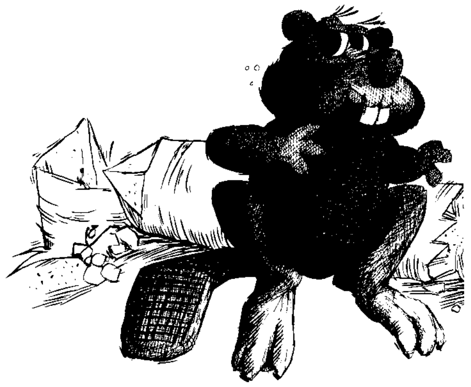


Real Time, Pseudo Real Time And Linear Time ITA

Karel Culik II
Sheng Yu

Research Report CS-86-03

January 1986

REAL TIME, PSEUDO REAL TIME AND LINEAR TIME ITA

Karel Culik II & Sheng Yu

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

Research Report CS-86-03

ABSTRACT

A k-ary iterative tree automaton (k-ary ITA) is a potentially infinite synchronous network of finite automata structured as k-ary tree with serial input and output at the root of the tree. The computational power of ITA in real-time, pseudo real-time and linear-time is compared. The pseudo real time is a new notion and means that every cell of an ITA makes a fixed number of computational steps for each input symbol. It is shown that every linear-time ITA can be simulated either by a pseudo real-time ITA of the same arity or by a real-time ITA of a higher arity. As an example, an ITA implementation of real-time infinite memory is described.

1. Introduction

A linear array of processors is a type of parallel architecture that is one of the easiest to implement. It is also a type of architecture which has been extensively studied since the sixties. A linear array of finite automata with parallel input is called a cellular automaton (see e.g.[7]); one with serial input (and output) at the leftmost cell is usually called an iterative array, see [2], [1]. In our earlier paper [4] we introduced a generalization of iterative arrays called iterative tree automata (ITA). It is a tree structured network of finite automata with serial input-output at the root of the tree. If the underlying structure of an ITA is a infinite k-ary tree, then we call this ITA a k-ary ITA. Actually, an iterative array is nothing but a unary ITA.

For simplicity we consider such automata only as language acceptors. A language L is said to be accepted by ITA A in time $T(n)$ if a string w is in L if and only if after reading w the root cell of A enters an accepting state within time $T(n)$, where n is the length of w. Specifically L is accepted by A in real time if the root cell enters an accepting or rejecting state immediately after reading the last (rightmost) symbol of input string w. Besides real time, we are mainly interested in linear time, i.e. time $T(n) = cn$ for some $c \geq 1$. A number of results about real-time and linear-time ITA have been shown in [4], we mention just one: Every language recognized by a deterministic Turing machine in time $T(n)$ can be recognized by a deterministic binary ITA in time $cT(n)$ for some $c \geq 1$. A practically interesting case of (binary) ITA with logarithmic depth has been studied in [3]. In this case an ITA A processing input w of length n can use only cells up to depth $\log n$. Hence A uses only $O(n)$ processors for an input of length n which is the same as a linear-time iterative array does.

The results in [4] indicated that for each $k \geq 1$ linear-time k-ary ITA are considerably more powerful than real-time k-ary ITA. Here we show the following rather surprising results. Every linear-time ITA can be simulated by a pseudo real-time ITA of the same arity. Pseudo real-time means that the systems reads an input symbol in every c^{th} computation step. i.e. that its internal clock is c times faster than that of its host. Or alternatively, we can obtain a real-time simulation if we are willing to increase the arity of the tree structure. For every linear-time k-ary ITA we can construct an equivalent real-time k^2-ary ITA.

If we consider the family of all ITA (of all positive integer arities) then we have the following result. The families of languages accepted by linear-time ITA, pseudo real-time ITA and real-time ITA coincide. The direct implication of this result is that every linear-time *NTM* language can be accepted by an ITA in real time, e.g. every *CF* language can be accepted by an ITA in real time.

As an example we describe an ITA implementation of real-time infinite memory. Originally, this example was proposed by the first author and Fris. We assume that the memory has an infinite number of locations addressed by all strings in $\{0,1\}^*$ and each address is read sequentially bit by bit. We also assume that at each location a finite amount of information, formally a symbol from a finite alphabet, can be stored. We show that this device can store and retrieve information to and from any location in real time. Specifically, when the ITA reads the last symbol of $w\#$ the information stored at address w is already available at the root.

A more practically interesting version of this example might be a (potentially infinite) hierarchy of memories. In this case each node of one ITA is a conventional memory for n k-bit words with an $\log n$-bit address. "Total" address of a location is then comprised of a variable-length prefix that is the address of the node and the "local" address. Storage and retrieval would work again in real time, but of course a longer prefix must be used for a more remote memory block.

2. General k-ary ITA's

Informally, an k-ary ITA is a systolic automaton with an underlying k-ary tree structure. Every cell of this system has k children. The root cell is a special cell which is the only place where the system interfaces with the external world. A cell which is the i^{th} child of a cell is called an *i_cell*. There are $k+1$ transition functions, denoted by $\delta_0, \delta_1, \cdots, \delta_k$ for a k-ary ITA. δ_0 is the transition function of the root cell. δ_i is the transition function of i*—cell*s, for $i = 1,2,...,k$, respectively. The next state of a cell is the function of the current states of this cell, its parent, and k children, except that the next state of the root cell also depends on the current input symbol.

Following is the formal definition.

Definition 2.1 *An k-ary ITA $A = (Q, \Sigma, \Delta_k, F)$ is the following :*

Q is the finite set of states;

Σ is the input alphabet;

$\Delta_k = \{\delta_0, \delta_1, \cdots, \delta_k\}$ is a set of transition functions where δ_0 is the transition function of the root cell, δ_i is the transition function of every i_cell, $1 \leq i \leq k$.

$F \subseteq Q$ is the set of final states.
$q_\lambda \in Q$ is the special quiescent state and $\delta_i(q_\lambda, \cdots, q_\lambda) = q_\lambda$ for all $1 \leq i \leq k$.

A linear iterative array is a special case of ITA's, i.e., it is a unary ITA.

A word w is said to be accepted by an ITA A if A reads w and the root cell of A eventually enters a final state at some time $t \geq n$, where n is the length of w. A language L is said to be accepted by an ITA A if every word of L is accepted by A and they are the only words accepted by A. Then we also say that A is an ITA language. A ITA language L is a real-time k-ary ITA language if every word w of L is accepted by A in $n = |w|$ (the length of w) steps, i.e. exactly at the time when the last input symbol is read. An ITA language L is a linear-time k-ary ITA language if there exists a constant c such that every word of L is accepted by A in cn steps. Note that the acceptance in exactly cn time is equivalent to the acceptance in at most cn time. This follows from the fact that the linear functions are computable by ITA's (see [1]).

Definition 2.2 *For $L \subseteq \Sigma^*$, positive integer c and special symbol $\# \notin \Sigma$, let $L_{c\#} = \{a_1\#^{c-1}a_2\#^{c-1} \cdots \#^{c-1}a_n\#^{c-1} \mid a_1a_2 \cdots a_n \in L\}$. L is called a pseudo real-time k-ary ITA language if $L_{c\#}$ is accepted by an ITA in real-time for some integer $c > 0$.*

Intuitively, a pseudo real-time ITA language is a language which is accepted in real time by a variation of ITA whose internal computation speed is c times as fast as its input speed.

Pseudo real-time and real-time ITA languages are related as follows.

Theorem 2.1 *Every pseudo real-time k-ary ITA language is a real-time k^c-ary ITA language, for some $c > 0$.*

Proof outline: Let L be a pseudo real-time k-ary ITA language. By Definition 2.2, there exists a constant c such that $L_{c\#} = \{a_1\#^{c-1}a_2\#^{c-1} \cdots \#^{c-1}a_n\#^{c-1} \mid a_1a_2 \cdots a_n \in L\}$ is accepted by an k-ary ITA A. We denote the levels of A by their distances from the root, i.e., level 0, level 1, ... , level n, ... Now, we divide ITA A into groups of k-level sub-trees such that each cell at level ci, i=0,1,2,..., together with all its descendents on the following $c-1$ levels are in a group. Figure 2.1 shows an example of the division for a binary ITA and $c=2$.

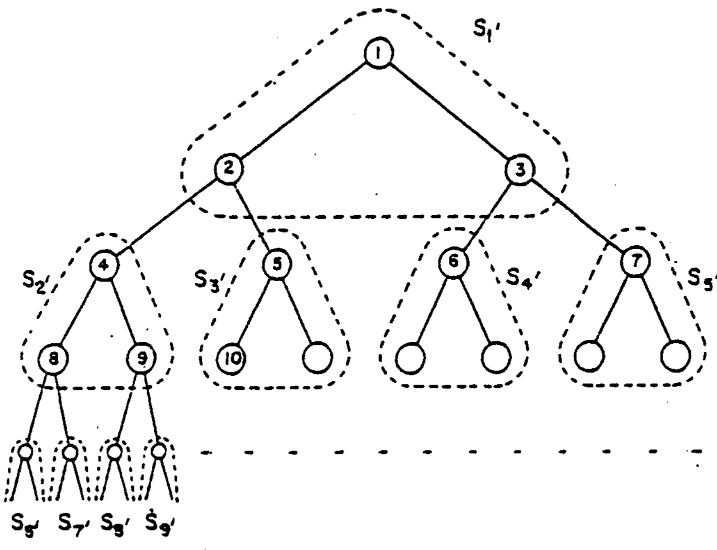


Figure 2.1

If a cell in a group, say F_1, is a child of a cell in another group, say F_2, then F_1 is said to be a child of F_2. We replace each group of A by a cell and keep their parent-child relation, then we get a k^c-ary ITA A'. Now we show that A' can accept L in real time. A' accepts L in the same way as A accepts $L_{c\#}$ except the former is $c-1$ times faster. The principle idea is that every cell of A' simulates c computation steps of a c-level subtree of A in one step. In showing this simulation can be done, we consider the following two cases:

(1) The root cell of A' is able to simulate c steps of the top c-level subtree of A in one step, because A reads $a_i\#^{c-1}$ in c steps while A' reads a_i only in one step without loss of any information; and the children of the root cell of A' have the information of the states of all the cells in the

next c levels of A which is enough for the root cell of A' to simulate c steps of the top c levels of A in one step.

(2) For a cell of A' which is not the root cell, the simulation can be also done because the parent and children of this cell can store enough information to support the simulation of c steps of a c-level subtree of A in one step. Technically, each cell of A' has to store the state information of a $2c-1$ level subtree of A.

For the detailed construction, see the proof of Theorem 6.2 in [4] which is essentially the same result.

The following results have also been shown in [4].

Theorem 2.2 *For any integer $0 < m < n$, the family of real-time m-ary ITA languages is a proper subset of the family of real-time n-ary ITA languages.*

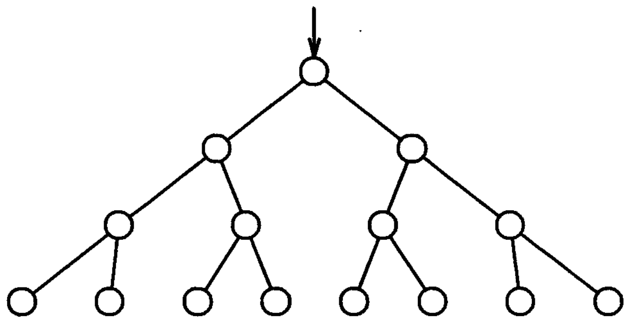
Theorem 2.3 *If L is accepted by an ITA in time $T(n)+c$ where c is a integer constant, then L is accepted by an ITA in time $T(n)$.*

3. ITA's with Global Broadcasting

First, we introduce the concept of ITA's with global broadcasting. Informally, an ITA with global broadcasting is an ITA except that every input symbol comes directly to every cell at the same time. The following is a formal definition which is a bit stronger than the informal definition.

Definition 3.1 *A is called an ITA with global broadcasting (ITAG) if A is formed from an ITA A_0 by adding a direct edge of 0 delay from the root cell to every cell of A.*

Figure 3.1 shows a binary ITAG. Note that the ITAG shown in Figure 3.1 can be simulated by the system shown in Figure 3.2. We actually use the latter presentation of ITAG's in the following proofs.



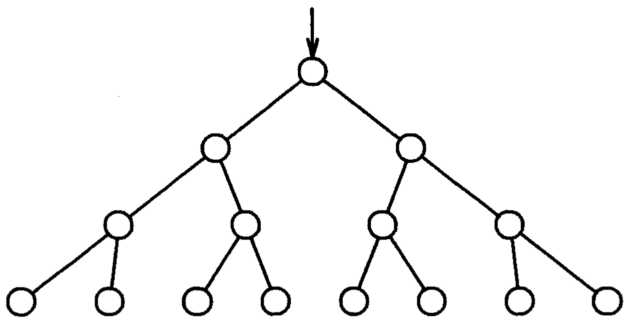
"——" represents "$\underset{1}{\overset{1}{\rightleftarrows}}$"

Figure 3.1

In the following, we will show that for any ITAG, there is an equivalent ITA of higher arity. Thanks to this property, ITAG's become useful tools for designing algorithms for ITA's, especially for real-time and pseudo real-time ITA's. The Retimimg Lemma from [6] is used for the proof of the following theorem.

Theorem 3.2 *For any k-ary ITAG A_G, there exists a k^2-ary ITA A which is equivalent to A_G.*

Proof: Let A_G be a k-ary ITAG. Note that every edge of A_G has one unit delay except the edges for global broadcasting. We prove the theorem in the following steps:

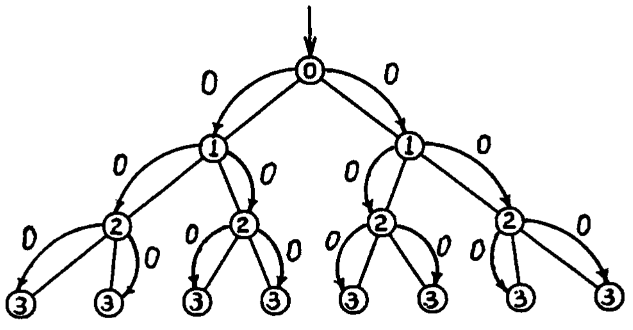


“——” represents “$\overset{1}{\underset{1}{\rightleftarrows}}$”

Figure 3.2

(1) A systolic automaton A_1 is obtained by doubling the delay of every édge in A_G and slowing down both the internal computation speed and input speed such that A_1 runs twice as slow as A.

(2) We define a mapping π which maps every cell of level i in A_1 into integer i as shown for $k = 2$ in Figure 3.3.



"———" represents "$\underset{2}{\overset{2}{\rightleftarrows}}$"

Figure 3.3

For every directed edge, e.g. (c_i, c_j) with delay of d, we calculate the new delay $d' = d + \pi(c_j) - \pi(c_i)$ and obtain a systolic automaton A_2 of the form shown for $k = 2$ in Figure 3.4.

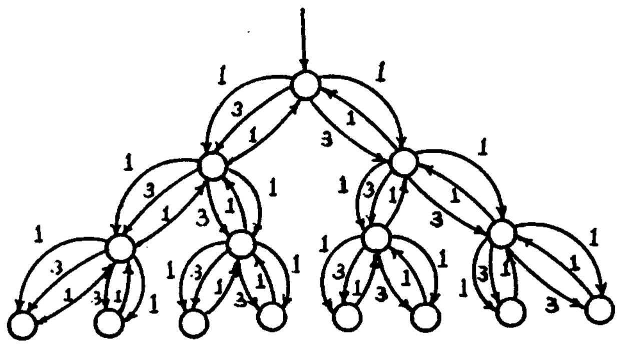


Figure 3.4

By the Retiming Lemma from [6], A_2 is equivalent to A_1, i.e. the root of

A_2 and the root of A_1 enter the same state for the same input string of any length.

(3) A_2 can be further transformed to an equivalent systolic automaton A_3 of the form shown for $k = 2$ in Figure 3.5. Notice that the input speed of A_3 is still slow, i.e. one symbol the every other step.

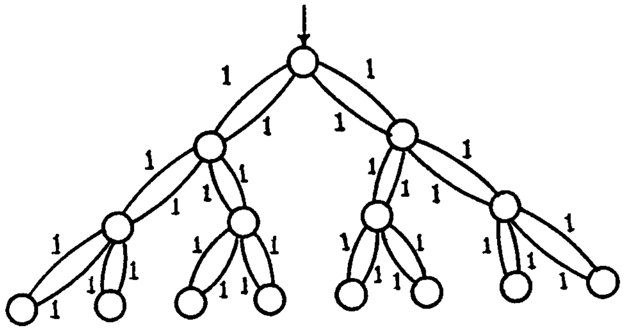


Figure 3.5

(4) By applying Theorem 2.1, we speed up the input speed of A_3 twice to obtain an k^2-ary ITA.

Corollary 3.3 *L is accepted by a k-ary ITAG in real time if and only if it is accepted by a k^2-ary ITA in real time.*

Corollary 3.4 *L is a real-time k-ary ITAG language if and only if it is a pseudo real-time k-ary ITA language.*

Corollary 3.3 is a directly consequence of Theorem 3.2. Corollary 3.4 can be easily proved by Theorem 3.2 and Theorem 2.1.

4. Relations Between Real-Time, Pseudo Real-Time, and Linear-Time ITA languages

Theorem 2.2 shows that for any integer $n > 0$, the family of real-time n-ary ITA languages is a proper subset of the family of real-time $(n+1)$-ary ITA languages. For any integer $m > n > 0$, it is easy to prove that any real-time m-ary ITA language is a linear-time n-ary ITA language. So, for any given k, the family of real-time k-ary ITA languages is properly contained in the family of linear-time k-ary ITA languages. In this section, we will show somewhat surprising result, namely, that for arbitrary ITA's, i.e., ITA's of any arity, the families of real-time, pseudo real-time, and linear-time ITA languages actually coincide. In particular, we show that a linear-time *ITA* can be transformed into a equivalent real-time *ITA* of higher arity.

Theorem 4.1 *L is a linear-time ITA language if and only if it is a pseudo real-time ITA language.*

Proof:

If : Let L be a pseudo real-time k-ary ITA language. Then there exists an integer c such that $L_{c\#} = \{a_1\#^{c-1}a_2\#^{c-1}\cdots\#^{c-1}a_n\#^{c-1} \mid a_1a_2\cdots a_n \in \mathrm{L}\}$ is accepted by a k-ary ITA A in real time. Let A' be a k-ary ITA which works in two phases:

(1) A' reads one input symbol at each step and puts this symbol at the tail of a queue. The queue is implemented by the leftmost (or rightmost) path of A'. See [5] for the details of the implementation of a systolic queue. This process is going on until the whole input is read.

(2) A' simulates A on the input symbols obtained by popping the queue once every c steps.

The above two phases start simultaneously. Phase (1) continues for n steps and then stops. Phase (2) runs for cn steps and gives its answer. So, A' is a linear-time ITA and it accepts L.

Only If : By Theorem 3.2, we know that any ITAG is equivalent to an ITA. It suffices to show that any linear-time ITA language can be accepted by an ITA with global broadcasting in pseudo real-time. Let L be accepted by a k-ary ITA A in cn time, for some integer constant c. We will show that there is an ITA A_G of higher arity with global broadcasting which accepts L in pseudo real time (c #'s after each input symbol). Let l be the cardinality of the input alphabet of A. Consider a systolic automaton A_G which is an l-ary ITAG except that every cell of which is a k-ary ITA rather than a finite automaton. Intuitively, the underlying structure of A_G is an infinite l-ary tree with a infinite k-ary tree growing on each of its nodes. Figure 3.6 shows a case for $l = 2$ and $k = 2$. It is clear that A_G can be simulated directly by a $(k+l)$-ary ITAG in real time. The following is a description of A_G.

(1) Let the input alphabet be $\Sigma = \{b_1, b_2, \cdots, b_l\}$. Every cell, except the root cell, of A_G represents a symbol in Σ. The root cell represents "\$" $\notin \Sigma$ which is the end-marker of any input string. A *i_cell* (an i^{th} cell of another cell) represents "b_i" for any $1 \leq i \leq l$. Then any path of A_G represents a string in $\Sigma^*\$$ and every string in $\Sigma^*\$$ is represented uniquely by a path from a cell to the root of A_G.

(2) When the first input symbol is read, the root cell broadcasts a "start" command to every cell. Immediately every cell of A_G starts to simulate A on the input string presented by the path from itself to the root. Since A is a cn-time ITA, every cell at level i (the root is at level 0) will take $c\,i$ steps to complete this simulation.

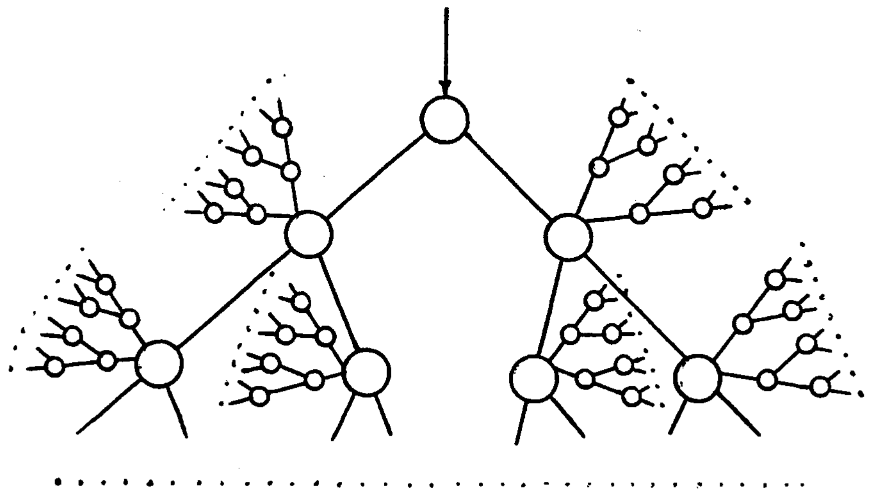


Figure 3.6

(3) A_G reads an input symbol every $c+1$ steps. In other words, A_G reads a input string $a_1\#^c a_2\#^c \cdots \#^c a_n\#^c$. The root cell broadcasts the input symbols in Σ to every cell of A_G. When a cell receives a broadcasted symbol, it compares this symbol with the value represented by itself and gets a truth value T if they are equal or F if they are not equal. For every cell, there is a truth-value queue to store these truth values. Since every cell of A_G is an ITA, the truth-value queue can be simply implemented by a path of this ITA. Note that (2) and (3) start at the same time and continue in parallel.

(4) Whenever a cell completes (2), it checks the top symbol of its truth-value queue and pop it. If the value is T, the signal, *accept* or *reject* according the result of the simulation from (2), will be sent up to its parent. If the value is F, a signal *nonresult* will be sent up. A parent will handle these signals from its children according the following rules: a) If all the signals from its children are *nonresult*, a signal *nonresult* will be

send up no matter what the top-of-queue symbol is; and the top-of-queue symbol is popped. b) If one of the signals from its children is *accept* or *reject*, it checks the top symbol of its truth-value queue. If it finds a T, it sends up a signal *accept* or *reject*. Otherwise, it will send the signal *nonresult*. It pops the truth-value queue. Note that at most one signal from its children has the value *accept* or *reject* because at any time exactly one of top-of-queue symbols of all its children has value T.

We claim that the above technique gives an correct answer, *accept* or *reject*, to every prefix of the input string.

In order to show that the result is correct, we have to prove that the path where the result has been passed through represents the same string as the input string broadcasted by the root. Consider the fact that every cell at level i will complete its simulation of A earlier than any cell at level $i+1$ because the cells at level i work on shorter input strings. If a result obtained at a cell of level n is coming up along the path to the root and all the top of queue symbols it encountered are T's, then it means that the i^{th} cell it encountered represents the i^{th} symbol in the input string, for all $i=1,2,...,n$, namely, this path represents the input string. Thus the result computed by this cell and sent up to the root is the result for the given input string. It is easy to prove that there is only one such path for any input string.

As immediate corollaries of Theorem 4.1 and Theorem 2.1, we have:

Corollary 4.2 *L is a linear-time ITA language if and only if it is a real-time ITA language.*

Corollary 4.3 *The families of real-time, pseudo real-time, and linear-time ITA languages coincide, if all the ITA's of positive integer arity are considered.*

5. ITA as a real-time infinite memory

as an example, we show an ITA implementation of a real-time infinite memory. We assume that memory locations are addressed by binary numbers and each address is read sequentially bit by bit. We also assume that at each location a finite amount of information can be stored. Formally, every store command is in the form xI, where $x \in \{0, 1\}^+$ is an address and $I \in \Delta$ is the information to be stored. Every retrieval command is in the form $x\#$, where $x \in \{0, 1\}^+$ is an address and # is a end-marker. An input string is an arbitrary stream of "store" and "retrieve" commands. If an $x\#$ command precedes any xI command, then a special "blank symbol" is retrieved. We show that such information storage and retrieval can be done on a binary ITAG in real time. Consequently, this can also be achieved on a 4-ary ITA by Theorem 3.2. The results of this section can be easily extended to the case when the addresses are not restricted to binary numbers; they are from an arbitrarily given base.

Claim 5.1 *The operation of storing can be done in real time on an ITAG.*

Proof: Let xI be an arbitrary store command, where $x \in \{0, 1\}^+$ is the address and I is the symbol to be stored. It is assumed that every cell of the ITAG represents a value. The root represents a '\$'. Each left cell, i.e. a left child of some cell, represents a '0' and each right cell represents a '1'. There are two modes of operations on the ITAG for an insertion:

(1) When the first bit of x is broadcasted, every cell initiates a signal whose value is the same as the one this cell represents. These signals start to flow down by one level at each step. For example, the signal '\$' is initiated at the root at first step; it goes to every cell on *level* 1 at second step, and reaches every cell on *level* n at n^{th} step. These signals all die out after the symbol I is broadcasted.

(2) During the address x is being broadcasted bit by bit, every cell, at each step, compares the value received from the broadcasting with the signal, as described in (1), which is passing through.

The cell which satisfies the following two conditions is the cell to store the information I: *a*) all the pairs of values it has compared are matching; *b*) the signal '$' is just arriving when I is being broadcasted.

It is easy to see that for any address x, one and only one cell will be chosen by the above algorithm to store the given symbol, and the information storage is performed in real time.

Claim 5.2 *The operation of a information retrieval on a binary ITAG can be done in real time.*

Proof: As in the proof of Claim 5.1, every cell of the ITAG represents a value. When the operation is initiated, i.e. when the first bit of the address is broadcasted, all the stored information symbols are motivated and started to climb up towards the root. Any information symbol will be killed during the climbing if it arrives at a cell at the time when the value represented by this cell is not equal to the broadcasting value of the current step. We can see that one and only one information symbol will reach at the root, for any given address, at the time the last bit of the key is just read. Actually, there is a stored symbol (or empty) reaching to the root for any prefix of a given address.

Based on the observation that the symbol retrieved by the algorithm of Claim 5.2 must be the symbol stored by the algorithm of Claim 5.1, we have the following theorem.

Theorem 5.1 *The operations of information storage and retrieval can be accomplished in real time on an ITA.*

By the result in Theorem 3.2, we have the following.

Corollary 5.2 *The operations of information storage and retrieval can be accomplished on an ITA in real time.*

References

[1] Choffrut, C., and Culik II, K., "On Real-Time Cellular Automata and Trellis Automata", *Acta Informatica* 21, pp.393-407, 1984.

[2] Cole, S.N., "Real-Time Computation by n-Dimensional Iterative Arrays of Finite-State Machines", *IEEE Trans. on Comp.,* vol. c-18, no.4, pp.349-365, 1969.

[3] Culik II, K., Ibarra, O.H., and Yu, S., "Iterative Tree Arrays with Logarithmic Depth", *Research Report* CS-85-03, Dept. of Computer Science, Univ. of Waterloo, 1985.

[4] Culik II, K. and Yu, S., Iterative Tree Automata, *Theoretical Computer Science, 32 (1984) 227-247.*

[5] Guibas, L.J. and Liang, F.M., Systolic Stacks, Queues, and Counters, *Proceedings of 1982 Conference on Advanced Research in VLSI,* M.I.T., 155-164.

[6] Leiserson, C.E. and Saxe, J.B., Optimizing Synchronous Systems, *Proc. of 22nd Ann. FOCS Symposium,* (1981), 23-36.

[7] Smith III, A. r., "Real-Time Languages Recognition by One-Dimensional Cellular Automata", *J. of Computer and System Science,* 6, pp.233-253, 1972.