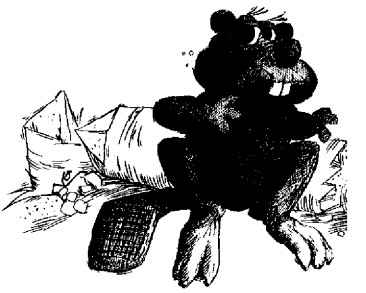
UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT



Iterative Tree Automata

Karel Culik II
Sheng Yu

CS-84-03

January, 1984

ITERATIVE TREE AUTOMATA*

Karel Culik II and Sheng Yu

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
December 1983

ABSTRACT

The iterative tree automaton is introduced as a binary tree-connected network with sequential input and output at the root of the tree. The real and linear time computational power of this type of systolic system as a language acceptor is studied. It is shown that for real-time computations the arity of the tree is essential while this is not the case for linear-time computations. Our main result is that every T(n)-time nondeterministic Turing machine can be simulated by an ITA in (deterministic) cT(n)-time. A number of properties of real and linear time ITA are proved.

Keywords Systolic system, iterative array automata, VLSI tree automata, nondeterministic Turing machine, formal languages.

1. Introduction

Motivated by the development of VLSI technology, there has been an increased interest in parallel computing, particularly in algorithms which use regular networks of identical simple processors. Such systems of processors were given the name systolic systems by Kung and Leiserson [13]. A large number of systolic algorithms have since been designed, see [14] for references.

[6] proposed a systematic study of the capabilities of systolic systems with a given geometrical configuration of processors by studying their power as language recognizers or transducers. It should be pointed out that the techniques used in systolic programs for language recognition or string manipulation are not much different from those used in numerical systolic algorithms. For example, a systolic system recognizing the language $\{ww \mid w \in \Sigma^+\}$ can be converted into a system which compute the scalar product of two vectors simply by reinterpreting the basic operations(* and + as comparison and "and").

---

* This work was supported by the Natural Science and Engineering Research Council of Canada under Grant A-7403.

Linear or multi-dimensional arrays of finite automata as language recognizers were already studied more than ten year ago, see for example [4, 8, 17]. They are usually called iterative arrays (when the input is serial at one processor) or cellular automata (when the input is parallel). Two distinct types of tree-connected networks with parallel input and a bottom-up direction of the data flow are studied in [7] and [3, 6].

In the present paper, we study tree-connected networks of finite state automata with serial input and output at the root of the tree and a bidirectional flow of data. An example of such a device is the dictionary (data base) machine of [16]. We mainly study iterative tree automata (ITA) which have an underlying binary-tree structure. However we also discuss briefly the ITA having a bounded multinary-tree structure. The well known iterative arrays are then, simply, ITA with an underlying unary-tree structure.

We focus our attention on real-time and linear-time ITA. We show that some quite complex tasks can be carried out on ITA in real time, for example, a version of dictionary look-up. On the other hand, some relatively simple context-free languages cannot be recognized by real-time ITA.

The computational power of ITA in linear time is much greater than in real-time. We show that linear-time (deterministic) ITA can simulate every linear-time nondeterministic multitape Turing machine. Moreover, the family of linear-time ITA languages are closed under Boolean operations, Kleene operations and inverse gsm mappings. Clearly, linear-time or even real-time ITA may use exponentially many processors, which is hardly feasible except for short inputs. As a continuation of this work, we will study linear-time ITA with a logarithmic bound on the depth of the tree accessed in a computation. Such ITA will use only $O(n)$ processors when given an input of length n, *for example,* dictionary machines [16] satisfy this restriction.

In Section 2, the definition of an iterative array and a lemma from [4] is reviewed. And in the next section, the formal definition of ITA is given. In Section 4, it is shown that the satisfiability problem for Boolean expressions can be solved in linear time by ITA. Then it is shown that every $T(n)$-time nondeterministic Turing machine can be simulated by an ITA in (deterministic) $cT(n)$-time. In the last section, the multinary or n-ary ITA are introduced and we show that the families of languages they recognize in real time form a proper hierarchy with respect to the arity of the underlying tree structure. However, in the case of linear and super-linear time, n-ary ITA ($n \geq 2$) define the same family of languages for $n \geq 2$.

2. Preliminaries

A linear iterative array automaton (IAA) is a one-dimensional one-way infinite sequence of cells. Each of the cells communicates with its left neighbor and right neighbor, except that the leftmost cell, called a special cell, communicates with the external world and its right neighbor. This device works synchronously. At the beginning, all the cells are in a quiescent state denoted by #. The next state of a cell is depending on the current state of this cell, the current state of the left neighbor (or external input for the special cell) and the current state of the right neighbor. The fact that the special cell enters a final state shows the

acceptance of the input string.

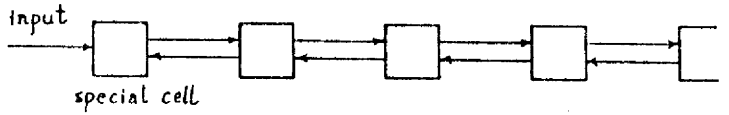


Fig. 2.1

Definition 2.1 *An iterative array automaton is a quintuple* $I = (Q, \Sigma, \delta_0, \delta, F)$ *where*

Q *is a finite set of states,*

Σ *is the input alphabet;*

$\delta_0 : \Sigma \cup \{\epsilon\} \times (Q \cup \{\#\})^2 \to Q \cup \{\#\}$ *is the state transition function for the special cell;*

δ: $(Q \cup \{\#\})^3 \to Q \cup \{\#\}$ *is the state transition function for every cell except the special cell, which satisfies* $\delta(\#,\#,\#) = \#$;

$F \subseteq Q$ *is the set of final states.*

$\# \notin Q$ *is a special quiescent state.* □

Iterative array automata in general and the real-time linear iterative array automata in particular have been studied quite extensively, see, e.g., [2, 4, 8, 15]. In [4] a property of real-time IAA has been shown which can be used to prove that certain languages are not accepted by any real-time IAA. We now introduce a concept related to this property and the property itself.

Definition 2.2 *Let* $U \subseteq \Sigma^*$ *be a set of strings. For any* $x,y \in \Sigma^*$, *we say that* $xR_k y (mod\ U)$ *if for any* $r \in \Sigma^*$ *and* $|r| \leq k$, $xr \in U$ *if and only if* $yr \in U$. □

Theorem 2.1 *For any n-dimension iterative array A, there exist an integer h and a polynomial g of degree n such that for any integer k, the number of equivalence classes of* $R_k (mod\ (L(A))$ *is at most* $h^{g(k)}$. [4] □

3. The iterative tree automata

An iterative tree automaton (ITA) is a systolic system in which the basic cells are connected in a infinite full binary tree structure. The root of the tree, call the root cell, is a special cell, like the leftmost one in the iterative array, which is the only place exposed to the external world.

The ITA works synchronously. Initially, every cell is in a quiescent state. The input string is read by the root cell one symbol at a time. The next state of the root cell is determined by the current state of the root, the current input symbol and the current states of its two sons. Similarly, the state transitions of the other cells depend on the states of themselves and their three neighbors, the parent, the left son and the right son.

The transition function for the left and right cells (the left or right sons of some cell) are different. The distinction between these two functions is essential to the power of the ITA. If one function is applied on both left and right cells, the ITA would be no more powerful than a linear iterative array. Alternatively, we could use an output function for every cell, which produces different outputs

to its three neighbors. In this case, one transition function instead of three or two can be used without any loss of generality. A homogeneous ITA of this kind is defined in [19] where it is also shown that these two definitions are equivalent.

Definition 3.1 *An iterative tree automaton A is a sextuple* $(Q, \Sigma, \delta_0, \delta_l, \delta_r, F)$, *where*

Q *is a finite set of states;*

Σ *is the input alphabet;*

$\delta_0: Q \times \Sigma \times Q^2 \to Q$ *is the transition function of the root cell written as* $\delta_0(X, a, Y, Z) = X'$ *where* $X, Y, Z \in Q$ *are the current states of the root cell, its left and right son, respectively,* $a \in \Sigma$ *is the current input symbol,* X' *is the next state of the root;*

$\delta_l, \delta_r: Q^4 \to Q$ *are the transition functions of the left and right cells, respectively, written as* $\delta(X, W, Y, Z) = X'$ *where* δ *is either* δ_l *or* δ_r, X, W, Y *and* Z *are the current states of this cell, its parent, its left and right son, respectively,* X' *is the next state of this cell;*

$F \subseteq Q$ *is a set of final states.*

q_λ *is the special quiescent state in* Q *and all* δ_l *and* δ_r *are satisfying the condition* $\delta_l(q_\lambda, q_\lambda, q_\lambda, q_\lambda) = q_\lambda$ *and* $\delta_r(q_\lambda, q_\lambda, q_\lambda, q_\lambda) = q_\lambda$. □

To define the computation of A, we specify the global transition function Δ (of A) which is defined on the cartesian product of the global states and input alphabet Σ. A global state of A is a labeled rooted ordered infinite full binary tree. The labels are from the state set Q and each node of the tree represents a cell of A. The initial global state, usually denoted by G_0, has all the nodes labeled by the quiescent state. An accepting global state is a global state with the label of the root cell in F. Let G and G' be two global state of A and $a \in \Sigma$. We define $\Delta(G, a) = G'$, if $\delta_0(U, a, V, W) = U'$ where U, V, W are the labels of the root, its left and right son in G and U' is the label of the root in G', and $\delta(q, X, Y, Z) = q'$ for any cell in A where δ is δ_l or δ_r according to whether the cell is a left cell or a right cell, q, X, Y and Z are the labels of the cell, its parent, its left and right son in G, respectively, and q' is the label of the cell in G'. Furthermore, for any $w \in \Sigma^*$, Δ^k *is defined as*

$$\Delta^k(G,w) = \begin{cases} G, \text{ if } k = 0; \\ \Delta(\Delta^{k-1}(G, x), a), w = xa, \text{ if } k = |w|; \\ \Delta(\Delta^{k-1}(G, w), B), \text{ if } k > |w|. \end{cases}$$

Notice that we always assume that $k \geq |w|$ and the B's are the blank symbols following the input string.

Let ρ be the projection of a global state to its root-label.

Definition 3.2 *A language L is said to be accepted by an ITA* $A = (Q, \Sigma, \delta_0, \delta_l, \delta_r, F)$, *if* $L = \{ w \mid \rho(\Delta^t(G_0, w)) \in F$, *for some* $t \geq |w| \}$. *We say that L is accepted by A in time* $f(n)$ *if L is accepted by A and* $L = \{ w \mid \rho(\Delta^t(G_0, w)) \in F, |w| \leq t \leq f(|w|) \}$. *We also call L a* $f(n)$*-time ITA language. Specially, if* $f(n) = n$, *L is called a real-time ITA language. If* $f(n) = cn + d$, *for some constant c and d, L is called a linear-time ITA language.* □

Definition 3.3 *Function f is said to be ITA constructable if* $\{a^{f(n)} \mid n \geq 0\}$ *is a real-time ITA language.* □

Obviously, the function f(n) = cn + d is ITA constructable for any constants c and d.

Theorem 3.1 *If L is a T(n)-time ITA language, T(n) ≤ f(n) and f(n) is ITA constructable, then L is also a f(n)-time language.* □

From this theorem, we can see that if T(n) is ITA constructable then it makes no differences whether to accept exactly in time T(n) or within time T(n) in defining a T(n)-time ITA language.

4. Real-time ITA languages

The real-time iterative arrays have been studied by many authors. The first question which is naturely posed about the real-time ITA is whether it is more powerful than the real-time IAA. We will answer this question by showing a language which is accepted by an ITA but not accepted by any n dimensional IAA, n > 0.

$$\text{Let } L_d = \{x_1\#x_2\#\ldots\#x_n\$y\$^t \mid n > 0, x_1,x_2,\ldots,x_n,y \in \{0,1\}^+, t=2|y|+1,$$
$$y=x_i \text{ for some } 1\leq i\leq n\}.$$

We call L_d a dictionary language because every word in L_d is a simulation of several key insertions and a successful query.

Lemma 4.1 *L_d is accepted by a real-time ITA.*

Proof : We can construct an ITA A_d which accepts L_d in real-time. The formal construction of A_d is tedious and hard to read. The reader interested in details is suggested to [19]. An informal description of A_d follows.

1. All the nodes of A_d are quiescent initially.

2. For each string x_i, the stream of 0's and 1's flow down a path from the root and is stored on the path one symbol at a time. Its value shows the direction of the path, i.e., if it is '0', then the path goes to the left, if '1' then goes to the right. The second symbol will be stored at the son of the root designated by the stored value in the root. Again, this stored value indicates the next direction of the path. In this way, the values of x_i form a path in A_d.

3. The symbol '#' which terminates x_i will go along the path, clean all the stored values and mark the node pointed by the last value stored on the path.

4. A '$' terminates the insertions of $x_1,\ldots,x_n$.

5. The processing of y determines a path in the same way as the x_i.

6. The first '$' after y checks if the node pointed by the path of y is marked. If it is, a success signal is returned to the root. Otherwise, a fail signal is passed back. □

Lemma 4.2 *L_d is not accepted by any n-dimensional real-time iterative array.*

Proof : Consider arbitary n > 0. For any h > 0 and any polynomial g of degree n, we certainly can find an integer t such that

$$2^{2^t} - 1 > h^{g(3t+1)}$$

Let $k = 3t + 1$. Now, our intention is to show that there are at least $2^{2^t} - 1$ equivalence classes over the relation $R_k(\text{mod } L_d)$.

Consider a string $y \in \{0, 1\}^t$ There are 2^t distinct possible strings y,i.e.,there are 2^t elements in the set $Y = \{y \mid y \in \{0, 1\}^t\}$. Furthermore, we know that there are $2^{2^t} - 1$ different nonempty subsets of Y. Let $X = \{x_1, x_2,..., x_m\}$ and $Z = \{z_1, z_2,...,z_n\}$ be arbitrary two distinct subset of Y, $m,n > 0$. Then, there must exist a y_0 such that $y_0 \in X$ and $y_0 \notin Z$ (or vice versa). We construct two strings

$x = x_1\#x_2\#...\#x_m\$$ and

$z = z_1\#z_2\#...\#z_n\$$,

which are corresponding to the sets X and Z, respectively, where the order of $x_1,...,x_m$ and $z_1,...,z_n$ is irrelevant. Consider

$$xy_0\$^{2t+1} \text{ and } zy_0\$^{2t+1}.$$

Since $y_0 \in X$ and $y_0 \notin L_d$, it is obvious that

$$xy_0\$^{2t+1} \in L_d \text{ and } zy_0\$^{2t+1} \notin L_d.$$

Notice that

$$|y_0\$^{2t+1}| = k,$$

therefore, x and z belong to different equivalence classes of $R_k(\text{mod } L_d)$. Since X and Z are arbitrary, we now can conclude that there are at least $2^{2^t} - 1$ distinct equivalence classes of $R_k(\text{mod } L_d)$. Because

$$2^{2^t} - 1 > h^{g(k)},$$

L_d is not accepted by any n-dimensional IAA by Theorem 2.1. □

Theorem 4.3 *There exist a language which is accepted by a real-time ITA but not accepted in real-time by any n-dimensional IAA.*

Proof : L_d is such a language proved by Lemma 4.1 and Lemma 4.2. □

The following is an obvious consequence of Theorem 4.1.

Theorem 4.4 *The set of the real-time (1D) IAA language is a proper subset of the real-time ITA languages.*

Proof : For every (1D) IAA, we can always construct an ITA which simulates the IAA with one path, e.g., the leftmost path of the ITA. And by the result of Theorem 4.1, we have proved the proper containment. □

It is still open whether real-time nD IAA ($n \geq 2$) languages are properly contained in the real-time ITA languages or they are incomparable.

Lemma 4.5 *If L is accepted by an ITA in time $T(n) > n$, then L is accepted by some ITA in time $T(n) - 1$.*

Proof : Let $A = (Q, \Sigma, \delta_0, \delta_1, \delta_r, F)$ be the ITA which accepts L in time T(n). We construct $A' = (Q', \Sigma, \delta_0', \delta_1', \delta_r', F)$ such that the root cell of A' simulates the root cell and its two sons of A. Every other cell in A' simulates two sons of the corresponding cell in A. See Fig.4.1, the nodes are indexed by the width-first order.

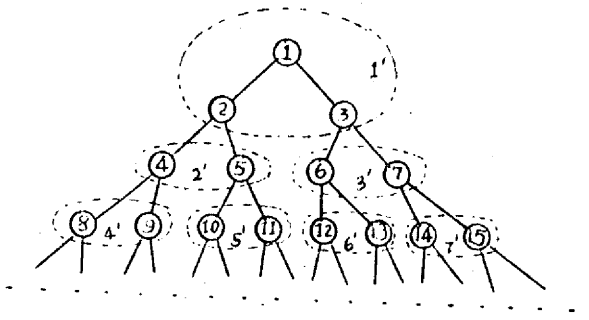


Fig. 4.1

Formally, we define A' as follows. For convenience, we assume $<q_\lambda, q_\lambda>$ and $<q_\lambda, q_\lambda, q_\lambda>$ are equivalent to q_λ.

$$Q' = \{<q_1,q_2> \mid q_1, q_2 \in Q\} \cup \{<q_1,q_2,q_3> \mid q_1, q_2, q_3 \in Q\}$$

$$\delta_0'(<p_0,p_l,p_r>,a,<l_1,l_2>,<r_1,r_2>)$$
$$= <\delta_0(p_0,a,p_l,p_r),\delta_l(p_l,p_0,l_1,l_2),\delta_r(p_r,p_0,r_1,r_2)>;$$

$$\delta_x'(<t_1,t_2>,<p_0,p_l,p_r>,<l_1,l_2>,<r_1,r_2>)$$
$$= <\delta_l(t_1,p_x,l_1,l_2),\delta_r(t_2,p_x,r_1,r_2)>, \text{ for } x = l \text{ or } r,$$

$$\delta_x'(<t_1,t_2>,<p_l,p_r>,<l_1,l_2>,<r_1,r_2>)$$
$$= <\delta_l(t_1,p_x,l_1,l_2),\delta_r(t_2,p_x,r_1,r_2)>,$$
$$x = l \text{ or } r, \text{ for } a \in \Sigma \cup \{*e, \text{ and } p_0,p_l,p_r,l_1,l_2,r_1,r_2,t_1,t_2 \in Q;$$

$$F' = \{<p_0,p_l,p_r> \mid \delta_0(p_0,B,p_l,p_r) \in F\}.$$

It is easy to see that there is a sequence of global states of A obtained by reading $w = a_1a_2...a_n$,

$$G_0 \xrightarrow{a_1} G_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} G_n \xrightarrow{B} \ldots \xrightarrow{B} G_{T(n)-1} \xrightarrow{B} G_{T(n)}$$

if only if there is a corresponding sequence of global states of A'

$$G_0' \xrightarrow{a_1} G_1' \xrightarrow{a_2} \ldots \xrightarrow{a_n} G_n' \xrightarrow{B} \ldots \xrightarrow{B} G_{T(n)-1}' \xrightarrow{B} G_{T(n)}',$$

and $G_{T(n)}$ is an accepting global state in A if and only if $G'_{T(n)-1}$ is an accepting global state in A' because of the definition of F'. So, L is accepted by A in time T(n) if and only if it is accepted by A' in time T(n)-1. □

Theorem 4.6 *If L is a T(n) + c time ITA language, $T(n) \geq n$ and c is a constant, then L is a T(n)-time ITA language. In particular, if L is an n + c time ITA language, then L is a real-time ITA language.* □

The next theorem shows a property of the real-time ITA languages. This

property can be used to show that some languages are not real-time ITA languages. The proof of this theorem is similar to that of [4].

Theorem 4.7 *Let L be a real-time ITA language. There is a constant h such that for any $k > 0$ there are no more than*

$$h^{2^k}$$

equivalence classes of R_k(mod L).

Proof : Let A be the ITA which accepts L. This proof is based on a simple fact that the next k states of the cell at root are only determined by the current states of the cells at the top $k+1$ levels of the ITA and the next k input symbols.

Consider the two sequences of the global states

$$G_{i_0} \xrightarrow{a_1} G_{i_1} \xrightarrow{a_2} \cdots \xrightarrow{a_t} G_{i_t}$$

and

$$G_{i_0}' \xrightarrow{a_1} G_{i_1}' \xrightarrow{a_2} \cdots \xrightarrow{a_t} G_{i_t}'$$

where $a_1a_2...a_t \in \Sigma_t$, $0 < t \leq k$. If the states of the cells at the top $k+1$ levels of the global states G_{i_0} and G_{i_0}' all the same, then we can easily prove that G_{i_1} and G_{i_1}' have the same states of the cells at top k levels. Similarly, G_{i_2} and G_{i_2}' have the same states of the cells at the top $k-1$ levels,..... G_{i_t} and G_{i_t}' have the same states at top $k-t+1$ levels. Since $t \leq k$, we have

$$\rho(G_{i_t}) = \rho(G_{i_t}').$$

We defined that $G =_k G'$ if and only if G and G' have the same states at the top $k+1$ levels. $=_k$ is obviously an equivalence relation. We know that there are

$$1+2+2^2+\cdots+2^k = 2^{k+1}-1$$

cells at the top $k+1$ levels of the ITA. Let n be the number of states in Q. Then there are at most

$$n^{2^{k+1}-1}$$

equivalence classes of $=_k$.

Since $w\ R_k\ w'$ if Let

$$\Delta(G_0^{|w|}, w) = G \quad \text{and} \quad \Delta(G_0^{|w'|}, w') = G' .$$

It is easy to see that $w\ R_k\ w'$ if $G =_k G'$. Let $h = n^2$. *Hence, there are no more then*

$$h^{2^k}$$

equivalence classes of R_k(mod L). □

Now we define

$$L_d(K) = \{x_1\#x_2\#...\#x_n\#\$y\$^t \mid n > 0,\ x_1,x_2,...,x_n,y \in \{0,1,...,K-1\}^*, t = 2|y|+1,$$
$$y = x_i \text{ for some } 1 \leq i \leq n\}.$$

Notice that the language L_d we defined before is actually $L_d(2)$.

Lemma 4.8 *$L_d(9)$ is not a real-time ITA language.*

Proof : Similar to the proof in Lemma 4.2, we can show that there are at least

$$2^{9^t}-1$$

equivalence classes of $R_k(\text{mod } L_d(9))$, where $k = \mathcal{S}t$. For big enough k (or t), we have

$$2^{9^t}-1 > k^{2^k}.$$

By Lemma 4.3, $L_d(9)$ is not a real-time ITA language.

Theorem 4.9 *Real-time ITA languages are incomparable with context free languages.*

Proof : We change the definition of $L_d(9)$ such that $y^R = x_i$ rather than $y = x_i$, for some $1 \le i \le n$. Then this language is not a real-time ITA language proved basically by the same argument. But we can easily construct a nondeterministic push-down automaton to accept it. on the other hand, we know that

$\{ww \mid w \in \Sigma^*\}$

is an real-time IAA language, therefore, a real-time ITA language but not a CF language. So, they are incomparable. □

5. Linear-time ITA languages

It is interesting to see that the deterministic ITA can compute the Satisfiability problem in linear time. This is not surprising since the number of cells used in the ITA is growing exponentially as the number of the Boolean variables is increasing linearly. But it is still interesting to have a deterministic program on a simply defined device to solve the problems such as the Satisfiability problem in linear time.

Let $X = \{x_1, x_2, ..., x_n\}$ be a set of Boolean variables. A truth assignment for X is a function $t : X \to \{T,F\}$. We call x or $\bar{x}$ to be a literal over X if $x \in X$. A clause over X is a set of literals over X. The truth value of a clause C is the disjunction of the literals in C. The truth vale of a collection of clauses E is the conjunction of the clauses in E. Given a collection of classes E, the Satisfiability problem is whether there is truth assignment t for X such that the truth value of E is true [9].

We use the reversed binary coding of i for literal x_i and a minus sign following it for literal $\bar{x}_i$. Commas are used to separate the literals. Parenthese are used to separate the clauses. For example,

$$(1,01\text{-},11)(001\text{-},01,101,1\text{-})$$

is the coding of the collection of clauses

$$\{x_1,\bar{x}_2,x_3\}\{\bar{x}_4,x_2,x_5,\bar{x}_1\}$$

Example *The Satisfiability problem can be computed by an ITA in linear time.*

Proof of the example: The formal construction and proof are given in [19]. Here, we only outline the idea.

We define level i of the ITA ($i \ge 0$) to be the set of all cells at the distance i from the root.

Let $X = \{x_1, x_2, ..., x_n\}$ be the set of variables. Each level from level 1 to level

n corresponds to a variable in X. Level i corresponds to x_i, $1 \leq i \leq n$. All the left cells are assumed to have value False and all the right cells assumed to have value True. Thus, each path of the ITA represents a truth assignment for X, There are 2^n possible distinct truth assignment for for X and we have 2^n distinct path (to level n) corresponding to them. When an input word which is coded as described above is read in, the same Boolean operations are performed on every path, i.e., on every possible truth assignment. After completing all the given operations, the results are sent to the root along each path. The given collection of clauses is satisfiable if and only if there is a "True" result from some path.

The following technical details are worth mentioning.

(A) Given a reversed binary number followed by a separator, the ITA can find out the corresponding level in the following way. When the reversed binary coding flows down a path, every node subtracts 1 from the coding. For example, "101" would change to "001" and "00001" would change to "11110" after those codes flowing through a cell. The cell would not know that a number has been decreased to zero until the separator following this number comes. The first cell which becomes less than zero will be marked to be an operand of a Boolean operation.

(B) The '-' sign following a binary coding will cause the complement of the assumed truth value produced. The separators after the second operand in a clause case the Boolean additions to be carried on. The separator brings the operand it first encounters to the cell where the other operand is stored. The operation is performed here and the result will replace the stored value. The Boolean multiplications are operated in the similar way. Since there are two levels of operations, we need markers to distinguish the Boolean multiplication operands from the Boolean addition operands. When each clause is ended, the ITA will mark the truth value of this clause to be an operand of Boolean multiplication.

(C) After the input string is ended (B's are assumed), the results from all the pathes are sent back to the root. When two paths are merged, the result are "or"ed together. Finally, the result at the root cell will be the answer to the question.

Now, let's consider the time consumed for this computation. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables and S be the coding of E, the collection of clauses over X. Without loss of generality, we assume all the variables in X are used in E. Then the last symbol of the input will not go further than depth n of the ITA and the result will travel not more than n cells to reach the root. The computation time on the input S is

$$T(S) \leq |S| + 1 + 2n \leq 3|S| + 1,$$

which is linear to the length of the input. □

Now, we generalize the previous result. We will show that T(n)-time computation on a nondeterministic Turing Machine (NTM) can be simulated by a cT(n)-time deterministic ITA for some $c > 0$. Here again, we see a trade-off between the number of cells and the computing time.

It is a well known fact that every one-head one-tape TM can be simulated by a two-stack machine. Because the tape symbols on the left of the head and the

ones on the right of the head can be placed on the two stacks, respectively. The actions of the head become the pushings and poppings of the stacks. The simulation of an NTM by an ITA will turn out to be the simulation of a two-stack machine.

An implementation of the stack by an IAA has been given in [10]. However we will give our own very simple algorithm for this task after formalizing the main theorem.

Theorem 5.1 *Any T(n)-time NTM language can be accepted by an ITA in cT(n) time, c > 0.*

In the following, we only consider the one-head one-tape TM's. It is fairly easy to extend this result to the multihead and multitape machines. To prove this theorem, we need the following lemmas.

Lemma 5.2 *Any t computations of an NTM with at most k choices can be simulated by ct computations of an NTM with at most 2 choices.*

Proof : Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be an NTM with at most k choices for any move. And $M' = (Q', \Sigma, \Gamma', \delta', q_0, B, F)$ is an NTM with at most 2 choices, which simulates M as follows. Let the i^{th} transition rule of M be

$$\delta(q_{i_0}, a_{i_0}) = \{(q_{i_1}, a_{i_1}, x_{i_1}), (q_{i_2}, a_{i_2}, x_{i_2}), \ldots, (q_{i_j}, a_{i_j}, x_{i_j})\},$$

for $q_{i_0}, q_{i_1}, \ldots, q_{i_j} \in Q$, $a_{i_0}, a_{i_1}, \ldots, a_{i_j} \in \Gamma$, $X_{i_1}, X_{i_2}, \ldots, X_{i_j} \in \{L, R, S\}$, $0 \leq j \leq k$.

If $j \leq 2$, then we have the same rule in δ'. Otherwise, δ' has the following corresponding rules,

$$\delta'(q_{i_0}, a_{i_0}) = \{(q_{i_1}, a_{i_1}, X_{i_1}), (t_{i_1}, a_{i_0}, S)\},$$

$$\delta'(t_{i_1}, a_{1_0}) = \{(q_{i_2}, a_{i_2}, X_{i_2}), (t_{i_2}, a_{i_0}, S)\},$$

.....

$$\delta'(t_{i_{j-2}}, a_{i_0}) = \{(q_{i_{j-1}}, a_{i_{j-1}}, X_{i_{j-1}}), (q_{i_j}, a_{i_j}, X_{i_j})\}.$$

The new state set Q' includes Q and the new states which are used in the decomposition of the multichoices (≥ 3) into double choices.

It is clear that each move of M can be simulated by M' with at most *k-1* moves. So, *(k-1)t* operations of M' is enough to simulate *t* operations of M. □

Lemma 5.3 *Any NTM can be simulated by a nondeterministic two stack machine (NTS). Moreover, t operations of the NTM can be simulated by no more than ct steps.*

Proof : This simulation is a well-known fact, see e.g. [12]. The simulation can be accomplished by at most 3 steps for a move of the NTM. So, *t* moves need at most $3t$ simulation steps. □

Proof of Theorem 5.1 :

By Lemma 5.2 and 5.3, we know that if a language L is accepted by an NTM in time *f(n)*, then there exists a nondeterministic two-stack machine (NTS) with at most two choices of every transition which accepts L in time $cf(n)$, $c > 0$. To prove Theorem 5.1, it suffices to show that any g(n)-time NTS language is

also a $kg(n)$-time ITA language, for some $k > 0$.

The deterministic ITA simulates the NTS as follows. At the beginning, the root is acting as the control unit and every path from the root is functioning as two stacks. Note that each node has two channels for the stacks, so every path has a pair of stacks working along it. All the paths are doing exactly the same job provided no transition has more than one choice. When two choices are available for the next move during the simulation, the two sons of the root become two control units and the stacks are pushed one level down on every path. Thus, each son of the root, together with the stacks below it, simulates one of the choice. Now we have two branches of simulation. Whenever two choices of moves appear at a branch of the simulation, the ITA will break it into two branches in the same way as described above except at the different levels. Thus, there may exist many control units and many pairs of stacks corresponding to them. They are working parallelly and each one simulating a different approach of the transitions of the NTS. When some control unit finishes the computation, if the result is positive (acceptance), the result will be sent to the root at the speed of one level up at a time unit without any delay. If the result is negative (nonacceptance), then the signal will be waiting at any node along the path to the root until the signal from the other branch comes over. In such way, a positive signal comes to the root if and only if there is one branch of simulation which is ended with acceptance, and a negative signal arrives at the root if and only if all the simulations get negative results.

It is obvious that this ITA accepts the same language as the simulated NTS since it simulates all the possible sequences of transitions of the NTS and only of them, too.

Consider the time consumed by the simulation. Let t be the operation time for the NTS to accept a word. Then the simulation need at most t push-downs for multiple choice transitions (≥ 3), t pure simulation steps and t operations to send the result signal back for the latest one. So, at most $3t$ time units are needed.

Combining with the result from Lemma 5.2 and 5.3, we have proved the theorem. □

In the proof of Theorem 5.1, we use each path of the ITA to simulate stacks. A path is actually a linear iterative array (IAA). To show how it works, we now give our own idea which shows how the stack can be easily implemented by an IAA.

Each cell of the IAA has three registers, so it can have up to three stack elements stored in it. The leftmost cell is the top of the stack. For each cell,

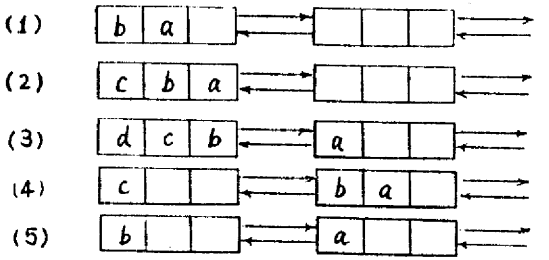
(1) if there are three elements in the cell, the right one will be sent to the right neighbor; (Expansion rule)

(2) if there is only one element in the cell, it will get one from the right neighbor provided there is one. (Contraction rule)

Fig.5.1 shows an example of the implementation of the stack.

To verify that this stack-simulation works correctly, it suffices to show (a) any cell will never be overloaded; (b) the stack will never be broken, i.e., no

empty cells will occur between nonempty ones. Clearly, (a) is guaranteed by the Expansion rule and (b) is confirmed by the Contraction rule.



(1) Stack has two elements a and b. (2) Push c.
(3) Push d. (4) Pop d. (5) Pop c.

Fig. 5.1

In the last section, we showed that the real-time ITA languages are incompatible with the context free languages. But this is no longer true to the linear-time ITA languages. We know that any context free language L can be generated by a grammar G in 2-standard form [p116 of 11]. A grammar $G = (V, \Sigma, P, S)$ is in 2-standard form if each production rule is in the form

$$A \to a\alpha \quad a \in \Sigma, \alpha \in (N - \{S\})^* \; 0 \le |\alpha| \le 2 \text{ or } S \to \lambda$$

From this grammar G, it is easy to construct a nondeterministic push-down automaton A which accepts the same language L in real time. Since G is in 2-standard form, each action of A handles at most two stack symbols. Then A can be easily simulated by a two-tape nondeterministic Turing Machine in $2n$ time. By Theorem 5.1, another result follows.

Corollary 5.4 *Context free languages are properly included in linear-time ITA languages.*

□

Theorem 5.5 *Linear-time ITA languages are closed under Boolean operations.*

Proof : Let L_1 and L_2 be two linear-time ITA languages accepted by ITA A_1 and A_2, respectively. To show $L_1 \cup L_2$ and $L_1 \cap L_2$ are linear-time ITA languages, we construct a new ITA A which has two channels for every cell. One channel simulates A_1 and the other simulates A_2. The two result are merged at the root cell by the required Boolean operation (union or intersection). Then it is clear that L(A) is the union or intersection of L_1 and L_2 and it is a linear-time ITA language.

Let $L = L(A)$, $A = (Q, \Sigma, \delta_0, \delta_1, \delta_r, F)$ be an ITA and A accepts L in linear time $f(n) \le cn + d$. Let $\bar{L} = \Sigma^* - L$. Then it is easy to construct an ITA $\bar{A}$ by simulating A and a f(n) time ITA at same time. The result is the reversed result of A in time f(n). □

Theorem 5.6 *Linear-time ITA languages are closed under concatenation.*

Proof : Let $L_1 = L(A_1)$ and $L_2 = L(A_2)$, respectively. Now, we are

constructing an ITA A to accept the language L_1L_2. For a given input string w, A will test whether $u \in L_1$ *and* $v \in L_2$ for all possible decompositions of w such that $w = uv$. The structure of ITA A is outlined in Fig.5.2. The rightmost path of A destributes the input to the subautomata S_0, S_1, ... For i = 0, 1, ..., the sub ITA S_i tests the decomposition $w = uv$ where $|u| = i$. The root of S_i sends

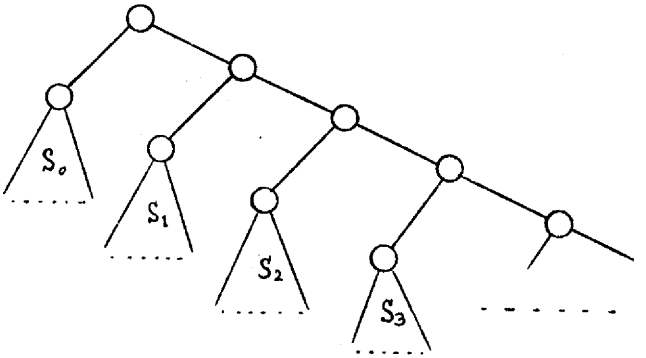


Fig. 5.2

the first i input symbols to its left son and the remainder to its right son. The left son and its descendants will simulate A_1 on the first i input symbols, and the right son and the subtree rooted from it will be working on the rest of the input with the rules of A_2. S_i accepts the input if and only if both of its sons get the positive answers. A accepts the input if and only if one of the sub ITA's S_0, S_1, ..., S_n accepts it.

It is worth mention how a sub ITA divides the string correctly for its left and right son, i.e., for the sub ITA S_i, the difficulty is how it counts i symbols for its left son and the remainder for the right son. We use a tagging technique. At beginning, every node on the rightmost path of A is assumed storing a tag. The input symbols are all untagged originally. The first untagged symbol passing a cell of the rightmost path will bring the tag away and become tagged itself when it goes to the right. Thus, S_0 will receive no tagged symbol. S_1 will receive one. S_2 will receive two,.....etc. In the sub ITA S_i, $0 \le i \le n$, the tagged symbols will be sent to the left son as the input string to A_1 and untagged symbols will be sent to the right son to be tested whether in L_2 or not.

We omit the tedious but quite straightforward formal construction of A and the details of the proof. □

Theorem 5.7 *Linear-time ITA languages are closed under inversed morphism.*

Proof : Let L be a linear-time language on Σ accepted by ITA A. h is a morphism : $\Sigma \to \Sigma^*$ and $h^{-1}(L) = L'$. We construct an ITA A' such that the root works as two processors, one is functioning as a translator which maps any symbol $a \in \Sigma$ to h(a), and the other together with all other cells simulates the ITA A. Then the proof becomes straightforward. □

Theorem 5.8 *Linear-time ITA languages are closed under Kleene closure.*

Proof : Let L be a linear-time ITA language accepted by A. We construct an ITA A' such that $L^{\bullet} = L(A')$. When constructing A', we expand the idea of the proof of Theorem 5.7.

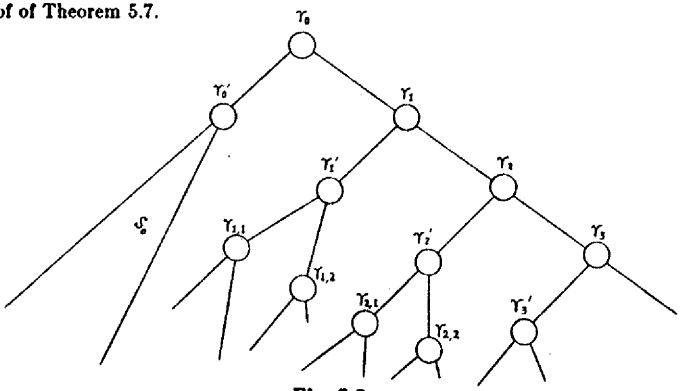


Fig. 5.3

See Fig.5.3. The left sub ITA S_0 will simulate ITA A on the whole input string. The rightmost path of A' is the main path. The input stream goes through this path and is tagged in the same way as in Theorem 5.7. Let n be the length of the input string. For example, when the input symbols going through cell r_i to cell r_i' ($i \leq n$), there will be i symbols tagged and n-i untagged. The tagged symbols will go to cell $r_{i,1}$ and the others will go to $r_{i,2}$. The sub ITA's with root $r_{i,1}$ and $r_{i,2}$ will recursively repeat the same processes as it is started from $r_0$except they will not work on the whole input string but only part of it. r_i' will send back an acceptance message if and only if both $r_{i,1}$ and $r_{i,2}$ accept. r_0 will enter a final state if either S_0 enters a final state of A or an acceptance message comes from the main path.

Since A' tried all the possible partitions of the input string, $w \in L(A')$ if and only if $w \in L^n$ for some $n \geq 0$. So, A' is the ITA we wanted to construct.

The proof that A' is a linear-time ITA is omitted. □

6. Generalization of the ITA

The ITA defined in Section 3 are actually the binary ITA, i.e., every cell has and only has two other cells as its two sons. In this section, the more general concept of arbitrary n-ary ITA will be introduced. The main result in this section is the hierarchies of the real-time ITA languages. We will show that for any $0 < m < n$ the family of the real-time m-ary ITA languages is properly contained in the family of the real-time n-ary ITA languages. To prove this result, several properties of the general ITA will be given. Although higher arity ITA are faster than the low arity ITA, the families of the linear-time languages of them are equal. So, only the real-time languages of the general ITA will be concerned in this section. First, we give the formal definition of the general ITA.

Definition 6.1 *An n-ary iterative tree automaton A is a quadruple* (Q, Σ, D_n, F) *where*

Q is a finite set of states;

Σ is the input alphabet;

$D_n = \{\delta_0, \delta_1, ..., \delta_n\}$ is a set of transition functions; δ_0 is the function of the root cell, $\delta_1, ..., \delta_n$ are the functions of the first cells, second cells, ..., the n^{th} cells (the cells which are the first sons, second sons, ..., n^{th} sons of some cells), respectively;

$F \subseteq Q$ is a set of final states.

q_λ is the special quiescent state in Q and all δ_i, $1 \le i \le n$, satisfy the condition $\delta_i(q_\lambda, q_\lambda, q_\lambda, ..., q_\lambda) = q_\lambda$. □

Notice that a 1-any ITA is a linear IAA. All the other definitions for the binary ITA in Section 3 are also valid for the general ITA.

It is an obvious fact that for any $0 < m < n$, the real-time n-ary ITA is at least as powerful as the real-time m-ary ITA, since the n-ary ITA can simulate m-ary ITA with only part of their branches. The n-ary ITA and n^k-ary ITA have some special relation. Any k steps of internal computations of the n-ary ITA can be simulated by an n^k-ary ITA in 1 step. The converse proposition is also true. If the external input is considered as well, we get the following theorem.

Theorem 6.1 *Let L be a language in Σ^*, $\# \notin \Sigma$, and*

$$L_{k\#} = \{a_1 \#^{k-1} a_2 \#^{k-1} ... \#^{k-1} a_n \mid a_1 a_2 \cdots a_n \in L\}.$$

Then L is accepted by an n^k-ary ITA in real time if and only if L is accepted by an n-any ITA in real time.

Proof : Assume $L_{k\#}$ is accepted by an n-ary ITA A in real time. We can construct an n^k-ary ITA A' in the following way. Each cell of A' simulates the operations of $1+n+...+n^{k-1}$ cells at k consecutive levels of A.

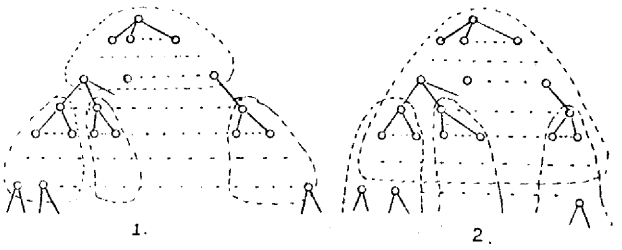


Fig. 6.1

Fig. 6.1.1 shows the correspondence of A and A'. But in order to simulate k operations of the k levels $1+n+...+n^{k-1}$ cells of A by one cell in one operation correctly, each cell of A' should store the state information of $1+n+...+n^{2k-2}$ cells at $2k-1$ levels of A. This is shown clearly in Fig. 6.1.2. An example is shown in Fig. 6.2 where a 4-ary ITA is simulating a binary ITA. Given the current states of

S_1', S_2', S_6', S_7', S_8 and S_9 to compute the next state of S_2', we certainly need the current state of cell 5. When the root cell of A' is doing the simulation, it always assumes an input string $a\#^{k-1}$ when it reads a input symbol a. Then it is easy to prove that A' accepts L.

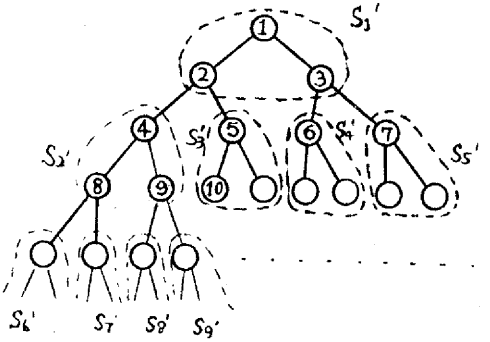


Fig. 6.2

Now assuming that L is accepted by an n^k-ary ITA A, we construct an n-ary ITA A' which accepts $L_{k\#}$. A' operates k time units a cycle. Look at Fig. 6.1.1 and interchange A and A'. Every block of A' simulates a cell of A. At time $tk+1$ $(t \geq 0)$, each top cell of the blocks changes to a new state according the transition rules of A. If the result is not quiescent, this state information is sent down one level at a time until reaches the bottom of the block. When the state information is sent down, each cell being passed adds a tag i $(1 \leq i \leq n)$ if this cell is a i^{th} cell. At the same time (from time $tk+2$ *to* $(t+1)k$), the new state information is passed up $k-1$ levels. So, at time $(t+1)k+1$, the transition of A' is defined as

$$\delta_i'(q, (p, i_1, i_2, \ldots, i_{k-1}), \langle q_{i_1}, q_{i_2}, \ldots, q_{i_{n^{k-1}}}\rangle, \ldots, \langle q_{i_{(n-1)n^{k-1}+1}}, \ldots, q_{i_{n^k}}\rangle)$$
$$= \delta_{i_1 n^{k-1} + i_2 n^{k-2} + \ldots + i_{k-1} n + i}(q, p, q_{i_1}, \ldots, q_{i_{n^k}}) \quad \text{if } 0 < i \leq n;$$
$$\delta_0'(q, a, \langle q_{i_1}, \ldots, q_{n^{k-1}}\rangle, \ldots, \langle q_{i_{(n-1)n^{k-1}+1}}, \ldots, q_{i_{n^k}}\rangle)$$
$$= \delta_0(q, a, q_{i_1}, \ldots, q_{i_{n^k}}) \quad \text{if } i = 0.$$

The details of the construction and proof are left to the reader. □

We use the notation $L_{\text{ITA}}(n)$ to represent the family of the real-time n-ary ITA languages.

Corollary 6.2 *If* $L_{\text{ITA}}(m) = L_{\text{ITA}}(n)$, *then* $L_{\text{ITA}}(m^k) = L_{\text{ITA}}(n^k)$ *for any* $k > 0$.

Proof : $L \in L_{\text{ITA}}(m^k) \Leftrightarrow L_{k\#} \in L_{\text{ITA}}(m) \Leftrightarrow L_{k\#} \in L_{\text{ITA}}(n) \Leftrightarrow L \in L_{\text{ITA}}(n^k)$. □

In Section 4, we showed that $L_d(9)$ is not a real-time binary ITA language.

It is fairly easy to see that $L_d(n) \in L_{TTA}(n)$. So, we know that $L_{TTA}(2)$ is properly contained in $L_{TTA}(9)$. To prove that $L_{TTA}(m) \subset L_{TTA}(n)$ for any $0 < m < n$, we need the generalized concept of Theorem 4.7. The proof of it is similar to that of Theorem 4.7.

Theorem 6.3 *Let L be a real-time n-ary ITA language. Then there is a constant h such that for any $k > 0$, the number of the equivalence classes of R_k(mod L) is less than*

$$h^{n^k} \qquad \square$$

This theorem is essential to the proof of the ITA hierarchies.

Example 6.1

(a) For all $0 < m^3 < n$, $L_{TTA}(m) \subset L_{TTA}(n)$.

(b) For all $k > 0$, $L_{TTA}(2^k) \subset L_{TTA}(2^{k+1})$.

The proof of the first part is easy, since we can prove that $L_d(n)$ is not an real-time m-ary ITA language by the similar method of Lemma 4.8. To prove (b), we assume that the two families of languages are equal. By Corollary 6.2, we get

$$L_{TTA}(2^{k^2}) = L_{TTA}(2^{k(k+1)}) \text{ and}$$
$$L_{TTA}(2^{k(k+1)}) = L_{TTA}(2^{(k+1)^2})$$
$$L_{TTA}(2^{k(k+2)}) = L_{TTA}(2^{(k+1)(k+2)})$$
$$\cdots\cdots\cdots$$
$$L_{TTA}(2^{3k^2}) = L_{TTA}(2^{3k(k+1)})$$

Since $k(k+i) < k(k+i+1) < (k+1)(k+i)$ is true for any $0 < i < 2k$, the arity of every left hand side from the third equation is between the arities of the two sides of the previous equation. So all the families in the above equations are equal. Hence,

$$L_{TTA}(2^{k^2}) = L_{TTA}(2^{3k(k+1)}).$$

But

$$0 < (2^{k^2})^3 < 2^{3k(k+1)}.$$

We have a contradiction to (a). Thus (b) holds.

Now we are ready for the main result of this section.

Theorem 6.4 *For any $n > 0$, $L_{TTA}(n) \subset L_{TTA}(n+1)$.*

Proof :Assume that $L_{TTA}(n) = L_{TTA}(n+1)$. *Then* $L_{TTA}(n^k) = L_{TTA}((n+1)^k)$ for any $k > 0$. Chose k be the integer such that

$$(\frac{n+1}{n})^k > 4 .$$

Then it is easy to show that there exist a t such that

$$n^k < 2^t < 2^{t+1} < (n+1)^k .$$

Since we know that $L_{TTA}(2^k) \subset L_{TTA}(2^{k+1})$, we have $L_{TTA}(n^k) \subset L_{TTA}((n+1)^k)$. *This is a contradiction.* □

Corollary 6.5 *For all integers* $0 < m < n$, $L_{TTA}(m) \subset L_{TTA}(n)$. □

Acknowledgment

The helpful suggestions and comments of I.P.Fris are grateful appreciated.

References

[1] J. I. Bentley and H. T. Kung, A Tree Machine for Searching Problems, Dept. Compt. Sci., Carnegie-Mellon Univ., Tech. Rep, Aug. (1979).

[2] C. Choffrut and K. Culik II, On Real-Time Cellular Automata and Trellis Automata, Research Report F114, Institute fur Informationsverarbeitung, Technical University of Graz, (1983).

[3] K. Culik II, J. Gruska and A. Salomaa , Systolic Automata for VLSI on Balanced Trees, Acta Informatica 18 (1983), 335-344.

[4] S. N. Cole, Real-Time Computation by n-Dimensional Iterative Arrays of Finite-State Machines, IEEE Trans. on Comp. 18 (1969) 349-365.

[5] K. Culik II and J. Pachl, Folding and Unrolling Systolic Arrays, ACM SIGACT-SIGOPS Symposium on Principles of Ditributed Computing (Ottawa, August 1982) 254-261.

[6] K. Culik II, A. Salomaa and D. Wood, VLSI Systolic Tree Acceptors, Research Report CS-81-32, University of Waterloo, Waterloo, Ont., Canada, RAIRO Theoretical Informatics, to appear.

[7] C. R. Dyer and A. Rosenfeld, Triangle Cellular Automata, Inf. and Control 48 (1981) 54-69.

[8] P. C. Fischer, Generation of Primes by a One-Dimensional Real-Time Iterative Array, J. ACM, 12 (1965) 388-394.

[9] M. R. Garey and D. S. Johnson, Computers and Intractability, W.H.Freeman and Company (1979).

[10] L. J. Guibas and F. M. Liang, Systolic Stacks, Queues and Counters, 1982 Conference on Advanced Research in VLSI, M.I.T..

[11] M. A. Harrison, Introduction to Formal Language Theory, Addison-Wesley Publishing Company (1978).

[12] J. Hopcroft and J. Ullman, Introduction to Automata Theory, Languages, and Computations, Addision-Wesley (1979).

[13] H. T. Kung and C. E. Leiserson, Systolic Arrays (for VLSI), Sparse Matrix Proceedings (1978).

[14] H. T. Kung, Why Systolic Architecture?, Computer Magazine, Jan. (1982).

[15] C. E. Leiserson and J. B. Saxe, Optimizing Synchronous Systems, Proceedings of the 22nd Annual Symposium on Foundation of Computer Science, IEEE Computer Science, (1981) 23-36.

[16] T. A. Ottmann, A. L. Rosenberg and L. J. Stockmeyer, A Dictionary Machine (for VLSI), IEEE Trans. on Computers 31 (9) (1982) 892-897.

[17] A. R. Smith III, Real-time Language Recognition by One-Dimensional Cellular Automata, J. Comput. System Sci., 6, (1972) 233-253.

[18] A. Salomaa, Formal Languages, Academic Press (1973).

[19] Yu S., Doctoral Thesis, in preparation (1984).