SYSTOLIC TRELLIS AUTOMATA (FOR VLSI)[1]

Karel Culik, Jozef Gruska[2], & Arto Salomaa[3]
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-81-34

December 1981

1. This work was supported by the Natural Science and Engineering Research Council of Canada under the grants A-7403 and A-1617.

2. On a leave of absence from the Computing Research Center, Bratislava, Czechoslovakia.

3. On a leave of absence from the University of Turku, Finland.

Abstract

A model for VLSI circuits, referred to as a systolic trellis automaton, is introduced. It consists of a triangularly shaped system of hexagonally connected processors (functional elements) where the flow of data is uni-directional and there is a fixed delay in all interconnections. Systolic trellis automata can be viewed as a special case of pure systolic systems [2]. Attention is focused on regular trellis automata which can be easily laid out on a chip.

We describe a variety of programming techniques for trellis automata. The techniques might be of interest for VLSI designers in the sense that a design methodology for VLSI circuits is involved. The power of these techniques is illustrated by several examples. Trellis automata are shown to possess quite surprising capabilities. The languages acceptable by trellis automata have time complexity $O(n^2)$ with respect to deterministic Turing machines, and the family of acceptable languages is closed under Boolean operations. Special attention is focused on a subclass of trellis automata, referred to as homogeneous.

1. Introduction.

The cost and effectiveness of a multiprocessor system implementation on a chip depends mainly on the uniformity of the processors and on the regularity and simplicity of the communications. VLSI technology, therefore, encourages the building of multiprocessor systems where the processors are uniform and simple and their interconnections are regular [11].

Pipelining and multiprocessing at each stage of a pipeline should be taken into account. With pipelining several inputs can be processed immediately one after the other. Multiprocessing increases the performance at each stage of a pipeline.

All these concepts - a multiprocessor system, pipelining and parallelism - are captured in the notion of a systolic system. Informally, a systolic system is a multiprocessor network rhythmically processing data in a parallel fashion. More formally, a systolic system is a node-labeled and edge-weighted graph where the labels specify the processors and the weights represent the non-zero delays in the registers along the interconnections [10].

Several specific models of systolic systems have been considered so far. In all of the models the processors have no memory. The models differ in the ways the processors are interconnected and the data are handled.

It has been shown in several papers ([7], [8], [9], [10]), and in the monograph [11] how systolic systems can be designed to perform efficiently various important tasks. For example, orthogonally and hexagonally connected systolic systems are suitable for efficient matrix multiplication and LU-decomposition [9], [11]. Tree-like systolic systems can be used to implement priority queues [8]. Linearly connected systems can be designed for matrix-vector multiplication [7], and palindrome recognition [10].

A systolic system is called *pure* if there is an integer k such that every path from an input node to an output node possesses delay k [2]. A systematic investigation of the capabilities of pure systolic systems was begun in [6] where tree-like systolic systems, called systolic tree automata, are studied. In such an automaton the flow of information is uni-directional from the leaves to the root. This leads naturally to the investigation of such networks as acceptors. Actually, cuts of infinite trees (of processors) satisfying a natural regularity condition form the basis of the systolic tree automaton of [6]. The regularity condition implies that both data processing and tree generation can be done in a context-free way and that the corresponding chips are programmable in a natural top-down context-free manner.

In the present paper another model of systolic systems is considered. They are hexagonally connected systolic systems with uni-directional information flow.

Some of the most important and interesting applications of systolic systems deal with hexagonally connected systems, [7], [8], [9]. Hexagonally connected systems seem to be quite powerful. They process data in a specific context-sensitive fashion and, therefore, require special techniques to "program" them.

The goal of the present paper is to develop some programming techniques for such systolic systems and to present some (mathematical) results giving more insight concerning the power and limitations of such systems.

This paper is restricted to the study of triangularly shaped systolic systems of hexagonally connected arrays of combinational circuits, referred to as systolic triangularly shaped trellis automata (in short, <u>trellis automata</u>). The delay on oblique interconnections is one and on vertical two. Figure 1.1 presents the trellis structure of a systolic trellis automaton. The vertex labels are names of the corresponding processors.

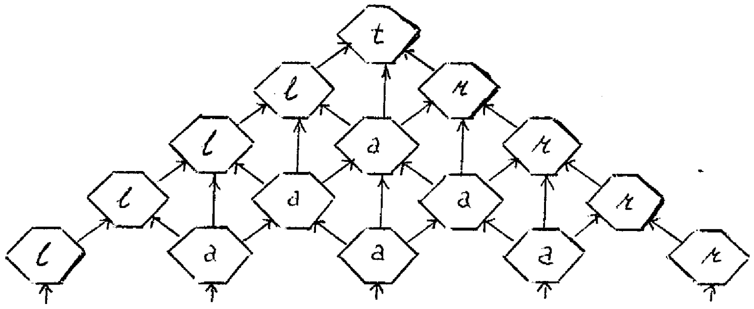


Figure 1.1

Our restrictions leave enough leeway to investigate programming techniques for uni-directional hexagonally connected systolic systems.

Systolic trellis automata get their input data, i.e., an input word, on the external inputs (input pins) of the processors at the bottom level. The flow of information is uni-directional (bottom-up). At every level all processors receive their inputs at the same time, after which they immediately output the data to the next level. If only one input word is considered, then, at any time during the computation, the processors on at most one level are active. On the other hand, if a sequence of input words is fed in, then after a while every processor may be activated, giving rise to full pipelining.

A trellis-like structure of processors has the property that data can move from one processor to another along different paths, i.e., different contexts may be used. This fact, together with the parallelism and pipelining, requires the use of special techniques to organize and synchronize the flow of information in a systolic trellis automaton.

The present paper considers systolic trellis automata as acceptors where an accepting or a rejecting signal is produced by the top processor. We hope to return, later on, to the disscussion of systolic trellis automata as transducers. (They can easily perform such operations as sorting, multiplication - see also Example 12 in Section 5.)

Exactly as in the case of systolic tree automata [6], our systolic trellis automata are actually cuts of infinite trellises satisfying a certain regularity condition. The regularity condition implies that the resulting chip can be programmed in a simple context-sensitive and top-down manner.

We also study languages accepted by trellis automata. Such a language consists of all words w accepted by the "partial" trellis obtained from the infinite trellis by cutting away all nodes beginning at the level with $|w|+1$ nodes. We shall see that systolic trellis automata are quite powerful: the acceptable languages lie quite high in traditional language - theoretical hierarchies. On the other hand, the languages are still accepted in quadratic time and linear space by deterministic Turing machines and in time n on trellis automata. Thus, we obtain also a practically interesting family of context-sensitive languages, which is rather unusual.

A couple of further points should be emphasized. All our proofs are constructive and, thus, give a method of "automatically" designing "correct" systolic arrays for a certain broad class of problems. Furthermore, if we do not require pipelining, then any program for a (homogeneous) trellis acceptor can be executed also on a linear systolic array. Alternatively, a systolic trellis automaton can be implemented as one combinational circuit (without registers). This will be further discussed in the concluding section.

A brief outline of the contents of this paper follows. Section 2 contains the basic notions concerning trellises, regularity conditions, systolic trellis automata and the systolic trellis languages. Also a normal form for systolic trellis automata is derived.

Section 3 introduces semihomogeneous and homogeneous systolic trellis automata. It is shown that given a semihomogeneous trellis automaton we can construct an equivalent homogeneous trellis automaton with only one type of processor. On the other hand, general trellis automata are strictly more powerful than (semi)homogeneous ones.

Basic programming techniques for trellis automata are introduced and investigated in Section 4. The techniques include "matching", "multiple processing" and "path automaton". These techniques are also used to establish some theoretical results: the family of languages accepted by semihomogeneous systolic trellis automata is closed under Boolean operations and contains languages not accepted by systolic tree automata.

More advanced programming techniques such as "parallel guessing" and "synchronization" will be discussed in Section 5. By these techniques, trellis automata can be programmed for surprisingly complex tasks. It also follows from the results in Section 5 that, given a linear context-free grammar G, a homogeneous trellis automaton accepting $L(G)$ can be constructed.

Finally Section 6 discusses implementations not requiring pipelining and some results contained in [3]. Those include decidablility and complexity considerations of somewhat more relaxed input conditions.

2. Systolic Trellis Automata

We now formally define the basic notions introduced in the previous section.

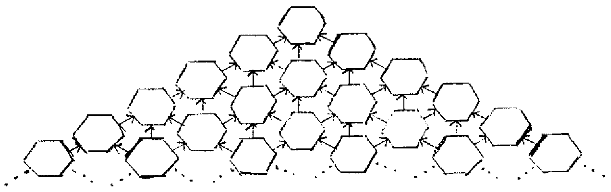
Definition 1. An infinite binary trellis (in short an infinite trellis for the rest of the paper) is an infinite oriented graph (with orientation of edges from sons to fathers) which satisfies the following conditions

(1) there is exactly one node (called the root) with no father,

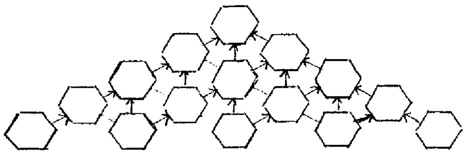
(2) every father N has three sons, the left one - N_ℓ , the middle one - N_m and the right one - N_r and always $(N_\ell)_r = N_m = (N_r)_\ell$.

An infinite labeled trellis T (see Figure 2.1a) is an infinite trellis with nodes labeled by symbols from a finite alphabet Δ_T (referred to as the alphabet of processors of T). The label of a node N in T is denoted by $\lambda_T(N)$. (A subscript T is often omitted if T is understood.) □

Trellis is a mathematical abstraction of a triangularly shaped and hexagonally connected network (see Figure 2.1).



(a)



(b)

Figure 2.1

If T is an infinite (labeled) trellis, then for $j = 1,2,\ldots$ $LEVEL_T(j)$ denotes the set of all (labeled) nodes of T whose maximal distance from the root is $j-1$. Clearly, there is a natural ordering (from left to right) of nodes in $LEVEL_T(j)$ with the first (the left-most) and the last (the rightmost) element. The i-th node in $LEVEL_T(j)$, $1 \le i \le j$, is denoted by $N_T(i,j)$ and its label $\lambda_T(i,j)$. The nodes $N_T(1,j)$ and $N_T(j,j)$, respectively are called the <u>left-leg</u> and <u>right-leg nodes</u>. All other nodes are called <u>internal</u>.

To any infinite (labeled) trellis T and to any integer $k \geq 1$ we associate the (labeled) <u>trellis</u> T_k <u>of height k</u> to be the maximal subgraph of T with the set of nodes $\bigcup_{i=1}^{k} \mathrm{LEVEL}(i)$, (see Figure 2.1b for a T_5). The nodes in LEVEL(k) in T_k are called the input-nodes or leaves of T_k .

The nodes will be interpreted as processors and then we shall use for processors similar terminology as above. For example, we shall talk about left-leg processors and so on.

In an (infinite) trellis T the edges between a node N and its left son, the right son and the middle son, respectively will be often called the left oblique, the right oblique and the vertical edge, respectively. When nodes are viewed as processors we often talk about interconnections instead of edges.

Let $\overline{T}, T$ be infinite labeled trellises and $c : \Delta_{\overline{T}} \to \Delta_T$. If $c(\lambda_{\overline{T}}(i,j)) = \lambda_T(i,j)$ for $1 \leq i \leq j$, then we write $c(\overline{T}) = T$ and say that T is obtained from $\overline{T}$ by <u>coding</u> c .

<u>Definition 2</u>. An infinite labeled trellis T is said to be (<u>strongly</u>) <u>top-down deterministic</u> if the label of any node is uniquely determined by labels of its fathers (by the label of any of its fathers). T is said to be a <u>regular</u> <u>trellis</u> if $T = c(\overline{T})$ for some top-down deterministic trellis $\overline{T}$. □

One sees easily that a trellis T is top-down deterministic if and only if for any $k > 1$ the label of any node on LEVEL(k) is uniquely determined by labels of its fathers on LEVEL(k-1) .

Example 1. The infinite trellis (see Figure 2.2a) with the root labeled by t , the left-leg (right-leg) nodes labeled by ℓ (by r), the nodes in the middle of levels (i.e. nodes $N(i, \frac{i+1}{2})$, i odd), labeled by m and with all other nodes labeled by a is not top-down deterministic. However, it is regular because it can be obtained by coding from the top-down deterministic trellis shown in Figure 2.2b.

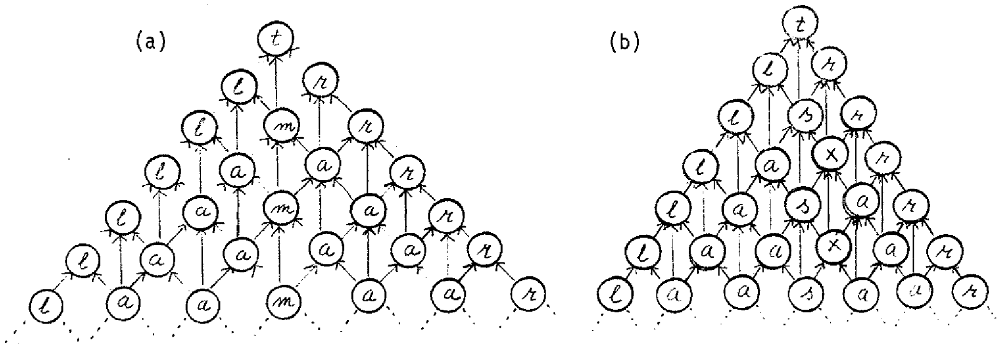


Figure 2.2

Now we are ready to define trellis automata and languages.

Definition 3. A systolic binary trellis automaton (in short trellis automaton) is a construct

$$K = (P, \Delta, \Sigma, \Gamma, \Gamma_0, f, g)$$

where Δ , Σ and Γ are finite alphabets referred to as the labeling, terminal and operating alphabets, respectively and $\Gamma_0 \subseteq \Gamma$ is the alphabet of accepting symbols. It is also assumed that $\Gamma - \Sigma$ contains the special symbol e . Moreover $P = (\ell, c)$ where

$$\ell : (\Delta \cup \{\#\}) \times (\Delta \cup \{\#\}) \to \Delta \quad \text{and} \quad c : \Delta \to \Delta$$

(we assume $\# \notin \Delta$) are referred to as the labeling function and coding, respectively. Finally

$f : \Delta \times \Sigma \to \Gamma$ is the input function

$g : \Delta \times \Gamma \times \Gamma \times \Gamma \to \Gamma$ is the transition function.

□

With every trellis automaton $K = ((\ell,c), \Delta, \Sigma, \Gamma, \Gamma_0, f, g)$ we can associate a regular trellis T (the so called underlying trellis of K) where $T = c(T')$ and T' is the trellis the labeling of which is recursively defined by the rules

$$\lambda(N(1,1)) = \ell(\#,\#)$$
$$\lambda(N(1,j)) = \ell(\#,\lambda(N(1,j-1))) \qquad \text{if } j > 1$$
$$\lambda(N(j,j)) = \ell(\lambda(N(j-1,j-1)),\#) \qquad \text{if } j > 1$$
$$\lambda(N(i,j)) = \ell(\lambda(N(i-1,j-1)),\lambda(N(i-1,j))) \qquad \text{if } 1 < i < j .$$

It is also clear that every regular trellis can be specified by a labeling function and by a coding.)

Quite often when a trellis automaton K is considered we shall refer to the underlying trellis of K but not to its labeling function or to its coding. In such cases we shall usually consider that K is given in the form

$$K = (T, \Delta, \Sigma, \Gamma, \Gamma_0, f, g)$$

where T is a regular trellis with nodes labeled by elements from Δ

(called also labels or processor names) and Δ , Σ , Γ , Γ_0 , f and g has the same meaning as in the Definition 3.

A node of a trellis labeled by a label b will be interpreted as an b-processor which computes the input function $f_b(x) = f(b,x)$ and the transition function $g_b(x,y,z) = g(b,x,y,z)$.

Now we proceed to define how a word is recognised by a trellis automaton $K = (T, \Delta, \Sigma, \Gamma, \Gamma_0, f, g)$.

Informally, we assume that every processor has three internal input pins (to receive operating alphabet symbols) and one external input pin (to receive a terminal alphabet symbol).

An input word w of length n is fed, symbol by symbol, to the external input pins of processors on the level n . (We may visualise this as if we are using a chip for the subtrellis of T of height n .) They compute in parallel values of their input functions and send results to their fathers. The outputs sent out along obliques reach fathers in one time unit, the outputs sent out along vertical outputs in two time units. At any level $j < n$ all processors receive inputs on their internal input pins at the same time. They compute in parallel the values of the corresponding transition functions and again send the results along output interconnections. (Here we assume that processors on the level $n-1$ receive the special input symbol e on the vertical internal inputs. At the end of this section we show that we can avoid this convention.)

If a word w is fed to K, then in time $n-1$ the processor at the root is activated and w is accepted if and only if its output is in Γ_0.

Formally, we proceed as follow. Let $w \in \Sigma^+$ be of length n, i.e. $w = w_1 w_2 \ldots w_n$, $w_k \in \Sigma$, $1 \le k \le n$. For $1 \le i \le j \le n$ define $w^{(i,j)} = w_i w_{i+1} \ldots w_{n-j+i}$.

Observe that if w is fed to K, then those processors on the level n which are in the leaves of the subtrellis with the node $N(i,j)$ as the root, receive just the word $w^{(i,j)}$ (see Figure 2.3).

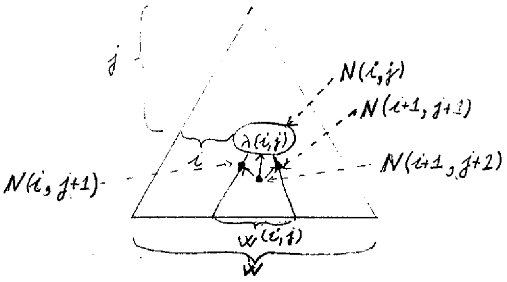


Figure 2.3

w determines a unique element of Γ, denoted by $\text{OUTPUT}(K,w,1,1)$, as follows. First we define for $1 \le i \le n$

$\text{OUTPUT}(K,w,i,n) = f_{\lambda(i,n)}(w_i)$

and for $1 \le i \le j < n$

$$\text{OUTPUT}(K,w,i,j) = g_{\lambda(i,j)}(\text{OUTPUT}(K,w,i,j+1), \text{OUTPUT}(K,w,i+1,j+2), \text{OUTPUT}(K,w,i+1,j+1))$$

(with the convention that $\text{OUTPUT}(K,w,i,n+1) = e$).

We say that a word w is accepted by K if and only if $\text{OUTPUT}(K,w,1,1) \in \Gamma_0$.

The language accepted by a trellis automaton K is defined by

$$L(K) = \{w \mid w \in \Sigma^{+} , \text{OUTPUT}(K,w,1,1) \in \Gamma_0\} \quad .$$

Languages of this form are referred to as trellis automata acceptable or trellis languages. Denote by $L(RT)$ the family of all trellis languages. (Here R refers to the fact that only trellis automata with regular trellises are considered.) Observe that no trellis language contains the empty word.

It is important to note that we can view trellis automata as follows.

An infinite unlabeled trellis represents a network of universal processors and their interconnections. A specification of a trellis automaton is actually a program for this network which attaches to any universal processor a label and feeds in this universal processor a program to compute the input function and the transition function associated with this label.

Vertical interconnections in trellis automata are exceptional in the sense that they need two time units to transmit data. They are quite often useful, as we shall see later, when transparency of specifications (of programs) for trellis automata is of concern. However, it is easy to see that no power is lost by not having vertical interconnections; the information can be transmitted via oblique outputs. This is shown in the following theorem together with the fact that we can always assume that the underlying trellis is top-down deterministic.

Theorem 1. (A normal form for trellis automata). To every trellis automaton $K = (P, \Delta, \Sigma, \Gamma, \Gamma_0, f, g)$ we can effectively construct an equivalent trellis automaton $\overline{K} = (\overline{P}, \overline{\Delta}, \Sigma, \overline{\Gamma}, \overline{\Gamma}_0, \overline{f}, \overline{g})$ such that the following conditions are satisfied.

(1) $\overline{g}(a,u,x,v) = \overline{g}(a,u,y,v)$ for any $a \in \Delta$, $u,x,y,v \in \overline{\Gamma}$.

(2) the coding in $\overline{P}$ is the identity function.

Proof. Let us assume that $P = (\lambda,c)$ and define $\overline{P} = (\lambda,\text{id})$ where id is the identity function on $\overline{\Delta}$. Moreover let $c : \overline{\Delta} \to \Delta$ and

$$\overline{\Gamma} = \Gamma \times \Gamma \cup \{e\} \qquad \overline{\Gamma}_0 = \Gamma_0 \times \Gamma$$

and for $a \in \overline{\Delta}$, $\xi \in \Sigma$, $x,y,u,v \in \Gamma$, $z \in \overline{\Gamma}$

$$\overline{f}(a,\xi) = (f(c(a),\xi),e)$$
$$\overline{g}(a,(x,y),z,(u,v)) = (g(c(a),x,y,u),u)$$

Clearly, the conditions (1) and (2) are satisfied. If we now adopt the convention, that for $w \in \Sigma^+$, $1 \le i < |w|$, $\text{OUTPUT}(K,w,i,|w|+1) = e$, then it is easy to show by induction on i and j, that for $1 \le i \le j \le n = |w|$

$$\text{OUTPUT}(\overline{K},w,i,j) = (\text{OUTPUT}(K,w,i,j),\text{OUTPUT}(K,w,i+1,j+1))$$

This immediately implies that trellis automata K and $\overline{K}$ are equivalent.

□

Theorem 1 implies, that in order to study the power of trellis automata it is sufficient to consider only trellis automata in the normal form $K = (P, \Delta, \Sigma, \Gamma, \Gamma_0, f, g)$ or

$$K = (T, \Delta, \Sigma, \Gamma, \Gamma_0, f, g)$$

where T is a top-down deterministic trellis and $g : \Delta \times \Gamma \times \Gamma \to \Gamma$. If, in addition the input functions are the identity functions we shall talk about a strong normal form. The interpretation is that in order to compute g_a , $a \in \Delta$, an a-processor takes as arguments only inputs received along obliques. It means that vertical interconnections play no fole for trellis automata in the normal form and therefore in such cases we omit them also in pictures. We can also assume that P contains only a labeling function.

We conclude this section with one methodological remark: Trellis automata are defined in such a way that every processor produces exactly one output which is immediately sent out to all its fathers (or produced as an external output in the case of the root). A specification (program) of a trellis automaton is sometimes more transparent if we assume that each processor a under the inputs (x, y, z) produces three outputs X, Y, Z, formally $g_a(x, y, z) = (X, Y, Z)$, which are sent out along the right oblique (X) , the vertical (Y) and the left oblique (Z) . (See Figure 2.4a.) (Similarly, see Figure 2.4b - in the case of trellis automata in the normal form.)

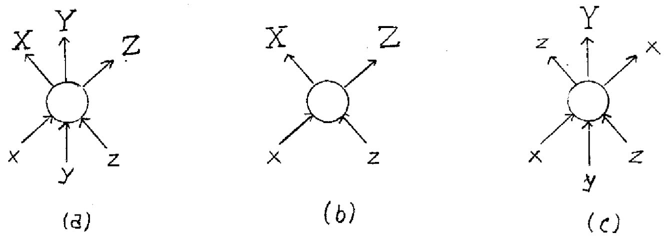


Figure 2.4

In such a case, if a processor outputs on an output oblique (vertical) the same symbol x it received on the corresponding input oblique (vertical), we say that x goes through this processor (see Figure 2.4c for the case that symbols coming along obliques go through a processor.)

3. Homogeneous Systolic Trellis Automata

Regular trellises have a labeling which can be generated in a very simple way and therefore trellis automata have fairly regular layout of processors.

In this section we introduce and study restricted types of trellis automata, the so called semihomogeneous and homogeneous trellis automata, with even a simpler labeling of underlying trellises. Even with the same processor at every node these trellis automata process data in a tricky way and accept complicated languages.

Definition 4. An infinite labeled trellis T is said to be a semihomogeneous trellis if $T = C(T')$ where T' is a strongly top-down deterministic trellis. T is a homogeneous trellis if all nodes are labeled by the same symbol. □

We shall often deal with trellis automata whose underlying trellis has the root labeled by t, the left-leg nodes by ℓ, the right-leg nodes by r and all other nodes labeled by a. It is a semihomogeneous trellis which will be called in short roof. (Trellises in Fig. 2.2 are not semihomogeneous.)

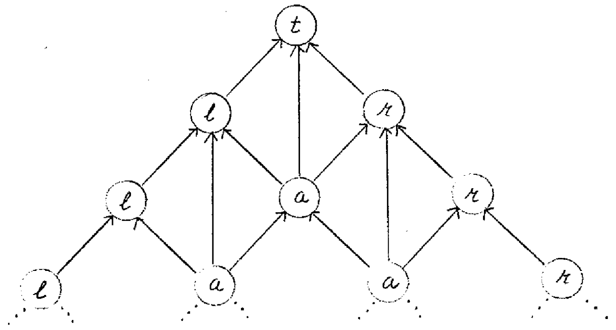


Fig. 3.1

An alternative characterisation of semihomogeneous trellises is given in the following lemma which is easy to prove.

Lemma 1: An infinite labeled subtrellis is semihomogeneous if and only if it has only a finite number of different infinite labeled subtrellises.

Definition 5. A trellis automaton is said to be semihomogeneous (homogeneous) if the underlying trellis is semihomogeneous (homogeneous). Denote by $L(ST)$ and $L(HT)$, respectively, the family of languages accepted by semihomogeneous trellis automata and homogeneous trellis automata, respectively.

In the case of homogeneous trellises a labeling of nodes is of no importance and therefore we can write a specification of a homogeneous trellis automaton shortly in the form

$$K = (\Sigma, \Gamma, \Gamma_0, f, g)$$

where $\Sigma, \Gamma, \Gamma_0, f$ and g have the usual meaning.

The following theorem shows that homogeneous trellis automata have the same power as semihomogeneous ones and, moreover, that in the case of homogeneous automata we can even assume that the input function is the identity function.

<u>Theorem 2</u>. (A normal form for semihomogeneous trellis automata) Given a semihomogeneous trellis automaton we can effectively construct an equivalent homogeneous trellis automaton, the input function of which is the identify function.

<u>Proof</u>. Let $K = (P, \Delta, \Sigma, \Gamma, \Gamma_0, f, g)$ be a semihomogeneous trellis automaton. In view of Theorem 1 we can assume that the underlying trellis T is strongly top-down deterministic and $g : \Delta \times \Gamma \times \Gamma \to \Gamma$. Without loss of generality we can also assume that $\Sigma \wedge \Gamma = \theta$ $\Delta = \{1, 2,\ldots,n\}$ and that the root of T is labeled by 1.

Now we give an informal description of a homogeneous trellis automaton $\overline{K}$ which accepts the language $L(K)$. $\overline{K}$ has the same processor in every node. Any time a processor of K is activated it simulates, in parallel, computations of all processors of K as follows. If this processor receives a symbol $t \in \Sigma$ on the external input pin, then it outputs t. If it receives two symbols $x, y \in \Sigma$ or two n-tuples $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$, then it outputs the n-tuple $(z_1, \ldots, z_n)$ where $z_i = g_i\,(f_{i_\ell}(x), f_{i_r}(y))$ or $z_i = g_i(x_{i_\ell}, y_{i_r})$ where i_ℓ and i_r, respectively are labels of the left son and the right son of the node labeled by i in T. (Remember that T is strongly top-down deterministic!) A word w is accepted by $\overline{K}$ if and only if the output n-tuple has the first component in Γ_0 (remember that the root of T is labeled by 1).

The construction of a specification of K' and the formal proof that $L(K') = L(K)$ are now straightforward. □

In view of this theorem we can always assume that a homogeneous trellis automata is given in the following strong normal form

$$S = (\Sigma, \Gamma, \Gamma_0, g)$$

Corollary. $L(HT) = L(ST)$

Homogeneous trellis automata are easier to deal with but they are not so powerful as nonsemihomogeneous ones.

<u>Theorem 3</u>. $L(HT) \subsetneq L(RT)$

<u>Proof</u>. We show that the language $L = \{a^{2^n} \mid n \geq 0\}$ is in $L(RT) - L(HT)$. Let T be the infinite regular trellis with the root, left-leg and right-leg nodes labeled by a and with all other nodes labeled according to the rules given in Fig. 3.2b (which show how labels of fathers specify the label of their common son.) Fig. 3.2a shows the labeling of nodes of the first 8 levels of T.

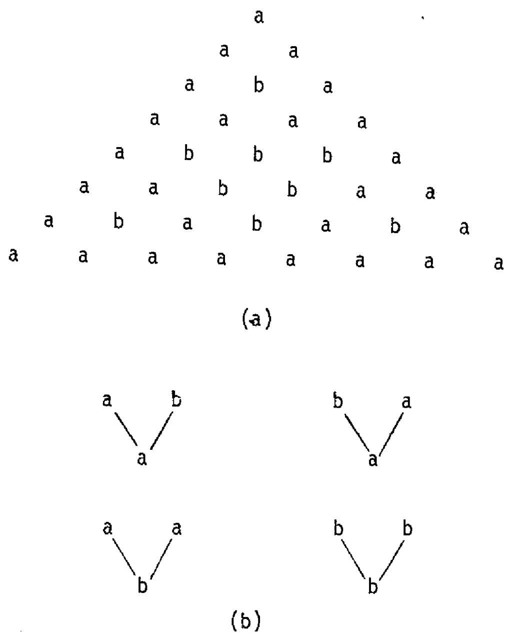
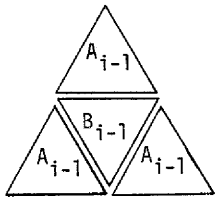


Figure 3.2

For $i = 1, 2, \ldots$ let A_i, B_i be top-down and bottom-up triangular matrices of a's and b's defined as follows: A_i is the top-down matrix (see Fig. 3.2a) formed by labels of the first 2^i rows of T and B_i is the bottom-up triangular matrix containing only b's which has $2^i - 1$ rows and $2^i - j$ b's in the j-th row.

Using mathematical induction we show that for $i > 1$, A_i has the form

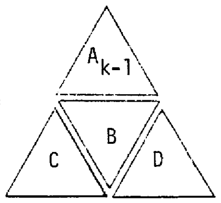


(1).

and that all bottom-row elements of A_i are a's (2^i of them).

An inspection of Fig. 3.2 shows that for $i = 2$ and $i = 3$ A_i has really such a form. Let us now assume that for a $k > 3$, A_{k-1} has form (1) and that all elements in the bottom row of A_{k-1} are a's. Then A_k has to have the form

A_k =



Since all elements in the bottom row of A_{k-1} are a's (2^{k-1} of them) B matrix must contain only b's (see Fig. 3.2b). Moreover, from the definition of A_k it follows that all left most elements in C and all right most elements in D are a's. So the top elements in C and D are a's. Since B contains only b's this implies that all right most elements in C and all left most elements in D are a's. Therefore $C = A_{k-1} = D$.

It is easy to see that the i-th row of T contains only a's if and only if $i = 2^k$ for some k.

Now we describe a trellis automaton K in the normal form which accepts the language L. The underlying trellis is the one from Fig. 3.2a, {a} is the terminal alphabet, {a,b} is the operating alphabet, a being the only accepting symbol. In the case of the external input an a-processor outputs a and a b-processor outputs b. In the case of internal outputs a processor produces a if and only if both inputs are a's.

It is now clear that if K recognises a word from $\{a\}^*$, then the root outputs a if and only if during the recognition process no processor outputs b, i.e., if and only if the input word has the form a^{2^k}, $k \geq 1$. Therefore $L(K) = L$.

The proof that L is not in *L(HT)* is by contradiction.

Let us assume that L is in *L(HT)*. Then there exists a homogeneous trellis automation K in the normal form which accepts L. {a} is the terminal alphabet of K and let its operating alphabet have k symbols. Let us choose a fixed n such that $n > k$ and $2^n - 2^{n-1} > k$.

Since K is homogeneous, for an input word $w = a^{2^n}$ and for any $0 \le i \le 2^n$. OUTPUT$(K,w,i) = b_i^i$ where b_i is a symbol in the operating alphabet of K. Since $n > k$, there must exist $0 \le i_1 \le i_2 \le 2^n$ such that $i_2 - i_1 \le k$ and $b_{i_2} = b_{i_1}$. This implies that OUTPUT$(K,a^{2^n},1,1)$ = OUTPUT$(K,a^{2^n-(i_2-i_1)},1,1)$. But $2^{n-1} < 2^n - (i_2-i_1) < 2^n$ and therefore $a^{2^n-(i_2-i_1)} \notin L$. This contradiction completes the proof of the theorem. $\square$

Remark 1: The infinite trellis defined in Fig. 3.1 is an example of an infinite regular trellis all infinite subtrellises of which are distinct.

Remark 2: Note that the regularity condition for the labeling of trellises is quite essential. If unrestricted labeling is allowed then every language over a one-letter alphabet becomes acceptable by a trellis automaton. On the other hand it can be shown in a similar way as in the proof of Theorem 3, that the family of languages over a one-letter alphabet which are acceptable by homogeneous trellis automata equals the family of regular languages over this one-letter alphabet.

4. Elementary Programming Techniques

Systolic trellis automata have several features which require the use of special techniques to program them. In this section we discuss several such simple techniques. In order to illustrate these techniques we use them to show how to design trellis automata to accept some important languages. We use also these techniques to derive some more general results.

Matching. (Context-free matching). The basic technique to deal with a remote context is to keep sending data along obliques till they properly match.

Example 2. Denote by D the Dyck language over the alphabet $\Sigma = \{(,),[,]\}$ (i.e., the language generated by the context free grammar $S \to (S) \mid [S] \mid SS \mid () \mid [].)$

We shall construct a semihomogeneous trellis automaton K in the normal form which accepts D. The underlying trellis will be the roof (see Figure 3.1). $\{(,), [,], Y, N\}$ is the operating alphabet of K with Y as the only accepting symbol. The input function of any processor is the identity function and the processors are driven by transition functions shown in Figure 4.1.

Informally, processors of K keep transmitting left parantheses (and [along left obliques and right parentheses) and] along right obliques till they either match properly {and then the corresponding processor outputs Y} or improperly {and N is outputed} or they hint a leg-node processor without being matched (and again N is produced).

x \ Y	(	)	[	]	Y	N
(	(	Y	(	N	(	N
)	Y	)	Y	]	Y	N
[	[	N	[	Y	[	N
]	Y	)	Y	]	Y	N
Y	Y	)	Y	]	Y	N
N	N	N	N	N	N	N

g_a

x \ Y	(	)	[	]	Y	N
(	N	Y	N	N	N	N
)	N	N	N	N	N	N
[	N	N	N	Y	N	N
]	N	N	N	N	N	N
Y	N	N	N	N	Y	N
N	N	N	N	N	N	N

g_t

x \ Y	(	)	[	]	Y	N
(	(	Y	(	N	(	N
)	N	N	N	N	N	N
[	[	N	[	Y	[	N
]	N	N	N	N	N	N
Y	Y	)	Y	]	Y	N
N	N	N	N	N	N	N

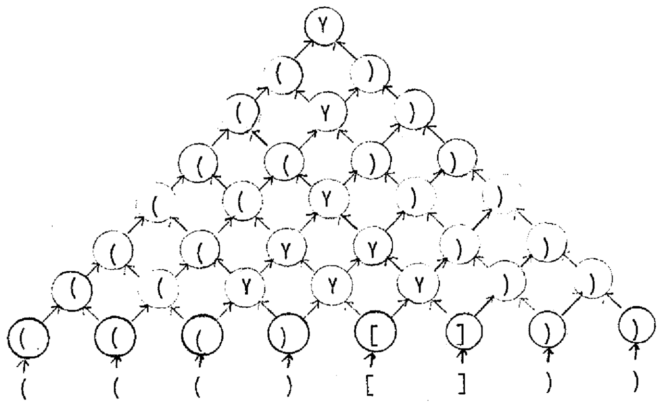
g_ℓ

x \ Y	(	)	[	]	Y	N
(	N	Y	N	N	N	N
)	N	)	N	]	Y	N
[	N	N	N	Y	N	N
]	N	)	N	]	Y	N
Y	N	)	N	]	Y	N
N	N	N	N	N	N	N

g_r

Figure 4.1

The Y outputs are consequently absorbed but the outputs N are propagating i.e., once an N is produced, then the final output symbol is N . Therefore a word is accepted by K if and only if all parentheses are properly matched. Figure 4.2 shows the processing of two words. The symbols written in the nodes are the outputs of the corresponding processors.



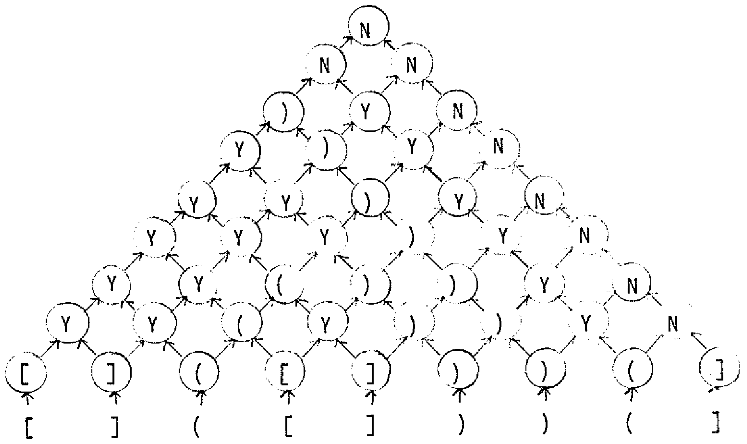


Figure 4.2

In order to prove that K accepts D it is sufficient to show that for $w \in \Sigma^+$, $1 \leq i \leq j \leq |w|$ and $z \in \{(,), [,], Y\}$, the following assertion (A) holds.

(A) OUTPUT$(K, w, i, j) = z$ if and only if $w^{(i,j)}$ has the form shown in the entries of the table in Figure 4.3 (where $\bar{d}$ is either the empty word or a word in D, d is a word in D, p is a prefix of a word in D and s is a suffix of a word in D.)

This assertion can be easily proven by induction on i and j (with the help of tables in Fig. 4.1).

OUTPUT(K,w,i,j) / λ(i,y)	(	)	[	]	Y
a	(p	s)	[p	s]	sdp
ℓ	(p	$\bar{d}$)	[p	$\bar{d}$]	$\bar{d}$p
r	($\bar{d}$	s)	[$\bar{d}$	s]	s$\bar{d}$
t	-	-	-	-	$\bar{d}$

Dash indicates an impossible combination.

Fig. 4.3

It has been shown in [6] that there is no systolic tree automaton to accept the Dyck language. Therefore we have obtained the following result.

Lemma 2. Systolic trellis automata accept some languages not acceptable by systolic tree automata.

Multiple processing. Quite a complicated task can be done by a trellis automaton which simulates in parallel several simpler trellis automata and at the end the processor at the root combines in a proper way the outputs of all simulated automata.

The main use of this technique is actually captured by the following theorem.

Theorem 4. The families of trellis languages and of homogeneous trellis languages are effectively closed under boolean operations and reversal.

This theorem can be proven easily in a similar way as Theorem 3 in [6]. One has only to realise that if T_1, T_2 and T are infinite trellises such that $\lambda_T(i,j) = (\lambda_{T_1}(i,j), \lambda_{T_2}(i,j))$, then T is regular (semihomogeneous) if T_1 and T_2 are regular (semihomogeneous).

Example 3. The language $L = \{a^n b^n \mid n \geq 1\}$ is in $\mathcal{L}(HT)$. Indeed, $L = D \wedge L_0$ where D is the Dyck language over the alphabet $\{(,),a,b\}$ and $L_0 = \{a^i b^i \mid i, j \geq 1\}$. $L_0 \in \mathcal{L}(HT)$ can be shown easily using the matching technique.

Path automaton. (Simulation of a finite automaton on a path). All processors of a trellis automaton are combinational circuits, i.e., they have no memory. In spite of that they can be used to activate and to simulate behaviour of a finite automaton (or several of them), which, once activated by a processor, climbs the trellis along a path till it either stops or reaches the root.

Indeed, processors of a trellis automaton can be designed in such a way, that if they receive, encoded in their input symbols, a state and an input symbol of a finite automaton A, they compute the new state and the output symbol of A and send them up along some output interconnections.

A finite automaton can be activated by a trellis automaton in many different ways. For example by letting a special symbol reach a particular processor or a processor receive on its inputs a special combination of symbols.

Simulation of a finite automaton on a path is a very powerful technique and we shall make quite an intensive use of this technique in the rest of the paper. To start with we illustrate this technique by a simple example.

Example 4. We show that the language $L = \{w \$ w^R \mid w \in \Sigma^+, \$ \notin \Sigma\}$ is in $L(HT)$. Let M be a finite state acceptor with two states $\$$ and N, with the input alphabet $\{(a,b) \mid a,b \in \Sigma \cup \{e\}\}$ and the transition function specified in Fig. 4.4.

Now let K by a trellis automaton with the roof as the underlying trellis, $\Sigma \cup \{\$, e, N\}$ as the operating alphabet and $\$$ as the only accepting symbol, which behaves as follows. (See Fig. 4.5.)

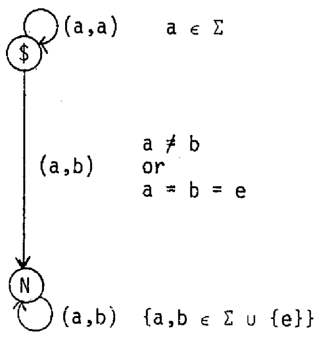


Figure 4.4

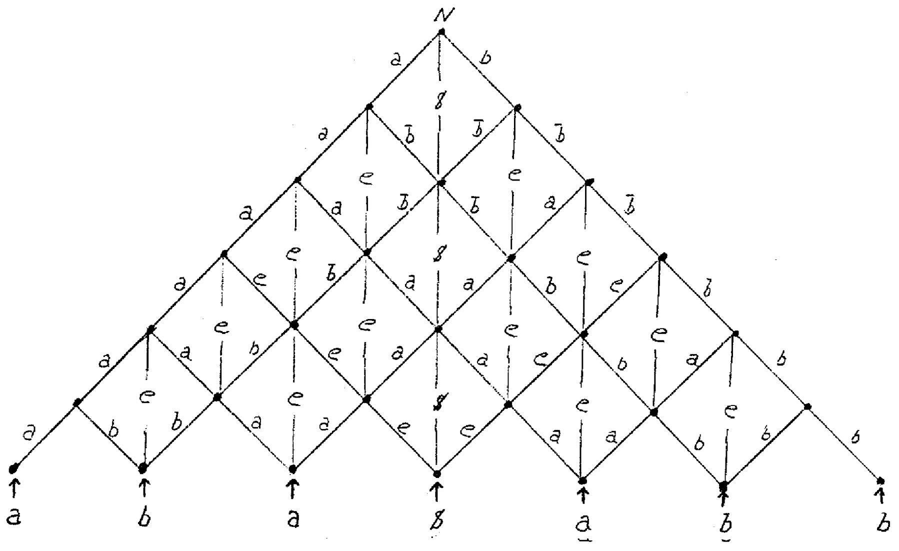


Figure 4.5

If the symbol $ is received on the external input of a processor, then this processor activates M and since then on "M climbs up along vertical edges". In every processor on this route M receives as the input a pair of symbols coming to the processor along obliques. This input changes the state of M and the processor sends M (in this new state) up along the vertical edge.

If an $a \in \Sigma$ is received on the external input of a processor, then this processor outputs a along all available output obliques and it outputs the symbol e along the vertical output. All processors except leg-node processors, let symbols coming along obliques go through the processor. The leg-node processors send e along the output obliques, if they receive $ or N on an input.

It is now easy to see that K outputs \$ if and only if the input word is in L.

A very simple use of the path automaton technique is actually in behind the constructions used to show the following result.

<u>Theorem 5</u>. Given two homogeneous trellis automata $K_1 = (\Sigma_1, \Gamma_1, \Gamma_{1,0}, g_1)$ and $K_2 = (\Sigma_2, \Gamma_2, \Gamma_{2,0}, g_2)$ and a symbol $\$ \notin \Sigma_1 \cup \Sigma_2$, we can effectively construct homogeneous trellis automata K, $\overline{K}$ and $\overline{\overline{K}}$ such that

(1) if $\Sigma_1 \wedge \Sigma_2 = \theta$ then $L(K) = L(K_1).L(K_2)$

(2) $L(\overline{K}) = L(K_1)\ \$\ L(K_2)$

(3) $L(\overline{\overline{K}}) = (L(K_1)\$)^+$

<u>Proof</u>. In order to show (1) it is sufficient to show how to construct a semihomogeneous trellis automaton K if $\Sigma_1 \wedge \Sigma_2 = \theta$. This we shall do now. In a similar way semihomogeneous trellis automata $\overline{K}$ and $\overline{\overline{K}}$ can be constructed.

We can assume without loss of generality that $\Gamma_1 \cap \Gamma_2 = \theta$ and that Y, N are two symbols not in $\Gamma_1 \cup \Gamma_2$. Let us define $K = (P, \Delta, \Sigma_1 \vee \Sigma_2, \Gamma, \Gamma_0, f, g)$ as follows.

Let $P = (\hat{\ell}, \mathrm{id})$ where $\hat{\ell}$ is the labeling function of the roof. Thus $\Delta = \{t, \ell, \pi, a\}$. Let $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \{Y,N\}$ and $\Gamma_0 = \{Y\}$ and let f_S, $S \in \Delta$ be the identity function on $\Sigma_1 \cup \Sigma_2$.

The transition function g is defined in Figure 4.6 for the case that no arguments equals N and its value is N if the value of at least one argument is N. In Figure 4.6 ξ_1 and η_1 denote arbitrary elements in Γ_1 and ξ_2, η_2 arbitrary elements in Γ_2. Moreover

g_S, $S \in \Delta$ / (x, y)	g_a	g_t	g_ℓ	g_r
(ξ_1, η_1)	$g_1(\xi_1, \eta_1)$	N	$g_1(\xi_1, \eta_1)$	N
(ξ_2, η_2)	$g_2(\xi_2, \eta_2)$	N	N	$g_2(\xi_2, \eta_2)$
(ξ_1, ξ_2)	Y	Z	Z_1	Z_2
(ξ_2, ξ_1)	N	N	N	N
(ξ_1, Y)	Y	Z_1	Z_1	N
(Y, ξ_2)	Y	Z_2	N	Z_2
(Y, ξ_1)	N	N	N	N
(ξ_2, Y)	N	N	N	N

Figure 4.6

z_1 = if $\xi_1 \in \Gamma_{1,0}$ then Y else N

z_2 = if $\xi_2 \in \Gamma_{2,0}$ then Y else N

z = if $\xi_1 \in \Gamma_{1,0}$ and $\xi_2 \in \Gamma_{2,0}$ then 1 else 0

To show that K accepts $L(K_1) . L(K_2)$ we proceed as follows: First observe that if $w \in (\Sigma_1 \cup \Sigma_2)^+ - \Sigma_1^+ \Sigma_2^+$ and $|w| = n$, then when w is being recognised by K, at least one of the processors at the level $n - 1$ produces N and therefore $w \notin L(K)$.

Let us therefore assume that $w = w_1 w_2$ with $w_1 \in \Sigma_1^+$ and $w_2 \in \Sigma_2^+$. Let T be the underlying trellis of K_1 i.e., the roof. Let $T^{(1)}$ and $T^{(2)}$ be those subtrellises of T (see Fig. 4.7) which have as their leaves exactly those leaves of T_n to which w_1 and w_2 are fed when w is fed to K.

When w is being recognised by K, then the processors in the nodes of T_n produce the following outputs (see Fig. 4.7).

The processors in the nodes of the subtrellises $T^{(1)}$ and $T^{(2)}$ produce the same outputs as the processors in the corresponding nodes of K_1 and K_2 when w_1 and w_2 is being recognised by K_1 and K_2, respectively.

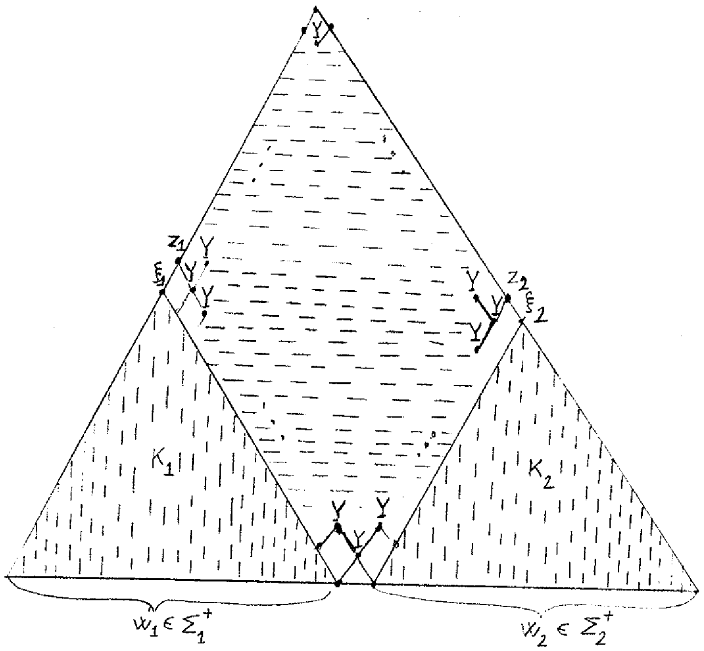


Figure 4.7

The processor in the left-leg (the right-leg) node of T_n, the left son (the right son) of which is the root of $T^{(1)}$ (of $T^{(2)}$), produces Y or N depending if the output produced by its left son (right son) is in $\Gamma_{1,0}$ (in $\Gamma_{2,0}$) or not.

All other processors produce Y or N and they produce N if and only if one of their input symbols is N .

Therefore, the root produces Y as the output symbol if and only if $w_1 \in L(K_1)$ and $w_2 \in L(K_2)$. □

<u>Corollary</u>. The family $\mathcal{L}(HT)$ is closed under marked concatenation and marked star.

<u>Example 5</u>. The language $L = \{a^n b^n c^n \mid n \geq 1\}$ is in $\mathcal{L}(HT)$. Indeed, $L = (\{a^n b^n \mid n \geq 1\} \cdot \{c\}^+) \cap (\{a\}^+ \cdot \{b^n c^n \mid n \geq 1\})$ and Theorems 4 and 5 imply that $L \in \mathcal{L}(HT)$.

<u>Example 6</u>. Let K be a homogeneous trellis automaton and R a regular language. It is easy to construct a semihomogeneous trellis automaton K' all processors of which simulate the corresponding processors of K and, moreover, K' simulates in its left-leg nodes the behaviour of an acceptor for R which interprets the output of a left-leg processor as 1 or 0 depending if this output is an accepting symbol of K or not. In such a way K' accepts the language

$$L = \{y \mid y \in \Sigma^+ \text{ and there is } z \in R \text{ such that } |z| = |y| \text{ and for } 1 \leq i \leq n \text{ the i-th symbol of } z \text{ is } 1 \text{ (is } 0) \text{ if the prefix of } y \text{ of length } i \text{ is in } L(K) \text{ (is not in } L(K))\}$$

5. Advanced Programming Techniques.

Two techniques are discussed in this section. They make an essential use of the facts that trellis automata process data in a parallel way and data flow is completely synchronized.

Parallel guessing. Discussing subroutine initiation technique we saw how separators can be utilisied. Their role seems to be often crucial, for example in the case of the trellis automaton in Section 4 which recognizes the language $\{w \$ w^R \mid w \in \Sigma^+, \$ \notin \Sigma\}$.

In this section we first establish the somehow surprising fact that also the language $\{ww^R \mid w \in \Sigma^+\}$ is recognisable by a homogeneous trellis automaton. This automaton is designed in such a way that when an input word w is recognized, all processors at the level $|w|-1$ assume (guess) in parallel that they are just in the middle of the row and, similarly as in the case of a trellis automaton recognizing the language $\{w \$ w^R; w \in \Sigma^+\}$, they send up a finite automaton checking in every encountered processor if its input signals along obliques are the same.

Parallel guessing technique consists in guessing in parallel a certain finite context (but maybe unboundedly remote) and processing data in parallel under the assumption that all guesses were correct till no contradiction is found. Of course, in order to be able to do it the amount of guesses each processor has to keep track of has to be bounded.

We proceed now to illustrate this technique.

<u>Example 7</u>. We design a homogeneous trellis automaton K which accepts the language $L = \{ww^R \mid w \in \Sigma^+\}$. Let $\Sigma \cup \{e,N\}$ be the operating alphabet of K with e , N not in Σ and e the only accepting symbol. The input and the transition function are defined as follows:

(In order to gain transparency we assume that each processor outputs three symbols which are sent out along three outputs edges.)

$$f(t) = (t,N,t) \qquad t \in \Sigma$$

$$g(t_1,N,t_2) = (t_2,N,t_1) \qquad t_1,t_2 \in \Sigma$$

$$g(t_1,e,t_2) = \text{if } t_1 = t_2 \text{ then } (t_2,e,t_1) \text{ else } (t_2,N,t_1) \ .$$

Informally, the symbol e plays here the role of the separator which is assumed to be wherever possible at the beginning. Each e moves up the trellis along verticals as long as it keeps meeting pairs of equal symbols (see Figure 5.1).

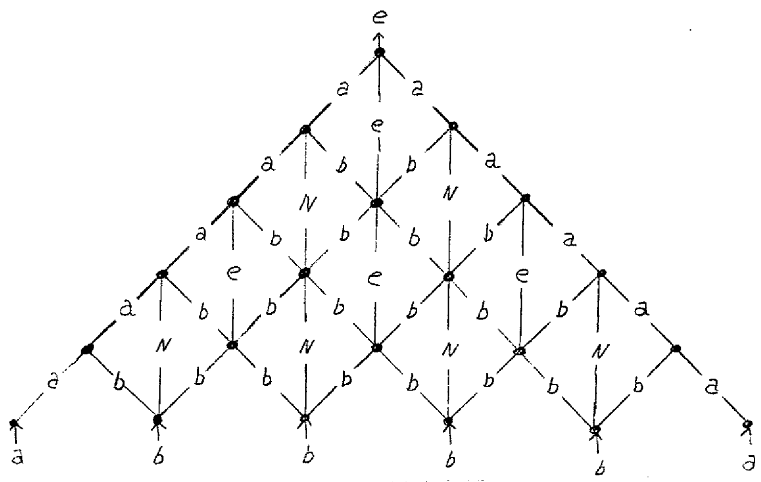


Figure 5.1

In order to show formally that K accepts the language L it is sufficient to prove (by induction) that the following assestion (A) holds for any $w \in \Sigma^+$, $1 \leq i \leq j \leq |w|$

(A) $\lambda(i,j)$ - processor outputs the symbol e along its vertical output if and only if $w^{(i,j)} = w_1 w_1^R$ for some $w_1 \in \Sigma^+$. □

As an easy modification of the Example 7 we can also construct a homogeneous trellis automaton accepting the language $c^*\{ww^R \mid w \in \Sigma^+ \text{ and } c \notin \Sigma\}c^*$.

These examples are special cases of the following rather surprising result, the proof of which is based on the parallel guessing technique.

Lemma 3. If τ is a rational transduction, then the language $L = \{u^R v \mid (u,v) \in \tau,\ u^R v \neq e\}$ is in $\mathcal{L}(HT)$.

Proof. Let $\mathcal{T}$ be a rational tranduction. Then there is a rational transducer (in a normal form)

$$R = (Q, \Sigma, \nu, Q_0, F)$$

where Q is the set of states, $Q_0 \subseteq Q$ and $F \subseteq Q$ are the sets of initial and final states, Σ is the input and output alphabet and

$$\nu \subseteq (Q \times \Sigma \times \{\varepsilon\} \times Q) \cup (Q \times \{\varepsilon\} \times \Sigma \times Q)$$

such that $\mathcal{T}$ is realised by R.

A homogeneous trellis automaton K which accepts L is now constructed as follows.

internal alphabet: $\Gamma = \Sigma \times 2^Q \times \Sigma$

accepting symbols $\Gamma_0 = \{(a,M,b) \mid a,b \in \Sigma\ ,\ M \subseteq Q$ and $M \cap F \neq \theta\} \subseteq \Gamma$

input function $f : \Sigma \to \Gamma$ where for all $a \in \Sigma$

$$f(a) = (a,M,a) \text{ with } M = \left\{q \mid (p,\varepsilon,a,q) \in \nu \text{ or } (p,a,\varepsilon,q) \in \nu \text{ for some } p \in Q_0\right\}$$

transition function $g : \Gamma \times \Gamma \to \Gamma,\ g = (\overleftarrow{g},\overline{g},\overrightarrow{g})$ where

$$\left.\begin{array}{l}\overleftarrow{g}((a,M,b),(c,N,d)) = a\\ \overrightarrow{g}((a,M,b),(c,N,d)) = d\end{array}\right\} \quad \begin{array}{l}a,b,c,d \in \Sigma\\ M,N \subseteq Q\end{array}$$

$$\overline{g}((a,M,b),(c,N,d)) = \{g \in Q \mid (p,\varepsilon,d,q) \in \nu \text{ for some } p \in M\} \cup \{q \in Q \mid (p,a,\varepsilon,q) \in \nu \text{ for some } p \in N\}$$

Informally, K works as follows. Initially K makes all possible guesses for the position of an imaginary marker in the input word w which decomposes $w = u\overset{R}{}v$ in such a way that v is a possible output of R under the input u . For any initially guessed decomposition $u\overset{R}{}v$ K simulates R on the input u and checks if v is a possible output of R . The input w is accepted if there is a correct guess. (This shows that systolic trellis automata, which are deterministic automata, can simulate in real time locally bounded but globally unbounded nondeterminism.)

To show formally that L(K) = L it is sufficient to prove (what can be easily done by induction) that the following assertion holds for any $w \in \Sigma^+$ and $1 \leq i \leq j \leq |w|$

$\text{OUTPUT}(K,w,i,j) = (a,M,b)$ where a is the first and b is the last symbol of $w^{(i,j)}$ and

(A) $M = \Big\{q \in Q \mid$ there are $u_1,v_1,\ldots,u_k,v_k \in \Sigma \cup \{e\}$ and $q_0,q_1,\ldots,q_k \in Q$ such that $w^{(i,j)} = u_k u_{k-1} \cdots v_2 u_1 v_1 v_2 \cdots v_{k-1} v_k$, $q_0 \in Q_0$ and $(q_{s-1},u_s,v_s,q_s) \in \nu$ for $s = 1,2,\ldots,k\Big\}$.

□

Before we state the following theorem,which is the main result of this section,we would like to remind that trellis languages do not contain the empty word,so all equalities of languages are considered modulo ε .

Theorem 6. Any linear (context-free) language is accepted by a homogeneous trellis automaton.

Proof. It is well known [1], that L is a linear language if and only if $L = \{u^R v \mid (u,v) \in T\}$ for a rational tranduction T . Now the theorem follows from Lemma 3.

□

Corollary. The family of homogeneous trellis languages contains the Boolean closure of linear languages.

Corollary. The families of homogeneous trellis languages and of trellis languages contain regular languages and are closed under intersection with regular languages.

Example 8. Theorem 6 has the following rather interesting application. Assume that you want to test whether or not one of the key words $X_1,\ldots,X_k$ appears in a text w . This amounts to accept all words of the form

$$X_1^R\#X_2^R\#\ldots\#X_k^R\#w \quad ,$$

where w can be written as

$$w = uX_iv \quad ,$$

for some u and v and i , $1 \leqq i \leqq k$. Reverse X_i^R is considered here only because, otherwise, the language is not linear context-free. But words of this form constitute a linear language - independently whether we consider a fixed k or a variable k . Hence, Theorem 6 tells us that key word testing can be accomplished by a trellis automaton. Moreover, for words of bounded size, one chip of an appropriate size is sufficient.

Synchronization. Quite often it is useful to construct a trellis automaton in such a way that it does in parallel two things. First, in all processors some computation is done. Secondly, it keeps "moving input data, or some other data along obliques or verticals through processors till they are (again) needed." This idea is behind of the matching technique and in such a way trellis automata accepting Dyck language and palindromes were constructed.

This technique can be generalized in the following way. A trellis automaton can not only keep moving data along some obliques or verticals but it can also change directions in which data "climb the trellis". By changing these directions in proper moments the data movement can be synchronized to achieve that proper data, often from quite remote sources, arrive to inputs of some processors.

In the rest of this section this technique is illustrated by several examples.

Example 9. The language $L = \{w \$ w \mid w \in \Sigma, \$ \notin \Sigma\}$ is in $L(HT)$.

Since the language $L_1 = \{w_1 \$ w_2 \mid w_1, w_2 \in \Sigma, |w_1| = |w_2|\}$ is linear, it is sufficient to show, in view of Theorems 4 and 6, that there is a homogeneous trellis automaton K such that K accepts a word $w = w_1 \$ w_2$, $w_1, w_2 \in \Sigma^+$, $|w_1| = |w_2|$, if and only if $w_1 = w_2$.

This seems to be a very simple task. For a given word $w = w_1 \$ w_2$ with w_1 and w_2 in Σ , $|w_1| = |w_2| = n$, K simply will keep sending symbols from w_1 (from w_2) along the left (right) obliques and the processors at the level n simply compare their inputs (see Figure 5.2a). (The i-th processors on the level n receives as inputs the i-th symbol of w_1 and the i-th symbols of w_2.) K accepts $w_1 \$ w_2$ if and only if every processors on the level n receives on both inputs equal symbols.

However, this does not work. There is no way to achieve, that for any word $w_1 \$ w_2$, $|w_1| = |w_2| = n$ all processors at the level n behave differently than processors at other levels.

K has therefore to be constructed in a different way and we show now how it can be done using our synchronization technique (see Figure 5.2b).

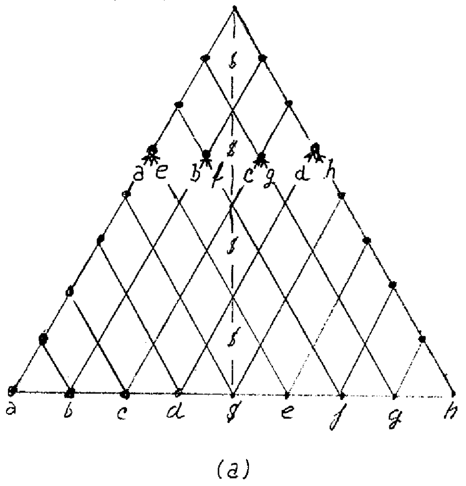
Let $\overline{\Sigma}$ denote the alphabet disjoint from Σ consisting of "barred" symbols, $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$.

K will have two kinds of processors. R-processors in the right-leg nodes and A-processors in all other nodes. Informally, the processors of K work as follows.

(1) A-processors. If the external input is $\$$, then this symbol is sent out along the right oblique and the empty symbol is sent out along other outputs. If an external input is from Σ , then this symbol is sent out along all output obliques and e is sent out along the vertical output.

In the case of three internal inputs, an A-processor lets an input symbol to pass through with the following exceptions. If a processor receives a symbol $\xi \in \Sigma$ along the left oblique and the symbol $\$$ along the right oblique, then it outputs e along the right oblique and the symbol $\overline{\xi}$ along the left oblique. If a processor receives a symbol $\overline{\xi}$ along the left oblique and a symbol $\eta \in \Sigma$ along the right oblique, then it sends $\overline{\xi}$ along the left oblique, e along the right oblique and η along the vertical output.

(2) R-processors. If a $\xi \in \Sigma$ is received on the external input then ξ is sent up along the right oblique. In the case of three internal inputs from $\Sigma \cup \{e\}$, an R-processor outputs the symbol coming along the right oblique. If a processor receives a $\overline{\xi} \in \overline{\Sigma}$ on its left oblique and an $\eta \in \Sigma$ on its right oblique, then it outputs Y if $\xi = \eta$ and N if $\xi \neq \eta$. If an R-processor receives N on one of its inputs, then it sends up N . Finally, if an R processor receives Y on its right oblique, then it sends out Y or N depending whether the input symbols on other inputs are equal or not. Y is the only accepting symbol of K .



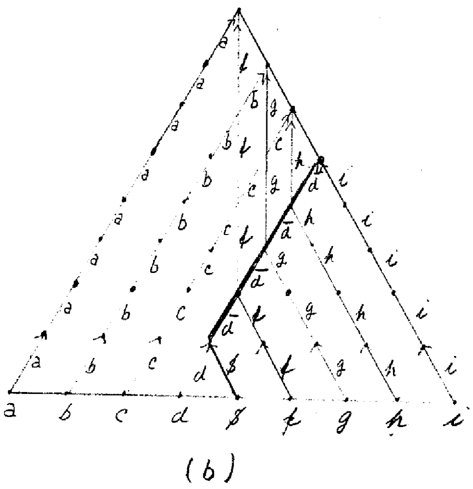


Figure 5.2

We can say that when a $\bar{\xi} \in \bar{\Sigma}$ hits an R-processor, then a finite automaton A is activated. A then moves along the right-leg edges and in each encountered processor A compares inputs on the left oblique and on the vertical.

Formal construction of K is now quite straightforward.

<u>Example 10</u>. The language $L_1 = \{ww \mid w \in \Sigma^+\}$ is in $L(RT)$. Indeed, a trellis automaton K_1 accepting L_1 can be constructed in a similar way as the automaton K in the Example 9. The underlying trellis of K_1 will be the trellis shown in Figure 2.2b, i.e. the roof with the nodes just in the middle of rows labeled by m. First time an m-processor receives an input it sends out the symbol \$ along its left output oblique. The rest of the construction of K_1 is similar to the construction of K in the Example 9.

It seems that the language $\{ww \mid w \in \Sigma^+\}$ is not in $L(HT)$, however, we have no proof of this.

We conclude this section with several examples of languages which can be shown to be homogeneous trellis languages using the synchronization techniques. These examples show both the power of homogeneous trellis automata and the power of our synchronization technique.

Example 11. Let $\Sigma = \{0,1\}$. Let $\rho : \Sigma \times \Sigma \to \Sigma$ be a binary operation on Σ. For $x,y \in \Sigma^+$, $|x| = |y| = n$, let $x\rho y = \rho(x_1,y_1)\rho(x_2,y_2)\ldots\rho(x_n,y_n)$. The language $L = \{x \$ y \$ z \mid x,y,z \in \Sigma^+, |x| = |y|, z = x\rho y\}$ is in $L(HT)$.

Example 12. Let $\Sigma = \{0,1\}$. The languages
$L_1 = \{x \$ y \$ z \mid x,y,z \in \Sigma^+, |x| = |y|, \perp z = \perp x + \perp y\}$ and
$L_2 = \{x \$ y \$ z \mid x,y,z \in \Sigma^+, |x| = |y|, \perp z = (\perp x)*(\perp y)\}$ are in $L(HT)$.1)

Example 13. Let $\Sigma = \{a\}$. The languages $L_1 = \{a^i \$ a^j \$ a^k \mid j = i+k\}$ and $L_2 = \{a^i \$ a^j \$ a^k \$ a^\ell \mid i+k = j+\ell\}$ are in $L(HT)$.

Example 14. Let Σ be an alphabet and $\$ \notin \Sigma$. The languages $L_2 = \{w \$ w^R \mid w \in \Sigma^+\}^2$ and $L_3 = \{w \$ w^R \mid w \in \Sigma^+\}^3$ are in $L(HT)$.

To show that L_2 is in $L(HT)$ is quite easy. The case of L_3 is more tricky. One can make a use here of the fact that the language L_2 from the Example 13 is in $L(HT)$.

1) For $x \in \Sigma^+$, $\perp x$ denotes an integer represented in binary by x.

Example 15. The following language L over the alphabet $\{0,1,\#\}$ is the homogeneous trellis language. L consists of all words

$$w_1\#w_2\#\ldots\#w_n$$

where the w's are words of equal length k over the alphabet $\{0,1\}$ and, moreover, $w_1 = 0^k$, $w_n = 1^k$ and each w_{i+1} represents in binary notation the successor of w_i . (0#1 and 00#01#10#11 are two shortest words of L .)

6. Conclusion.

We discuss here briefly some modifications, as well as results from further work in progress [3, 5].

If we wish to process only one input string at a time, i.e., if we give up the possibility of pipelining, then there are two rather distinct possibilities how to interpret a systolic trellis automaton. First we can view the whole (finite) trellis as one large processor (with no registers) computing a combinational function in one step of a (slow) external clock. Alternately, any program for a homogeneous trellis automaton can be executed also on the linear systolic array (not pure) shown in Figure 6.1, one trellis-level per each (fast) clock cycle. The circles show the (identical) processors and the rectangulars show the memory registers. The corresponding output is produced n-1 steps (clock cycles) after an input is read.

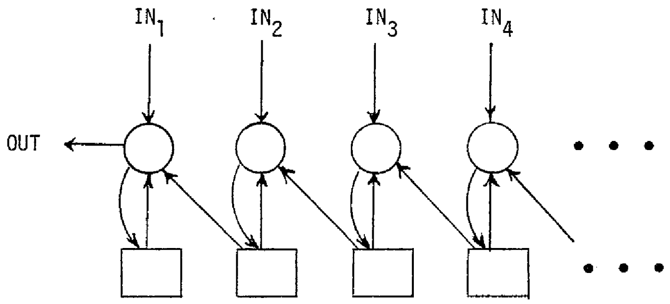


Figure 6.1

Actually, it can be even shown that a program for a trellis automaton can also be translated to a program for a linear systolic array (see Figure 6.2) which reads the input in a serial way and produces the output with delay n , where n is the length of input (see [5]).

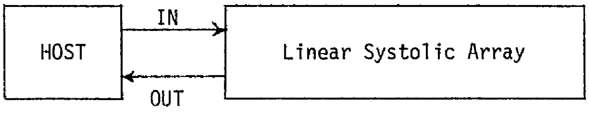


Figure 6.2

Therefore, as an application of Theorem 6 we can mechanically compile a program for any of the above devices from a linear context-free grammar specifying a given problem. Such program would not be necessary an optimal one but will be guaranteed correct. See Example 8 in Section 5.

Even if we cannot design a program for a linear systolic array fully mechanically or if such program would not be efficient enough, it seems to be much easier to write first a program for a trellis automaton since the trellis structure gives us good geometric intuition about the information flow and synchronization problems. Such program can then be executed on the array of Figure 6.1 or mechanically translated for sequential linear array of Figure 6.2. We consider this approach an important methodological tool in the design of systolic arrays.

We have not discussed in this paper any decidability or complexity results. It follows from Theorems 4 and 6, see [3] for details, that the emptiness problem for homogeneous trellis automata is undecidable. Consequently, also the equivalence problem is undecidable for homogeneous trellis automata. On the other hand, the membership problem is decidable in $O(n^2)$ time on (deterministic) multi-tape Turing machines. This result and some of its generalizations will be further discussed in [3].

Our definitions of trellis automata above require that an input word is fed to a trellis automaton always on a sepcific level, i.e., always on the shortest possible level. Important generalizations on input conditions allow the possibility of the input being fed on an arbitrary (sufficiently long) level. In other words, the use of any sufficiently large chip is allowed to recognize an input.

There are two modifications of such a generalization. In a <u>stable</u> automaton an input must be fed to the external input pins of the leftmost processors on any sufficiently long level of processors. Blanks are fed to the remaining processors on this level. The final output must be independent of the level chosen.

In a <u>superstable</u> automaton an input can be fed to the external input pins of arbitrary processors on any sufficiently long level (preserving the left-to-right order of the input letters). Again, blanks are fed to the remaining processors on this level. Now the final putput must be independent of the level chosen, as well as of the choice of the processors on this level actually taking the input.

It will be shown in [3] that, for any homogeneous trellis automaton, an equivalent superstable (and therefore also stable) homogeneous trellis automaton can be constructed. The practical significance of this result is obvious: any sufficiently large chip is suitable for the processing of words within its range. From the theoretical point of view, stability results can be used, for instance, in various considerations concerning closure properties and the characterization of acceptable languages. As regards stability and superstability, it is interesting to note that there is a remarkable difference between systolic tree and systolic trellis automata [3, 4].

In this paper we studied systolic automaton based on the trellis structure. In [6] tree structure was considered. It is clear that some aspects of systolic automata and some proof techniques are infinite independent on the underlying structure. For example the next state (configuration) of a systolic automaton is recurrently computed so that each processor locally computes a new value based on the values produced by its neighbours according to the transition function. Actually, under very general conditions on the underlying structure, we can show that the homogeneous systolic automata constitute a normal form, that the defined family of languages is closed under Boolean operations and contains all regular sets.

REFERENCES

1. Berstel, J. Transductions and context-free languages, B.G. Teubner, Sttatgart, 1979.

2. Culik, K. II, Gruska, J., and Salomaa, A. Systolic automata for VLSI. Research Report CS-81-33, Department of Computer Science, University of Waterloo, 1981.

3. Culik, K. II, Gruska, J., and Salomaa, A. Systolic trellis automata: stability, decidability and complexity. Research Report CS-82-04, Department of Computer Science, University of Waterloo, 1982.

4. Culik, K. II, Gruska, J., and Salomaa, A. Systolic automata for VLSI on balanced trees. Research Report CS-82-01, Department of Computer Science, University of Waterloo, 1982.

5. Culik, K. II, and Pachl, J.K. Designing systolic arrays (in preparation).

6. Culik, K. II, Salomaa, A., and Wood, D. VLSI systolic trees as acceptors, Research Report CS-81-32, Department of Computer Science, University of Waterloo, 1981.

7. Kung, H.T. Let's design algorithms for VLSI systems. Proc. Caltech Conference on Very Large Scale Integration, Ch. L. Seitz, Ed., Pasadena, 1979, pp. 65-90.

8. Kung, H.T. The structure of parallel algorithms. Research Report, Department of Computer Science, Carnegie-Mellon University, 1979.

9. Kung, H.T., and Leiserson, C.E. Systolic arrays (for VLSI). Proc. Sparse Matrix, I.S. Duff and G.W. Stewart, Ed., Society for Industrial and Applied Mathematics, 1979, pp. 256-282.

10. Leiserson, C.E., and Saxe, J.B. Optimising synchronous systems. Proc. 22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, 1981, pp. 23-36.

11. Mead, C.A., and Conway, L.A. Introduction to VLSI systems. Addison-Wesley, Reading, Massachussets, 1980.