ELSEVIER

Information Processing Letters 50 (1994) 211–216

Information Processing Letters

Bounded-width polynomial-size Boolean formulas compute exactly those functions in AC^0

Sorin Istrail [a,*], Dejan Zivkovic [b,**]

[a] *Sandia National Laboratories, Dept. 1423, Algorithms and Discrete Mathematics, Albuquerque, NM 87185-5800, USA*
[b] *Department of Mathematics and Computer Science, Savannah State College, Savannah, GA 31404, USA*

Communicated by D. Dolev; received 2 April 1992; revised 16 December 1993

Abstract

We show that the complexity classes AC^0 and NC^1 consist exactly of, respectively, constant and $O(\log n)$ width polynomial-size Boolean formulas.

Key words: Computational complexity; Boolean functions; Circuit complexity

1. Introduction

The complexity classes AC^0 and NC^1 and their modifications are well studied in the literature. For example, Furst et al. [4] and independently Ajtai [1] proved that $AC^0 \neq NC^1$. Spira [5] showed that a Boolean function is computable by polynomial-size formulas iff it is computable by logarithmic-depth circuits. In other words, the class NC^1 consists exactly of polynomial-size formulas. Barrington [2] gave another interpretation by showing that NC^1 consists exactly of those functions computed by bounded-width polynomial-size branching programs. In this paper we consider similar characterizations of AC^0 and NC^1 via the width of Boolean formulas.

2. Definitions

In this section we review some of the basic notions from Boolean circuit complexity (for more details see, for example, [6] and [3]).

A *Boolean circuit* on n Boolean variables $x_1, \ldots, x_n$ is a directed acyclic graph with the following properties. Each node of fan-in zero, called *input*, is labeled with a variable, the negation of a

* Supported in part by U.S. Department of Energy under contract DE-AC04-76DP00789. Email: scistra@cs.sandia.gov.
** Corresponding author. Email: dzivkov@uscn.cc.uga.edu.

SSDI 0020-0190(94)00036-X

variable, or the constants 0 or 1. The nodes of fan-in greater than zero, called *gates*, are labeled with AND or OR function. Lastly, there is a single sink node of fan-out zero called *output*. The *size* of a circuit is the number of its gates, and the *depth* is the length of the longest path from an input to the output.

A Boolean circuit computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$ in the natural way, i.e., for every $b = (b_1, \ldots, b_n) \in \{0,1\}^n$ we let $f(b)$ be the result of the output gate when the tuple $(b_1, \ldots, b_n)$ is given to the corresponding inputs. The class AC^0 consists of functions computed by polynomial-size circuits with unbounded fan-in and constant depth, and NC^1 is the class of those functions computed by circuits of fan-in two and depth $O(\log n)$ (note that here polynomial-size comes for free).

A *Boolean formula* is a circuit whose gates have fan-out at most one. Since the size of a Boolean formula does not count the inputs, we can attach to each gate that has an input its own copy of the input. In this way we can conveniently represent the Boolean formulas as trees and, in fact, we use the two terms interchangeably. A formula that can be represented by a binary tree is called a binary formula.

The *width* of a Boolean formula is formally defined using its underlying tree. Given a tree T, we first level the tree; that is, level 0 contains the root, level 1 contains the children of the root, level 2 contains the children of the children of the root, and so on. Let w_l be the number of nodes on level l. The width w of the tree T is simply $w = \max\{w_l \mid 0 \leqslant l \leqslant d\}$, where d is the depth of T. Of course, the width of a Boolean formula is the width of its underlying tree.

We define BFW^0 and BFW^1 to be the classes of Boolean functions computed by polynomial-size Boolean formulas that have, respectively, constant and $O(\log n)$ width.

Finally, we use $s(F)$ and $w(F)$ to denote the size and width of a formula F.

3. Preliminaries

In this section we first discuss an easy tree property about the width. After that we present several technical simulations needed for the proof of our main result.

There is a natural way to assign the notion of width to particular nodes of a tree T. Namely, the width of a node is the width of the subtree rooted at the node. Clearly, the width of the tree T is the width of its root. Also, all the nodes that have full width of T lie on a path from the root. This follows from the fact that no two children of a node can have the same width as the node itself. To see this, suppose a node x has width w_x and children nodes y and z of width w_y and w_z, respectively. Denote by T_y and T_z the subtrees rooted at the nodes y and z, and let l_y and l_z be the levels at which T_y and T_z attain the width. We now argue that $w_x = w_y = w_z$ is not possible. Indeed, if it were, then $l_y \neq l_z$ since otherwise one would have $w_x \geqslant w_y + w_z = 2w_x$, a contradiction. Therefore, either $l_y < l_z$ or $l_y > l_z$. But if $l_y < l_z$ then $w_x \geqslant w_y + 1 > w_x$, a contradiction. Similarly, if $l_y > l_z$ then $w_x \geqslant w_z + 1 > w_x$, again a contradiction. This completes the proof of the fact. The path along which lie all the nodes that have full width of T is called the *trunk* of the tree T.

Lemma 1. *A formula of width w and size s on n variables can be computed by a binary formula of width $2w$ and size $2^w ns$.*

Proof. We argue by induction on the width w. The case $w = 1$ is trivial, and for $w > 1$ consider the trunk of the tree T representing a formula F of width w and size s. To simplify the argument, we suppose that the trunk contains exactly two nodes; it will be clear later what modifications are needed in case the trunk contains one or more than two nodes. Denote the two gates G_1 and G_2 and suppose they are, say, $\wedge$ and $\vee$ node, respectively. Further assume that G_1 is at the root, and $P_1, \ldots, P_k s$ are its subtrees of depth greater than 0 not including the subtree rooted at the node G_2. Also, denote $p_1, \ldots, p_u$ the depth-0 children of G_1, i.e., all the literals feeding into it. Next, let $Q_1, \ldots, Q_m$ be all the subtrees of depth greater than 0 of the node G_2, and $q_1, \ldots, q_v$ its depth-0 children. We can assume that both u and v are at most n and no constants feed into G_1 or G_2, since otherwise the formula F is determined or we can remove the gate G_2.

Since the width of each P_i $(i = 1, \ldots, k)$ and Q_j $(j = 1, \ldots, m)$ is less than w, by the induction hypothesis we have equivalent binary formulas P_i' $(i = 1, \ldots, k)$ and Q_j', $(j = 1, \ldots, m)$. Moreover, for every $i = 1, \ldots, k$ and $j = 1, \ldots, m$, $w(P_i')$ and $w(Q_j')$ are at most $2(w-1)$, and $s(P_i') \leqslant 2^{w-1} n s(P_i)$ and $s(Q_i') \leqslant 2^{w-1} n s(Q_i)$.

We now build the binary tree T' that simulates T as follows. The top gate of T' is same as G_1, i.e., an $\wedge$ gate. The right subtree of the root is the binary tree P_1', and the left subtree begins with a line of $\wedge$ gates whose other input is the constant 1. The number of $\wedge$ gates on the line is equal to the depth of P_1'. Next, the last gate on the line has the right subtree being the binary tree P_2'. Again, the left subtree of the last gate begins with a line of $\wedge$ gates whose other input is the constant 1, and the number of them is equal to the depth of P_2'. Continuing in this way, we partially build the tree T' that has as the bone a long line of $\wedge$ gates except the last gate, and the other input of the $\wedge$ gates is either the constant 1 or one of the binary trees $P_1', \ldots, P_k'$. Finally, we extend the bone down with a line of u $\wedge$ gates whose other inputs are the literals $p_1, \ldots, p_u$. Now, the last $\wedge$ gate on the line has one input p_u, and the other input is the same node as the gate G_2, i.e., an $\vee$ gate. The right subtree of the $\vee$ gate will be the tree Q_1', and the left subtree begins with a line of $\vee$ gate whose other input is the constant 0. The number of the $\vee$ gates is equal to the depth of the tree Q_1'. Proceeding just as before, we finish up the construction of T' by extending the last node on the line of $\vee$ gates that corresponds to the tree Q_m' with a line of v $\vee$ nodes whose other inputs are the literals $q_1, \ldots, q_v$.

Clearly, the tree T' thus constructed is binary and

$$\begin{aligned} w(T') &\leqslant \max\{\max\{w(P_i') \mid 1 \leqslant i \leqslant k\}, \max\{w(Q_j') \mid 1 \leqslant j \leqslant m\}\} + 2 \\ &\leqslant 2(w-1) + 2 = 2w, \end{aligned}$$

$$\begin{aligned} s(T') &\leqslant \sum_{i=1}^{k} 2s(P_i') + u + \sum_{j=1}^{m} 2s(Q_j') + v \\ &\leqslant 2\Big(\sum_{i=1}^{k} s(P_i') + \sum_{j=1}^{m} s(Q_j') + n\Big) \\ &\leqslant 2\Big(2^{w-1} n \sum_{i=1}^{k} s(P_i) + 2^j w - 1 n \sum_{j=1}^{m} s(Q_j) + n\Big) \\ &\leqslant 2^w n \Big(\sum_{i=1}^{k} s(P_i) + \sum_{j=1}^{m} s(Q_j) + 1\Big) \\ &\leqslant 2^w n s. \qquad \square \end{aligned}$$

As an aside and no surprise, note that the above lemma shows that the power of polynomial-size binary and general formulas is the same provided their width is bounded or logarithmic.

Lemma 2. *A formula of depth d and size s on n variables can be computed by a binary formula of width $2d+1$ and size $4^d(n+1)(s+1)$.*

Proof. We argue by induction on the depth d. For $d=0$ the claim is obvious. For the induction step, suppose F is a formula of depth $d>1$ and size s. Let T be the tree that represents F. Denote by $T_1,\ldots,T_k$ the subtrees rooted at the children of the root of T whose depth is greater than 0. Also, denote $t_1,\ldots,t_l$ the depth-0 children of the root of T, i.e., all the literals feeding into it. We can assume that $l \leqslant n$ and no constants appear as children of the root of T, since otherwise the formula is determined and we are done.

Since the depth of $T_1,\ldots,T_k$ is less than d, by the induction hypothesis we have equivalent binary formulas $T_1',\ldots,T_k'$ such that $w(T_i') \leqslant 2d-1$ and $s(T_i') \leqslant 4^{d-1}(n+1)(s(T_i)+1)$, $(i=1,\ldots,k)$.

Now, to construct the desired tree T' equivalent to T we proceed in pretty much the same way as in the proof of Lemma 1. The root of T' is the same gate as the root of T, say an $\vee$ gate. The right subtree of the root is T_1', and the left subtree begins with a line of $\vee$ gates whose other input is the constant 0. The number of the $\vee$ gates on the line is equal to the depth of T_1'. Next, the last gate on the line has as its right subtree the tree T_2', and the left subtree begins with a line of $\vee$ gates whose other input is 0 and whose number is equal to the depth of T_2'. The process is clear now for each of the rest of $T_3',\ldots,T_k'$. To finish up the construction, we extend the last gate on the line associated with T_k' to a line of $\vee$ gates whose other inputs are the literals $t_1,\ldots,t_l$.

Clearly the tree T' thus constructed is binary and

$$w(T') \leqslant \max\{w(T_i') \mid 1 \leqslant i \leqslant k\} + 2 \leqslant (2d-1)+2 = 2d+1,$$

$$\begin{aligned}
s(T') &\leqslant \sum_{i=1}^{k} 2s(T_i') + l \leqslant \sum_{i=1}^{k} 2\cdot 4^{d-1}(n+1)(s(T_i)+1) + n \\
&\leqslant 2\cdot 4^{d-1}(n+1)\Big(\sum_{i=1}^{k} s(T_i) + k\Big) + n \\
&\leqslant 2\cdot 4^{d-1}(n+1)\cdot 2s + n = 4^d(n+1)s + n \\
&\leqslant 4^d(n+1)(s+1). \qquad \Box
\end{aligned}$$

Lemma 3. *A binary formula of width w and size s can be computed by an unbounded fan-in circuit of depth $2w$ and size s.*

Proof. Let F be a formula represented by a binary tree T that has the width w and size s. By induction on w, we will construct a circuit C of depth at most $2w$, the size at most s, and such that C computes F.

The base case $w=1$ is trivial. For $w>1$, consider the trunk of the tree T. It contains a sequence of $\vee$ and $\wedge$ gates in any order. For the sake of concreteness, suppose that from the root of T down the trunk there first is a run of g_1 $\vee$ gates, then a run of g_2 $\wedge$ gates, and so on alternating until the end of the trunk. Further suppose that the trunk ends with, say, a run of g_r $\vee$ gates. Thus, the trunk is a sequence of r alternating runs of $\vee$ or $\wedge$ gates, it starts and ends with a run of $\vee$ gates, and the kth run has g_k gates $(k=1,\ldots,r)$. With these assumptions we have that r is an odd number, i.e., $r=2m+1$ for some $m=0,1,2,\ldots$.

Let $T_1^k, T_2^k, \ldots, T_{g_k}^k$ be the subtrees that feed into the gates of the kth run ($k = 1, \ldots, r$). If $T_1^k, T_2^k, \ldots, T_{g_k}^k$ denote also formulas computed by the corresponding subtrees, we can write

$$F = T_1^1 \vee \cdots \vee T_{g_1}^1 \vee (T_1^2 \wedge \cdots \wedge T_{g_2}^2 \wedge (\cdots \wedge (T_1^r \vee \cdots \vee T_{g_r}^r) \cdots)). \tag{1}$$

By applying the distributive law and expanding the expression on the right-hand side of (1) as much as possible, we have

$$\begin{aligned} F = {} & T_1^1 \vee \cdots \vee T_{g_1}^1 \\ & \vee \bigvee_{i=1}^{m} \Bigl(\bigvee_{j=1}^{g_{2i+1}} T_1^2 \wedge \cdots \wedge T_{g_2}^2 \wedge T_1^4 \wedge \cdots \wedge T_{g_4}^4 \wedge \cdots \wedge T_1^{2i} \wedge \cdots \wedge T_{g_{2i}}^{2i} \wedge T_j^{2i+1} \Bigr). \end{aligned} \tag{2}$$

Now we build the circuit C that computes the right-hand side of (2) as follows. The root is an $\vee$ gate, and the next level consists of $g_3 + g_5 + \cdots + g_r$ $\wedge$ gates. Below these come the circuits for every tree $T_1^k, T_2^k, \ldots, T_{g_k}^k$, ($k = 1, \ldots, r$), given by the induction hypothesis. If we index the $\wedge$ gates on the first level by (i, j), where $i = 1, \ldots, m$ and $j = 1, \ldots, g_{2i+1}$, then the (i, j)th $\wedge$ gate connects to the roots of the circuits for $T_1^2, \ldots, T_{g_2}^2, T_1^4, \ldots, T_{g_4}^4, \ldots, T_1^{2i}, \ldots, T_{g_{2i}}^{2i}, T_j^{2i+1}$, thus computing the (i, j)th product from (2). Finally, the top $\vee$ gate is connected to the roots of the equivalent circuits for $T_1^1, \ldots, T_{g_1}^1$ and every $\wedge$ gate on the first level.

Therefore, the circuit C correctly computes the formula F and by induction has the depth at most $2(w-1) + 2 = 2w$. Moreover, the size of C is

$$s(C) \leqslant \sum_{i=1}^{r} \sum_{j=1}^{g_i} s(T_j^i) + g_3 + g_5 + \cdots + g_r + 1 \leqslant s. \qquad \square$$

4. The result

The next theorem shows that the complexity classes AC^0 and NC^1 consist precisely of, respectively, constant and $\mathrm{O}(\log n)$ width polynomial-size Boolean formulas.

Theorem 4. $\mathrm{AC}^0 = \mathrm{BFW}^0$ *and* $\mathrm{NC}^1 = \mathrm{BFW}^1$.

Proof. The inclusion $\mathrm{BFW}^0 \subseteq \mathrm{AC}^0$ is a consequence of Lemma 1 and Lemma 3. To see the converse $\mathrm{AC}^0 \subseteq \mathrm{BFW}^0$, we first observe that an AC^0 circuit of depth d and size s can easily be made into a formula of depth d and size $\mathrm{O}(s^d)$. Now the inclusion follows from Lemma 2.

To prove $\mathrm{NC}^1 = \mathrm{BFW}^1$ we use the fact that NC^1 consists of polynomial-size $\mathrm{O}(\log n)$ depth formulas, which in turn have the same power as the polynomial-size formulas. Then $\mathrm{BFW}^1 \subseteq \mathrm{NC}^1$ is trivial, and $\mathrm{NC}^1 \subseteq \mathrm{BFW}^1$ follows from Lemma 2. $\square$

Acknowledgment

We would like to thank David Barrington for many helpful discussions. His valuable ideas and comments were influential to the paper. Suggestions generated in discussions with Steve Lindell are also acknowledged with pleasure.

References

[1] M. Ajtai, Σ_1^1-formulae on finite structures, *Ann. Pure Appl. Logic* **24** (1983) 1–48.
[2] D.A. Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, in: *Proc. 18th Ann. ACM Symp. on Theory of Computing* (1986) 1–5.
[3] R.B. Boppana and M. Sipser, The complexity of finite functions, MIT/LCS Tech. Rept. No. 405, 1989.
[4] M. Furst, J. Saxe and M. Sipser, Parity, circuits, and the polynomial time hierarchy, *Math. Systems Theory* **17** (1984) 13–27.
[5] P.M. Spira, On time-hardware complexity tradeoffs for Boolean functions, in: *Proc. 4th Hawaii Symp. on System Sciences* (1971) 525–527.
[6] I. Wegener, *The Complexity of Boolean Functions* (Wiley–Teubner, New York, 1987).
[7] D. Zivkovic, Non-probabilistic techniques in circuit complexity, Ph.D. Thesis, Wesleyan University, 1992.