# Size–Time Complexity of Boolean Networks for Prefix Computations

G. BILARDI

*Cornell University, Ithaca, New York*

AND

F. P. PREPARATA

*University of Illinois at Urbana-Champaign, Urbana, Illinois*

Abstract. The prefix problem consists of computing all the products $x_0 x_1 \cdots x_j$ $(j = 0, \ldots, N-1)$, given a sequence $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})$ of elements in a semigroup. In this paper we completely characterize the size–time complexity of computing prefixes with Boolean networks, which are synchronized interconnections of Boolean gates and one-bit storage devices. This complexity crucially depends upon two properties of the underlying semigroup, which we call cycle-freedom (no cycle of length greater than one in the Cayley graph of the semigroup), and memory-induciveness (arbitrarily long products of semigroup elements are true functions of all their factors). A nontrivial characterization is given of non-memory-inducive semigroups as those whose recurrent subsemigroup (formed by the elements with self-loops in the Cayley graph) is the direct product of a left-zero semigroup and a right-zero semigroup. Denoting by $S$ and $T$ size and computation time, respectively, we have $S = \Theta((N/T)\log(N/T))$ for memory-inducive non-cycle-free semigroups, and $S = \Theta(N/T)$ for all other semigroups. We have $T \in [\Omega(\log N), O(N)]$ for all semigroups, with the exception of those whose recurrent subsemigroup is a right-zero semigroup, for which $T \in [\Omega(1), O(N)]$. The preceding results are also extended to the VLSI model of computation. Area-time optimal circuits are obtained for both boundary and nonboundary I/O protocols.

Categories and Subject Descriptors: F.1.1 **[Computation by Abstract Devices]**: Models of Computation—*unbounded-action devices*; F.2.2 **[Analysis of Algorithms and Problem Complexity]**: Nonnumerical Algorithms and Problems—*computations on discrete structures*; F.2.3 **[Analysis of Algorithms and Problem Complexity]**: Trade-off among Complexity Measures

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Boolean networks, parallel computation, prefix computation, semigroups, size–time trade-offs

---

A preliminary version of this work appears in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing* (New York, N.Y., May 25–27). ACM, New York, 1987, pp. 436–442.

The research of G. Bilardi was supported in part by National Science Foundation grant DCI 86-02256. The research of F. P. Preparata was supported in part by National Science Foundation grant ECS 84-10902.

Authors' present addresses: G. Bilardi, Department of Computer Science, Cornell University, Ithaca, NY 14853. F. P. Preparata, Coordinated Science Laboratory, Departments of Electrical and Computer Engineering and of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL 61801.

## 1. *Introduction*

The *prefix problem* consists of computing all the products of $x_0 x_1 \cdots x_j$ ($j = 0, \ldots, N-1$), given a sequence $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})$ of elements in a semigroup. Prefix computations occur in the solution of several significant problems such as carry-look-ahead addition [7, 21], the evolution of finite-state machines [19], linear recurrences [18], digital filtering [5], various graph problems [17], sorting in bit-models of computation [3, 11], scheduling [13], and others.

The prefix problem has been extensively investigated in the *circuit* model, where the computation is carried out by an acyclic network of *gates*. Various complexity measures such as *size*, *depth*, *width*, and their trade-offs have been studied in [15], [19], and [22] for circuits whose gates are semigroup multipliers, and in [8] and [9] for circuits of Boolean gates. Algorithms for the EREW-PRAM model have been proposed in [17]. Implementations on a *tree-connected* network are discussed in [13].

In this paper we study the complexity of computing prefixes with *Boolean networks*, which are synchronized interconnections of Boolean gates and one-bit storage devices. Relevant measures are *computation time* $T$ and *size* $S$, defined as the total number of components (combinational and sequential) in the network. Our model of computation is essentially the same as the *aggregate* of [14], from which it differs only in the input/output conventions. Both models afford the study of the role of sequential logic in circuits, and allow the consideration of circuits of size sublinear in the input size. Results in Boolean networks have also interesting implications for other models of parallel computation such as fixed interconnections of processors and VLSI circuits [23].

We have found that the size–time complexity of the prefix problem is determined by two properties of the underlying semigroup, which we call *cycle-freedom* and *memory-inducivenes*s. We call a semigroup cycle-free if its Cayley graph has no cycle of length greater than one and non-cycle-free otherwise. We call a semigroup memory-inducive if products of arbitrary length are true functions of all their factors, and non-memory-inducive otherwise. Our results, which completely characterize the size–time complexity of the prefix problem, are encompassed by the following theorem, which summarizes Theorems 4 to 9.

THEOREM 1. *The size–time complexity of the prefix problem on a Boolean network is* $S = \Theta((N/T)\log(N/T))$, *for memory-inducive non-cycle-free semigroups, and is* $S = \Theta(N/T)$, *for all other semigroups. The bounds hold for* $T \in [\Omega(\log N), O(N)]$ *for all semigroups, with the exception of those whose recurrent subsemigroup is a right-zero semigroup, for which* $T \in [\Omega(1), O(N)]$.

For memory-inducive non-cycle-free semigroups, the upper bound can be achieved by known constructions based on binary-tree networks [13], or twisted-reflected-tree networks [3, 7, 19, 21], whereas the lower bound (Sect. 3.4, Theorem 4) is less obvious, and is based on arguments of computational friction [4]. We also give a nontrivial characterization of non-memory-inducive semigroups (Sect. 3.3, Theorem 3).

For the remaining semigroups, the lower bound is based on a trivial input/output argument, while the upper bound is achieved by nontrivial algorithms (Section 4, Theorems 7 and 9) executed by tree-connected networks.

It may be interesting to contrast Theorem 1 with the result of [8] that there are constant-depth, polynomial-size (unbounded fan-in) Boolean circuits to compute prefixes for a semigroup, if and only if the semigroup is *group-free*, an attribute weaker than cycle-free.

In Section 5, we extend our results to the area–time complexity of the prefix problem in the VLSI model of computation [23].

We conclude in Section 6 by considering some open problems.

## 2. *Definitions and Problem Statement*

A *finite semigroup* is a pair $\langle A, \cdot \rangle$ where $A = \{a_1, a_2, \ldots, a_s\}$ is a set of *size* $s$ and $\cdot$ is an *associative* binary operation on $A$, which we call *product*. We denote by $xy$ the product of elements $x, y \in A$. A *finite monoid* is a finite semigroup with a distinguished element $e$, called the *identity*, such that $xe = ex = x$, for all $x \in A$. Any semigroup can be easily transformed into a monoid by the addition of an element with the properties of the identity. Perhaps surprisingly, the addition of the identity may increase the complexity of computing prefixes.

For a sequence $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1}) \in A^N$, the sequence of *prefixes* of $\mathbf{x}$ is defined as $\mathbf{y} = (y_0, y_1, \ldots, y_{N-1})$, with $y_j = x_0 x_1 \cdots x_j$. The *prefix problem* consists in computing $\mathbf{y}$ from $\mathbf{x}$.

In the study of the complexity of the prefix problem, an important role is played by the *Cayley graph* $G(A) = (A, E)$ of $A$, containing for each ordered pair $(x, y)$ an arc of the form $(x, xy)$, labeled by $y$. It is easy to see that each node of $G(A)$ has out-degree $s$, that the labels of the self-loops of a given node form a subsemigroup of $A$, and that $G(A)$ is transitively closed.

An element $a$ of $A$ is said to be *recurrent* if $G(A)$ contains a self-loop at $a$, that is, if there is an element $b$ of $A$, not necessarily distinct from $a$, such that $ab = a$. We call *recur*$(A)$ the set of recurrent elements of $A$. It is easy to see that *recur*$(A)$ is a subsemigroup.

Two elements $a$ and $b$ of $A$ are *equivalent*, if either $a = b$ or there exist elements $c$ and $d$ such that $ac = b$, and $bd = a$. We observe that the relation "equivalent" is a true equivalence, and that its equivalence classes are the strongly connected components of $G(A)$.

We call a semigroup *cycle-free* (*CF*) if the only cycles in its Cayley graph are self-loops, and *non-cycle-free* (*NCF*), otherwise. We shall see that cycle-freedom is a crucial property of a semigroup in determining the complexity of the prefix problem.

It is well known (and also easy to prove) that for each element $a$ in a finite semigroup there are two positive integers $k$ and $p$ such that $a, a^2, \ldots, a^{k+p-1}$ are all distinct, and $a^{k+p} = a^k$. Moreover, if the *period* $p$ of $a$ is larger than 1, then $\{a^k, a^{k+1}, \ldots, a^{k+p-1}\}$ from a group. If $p = 1$ for all the elements, then the semigroup is called *group-free* [20].

We now give examples of semigroups that belong to the various classes introduced above. If any element $x \in A$ different from the identity has an inverse $x^{-1}$ such that $xx^{-1} = e$, then $(e, x, e)$ forms a nontrivial cycle in $G(A)$ and $A$ is NCF. As a corollary, all *groups* are NCF, and cycle-freedom implies group-freedom.

An example of CF semigroup is the *left-zero* semigroup $\langle L_p, \circ \rangle$, where $L_p = \{l_1, l_2, \ldots, l_p\}$ and $l_i \circ l_j = l_i$, for all $l_i$ and $l_j$. An example of semigroup that is NCF but group-free is the *right-zero* semigroup $\langle R_q, * \rangle$, where $R_q = \{r_1, r_2, \ldots, r_q\}$ and $q \geq 2$, and $r_i * r_j = r_j$, for all $r_i$ and $r_j$. This semigroup with $q = 2$, and with the adjunction of the identity, becomes the monoid that models the function "carry" in binary addition (the identity representing carry propagation and the two zeros representing carry setting and carry resetting, respectively).

Among CF semigroups, of particular interest are *insertion* semigroups, characterized by the following property: For all $x, y, z, w \in A$,

$$xyz = xy \Rightarrow xwyz = xwy. \tag{1}$$

All *abelian* CF semigroups are also insertion semigroups. An instance of abelian CF semigroup is given by a set $A = \{0, 1, \ldots, s-1\}$ with respect to the operation *threshold-*$(s-1)$ *addition* defined as $xy = \min(x + y, s - 1)$. The prefix operation on this semigroup represents the cumulative sum of the sequence **x** with the value $(s-1)$ replacing each larger value.

Further examples of insertion semigroups are all *semilattices*, where the semigroup operation is commutative and idempotent. Examples of semilattices are the set of the 0-1 vectors of length $n$ with respect to component-wise OR (AND), and the set of the first $s$ nonnegative integers with the MINIMUM (MAXIMUM) operation.

An interesting insertion semigroup that is not abelian is the set of the rankings of $n$ items with respect to the operation of rank concatenation. The computation of prefixes in this semigroup is used in the construction of optimal VLSI-sorting circuits [3, 4, 11]. Identifying the $n$ items with the integers from 1 to $n$, a *ranking* is an ordered partition of the set $\{1, 2, \ldots, n\}$, that is, a sequence of disjoint sets whose union equals $\{1, 2, \ldots, n\}$. Intuitively, all the elements in a given set have the same rank, and have rank higher than those in the next set. The *concatenation* of two rankings $u = (u_1, u_2, \ldots, u_p)$ and $v = (v_1, v_2, \ldots, v_q)$ is $uv = (w_1, w_2, \ldots, w_p)$ with $w_j$ equal to the subsequence of the nonempty terms of $(u_j \cap v_1, u_j \cap v_2, \ldots, u_j \cap v_q)$.

## 3. *Lower Bounds*

3.1 MODEL. A *Boolean network* is a directed graph with the following types of nodes: (1) input nodes, with in-degree zero and out-degree one; (2) output nodes, with in-degree one and out-degree zero; (3) combinational nodes, each labeled by a Boolean function of one- or two-input variables, with in-degree equal to the number of input variables, and out-degree one or two (to allow fan-out); (4) one-bit storage nodes, with in-degree one and out-degree one or two.

The notions of computation of, and of function computed by, a Boolean network can be formalized as done in [14]. Here we appeal to the intuitive meaning of these notions, and just discuss the input/output protocol, since it differs slightly from that of [14]. We assume that each input (output) variable of the problem is assigned one input (output) node and one input (output) time. Two variables can be assigned the same node, but only at different times. Only one node and one time are assigned to a given variable (unilocal, semellective protocol), and this node and time are independent of the input value (place-determinate, time-determinate protocol).

Clearly, when solving the prefix problem by a Boolean network, a specific binary encoding of the semigroup elements must be chosen. Since our present aim is to study the dependence of the complexity of the prefix problem upon the length $N$ of the input sequence, and not its dependence on semigroup size or representation, we assume that the bits that encode a given semigroup variable are input (output) all at the same time. We call an input/output protocol with this property *word-instantaneous*, in analogy with the term *word-local* introduced in [23].

3.2 COMPUTATIONAL FRICTION. Our lower bound for the prefix problem is based on the mechanism of *computational friction* developed in [4] as a generalization of arguments previously applied to binary addition in [1] and [16]. Computational friction, so denoted in the context of a fluidodynamic analogy for VLSI computations, is a phenomenon that slows down the flow of information from input to output nodes below the rate allowed by the number of I/O nodes, and

therefore, when present, yields lower bounds stronger than the trivial $ST = \Omega(N)$ bound. As noted in [4], computational friction rests crucially on the property that both fan-in and fan-out of logical elements are bounded, and it can be analyzed—in perfectly symmetrical ways—with respect to either fan-in or fan-out. With respect to fan-in, two phenomena contribute to the appearance of friction: (i) A substantial fraction of the information carried by each wavefront of input variables is transferred to the output variables, and (ii) this information must be stored within the network for a time logarithmic in the input wavefront size hence, for bounded fan-in, functional dependence imposes a delay between reading the input and computing an output depending upon that input. Symmetrically (with respect to fan-out), we have: (i) each wavefront of output variables carries a substantial fraction of the information contained in the input variables, and (ii) this information must be stored within the network for a time logarithmic in the output wavefront size since, for bounded fan-out, functional dependence imposes a delay between reading an input and computing the outputs depending upon it. For the sake of the ensuing discussion, we precisely analyze the latter phenomenon (fan-out constraint) in Theorem 2 below, an earlier version of which is proved in [4].

We begin with the following definition of functional dependence:

*Definition* 1. Let $X = \{x_0, \ldots, x_{n-1}\}$ and $Y = \{y_0, \ldots, y_{m-1}\}$ be sets of input and output variables, respectively; each variable ranges over a finite alphabet. All elements of $X$ and $Y$ are encoded in binary, and the binary encoding of a variable will be referred to as a *word*. Let $\mathbf{x} = (x_0, \ldots, x_{n-1})$ and $\mathbf{y} = (y_1, \ldots, y_{m-1})$ denote vectors and let $f$: $\mathbf{x} \mapsto \mathbf{y}$ be a function. We say that $y_j$ is *functionally dependent* upon $x_i$ (and that $x_i$ *functionally affects* $y_j$) if there exist two vectors $\mathbf{x}'$ and $\mathbf{x}''$ that differ only in their $x_i$-component and such that $f(\mathbf{x}')$ and $f(\mathbf{x}'')$ differ in their $y_j$-component.

Next, we introduce the notion of friction set, which formally captures the features of computational friction intuitively described above.

*Definition* 2. Let $f$, $X$, and $Y$ be as introduced in Definition 1. Let $\alpha$ and $\beta$ be nondecreasing functions. Set $U \subseteq Y$ is called an $(\alpha, \beta)$-*friction set* for $f$ if for any subset $W$ of $U$ there exists a subset $V$ of $X$ with the two following properties:

(1) Each variable of $V$ functionally affects at least $\alpha(|W|)$ variables of $W$.
(2) There is an assignment of values to the variables of $X - V$ such that, in the resulting restriction of $f$, the variables in $W$ assume at least $2^{\beta(|W|)}$ distinct configurations.

The significance of friction sets rests on the following theorem.

THEOREM 2. *Let $f$, $X$, and $Y$ be as introduced in Definition* 1, *and let $U \subseteq Y$ be an $(\alpha, \beta)$-friction set for $f$. Then for any time-determinate, semellective, word-instantaneous Boolean network computing $f$, size $S$ and time $T$ satisfy the bound*

$$S = \Omega\left(\beta\left(\frac{|U|}{T} \log \alpha\left(\frac{|U|}{T}\right)\right)\right). \tag{2}$$

In the proof of Theorem 2 we shall use the simple combinatorial lemma stated below without proof.

LEMMA 1. *Let $(u_1, u_2, \ldots, u_T)$ be a sequence of integers, and let $m = (u_1 + u_2 + \cdots + u_T)/T$ be their average. Then, for every integer $\tau \in [1, T]$, there are $\tau$*

*consecutive terms of the sequence whose average is at least m/2. More formally, there is an integer* $t_0 \in [1, T - \tau + 1]$ *such that* $(u_{t_0} + u_{t_0+1} + \cdots + u_{t_0+\tau-1})/\tau \geq m/2$.

PROOF OF THEOREM 2. Let $C$ be a Boolean network that computes $f$, and let the origin of time be chosen so that the computation takes place in the interval $[1, T]$. Since $C$ is word-instantaneous, we can define the set $U_t$ of the variables in $U$ that are output exactly at time $t \in [1, T]$. ($U_t$ may be empty for some $t$.) Since $C$ is time-determinate, $U_t$ is independent of the input values, and since $C$ is semellective (in the output), the $U_t$'s are pairwise disjoint. Finally, the union of the $U_t$'s equals $U$.

From Lemma 1, with $u_t = |U_t|$, and $m = |U|/T$, we have that, for any $\tau \in [1, T]$, there is a $t_0 \in [1, T - \tau + 1]$ such that $(|U_{t_0}| + |U_{t_0+1}| + \cdots + |U_{t_0+\tau-1}|) \geq \tau|U|/(2T)$. Then, the set $W \triangleq \bigcup_{t=t_0}^{t_0+\tau-1} U_t$ satisfies the bound

$$|W| \geq \frac{\tau|U|}{2T}. \tag{3}$$

Since $W$ is a subset of a friction set, there is by hypothesis a subset $V$ of $X$ satisfying properties (1) and (2) of Definition 2.

We claim that any choice $\tau \leq \log \alpha(|U|/2T)$ guarantees the separation of the time interval $[t_0, t_0 + \tau - 1]$ (during which $W$ is output) and the time interval during which the set $V$ of the variables affecting $W$ is input.

Indeed, noting that $\tau \geq 1$, that $\alpha$ is nondecreasing, and that $|W|$ satisfies (3), we can write

$$\tau \leq \log \alpha\left(\frac{|U|}{2T}\right) \leq \log \alpha\left(\frac{\tau|U|}{2T}\right) \leq \log \alpha(|W|).$$

All variables of $W$ are output no earlier than $t_0$, and no later than $t_0 + \tau - 1$. Since, by Property 1, each variable of $V$ functionally affects at least $\alpha(|W|)$ variables of $W$, no variable of $V$ can be input later than $(t_0 + \tau - 1) - \log \alpha(|W|) \leq t_0 - 1$. This establishes the claim.

We therefore choose $\tau = \lfloor \log \alpha(|U|/2T) \rfloor$. In order to use Property 2 of Definition 1, the values of $X - V$ are held fixed. This implies that the variables of $W$ become function solely of the variables in $V$. By the claim just established, the values of the variables of $W$ must be encoded in the network at time $t_0 - 1$. Property 2 ensures that the assignment of $X - V$ can be chosen so that $W$ assumes at least $2^{\beta(|W|)}$ distinct configurations by varying the values of $V$. Thus, the number of binary storage devices in the network is at least $\beta(|W|)$. We conclude that

$$S \geq \beta(|W|) \geq \beta\left(\frac{|U|}{2T} \log \alpha\left(\frac{|U|}{2T}\right)\right).$$

The latter bound implies (2). □

COROLLARY 1. *If, in Theorem* 2, $\alpha(n) = \Omega(n)$, *and* $\beta(n) = \Omega(n)$, *then*

$$S = \Omega\left(\left(\frac{|U|}{T}\right)\log\left(\frac{|U|}{T}\right)\right). \tag{4}$$

The above discussion indicates that the two crucial ingredients of computational friction are functional dependence of the output upon the input and the transfer of information from input to output (I/O transfer). In the next section we shall investigate the dependence of semigroup products upon their factors.

3.3 FUNCTIONAL DEPENDENCE IN SEMIGROUP PRODUCTS. We now specialize the set $X$ and $Y$ introduced above so that each of their elements is a variable ranging over a finite alphabet $A$, where $\langle A, \cdot \rangle$ is a semigroup. Moreover, the elements of $Y$ are defined as semigroup products of elements of $A$, that is:

$$y_j = x_0 x_1 \cdots x_j \qquad j = 0, 1, \ldots, N-1. \tag{5}$$

The term *$x_i$-factor* denotes the value of the $(i+1)$st term of product (5).

It is convenient to introduce the following notion.

*Definition* 3. A semigroup is called *memory-inducive* (MI) if, for every $j \geq 0$, and for every $0 \leq i \leq j$, $y_j = x_0 x_1 \cdots x_j$ is functionally dependent upon $x_i$. A semigroup is called *non-memory-inducive* (*NMI*) otherwise.

We shall see that most semigroups are MI, and we shall give an exact characterization of those that are NMI. All proofs of memory-inducive ness consider the generic product $y_j$ and, for each $i$ in $[0, j]$, exhibit two distinct selections of factors differing exactly for their $x_i$-factor and producing different values for $y_j$. First we need to define one important class of semigroups:

*Definition* 4. Let $\langle L_p, \circ \rangle$ be a left-zero semigroup and let $\langle R_q, * \rangle$ be a right-zero semigroup, as defined in Section 2. Define $Z_{p,q} = \langle L_p \times R_q, \cdot \rangle$ as the direct-product semigroup [10] of $L_p$ and $R_q$, that is, $(l_h, r_k) \cdot (l_{h'}, r_{k'}) = (l_h, r_{k'})$.

From this definition, it follows that $(l_{h_0}, r_{k_0}) \cdot (l_{h_1}, r_{k_1}) \cdots (l_{h_j}, r_{k_j}) = (l_{h_0}, r_{k_j})$. Thus, $y_j = x_0 x_1 \cdots x_j = x_0 x_j$ depends only on the first and last factor, so that $Z_{p,q}$ is NMI. The significance of the semigroups $Z_{p,q}$'s rests on the role they play in the following theorem.

THEOREM 3. *A semigroup $\langle A, \cdot \rangle$ is NMI if and only if the subsemigroup $\langle recur(A), \cdot \rangle$ is isomorphic to $Z_{p,q}$, for some positive p and q.*

Theorem 3 is crucial for obtaining a complete classification of semigroups based on the complexity of the prefix problem. The proof is somewhat lengthy and technical, and is given in the Appendix.

Theorem 3 has the following interesting consequence, which could also be derived directly from the argument used to prove Claim 2 (see the Appendix).

COROLLARY 2. *An NMI semigroup is monoid-free (none of its subsemigroups is a monoid with at least two elements) and, a fortiori, group-free.*

3.4 I/O INFORMATION TRANSFER IN PREFIX COMPUTATIONS. We now apply the general results embodied by Theorems 2 and 3 to the set of MI-NCF semigroups.

THEOREM 4. *For any time-determinate, semellective, word-instantaneous Boolean network that solves the prefix problem of size N for a MI-NCF semigroup $\langle A, \cdot \rangle$, size and time satisfy the bound*

$$S = \Omega\left(\left(\frac{N}{T}\right) log\left(\frac{N}{T}\right)\right). \tag{6}$$

PROOF. We show that Theorem 2 can be applied, with $X = \{x_0, x_1, \ldots, x_{N-1}\}$, $Y = \{y_0, y_1, \ldots, y_{N-1}\}$, and function $f$ defined by (5). We select $U = \{y_1, y_3, \ldots, y_{2i+1}, \ldots\} \subset Y$ and claim that $U$ is an $(\alpha, \beta)$-friction set for $f$, with $\alpha(n) = \Omega(n)$ and $\beta(n) = \Omega(n)$. Indeed, let $W$ be an arbitrary set of $U$, and let $J$ be such that $W = \{y_j : j \in J\}$. Let $I$ be the set of the $\lfloor |W|/2 \rfloor$ smallest indices in $J$, and

select $V$ as the set $\{x_i, x_{i+1} : i \in I\}$. We now observe:

(1) Since $\langle A, \cdot \rangle$ is MI, each variable of $V$ affects each $y_j$ with $j \in (J - I)$, that is, at least $\lfloor |W|/2 \rfloor$ variables of $W$. (This means that $\alpha(|W|) \geq \lfloor |W|/2 \rfloor$.)

(2) Since $\langle A, \cdot \rangle$ is NCF, then its Cayley diagram has a cycle of length $\geq 2$, and hence a cycle of length exactly 2, that is, there are four elements $a, b, c, d \in A$ (not necessarily all distinct, but with $a \neq b$), such that $ac = b$ and $bd = a$. Consider the input sequences for which $x_0 = a$, and $x_i = (cd)$ for $x_i \in (X - V)$ and $i \neq 0$ (this is the fixed assignment of the variables in $X - V$). For any partition $I = (I_1, I_2)$ of $I$, selecting the variables in $V$ as $x_i = x_{i+1} = (cd)$, for $i \in I_1$, and $x_i = c$ and $x_{i+1} = d$, for $i \in I_2$, results in $y_i = a$ for $i \in I_1$, and $y_i = b$ for $i \in I_2$. Therefore, there are at least $2^{|I|} = 2^{\lfloor |W|/2 \rfloor}$ output configurations for the variables of $W$. (This means that $\beta(|W|) \geq \lfloor |W|/2 \rfloor$.)

Since both conditions of Definition 2 are met, Theorem 2 applies; moreover, since both $\alpha$ and $\beta$ are linear functions, Corollary 2 applies with $|U| = \lfloor N/2 \rfloor$, thereby establishing (6). □

The previous argument cannot be extended to cover either CF or NMI-NCF semigroups; in fact, in the next section we shall describe networks for prefix computation for such semigroups that violate bound (6). To gain some intuition on this phenomenon, we observe that a cycle in the Cayley diagram—as illustrated in the proof of Theorem 4—is the machinery necessary for sustained information transfer, while memory-inducivenes forces temporary storage of this information in the network. The absence of cycles drastically reduces the information that can be transferred: Indeed the (generally nonsimple) path of $G(A)$ corresponding to a sequence $\mathbf{y} = (y_0, \ldots, y_{N-1})$ of prefixes can be described with $O(\log N)$ bits, rather than the $\Omega(N)$ necessary for NCF semigroups. On the other hand, the absence of memory-inducivenes means that the semigroup $\langle A, \cdot \rangle$ satisfies the condition in the statement of Theorem 3. This means that the path corresponding to $\mathbf{y}$ reaches $recur(A)$ in at most $m_A$ steps, and thereafter each output variable becomes a function of a fixed state and of a constant number of input variables, thus requiring no essential temporary storage in the network.

Thus, Theorem 4 precisely partitions the semigroups into two classes: the MI-NCF semigroups, briefly referred to as *friction semigroups*, form one class, while the CF and the NMI-NCF form the other, correspondingly referred to as *frictionless semigroups*.

3.5 Computation Time. Based on the results of Section 3.4, we derive the following lower bound on computation time of prefix networks.

Theorem 5. *Let $\langle A, \cdot \rangle$ be a semigroup such that $recur(A)$ is* not *a right-zero semigroup. Then, the computation time of any Boolean network that solves the prefix problem of size $N$ satisfies the bound $T = \Omega(\log N)$.*

Proof. A simple consequence of bounded fan-out is that, if a given input variable functionally affects $N$ output variables, then $T = \Omega(\log N)$.

If $recur(A)$ is *not* a right-zero semigroup, then either $\langle A, \cdot \rangle$ is MI, or $recur(A)$ is isomorphic to some $Z_{p,q}$, with $p \geq 2$. We claim that in either case $x_0$ functionally affects $y_0, y_1, \ldots, y_{N-1}$. If $\langle A, \cdot \rangle$ is MI, the claim is an obvious consequence of Definition 3. If, on the other hand, $recur(A)$ has at least two strongly connected components, there are two elements $a$ and $b$ that belong to different components.

To see that $x_0$ affects $y_j$, for all $j \geq 0$, we need only to consider that $b = bb^j \neq ab^j$, since $ab^j$ belongs to the same component as $a$. □

## 4. *Upper Bounds*

In this section we present size-time optimal network implementations of prefix algorithms. A network for general semigroups, with size $S = \Theta(N/T \log(N/T))$, is considered in Section 4.1. This network is optimal for MI-NCF (friction) semigroups. More efficient networks, with size $S = \Theta(N/T)$, are possible for both CF and NMI semigroups.

For CF semigroups, the reduction in size is achieved by reducing the amount of information relative to the input that the network stores while computing the output. The details of the design are presented in Section 4.2. In the same section we present an alternative scheme suited for CF insertion semigroups, which represents a simplification of the network for general semigroups.

For NMI semigroups, the reduction in size is achieved by reducing the delay between the input time of a variable $x_i$, and the output time of the corresponding output $y_i$. The network for NMI semigroups is illustrated in Section 4.3.

The constructions of both Section 4.2 and Section 4.3 apply to NMI-CF semigroups. Indeed, the prefix problem for these semigroups degenerates: for $j \geq m_A$, output $y_j$ is guaranteed to be constant with $j$, and equal to some left-zero of $A$.

4.1 GENERAL SEMIGROUPS. The algorithms are executed by a network having the structure of a binary tree $K$ with $w$ leaves. Tree realizations of prefix circuits were reported earlier in [13], and can be viewed as emulations of the well-known prefix network described in [5], [7], [19], and [21] and called *twisted-reflected-tree* in [5].

In tree $K$, leaf nodes perform input/output operations, while internal nodes perform data processing. Each node is bidirectionally connected to its parent and its offsprings.

The input sequence $\mathbf{x} = (x_0, \ldots, x_{N-1})$ is segmented into $N/w$ wavefronts of width $w$, where $1 \leq w \leq N/\log N$ (for ease of discussion, we assume that $N$ is a power of two). The $i$th wavefront is denoted $\mathbf{x}_i = (x_{iw}, x_{iw+1}, \ldots, x_{(i+1)w-1})$, where $i = 0, 1, \ldots, N/w - 1$. The wavefronts are sequentially fed to the network, with $x_{iw+j}$ input at the $j$th leaf (see Figure 1). A fixed wavefront is processed by the network in two phases: an *ascending* phase (consisting of one input step and log $w$ processing steps), when information flows from leaves to root, and a *descending* phase (consisting of log $w$ processing steps and one output step), when the direction of the flow is reversed.

Let the level of a node $V$, denoted $level(V)$, be the number of edges on the path between $V$ and the root of $K$. For each of the algorithms described below, a step takes time $O(1)$ (independent of $N$). Moreover, a given step on a given wavefront is carried out by a single level of nodes, so that the network can be pipelined at a constant rate. Clearly, processing of the $N$-term sequence is completed in $N/w + 2 \log w + 2$ steps, and hence in $T = O(N/w)$.

More subtle is the use of storage at each node, which determines the global size of the network. A fixed wavefront is processed by a given level twice: once during the ascending phase, and then again—$2\ level(V) + 1$ steps later—during the descending phase. (For uniformity of presentation, we assume that the root too processes a wavefront in two (contiguous) steps, although these actions are obviously combinable into a single step.) In the interval of time between the two steps (one in the ascending phase, the other in the descending phase) performed by nodes

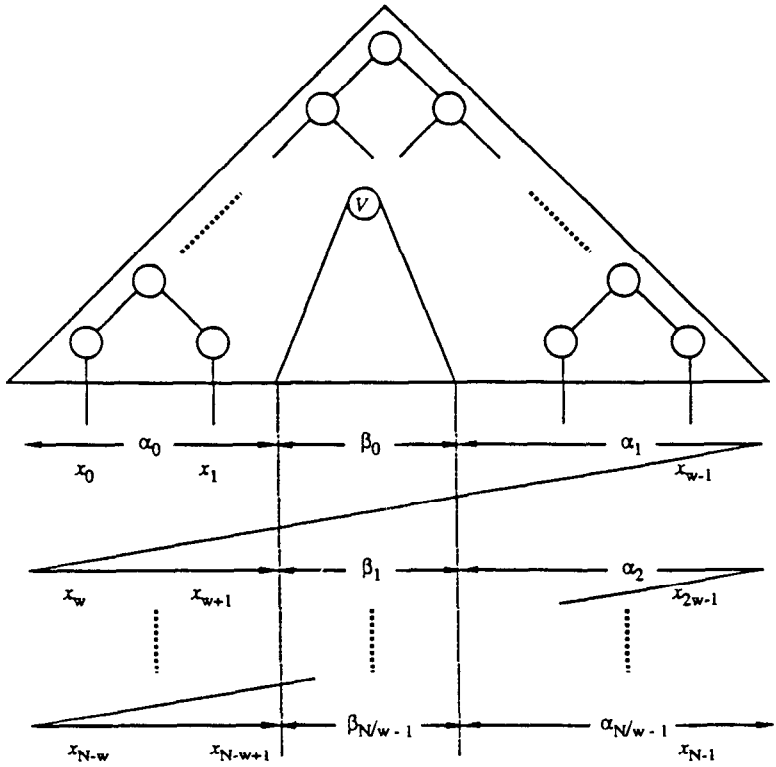


FIG. 1. Input protocol for prefix computation on a binary-tree network.

of a given level on the same wavefront, some information relative to that wavefront must be stored at the nodes.

A given internal node $V$ of $K$ determines a segmentation of the input sequence $\mathbf{x}$ as $\mathbf{x} = \alpha_0\beta_0\alpha_1\beta_1 \cdots \alpha_{N/w-1}\beta_{N/w-1}\alpha_{N/w}$ where $\beta_0\beta_1 \cdots$ is a subsequence of $\mathbf{x}$ that is input by the successive wavefronts to the leaves of the subtree rooted at $V$ ($\alpha_0$ and $\alpha_{N/w}$ may be empty). For a given sequence $\mathbf{x}$, let $x$ denote the product of its terms. Each $\beta_j$ is further segmented as $\beta_j = \beta_j'\beta_j''$, where $\beta_j'$ and $\beta_j''$ are input at the left and right subtrees of $V$, respectively. For convenience, we adjoin to $\langle A, \cdot \rangle$ an identity $e$, if $\langle A, \cdot \rangle$ is not a monoid. This is simply a technical device to initialize the states of the processing modules, and in no way affects the input/output transformation.

Referring to the $j$th wavefront, during the ascending phase internal node $V$ computes $\beta_j = \beta_j'\beta_j''$ from the values $\beta_j'$ and $\beta_j''$ received from its offsprings. In addition, the root (for which all the $\alpha$'s are empty) maintains a state $\sigma$ initialized to $e$ (the identity introduced above) and updated as $\sigma := \sigma\beta_j$. During the descending phase, nonroot node $V$ must receive from its parent the prefix $\gamma = \alpha_1\beta_1 \cdots \alpha_{j-1}\beta_{j-1}\alpha_j$; if $V$ has stored $\beta_j'$, then it can provide the correct prefixes $\gamma$ and $\gamma\beta_j'$ to its offsprings.

Below we describe in detail the actions of each node. We use a comma to separate concurrently executable actions, and a semicolon to separate actions to be sequentially executed. The *ascending phase* substep below is thought of as preceding the *descending phase* substep, although various degrees of concurrency are realizable. Note that, for correct synchronization, each internal

node $V$ uses a queue (called $\beta'$-queue) capable of storing $2\,level(V) + 1$ semigroup elements (the $\beta_j'$). In addition, each nonroot node has cells to store the elements to be forwarded in the next step; note that, for the root, one of these elements, $\beta_j$, is "forwarded" to the root itself. The content of all cells are initialized to the identity $e$. In summary, the generic step runs as follows:

Generic Step

```
ASCENDING PHASE
  begin forward β to parent, [root: σ := σβ]
    β' := term received from left child,
    β'' := term received from right child;
    insert β' into β'-queue;
    β := β'β'';
  end

DESCENDING PHASE
  begin forward γ to left child and γδ' to right child;
    γ := term received from parent; [root: γ := σ;]
    extract δ' from β'-queue;
    compute γδ';
  end
```

The algorithm is readily implemented by endowing the module of a node with a semigroup multiplier and a queue capable to store $(2\,level(V) + 1) = O(\log w)$ semigroup elements. Thus, the total size of the network is $S = O(w \log w)$ (ignoring the dependence upon the semigroup size and operation). Therefore we have:

THEOREM 6. *The size–time complexity of the prefix problem on a Boolean network is* $S = O((N/T)\log(N/T))$, *for* $T \in [\Omega(\log N), O(N)]$.

For friction semigroups the bound of Theorem 6 is optimal, as shown by Theorem 4. Since the recurrent subsemigroup of a friction semigroup is not a right-zero semigroup, by Theorem 5 we have $T = \Omega(\log N)$. For $T = \Theta(N)$, the obvious $S = \Omega(1)$ lower bound is achieved.

4.2 CYCLE-FREE SEMIGROUPS. As we have already noted in the concluding remarks of Section 3.4, the information content of a sequence of prefixes in a CF semigroup is only logarithmic in the length of the sequence. This fact indicates the possibility of reducing the amount of information relative to a given wavefront that the network must process.

We shall represent a sequence $\mathbf{y} = (y_0, y_1, \ldots, y_{m_p-1}) \in A^{m_p}$ of semigroup elements by the sequence of pairs $((a_1, m_1), (a_2, m_2), \ldots, (a_p, m_p))$ that is uniquely determined by the conditions: $a_i \neq a_{i+1}$, $(i = 1, \ldots, p-1)$; $y_j = a_1$, for $0 \leq j < m_1$ and $y_j = a_i$ for $m_{i-1} \leq j < m_i$, $(i = 2, \ldots, p)$. This representation is particularly efficient when $\mathbf{y}$ is a prefix sequence for a CF semigroup since $p \leq l_A$, where $l_A$ is the length of the longest simple path in the Cayley graph $G(A)$.

We then consider a recursive scheme for the prefix problem whereby:

(1) The input sequence (whose length $N$ is assumed to be a power of two) is split into a left and a right half of identical size.
(2) The prefix sequences of the two halves are recursively computed (in parallel), and
(3) The results are combined to form the prefix sequence of the entire input.

With the selected representation, let the two sequences computed by Step (2) be

$$\left((a_1, m_1), \ldots, \left(a_p, m_p = \frac{N}{2}\right)\right)$$

and

$$\left((b_1, n_1), \ldots, \left(b_q, n_q = \frac{N}{2}\right)\right),$$

where both $p$ and $q$ are no greater than $l_A$. The representation of the output sequence to be produced by Step (3) is obtained by first forming the sequence of pairs

$$((a_1, m_1), \ldots, (a_p, m_p), (a_p b_1, m_p + m_1), \ldots, (a_p b_q, m_p + m_q)),$$

and by subsequently deleting all but the last pair from any maximal run of pairs with identical first term. At the bottom of the recursion, an individual element $x \in A$ is represented as the pair $(x, 1)$.

The outlined scheme is naturally implemented by a tree-connected network $K$ whose generic node is designed to perform the concatenation of two prefix sequences. Such node will be equipped with a storage array of $2l_A$ cells, a semigroup multiplier, and some other simple logic. A cell of the array will be used to store a pair of type $(a_i, m_i)$. To determine the bit capacity of a cell, we note that:

(i) The term $a_i$ can always be encoded by $\lceil \log s \rceil$ bits, (where $s = |A|$).
(ii) For a node $V$ of $K$ with $w$ leaves, the term $m_i$ has value at most $w/2^{level(V)}$, so that $\log w - level(V) + 1$ bits are sufficient to represent it. Thus, the capacity of a cell at node $V$ is $\lceil \log s \rceil + \log w - level(V) + 1$.

The basic operation of node $V$ consists of loading into the first and last $l_A$ cells of the array the representations of the prefix sequences produced by the left and right children, respectively (some of the cells may actually be unused). The steps required to concatenate the two representations are fairly straightforward if the rows of the array are organized as shift registers. The update of the second term of each pair is done by appending a single bit, which is 0 for the pairs originating from the left child and 1 for those originating from the right child, since $m_p$ is a power of 2. Notice that, since $\lceil \log s \rceil + \log w - level(V)$ is the number of bits for the pairs of the children, the array at $V$ has the correct capacity.

We now describe how to obtain the ordinary form of the prefix sequence from its representation as a sequence of pairs $((a_1, m_1), \ldots, (a_p, m_p))$. Again, a recursive approach is appropriate. Indeed the first and the second half of $Y$ have respective representations

$$\left((a_1, m_1), \ldots, \left(a_i, \frac{m_p}{2}\right)\right)$$

and

$$((a_i', m_i'), \ldots, (a_p, m_p)),$$

where $i$ is such that $m_{i-1} < m_p/2 \leq m_i$ and $i' = i$ if $m_i \neq m_p/2$ and $i' = i + 1$ if $m_i = m_p/2$. (Note that $i$ can be determined in time $O(l_A)$.)

The conversion of representation is again performed by tree $K$, with each wavefront starting at the root and propagating toward the leaves. Pair $(y_j, j)$ is correctly delivered at leaf $j$.

The preceding description indicates that, referring to a single wavefront $\mathbf{x}$, tree $K$ is used in an ascending phase to compute the pair representation of the prefix sequence $\mathbf{y}$ of $\mathbf{x}$, and that a subsequent descending phase obtains the ordinary form of the latter. Note, however, that in marked contrast to the interaction between ascending and descending phases of the general algorithm of Section 4.1, the two phases are now completely decoupled. It is also simple to see that the operation of tree $K$ can be pipelined by just combining at the root the results of successive wavefronts.

Finally we note that $K$ consists of $\log w$ levels, and that the $2^j$ modules at level $j$ have each $l_A \times (\log w - j + \lceil \log s \rceil)$ bits of storage and fixed-size Boolean circuitry. Thus, the total size of $K$ is given by $O(\sum_{j=0}^{\log w - 1} 2^j(\log w - j)) = O(w)$. Computation is completed after $O(N/w)$ steps both for the input phase and the output phase, and again, the time used by each step is independent of $N$. The above construction yields:

THEOREM 7. *For a CF semigroup, the prefix computation for an N-term sequence can be done in time T and size S, with* $S = O(N/T)$, *for* $T \in [\Omega(\log N), O(N)]$.

*Remark.* The above result holds for any CF semigroup and is clearly optimal. However, for the very important case of insertion monoids, there exists an alternative method of the same flavor as the general technique outlined in Section 4.1. Each internal node $V$ of $K$ still contains a semigroup multiplier, and a queue with $2l_A$ cells, each capable of storing a semigroup element; an additional cell stores the node state $state(V)$.

Initially, for each $V$ in $K$, $state(V) := e$. Nonroot internal node $V$ performs the following actions:

Generic Step

```
ASCENDING PHASE
  begin forward β to parent,
    β' := term received from left child,
    β'' := term received from right child;
    if state(V)β ≠ state(V) then
      begin state(V) := state(V)β;
        insert (β', β'') into queue
      end;
    β := β'β''
  end

DESCENDING PHASE
  begin forward (γ_L, β_L) to left child and (γ_R, β_R) to right child;
    (γ, β) := term received from parent;
    (β', β'') := next pair in queue;
    if γβ', β'' = γβ
      begin (γ_L, β_L) := (γ, β'), (γ_R, β_R) := (γβ', β'');
        extract (β', β'') from queue
      end
    else (γ_L, β_L) := (γ_R, β_R) := (γ, e)
  end
```

LEMMA 2. *The above scheme correctly computes the prefix sequence for insertion monoids.*

PROOF. We say that $\beta_j$ is a *local* transition of $\beta_0\beta_1 \cdots \beta_{j-1}\beta_j \neq \beta_0\beta_1 \cdots \beta_{j-1}$, and a *global* transition if $\alpha_0\beta_0 \cdots \alpha_j\beta_j \neq \alpha_0\beta_0 \cdots \alpha_j$. Let $e$ denote the monoid identity.

CLAIM 1. *A global transition is also a local transition. Indeed* $\beta_0 \cdots \beta_{j-1}\beta_j = \beta_0 \cdots \beta_{j-1}$ *implies* $\beta_0\alpha_1 \cdots \beta_{j-1}\alpha_j\beta_j = \beta_0\alpha_1 \cdots \beta_{j-1}\alpha_j e\beta_j = \beta_0\alpha_1 \cdots \beta_{j-1}\alpha_j e = \beta_0\alpha_1 \cdots \beta_{j-1}\alpha_j$ *by the insertion property* (1), *and therefore* $\alpha_0\beta_0 \cdots \alpha_j\beta_j = \alpha_0\beta_0 \cdots \alpha_j$. *This means that the global transitions form a subset of the pairs of the form* $(\beta', \beta'')$ *inserted into the queues of the nodes of the network during the ascending phase.*

A local transition $\beta_j$ is not global because we may have $\alpha_0\beta_0 \cdots \alpha_j\beta_j = \alpha_0\beta_0 \cdots \beta_{j-1}\alpha_j$, even if $\beta_0 \cdots \beta_j \neq \beta_0 \cdots \beta_{0-1}$. Let $\gamma = \alpha_0\beta_0 \cdots \alpha_j$ ($\gamma$ is called the prefix associated with $\beta_j$).

CLAIM 2. *If* $\gamma_1\beta_j = \gamma_1$ *for a left factor* $\gamma_1$ *of* $\gamma$, *then* $\gamma\beta_j = \gamma$. *Indeed, let* $\gamma = \gamma_1\gamma_2$. *Then,* $\gamma_1\beta_j = \gamma_1$ *trivially implies* $\gamma_1 e\beta_j = \gamma_1 e$, *which in turn, by* (1), *implies* $\gamma_1\gamma_2 e\beta_j = \gamma_1\gamma_2 e$, *whence* $\gamma\beta_j = \gamma$.

Suppose now that $(\beta_j', \beta_j'')$ is a local transition at the head of the queue at $V$. When $V$ receives the pair $(\gamma, \beta)$, with $\beta = \beta_j$, we have $\gamma\beta = \gamma$ and $\gamma\beta_j'\beta_j'' = \gamma\beta$, that is, $(\beta_j', \beta_j'')$ is correctly eliminated from the queue and the pairs $(\gamma\beta_j')$ and $(\gamma\beta_j', \beta_j'')$ are passed on to the left and right children of $V$. This is correct, since $\gamma\beta_j' = \gamma$ and $\gamma\beta_j'\beta_j'' = \gamma$. Thus, $(\beta_j', \beta_j'')$ is certainly eliminated from the queue upon arrival of its associated prefix, but it may be eliminated sooner by a left factor $\gamma_1$ of it such that $\gamma_1\beta_j'\beta_j'' = \gamma_1$. This shows that a local transition is always eliminated before any subsequent global transition residing in the queue, which is correctly detected by the condition $(\gamma\beta'\beta'' = \gamma\beta)$ of the algorithm. In the majority of cases the action $(\gamma_L, \beta_L) = (\gamma_R, \beta_R) := (\gamma, e)$ occurs, corresponding to no transition, either local or global. □

We can therefore conclude:

THEOREM 8. *For an insertion monoid A, the prefix computation for an N-term sequence can be done in time T and size S with* $S = O(N/T)$ *for* $T \in [\Omega(\log N), O(N)]$. *The memory used by each nonroot module is* $O(sl_A \cdot \log s)$ *bits, where* $s = |A|$ *and* $l_A$ *is the height of the Cayley graph of A.*

PROOF. Indeed, the result $S = O(N/T)$ follows from Theorem 7. Each nonroot module has a queue of $2l_A$ cells, each with $\lceil \log s \rceil$ bits. □

4.3 NON-MEMORY-INDUCIVE SEMIGROUPS. As we noted earlier, for this class of semigroups we have

$$y_j = \begin{cases} x_0 \cdots x_j & \text{for } 0 \le j \le 2m_A - 1, \\ x_0 \cdots x_{m_A-1}x_{j-m_A+1} \cdots x_j & \text{for } 2m_A \le j \le N-1. \end{cases}$$

We assume, without loss of generality, that $m_A$ is a power of 2 (else, we replace $m_A$ with $2^{\lceil \log_2 m_A \rceil}$). Note, also, that for $j \ge 2m_A$ $y_j = x_0 \cdots x_{m_A-1}x_h \cdots x_j$ for any $m_A \le h \le j - m_A$.

As usual, the sequence $\mathbf{x}$ is supplied in $N/w$ wavefronts each of width $w$, a divisor of $N$ and therefore a power of 2. We assume, for simplicity, that $w \ge m_A$ and leave to the reader the simple task of developing the details for the opposite case.

The network consists of a binary tree $K$ of depth $\log w$. The lower $\log m_A$ levels realize a collection of $w/m_A$ prefix trees as those described in Section 4.1, whereas the upper $\log w - \log m_A$ levels realize a straightforward semigroup multiplier tree. The root of each of the lower prefix trees, except the rightmost one, has an arc to the root of the adjacent tree on the right.

The operation of this network is as follows: The first wavefront, $(x_0, \ldots, x_{w-1})$ is supplied in an isolated fashion and stored at the leaves, while in time $O(\log w)$ the tree computes the prefix $y_{m_A-1}$ and broadcasts it to all leaves, where it is stored (broadcast to the leaves with indices $0, \ldots, 2m_A - 1$ is in reality redundant).

Once this initial step is completed, the usual pipeline operation takes place, starting with the first wavefront that was temporarily stored earlier. The delay between the first wavefront and the subsequent ones is in compliance with the stated I/O protocol. During this phase only the lower trees are active, and each nonroot node operates as described in Section 4.1. A slight modification is in order for the operation of the root. Referring to the Generic Step algorithm of Section 4.1 (ASCENDING PHASE), $\beta$ is now passed the root of the tree adjacent to the right. The term simultaneously received from the root of the tree adjacent on the left is called $\beta^*$, and the root state is updated as $\sigma := \beta^*$.

Note that this implementation actually computes prefix $y_j$ as $x_0 \cdots x_{m_A-1} x_{(i-1)m_A} \cdots x_{im_A-1} x_{im_A} \cdots x_j$ where $i = \lfloor j/m_A \rfloor$, for $j \geq m_A$. This is obviously correct since $(i-1)m_A < j - m_A + 1$.

It is easily realized that $K$ consist of $O(w)$ modules, each independent of $N$, and that computation is completed in $2 \log w + 2 \log m_A + N/w + 4$ steps, that is, in time $T = O(N/w)$. Thus we have:

THEOREM 9. *For an NMI semigroup $\langle A, \cdot \rangle$, the prefix computation for an $N$-term sequence can be done in time $T$ and size $S$, with $S = O(N/T)$, for $T \in [\Omega(\log N), O(N)]$.*

Note that if $recur\langle A, \cdot \rangle$ is a right-zero semigroup, then for $j \geq m_A$, $y_j = x_{j-m_A+1} \cdots x_j$. This shows that the broadcast of $y_{m_A-1}$ is no longer necessary and that the network becomes a collection of $w$ combinational machines each computing $a$ product of at most $m_A$ terms. In this case, the time range in Theorem 9 can be extended as $T \in [\Omega(1), O(N)]$.

## 5. *Area–Time Complexity*

The results of the preceding sections have some consequences on the *area–time* complexity of synchronous VLSI circuits for the prefix problem. These consequences are summarized below by Theorem 10 for *nonboundary* circuits (those for which the position of the I/O ports is unconstrained), and by Theorem 11 for *boundary* circuits (those for which I/O ports are constrained to lie on the boundary of a convex chip).

THEOREM 10. *The area–time complexity of nonboundary VLSI circuits for the prefix problem is $A = \Theta((N/T)\log(N/T))$, for MI-NCF semigroups, and is $A = \Theta(N/T)$, for all other semigroups. The bounds hold for $T \in [\Omega(\log N), O(N)]$ for all semigroups, with the exception of those whose recurrent subsemigroup is a right-zero semigroup, for which $T \in [\Omega(1), O(N)]$.*

PROOF. A VLSI circuit is the layout of a Boolean network. Since our prefix networks are balanced trees, they can be laid out (according to the H-scheme [6]) in area proportional to the number of nodes times the area of a node, that is, in area proportional to their size. Thus, Theorem 10 follows from Theorem 1, and from the fact that size is a lower bound for area. □

The fact that our prefix networks admit of a nonboundary layout whose area is proportional to the size is accidental. In general, layout area is larger than size, owing to the space occupied by wires.

We now consider boundary layouts. We begin by establishing the following lemma:

LEMMA 3. *Let $\langle A, \cdot \rangle$ be such that recur($A$) is not a right-zero semigroup. Any boundary circuit $C$ that solves the prefix problem of size $N$ for $A$ in time $T$ (where $T = \Omega(\log N)$) contains a binary tree of height $O(T)$ with $\Omega(N/T)$ leaves on the boundary.*

PROOF. In order to generate $N$ output symbols in time $T$, circuit $C$ must contain $w \geq (N/T)$ output ports. Since each output $y_j$, $0 \leq j \leq N-1$, is functionally dependent upon input $x_0$ (see proof of Theorem 5), there must be a path in $C$ from the input port reading $x_0$ to each output port. The union of the $w$ paths can be viewed as an undirected graph $G$, whose edges are wires each carrying a semigroup symbol. Since each output port is connected to the exterior by one such wire, we may correctly assume that the output port is itself on the boundary of the layout (by transplanting it on the boundary if necessary). Moreover, we may assume that each output port is a degree-1 node of $G$; otherwise, we create one such node on the layout boundary and connect it to the port, with an insignificant increase in wire area. Let $H$ be a minimum-height spanning tree of $G$ rooted at the input port of $x_0$. Owing to the hypothesis of bounded fan-in and fan-out, we may assume without loss of generality that $H$ is a rooted binary tree. Tree $H$ has $w$, that is, $\Omega(N/T)$, leaves. Since $T$ is the computation time of $C$, the height of $H$ is $O(T)$. □

Lemma 3 establishes a crucial connection between binary trees and prefix circuits. Our problem is then reduced to the boundary layout of binary trees, a topic that has been adequately investigated [6, 12, 24]. The results most relevant to our discussion are expressed by the following three lemmas:

LEMMA 4 [6]. *A complete binary tree with $w$ leaves can be laid out with the leaves on the boundary of a rectangular region of height $O(\log w)$ and width $O(w)$. The resulting $O(w \log w)$ area is optimal.*

The constructive proof of this lemma yields a layout of a complete binary tree with $w$ leaves in a rectangular region consisting of $2w-1$ vertical tracks (one per node) and $\log w + 1$ horizontal tracks (one per level of nodes in the tree). More specifically, nodes appear from left to right according to the in-order traversal of the tree, and consecutive levels appear in consecutive horizontal tracks, with the root at the top, and the leaves at the bottom.

LEMMA 5 [24]. *Any layout of a binary tree of height $h$ with $w$ leaves placed on the boundary has area $A = \Omega((w \log w)/\log(2h/\log w))$.*

LEMMA 6 [12]. *There exist binary trees with $w$ leaves, height $h$, and boundary layout area $A = O((w \log w)/\log(2h/\log w))$, which can emulate a balanced binary tree of $w$ leaves in time $O(h)$.*

With these premises, we can now give tight area–time bounds for boundary prefix circuits.

THEOREM 11. *For boundary VLSI circuits for the prefix problem we have the following area–time complexity results.*

(1) *MI-NCF Semigroups*: $A = \Theta((N/T)\log(N/T))$, *with* $T \in [\Omega(\log N), O(N)]$;
(2) *CF Semigroups*: $A = \Theta((N/T)\log(N/T))$, *and* $A = \Omega((N/T \log(N/T))/\log(T/\log N))$, *with* $T \in [\Omega(\log N), O(N)]$;

(3) *CF-Insertion and NMI Semigroups, Except for Those with Right-Zero Recurrent Subsemigroup*: $A = \Theta((N/T \log(N/T))/\log(T/\log N))$, *with* $T \in [\Omega(\log N), O(N)]$;

(4) *Semigroups with Right-Zero Recurrent Subsemigroup*: $A = \Theta(N/T)$, *with* $T \in [\Omega(1), O(N)]$.

PROOF. First of all we note that the bounds on computation time are not affected by layout issues, and hence are as in Theorem 1. As regards the wire area, we distinguish the following four cases:

(1) *MI-NCF Semigroups.* These semigroups are processed by the networks of Section 4.1, which are complete binary trees with $w = \Theta(N/T)$ leaves, where each node is equipped with a queue of size $O(\log w)$ and logic circuitry of constant size. These networks can be laid out in area $A = O(w \log w)$ according to the scheme provided by Lemma 4. The queue of each node can be easily accommodated in the vertical track of that node. In conclusion, we obtain $A = O((N/T)\log(N/T))$. The area is of the same order as the optimal size, and is therefore optimal.

(2) *CF Semigroups.* The upper bound for this case is obtained, as in case (1), by the network of Section 4.1. It turns out that the obvious layout for the network given in Section 4.2 for CF semigroups (a balanced binary tree with edges of bandwidth proportional to their height) would result in a larger area, in spite of a smaller size. The boundary layout of this network needs further investigation. The lower bound follows from Lemmas 3 and 5, with $w = \Omega(N/T)$ and $h = O(T)$.

(3) *CF-Insertion and NMI Semigroups, Except for Those with Right-Zero Recurrent Subsemigroup.* These semigroups (see Remark at the end of Section 4.2 and Section 4.3) are processed by complete binary trees with constant-size nodes and $w = \Theta(N/T)$ leaves. The upper bounds stated in the theorem can be achieved by emulating the complete binary trees of our prefix networks with the trees of Lemma 6 with $h = \Theta(T)$. The lower bound follows from Lemmas 3 and 5, with $w = \Omega(N/T)$ and $h = O(T)$.

(4) *Semigroups with Right-Zero Recurrent Subsemigroup.* As observed in Section 4.3, the prefix networks for these semigroups consist of $w = O(N/T)$ combinational circuits of constant size. A simple layout having the shape of an array of constant height and $O(w)$ width yields a boundary circuit of area $A = O(N/T)$, which is trivially optimal. □

## 6. *Remarks and Open Problems*

In this paper, we have considered the computation of the prefixes of an *N-term* sequence of semigroup elements on Boolean networks. We have completely characterized the size–time and the area–time complexity of the networks as a function of $N$, under some assumptions on the network's I/O protocol.

A problem intimately related to the computation of prefixes is the computation of suffixes $z_j = x_j x_{j+1} \cdots x_{N-1}$, for $j = 0, 1, \ldots, N-1$. It is readily observed that the suffixes for a semigroup $\langle A, \cdot \rangle$ are the prefixes of its *reverse* semigroup $\langle A, * \rangle$, where $x * y = y \cdot x$. It must be noted, however, that a semigroup and its reverse are not necessarily in the same class, as characterized in this paper. A notable case in point is the semigroup of the *carry* function in binary addition mentioned in Section 2, which is a friction semigroup, where its reverse semigroup (a left-zero two-element semigroup with adjoined identity) is frictionless because it is CF. The latter is easily seen to model the *comparison-exchange* operation of two binary

integers. Nevertheless, the operation of comparison-exchange exhibits computational friction for reasons unrelated to the underlying prefix computation (see [2]).

It would be interesting to investigate to which extent the results of this paper depend on the I/O protocol assumptions.

In this direction, it is appropriate to observe that, for some semigroups, non-word-instantaneous circuits are more efficient than word-instantaneous ones. A simple example is provided by the direct product of the OR semigroup $\langle \{0, 1\}, \text{OR} \rangle$ (which is CF) with the right-zero semigroup of two elements $R_2$ (which is NMI). It is straightforward to verify that this direct product is a friction semigroup. Now let $C_1$ be the network of Section 4.2 specialized for the OR semigroup, and let $C_2$ be the network of Section 4.3 specialized for $R_2$. Both $C_1$ and $C_2$ have size $S = O(N/T)$. Taken as a pair, they clearly compute the direct product of their respective semigroups with size $S = O(N/T)$, seemingly contradicting Theorem 4. A closer look at $C_1$ and $C_2$ reveals the delay between the input time of $x_i$ and the output time of $y_i$ is constant in the latter and is logarithmic in the former. There is simply no way to combine the two networks so that the protocol of the result is word-instantaneous both in the input and in the output. Therefore, Theorem 4 does not apply to the pair $(C_1, C_2)$, and there is no contradiction.

The major outstanding problem is the investigation of the dependence of network complexity upon semigroup size and operation. For example, in Theorem 8 we have shown that, for the important case of insertion semigroups, the upper bounds of Theorem 7 can be considerably improved. However, the construction of prefix Boolean networks that are optimal also with reference to semigroup size remains an open problem.

## *Appendix. Proof of Theorem 3*

*Sufficiency.* (If $\langle recur(A), \cdot \rangle$ is isomorphic to $Z_{p,q}$, then $\langle A, \cdot \rangle$ is NMI.) We begin by observing that every $b \in A$ has a recurrent power $b^i$. In fact, each nonrecurrent element may appear at most once in the sequence $b, b^2, b^3, \ldots$.

Next, we claim that if $a \in recur(A)$, then $(ab) \in recur(A)$, for any $b \in A$. Let $i$ be such that $b^i \in recur(A)$. By the isomorphism between $recur(A)$ and $Z_{p,q}$, $ab^i$ and $a$ belong to the same strongly connected component of $G(A)$. Therefore, for some $z$, $ab^i z = a$, whence $(ab)(b^{i-1}zb) = (ab)$, which implies that $(ab)$ is recurrent.

Each nonrecurrent element may appear at most once in any sequence of prefixes and, by the preceding claim, if prefix $y_i$ is recurrent then $y_j$ will also be recurrent for every $j > i$. We conclude that there is a (minimum) integer $m_A$ such that $(x_0 x_1 \cdots x_{m_A-1}) \in recur(A)$ for any choice of $x_0, x_1, \ldots, x_{m_A-1}$. It follows that for $j \geq 2m_A - 1$ both $(x_0 \cdots x_{m_A-1})$ and $(x_{j-m_A+1} \cdots x_j)$ belong to $recur(A)$. Since $recur(A)$ is isomorphic to some $Z_{p,q}$, by Definition 4, $y_j = x_0 \cdots x_{m_A-1} x_{j-m_A+1} \cdots x_j$ for any $j \geq 2m_A - 1$. This shows that for $j > 2m_A - 1$, $y_j$ does not functionally depend upon $x_{m_A}, x_{m_A+1}, \ldots, x_{j-m_A}$, and therefore $\langle A, \cdot \rangle$ is NMI.

*Necessity.* (If $\langle A, \cdot \rangle$ is NMI, then $\langle recur(A), \cdot \rangle$ is isomorphic to $Z_{p,q}$.) This part of the proof is based on a number of claims stated and proved below. Let $recur(A)$ be the union of strongly connected components $C_1, C_2, \ldots, C_p$.

CLAIM A1. *The components $C_1, C_2, \ldots, C_p$ of $recur(A)$ have no outgoing arcs in $G(A)$, that is, if $a \in C_r$, then $(ac) \in C_r$ for all $c \in A$.*

PROOF OF CLAIM A1. Assume, for a contradiction, that $a \in C_r$ and that, for some $c \in A$, $ac \in C_s$ with $s \neq r$. Since $a \in recur(A)$, there is a $b \in A$ such

that $ab = a$. We now show that $y_j$ functionally depends upon $x_i$ for $i = 0, \ldots, j$, that is, $\langle A, \cdot \rangle$ is MI. To see that $y_j$ depends upon $x_0$, consider the products $ab^j$ and $(ac)b^j$ that differ only for the $x_0$-factor. Clearly, $ab^j = a$. On the other hand, since $ac \in C_s$, then $(ab)b^j \notin C_r$, otherwise $C_r$ and $C_s$ would not be distinct strongly connected components of $recur(A)$; thus $(ac)b^j \neq a$. To see that $y_j$ depends on $x_i$ for $1 \leq i \leq j$, consider the products $ab^{i-1}bb^{j-i} = ab^j = a$ and $ab^{i-1}cb^{j-i} = (ac)b^{j-1} \notin C_r$ which differ only in the $x_i$-factor and have different values. We have a contradiction, which establishes Claim A1. □

CLAIM A2. *If $\langle A, \cdot \rangle$ is NMI, then each element of $A$ has period* 1.

PROOF OF CLAIM A2. Assume, for a contradiction, that there is an element $a$ with period $\geq 2$. Then $\langle A, \cdot \rangle$ has a subsemigroup $A'$ with an identity $e$ and at least another element, say $a$. Then, for every $j \geq 0$, and $0 \leq i \leq j$, $e = e^{j+1} = e^i e e^{j-i} \neq e^i a e^{j-i} = a$. Thus $\langle A, \cdot \rangle$ is MI, a contradiction. □

Below, we make use of the following notation: If $C \subseteq A$ and $a \in A$ are such that, for $u \in C$, $ua$ is independent of $u$, then $ua$ is denoted by $Ca$.

CLAIM A3. *For $r = 1, 2, \ldots, p$, if $a$ and $b$ are in $C_r$, then $ba = a$. Thus, each $C_r$ is a right-zero subsemigroup, and for $a \in C_r$, we can write $C_r a = a$.*

PROOF OF CLAIM A3. Let $u \in C_r$. Since $\langle A, \cdot \rangle$ is not MI, by Claim A2, $u$ has period 1 and hence, for some $k$, $u^{k+1} = u^k$. Let $a = u^k$. Clearly, $a \in C_r$, and $a^2 = a$. Thus, if $C_r = \{a\}$, then Claim A3 is established. Assume then that $C_r$ contains also $b$, with $b \neq a$. By definition of strongly connected component, there is a $c \in A$ such that $ac = b$. We claim that $ba = a$. We prove the claim by contradiction by showing that if $ba \neq a$, then for $j \geq 0$ and $0 \leq i \leq j$, $y_j$ functionally depends upon $x_i$, and therefore $\langle A, \cdot \rangle$ is MI.

We consider four cases. For $j = 0$, $y_0 = x_0$ obviously depends upon $x_0$. For $j = 1$, $y_1 = x_0 x_1$ depends upon both factors as shown by the inequalities $ba \neq aa$ and $aa \neq ac$. For $j \geq 2$ and $i < j$, the dependence is established considering the products $a = a^{j+1} = a^i a a^{j-i}$, and $a^i c a^{j-i} = aca = ba$. For $i = j$, the dependence follows from the fact that $a^j = a^{j-1}a \neq a^{j-1}c$.

Summarizing, for every $b \in C_r$, $ba = a$, that is, $C_r a = a$. Moreover, $b^2 = b(ac) = (ba)c = ac = b$, so that all the elements of $C_r$ are idempotent. Since the argument to show that $C_r a = a$ is based solely on the idempotence of $a$, it can be extended to all elements of $C_r$, yielding the desired conclusion. □

CLAIM A4. *If $a \in recur(A)$ and $b \in C_s$, then $ba$ is independent of $b$, and can be written as $C_s a$.*

PROOF OF CLAIM A4. For a contradiction, let $b_1$ and $b_2$ be elements of $C_s$ such that $b_1 a \neq b_2 a$. Since $a \in recur(A)$, then $a^p = a$ for any $p \geq 1$. Then, for $j \geq 2$ and $i < j$, $b_1 a = b_1^i b_1 a^{j-1} \neq b_1^i b_2 a^{j-1} = b_2 a$, so that $y_j$ depends upon $x_i$, and the semigroup is MI. □

CLAIM A5. *Let $a \in C_r$ and $b \in C_s$, with $r \neq s$. If $C_s a = b$, then $C_r b = a$, and for no other $c \in C_r$ we have $C_s c = b$.*

PROOF OF CLAIM A5. Since, by hypothesis, $C_s a = b$, then in particular $ba = b$. Then $C_r b = C_r(ba) = (C_r b)a$. But $C_r b \in C_r$, and, by Claim A3, $(C_r b)a = a$. This establishes $C_r b = a$. Assume now, for a contradiction, that for some $c \in C_r$, $c \neq a$, $C_s c = b$. This implies $C_r b = c$, a contradiction. □

Claim A5 shows that, for any $r$ and $s$ in $\{1, \ldots, p\}$, there is a bijection between $C_r$ and $C_s$. Thus, $C_1, C_2, \ldots, C_p$ have identical cardinality $q$. If we let $C_1 = \{a_1, a_2, \ldots, a_q\}$, then an arbitrary element of $C_h$ can be written in a unique way as $C_h a_k$. Therefore $recur(A) = \{C_h a_k : 1 \leq h \leq p, 1 \leq k \leq q\}$.

To establish that $recur(A)$ is isomorphic to $Z_{p,q}$, it remains to show that $(C_h a_k)(C_{h'} a_{k'}) = C_{h a_{k'}}$. Indeed, $(C_h a_k)(C_{h'} a_{k'}) = C_h(a_k C_{h'})a_{k'}$. Since $a_k \in C_1$, by Claim A1, $a_k C_{h'} \in C_1$, whence, by Claim A3, $(a_k C_{h'})a_{k'} = a_{k'}$. □

ACKNOWLEDGMENTS. We are indebted to D. E. Muller and to N. Pippenger for their valuable suggestions. We are also grateful to the referees whose constructive criticisms have led to considerable improvements over an earlier version of the manuscript.

REFERENCES

1. BAUDET, G. M. On the area required by VLSI circuits. In *VLSI Systems and Computations*, H. T. Kung, R. Sproull, and G. Steele, Eds. Computer Science Press, Rockville, Md., 1981, pp. 100–107.
2. BILARDI, G. The area–time complexity of sorting. Ph.D. dissertation TR-R-1024, Univ. Illinois at Urbana-Champaign, Urbana, Ill., Dec. 1984.
3. BILARDI, G., AND PREPARATA, F. P. The influence of key length on the area–time complexity of sorting. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, W. Brauer, Ed. (Nafplion, Greece, July). Springer-Verlag, New York, 1985, pp. 53–62.
4. BILARDI, G., AND PREPARATA, F. P. Area–time lower-bound techniques with applications to sorting. *Algorithmica 1*, 1 (1986), 65–91.
5. BILARDI, G., AND PREPARATA, F. P. Digital filtering in VLSI. In *Proceedings of Aegean Workshop on Computing*. F. Makedon, K. Mehlhorn, T. Papathedorou, and P. Spirakis, Eds. (Loutraki, Greece, July). Springer-Verlag, New York, 1986, pp. 1–11.
6. BRENT, R. P., AND KUNG, H. T. On the area of binary tree layouts. *Inf. Proc. Lett. 11*, 1 (Aug. 1980), 46–48.
7. BRENT, R. P., AND KUNG, H. T. A regular layout for parallel adders. *IEEE Trans. Comput. C-31*, 3 (Mar. 1982), 260–264.
8. CHANDRA, A. K., FORTUNE, S., AND LIPTON, R. Unbounded fan-in circuits and associative functions. In *Proceedings of the 15th Annual Symposium on Theory of Computing* (Boston, Mass., Apr. 25–27). ACM, New York, 1983, pp. 52–60.
9. CHANDRA, A. K., FORTUNE, S., AND LIPTON, R. Lower bounds for constant depth circuits for prefix problems. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*. Springer-Verlag, New York, 1983, pp. 109–117.
10. CLIFFORD, A. H., AND PRESTON, G. B. *The Algebraic Theory of Semigroups*, vol. 1. American Mathematical Society, Providence, R.I., 1961.
11. COLE, R., AND SIEGEL, A. On information flow and sorting: New upper and lower bounds for VLSI circuits. In *Proceedings of the 26th Annual Symposium on the Foundations of Computer Science* (Portland, Ore., Oct.). IEEE, New York, 1985, pp. 208–221.
12. COLE, R., AND SIEGEL, A. Optimal VLSI circuits for sorting. *J. ACM 35*, 4 (Oct. 1988), 777–810.
13. DEKEL, E., AND SAHNI, S. Binary trees and parallel scheduling algorithms. *IEEE Trans. Comput. C-32*, 3 (Mar. 1982), 307–315.
14. DYMOND, P. W., AND COOK, S. A. Hardware complexity and parallel computation. In *Proceedings of the 21st Annual Symposium on the Foundations of Computer Science* (Syracuse, N.Y., Oct.). IEEE, New York, 1980, pp. 360–372.
15. FITCH, F. E. New bounds for parallel prefix circuits. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing* (Boston, Mass., Apr. 25–27). ACM, New York, 1983, pp. 100–109.
16. JOHNSON, R. B. The complexity of a VLSI adder. *Infor. Process. Lett. 11*, 2 (Oct. 1980), 92–93.
17. KRUSKAL, C. P., RUDOLPH, L., AND SNIR, M. The power of parallel prefix. *IEEE Trans. Comput. C-34*, 10 (Oct. 1985), 965–968.
18. KUCK, D. J. *The Structure of Computers and Computations*. Wiley, New York, 1978.
19. LADNER, R. E., AND FISCHER, M. J. Parallel prefix computation. *J. ACM 27*, 4 (Oct. 1980), 831–838.

20. MCNAUGHTON, R., AND PAPERT, S. *Counter-Free Automata.* M.I.T. Press, Cambridge, Mass., 1971.
21. OFMAN, YU. On the algorithmic complexity of discrete functions. *Sov. Phys. Dokl. 7*, 7 (Jan. 1963), 589–591.
22. SNIR, M. Depth-size trade-offs for parallel prefix computations. *J. Algorithms 7*, 2 (June 1986), 185–201.
23. THOMPSON, C. D. A complexity theory for VLSI. Ph.D. dissertation, Dept. Comput. Sci., Carnegie-Mellon Univ., Aug. 1980.
24. YAO, A. C. The entropic limitations on VLSI computations. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing* (Milwaukee, Wis., May 11–13). ACM, New York, 1981, pp. 308–311.

RECEIVED JANUARY 1987; REVISED OCTOBER 1987 AND SEPTEMBER 1988; ACCEPTED SEPTEMBER 1988