Systolic Automata for VLSI On Balanced Trees*

K. Culik II, J. Gruska+, A. Salomaa†

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

Research Report CS-82-01

January 1982

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

+ On a leave of absence from the Computer Research Center in Bratislava, Czechoslovakia

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

A B S T R A C T

Systolic tree automata with a binary (or, more generally, balanced) underlying tree are investigated. The main emphasis is on input conditions, decidability, and characterization of acceptable languages.

1. Introduction

Systolic tree automata were introduced in [1], basically as a model for VLSI. (An essentially different model, a systolic trellis automaton, was introduced in [2].) It was observed already in [1] that the new model gives rise to a number of new problems and problem areas which are interesting also from the point of view of classical language theory. This aspect was further exploited in [3].

The present paper deals with systolic tree automata, where the underlying tree is the complete binary tree. The definitions and results are stated for this binary case only. However, this is done only because of notational simplicity. The reader should have no difficulties in verifying the results for arbitrary balanced trees. (We call a tree balanced if there is an integer $k \geq 2$ such that every node has exactly k sons.) Even the proofs remain almost the same in this more general case. Some of our results (such as the stability theorem in Section 3) can be easily extended to concern a slightly more general class of trees (for instance leftmost prefix preserving trees) than balanced trees. However, they can not be extended (at least not by the methods of this paper) to concern arbitrary systolic tree automata of [1]. Some of the difficulties in generalizing the decidability of the emptiness problem (established in Section 6 below) are discussed in [3].

We shall investigate in this paper structural results concerning systolic binary tree automata, as well as properties of the accepted languages. Sections 3 and 4 deal with structural results as regards the input format. We prove first the „stability" result: there is no loss

of generality in assuming that the input is fed on an arbitrary level (rather than on the first possible level as done according to the original definition). We then show that this result can not be extended to a „super-stability" result, where we could also scatter the input in an arbitrary fashion (preserving the order of the letters) on the level in question.

The effect of nondeterminism is discussed in Section 5. It is shown that nondeterminism does not buy you anything. Although this result in itself is not surprising, it is a useful tool in several constructions, for instance, in [3]. Section 6 deals with decidability: it is shown that the basic problems dealing with systolic binary tree automata and the accepted languages are decidable. Finally, Section 7 discusses some open problems concerning especially the characterization of the accepted languages. It seems very likely that the family of languages acceptable by systolic binary tree automata has the following properties, which make the family rather unique among the language families investigated so far: the family contains all regular languages and some languages „high up" in the customary hierarchies but no nonregular context-free languages!

This paper is largely self-contained: only the very basics of formal language theory is assumed on the part of the reader. The reader is referred to [1] for further motivation and background material concerning systolic tree automata. References [1] and [2] contain, moreover, some further discussion concerning the different models for VLSI (starting with arbitrary graphs), as well as the notion of "systolic" (the information is pumped in to form a uni-directional flow).

2. Basic Definitions and Preliminary Results

We begin with an informal description of a systolic binary tree automaton, shortly, BT-VLSI. Consider an infinite binary tree without leaves. We may define the „levels" of the tree in the natural way: each level consists of all nodes whose distance from the root is a fixed number. Consider now an input word w over Σ with length t. We choose the first level in the tree with $n \geq t$ vertices. The word $w\#^{n-t}$ (where $\#$ is a special symbol outside Σ) is now „fed", letter by letter to the level in question. This means the following. We consider also another alphabet Σ_0 (referred to as the operating alphabet) and a function $g : \Sigma \cup \{\#\} \to \Sigma_0$. The nodes of the level in question are labeled (in the correct order!) with the g-values of the letters in the word $w\#^{n-t}$.

Information now flows bottom-up and in parallel. We consider also another function $h : \Sigma_0^2 \to \Sigma_0$. If the sons of a node have already been assigned (from left to right) the values a and b, their father is assigned the value $h(a,b)$. The word w is accepted by our BT-VLSI if the root of the tree gets in this way a value from Σ_0', where Σ_0' is a designated subset of Σ_0.

Formally, a BT-VLSI is a sixtuple

$$G = (\Sigma, \Sigma_0, \Sigma_0', \#, h, g) ,$$

where the Σ's are alphabets such that $\Sigma_0' \subseteq \Sigma_0$, $\#$ is a letter not in Σ, and h and g are mappings of Σ_0^2 and $\Sigma \cup \{\#\}$, respectively, into Σ_0.

The domain of h is extended to the set

$$\bigcup_{n \geq 1} \Sigma_0^{2^n} = A$$

as follows. Consider a word x in A. Hence, for some $n \geq 1$, the length of x satisfies $|x| = 2^n$. We write $x = y_1 y_2 \ldots y_{2^n-1}$ where each y is a word of length 2. We now apply h to each of the words y, yielding a new word

$$x_1 = h(y_1)\, h(y_2) \ldots h(y_{2^n-1}) .$$

If the length of x_1 exceeds 1, i.e., if $n \geq 2$, we repeat the same procedure. Continuing in this way, we finally obtain a word z of length 1, i.e., a letter of Σ_0. We write $H(x) = z$. Clearly, H is an extension of h, i.e., $H(x) = h(x)$ for words x of length 2.

The mapping g can be viewed in the natural way as a homomorphism of $(\Sigma \cup \{\#\})^*$ into Σ_0^*. A word w over Σ of length ≥ 2 is <u>accepted</u> by our BT-VLSI G if $H(g(w\,\#^i)$ is in Σ_0' where $i \geq 0$ is the unique integer such that

$$|w\#^i| = 2^n \text{ for some } n \geq 1, \text{ and } w > i .$$

A word w of length 1 (resp. the empty word ε) is accepted if $g(w)$ (resp. $g(\#)$) is in Σ_0'.

The collection of all words accepted by G is denoted by $L(G)$ and referred to as the language accepted by G. A language L is BT-VLSI acceptable if $L = L(G)$, for some BT-VLSI G. The family of all BT-VLSI acceptable languages is denoted by $\mathcal{L}$(BT-VLSI).

Some elaborations of the notions introduced above will be considered in the sequel. Instead of an unlabeled binary tree, we could begin with a labeled one. Then the two functions h and g are replaced by the collection of functions h_a and g_a, where a ranges through the alphabet of labels. The definitions remain the same except that now h_a and g_a are used for nodes labeled with a. This generalization was considered in [1], where it was also assumed that the labeling satisfies the following regularity condition: the infinite labeled tree possesses only finitely many infinite labeled subtrees. It was shown in [1] that every language acceptable by some BT-VLSI with an underlying regularly labeled tree belongs to the family $\mathcal{L}$(BT-VLSI) as defined above. Because of this „normal form" result we have given above the simpler definition. It is to be emphasized, however, that in this result the regularity condition for the labeling is quite essential: if unrestricted labeling is allowed then, for instance, every language over a one-letter alphabet becomes BT-VLSI acceptable, although only finitely many labels are used.

At a first glance, it might seem unnatural that the input has to be fed to a specific level of the tree. (From the VLSI point of view, this means that there is also a lower bound on the size of words that can be processed by a chip of a certain size - an upper bound being necessary in any case.) Hence, it would be desirable that an arbitrary BT-VLSI can always be replaced by an equivalent (i.e., accepting the same language) BT-VLSI which is stable in the following sense: the accepted language remains invariant if we drop the condition that a word must always be fed on the first possible level of the tree. (In the formal definition above, this condition amounts to the inequality $|w| > i$.) Indeed, we shall establish this result in the next section. Here we give now the formal definitions.

A BT-VLSI (where the above notation is used for the different items) G is termed stable if, for all words w over Σ and all integers i and j such that both $|w\#^i|$ and $|w\#^j|$ are powers of 2, $H(g(w\#^i))$ is in Σ_0' if and only if $H(g(w\#^j))$ is in Σ_0' .

For a word w over Σ , all words w' over $\Sigma \cup \{\#\}$ satisfying conditions (i) and (ii) below are referred to as #-versions of w .

(i) The length of w' equals a power of 2.

(ii) w' is obtained by shuffling (in the usual sense of the shuffle operation) w and some word in $\#^*$.

(Thus, if we erase from w' all occurrences of $\#$, leaving all letters of Σ unchanged, we obtain w .)

A BT-VLSI G is termed super-stable if, for all #-versions w' and w" of the same word w , the letter H(g(w')) is in Σ_0' if and only if H(g(w")) is in Σ_0' .

Thus, G being stable means that the acceptance of a word w does not depend on the level w is fed in. G being super-stable means that, in addition, the acceptance of w is independent also of the choice of the nodes in the level chosen as the input level for w . We shall prove in the next two sections that, for every G , an equivalent stable BT-VLSI can be constructed but that there are BT-VLSI's for which no equivalent super-stable BT-VLSI exists.

This section is concluded with a result from [1] needed in the sequel.

Theorem 1. The family $\mathcal{L}$(BT-VLSI) contains all regular languages and is (effectively) closed under Boolean operations.

3. Stability

The purpose of this section is to establish the result concerning stability, mentioned already in Section 2. The result is especially pleasing because it shows that the somewhat unnatural input condition of the original definition can always be avoided.

Theorem 2. For every BT-VLSI G, an equivalent stable BT-VLSI G' can be effectively constructed.

Proof. Consider a BT-VLSI

$$G = (\Sigma, \Sigma_0, \Sigma_0', \#, h, g) ,$$

where the different items are as in Section 2. We now define a BT-VLSI G' as follows. It will be convenient to use a labeled tree and apply the normal form result of [1] (referred to in Section 2), according to which such a labeling causes no loss of generality. Clearly, stability is not affected by this construction.

The tree of G' is labeled as follows. The root gets the label R, all other nodes in the leftmost path get the label A, and all remaining nodes the label B. Clearly, the regularity condition for the labeling is satisfied: the number of distinct infinite labeled subtrees equals three.

The terminal alphabet Σ, as well as the special symbol $\#$, remain unaltered for G'. The operating alphabet of G' consists of the letters a of Σ_0', as well as of their „primed versions" a', and of two additional letters s and r. The letter s is the only designated (accepting) letter of G'. The functions g_R, g_A, g_B, h_R, h_A, h_B, corresponding to our labeling, are defined as follows.

Consider first the g-functions. The value $g_R(b)$ equals s or r, for b being a letter of Σ (resp. #), depending on whether or not b (resp. the empty word) is in L(G). The value $g_B(b)$ equals g(b) (resp. (g(b))'), for b being a letter of Σ (resp. #). The value $g_A(\#)$ equals s or r, depending whether or not the empty word is in L(G). For letters b of Σ, $g_A(b)$ equals (g(b))' or g(b), depending on whether or not b is in L(G).

Consider then the h-functions. For all pairs unlisted in the sequel, the value of the function equals r.

The value $h_R(x,y)$ equals s in the following cases. (i) h(x,y) is defined and is in Σ_0'. (ii) x = s and y is a primed letter. (iii) y is a primed letter, y = z', h(x,z) is defined and is in Σ_0'. (iv) x is in Σ_0' and y results from a tree with all leaves labeled by #.

The value $h_B(x,y)$ (resp. $h_B(x,y')$ and $h_B(x',y')$) equals h(x,y) (resp. (h(x,y))'), whenever h(x,y) is defined. The value $h_A(x,y)$ equals h(x,y), whenever the latter is defined. The value $h_A(x,y')$ equals s if h(x,y) is in Σ_0' and y' results from a tree where all leaves are labeled with #. (Clearly, we can check the latter condition. According to our convention, $h_A(x,y')$ equals r in the remaining cases.) The value $h_A(s,y')$ equals s, for all primed letters y'. The value $h_A(x',y')$ equals s, for all x' and y'. Finally, $h_A(x',y) = h(x,y)$ whenever the latter value is defined.

It is now fairly straightforward to verify that G' is stable and equivalent to G. Indeed, our definitions above guarantee that, if the empty word or a word of length one is in $L(G)$, then it is also in $L(G')$ and, moreover, can be accepted from an arbitrary level, and vice versa.

Consider then a word w over Σ with length ≥ 2. We claim that w is in $L(G)$ if and only if w is in $L(G')$ and, moreover, that whenever w is in $L(G')$, it can be accepted from an arbitrary (sufficiently long) level.

Assume first that w is in $L(G)$. Consider an arbitrary sufficiently long level in the tree for G'. Feed w to this level. Several cases arise, depending on the length of w as compared with the length of the level. (i) The length of w exceeds half of the length of the level. Then w is accepted, by points (i) or (iii) in the definition of h_R.

(ii) The length of w equals half of the length of the level. w is now accepted, by point (iv) of the definition of h_R, as well as by the fact that $h_A(x,y) = h(x,y)$ in appropriate cases.

(iii) The length of w is less than half of the length of the level. If the length of w is not a power of 2, w is accepted, because of point (ii) in the definition of h_R, and because of the definition of h_A. (Recall that w is in $L(G)$.) If the length of w equals a power of 2, w is still accepted, because of point (ii) in the definition of h_R, as well as the definition of h_A.

Assume, secondly, that w is not in $L(G)$. An analysis similar to the one outlined above shows that w is not in $L(G')$ (no matter on what level w is fed in). This completes the proof of Theorem 2. □

4. Super-stability

Although the result of the last section shows that the input can be fed on an arbitrary level, we shall now prove that one cannot go further: super-stability cannot be achieved.

Theorem 3. Every language accepted by a super-stable BT-VLSI is regular. Consequently, there are BT-VLSI's for which no equivalent super-stable BT-VLSI can be constructed.

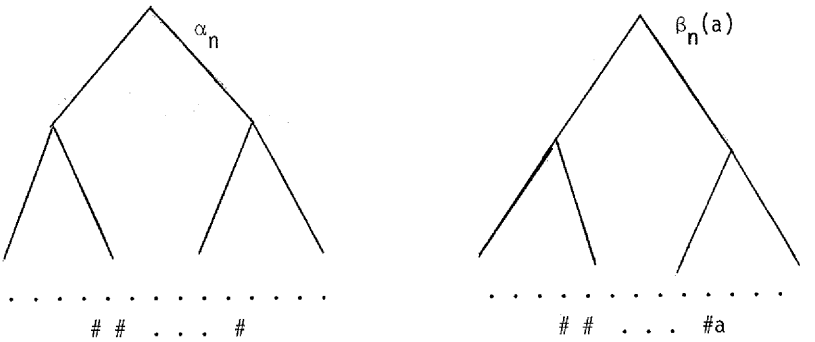
Proof. Clearly, the second sentence is a consequence of the first sentence. This follows because, for instance, the language $\{a^{2^n} \mid n \geq 0\}$ is obviously in $\mathcal{L}$(BT-VLSI).

To prove the first sentence, assume that G is a super-stable BT-VLSI. (Our notation for the different items of G is the same as before.) We shall prove that $L(G)$ is regular, by constructing a deterministic finite automaton M accepting $L(G)$.

First some initial observations. Consider the letters

$$\alpha_n = H(g(\#^{2^n})) \quad \text{and} \quad \beta_n(a) = H(g(\#^{2^n-1}\, a)) ,$$

belonging to the operating alphabet of G, where a is an arbitrary letter in the terminal alphabet of G, and $n \geq 1$ is an arbitrary integer. Thus, α_n and $\beta_n(a)$ are the outputs associated with the root of the following trees:



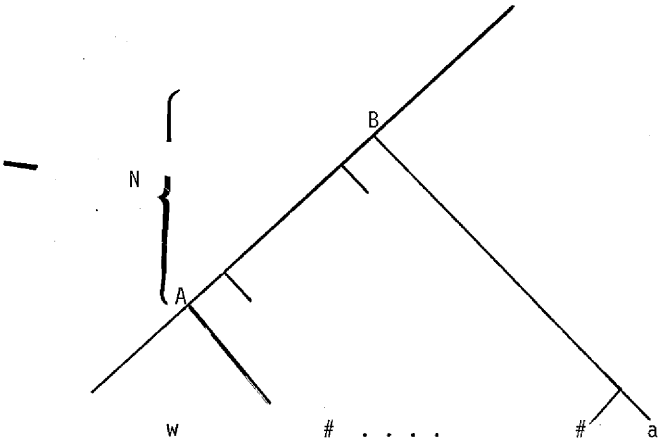
It is obvious that α_n , as well as each $\beta_n(a)$, is an ultimately periodic function of n . Let N be an integer divisible by all the periods (i.e., the period of α_n and $\beta_n(a)$, for every a in the terminal alphabet) and also greater than all the thresholds („initial mess").

The states of our automaton M are the letters of the operating alphabet of G . The input alphabet of M equals the terminal alphabet of G . The initial state is $H(g(\#^{2^n}))$.

The transition function δ is defined as follows. Let A be a state (i.e., a letter of the operating alphabet of G) and let a belong to the terminal alphabet of G . Choose any word w with $|w| = 2^{kN}$, for some k , such that $H(g(w)) = A$. Define

$$\delta(A, a) = H(g(w\ \#^i\ a))$$

where $i = |w| \cdot 2^N - |w| - 1$. (If no word w satisfying our conditions exists, the definition of $\delta(A,a)$ is arbitrary because A will not be a reachable state.) Our choice of N guarantees that $\delta(A,a)$ is independent of w . Intuitively, $\delta(A,a)$ indicates how the output A changes in the leftmost path in N steps when the word fed to the additional leaves is $\#^i a$:



Thus, in our picture $\delta(A,a) = B$.

To complete the definition of our finite automaton M, we now specify the final states. Consider an arbitrary state A and choose again a word w with $H(g(w)) = A$ and $|w| = 2^{kN}$, for some k. By definition, A is final if and only if

$$H(g(w\,\#^{i})) \quad , \qquad \text{where} \qquad i = |w| \cdot 2^{N} - |w| \quad .$$

belongs to the set Σ_0' of designated letters of G. By our choice of N, this definition is again independent of w.

It is now easy to verify that $L(M) = L(G)$. Indeed, the inclusion $L(M) \subseteq L(G)$ is obvious. To establish the reverse inclusion, consider an arbitrary word $w = a_1 \ldots a_t$ (where the a's are letters) in $L(G)$. Since G is super-stable, we may input (and accept!) w in the form

$$\#^{2^N}\,\#^{i_0}\,a_1\,\#^{i_1} \ldots a_t\,\#^{i_t} \quad ,$$

where the i's are chosen in such a way that, in every step between the terminals the height of the tree grows by N. This implies that w causes in M a transition from the initial state to one of the final states, showing that w is in $L(M)$. □

Theorems 2 and 3 give a pretty good idea how much leeway we have in the input format. It seems likely that Theorem 3 can be extended to concern „weak" super-stability, meaning that every input w can be fed on an arbitrary level and to an arbitrary position in such a way that letters of w remain adjacent, i.e., only shuffles of the form $\#^i w \#^j$ are allowed.

5. Nondeterminism

The purpose of this section is to discuss the familiar generalization to nondeterminism for BT-VLSI's. We skip the detailed definitions and proofs because it is straightforward to fill them in. Although the idea of nondeterminism is very simple, it is very useful in several constructions, as exemplified in [3].

A nondeterministic BT-VLSI differs from the one defined in Section 2 in that the range of the functions g and h is the set of subsets of the operating alphabet. Otherwise, everything remains the same but $H(g(w\,\#^i))$ may consist of several letters of the operating alphabet. The word w is accepted if at least one designated letter is among the letters of $H(g(w\,\#^i))$. Thus, as is customary in connection with nondeterministic devices, the same input may give rise to several computations. All failures are disregarded if there is one successful computation.

Theorem 4. A language is acceptable by a nondeterministic BT-VLSI if and only if it is acceptable by a deterministic BT-VLSI. Consequently, every language acceptable by a nondeterministic BT-VLSI is accepted by a (deterministic) stable BT-VLSI.

Proof. The second sentence is a consequence of the first sentence, by Theorem 2. The „if" - part of the first sentence is trivial because deterministic BT-VLSI's are a special case of nondeterministic ones. The „only if" - part of the first sentence is established by the familiar subset construction: for a nondeterministic BT-VLSI G, an

equivalent deterministic one, G', is constructed by letting the operating alphabet of G' consist of the subsets of the operating alphabet of G and by defining the new functions h' and g' in the appropriate manner to take care of all transitions of G. □

6. Decidability

By Theorem 1 and by the fact (see, for instance, [1]) that L(BT-VLSI) contains rather complicated exponential languages, we can conclude that L(BT-VLSI) is a rather „large" family of languages. On the other hand, we shall prove in this section that this family has all the reasonable decidability properties and is, consequently, a relatively „small" family.

Theorem 5. It is decidable whether or not the language accepted by a given BT-VLSI is empty.

Proof. Consider an arbitrary BT-VLSI

$$G = (\Sigma, \Sigma_0, \Sigma_0', \#, h, g) .$$

We have to decide whether or not $L(G)$ is empty. By Theorem 2, we may assume that G is stable.

We define subsets A_i , B_i , C_i , $i = 0, 1, 2, \ldots,$ of the set Σ_0 as follows. By definition,

$$A_0 = \{g(a) \mid a \text{ in } \Sigma\}, \quad B_0 = \phi , \quad C_0 = \{g(\#)\} .$$

For any $i \geq 0$, a letter x is in A_{i+1} if and only if $x = h(y,z)$ where both y and z are in A_i . For any $i \geq 0$, a letter x is in C_{i+1} if and only if $x = h(y,z)$ where both y and z are in C_i . Finally, for any $i \geq 0$, a letter x is in B_{i+1} if and only if $x = h(y,z)$ where one of the following three alternatives (i) - (iii) is satisfied:

(i) y is in A_i and z is in B_i, (ii) y is in A_i and z is in C_i, (iii) y is in B_i and z is in C_i.

We can immediately check whether or not the empty word is in $L(G)$. Hence, we may assume this is not the case. Then $L(G)$ is nonempty if and only if some letter of Σ_0' is in $A_i \cup B_i$, for some i. (Recall that G is stable.) On the other hand, for all i, the triple $(A_{i+1}, B_{i+1}, C_{i+1})$ is completely determined by the triple (A_i, B_i, C_i). Since all of the sets involved are subsets of Σ_0, this implies that repetitions must occur, and we obtain the decidability. □

Theorem 1 now immediately yields the following corollary of Theorem 5.

Theorem 6. The equivalence problem for BT-VLSI's is decidable.

We give, finally, the following decidability result of a somewhat different kind. The corresponding problem for trellis automata is undecidable.

It is interesting to observe that most of the important decision problems for BT-VLSI's are, in fact, decidable, whereas undecidability holds for corresponding problems (for instance, emptiness) concerning trellis automata.

Theorem 7. It is decidable whether or not a given BT-VLSI is stable.

Proof. Given an arbitrary BT-VLSI G , we first construct, by Theorem 2, an equivalent stable BT-VLSI G' . We then modify G and G' in such a way that the original special symbol # will belong to the terminal alphabet and that all words whose length is not a power of 2, as well as all words where the special symbol # occurs somewhere else than at the end of the word, are rejected. These modifications are easy to perform. Call by K and K' the modified BT-VLSI's.

By Theorem 6, we can decide whether or not K and K' are equivalent. But clearly G is stable if and only if K and K' are equivalent. □

7. Open Problems

Although a number of results about the basic properties of BT-VLSI's, as well as the accepted languages, have been established, some important open problems still remain. In our estimation, the most important ones deal with the characterization of the family L(BT-VLSI). For instance, it seems most likely that this family contains no non-regular context-free languages (which, in view of Theorem 1, would make this family quite unique in language theory). However, we have not been able to establish this conjecture. The reference [1] contains a criterion that could perhaps be used. However, the corresponding very natural-looking result concerning context-free languages fails as shown in [4].

It seems very likely that an exact characterization of languages over a one-letter alphabet in L(BT-VLSI) can be obtained. Such a characterization would involve the Boolean closure of regular languages and some suitably chosen exponential languages.

In addition, some important decision problems, such as the decidability of the super-stability of a BT-VLSI or of the regularity of a language in L(BT-VLSI), remain open.

References

[1] K. Culik II, A. Salomaa and D. Wood, VLSI systolic trees as acceptors. Res. Rept. CS-81-32, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario (1981).

[2] K. Culik II, J. Gruska and A. Salomaa, Systolic trellis automata (for VLSI). Ibid., CS-81-34.

[3] K. Culik II, J. Gruska and A. Salomaa, On a family of L languages resulting from systolic tree automata. Ibid., CS-81-36.

[4] K. Culik II, J. Gruska and A. Salomaa, On nonregular context-free languages and pumping. EATCS Bulletin, to appear.