broad class of properties, which we call *boundedness properties,* that are preserved by the introduction of antitokens: if a balancing network satisfies a particular boundedness property when inputs are tokens, then it continues to satisfy that property when inputs include both tokens and antitokens. The step property and the K-smoothing property are examples of boundedness properties.

Unlike earlier work [18], our proofs are combinatorial, not operational. They rely on the novel concept of a *fooling pair* of input vectors, which, we believe, is of independent interest.

We assign the value 1 to each token and -1 to each antitoken. We treat a balancer as an operator carrying an integer *input vector* to an integer *output vector*. The i-th entry in the input vector represents the *algebraic sum* of the tokens and antitokens received on the i-th input wire, and similarly for the output vector. For example, if this value is zero, then the same number of tokens and antitokens have arrived on that wire. We treat a balancing network in the same way, as an "operator" on integer vectors.

A *boundedness property* is a set of possible output vectors satisfying

- it is a subset of the K-smoothing property, for some $K \geq 1$, and
- it is closed under the addition of any *constant* vector.

Both the K-smoothing and the step property are examples of boundedness properties. Our principal result is that any balancing network that satisfies a boundedness property for non-negative integer input vectors also satisfies that property for arbitrary integer input vectors.

The *state* of a balancer is the "position" of its toggle. Two input vectors form a *fooling pair* to a balancer if, starting in the same state, each "drives" the balancer to the same state. Similarly, a balancing network state is given by its balancers' states. Two input vectors form a *fooling pair* for that network if, starting from the same state, each drives the network to the same state. For a specific initial state of a balancing network, its fooling pairs define equivalence classes of input vectors.

Roughly speaking, we prove our main equivalence result as follows. Consider any balancing network with some boundedness property; take any arbitrary integer input vector and the corresponding integer output vector. By adding to the input vector an appropriate vector that belongs to the equivalence class for some given initial state, we obtain a new input vector such that all of its entries are non-negative integers. We show that the output vector corresponding to the new input vector is, in fact, equal to the original output vector plus a constant vector. Hence, our main equivalence result follows from closure of the boundedness property under addition with a constant vector.

2 Framework

For any integer $g \geq 2$, $\mathbf{x}^{(g)}$ denotes the vector $\langle x_0, x_1, \ldots, x_{g-1}\rangle^{\mathrm{T}}$, while $\lceil \mathbf{x}^{(g)} \rceil$ denotes the integer vector $\langle \lceil x_0 \rceil, \lceil x_1 \rceil, \ldots, \lceil x_{g-1} \rceil \rangle^{\mathrm{T}}$. For any vector $\mathbf{x}^{(g)}$, denote $||\mathbf{x}||_1 = \sum_{i=0}^{g-1} x_i$. We use $\mathbf{0}^{(g)}$ to denote $\langle 0, 0, \ldots, 0\rangle^{\mathrm{T}}$, a vector with g zero entries; similarly, we use $\mathbf{1}^{(g)}$ to denote $\langle 1, 1, \ldots, 1\rangle^{\mathrm{T}}$, a vector with g unit entries. We use $\mathbf{r}^{(g)}$ to denote the *ramp vector* $\langle 0, 1, \ldots, g-1\rangle^{\mathrm{T}}$. A *constant vector* is any vector of the form $c\,\mathbf{1}^{(g)}$, for any constant c.

Balancing networks are constructed from acyclically wired elements, called balancers, that route *tokens* and *antitokens* through the network, and *wires*. For generality, balancers may have arbitrary fan-in and fan-out, and they handle both tokens and antitokens.

For any pair of positive integers f_{in} and f_{out}, an $(f_{\mathrm{in}}, f_{\mathrm{out}})$-*balancer,* or *balancer* for short, is a routing element receiving tokens and antitokens on f_{in} input wires, numbered $0, 1, \ldots, f_{\mathrm{in}} - 1$, and sending out tokens and antitokens to f_{out} output wires, numbered $0, 1, \ldots, f_{\mathrm{out}} - 1$; f_{in} and f_{out} are called the balancer's *fan-in* and *fan-out,* respectively. Tokens and antitokens arrive on the balancer's input wires at arbitrary times, and they are output on its output wires. Roughly speaking, a balancer acts like a "generalized" *toggle,* which, on a stream of input tokens and antitokens, alternately forwards them to its output wires, going either down or up on each input token and antitoken, respectively. For clarity, we assume that all tokens and antitokens are distinct. Figure 1 depicts a balancer with three input wires and five output wires, stretched horizontally; the balancer is stretched vertically. In the left part, tokens and antitokens are denoted with full and empty circles, respectively; the numbering reflects the real-time order of tokens and antitokens in an execution where they traverse the balancer one by one (called a *sequential* execution).

For each input index i, $0 \leq i \leq f_{\mathrm{in}} - 1$, we denote by x_i the *balancer input state variable* that stands for the algebraic sum of the numbers of tokens and antitokens that have entered on input wire i; that is, x_i is the number of tokens

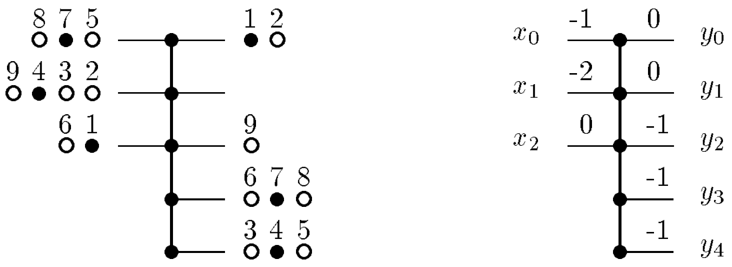

Fig. 1. A balancer

that have entered on input wire i *minus* the number of antitokens that have entered on input wire i. Denote $\mathbf{x}^{(f_{\text{in}})} = \langle x_0, x_1, \ldots, x_{f_{\text{in}}-1}\rangle^{\text{T}}$; call $\mathbf{x}^{(w_{\text{in}})}$ an *input vector.* For each output index j, $0 \leq j \leq f_{\text{out}} - 1$, we denote by y_j the *balancer output state variable* that stands for the algebraic sum of the numbers of tokens and antitokens that have exited on output wire j; that is, y_j is the number of tokens that have exited on output wire j *minus* the number of antitokens that have exited on output wire j. The right part of Fig. 1 shows the corresponding input and output state variables. Denote $\mathbf{y}^{(f_{\text{out}})} = \langle y_0, y_1, \ldots, y_{f_{\text{out}}-1}\rangle^{\text{T}}$; call $\mathbf{y}^{(f_{\text{out}})}$ an *output vector.*

The *configuration* of a balancer at any given time is the tuple $\langle \mathbf{x}^{(f_{\text{in}})}, \mathbf{y}^{(f_{\text{out}})}\rangle$; roughly speaking, the configuration is the collection of its input and output state variables. In the *initial configuration,* all input and output wires are empty; that is, in the initial configuration, $\mathbf{x}^{(f_{\text{in}})} = \mathbf{0}^{(f_{\text{in}})}$, and $\mathbf{y}^{(f_{\text{out}})} = \mathbf{0}^{(f_{\text{out}})}$. A configuration of a balancer is *quiescent* if there are no tokens or antitokens in the balancer. Note that the initial configuration is a quiescent one. The following formal properties are required for an $(f_{\text{in}}, f_{\text{out}})$-balancer.

1. *Safety property:* in any configuration, a balancer never creates either tokens or antitokens spontaneously.
2. *Liveness property:* for any finite number t of tokens and a of antitokens that enter the balancer, the balancer reaches within a finite amount of time a quiescent configuration where $t - e$ tokens and $a - e$ antitokens have exited the network, where e, $0 \leq e \leq \min\{t, a\}$, is the number of tokens and antitokens that are "eliminated" in the balancer.
3. *Step property:* in any quiescent configuration, for any pair of output indices j and k such that $0 \leq j < k \leq f_{\text{out}} - 1$, $0 \leq y_j - y_k \leq 1$.

From the safety and liveness properties it follows, for any quiescent configuration $\langle \mathbf{x}^{(f_{\text{in}})}, \mathbf{y}^{(f_{\text{out}})}\rangle$ of a balancer, that $\|\mathbf{x}^{(f_{\text{in}})}\|_1 = \|\mathbf{y}^{(f_{\text{out}})}\|_1$; that is, in a quiescent configuration, the algebraic sum of tokens and antitokens that exited the balancer is equal to the algebraic sum of tokens and antitokens that entered it. The equality of sums holds also for the case where some the tokens and antitokens are "eliminated" in the balancer.

For any input vector $\mathbf{x}^{(f_{\text{in}})}$, denote $\mathbf{y}^{(f_{\text{out}})} = b(\mathbf{x}^{(f_{\text{in}})})$ the output vector in the quiescent configuration that b will reach after all $||\mathbf{x}^{(f_{\text{in}})}||_1$ tokens and antitokens that entered b have exited; write also $b : \mathbf{x}^{(f_{\text{in}})} \rightarrow \mathbf{y}^{(f_{\text{out}})}$ to denote the balancer b. The output vector can also be written [1,4,8] as

$$\mathbf{y}^{(f_{\text{out}})} = \left\lceil \frac{||\mathbf{x}^{(f_{\text{in}})}||_1 \, \mathbf{1}^{(f_{\text{out}})} - \mathbf{r}^{(f_{\text{out}})}}{f_{\text{out}}} \right\rceil .$$

For any quiescent configuration $\langle \mathbf{x}^{(f_{\text{in}})}, \mathbf{y}^{(f_{\text{out}})} \rangle$ of a balancer $b : \mathbf{x}^{(f_{\text{in}})} \rightarrow \mathbf{y}^{(f_{\text{out}})}$, the *state* of the balancer b, denoted $\text{state}_b(\langle \mathbf{x}^{(f_{\text{in}})}, \mathbf{y}^{(f_{\text{out}})} \rangle)$, is defined to be

$$\text{state}_b(\langle \mathbf{x}^{(f_{\text{in}})}, \mathbf{y}^{(f_{\text{out}})} \rangle) = ||\mathbf{x}^{(f_{\text{in}})}||_1 \bmod f_{\text{out}} ;$$

since the configuration is quiescent, it follows that

$$\text{state}_b(\langle \mathbf{x}^{(f_{\text{in}})}, \mathbf{y}^{(f_{\text{out}})} \rangle) = ||\mathbf{y}^{(f_{\text{out}})}||_1 \bmod f_{\text{out}} .$$

Thus, for the sake of simplicity, we will denote

$$\text{state}_b(\mathbf{x}^{(f_{\text{in}})}) = \text{state}_b(\langle \mathbf{x}^{(f_{\text{in}})}, \mathbf{y}^{(f_{\text{out}})} \rangle) .$$

We remark that the state of an $(f_{\text{in}}, f_{\text{out}})$-balancer is some integer in the set $\{0, 1, \ldots, f_{\text{out}} - 1\}$, which captures the "position" to which it is set as a toggle mechanism. This integer is determined by either the balancer input state variables or the balancer output state variables in the quiescent configuration. Note that the state of the balancer in the initial configuration is 0. Moreover, the linearity of the modulus operation immediately implies linearity for the balancer state.

Lemma 1. *Consider a balancer* $b : \mathbf{x}^{(f_{\text{in}})} \rightarrow \mathbf{y}^{(f_{\text{out}})}$*. Then, for any input vectors* $\mathbf{x}_1^{(f_{\text{in}})}$ *and* $\mathbf{x}_2^{(f_{\text{in}})}$*,*

$$\mathit{state}_b(\mathbf{x}_1^{(f_{\text{in}})} + \mathbf{x}_2^{(f_{\text{in}})}) = (\mathit{state}_b(\mathbf{x}_1^{(f_{\text{in}})}) + \mathit{state}_b(\mathbf{x}_2^{(f_{\text{in}})})) \ \mathit{mod}\ f_{\text{out}} .$$

A $(w_{\text{in}}, w_{\text{out}})$-*balancing network* $\mathcal{B}$ is a collection of interwired balancers, where output wires are connected to input wires, having w_{in} designated input wires, numbered $0, 1, \ldots, w_{\text{in}} - 1$, which are not connected to output wires of balancers, having w_{out} designated output wires, numbered $0, 1, \ldots, w_{\text{out}} - 1$, similarly not connected to input wires of balancers, and containing no cycles. Tokens and antitokens arrive on the network's input wires at arbitrary times, and they traverse a sequence of balancers in the network in a completely asynchronous way till they exit on the output wires of the network.

For each input index i, $0 \leq i \leq w_{\text{in}} - 1$, we denote by x_i the *network input state variable* that stands for the algebraic sum of the numbers of tokens and antitokens that have entered on input wire i; that is, x_i is the difference of the number of tokens that have entered on input wire i *minus* the number of antitokens that have entered on input wire i. Denote $\mathbf{x}^{(w_{\text{in}})} = \langle x_0, x_1, \ldots, x_{w_{\text{in}}-1} \rangle^{\text{T}}$;

call $\mathbf{x}^{(w_{\text{in}})}$ an *input vector*. For each output index j, $0 \leq j \leq w_{\textbf{out}} - 1$, we denote by y_j the *network output state variable* that stands for the algebraic sum of the numbers of tokens and antitokens that have exited on output wire j; that is, y_j is the number of tokens that have exited on output wire j *minus* the number of antitokens that have exited on output wire i. Denote $\mathbf{y}^{(w_{\text{out}})} = \langle y_0, y_1, \ldots, y_{w_{\text{in}}-1} \rangle^{\text{T}}$; call $\mathbf{y}^{(w_{\text{out}})}$ an *output vector*.

The *configuration* of a network at any given time is the tuple of configurations of its individual balancers. In the *initial configuration,* all input and output wires of balancers are empty. The safety and liveness property for a balancing network follow naturally from those of its balancers. Thus, a balancing network eventually reaches a *quiescent configuration* in which all tokens and antitokens that entered the network have exited. In any quiescent configuration of $\mathcal{B}$ we have $\|\mathbf{x}^{(w_{\text{in}})}\|_1 = \|\mathbf{y}^{(w_{\text{out}})}\|_1$; that is, in a quiescent configuration, the algebraic sum of tokens and antitokens that exited the network is equal to the algebraic sum of tokens and antitokens that entered it.

Naturally, we are interested in quiescent configurations of a network. For any quiescent configuration of a network $\mathcal{B}$ with corresponding input and output vectors $\mathbf{x}^{(w_{\text{in}})}$ and $\mathbf{y}^{(w_{\text{out}})}$, respectively, the *state* of $\mathcal{B}$, denoted $\text{state}_{\mathcal{B}}(\mathbf{x}^{(w_{\text{in}})})$, is defined to be the collection of the states of its individual balancers. We remark that we have specified $\mathbf{x}^{(w_{\text{in}})}$ as the single argument of $\text{state}_{\mathcal{B}}$, since $\mathbf{x}^{(w_{\text{in}})}$ uniquely determines all input and output vectors of balancers of $\mathcal{B}$, which are used for defining the states of the individual balancers. Note that the state of the network in its initial configuration is a collection of 0's. For any input vector $\mathbf{x}^{(w_{\text{in}})}$, denote $\mathbf{y}^{(w_{\text{out}})} = \mathcal{B}(\mathbf{x}^{(w_{\text{in}})})$ the output vector in the quiescent configuration that $\mathcal{B}$ will reach after all $\|\mathbf{x}^{(w_{\text{in}})}\|_1$ tokens and antitokens that entered $\mathcal{B}$ have exited; write also $\mathcal{B} : \mathbf{x}^{(w_{\text{in}})} \rightarrow \mathbf{y}^{(w_{\text{out}})}$ to denote the network $\mathcal{B}$.

Not all balancing networks satisfy the step property. A $(w_{\text{in}}, w_{\text{out}})$-*counting network* is a $(w_{\text{in}}, w_{\text{out}})$-balancing network for which, in any quiescent configuration, for any pair of indices j and k such that $0 \leq j < k \leq w_{\text{out}} - 1$, $0 \leq y_j - y_k \leq 1$; that is, the output of a counting network has the step property.

The definition of a counting network can be weakened as follows [1, 4]. For any integer $K \geq 1$, a $(w_{\text{in}}, w_{\text{out}})$-*$K$-smoothing network* is a $(w_{\text{in}}, w_{\text{out}})$-balancing network for which, in any quiescent configuration, for any pair of indices j and k such that $0 \leq j, k \leq w_{\text{out}} - 1$, $0 \leq |y_j - y_k| \leq K$; that is, the output vector of a K-smoothing network has the *K-smoothing property*: all outputs are within K to each other.

For a balancing network $\mathcal{B}$, the *depth* of $\mathcal{B}$, denoted $\text{depth}(\mathcal{B})$, is defined to be the maximum distance from any of its input wires to any of its output wires. In case $\text{depth}(\mathcal{B}) = 1$, $\mathcal{B}$ will be called a *layer*. If $\text{depth}(\mathcal{B}) = d$ is greater than one, then $\mathcal{B}$ can be uniquely partitioned into layers $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_d$ from left to right in the obvious way.

Fix any integer $g \geq 2$. For any integer $K \geq 1$, the *K-smoothing property* [1] is defined to be the set of all vectors $\mathbf{y}^{(g)}$ such that for any entries y_j and y_k of $\mathbf{y}^{(g)}$, where $0 \leq j, k \leq g - 1$, $|y_j - y_k| \leq K$. A *boundedness property* is any subset of some K-smoothing property, for any integer $K \geq 1$, that is closed under addition with a constant vector. Clearly, the K-smoothing property is trivially a boundedness property; moreover, the set of all vectors $\mathbf{y}^{(g)}$ that have the *step property* [4] is a boundedness property, since any step vector is 1-smooth (but not vice versa). We remark that there are infinitely many boundedness properties.

A boundedness property captures precisely the two properties possessed by both K-smooth and step vectors upon which our later proofs will rely. Although we are unaware of any interesting property, other than the K-smoothing and step, that is a boundedness one, we chose to state our results for any general boundedness property in order to make explicit the two critical properties that are common to the classes of K-smooth vectors and step vectors; moreover, arguing in terms of a boundedness property will allow for a single proof of all claims found to hold for both the K-smoothing property and the step property.

Say that *a vector* $\mathbf{y}$ *has the boundedness property* $\mathbf{\Pi}$ if $\mathbf{y} \in \mathbf{\Pi}$. Say that *a balancing network* $\mathcal{B} : \mathbf{x}^{(w_{\text{in}})} \rightarrow \mathbf{y}^{(w_{\text{out}})}$ *has the boundedness property* $\mathbf{\Pi}$ if for every input vector $\mathbf{x}^{(w_{\text{in}})}$, $\mathcal{B}(\mathbf{x}^{(w_{\text{in}})}) \in \mathbf{\Pi}$.

3 Results

Input vectors $\mathbf{x}_1^{(f_{\text{in}})}$ and $\mathbf{x}_2^{(f_{\text{in}})}$ are a *fooling pair to balancer* $b : \mathbf{x}^{(f_{\text{in}})} \rightarrow \mathbf{y}^{(f_{\text{out}})}$ if

$$\text{state}_b(\mathbf{x}_1^{(f_{\text{in}})}) = \text{state}_b(\mathbf{x}_2^{(f_{\text{in}})})\,;$$

roughly speaking, inputs in a fooling pair drive the balancer to identical states.

Proposition 1. *Consider a balancer* $b : \mathbf{x}^{(f_{\text{in}})} \rightarrow \mathbf{y}^{(f_{\text{out}})}$. *Take any input vectors* $\mathbf{x}_1^{(f_{\text{in}})}$ *and* $\mathbf{x}_2^{(f_{\text{in}})}$ *that are a fooling pair to balancer* b. *Then, for any input vector* $\mathbf{x}^{(f_{\text{in}})}$,

(1) *the input vectors* $\mathbf{x}_1^{(f_{\text{in}})} + \mathbf{x}^{(f_{\text{in}})}$ *and* $\mathbf{x}_2^{(f_{\text{in}})} + \mathbf{x}^{(f_{\text{in}})}$ *are a fooling pair to balancer* b*;*
(2) $b(\mathbf{x}_1^{(f_{\text{in}})} + \mathbf{x}^{(f_{\text{in}})}) - b(\mathbf{x}_1^{(f_{\text{in}})}) = b(\mathbf{x}_2^{(f_{\text{in}})} + \mathbf{x}^{(f_{\text{in}})}) - b(\mathbf{x}_2^{(f_{\text{in}})})$.

Input vectors $\mathbf{x}_1^{(w_{\text{in}})}$ and $\mathbf{x}_2^{(w_{\text{in}})}$ are *a fooling pair to network* $\mathcal{B} : \mathbf{x}^{(w_{\text{in}})} \rightarrow \mathbf{y}^{(w_{\text{out}})}$ if for each balancer b of $\mathcal{B}$, the input vectors of b in quiescent configurations corresponding to $\mathbf{x}_1^{(w_{\text{in}})}$ and $\mathbf{x}_2^{(w_{\text{in}})}$, respectively, are a fooling pair to b; roughly speaking, a fooling pair "drives" all balancers of the network to identical states in the two corresponding quiescent configurations.

Proposition 2. *Consider a balancing network* $\mathcal{B} : \mathbf{x}^{(w_{\text{in}})} \rightarrow \mathbf{y}^{(w_{\text{out}})}$. *Take any input vectors* $\mathbf{x}_1^{(w_{\text{in}})}$ *and* $\mathbf{x}_2^{(w_{\text{in}})}$ *that are a fooling pair to network* $\mathcal{B}$. *Then, for any input vector* $\mathbf{x}^{(w_{\text{in}})}$,

(1) *the input vectors* $\mathbf{x}_1^{(w_{\text{in}})} + \mathbf{x}^{(w_{\text{in}})}$ *and* $\mathbf{x}_2^{(w_{\text{in}})} + \mathbf{x}^{(w_{\text{in}})}$ *are a fooling pair to network* $\mathcal{B}$*;*
(2) $\mathcal{B}(\mathbf{x}_1^{(w_{\text{in}})} + \mathbf{x}^{(w_{\text{in}})}) - \mathcal{B}(\mathbf{x}_1^{(w_{\text{in}})}) = \mathcal{B}(\mathbf{x}_2^{(w_{\text{in}})} + \mathbf{x}^{(w_{\text{in}})}) - \mathcal{B}(\mathbf{x}_2^{(w_{\text{in}})})$.

Say that $\mathbf{x}^{(w_{\rm in})}$ is a *null vector* to network $\mathcal{B} : \mathbf{x}^{(w_{\rm in})} \rightarrow \mathbf{y}^{(w_{\rm out})}$ if the vectors $\mathbf{x}^{(w_{\rm in})}$ and $\mathbf{0}^{(w_{\rm in})}$ are a fooling pair to $\mathcal{B}$. Intuitively, a null vector "hides" itself in the sense that it does not alter the state of $\mathcal{B}$ by traversing it.

Proposition 3. *Consider a balancing network $\mathcal{B} : \mathbf{x}^{(w_{\rm in})} \rightarrow \mathbf{y}^{(w_{\rm out})}$. Take any input vectors $\mathbf{x}_1^{(w_{\rm in})}$ and $\mathbf{x}_2^{(w_{\rm in})}$ that are a fooling pair to network $\mathcal{B}$. If $\mathbf{x}_2^{(w_{\rm in})}$ is a null vector to network $\mathcal{B}$, then, $\mathbf{x}_1^{(w_{\rm in})}$ is also a null vector to network $\mathcal{B}$.*

Proposition 4. *Consider a balancing network $\mathcal{B} : \mathbf{x}^{(w_{\rm in})} \rightarrow \mathbf{y}^{(w_{\rm out})}$. Take any input vector $\mathbf{x}^{(w_{\rm in})}$ that is null to $\mathcal{B}$. Then, for any integer $k \geq 0$,*
(1) $\mathcal{B}(k\mathbf{x}^{(w_{\rm in})}) = k\mathcal{B}(\mathbf{x}^{(w_{\rm in})})$;
(2) $k\mathbf{x}^{(w_{\rm in})}$ *is a null vector to* $\mathcal{B}$.

For any balancing network $\mathcal{B}$, let $W_{\rm out}(\mathcal{B})$ denote the product of the fan-outs of balancers of $\mathcal{B}$. For positive integer δ, say that δ *divides* $\mathbf{x}^{(g)}$ if δ divides each entry of $\mathbf{x}^{(g)}$.

Proposition 5. *Consider a balancing network $\mathcal{B} : \mathbf{x}^{(w_{\rm in})} \rightarrow \mathbf{y}^{(w_{\rm out})}$. If $W_{\rm out}(\mathcal{B})$ divides $\mathbf{x}^{(w_{\rm in})}$, then, $\mathbf{x}^{(w_{\rm in})}$ is a null vector to $\mathcal{B}$.*

Proposition 6. *Consider any balancing network $\mathcal{B} : \mathbf{x}^{(w_{\rm in})} \rightarrow \mathbf{y}^{(w_{\rm out})}$ that has a boundedness property $\mathbf{\Pi}$. If $W_{\rm out}(\mathcal{B})$ divides $\mathbf{x}^{(w_{\rm in})}$, then, $\mathbf{y}^{(w_{\rm out})}$ is a constant vector.*

Here is our main result:

Theorem 1. *Fix any boundedness property $\mathbf{\Pi}$. Consider any balancing network $\mathcal{B} : \mathbf{x}^{(w_{\rm in})} \rightarrow \mathbf{y}^{(w_{\rm out})}$ such that $\mathbf{y}^{(w_{\rm out})}$ has the boundedness property $\mathbf{\Pi}$ whenever $\mathbf{x}^{(w_{\rm in})}$ is a non-negative vector. Then, $\mathcal{B}$ has the boundedness property $\mathbf{\Pi}$.*

4 Conclusion

We have shown that any balancing network that satisfies any boundedness property on all non-negative input vectors, continues to do so for any arbitrary input vector. Interesting examples of such properties are the step property and the K-smoothing property. A significant consequence of our result is that all known (deterministic) constructions of counting and smoothing networks [1, 3–5, 8, 10, 11, 14, 15, 19] will correctly handle both tokens and antitokens, and therefore support both increment and decrement operations. Another significant consequence is that the sufficient timing conditions for linearizability in counting networks established in [16, 17] immediately carry over to antitokens.

Aiello *et al.* [3] present a *randomized* counting network based on *randomized balancers* that toggle tokens according to some random permutation. We do not know whether such randomized networks can support antitokens.

A balancing network has the *threshold property* [4, 7] if $y_0 = \lceil \|\mathbf{x}^{(w_{\rm in})}\|_1 / w_{\rm out} \rceil$, and the *weak threshold property* [7] if there is *some* output index j, possibly $j \neq 0$, such that $y_j = \lceil \|\mathbf{x}^{(w_{\rm in})}\|_1 / w_{\rm out} \rceil$. Since we have not established that either of these properties is a boundedness property, our result does not necessarily apply, and it remains unknown whether these properties are preserved by the introduction of antitokens.

References

1. E. Aharonson and H. Attiya. Counting networks with arbitrary fan-out. *Distributed Computing*, 8(4):163–169, 1995.
2. W. Aiello, M. Herlihy, N. Shavit, and D. Touitou. Inc/dec counting networks. Manuscript, Dec. 1995.
3. W. Aiello, R. Venkatesan, and M. Yung. Coins, weights and contention in balancing networks. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing (PODC'94)*, pages 193–205, Los Angeles, Aug. 1994.
4. J. Aspnes, M. Herlihy, and N. Shavit. Counting networks. *Journal of the ACM*, 41(5):1020–1048, Sept. 1994.
5. C. Busch, N. Hardavellas, and M. Mavronicolas. Contention in counting networks (abstract). In *Proceedings of the 13th annual ACM Symposium on Principles of Distributed Computing (PODC'94)*, page 404, Los Angeles, Aug. 1994.
6. C. Busch and M. Mavronicolas. The strength of counting networks (abstract). In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, page 311, Philadelphia, May 1996.
7. C. Busch and M. Mavronicolas. Impossibility results for weak threshold networks. *Information Processing Letters*, 63(2):85–90, July 1997.
8. C. Busch and M. Mavronicolas. An efficient counting network. In *Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP'98)*, pages 380–385, Mar. 1998.
9. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press and McGraw-Hill Book Company, Cambridge, MA, 1992.
10. E. W. Felten, A. LaMarca, and R. Ladner. Building counting networks from larger balancers. Technical Report TR 93-04-09, University of Washington, Apr. 1993.
11. N. Hardavellas, D. Karakos, and M. Mavronicolas. Notes on sorting and counting networks. In *Proceedings of the 7th International Workshop on Distributed Algorithms (WDAG'93)*, volume 725 of *Lecture Notes in Computer Science*, pages 234–248, Lausanne, Switzerland, Sept. 1993. Springer-Verlag.
12. M. Herlihy, B.-H. Lim, and N. Shavit. Scalable concurrent counting. *ACM Transactions on Computer Systems*, 13(4):343–364, Nov. 1995.
13. S. Kapidakis and M. Mavronicolas. Distributed, low contention task allocation. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing (SPDP'96)*, pages 358–365, Washington, Oct. 1996.
14. M. Klugerman. *Small-Depth Counting Networks and Related Topics.* PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, Sept. 1994.
15. M. Klugerman and C. G. Plaxton. Small-depth counting networks. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing (STOC'92)*, pages 417–428, Victoria, B.C., Canada, May 1992.
16. N. Lynch, N. Shavit, A. Shvartsman, and D. Touitou. Counting networks are practically linearizable. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, pages 280–289, New York, May 1996.
17. M. Mavronicolas, M. Papatriantafilou, and P. Tsigas. The impact of timing on linearizability in counting networks. In *Proceedings of the 11th International Parallel Processing Symposium (IPPS'97)*, pages 684–688, Los Alamitos, Apr. 1997.
18. N. Shavit and D. Touitou. Elimination trees and the construction of pools and stacks. *Theory of Computing Systems*, 30(6):545–570, Nov./Dec. 1997.
19. N. Shavit and A. Zemach. Diffracting trees. *ACM Transactions on Computer Systems*, 14(4):385–428, Nov. 1996.