Systolic Trellis Automata: Stability, Decidability and Complexity*

K. Culik II, J. Gruska[1], A. Salomaa[2]

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-82-04

January, 1982

* This work was supported by Natural Science and Engineering Research Council of Canada, Grant Nos. A 7403 & A 1617.

1) On a leave of absence from Computer Research Centre, Bratislava, Czechoslovakia

2) On a leave of absence from the University of Turku, Turku, Finland

ABSTRACT

Systolic trellis automata are models of hexagonally connected and triangularly shaped arrays of processors for VLSI circuits. This paper studies the problems of stability, decidability and complexity for them.

The original definition requires that an input string is fed to a specific row of processors. Here, it is shown that given a homogeneous trellis automaton we can construct an equivalent one which allows to feed the input string to any sufficiently long row of processors.

Moreover, some closure and decidability results for trellis automata are shown and the computational complexity of languages accepted by trellis automata is investigated.

1. Introduction

Systolic trellis automata considered in this paper have been introduced in (Culik and al., 1981a).

A systolic trellis automaton is a model for VLSI circuits. It is a triangularly shaped systolic system of hexagonally connected processors (functional elements) with uni-directional flow of data and with fixed delays in all interconnections.

The notion of a systolic system captures such concepts as pipelining, paralelism and a multiprocessor system. The importance of systolic systems has been primarily connected with VLSI technology.

VLSI technology encourages building of multi-processor systems where the processors are uniform and simple, the interconnections are regular and the external connections are minimised. Several models of systolic systems have been considered so far and it has been shown how systolic systems can perform efficiently some important computations, see (Conway and Mead, 1980) and (Kung, 1979). Some of the most important and interesting applications of systolic systems deal with hexagonally connected systems.

A formal investigation of systolic systems has started in (Culik and al., 1981c), where the so called systolic tree automata have been introduced and investigated. In systolic tree automata the underlying structures of processors and their interconnections form trees. Some additional results concerning systolic tree automata are presented in (Culik and al., 1981b and 1982).

In the present paper we deal only with trellis automata in the so called normal form (Culik and al., 1981a). The underlying structure (called also an infinite labeled trellis) of such a systolic trellis automaton is shown in Fig. 1.1. The vertex labels represent the names of the corresponding processors.

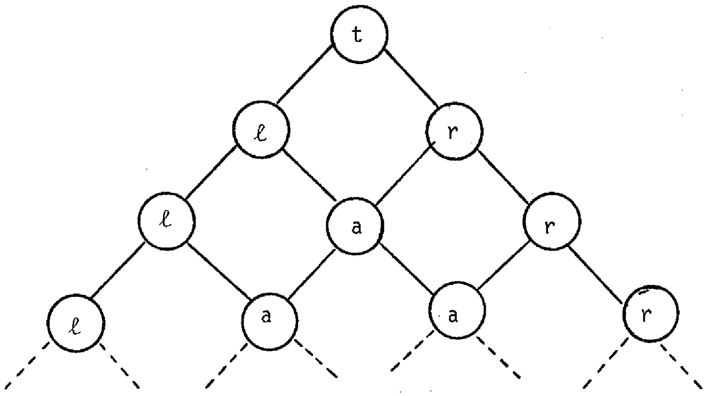


Fig. 1.1

Systolic trellis automata get their input data, i.e., an input word w, on the external inputs of processors on the level with $|w|$ processors. On every level all processors receive their inputs at the same time. Then they immediately do their computations and without any delay they immediately output their results to the processor on the next level. It akes one time unit to transmit data along an edge.

It is natural to consider systolic trellis automata as acceptors where an accepting or a rejecting symbol is produced by the processor at the root of the trellis. Similarly as in (Culik and al., 1981a) we consider here only trellis automata the underlying trellises of which are regular.

In this paper we first introduce two special classes of trellis automata which are much more flexible as far as their inputs are concerned. The so called stable trellis automata allow that an input is fed to the external inputs of the leftmost processors on any sufficiently large level of processors. Superstable trellis automata allow to feed an input word to the external inputs of any subsequence of processors on any sufficiently large level of processors. That actually means that in the case of stable or superstable trellis automata any sufficiently large chip can be used to recognise a word.

It is shown in Section 3 that to any homogeneous trellis automaton we can effectively construct an equivalent superstable (and therefore also stable) trellis automaton.

This result is then used in Section 4 to show that the family of languages accepted by homogeneous trellis automata is closed under inverse morphisms. Moreover, this family of languages is shown to be closed under injective length multiplying morphisms but not to be closed under arbitrary length multiplying morphisms. A linear speed-up theorem for homogeneous trellis automata is also derived in Section 4.

In Section 5 the emptiness problem for homogeneous trellis automata is shown to be undecidable. From this result some other undecidability results are derived. For example it is undecidable whether a homogeneous trellis automaton is superstable.

Time and space complexity of trellis languages is investigated in Section 6. It is shown here that the family of trellis languages is contained in the family of deterministic context-sensitive languages.

In the last section it is indicated that any linear bounded automaton can be "simulated by a trellis automaton with a sufficiently large auxiliary space". For an illustration it is shown how trellis automata can recognise the twin shuffle languages given some auxiliary space.

2. Trellis Automata and Languages

In this section we present the basic definitions concerning trellis automata and languages. For more motivation we refer the reader to the original paper (Culik and al., 1981a). The basic results of the paper (Culik and al., 1981a) which are needed here are summarised in the Theorem 1.

Definition 1. An infinite trellis is an infinite oriented graph (with orientation of edges from sons to fathers) satisfying the following conditions.

(1) there is exactly one node (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 $(N_\ell)_r = N_m = (N_r)_\ell$ □

Since only trellis automata in the normal form are considered in this paper we shall not make use of (vertical) edges in the trellises between fathers and their middle sons and we shall also omit these edges in figures.

If T is an infinite trellis, then for $j = 1, 2, \ldots$ $\mathrm{LEVEL}_T(j)$ denotes the set of all labeled nodes of T the maximal distance of which from the root is $j - 1$. For any $j \geq 1$ there is a natural ordering of nodes in $\mathrm{LEVEL}_T(j)$ and the i-th node (from left to right) in $\mathrm{LEVEL}_T(j)$ is denoted by $N_T(i,j)$. The nodes $N_T(1,j)$ and $N_T(j,j)$, are called the left-leg and the right-leg nodes, respectively. All other nodes are called internal.

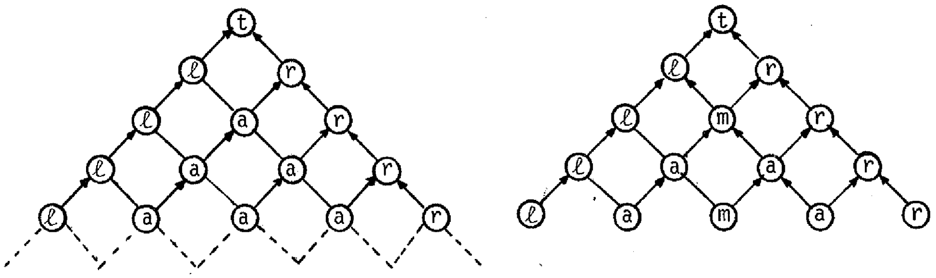


Fig. 2.1

<u>Definition 2</u>. An infinite labeled trellis T (see Fig. 2.1) is an infinite trellis with nodes labeled by symbols from a finite alphabet Δ_T. The label of the node $N_T(i,j)$ is denoted by $\lambda_T(i,j)$.

If $\bar{T}$ and T are infinite labeled trellises, then we say that T is obtained from $\bar{T}$ by a coding $c : \Delta_{\bar{T}} \to \Delta_T$ if $c(\lambda_{\bar{T}}(i,j)) = \lambda_T(i,j)$ for $1 \le i \le j$.

An infinite labeled trellis is said to be top-down deterministic (strongly top-down deterministic) if the label of any node is uniquely determined by the labels of its fathers (by the label of any of its fathers). T is said to be a <u>regular trellis</u> (<u>a semihomogeneous trellis</u>) if $T = c(\bar{T})$ for some top-down deterministic trellis (for some strongly top-down deterministic trellis) $\bar{T}$ and a coding c. T is said to be homogeneous if all nodes of T are labeled by the same label.

Definition 3. A systolic trellis automaton K 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 $\Gamma_0 \leq \Gamma$ is the alphabet of accepting symbols. Moreover $P = (\ell, c)$ where (we assume $\# \notin \Delta$)

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

are referred to as the labeling function and coding, respectively. Finally $f : \Delta \times \Sigma \to \Gamma$ is the input function and $g : \Delta \times \Gamma \times \Gamma \to \Gamma$ is the transition function. □

With every trellis automaton $K = ((\ell, c), \Delta, \Sigma, \Gamma, \Gamma_0, f, g)$ we 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: $\lambda_{T'}(1, 1) = \ell(\#, \#)$; $\lambda_{T'}(1, j) = \ell(\#, \lambda_{T'}(1, j-1))$ if $1 \leq j$; $\lambda_{T'}(j, j) = \ell(\lambda_{T'}(j-1, j-1), \#)$ if $1 < j$ and $\lambda_{T'}(i, j) = \ell(\lambda_{T'}(i-1, j-1), \lambda_{T'}(i-1, j))$ if $1 < i < j$.

Quite often we shall refer to the underlying trellis of a trellis atuomaton K but not to its labeling function or to its coding. In such cases we shall assume 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 .

A trellis automaton K is said to be semihomogeneous (homogeneous) if so is the underlying trellis of K .

Informally, we assume that every processor of K has one external input (pin) - to receive a terminal alphabet symbol - and two internal inputs - to receive operating alphabet symbols. An input word $w \in \Sigma^+$, $|w| = n$, is reorganised by K as follows v : w is fed, symbol by symbol, to the external input pins of processors on the level n . (We may visualise it as if we are using a chip which has only processors at the first n levels of T .) All processors on the level n compute in parallel the values of their input functions and send the results to their fathers on the level $n - 1$. The outputs reach fathers in one time unit. On any level $j < n$ all processors receive inputs on their internal inputs at the same time. They compute in parallel the values of the corresponding transition functions and again send results along output edges. 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 .

More formally we can proceed as follows. Let $w \in \Sigma^+$, $|w| = n$, $w = w_1 w_2 \dots w_n$, $w_k \in \Sigma$, $1 \le k \le n$. For $1 \le i \le j \le n$ let OUTPUT(K, w, i, j) be defined as follows:

$$\text{OUTPUT}(K, w, i, n) = f(\lambda_T(i,j), w_i) \qquad 1 \le i \le n$$

and

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

for $1 \le j < n$.

The language accepted by a trellis automaton K is now defined by

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

Denote by $\mathcal{L}(RT)$ and $\mathcal{L}(HT)$ the families of languages accepted by trellis automata and by homogeneous trellis automata, respectively.

Two trellis automata are said to be equivalent if they accept the same language. It was shown in (Culik and al., 1981a) that to any semihomogeneous trellis automaton we can effectively construct an equivalent homogeneous trellis automaton such that its input function is the "identity function" in the sense that $f(v, \xi) = \xi$ for any label v and any terminal symbol ξ . We can therefore assume the following normal form

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

for homogeneous trellis automata.

In the rest of this paper we shall often make use of the fact that if a $w \in \Sigma^+$ is fed to a trellis automaton K , then for any $1 \leq i \leq j \leq n = |w|$, the processors of K on the level n at the leaves of the subtrellis with the node $N(i,j)$ as the root receive on their inputs the word $w^{(i,j)} = w_i\, w_{i+1} \cdots w_{n-j+i}$.

We now summarise some results from (Culik and al., 1981a) that will be needed in the rest of the paper.

Theorem 1. The family of languages acceptable by trellis automata (by homogeneous trellis automata) contains effectively all linear context-free languages and is effectively closed under boolean operations.

3. Stable and Superstable Trellis Automata

Trellis automata require that an input word is fed to external input pins of processors on a specific level. In other words they require to use a chip of a specific size.

In this section we consider two special classes of trellis automata which are much more flexible as far as inputs are concerned. They allow to use any sufficiently large chip to recognise a word.

Trellis automata in the first class - called stable - require only that an input is fed to the external input pins of the left most processors on any sufficiently large level of processors.

Superstable trellis automata which are in the second class allow to feed an input word to the external input pins of any subsequence of processors of any sufficiently large level of processors.

These intuitive notions are captured in the following formal definition where the symbol # plays the role of blanks.

Definition 4. Let $\# \in \Sigma$. A trellis automaton K with the terminal alphabet Σ is said to be stable if for any $u \in (\Sigma - \{\#\})^{+}$, $u \in L(K)$ if and only if $u\,\#^{n} \in L(K)$ for each $n \geq 0$; K is superstable if $v \in L(K)$ if and only if $\Pi_{\Sigma-\{\#\}}(v) \in L(K)$ for each $v \in \Sigma^{+}$ (for $w \in \Sigma^{+}$ and $\Delta \subset \Sigma$, $\Pi_{\Delta}(w)$ is the projection of w into Δ^{*}).

Theorem 2. Given a homogeneous trellis automaton K with terminal alphabet Σ we can effectively construct the homogeneous trellis automaton $\overline{K}$ with the terminal alphabet $\Sigma \cup \{\#\}$, $\# \notin \Sigma$, which is superstable and $L(K) = L(\overline{K}) \cap \Sigma^+$.

Proof. In view of the remark at the end of the previous section we can assume that $K = (\Sigma, \Gamma, \Gamma_0, g)$ i.e., that the input function of K is the identity function. For any $x \in \Gamma$ let $\overleftarrow{x}$ and $\overrightarrow{x}$ be two new distinct symbols. Let $\# \notin \{x, \overrightarrow{x}, \overleftarrow{x} \mid x \in \Gamma\}$.

The semihomogeneous trellis automaton $\overline{K}_0 = (\overline{P}, \overline{\Delta}, \overline{\Sigma}, \overline{\Gamma}, \overline{\Gamma}_0, \overline{f}, \overline{g})$ is now defined as follows.

$\overline{P} = (\ell, \text{id})$ where ℓ is the labeling function such that the corresponding trellis is the roof (Culik and al., 1981a), i.e., it has the root labeled by t , the left-leg nodes by ℓ , the right-leg nodes labeled by r and the remaining nodes labeled by a . $\overline{\Delta} = \{t, \ell, r, a\}$ and id is the identity function in $\overline{\Delta}$.

Moreover $\overline{\Sigma} = \Sigma \cup \{\#\}$, $\overline{\Gamma} = \{x, \overleftarrow{x}, \overrightarrow{x} \mid x \in \Gamma\} \vee \{\#\}$, $\Gamma_0 = \{x, \overleftarrow{x}, \overrightarrow{x} \mid x \in \Gamma_0\}$

$\vartheta \in \bar{\Delta}$ / $x,y \in \Gamma$	a	ℓ	r	t
(x,y) $(\vec{x},\overleftarrow{y})$ $(x,\overleftarrow{y})$ $(\vec{x},y)$	$g(x,y)$	$g(x,y)$	$g(x,y)$	$g(x,y)$
$(x,\#)$ $(x,\vec{y})$ $(\vec{x},\vec{y})$ $(\vec{x},\#)$	$\vec{x}$	$\vec{x}$	$\vec{x}$	$\vec{x}$
$(\#,x)$ $(\overleftarrow{y},x)$ $(\overleftarrow{y},\overleftarrow{x})$ $(\#,\overleftarrow{x})$	$\overleftarrow{x}$	$\overleftarrow{x}$	$\overleftarrow{x}$	$\overleftarrow{x}$
$(\#,\vec{x})$	#	$\vec{x}$	#	#
$(\overleftarrow{x},\#)$	#	#	$\overleftarrow{x}$	#
$(\overleftarrow{x},\vec{x})$	#	$\overleftarrow{x}$	$\vec{x}$	$\overleftarrow{x}$

Fig. 3.1

The input function of $\bar{K}_0$ is the identity function and the transition function $\bar{g}$ is defined for some arguments in Fig. 3.1 and its value is # for all other arguments.

Informally (see Fig. 3.2) $\bar{K}_0$ works in such a way that it always ignores the symbols # and tries to simulate K whenever possible.

If a processor of $\overline{K}_0$ receives # on one input and a symbol from Γ on the other input, then it outputs the symbol from Γ marked by $\leftarrow$ or $\rightarrow$ to show the direction the symbol came from. If a processor receives on inputs two symbols from Γ, perhaps marked, and if they are not of divergent directions (for example $\overleftarrow{a}$ and $\overrightarrow{b}$), then this processor computes the value of the function g for the given input symbols ignoring their marks (if there are any). If a processor receives # on one input and a symbol marked by $\leftarrow$ (by $\rightarrow$) on its right (left) input, then this processor lets the input symbol pass through. However leg-processors behave in this last case differently. They take care that no result of previous computations can get lost if it maybe still needed. A leg-node processor does not allow a symbol "to fall out of the trellis", it sends such a symbol to its father.

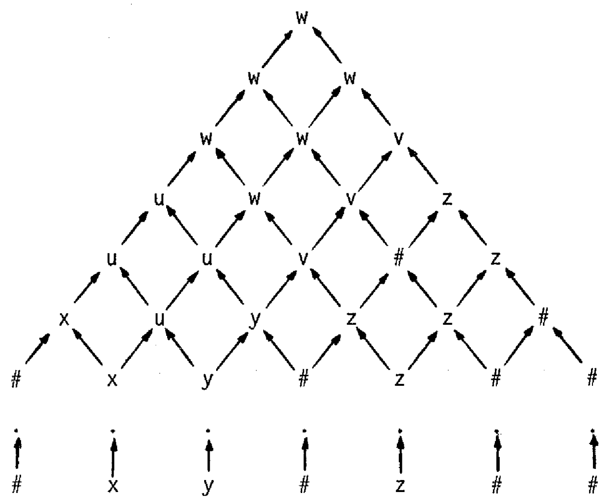


Fig. 3.2

Formally, we can proceed as follows. Let $w \in \overline{\Sigma}$. Using the definition of $\overline{g}$ it is easy to prove by induction (in i and j) the following assertion (A) where $1 \le i \le j \le |w|$ and $z = \mathrm{OUTPUT}(K, w_{\Sigma}^{(i,j)}, 1, 1)$.

$$
\text{(A) } \mathrm{OUTPUT}(\overline{K}_0, w, i, j) = \begin{cases} z \text{ if } w^{(i,j)} \in \Sigma \cup \Sigma \overline{\Sigma}^{*} \Sigma \\ \overleftarrow{z} \text{ if } (w^{(i,j)} \in \# \overline{\Sigma}^{*} \Sigma) \text{ or } (w^{(i,j)} \in \# \overrightarrow{\Sigma}^{*} \Sigma \overrightarrow{\Sigma}^{*} \# \\ \qquad \text{and } i = 1) \\ \overrightarrow{z} \text{ if } (w^{(i,j)} \in \Sigma \overrightarrow{\Sigma}^{*} \#) \text{ or } (w^{(i,j)} \in \# \overrightarrow{\Sigma}^{*} \Sigma \overrightarrow{\Sigma}^{*} \# \\ \qquad \text{and } i = j \ne 1) \\ \# \text{ if } (w^{(i,j)} \in \# \overrightarrow{\Sigma}^{*} \# \text{ and } 1 < i < j) \text{ or} \\ \qquad (w^{(i,j)} \in \{\#\}^{*} \text{ and } i = 1 \text{ or } i = j) \end{cases}
$$

(A) immediately implies that $\overline{K}_0$ is superstable and $L(\overline{K}_0) \cap \Sigma^{+} = L(K)$. Now the theorem follows from the fact that to any semihomogeneous trellis automaton we can effectively construct an equivalent homogeneous trellis automaton.

Theorem 2 implies that superstable homogeneous trellis automata are as powerful as homogeneous trellis automata. Quite a different situation is in the case of systolic tree automata with balanced trees. Such tree automata accept context-sensitive languages which are not context-free but superstable systolic tree automata on balanced trees accept only regular languages. We conjecture that when all systolic trellis automata are considered, not only homogeneous ones, then superstable trellis automata are less powerful and they accept only homogeneous trellis languages.

It has also been shown that stable systolic tree automata on balanced trees are more powerful than superstable ones. This is not the case for homogeneous trellis automata but we don't know the relation between stable and superstable regular trellis automata.

4. Closure of Homogeneous Trellis Languages Under Morhpisms and Inverse Morphisms

In this section we investigate the closure of homogeneous trellis languages under morphisms and inverse morphisms. In doing so we shall make use of Theorem 2.

The closure under inverse morphism will follow from Lemma 1. To formulate and to prove this lemma we shall use the following notions.

Let Σ be a finite alphabet, $k \geq 1$ an integer. For any $w \in \Sigma^+$ $|w| \leq k$, let $[w]$ be a distinct symbol. Denote $\Sigma^{[k]} = \{[w] ; \ w \in \Sigma^+ , \ |w| \leq k\}$.

If $w \in \Sigma^+$, then the k-code of w is defined to be the word $[w_1] \ldots [w_s] [w_{s+1}]$ such that $w = w_1 \ldots w_s w_{s+1}$, $|w_i| = k$ for $1 \leq i \leq s$ and $|w_{s+1}| \leq k$.

A morphism $h : \Sigma_1^* \to \Sigma_2^*$ is said to be length multiplying if for any $a_1 , a_2 \in \Sigma$, $|h(a_1)| = |h(a_2)| > 0$.

Lemma I. Given a homogeneous trellis automaton $K = (\Sigma, \Gamma, \Gamma_0, g)$ and a length multiplying morphism $h : \Sigma_1 \to \Sigma$ we can effectively construct a homogeneous trellis automaton K_1 such that $L(K_1) = h^{-1}(L(K))$.

Proof. We first define $K_1 = (T_1, \Delta_1, \Sigma_1, \Gamma_1, \Gamma_{1,0}, f_1, g_1)$ as follows. T_1 is the trellis with the root labeled by t and with all other nodes labeled by a, i.e., $\Delta_1 = \{t, a\}$. $\Gamma_1 = \{[w] ;$ $w \in \Sigma^+ \cup \Gamma^+$, $|w| = k$ where $k = |h(a)|$ if $a \in \Sigma_1\} \cup \Gamma_0$, $\Gamma_{1,0} = \Gamma_0$.

The input function f_1 is defined as follows: $f_1(a, \xi) = h(\xi)$ if $\xi \in \Sigma_1$ and $f_1(t, \xi) = \mathrm{OUTPUT}(K, h(\xi), 1, 1)$.

In order to define g_1 the following notation will be useful. For $w \in \Gamma^+$ and $1 \le i \le |w|$ let $\overline{\mathrm{OUTPUT}}(K, w, i, |w|) = w_i$ and for $1 \le i \le j < |w|$ let $\overline{\mathrm{OUTPUT}}(K, w, i, j) = g(\overline{\mathrm{OUTPUT}}(K, w, i+1, j), \overline{\mathrm{OUTPUT}}(K, w, i+1, j+1))$. Moreover,
$\overline{\mathrm{OUTPUT}}(K, w, j) = \overline{\mathrm{OUTPUT}}(K, \overline{w}, 1, j)\ \overline{\mathrm{OUTPUT}}(K, w, 2, j) \dots \overline{\mathrm{OUTPUT}}(K, w, j, j)$.

Now we are ready to define the transition function g_1. If $w_1 \in \Gamma^k$ and $w_2 \in \Gamma^k$, then

$$g_1(a, [w_1], [w_2]) = \overline{\mathrm{OUTPUT}}(K, w_1\, w_2, k) \tag{1}$$

and

$$g_1(t, [w_1], [w_2]) = \mathrm{OUTPUT}(K, w_1\, w_2, 1) \tag{2}$$

By induction it is easier to prove, using (1), that for $1 \le i < |w|$, and $w \in \Sigma_1^+$ the following holds

$$\overline{\mathrm{OUTPUT}}(K_1, w, i) \text{ is the k-code of } \overline{\mathrm{OUTPUT}}(K, h(w), ki)\ .$$

Finally, using (2), one can show that

$$\mathrm{OUTPUT}(K_1, w, 1, 1) = \overline{\mathrm{OUTPUT}}(K_1, w, 1) = \mathrm{OUTPUT}(K, h(w), 1) = \mathrm{OUTPUT}(K, h(w), 1, 1)\ ,$$

and therefore

$$L(K_1) = h^{-1}(L(K))\ .$$

Since K_1 is semihomogeneous an equivalent homogeneous trellis automaton can be effectively constructed.

Lemma 1 is now used to prove two theorems.

<u>Theorem 3.</u> Given a homogeneous trellis automaton $K = (\Sigma, \Gamma, \Gamma_0, g)$ and morphism $h : \Sigma_1 \to \Sigma^*$ we can effectively construct a homogeneous trellis automaton which accepts the language $h^{-1}(L(K))$.

<u>Proof.</u> According to Theorem 2 we can effectively construct a homogeneous trellis automaton $\overline{K} = (\Sigma \cup \#, \overline{\Gamma}, \overline{\Gamma}_0, \overline{g})$, $\# \notin \Sigma \cup \overline{\Gamma}$ which is superstable with respect to Σ and $L(K) \cap \Sigma^* = L(K)$. Let $k = \max \{|h(a)| \mid a \in \Sigma_1\}$ and let $h_\# : \Sigma_1 \to (\Sigma \cup \{\#\})^+$ be the morphism defined for $a \in \Sigma_1$ as $h_\#(a) = h(a)\, \#^{k-|h(a)|}$. Clearly $h_\#$ is the length multiplying morphism. According to Lemma 1 we can effectively construct a homogeneous trellis automaton which accepts the language

$$h_\#^{-1}(L(\overline{K})) = \{w \mid w \in \Sigma_1^+ , h_\#(w) \in L(\overline{K})\} = \{w \mid w \in \Sigma_1^+ , h(w) \in L(K)\} = h^{-1}(L(K)) \qquad \square$$

<u>Corollary.</u> The family of languages $L(HT)$ is closed under inverse morphisms.

<u>Theorem 4.</u> (Linear speed-up for homogeneous trellis automata). Given a homogeneous trellis automaton $K = (\Sigma, \Gamma, \Gamma_0, g)$ (which recognises a $w \in \Sigma^+$ in time $|w| - 1$) and an integer $k > 1$ we can effectively construct a homogeneous trellis automaton K' with $\Sigma^{[k]}$ as the terminal alphabet which accepts a word $\overline{w} \in (\Sigma^{[k]})^+$ if and only if $\overline{w}$ is the k-code of a $w \in L(K)$ (and K' accepts $\overline{w}$ in time $\lceil \frac{|w|}{k} \rceil - 1$).

Proof. According to Theorem 2 we can construct a homogeneous trellis automaton $\overline{K}$ with terminal alphabet $\Sigma \cup \{\#\}$ which is superstable and $L(\overline{K}) \wedge \Sigma^+ = L(K)$.

Let now $h : \Sigma^{[k]} \to (\Sigma \cup \{\#\})^*$ be the morphism defined by $h([w]) = w\,\#^{k-|w|}$. Since h is length multiplying we can construct, according to Lemma 1, a homogeneous trellis automaton K_0 which accepts the language $h^{-1}(L(\overline{K}))$. K_0 can be easily modified to get a homogeneous trellis automaton which rejects all $w_0 \in (\Sigma^{[k]})^+$ that are not k-codes of a word in Σ^+ and which accepts all other words if and only if K_0 does. Hence K_0 accepts a word $\overline{w} \in (\Sigma^{[k]})^+$ if and only if $\overline{w}$ is the k-code of a $w \in L(K)$. □

Now we proceed to study the closure of homogeneous trellis languages under morphisms. First one positive result.

Theorem 5. Given a homogeneous trellis automaton $K = (\Sigma, \Gamma, \Gamma_0, g)$ and an injective length multiplying morphism $h : \Sigma \to \Sigma_1$, we can effectively construct a homogeneous trellis automaton $\overline{K}$ which accepts the language $h(L(K))$.

Proof. Since h is a length multiplying morphism; there is an integer k such that $k = |k(a)|$ for any $a \in \Sigma$. According to Theorem 1 the family of homogeneous trellis languages contains effectively all regular languages and it is effectively closed under intersection. It means that in order to prove the theorem it is sufficient to show that one can construct a homogeneous trellis automaton $\bar{\bar{K}}$ with Σ_1 as the terminal alphabet which has the following property: if an input word w is in $(\Sigma_1^k)^+$, then $\bar{\bar{K}}$ accepts w if and only if $w \in h(L(K))$.

We don't give a formal specification of $\bar{\bar{K}}$ here. We only describe how $\bar{\bar{K}}$ recognises words. From this description the formal construction of $\bar{\bar{K}}$ should be straightforward.

In order to show transparently how $\bar{\bar{K}}$ works we assume that all processors of $\bar{\bar{K}}$, with the exception of the root, output always a pair of symbols. The first symbol is sent up along the left output edge, the second along the right one. It is easy to see how to modify $\bar{\bar{K}}$ in order to satisfy formally our definition of a trellis automaton.

Let N be a symbol not in Γ. For any $\eta \in \Gamma \cup \{N\}$ the internal alphabet of $\bar{\bar{K}}$ will contain K mutually distinct symbols $\eta = \eta^{(1)}, \eta^{(2)}, \dots, \eta^{(k)}$. Moreover, the internal alphabet of $\bar{\bar{K}}$ will contain a special symbol $[w]$ for any $w \in \Sigma_1^{[k]}$.

Let now $w \in (\Sigma_1^k)^+$ and $|w| = kn$ $\bar{\bar{K}}$ will be constructed in such a way that for $1 \le i \le j$ and $(n-1)k+1 \le j \le nk$

$$\text{OUTPUT}(K, w, i, j) = ([w^{(i,j)}], [w^{(i,j)}]) \text{ if } (n-1)k + 1 \le j \le nk$$
$$\text{OUTPUT}(\bar{\bar{K}}, w, i, j) = (h^{-1}(w^{(i,j)}), h^{-1}(w^{(i,j)}))$$

where $h^{-1}(w^{(i,j)})$ is defined to be N if $w^{(i,j)} \in \{h(a) \mid a \in \Sigma\}$.
(See Fig. 4.1 for the case $k = 2$.)

Moreover, if a processor P of $\bar{\bar{K}}$ receives on its left input a symbol $\eta_1^{(j)}$ and on its right input a symbol $\eta_2^{(j)}$ then it preceeds as follows.

If $j < k$, then P outputs $\eta_2^{(j+1)}$ on its left output and $\eta_1^{(j+1)}$ on its right output. If $j = k$, then P outputs $g(\eta_1, \eta_2)$ on both inputs if η_1 and η_2 are in Γ and P outputs N on both outputs otherwise.

Using induction it is now easy to prove that the following assertion (A) holds for any $w \in (\Sigma_1^k)^+$ such that $w = h(w_1)$ for some $w_1 \in \Sigma^+$.

(A) if $0 \leq i \leq j < n$, then $\mathrm{OUTPUT}(\bar{\bar{K}}, w, ik+1, jk+1) = \mathrm{OUTPUT}(k, h^{-1}(w), i+1, j+1)$

Moreover, if $w \in (\Sigma_1^k)^+$, $|w| = kn$ and there is no $w_1 \in \Sigma^+$ such that $h(w_1) = w$, then at least one processor at the nodes $N(ik+1, (n-1)k+1)$, $0 \leq i \leq n-1$ produces N on both outputs and therefore $\mathrm{OUTPUT}(\bar{\bar{K}}, w, 1, 1) = N$. Hence $L(\bar{\bar{K}}) = h(L(K))$. □

<u>Corollary</u>. The family of homogeneous trellis languages is closed under injective length multiplying morphisms.

Now, we proceed to show that we can not omit the assumptions of injectivity in Theorem 5 not even in the case of letter-to-letter morphisms. To show that we shall make use of the following lemma which is interesting by itself.

<u>Lemma 2</u>. For every recursively enumerable language $L \subseteq \Sigma^*$ (given by a grammar or a Turing machine) there effectively exists a homogeneous trellis automaton A such that $L = \Pi_\Sigma(L(A))$ and $L(A) \subset \Delta^* \Sigma^+$, $\Delta \cap \Sigma = 0$.

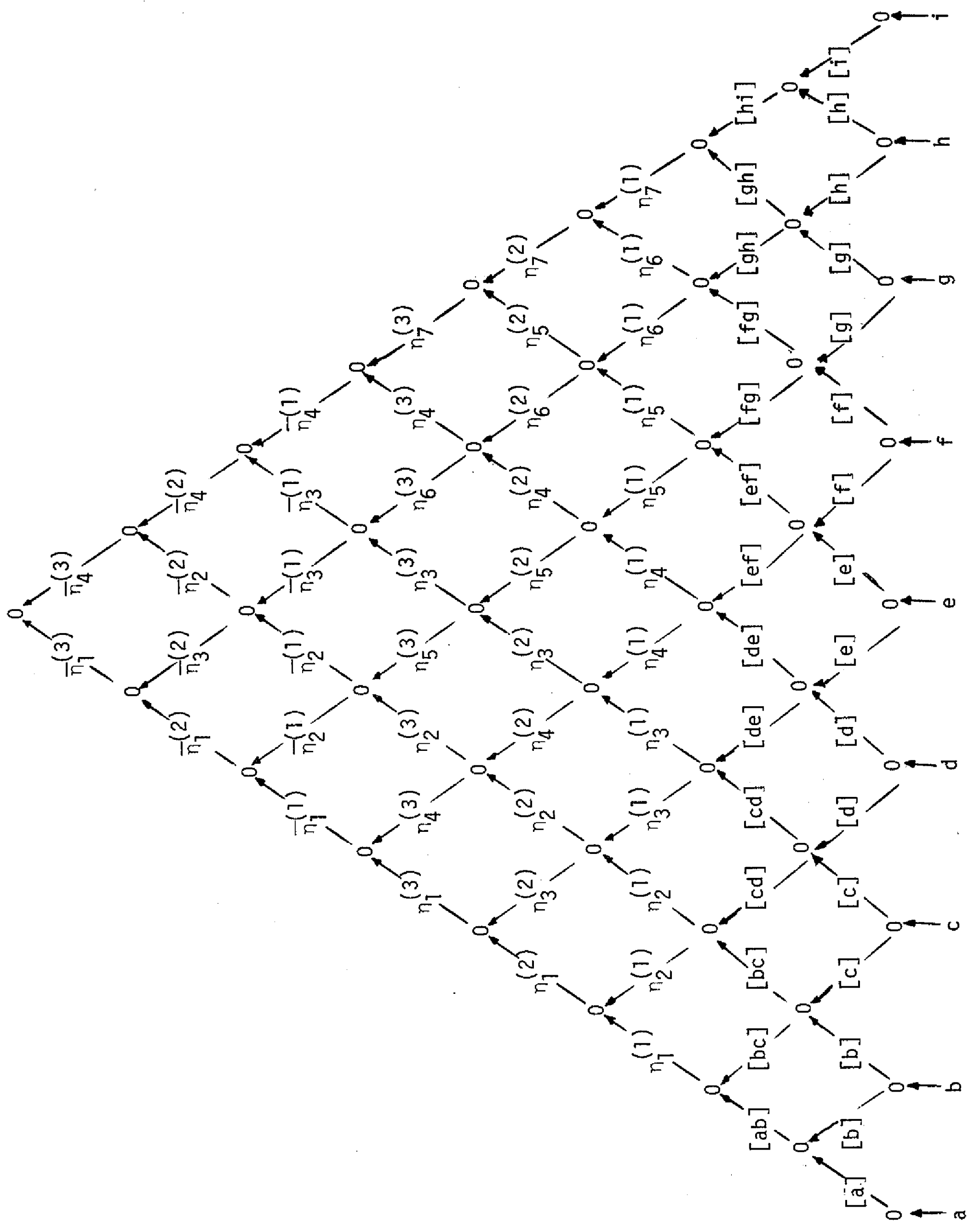


Fig. 4.1

Proof. It is well known that for every recursively enumerable language $L \subseteq \Sigma^*$ there effectively exists two linear context-free languages L_1 and L_2 such that $L_i \subseteq \Delta^* L \subseteq \Delta^* \Sigma^*$ where $\Delta \cap \Sigma = 0$, for $i = 1, 2$, and $L = \Pi_\Sigma (L_1 \cap L_2)$. Now the lemma follows from Theorem 1.

Corollary. *L(HT)* is not closed under morphisms.

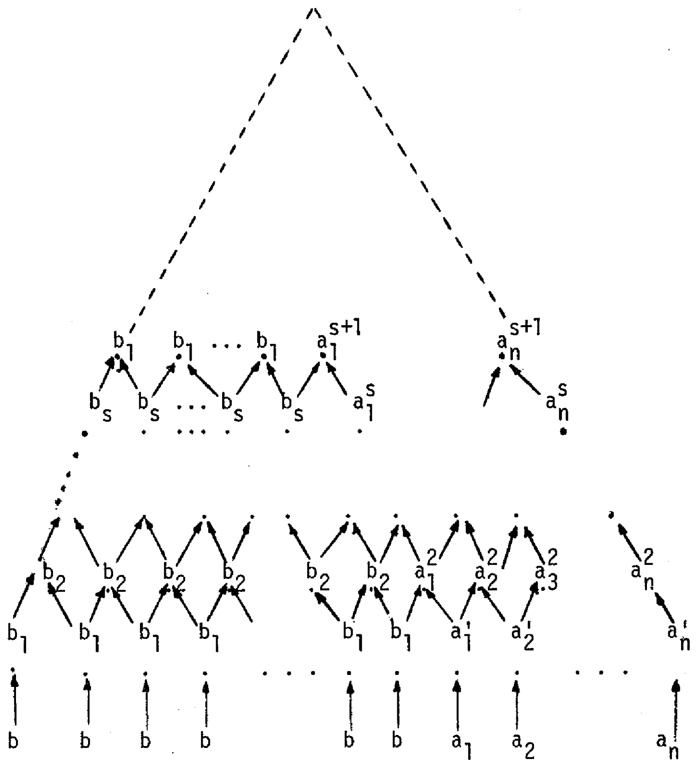
The following theorem strengthens this observation.

Theorem 5. The family of homogeneous trellis languages is not closed under letter-to-letter morphisms.

Proof. Consider arbitrary recursively enumerable language $L \subseteq \Sigma^*$. By Lemma 2 we can write $L = \Pi_\Sigma(L(A))$ for more homogeneous trellis automaton A where $L(A) \subset \Delta^* L \subseteq \Delta^* \Sigma^*$, $\Sigma \cap \Delta = 0$ (see the proof of Lemma 2). Let b be a new symbol not in Σ and let $h : (\Delta \cup \Sigma)^* \to (\Sigma \cup \{b\})^*$ be the morphism defined by $H(a) = b$ for $a \in \Delta$ and $h(a) = a$ for $a \in \Sigma$.

We show now that $h(L(A)) \subseteq b^* \Sigma^*$ is not a homogeneous trellis language.

Assume that there exists a homogeneous trellis automaton B such that $L(B) = h(L(A))$. Then every computation of B must be of the form shown in Fig. 4.2.



b_1 b_1 ... b_1 a_1^{s+1} a_n^{s+1}
b_s b_s ... b_s b_s a_1^s a_n^s
b_2 b_2 b_2 b_2 b_2 b_2 a_1^2 a_2^2 a_3^2 a_n^2
b_1 b_1 b_1 b_1 b_1 b_1 a_1' a_2' a_n'
b b b b b b a_1 a_2 a_n

Fig. 4.2

Clearly, all information on each level is contained in the last n + 1 symbols. Therefore we can construct a linear bounded automaton which simulates the "initial part of the trellis" and accepts the language L . This is, however, a contradiction since L is an arbitrary recursively enumerable language.

5. Undecidability

In this section some undecidability results concerning homogeneous trellis automata are derived.

Theorem 7. The emptiness problem for homogeneous trellis automata is undecidable.

Proof. By Lemma 2 since $L = \theta$ if and only if $\Pi_{\Sigma}(L) = \theta$.

Combining this result with Theorem 1 we get:

Corollary 1. The equivalence problem for homogeneous trellis automata is undecidable.

Corollary 2. It is undecidable if $L(K) = \Sigma^{+}$ for a homogeneous trellis automaton K with terminal alphabet Σ .

Theorem 8. It is undecidable whether a given homogeneous trellis automaton is superstable (stable).

Proof. In order to show that superstability (stability) is undecidable it is sufficient to realise that to a given homogeneous trellis automaton $K = (\Sigma, \Gamma, \Gamma_0, g)$ we can easily construct a homogeneous trellis automaton $\overline{K}$ with the terminal alphabet $\Sigma \cup \{\#\}$ where $\#$ is not in Σ such that $L(\overline{K}) = L(K)$. Then $\overline{K}$ is (stable) superstable with respect to Σ if and only if $L(K) = \theta$. Now the undecidability of (stablity) superstability follows from the undecidability of the emptiness problem.

6. Time and Space Complexity

The membership problem for trellis automata is clearly decidable. This section deals with its deterministic Turing machine time and space complexity.

First we introduce two special classes of trellis automata.

Definition 5. A labeled infinite trellis is said to be bottom-up deterministic if the label of any leg-node uniquelly determines the label of its father and the labels of any internal node and any of its fathers uniquelly determines the label of the second father.

Definition 6. A trellis automaton is said to be internally homogeneous if all its processes have the same transition function.

It was shown in (Culik and al., 1981a) that the language $\{a^{2^n} \mid n \geq 1\}$ is not accepted by any homogeneous trellis automaton but it is accepted by a trellis automaton which is internally homogeneous and has a bottom-up deterministic trellis. Therefore the following lemma holds.

Lemma 3. The family of homogeneous trellis languages is strictly contained in the family of languages accepted by internally homogeneous trellis automata with bottom-up deterministic trellises.

On the other hand languages accepted by internally homogeneous trellis automata or by trellis automata with bottom-up deterministic trellises do not seem to be harder to recognise than homogeneous trellis languages.

Theorem 9. (1) Any trellis language can be recognised in $O(n^2)$ time on a multitape Turing machine and in $O(n^3)$ time on a one-tape Turing machine.

(2) Any trellis language which is recognisable by an internally homogeneous trellis automaton or by a trellis automaton with a bottom-up deterministic trellis can be recognised in $O(n^2)$ time on a one-tape Turing machine.

Proof. Let K be a trellis automaton. We show how to design a three tape Turing machine M which recognises the language $L(K)$ in $O(n^2)$ time.

M starts with the input word w on the first tape and with the head of the first tape on the first symbol of w .

Second tape	$\cdots$	ℓ_{11}	\$	ℓ_{21}	ℓ_{22}	\$	ℓ_{31}	ℓ_{32}	ℓ_{33}	\$	ℓ_{41}	$\cdots$

Fig. 6.1

At first M generates on the second tape, using the labeling function of K , the labels of nodes of T , level by level (See Fig. 6.1). To do that M uses the third tape as the scratch tape. M starts this generation by printing ℓ_{11} - the label of the root - on the second tape. Each time a new level of labels is generated in the second tape, M moves the head on the first tape one symbol to the right to check if enough levels have been generated. If not, M first copies the labels of the last generated level on the third tape and then, using the third tape, it generates a new level of labels on the second tape. In this way M generates labels of all $|w|$ levels in $O(n^2)$ time where $n = |w|$.

Now M begins to simulate, in a bottom up way and level by level, the recognition of w by K.

M first moves the head of the first tape on the rightmost symbol of w and the head of the second tape on the rightmost label of the last level. Then, moving heads on the first two tapes from right to left, symbol by symbol, M computes the outputs of processors in the leaves and M writes the results from right to left on the third tape. After this is done for the last level of labels, M copies the content of the third tape on the first tape, moves the head of the first tape on the right most symbol of the rewritten word. Now M is prepared to simulate the computation of processors on the last but one level. Labels of processors are on the second tape and their input values on the first tape. In this way M needs $O(n^2)$ time to carry out the whole simulation of K.

Then M starts to simulate recognition of w on K. Moving from right to left M rewrites subsequently all labels by the output values, the corresponding processors have when w is being recognised by K. It can be done easily because, starting with level $n - 1$, all necessary inputs symbols can be found in the squares which contained originally labels of sons of the corresponding nodes. In this way the whole simulation needs $O(n^3)$ time.

Situation is simpler in the case of bottom-up deterministic trellises. In such a case we can assume that M has one two-track tape and an input word is written on the first track. M can now generate on the second track, under the word w, level by level, labels of K, always rewriting labels of the preceding level. That is, only labels of the last generated level are kept. In this way M can generate labels of the leaves in $O(n^2)$ time. Then M starts to simulate K, level by level. Simulation of processors of every level k starts with input symbols on the first track and with labels on the second track. During the simulation the outputs are computed and written on the first track to replace input symbols which are not needed anymore. In parallel M computes labels of processors on the level $k - 1$ and writes them on the second track to replace labels which will not be needed anymore. In this way the whole simulation can be done in $O(n^2)$ time.

The situation is even simpler in the case of internally homogeneous trellis automata. The generation of the labels on the leaves is done as in the previous case. Simulation of the recognition process is however simpler because it is not necessary to generate labels on the other levels because all processors in these levels have the same transition function.

<u>Corollary 1</u>. Time complexity of languages accepted by internally homogeneous trellis automata or by trellis automata with bottom-up deterministic trellises is $\theta(n^2)$.

<u>Proof</u>. The upper bound follows from Theorem 9. The lower bound follows from the fact that the recognition of the homogeneous trellis language $\{w \$ w \mid w \in \Sigma^+, \$ \in \Sigma\}$ (Culik and al., 1981a) needs $\Omega(n^2)$ time on a one-tape Turing machine (Hennie, 1965).

<u>Corollary 2</u>. Time complexity of homogeneous trellis languages on one-tape Turing machines is $\theta(n^2)$.

We don't know whether the upper bound $O(n^3)$ stated in Theorem 9 for recognition of trellis languages on one-tape Turing machines can be improved.

The followign theorem summarises results concerning space complexity.

<u>Theorem 8</u>. (1) Any trellis language can be recognised in $O(n)$ space and $O(n^3)$ time on Turing machines.

(2) Any trellis language which is recognisable by an internally homogeneous trellis automaton or by a trellis automaton with a bottom-up deterministic trellis can be recognised in $O(n)$ space and $O(n^2)$ time on Turing machines.

<u>Proof</u>. (1) It is enough to use a one-tape Turing machines with two tracks on the tape. To begin with let an input word be written on the first track. Simulation of a trellis automaton K proceeds, level

by level, as follows. For every level k M generates on the second track, always from the scratch, level by level, all labels on levels 1, 2, ..., k . When that is done M simulates the computations of all processors on the level k . The input symbols are on the first track, the processors names on the second track. M writes the output values of the processors on the level k , from right to left, on the first track to replace the input symbol which are not needed anymore. In this way M needs only space n and time $O(n^3)$.

(2) This was actually shown when the part (2) of the Theorem 9 was proven.

Corollary. The family of trellis languages is contained in the family of deterministic context-sensitive languages.

Observe, that it follows from Theorem 7 that there are trellis languages which are not indexed languages. Indeed, the emptiness problem is known to be decidable for induced languages.

We conclude this section with some additional observations concerning computational complexity of trellis languages.

1. Since any linear context-free language is also a homogeneous trellis language we get immediately
   (a) The family of homogeneous trellis languages contains languages which are not recognisable in real time by any multitape Turing machine (Hennie, 1965).
   (b) There are homogeneous trellis languages which need $\Omega(\log n)$ space on deterministic Turing machines (Cobham, 1966).

2. There is still a large gap between the upper bound for space complexity given in Theorem 8 and the lower bound mentioned above.
3. We don't know whether the upper bound $O(n^2)$ for recognition of trellis languages on multitape Turing machines can be improved. Observe only that any improvement of this upper bound would improve the best known upper bound for recognition of linear context-free languages on multitape Turing machines.

7. Simulation of Linearly Bounded Automata

Trellis automata are easily simulated by linearly bounded automata. On the other hand trellis automata have problems to simulate linearly bounded automata for two reasons: (i) trellis automata have only time n and space n to recognise a word of the length n, (ii) in each parallel computation step they loose one processor or one "piece of memory".

On the other hand, a given computation of any linearly bounded automaton can be simulated by a trellis automaton given a sufficiently large auxiliary space.

In order to indicate how this can be done we show in this section that any twin shuffle can be recognised, in a sense, by a trellis automaton given some auxiliary space.

Let Σ be a finite alphabet. For $a \in \Sigma$ let $\overline{a}$ be a distinct symbol. Denote $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$. For a $w = w_1 w_2 \dots w_n \in \Sigma^+$ with $w_i \in \Sigma$ for $1 \leq i \leq n$ let $\overline{w} = \overline{w}_1 \overline{w}_2 \dots \overline{w}_n$.

The <u>twin shuffle</u> over Σ is now defined to be the language

$$TS(\Sigma) = \{w \mid w \in (\Sigma \cup \overline{\Sigma})^+ , \ \overline{\Pi_\Sigma(w)} = \Pi_{\overline{\Sigma}}(w)\}$$

It seems that for no Σ the language $TS(\Sigma)$ can be recognized by a trellis automaton. On the other hand we show now that the language (c is a symbol not in $\Sigma \cup \overline{\Sigma}$):

$$L = \{c^{2n} w c^{2n} \mid w \in (\Sigma \cup \overline{\Sigma})^{2n} , \ \overline{\Pi_\Sigma(w)} = \Pi_{\overline{\Sigma}}(w)\}$$

is a homogeneous trellis language.

First observe that the language $L_0 = \{c^{2n} w c^{2n} \mid w \in (\Sigma \cup \bar{\Sigma})^{2n}$ is a linear context free language and therefore a homogeneous trellis language. Since homogeneous trellis languages are closed under intersection, it implies that in order to show $L \in \mathcal{L}(HT)$ it is sufficient to construct a semihomogeneous trellis automaton K with terminal alphabet $\Sigma \cup \bar{\Sigma} \cup \{c\}$ which, given an input word $w_0 = c^{2n} w c^{2n}$ with $w \in (\Sigma \cup \bar{\Sigma})^{2n}$, accepts w_0 if and only if $\overline{\Pi_{\Sigma}(w)} = \Pi_{\bar{\Sigma}}(w)$. An informal description of such a trellis automaton is now given.

K will have two types of processors. ℓ-processors will be in the left-leg nodes, a-processors in all other nodes.

Let # be a symbol not in $\Sigma \cup \bar{\Sigma} \cup \{c\}$. The oeprating alphabet of K will contain the symbol c and all pairs (a,b), $a \in \Sigma \cup \{\#\}$, $b \in \bar{\Sigma} \cup \{\#\}$.

To give a more transparent description of K we will assume that every processor of K produces two output symbols, one is sent up along the left output edge, the second along the right output edge.

If c is received on an external input of a processor of K then c is produced on all output edges.

If $a \in \Sigma$ $(\in \bar{\Sigma})$ is received on the external input then the symbol $(a, \#)$ $((\#, a))$ is produced on the left output edges and the symbol $(\#, \#)$ on the right output edge.

If a processors receives c on the right input, then it produces on the left (right) output the symbol it received on the left (right) input.

If a processor receives on its left input a pair (a, b) and on its right input a pair (c, d), then it outputs the pair (μ(a, c) , μ(b, d)) on its left input and the pair (ν(a, c), ν(b, d)) as its right input where the functions μ and ν are defined as follows.

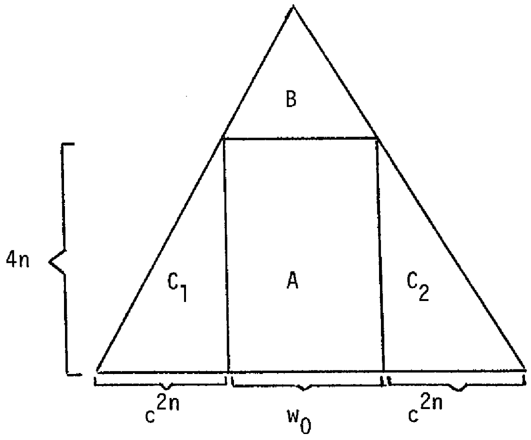
$$\mu(a, b) = \underline{\text{if}}\ \ a = \#\ \ \underline{\text{then}}\ \ b\ \ \underline{\text{else}}\ \ a$$

$$\nu(a, b) = \underline{\text{if}}\ \ a = \#\ \ \text{then}\ \ a\ \ \underline{\text{else}}\ \ b$$

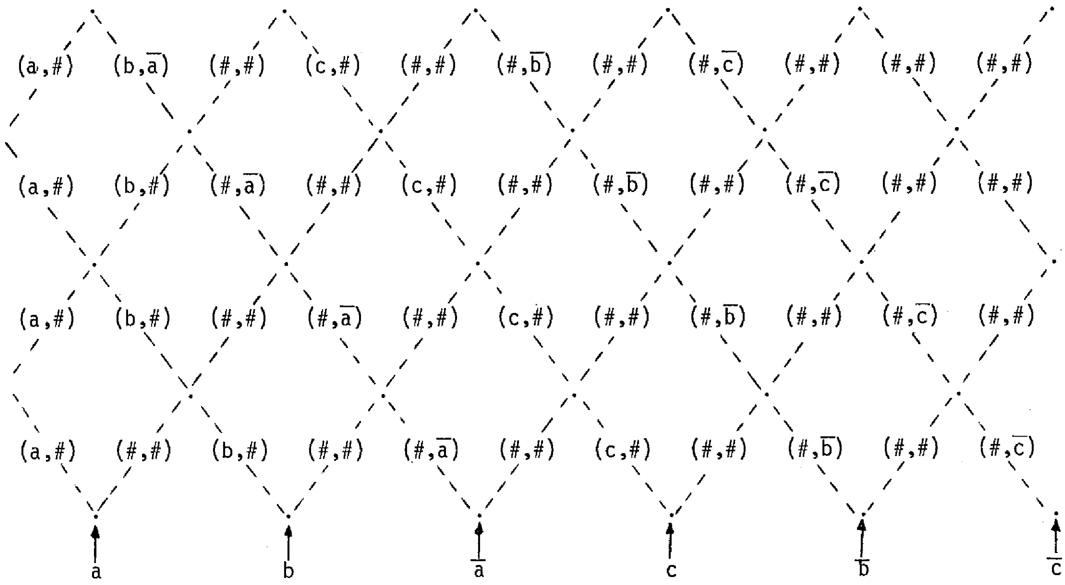
If an ℓ-processor receives c on its left input and a pair (a, b) on its right input then it activates a finite state automaton which runs along left-leg edges to the root and checks if the symbols on right input edges of processors are all of the form (a, b) $a \in \Sigma$, $b = \bar{a}$ or $a = \# = b$. If yes the input word is accepted, if not, it is rejected.

Fig. 7.1a shows how K works. All processors in parts c_1 and c_2 outputs on their left (right) outputs what they receive on their left (right) inputs.

The processors in the part A do in parallel "stable sorting" of two sequences. One is obtained from w by replacing all symbols from $\bar{\Sigma}$ by # , the second is obtained from w by replacing all symbols from Σ by # . Sorting is done under the assumption that all symbols from Σ ($\bar{\Sigma}$) are equal but less than # . This sorting is shown in Fig. 7.1b . After 4n computation steps sorting is completed and all processors in the part B except left-leg processes are stable in the sense that they output on its left (right) output what they receive on their left (right) input.


B
4n
C_1
A
C_2
c^{2n}
w_0
c^{2n}

(a)


(a,#) (b,$\bar{a}$) (#,#) (c,#) (#,#) (#,$\bar{b}$) (#,#) (#,$\bar{c}$) (#,#) (#,#) (#,#)
(a,#) (b,#) (#,$\bar{a}$) (#,#) (c,#) (#,#) (#,$\bar{b}$) (#,#) (#,$\bar{c}$) (#,#) (#,#)
(a,#) (b,#) (#,#) (#,$\bar{a}$) (#,#) (c,#) (#,#) (#,$\bar{b}$) (#,#) (#,$\bar{c}$) (#,#)
(a,#) (#,#) (b,#) (#,#) (#,$\bar{a}$) (#,#) (c,#) (#,#) (#,$\bar{b}$) (#,#) (#,$\bar{c}$)
a
b
$\bar{a}$
c
$\bar{b}$
$\bar{c}$

(b)

Fig. 7.1

REFERENCES

1. Cobham, A. (1966), Time and memory requirements for machines which recognise squares and polyndrons. IBM Research Report RC-1621, Yorktown Heights.

2. Conway, L. and Mead, C. (1980), Introduction to VLSI systems. Addison-Wesley.

3. Culik, K. II, Gruska, J. and Salomaa, A. (1981a), Systolic trellis automata (for VLSI). Research Report CS-81-34, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario.

4. Culik, K. II, Gruska, J. and Salomaa, A. (1981b), On a family of L languages resulting from systolic tree automata. Research Report CS-81-36, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario.

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

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

7. Hennie, F.C. (1965), One-tape, off-line Turing machine computations. Information and Control, V8, pp. 553-578.

8. Kung, H.T. (1979), Let's design algorithms for VLSI systems. Proc. of the Caltech Conference on VLSI, CL.L Seifz Ed., Pasadena, California, pp. 65-90.