<table>
<thead>
<tr>
<th>Title</th>
<th>A Rich Hierarchy on the Time Complexity of Uniform PRAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Iwama, Kazuo</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 1988, 666: 101-104</td>
</tr>
<tr>
<td>Issue Date</td>
<td>1988-07</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/100681">http://hdl.handle.net/2433/100681</a></td>
</tr>
<tr>
<td>Type</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td>Textversion</td>
<td>publisher</td>
</tr>
</tbody>
</table>

Kyoto University
A Rich Hierarchy on the Time Complexity of Uniform PRAMs

Kazuo Iwama

Computer Sciences Institute
Kyoto Sangyo University

1. Introduction.

Motivation of introducing the unbounded fan-in circuit model is not unique: In [1] the authors found a connection between the complexity theory over this model and a famous open question (at that time) on the polynomial hierarchy. [4] proves that the model can simulate the parallel random access machines with simultaneous writes (CRCW-PRAMs) of almost the same depth and size. Recently the present author claims in [2] that the model can be much more useful than the bounded fan-in model to study the depth hierarchy of Boolean functions. To this goal, however, it turns out that the essential nonuniformity circuit models generally possess is a major obstacle and that the best possible model we have currently seems to be (uniform) CRCW-PRAMs. The main result of [2] is the following.

Let $A(i,j)$ be the Ackermann function and let $\bar{A}_{k}(n)$ be its inverse function defined by $\bar{A}_{k}(n) = \text{least } j \text{ such that } A(k,j) \geq n$. CRCW-PRAMs here denote parallel RAMs with simultaneous writes and with operations $+$, $-$ and $1$ (bitwise OR). It should be noted again that CRCW-PRAMs in this paper are uniform, namely, each RAM has the same program not depending on the size of inputs. Then it follows:

**Theorem 1.** For any constant $c \geq 4$, there is a nondegenerate Boolean function $G_{c}$ of $n$ variables such that it takes $\Theta(\bar{A}_{c}(n))$ steps to compute $G_{c}$ by CRCW-PRAMs with polynomial number of processors.

In this paper we extend this result, which suggests the existence of a rich hierarchy on the time complexity of Boolean functions. We will also show a result on a relation between the star-free regular expressions and the unbounded fan-in circuits which might help removing the obstacle above mentioned (the nonuniformity of the circuit models).

2. Time Hierarchy.

It should be noted that Theorem 1 is still true if $\bar{A}_{c}(n)$ is replaced by its composition like $\bar{A}_{c_{1}}(\bar{A}_{c_{2}}(\cdots (\bar{A}_{c_{i}}(n)) \cdots ))$ for constants $c_{1}$, $c_{2}$, $\cdots$, $c_{i} \geq 4$ (e.g., $\bar{A}_{4}(\bar{A}_{4}(\bar{A}_{4}(n))) = \log^{*} \log^{*} \log^{*} n$). In this section we demonstrate that the hierarchy is much more dense. Let $M$ be a Turing machine computing an integer function $f(n)$ by producing $1^{f(n)}$ on its output tape from $1^{n}$ on its input tape. Then $f$ is called a
polynomial-time function if $M$ halts within $T(n)$ steps for some polynomial $T$. For simplicity, we assume that $f$ is monotone, i.e., $f(n+1) \geq f(n)$ for all $n$.

**Theorem 2.** Theorem 1 is also true if $A_c(n)$ is replaced by any polynomial-time function $f$.

**Proof.** A straightforward modification of the proof of Theorem 1 [2]. Use a sequence of configurations of the Turing machine that computes the polynomial-time function $f$ instead of the sequence of binary numbers. Details are omitted. □

**Corollary 1.** Theorem 1 is still true for $A_n(n)$.

**Proof.** We will show that Ackermann function $A(i,j)$ can be computed in a polynomial number of steps on its answer (not on its input). Then it immediately follows that $A_n(n)$ is a polynomial-time function.

By definition $A(i,j)$ can be obtained by applying the following operations as far as possible:

\[ A(0, y) = y + 1 \]  \hspace{1cm} (1)
\[ A(x+1, 0) = A(x, 1) \]  \hspace{1cm} (2)
\[ A(x+1, y+1) = A(x, A(x+1, y)) \]  \hspace{1cm} (3)

Thus, during the computation we always handle the form like

\[ A(a_0, A(a_1, \cdots, A(a_{n-3}, A(a_{n-2}, a_{n-1})) \cdots), \]

which we denote by string

\[ \sigma = a_0a_1 \cdots a_{n-1}. \]

Let $L(\sigma)$ be the length of $\sigma (=n)$ and $S(\sigma)$ be the sum of the integers ($=a_0 + \cdots + a_{n-1}$). Then one can see that:

(i) $L(\sigma)+S(\sigma)$ does not change before and after applying operation (1) or (2).

(ii) Applying operation (3) increases $L(\sigma)+S(\sigma)$ by $x$. Namely if $x=0$ then $L(\sigma)+S(\sigma)$ does not change either.

Since the final answer clearly does not exceed $L(\sigma)+S(\sigma)$, the number of times operation (3) for positive $x$ (that makes $L(\sigma)+S(\sigma)$ larger at least one) is applied must be less than the value of the answer. Therefore the total number of necessary applications of operations of (1)-(3) depends on the number of applications of those that do not change $L(\sigma)+S(\sigma)$ or operations (1), (2) and (3) for $x=0$. It should be observed that the application of the operations is deterministic, i.e., the next operation is determined uniquely by the current $\sigma$. For example, when $\sigma = a_0 \cdots a_{n-2}a_{n-1}$, operation (3) is applied if and only if both $a_{n-2}$ and $a_{n-1}$ are positive. Now one can see that a lot of consecutive applications of operations (1), (2) and (3) for $x=0$ occur when $\sigma$ looks like

\[ \sigma = \sigma_1a_{11} \cdots 1b \quad (m \text{ 1's, } a \geq 2 \text{ and } b \geq 1). \]

For $\sigma$ of this form, operation (3) has to be applied $b$ times, operation (2) once and then operation (1) $b+1$ times, which leaves

\[ \sigma = \sigma_1a_{11} \cdots 1(b+1) \quad (m-1 \text{ 1's}). \]
This sequence of operations is repeated until we run out all the 1's. Thus those operations that do not change $L(\sigma)+S(\sigma)$ can only continue $O(y \cdot m)$ times. Since both $y$ and $m$ are less than the final answer, the total number of the whole operations do not exceed the cube of the answer.

To get $\bar{A}_n(n)$, we compute $A(i,i)$ for $i=1,2, \cdots$ successively until the value exceeds the input $n$. On the tape, each $a_i$ on the string $\sigma$ may be represented by a binary number or a unary number. Clearly it does not take so many steps to carry out the operations (1)-(3). Also it should be noted that if $L(\sigma)$ becomes larger than $n$ for the first time when computing $A(i,i)$ then $\bar{A}_n(n)$ is $i-1$. □

**Corollary 2.** There do not exist nonconstant lower bounds for the computation time of CRCW-PRAMs.

**Proof.** It is enough to show that for any recursive function $g(n)$ there exists a polynomial-time function $f(n)$ such that:

(i) $f(n) \leq \max_{0 \leq i \leq n} g(i)$ and

(ii) $f(n)$ is not bounded by any constant if $g$ is not.

Let $M$ be a Turing machine which computes $g(n)$. We construct the Turing machine $T$ that computes $f(n)$ as follows: By simulating $M$, $T$ computes $f(1), f(2)$ and so on successively and at the same time it counts the number $N$ of its moving steps (one step for the simulation of $M$'s one step). Suppose that when $N$ becomes $n$ (the input to $T$) $T$ is computing $f(i)$. Then $T$ halts with leaving on its output tape the maximum value in $f(1), f(2), \cdots$, and $f(i-1)$. It is not hard to see this $f(n)$ meets the above conditions (i) and (ii). □

3. **Star-Free Regular Expressions vs. Unbounded Fan-In Circuits.**

In this section we show a sufficient condition analogous to the Unger's well-known one [5] that says if a language $L$ over \{0,1\} is a regular set then $L$ can be recognized by bounded fan-in circuits of depth $O(\log n)$ and size $O(n)$. Our present one is:

**Theorem 3.** Let $\Sigma$ be an alphabet, $h$ be a homomorphism from $\Sigma$ into \{0,1\}* and $R$ be a star-free regular expression over $\Sigma$. Then if the on-set of a Boolean function $f$ can be given by $h(L(R))$ ($L(R)$ is the language generated by $R$), $f$ is computed by an unbounded fan-in circuit $C$ of a constant depth and a polynomial size on $n$.

By definition, a **star-free regular expression** over alphabet $\Sigma$ is a regular expression that can use $\phi$, $\varepsilon$, $\sigma$ for each $\sigma$ in $\Sigma$ and, as operations, complement $\bar{\cdot}$, union $\cup$, intersection $\cap$ and concatenation $\cdot$. Theorem 3 could help to introduce a desirable uniformity to the unbounded fan-in circuits. (The common uniformity for the bounded fan-in circuits [3] can of course be applied to the unbounded fan-in model but it is too weak to discuss relatively low depth complexity. See [2].)

**Proof of Theorem 3.** Let $\Sigma=\{a_1, a_2, \cdots, a_m\}$ and suppose that the regular expression $R$ consists of $k$ subexpressions $R_1, R_2, \cdots, R_k=R$. Then we construct Boolean expressions $f_{i,j}^1, f_{i,j}^2, \cdots, f_{i,j}^k$ for each $i$ and $j$ such that $0 \leq i \leq j \leq n$ where $n$ is the number of variables $x_1, x_2, \cdots, x_n$ of the target Boolean expression (or
equivalently the circuit $C$). $f$ is obtained as $f = f_0^R$. Now the expressions $f_{i,j}^1$ are of the following form:

(i) $R_l = \phi$. Then $f_{i,j}^1 = 0$ for all $i$ and $j$.

(ii) $R_l = \epsilon$. Then $f_{i,j}^1 = 1$ for all $i$ and $f_{i,j}^1 = 0$ for all $i$ and $j$ such that $i \neq j$.

(iii) $R_l = a_t \ (\epsilon \Sigma)$. Suppose that $h(a_t) = c_1 c_2 \cdots c_p \ (c_1, \cdots, c_p \in \{0,1\})$. Then $f_{i,j}^1 = 0$ if $j \neq i + p$. Otherwise $f_{i+i+p} = x_{i+1} x_{i+2} \cdots x_{i+p}$ where $x_{i+s}$ is $x_{i+s}$ if $c_s = 1$ and $x_{i+s}$ if $c_s = 0$.

(iv) $R_l = R_p \cup R_q$. Then $f_{i,j}^1 = (f_{i,j}^p + f_{i,j}^q)$ for all $i$ and $j$.

(v) $R_l = R_p \cap R_q$. Then $f_{i,j}^1 = (f_{i,j}^p f_{i,j}^q)$ for all $i$ and $j$.

(vi) $R_l = \overline{R_p}$. Then $f_{i,j}^1 = (f_{i,j}^p)$ for all $i$ and $j$.

(vii) $R_l = R_p \cdot R_q$. Then $f_{i,j}^1 = (\sum_{i \leq s \leq j} f_{i,j}^p f_{i,j}^q)$ for all $i$ and $j$.

To show the correctness of the construction, we prove the validity of the following sentence by the mathematical induction on the number of operations involved in the expression $R$:

$$f_{i,j}^1(x_1, x_2, \cdots, x_n) = 1 \text{ if and only if } v_{i+1} v_{i+2} \cdots v_j \in h(L(R))$$

where $v_{i+s}$ is the value (0 or 1) of the variable $x_{i+s}$. Details may be omitted since it is a standard application of the induction method.

As for the number of (unbounded fan-in) gates to realize the expression $f$, the following observation will be enough: (a) The number of Boolean expressions $f_{i,j}^1$ is $O(n^2)$. (Note that the length of $R$ or the number $k$ of its subexpressions is a constant.)

(b) To realize $f_{i,j}^1$ by circuit, we need only $O(1)$ gates for all the construction rules (i)-(vi). (c) For the rule (vii) also, one can see that $O(n)$ gates are enough. Thus the total number of gates necessary for the above construction is $O(n^3)$. \(\square\)

References


