KAREL II CULIK
ARTO SALOMAA
DERICK WOOD

# Systolic tree acceptors

*RAIRO – Informatique théorique*, tome 18, nº 1 (1984), p. 53-69.
<http://www.numdam.org/item?id=ITA_1984__18_1_53_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 18, n° 1, 1984, p. 53 à 69)

# SYSTOLIC TREE ACCEPTORS (*)

by Karel CULIK II (1), Arto SALOMAA (2), Derick WOOD (1)

Communicated by J. BERSTEL

Abstract. — *This paper introduces systolic automata for VLSI as highly concurrent acceptors. Attention is focused on tree automata although other types are also discussed briefly. A number of preliminary results about the capabilities of such automata are obtained. The approach not only opens up a new and interesting area of language theory but also its results give useful hints for VLSI system design particularly with regard to the pros and cons of specific geometric structures.*

Résumé. — *Cet article introduit les automates systoliques pour VLSI comme accepteurs avec un grand degré de parallélisme. L'attention est concentrée sur les automates d'arbres, même si d'autres types sont aussi rapidement discutés. On obtient un certain nombre de résultats préliminaires sur de tels automates. L'approche ouvre un domaine nouveau et intéressant en théorie des langages; mais ses résultats donnent aussi des indications utiles pour la conception de systèmes VLSI, en particulier en ce qui concerne les « pros » et « cons » de certaines structures géométriques.*

## 1. INTRODUCTION

It is expected that one of the best ways to exploit the potential of VLSI is to put specialized large multiprocessor systems on a chip. To cope with the problems of communication in large parallel systems the use of the design methodology of systolic systems has been proposed ([1], Chapter B; [5]). A great variety of examples and specific constructions of such systems have been presented, see for example [1, 5, 6] and are often combined with complexity considerations. On the other hand, we do not know of any attempts to characterize the capabilities of VLSI circuits of specific types.

This paper constitutes such an attempt for systolic systems when the communication structure is a tree. Other types of communication structures (e. g. trellis and hexagonal in [7]) have also been investigated. Common to these types is that the processors perform combinational functions, all being

(*) Received and revised in August 1982.
This work was supported by Natural Sciences and Engineering Research Council of Canada Grants Nos. A 7403 and A 7700.
(1) Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.
(2) Department of Mathematics, University of Turku, Turku, Finland.

synchronized, and communicate through communication lines involving unit delay (i. e. physically there would be a register on each communication line).

Specifically, in this paper, we consider tree structures of processors. The leaves take the input which is then processed towards the root, bottom-up and in parallel. At the root the acceptance decision is made. Since the information flows only one-way (bottom-up), the intermediate results are always all on one level of the tree at any moment of time. This allows for "pipelining" (*see* [1, 5]). Such systolic systems can be viewed as a special type of bottom-up deterministic tree automata, and are also similar to cellular automata, e. g. the triangle acceptors of [8]. However, the emphasis in traditional cellular automata theory is different: one considers mainly computability.

Our specific model arose from a study of the use of VLSI circuits for text-editing; this is discussed further in Section 5.

There are also interconnections between our theory and the theory of iterative arrays, as well as the theory of circuits for language recognition. The reader might also want to compare [9] and [10] with our approach. Actually, systolic arrays are not considered in [9] and [10].

Systolic tree automata are very efficient acceptors, since they operate in logarithmic time and the number of processors is $0(n)$. (This is achieved by letting the underlying tree grow exponentially, although this requirement is superfluous for our technical results.) In this paper we focus attention on the over-all capability of systolic tree automata: what can be done within *fixed* complexity. Observe, however, that because of the mode of operation a new word is fed at every time instant. Issues of complexity should be viewed accordingly.

The automata are very interesting from a purely language-theoretic point of view. Deterministic and parallel bottom-up processing bring about entirely new language-theoretic features. Traditional language theory, [4], is sequential in nature. Even the parallelism of $L$ systems, [3], is different from that in our case: it can be viewed as top-down parallelism. Compared with traditional hierarchies in language theory, our new language families contain not only everything simple (regular languages and essentially also DOL languages) but also languages very high up in the classical hierarchies.

Our results concerning what cannot be done give some idea of the restrictions of systolic arrays associated with a certain geometry. We believe that such results also have pratical significance for choosing an appropriate geometry for a particular task.

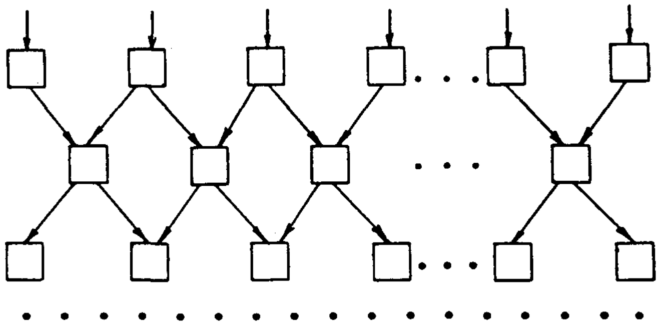
A brief outline of the contents of the paper follows. Section 2 contains a brief discussion of possible models and introduces the specific model investigated in this paper. Even the simplest version of this model turns out to be quite powerful and is capable of accepting every regular language, as shown in Section 3, which also contains further constructions involving acceptable languages of different types. Negative results and a criterion of nonacceptability are presented in Section 4 and, finally in Section 5 some remarks on further questions, structures, and results are given, and also some comments on the practical implications of our results for VLSI are given.

We refer the reader to [1] for further background and motivation as regards VLSI systems. The reader should also know the basics of language theory. Whenever need arises, [4] can be consulted. Our discussion about trees and related notions (such as "the underlying structure") is rather informal. The reader is referred to [11], which is actually a follow-up paper to this paper, for more formal definitions and discussions based on traditional tree-automata theory.

## 2. THE MODEL: DISCUSSION AND DEFINITIONS

In every formal definition of a "VLSI circuit" or "VLSI system" the underlying geometric pattern plays an important role. In order to guarantee efficiency, processors are only supposed to communicate with their neighboring processors. This immediately excludes many patterns. However, a number of patterns still remain, such as rectangular, hexagonal and tree-like patterns. One possibility would be to include the geometric pattern as one variable component in the definition. However, we found this to be too abstract and too general and prefer to study different patterns separately.

Consider the "rectangular" pattern:



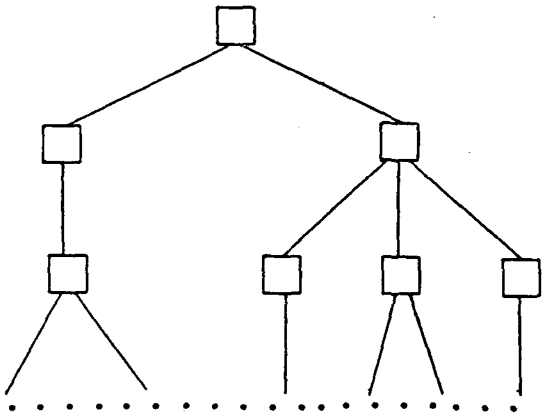
The processors are indicated by squares and the flow of information by arrows. The input word is fed, letter by letter, to the processors in the top row. The output of each processor depends only on its current inputs. The final output is produced by the processors in the bottom row. Thus, the whole circuit can be viewed as a transducer. Of course, a suitable modification can be viewed as an acceptor.

Since we are not studying rectangular patterns in this paper, we omit more detailed and formal definitions. It is to be emphasized, though, that the most essential feature of this model is the uni-directional flow of information. This is the basic difference with respect to cellular automata, for instance.

It is clear that the rectangular model is capable of realizing, among other things, sequential machine mappings. (In fact, this is achieved by leftward arrows only.) The hexagonal model, similarly defined, possesses somewhat different capabilities. For instance, it can do Boolean matrix multiplication (with proper modifications in the input format), which can be shown similarly as in [1].

Before introducing our tree models formally, we discuss them intuitively. The models investigated in this paper are *systolic tree automata* and *binary systolic tree automata* (STA and BSTA).

Consider an infinite tree $T$.



It is assumed that $T$ has no leaves, i. e. all branches are continued. Moreover, the number of nodes at the $k$-th level (i. e., nodes whose distance from the root equals $k$) grows exponentially in terms of $k$. The nodes are labelled by letters from a finite alphabet. A requirement of regularity is that the infinite labelled tree has only a finite number of non-isomorphic subtrees. (This is only one of the reasonable requirements. We might also allow different structures for even and odd levels, for instance.)

Each labelled node is viewed as a processor of the correct "arity", i. e., if it has $n$ sons then an $n$-place function is associated to it. Furthermore, the domains and ranges should match. The processors are also capable of taking external inputs in the following fashion.

Consider an input word $w$ with length $t$. We choose the first level in the tree with $u \geqq t$ nodes. The word $w \#^{u-t}$ (where # is a special end marker) is now fed, letter by letter, to the processors on the level in question. (We may visualize this as if we were using a chip of the correct size, and the last line of the chip has pins for external inputs.) Information now flows bottom-up. Whether or not the word $w$ is accepted, depends on the output of the root. Observe that this tree-like pattern is especially suitable for acceptors because information converges to one particular node.

It is worth noting that a finite cut of an STA with $n$ leaves can be laid out in a chip using only $0(n)$ area, *see* [1].

Our STA operates exactly as a deterministic bottom-up tree automaton. However, as regards traditional tree automata theory, there are two essential differences:

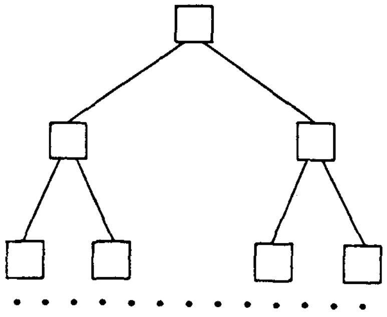
(i) For STAs the tree is a part of the automaton and, consequently, arbitrary trees cannot be regarded as inputs.

(ii) Synchronization is forced in case of STAs by cutting the given (infinite) tree on a specific level.

Thus, for instance, Theorems 1 and 5 below differ basically from their counterparts in tree automata theory. There are however some similarities between STAs and the tree controlled grammars investigated in [2].

A special case of STA are the binary systolic tree automata (BSTA). The starting point here is the complete binary infinite tree where all nodes are labelled by the same letter, that is all processors are identical. Otherwise, everything remains the same as before.

As an example of BSTA, consider the binary infinite tree:



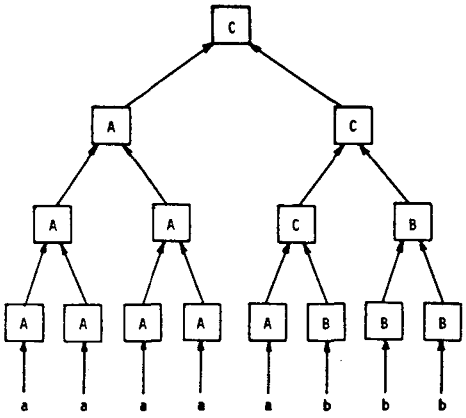
The processor computes the binary function:

$$f: \{A, B, C, \#\}^2 \rightarrow \{A, B, C, \#\},$$

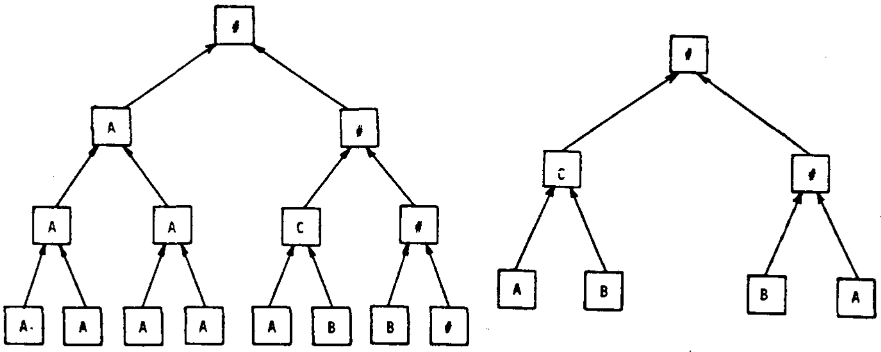
defined as follows:

$$f(A, A)=A;$$
$$f(B, B)=B;$$
$$f(A, B)=f(A, C)=f(C, B)=C;$$
$$f(x, y)=\# \quad \text{in all other cases.}$$

The external input alphabet is $\{a, b\}$, and the external inputs $a, b, \#$ produce the processor outputs $A, B, \#$, respectively. The convention of acceptance is the output $C$ from the root. Thus, the input $a^5 b^3$ gives rise to the following computation:



The computations from the inputs $a^5 b^2$ and *abba* are given below:



Thus, only $a^5 b^3$ of these three words is accepted. It can be easily verified that the accepted language is in this case:

$$L = \{ w \in a^+ b^+ \mid w \text{ is of length } 2^n, \text{ for some } n \geqq 1 \}.$$

Observe that acceptance takes logarithmic time.

We now define formally the notions discussed above.

Consider an infinite tree $T$ with no leaves, i. e., all branches are continued *ad infinitum*. For $k = 0, 1, 2, \ldots, LEVEL(k)$ denotes the set of all nodes whose distance from the root equals $k$. Clearly, there is a natural ordering (from left to right) of the elements of $LEVEL(k)$. The nodes of $T$ are labelled by letters from a finite alphabet $\Sigma_p$, referred to as the alphabet of *processors*, subject to the arity condition stated below.

Assume that $T$ has only finitely many (labelled) non-isomorphic subtrees. (This condition is referred to as the *regularity condition* in the sequel.) Assume, further, that there is a constant $\alpha > 1$ such that $LEVEL(k)$ contains more than $\alpha^k$ nodes for every $k \geqq 1$. (This condition will be referred to as the *exponential growth condition* or, briefly, the *growth condition.*)

Let $\Sigma_T$ and $\Sigma_O$ be two further finite alphabets, the *terminal* and *operating* alphabet, respectively.

With each letter $A$ in $\Sigma_p$, an integer $n(A) \geqq 1$ is associated, called the *arity* of $A$. (Thus, $\Sigma_p$ can be viewed as a ranked alphabet.) It is assumed that the labelling of $T$ satisfies the following *arity condition:* each node labelled by $A$ has $n(A)$ sons.

To every letter $A$ in $\Sigma_p$ we associate two functions:

$$g_A : \quad \Sigma_T \rightarrow \Sigma_O \qquad \text{and} \qquad f_A : \quad \Sigma_O^{n(A)} \rightarrow \Sigma_O.$$

Furthermore, we specify a subset $\Sigma'_O$ of $\Sigma_O$, referred to as the set of *accepting letters.*

We are now ready for our basic definition.

DEFINITION: A *systolic tree automaton* ($STA$), is a construct:

$$K = (T, \Sigma_p, \Sigma_O, \Sigma'_O, \Sigma_T, \{ g_A \}, \{ f_A \} \mid A \in \Sigma_p).$$

where $T$ is an infinite tree satisfying the growth condition with nodes labelled by letters of the alphabet $\Sigma_p$ in such a way that both the arity and the regularity condition are satisfied. Moreover, $\Sigma_O, \Sigma_T$ and $\Sigma'_O \subseteq \Sigma_O$ are alphabets and, for each $A \in \Sigma_p$:

$$g_A : \quad \Sigma_T \rightarrow \Sigma_O \qquad \text{and} \qquad f_A : \Sigma_O^{n(A)} \rightarrow \Sigma_O,$$

are functions, where $n(A)$ is the arity of $A$ (when $\Sigma_p$ is viewed as a ranked alphabet). It is also assumed that $\Sigma_T$ contains the special symbol $\#$.

Every word $w$ over the alphabet $\Sigma_T - \{\#\}$ determines a unique element of $\Sigma_0$, denoted by $OUTPUT(K, w)$, as follows. Assume that $w$ is of length $t$. Set $k$ be the smallest integer such that $LEVEL(k)$ in $T$ contains $u \geqq t$ nodes. (Clearly, $k$ exists and is unique.) Let $A_1, \ldots, A_u$ be the nodes (from left to right) in $LEVEL(k)$. We now define:

$$OUTPUT(K, w, k),$$

to be the word $x_k$ of length $u$ over the alphabet $\Sigma_0$ such that the $i$-th letter of $x_k$, for $i=1, \ldots, u$, equals the result of applying $g_{A_i}$ to the $i$-th letter of $w\#^{u-t}$.

Assume that we have already defined $OUTPUT(K, w, j) = x_j$, for some $j$ with $1 \leqq j \leqq k$. Then $OUTPUT\ (k, w, j-1)$ is defined as follows. Let $LEVEL\ (j-1)$ contain $r$ nodes. [Clearly, $r \leqq |x_j|$ and $|x_j|$ equals the number of nodes in $LEVEL(j)$.] We now write:

$$x_j = y_1 \ldots y_r,$$

where each $y_i$ corresponds to those nodes in $LEVEL\ (j)$ that are sons of the same node in $LEVEL\ (j-1)$. Let the latter node be labelled by $B_i$. Then $OUTPUT\ (K, w, j-1)$ is the word of length $r$ over $\Sigma_0$ whose $i$-th letter, for $i=1, \ldots, r$, equals the result of applying $f_{B_i}$ to the letters of $y_i$ (in the correct order). The arity condition guarantees that $f_{B_i}$ is an $s$-place function where $s$ is the length of $y_i$.

We now define:

$$OUTPUT(K, w) = OUTPUT(K, w, 0).$$

The word $w$ is accepted by $K$ if and only if $OUTPUT\ (K, w)$ is in $\Sigma_0'$. [Observe that $OUTPUT\ (K, w)$ is always a *letter* of $\Sigma_0$.]

The *language accepted by* an STA is defined by:

$$L(K) = \{ w \in (\Sigma_T - \{\#\})^* \mid OUTPUT(K, w) \in \Sigma_0' \}.$$

Languages of this form are referred to as *STA acceptable.*

An STA is referred to as *binary* if:

(i) the underlying tree is the complete binary tree, and;

(ii) the alphabet of processors, $\Sigma_p$, consists of one letter only.

We use the abbreviation *BSTA* in this case. Thus, we may speak, for instance of *BSTA acceptable* languages.

We conclude this section with a brief discussion concerning the assumptions about the tree $T$. Clearly the arity condition must be satisfied, otherwise our model does not work. The other two conditions are to some extent arbitrary.

The growth condition guarantees efficiency: acceptance is always in logarithmic time.

Recall that the basic idea behind the infinite tree is that, for each task of a particular size, a horizontal "cut" in the infinite tree determines a "chip" for the solution of the task. This would not be realistic for very irregular infinite trees. Therefore, some regularity condition is definitely needed. We want to emphasize that it is not even necessary to take always cuts of the *same* infinite trees. It is completely reasonable, for instance, to start with two trees and take cuts of an even depth from one of them and cuts of an odd depth from the other. Results such as the "normal from theorem" established in Section 3 give further motivation for our regularity condition. Namely, we show that substantially more restricted structures still accept the same family of languages.

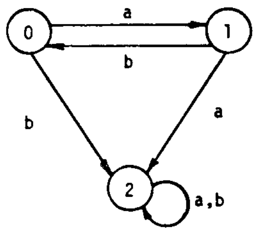
## 3. PROPERTIES OF STA ACCEPTABLE LANGUAGES

Most of the results in this section are straightforward and therefore their proofs are omitted. While they are not, perhaps, the most meaningful results from a VLSI point of view they are basic. Further results about decidability are to be found in [11].

THEOREM 1: *Every regular language is BSTA acceptable.*

The proof of this result is illustrated by the following example.

Consider the language $(ab)^*$ accepted by the automaton:



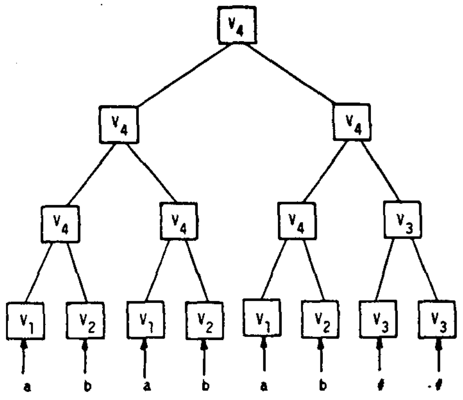
where 0 is the initial and the only final state. We have now:

$$g(a)=\{(0,1),(1,2),(2,2)\}=V_1,$$
$$g(b)=\{(0,2),(1,0),(2,2)\}=V_2,$$
$$g(\#)=\{(0,0),(1,1),(2,2)\}=V_3.$$

and, for instance:

$$f(V_1, V_1) = \{(0,2), (1,2), (2,2)\},$$
$$f(V_2, V_2) = \{(0,0), (1,2), (2,2)\} = V_4,$$
$$f(V_4, V_4) = V_4,$$
$$f(V_4, V_3) = V_4.$$

The acceptance of the word $(ab)^3$ is illustrated by the following tree [$V_4$ is an element of $\Sigma_0'$ because it contains the pair (0, 0).]



Observe that the acceptance of the empty word works in a proper way because $(0, 0)$ is an element of $V_3$.

In general, the acceptance of a particular language $L$ by some STA depends heavily on the underlying infinite tree $T$ (referred to as the underlying *structure* in the sequel), i. e., a different tree structure is not capable of accepting $L$. Therefore, the following result is of interest.

THEOREM 2: *Let $K$ be an arbitrary STA and let $T$ be the first component in $K$, i. e., $T$ is the underlying infinite tree. Then, for every regular language $R$, there is a STA $K_R$ whose first component equals $T$ and such that $L(K_R) = R$.*

Because of the "malleability" of regular languages this result is not too surprising. Whether or not this is the largest family for which such a result holds is an open question.

We have already given an example of a nonregular STA (in fact BSTA) acceptable language in Section 2. Essentially the tree structure can check exponential length, which induces a large extension to the family of acceptable languages. Another extension is due to the closure properties established below.

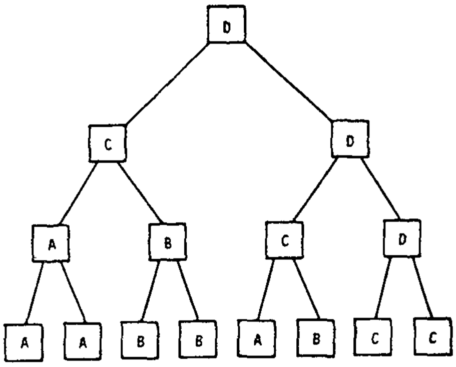
We begin with another example. It is a somewhat sophisticated BSTA acceptable language. The terminal alphabet $\Sigma_T$ equals $\{a, b, c\}$. Furthermore, $\Sigma_0 = \{A, B, C, D, \#\}$ and $\Sigma_0' = \{D\}$. The function $g$ is defined by:

$$g(a) = A, \quad g(b) = B, \quad g(c) = C, \quad g(\#) = \#.$$

and the function $f$ by:

$$f(A, A) = A, \quad f(B, B) = B, \quad f(A, B) = C,$$
$$f(C, C) = D = f(C, D),$$
$$f(x, y) = \#, \quad \text{otherwise.}$$

The following tree depicts the acceptance of the word $a^2 b^2 abc^2$:



The exhaustive characterization of the accepted language is left to the reader.

In general, the underlying infinite tree influences profoundly what a particular STA is able to do. For instance, if a language is acceptable with the complete binary tree as the underlying tree, it need not be acceptable if the complete ternary tree is chosen instead, and vice versa. However, our main results in this section show that only the *structure* of the tree is important, not the labelling. This is also the basic reason why we defined BSTA in the way we did.

For every STA $K$, its underlying unlabelled infinite tree is referred to as the *structure* of $K$. Although the family of all STA acceptable languages has weak closure properties, the situation becomes different if languages acceptable by STAs with the same structure are considered.

THEOREM 3: *For every $T$, the family of language acceptable by STAs with the underlying structure $T$ is closed under Boolean operations.*

By Theorem 2, we obtain now the following corollary of Theorem 3.

THEOREM 4: *The family of STA acceptable languages is closed under union and intersection with regular languages.*

We are now ready to establish the main result in this section. The result can be viewed as a "normal form theorem": any language acceptable by a STA with a fixed structure is acceptable by a STA with the same structure where the labelling of the tree depends on the arity alone, i. e., all nodes with the same arity have identical labels. It can also be viewed as a universality result: any given tree structure is universal in the sense that, for the acceptance of a particular language, it suffices to label the tree in a deterministic top-down fashion. (We note in passing that our Theorems 2-5 remain valid even if the growth requirement is dropped from the definition of STA.)

THEOREM 5: *Assume that L is the language accepted by the STA:*

$$K=(T, \Sigma_p, \Sigma_0, \Sigma_0', \Sigma_T, \{g_A\}, \{f_A\} \mid A \in \Sigma_p).$$

*Then L is also accepted by some STA:*

$$K_1=(T, \Sigma_p^1, \Sigma_0^1, (\Sigma_0')^1, \Sigma_T, \{g_B^1\}, \{f_B^1\} \mid B \in \Sigma_p^1).$$

*where the cardinality of $\Sigma_p^1$ equals the number of different arities in T. Consequently, any language accepted by some STA whose structure is the complete binary tree is, in fact, BSTA acceptable.*

*Proof:* To specify $K_1$, we have to say what the new items are. The regularity condition guarantees that there are only finitely many arities in $T$. To each of them we associate a letter-these letters form the alphabet $\Sigma_p^1$.

Consider the STA $K$. The nodes of $T$ are labelled by letters from $\Sigma_p$. Moreover, the regularity condition guarantees that the tree so labelled has only finitely many, say $n$, subtrees. We now introduce an additional node labelling of $T$ by the numbers $1, \ldots, n$ in such a way that nodes determining identical subtrees get the same number. Clearly, if two nodes are labelled by the same number, they are also labelled by the same letter from $\Sigma_p$. Moreover, the node labelling by the numbers $1, \ldots, n$ is top-down deterministic: each number attached to a node uniquely determines the numbers attached to its sons and, hence, also the labels from $\Sigma_p$ attached to its sons. This is the basic idea behind the following construction.

The alphabet $\Sigma_0^1$ is now defined by:

$$\Sigma_0^1=\{(a_1, \ldots, a_n) \mid \text{each } a_i \text{ in } \Sigma_0 \cup \{E\}\}.$$

Thus, $\Sigma_0^1$ consists of ordered $n$-tuples of letters of the original operating alphabet augmented with an additional letter $E$ (blank), where $n$ is the constant determined by the regularity condition.

Let $j$ be the number of the root of the tree (in the labelling by numbers introduced above). Then $(\Sigma_0')^1$ is defined to be the subset of $\Sigma_0^1$, consisting of all those $n$-tuples, where an element of $\Sigma_0'$ appears in the $j$-th position. (The letters appearing in other positions are arbitrary).

We still define the new functions $g_B^1$ and $f_B^1$. The values of all of these functions are elements of $\Sigma_0^1$. Let $b$ be an arbitrary terminal letter in $\Sigma_T$. Then, for each $i$, the $i$-th component in $g_B^1(b)$ equals the value $g_{A(i)}(b)$. [Recall that $A(i)$ is the label from $\Sigma_p$ of the node numbered $i$. Thus, the $i$-th component indicates what would happen in the original STA if the processor in question would be $A(i)$. Indeed, $K_1$ has only one $g$-function, since the definition of $g_B^1$ does not depend on $B$.]

We define, finally, the function $f_B^1$ where $B$ has arity $k$. (Recall that, for each arity, there is only one letter of that arity in $\Sigma_p^1$.) Consider an arbitrary argument $(c_1, \ldots, c_k)$, where each $c$ is an element of $\Sigma_0^1$. It suffices to define, for each $i=1, \ldots, n$, the $i$-th component $d_i$ of the value $f_B^1(c_1, \ldots, c_k)$. If the arity of $A(i) \neq k$, we define $d_i = E$. If the arity of $A(i)=k$, let $j_1, \ldots, j_k$ be the numbers associated to the sons (from left to right) of the node labelled by $i$. (As observed above, the $j$'s are uniquely determined by $i$.) Furthermore, let $e(j, t)$ be the $j$-th component in $c_t$, for:

$$j=1, \ldots, n \qquad \text{and} \qquad t=1, \ldots, k.$$

We now define:

$$d_i = \begin{cases} f_{A(i)}(e(j_1, 1), \ldots, e(j_k, k)), & \text{if each } e \neq E, \\ E \quad \text{if at least one of the } e\text{'s is } E. \end{cases}$$

This completes the definition of $K_1$.

It can now be verified that $K_1$ accepts $L$. Indeed, $K_1$ simulates all possible computations of STAs obtained from $K$ by changing the labelling but still preserving the arities and the correct subtrees. In the computation of $K_1$ all possibilities corresponding to different labels are taken into account. However if the arity is wrong [in the above construction this is indicated by saying that the arity of $A(i) \neq k$], then the letter in question (i. e. $A$) cannot appear as the label of that node. Finally, the accepting letters check that the correct processor appears at the root and produces an accepting output. ■

## 4. NONACCEPTABILITY

We have seen above that all regular languages are BSTA acceptable (and also STA acceptable with any underlying structure). On the other hand, even the family of BSTA acceptable languages contains very complicated languages. Since the language

$$L=\{a^{2n} \mid n \geqq 0\},$$

is clearly BSTA acceptable, Theorem 3 implies that also the completement of $L$ is BSTA acceptable. But the complement of $L$ lies quite high up in all of the standard language-theoretic hierarchies, *see* [3] and [4].

On the other hand, quite simple languages (from the standard language-theoretic point of view) are not BSTA acceptable. We shall show below that $\{a^n b^n \mid n \geqq 1\}$ is not BSTA acceptable. Indeed, we do not know any nonregular context-free BSTA acceptable language. Basically, this family seems to consist of the Boolean closure of regular sets and languages with a suitable exponential growth. For instance, it can be shown that the language:

$$\{a^{3^n} \mid n \geqq 1\},$$

is not BSTA acceptable. The same holds true with respect to the language

$$L_1=\{(abc)^{2^n} \mid n \geqq 1\}.$$

which is of course somewhat unnatural compared with the fact that the language $L$ above is BSTA acceptable. However, $L_1$ becomes acceptable if we make slight modifications in the input format ($g$-functions). For instance, we can consider processors "without input pins", i. e., only some specified processors take letters of $\Sigma_T$ as inputs.

We would like to stress also the following facts as regards our input format. At a first glance, it might seem unnatural that the input has to be fed to a specific level of the tree. This means in practice that a chip of a certain size is not able to process very short words, which is of course somewhat awkward. Hence, it would be desirable that STAs are *stable* in the following sense: the same language is accepted even if we drop the condition that a word must be fed on the minimal level. (Thus, we can input a word on any level providing enough space.)

But now it is possible to establish the following result which eases the difficulty described above and, thus, gives further motivation for our input format. Assume that the underlying structure $T$ is initially prefix preserving in the sense that all the subtrees defined by the nodes on the left-most path of $T$ are identical. (This condition is immediately satisfied for BSTAs, as well as for any balanced trees.) Then we can effectively construct an equivalent stable STA (with the same structure).

We give, finally, a method of showing nonacceptability. For simplicity, we restrict attention to BSTA.

We shall show first that the language $\{a^n b^n \mid n \geqq 1\}$ is not BSTA acceptable. Assume it were. Then, for each $k \geqq 1$, the binary tree of height $k$ would have to accept all words $a^n b^n$ such that:

$$2^{k-1} < 2n \leqq 2^k.$$

Recall that these words appear as inputs in the form $a^n b^n \#^i$ where $i = 2^k - 2n$.

We now choose $k$ large enough such that the number of words of length $2^{k-1}$ and of the form:

$$(*) \qquad b^{2i} \#^{2j}, \qquad i \geqq 1, \qquad j \geqq 0,$$

is greater than the cardinality of the operating alphabet of the $B\ STA\ K$ accepting $L = \{a^n b^n \mid n \geqq 1\}$. For each word $(*)$, there is a unique word $w_i$ of length $2^{k-1}$ such that the word $w_i b^{2i}$ is in $L$. Moreover, for two distinct words $(*)$, the corresponding words $w_i$ are also distinct.

On the other hand, by the choice of $k$, there are two distinct numbers $r$ and $s$ such that:

$$OUTPUT(K, w_r b^{2r}, 1) = OUTPUT(K, w_r b^{2s}, 1).$$

But this is a contradiction because $w_r b^{2r}$ is in $L$, whereas $w_r b^{2s}$ is not in $L$.

Essentially the same argument can be used to show that, for instance, the language:

$$\{a^{3^n} \mid n \geqq 1\},$$

is not BSTA acceptable.

Intuitively, the argument concerning the language $\{a^n b^n \mid n \geqq 1\}$ is based on the fact that $K$ cannot simultaneously keep track of the borderline between the $b$'s and the #'s and also check that the number of $a$'s equals that of $b$'s. Here the exponential growth is essential: if arbitrary STA without growth condition are considered, the language $\{a^n b^n \mid n \geqq 1\}$ becomes acceptable.

A similar idea is behind the proof of the following result which can be viewed as a lemma for proving nonacceptability.

We need a couple of simple notions. Let $L$ be a language and $k$ a positive integer. Denote by $\#(L, k)$ the cardinality of the set of words in $L$ whose length is $>2^{k-1}$ and $\leqq 2^k$. We say that $L$ is *subexponential* if, for every $n$, there is a $k$ such that:

$$\#(L, k) > n.$$

We say that $L$ has *bounded tail ambiguity* if there is an integer $t$ such that, for every $k$ and every word $w$ of length $2^{k-1}$, there are at most $t$ words $w'$ with length $\leqq 2^{k-1}$ such that $ww'$ is in $L$.

THEOREM 6: *No subexponential language L with bounded tail ambiguity is* BSTA *acceptable.*

*Proof:* The argument is basically the same as in the example considered above, and so we give only an outline. Let $K$ be an arbitrary BSTA accepting $L$. Because $L$ is subexponential we can, for any integer $m$, choose a large enough $k$ such that the "tail half" of at least $m$ words in $L$ produces the same output from the right son of the root. (In the example it was sufficient to take two such words.) We obtain a contradiction by choosing $m$ to be greater than the constant $t$ defining the bounded tail ambiguity of $L$. ■

## 5. CONCLUDING REMARKS

This paper focuses attention on systolic *tree* automata as a possible model for VLSI. The study seems to open a new and interesting area in language theory. Some other questions, such as decidability are considered in a separate paper. Work dealing with other geometric patterns can be found in [7].

In practical applications it might be desirable to have one VLSI chip with a fixed number of input pins but universal in the sense that after feeding it with a program, i. e. the description of a particular language $L$, it will be able to test membership of an input string in $L$. For example, it might be very desirable to have a chip-subroutine as a part of a text editor capable of testing the membership of a string of characters (of length fitting on one line of the screen of a computer terminal) in an arbitrary regular set.

Our theory can be used to formalize this situation. The structure of the chip will be fixed (any regular structure will be suitable for testing membership in regular sets as we have shown) to be a balanced binary tree, say. We may assume that at every (branching) node of the tree there is a universal processor able to realize any function of bounded complexity (e. g. driven by a limited-size table of a function).

In this setting a VLSI-program is very simple, namely a finite number of rules for naming sons (from left to right) of a father which is already named.

The names of nodes can be identified with the tables of realized functions. Since every tree of processors (satisfying the regularity condition) has only a finite number of different (infinite) subtrees we can choose the father's name to determine uniquely their son's names. Thus we can use finitely many rules to fill in a top-down manner the names of all the processors (node labels).

Mathematically, we are generating a DOL derivation tree [3] starting from a single starting symbol. Practically, this would require that all communication lines in the tree-structured processor are bi-directional. We consider two phases, in Phase I we feed a program (language description compiled elsewhere, e. g. on another chip) into the root and communicate it top-down throughout the tree. After Phase I is completed we have a specific STA as defined in this paper. It can then be used in Phase II as a bottom-up string acceptor, which makes use only of the bottom-up communication directions in the tree.

## REFERENCES

1. L. Conway and C. Mead, *Introduction to VLSI Systems*, Addison-Wesley, 1980.
2. K. Culik II and H. A. Maurer, *Tree Controlled Grammars*, Computing, Vol. 19, 1977, pp. 129-139.
3 G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, 1980.
4. A. Salomaa, *Formal Languages*, Academic Press, 1973.
5. H. T. Kung, *Let's Design Algorithms for VLSI Systems*, Proceedings of the Caltech Conference on VLSI, Charles L. Seitz, Ed., Pasadena, California, January 1979, pp. 65-90.
6. C. E. Leiserson and J. B. Saxe, *Optimizing Synchronous Systems*, 22nd Annual Symposium on Foundations of Computer Science, October 1981, pp. 270-281.
7. K. Culik II, J. Gruska and A. Salomaa, *Systolic Trellis Automata*, International Journal of Computer Mathematics, to appear.
8. C. R. Dyer and A. Rosenfeld, *Triangle Cellular Automata*, Information and Control, Vol. 48, 1981, pp. 54-69.
9. R. W. Floyd and J. D. Ullman, *The Compilation of Regular Expressions into Integrated Circuits*, Proceedings 21st Annual Symposium on Foundations of Computer Science, I.E.E.E. Computer Society, 1980, pp. 260-269.
10. M. J. Foster and H. T. Kung, *Recognizable Regular Languages with Programmable Building-Blocks*, in J. P. Gray, Ed., VLSI 81, Academic Press 1981, pp. 75-84.
11. M. Steinby, *Systolic Trees and Systolic Language Recognition by Tree Automata*, Theoretical Computer Science, to appear.