Size-Time Complexity of Boolean Networks for Prefix Computations

Gianfranco Bilardi[1]
Franco P. Preparata[2]

TR 87-805

January 1987

Department of Computer Science
Cornell University
Ithaca, NY 14853

1 Department of Computer Science, Cornell University, Ithaca, NY 14853. Research supported in part by NSF Grant DCI-8602256.

2 Coordinated Science Laboratory, Departments of Electrical and Computer Engineering and of Computer Science, University of Illinois at Urbana-Champaign, IL 61801. Research supported in part by NSF Grant ECS-84-10902.

SIZE-TIME COMPLEXITY OF BOOLEAN NETWORKS FOR PREFIX COMPUTATIONS

G. Bilardi[1] and F.P. Preparata[2]

ABSTRACT

The *prefix problem* consists of computing all the products $x_0x_1...x_j$ $(j=0,...,N-1)$, given a sequence $\mathbf{x} = (x_0,x_1,...,x_{N-1})$ 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 a property of the underlying semigroup, which we call *cycle-freedom* (no cycle of length greater than one in the Cayley graph of the semigroup). Denoting by S and T size and computation time, respectively, we have $S = \Theta((N/T)\ \log(N/T))$, for non-cycle-free semigroups, and $S = \Theta(N/T)$, for cycle-free semigroups. In both cases, $T \in [\Omega(\log N), O(N)]$.

1. Introduction

The *prefix problem* consists of computing all the products $x_0x_1...x_j$ $(j=0,...,N-1)$, given a sequence $\mathbf{x} = (x_0,x_1,...,x_{N-1})$ of elements in a semigroup. Prefix computations occur in the solution of several significant problems such as carry-look-ahead addition [BK82], the evolution of finite-state machines [LF80], linear recurrences [K78], digital filtering [BP86b], various graph problems [KRS85], sorting in bit-models of computation [CS85, BP85], and others.

The prefix problem has been extensively investigated in the *boolean-circuit* model, where the computation is carried out by an acyclic network of *combinational gates*. Various complexity measures such as *size*, *depth*, *width*, and their trade-offs have been studied in this context [LF80, F83, CFL83, S86]. Algorithms for the EREW-PRAM model have been proposed in [KRS85].

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 [DC80], 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

[1] Department of Computer Science, Cornell University, Ithaca, NY 14853. Research supported in part by NSF Grant DCI-8602256.

[2] Coordinated Science Laboratory, Departments of Electrical and Computer Engineering and of Computer Science, University of Illinois at Urbana-Champaign, IL 61801, USA. Research supported in part by NSF Grant ECS-84-10902.

input size. Results on boolean networks have also interesting implications for other models of parallel computation such as fixed interconnections of processors and VLSI circuits [T80].

We have found that the size-time complexity of the prefix problem is determined by a property of the underlying semigroup, which we call *cycle-freedom*. We call a semigroup cycle-free if its Cayley graph has no cycle of length greater than one and non-cycle-free otherwise. Our results, which completely characterize the size-time complexity of the prefix problem, are encompassed by the following theorem, which summarizes Theorems 3,4, and 5.

Theorem 1. The size-time complexity of the prefix problem on a boolean network is $S = \Theta((N/T)\log(N/T))$, for non-cycle-free semigroups, and is $S = \Theta(N/T)$, for cycle-free semigroups. In both cases, $T \in [\Omega(\log N), O(N)]$.

For non-cycle-free semigroups, the upper bound can be achieved by known constructions based on binary-tree networks, or twisted-reflected-tree networks [LF80, BK82, BP86b], whereas the lower bound (Section 3, Theorem 3) is less obvious, and is based on arguments of computational friction [BP86a]. For cycle-free semigroups, the lower bound is based on a trivial input/output argument, while the upper bound is achieved by a rather sophisticated algorithm (Section 4, Theorem 5) executed by a tree-connected network.

It may be interesting to contrast Theorem 1 with the result of [CFL85] 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 stronger than cycle free.

A result completely anologous to Theorem 1 could be stated for the *area-time* complexity of prefix computation in the VLSI model [T80]. Indeed, since a VLSI circuit is the layout of a boolean network, size-time lower bounds for the latter immediately translate into area-time lower bounds for the former. In general, area is larger than size due to space occupied by wires. However, the circuits considered in this paper can be laid out so that the total wire area is of the same order as the size, hence they are area-time optimal as well as size-time optimal.

2. Definitions and Problem Statement

A *finite semigroup* is a pair $<A,\cdot>$ where $A = \{a_1,a_2,...,a_s\}$ 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.

For a sequence $\mathbf{x} = (x_0,x_1,...,x_{N-1}) \in A^N$, the sequence of *prefixes* of $\mathbf{x}$ is defined as $\mathbf{y} = (y_0,y_1,...,y_{N-1})$, with $y_j = x_0x_1 \ldots x_j$. 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), labelled 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 always form a subsemigroup of A, and that $G(A)$ is transitively closed.

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 avoid the term "cyclic" here, since it has a different established meaning in group theory). We shall see that cycle-freedom is the crucial property of a semigroup in determining the complexity of the prefix problem. Among CF semigroups, of particular interest are *insertion* semigroups, characterized by the following property: for all $x,y,z,w \in A$,

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

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.

All abelian CF semigroups are also insertion semigroups. An instance of abelian CF semigroup is given by the set $A = \{0,1,...,s-1\}$ 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 $\mathbf{x}$ 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 a abelian is the set of the rankings of n items with respect to the operation of rank concatenation, which plays an important role in VLSI sorting [ČS85, BP85, BP86a]. 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 $(u_j \cap v_1, u_j \cap v_2,\ldots,u_j \cap v_q)$.

Yet another class of semigroups is that of *strongly cycle-free* (SCF) semigroups defined by the property that, for all $x \in A$, the set of solutions y of the equation $xy = x$ is either empty or is A itself. In the latter case, x is a left zero of A. The prefix problem for SCF semigroups degenerates, in the sense that arbitrarily long input sequences are not of interest. Indeed, for j greater than the length of the longest simple path of $G(A)$, output y_j is guaranteed to be constant with j (and equal to some left zero of A).

To exclude this uninteresting case, and without any substantial loss of generality, all semigroups in this paper are assumed to be monoids.

3. Lower bounds

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 labelled 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 [DC80]. Here we appeal to the intuitive meaning of these notions, and just discuss the input/output protocol, since it slightly differs from that of [DC80]. 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 (or 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 [Th80].

The following result is a simple consequence of the bounded fan-in assumption and of the fact that, in a semigroup which is not SCF, $y_j = x_0 x_1 ... x_j$ is a true function of $x_0, x_1, ..., x_j$:

Proposition 1. For any boolean network that solves the prefix problem of size N for a non-SCF semigroup, the computation time satisfies the bound $T = \Omega(\log N)$.

Our lower bound for the prefix problem is based on the mechanism of *computational friction* developed in [BP86a] as a generalization of arguments previously applied to binary addition in [J80] and [B81]. 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. Two phenomena contribute to the appearance of friction: (i) A fixed 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 wavefront size since, for bounded fan-in, functional dependence imposes a delay between reading the inputs and computing the outputs. These phenomena can be precisely analyzed and lead to the quantitative bounds embodied by the following theorem, a more general version of which is proved in [BP86a].

Theorem 2. Given a computational problem P with a set X of input variables and a set Y of output variables, let U be a subset of X such that for any *partition* $U_1,...,U_T$ of U there exists a collection $W_1,...,W_T$ of *disjoint subsets* of Y (not necessarily a partition) satisfying the following properties:

(1) Each variable in W_t is functionally dependent upon $\Omega(|U_t|)$ variables of U_t.

(2) The values of the variables in $X-U$ can be selected so that, for each $t = 1,2,...,T$ the variables of W_t carry $\Omega(|U_t|)$ bits of information about U_t.

Then for any word-istantaneous boolean network that solves P, size and time satisfy the bound

$$S = \Omega((|U|/T)\ \log(|U|/T)). \tag{2}$$

We now have the tools to prove the following result.

Theorem 3. For any word-istantaneous boolean network that solves the prefix problem of size N for a NCF semigroup A, size and time satisfy the bound

$$S = \Omega((N/T)\ \log(N/T)). \tag{3}$$

Proof. We show that Theorem 2 can be applied, with $X = \{x_0,x_1,...,x_{N-1}\}$, $Y = \{y_0,y_1,...,y_{N-1}\}$, and $U = \{x_1,x_3,...,x_{2i+1},...\}$ containing all odd-indexed input variables. Given a partition $U_1,...,U_T$ of U, let us define the disjoint subsets of Y

$$W_t \stackrel{\Delta}{=} \{y_j : j \text{ is among the } \lceil |U_t|/2 \rceil \text{ largest } k\text{'s such that } u_k \in U_t\}.$$

(1) Clearly each y_j in W_t is functionally dependent upon all the x_i's with $i \leq j$, and there are at least $\lceil |U_t|/2 \rceil = \Omega(|U_t|)$ of them in U_t.

(2) By assumption, the Cayley graph of A has a cycle of length ≥ 2, and hence a cycle of length exactly two, 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 $x_0 = a$, $x_i \in \{c,d,cd\}$ for $i \geq 1$, and satisfying the following two properties: (i) If $y_{i-1} = b$, then $x_i = d$;

(ii) If $y_{i-1} = a$ and i is even, then $x_i = cd$. These properties guarantee that the even-indexed outputs are all equal to a, and the odd-indexed outputs can be either a (if the corresponding input is cd) or b (if the corresponding input is c). In conclusion, each of the $\lceil |U_i|/2 \rceil$ variables of W_i carries one bit of information about U_i. Then Equation (3) follows from Equation (2), considering that $|U| = \Theta(N)$. $\square$

The previous argument cannot be extended to cover CF semigroups; in fact, in the next section we shall describe networks for prefix computation in CF semigroups that violate bound (3). Indeed, the low information content of the output suggests the absence of friction. More specifically, let A be CF. A sequence $\mathbf{y} = (y_0,...,y_{N-1})$ of prefixes corresponds to a generally non-simple path of length N in G(A). Let f_A be the minimum number of bits required to describe a simple path in G(A), and let l_A be the number of arcs of the longest such path (the height of G(A)). Then, an arbitrary path of length N can be described with $O(f_A + l_A \log N)$ bits, $O(f_A)$ for the underlying simple path and $O(\log N)$ to specify the length of each of the $O(l_A)$ runs of self-loops. Therefore, $O(\log N)$ bits are sufficient to describe the output prefixes on a CF semigroup, whereas $\Omega(N)$ are necessary for NCF semigroups.

4. Upper Bounds

In this section, we present three algorithms for the prefix problem, which are respectively designed for general semigroups, for CF semigroups, and for insertion semigroups. We first describe some general features of the three algorithms, and then present their specific details.

The algorithms are executed by a network having the structure of a binary tree K with w leaves. 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 $\mathbf{x} = (x_0,...,x_{N-1})$ is segmented into N/w wavefronts of width w, where $1 \le w \le N/\log N$ (for ease of discussion, we assume that N is a power of two). The i-th wavefront is denoted $\mathbf{x}_i = (x_{iw}, x_{iw+1},...,x_{(i+1)w-1})$, where $i = 0,1,...,N/w-1$. 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

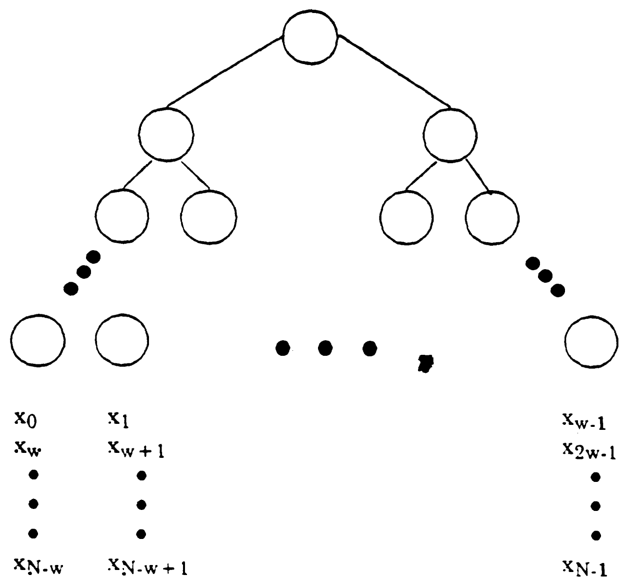

Figure 1. Input protocol for prefix computation on a binary-tree network.

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 - $2level(V)+1$ 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 of a given level on the same wavefront, some information relative to that wavefront must be stored at the nodes. As we shall see, the algorithms for CF semigroups are more size-efficient than the ones for general semigroups exactly because less information is explicitly stored at the nodes. Correspondingly, the correctness of these algorithms is less immediate to establish.

4.1 *General Semigroups*

Binary tree K emulates in a straightforward manner the behavior of the well-known prefix network described in [LF80, BK82, BP86b] and called "twisted-reflected-tree" in [BP86b]. The algorithms are best explained as follows.

A given internal node V of K determines a segmentation of the input sequence $\mathbf{x}$ as $\mathbf{x} = \boldsymbol{\alpha}_0\boldsymbol{\beta}_0\boldsymbol{\alpha}_1\boldsymbol{\beta}_1 ... \boldsymbol{\alpha}_{N/w-1}\boldsymbol{\beta}_{N/w-1}$, where $\boldsymbol{\beta}_0\boldsymbol{\beta}_1...$ is the subsequence of $\mathbf{x}$ which is input by the successive wavefronts to the leaves of the subtree rooted at V ($\boldsymbol{\alpha}_0$ may be empty). For a given sequence $\mathbf{x}$, let x denote the product of its terms. Each β_j is further segmented as $\beta_j = \beta_j'\beta_j''$, where β'_j and β_j'' are input at the left and right subtrees of V, respectively.

Referring to the j-th wavefront, during the ascending phase internal node V computes $\beta_j = \beta_j'\beta''_j$ from the values β_j' and β''_j received from its offsprings. In addition, the root (for which all the $\boldsymbol{\alpha}$'s are empty) maintains a state σ initialized to e (the monoid identity) and updated as $\sigma := \sigma\beta_j$. During the descending phase, nonroot node V must receive from its parent the prefix $\gamma = \alpha_0\beta_0...\alpha_{j-1}\beta_{j-1}\alpha_j$; if V has stored β_j', then it can provide the correct prefixes γ 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 β'-queue) capable of storing $2level(V)+1$ semigroup elements (the β_j'). In addition, each nonroot node has three cells to store the elements to be forwarded in the next step; note that, for the root, one of these elements, β_j, is "forwarded" to the root itself. The contents of all cells are initialized to e, the monoid identity. In summary, the generic step runs as follows:

Generic Step

ASCENDING PHASE

begin forward β to parent, [root: $\sigma := \sigma\beta$,]
β': = term received from left child,
β'': = term received from right child;
insert β' into β'-queue;
β: = $\beta'\beta''$
end

DESCENDING PHASE

begin forward γ to left child,
forward $\gamma\delta'$ to right child;
γ: = term received from parent; [root: $\gamma := \sigma$;]
extract δ' from β'-queue;
compute $\gamma\delta'$
end

The algorithm is readily implemented by endowing the module of a node with a semigroup multiplier and a queue capable to store $(2level(V)+1) = O(\log\ w)$ 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 4. The size-time complexity of the prefix problem on a boolean network is $S = 0((N/T)\ \log(N/T))$, for $T \in [\Omega(logN), O(N)]$.

For NCF semigroups the bound of Theorem 4 is optimal, as shown by Theorem 3. For $T = \Theta(log\ N)$, the time lower bound of Proposition 1 is achieved. 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, 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 has to store in the ascending phase for completing processing in the descending phase.

The memory requirement in the algorithm of Section 4.1 comes from the necessity to store at each node V the product β' forwarded by the left child for $(2level(V)+1)$ steps. In the steady state, this implies the simultaneous storage of data relative to $(2\ level(V)+1)$ wavefronts. On the other hand, if $\gamma\beta_j = \gamma$, the prefix is constant for all leaves of the subtree rooted at V for the j-th output wavefront: in this case just γ must be passed to both offsprings. Therefore a record of (β_j', β_j'') must be kept from the ascending phase only for those values of j for which $\gamma\beta_j \neq \gamma$, a situation that in a CF semigroup can occur at most l_A times.

How can this condition be tested in the input phase if γ is yet unknown? The following scheme is proposed to answer this question. The root of K maintains a state σ initially set to e and updated as $\sigma = \sigma\beta$, as in the previous algorithm. Each remaining internal node V of K constructs during the ascending phase a *history tree* $H(V)$ of depth at most l_A as follows. Each vertex of $H(V)$ is labelled with an element of A and is either active or inactive; each arc is labelled either by α_j or by β_j, where $j = 0,1, \ldots (N/w-1)$. Initially, $H(V)$ consists of its root, labelled by the identity e. The general j-th step consists of two substeps: the first for processing (i.e., guessing) α_j, the second for processing (i.e., observing) β_j. Let a vertex v of $H(V)$ labelled a be active at the beginning of the first substep: the offsprings of v are a set of vertices labelled by $\{a\alpha_j : \alpha_j \in A, a\alpha_j \neq a\}$, where the arc from a to $a\alpha_j$ is labelled α_j; v remains active if, for some α_j, $a\alpha_j = a$. The second substep is analogous, except that an active v labelled by a has an offspring only if $a\beta_j \neq a$. In this case, a record of β_j is kept at v. In other words, a vertex v of $H(V)$ keeps record only of a transition in $G(A)$ caused by an input β_j as the pair (β_j', β_j''). For each V in K, $H(V)$ has $O(s^{l_A})$ vertices.

When processing the j-th wavefront in the ascending phase, the history tree $H(V)$ can be updated at V in time dependent only upon the semigroup size and operation.

When constructing the j-th wavefront in the descending phase, each nonroot internal node V traces its history tree under the control of labels provided by its parent and of labels stored in $H(V)$. Specifically, each instance $\alpha_0\beta_0\alpha_1\beta_1...$ corresponds to a unique path in the history tree: α_j is implicitly provided by the parent in the form of the product $\alpha_0\beta_0...\alpha_j$; β_j is picked up, if nonempty, at the node reached by $\alpha_0\beta_0 ... \alpha_j$. V is now in a position to pass the appropriate terms to its offsprings.

In summary, the generic node V of K performs the following actions:

Generic Step

ASCENDING PHASE
begin forward β to parent, [root $\sigma := \sigma\beta$,]
    $\beta' :=$ term received from left child,
    $\beta'' :=$ term received from right child;
    update $H(V)$;
    $\beta := \beta'\beta''$
end

DESCENDING PHASE
begin forward γ to left child and $\gamma\beta'$ to right child;
    $\gamma :=$ term received from parent; [root $\gamma := \sigma$;]
    in $H(V)$ move pointer to vertex v labelled γ among children
      of current vertex (and obtain β',β''));
    if $(\beta',\beta'') \neq (\Lambda,\Lambda)$ **then** in $H(V)$ move pointer to (unique) child of v
      labelled $\gamma\beta'\beta''$;
      store γ and $\gamma\beta'$
end

Finally we note that K consists of $O(w)$ modules, each of size independent of N. 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 5. For a CF semigroup A the prefix computation for an N-term sequence can be done in time T and size S, with $S = O(N/T)$, for $T \in [\Omega(\log N), O(N)]$.

4.3 *Insertion Semigroups*

The above result holds for any CF semigroup and is clearly optimal. However, for the very important case of insertion semigroups the tree module need not be as complicated as outlined above. 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) := terms received from parent;
        (β',β'') := next pair in queue;
        if β = β'β'' then
            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. The above scheme correctly computes the prefix sequence for insertion semigroups.

Proof. The decision whether to retain or not the pair (β',β'') in the queue at node V rests on the condition $state(V)\beta \neq state(V)$ or, equivalently, (for the j-th step) $\beta_0\beta_1 \ldots \beta_{j-1}\beta_j \neq \beta_0\beta_1 \ldots \beta_{j-1}$. We must show that this condition is implied by

$$\alpha_0\beta_0\alpha_1\beta_1 \ldots \beta_{j-1}\alpha_j\beta_j \neq \alpha_1\beta_1 \ldots \alpha_{j-1}\beta_{j-1}\alpha_j .$$

Indeed $\beta_0 \ldots \beta_{j-1}\beta_j = \beta_0 \ldots \beta_{j-1}$ implies $\beta_0\alpha_1 \ldots \beta_{j-1}\alpha_j\beta_j = \beta_0\alpha_1 \ldots \beta_{j-1}\alpha_j$ by the insertion property (1) and therefore $\alpha_0\beta_0 \ldots \alpha_j\beta_j = \alpha_0\beta_0 \ldots \alpha_j$. □

We can therefore conclude:

Theorem 6. For an insertion semigroup 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 $O(l_A \cdot logs)$ 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 5. Each nonroot module has a queue of $2l_A$ cells, each with $\lceil \log_2 s \rceil$ bits. □

5. 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 complexity of the networks as a function of N.

The major outstanding problem is the investigation of the dependence of network complexity upon semigroup size and operation. For example, in Theorem 6 we have shown that, for the important case of insertion semigroups, the upper bounds of Theorem 5 can be considerably improved. However, the construction of prefix boolean networks which are optimal also with reference to semigroup size remains an open problem.

6. Acknowledgement

We are indebted to D.E. Muller for his valuable suggestions.

References

[B81] G.M. Baudet, "On the area required by VLSI circuits," in H.T. Kung, R. Sproull, and G. Steele (eds.) *VLSI Systems and Computations*, pp. 100-107, Computer Science Press, Rockville, MD, 1981.

[BK82] R.P. Brent and H.T. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. C-31, n. 3, pp. 260-264, March 1982.

[BP85] G. Bilardi and F.P. Preparata, "The influence of key length on the area-time complexity of sorting," *Proceedings I.C.A.L.P., (Ed. W. Brauer)* Nafplion, Greece, Springer-Verlag, pp. 53-62, July 1985.

[BP86a] G. Bilardi, and F.P. Preparata, "Area-time lower-bound techniques with applications to sorting," *Algorithmica*, vol. 1, n. 1, pp. 65-91, 1986.

[BP86b] G. Bilardi and F.P. Preparata, "Digital filtering in VLSI," *Proceedings of Aegean Workshop on Computing* (Eds.F. Makedon et al.) Loutraki, Greece, Springer-Verlag, pp. 1-11, July 1986.

[CFL83] A.K. Chandra, S. Fortune, R. Lipton, "Unbounded fan-in circuits and associative functions," *Proceedings of the 15th Annual Symposium on Theory of Computing*, Boston, MA, pp. 52-60, April 1983.

[CS85] R. Cole and A. Siegel, "On information flow and sorting: New upper and lower bounds for VLSI circuits," *Proceedings 26th Annual Symposium on the Foundations of Computer* Science, Portland, OR, pp. 208-221, October 1985.

[DC80] P.W. Dymond and S.A. Cook, "Hardware complexity and parallel computation," *Proceedings 21st Annual Symposium on the Foundations of Computer* Science, Syracuse, NY, pp. 360-372, October 1980.

[F83] F.E. Fitch, "New bounds for parallel prefix circuits," *Proceedings 15th Annual ACM Symposium on Theory of Computing, Boston,* MA, pp. 100-109, April 1983.

[J80] R. B. Johnson, "The complexity of a VLSI adder," *Information Processing Letters*, vol. 11, n. 2, pp. 92-93, October 1980.

[K78] D.J. Kuck, *The structure of computers and computations*, New York: Wiley, 1978.

[KRS85] C.P. Kruskal, L. Rudolph, M. Snir, "The power of parallel prefix," *IEEE Transactions on Computers*, vol. C-34, n. 10, pp. 965-968, October 1985.

[LF80] R.E. Ladner and M.J. Fischer, "Parallel prefix computation," *Journal of the ACM*, vol. 27, n. 4, pp. 831-838, October 1980.

[S86] M. Snir, "Depth-size trade-offs for parallel prefix computation," *Journal of Algorithms*, vol. 7, no. 2, pp. 185-201, June 1986.

[T80] C.D. Thompson, "A complexity theory for VLSI," *PhD. Thesis, Department of Computer Science, Carnegie-Mellon University*, August 1980.