REALIZING BOOLEAN FUNCTIONS ON DISJOINT SETS OF VARIABLES[+]

Wolfgang J. Paul[++]

TR 75-215
(Revised version)

June 1975

Department of Computer Science
Cornell University
Ithaca, New York 14853

[+] Revised version of "Ashenhurst Decomposition of Switching Functions is not Optimal", Cornell Computer Science Technical Report 74-215.

[++] This research was supported by DAAD (German Academic Exchange Service) Grant No. .530/402/563/5.

REALIZING BOOLEAN FUNCTIONS ON DISJOINT SETS OF VARIABLES+

Wolfgang J. Paul++

Cornell University

Abstract:

For switching functions f let C(f) be the combinational complexity of f. We prove that for every $\varepsilon > 0$ there are arbitrarily complex functions $f:\{0,1\}^n \to \{0,1\}^n$ such that $C(f\times f) \leq (1+\varepsilon)C(f)$ and arbitrarily complex functions $g:\{0,1\}^n \to \{0,1\}$ such that $C(v \circ (f\times f)) \leq (1+\varepsilon)C(f)$. These results and the techniques developed to obtain them are used to show, that Ashenhurst decomposition of switching functions does not always yield optimal circuits and to prove a new result concerning the trade off between circuit size and monotone circuit size.

+Revised version of "Ashenhurst Decomposition of Switching Functions is not Optimal", Cornell Computer Science Technical Report 74-215.

++This research was supported by DAAD (German Academic Exchange Service) Grant No. .530/402/563/5.

REALIZING BOOLEAN FUNCTIONS ON DISJOINT SETS OF VARIABLES

Wolfgang J. Paul
Cornell University

1. Introduction:

Let f be a switching function and C(f) the size of the smallest circuit with 2-input 1-output gates which realizes f. Define $(f \times g)(x,y) = (f(x),g(y))$ and define $(f \circ g)(s) = f(g(x))$.+ Let $v:\{0,1\}^2 \to \{0,1\}$ denote the logical OR.

In this paper we deal with the following question: How behave $C(f \times f)$ and $C(v \circ (f \times f))$ in terms of C(f)? Intuition suggests at least for the first problem $C(f \times f) = 2C(f)$ for any f, because two evaluations of f on disjoint sets of variable "have nothing to do with each other".

We show that this intuition is wrong and that for certain functions f the above conjectures are far off reality. Our main results are: For any $\varepsilon > 0$ there are arbitrarily complex functions $f:\{0,1\}^n \to \{0,1\}^n$ such that $C(f \times f) \leq (1+\varepsilon)C(f)$ and there are arbitrarily complex functions $g:\{0,1\}^n \to \{0,1\}$ such that $C(v \circ (g \times g)) \leq (1+\varepsilon)C(g)$.

However we were not able to show, that for any $d > 0$ there are arbitrarily complex <u>one output</u> functions $f:\{0,1\}^n \to \{0,1\}$

+x and y are 0-1 vectors of appropriate length.

such that $C(f \times f) \leq 2C(f) - d$. We cannot show this even for $d = 1$ and consider this as an interesting problem.

It turns out, that with the above theorems and the tools developed to obtain them we can do more than only exhibit the existence of pathological behavior of circuit complexity.

We describe a well known heuristic approach for circuit design: a function $f:\{0,1\}^n \to \{0,1\}$ has simple disjoint decomposition if there are functions $h:\{0,1\}^s \to \{0,1\}$ and $g:\{0,1\}^{n-s+1} \to \{0,1\}$ such that $f(x,y) = g(h(x),y)$[+]. Such a decomposition of f suggests realizations of f which make use of this decomposition in an obvious way. We show, that these realizations are not always optimal, hereby settling a conjecture stated in [3, p. 430]. However we can prove optimality of such realizations under the severe restrictions that $C(h) = s-1$ and f depends on all its arguments.

Finally we turn to monotone circuits; these are circuits in which all gates are AND-gates or OR-gates. For a monotone function f denote by MC(f) the size of the smallest monotone circuit which realizes f. Using a result due to M.J. Fischer [5], which establishes $MC(f \times g) = MC(f) + MC(g)$ for all monotone f and g, we exhibit a sequence of monotone functions $f_n:\{0,1\}^n \to \{0,1\}^I$ such that for almost all n: $MC(f_n) \geq n^{1.5}/(8 \log n)$[++] but $C(f_n) \leq O(n\log^2 n)$[++].

[+] x and y are 0-1 vectors of appropriate length.

[++] all logarithms in this paper are taken to the base 2.

[+++] $\log^i n = (\log n)^i$.

2. Preliminaries:

We introduce some notations and definitions which we will use throughout this paper.

$\mathbb{N} = \{1,2,\ldots\}$, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

For $m,n \in \mathbb{N}_0$: $[m:n] = \{m,m+1,\ldots,n\}$

If Σ is a finite alphabet, then Σ^* is the set of finite words over Σ. For $w \in \Sigma^*$ let $|w|$ denote the length of w and w_i the i^{th} symbol of w.

For finite sets A and $n \in \mathbb{N}$, A^n is the n-fold Cartesian product of **A** . For $w \in \{0,1\}^n$, w_i denotes the i^{th} component of w.

Any function $f:\{0,1\}^n \to \{0,1\}^m$ is called a <u>switching function</u>.

We use the special symbols $\wedge$, $\vee$, $-$, $\oplus$, $\equiv$ for the functions logical AND, logical OR, negation, + mod 2, equivalence.

For functions $f:\{0,1\}^n \to \{0,1\}^m$ and $g:\{0,1\}^m \to \{0,1\}^p$, $g \circ f:\{0,1\}^n \to \{0,1\}^p$ is the usual composition of functions.
For functions $f:\{0,1\}^n \to \{0,1\}^m$ and $g:\{0,1\}^p \to \{0,1\}^q$, $f \times g: \{0,1\}^{n+p} \to \{0,1\}^{m+q}$ is defined by

$$((f\times g)(a_1,\ldots,a_n,b_1,\ldots,b_p))_i = \begin{cases} (f(a_1,\ldots,a_n))_i & \text{if } 1 \le i \le m \\ (g(b_1,\ldots,b_p))_{i-m} & \text{if } m < i \le m+q \end{cases}$$

A switching circuit is a finite directed acyclic graph G together with three mappings λ, r, ℓ. Each node in G has indegree 0 or 2. Nodes with indegree 0 are input nodes and are usually denoted $x_1, x_2, \ldots$; nodes with outdegree 0 are output nodes.

λ: {nodes of G} $\to \Omega$, $\Omega = \{h \mid h:\{0,1\}^2 \to \{0,1\}\}$ labels the nodes of G with operations.

r, ℓ: {nodes of G} $\to$ {nodes of G} $\cup \{\phi\}$ are mappings such that

i) for each input node x_i: $r(x_i) = \ell(x_i) = \phi$

ii) if node c has indegree 2 and c_1 and c_2 are the fathers of c_1 then $r(c) \cup \ell(c) = \{c_1, c_2\}$

If G has n input nodes, then with each node c in G one associates in a natural way a function $res_G(c)$: $\{0,1\}^n \to \{0,1\}$ by defining for $a \in \{0,1\}^n$

$$res_G(c)(a) = \begin{cases} a_i & \text{if c is the input node } x_i \\ \lambda(c)(res_G(\ell(c))(a), res_G(r(c))(a)) & \text{if indegree(c)=2.} \end{cases}$$

A function f: $\{0,1\}^n \to \{0,1\}^m$ is computed by the circuit H if in the circuit H there are m output nodes $c_1,\ldots,c_m$ such that for all $a \in \{0,1\}^n$: $f(a) = (res_G(c_1)(a),\ldots,res_G(c_m)(a))$.

Define $C(G)$, the complexity of the circuit G as the number of nodes with indegree 2 (number of gates) in G, and define $C(f)$, the (combinational) complexity of the function f, by $C(f) = \min\{C(G) \mid G \text{ computes } f\}$. A circuit G is an optimal circuit for f if G computes f and $C(G) = C(f)$.

Some facts about optimal circuits which we often use implicitly are stated in:

Lemma 1:

a) if a circuit G has a node c with indegree 2 such that $res_G(r(c)) = const$ or $res_G(\ell(c)) = const$, then G can be transformed into a circuit G' such that G' computes the same functions as G and $C(G') \leq C(G) - 1$, i.e. G is not optimal.

b) if G is an optimal circuit, then for all nodes c with indegree 2, $\lambda(c)$ depends properly on both arguments.

c) for each $f: \{0,1\}^n \to \{0,1\}^m$ there is an optimal circuit G for f with at most m output nodes.

3. A technical lemma

In this section we show how to save gates by simultaneously evaluating the same function g on many disjoint sets of variables instead of making many copies of a circuit for g.

For $g: \{0,1\}^n \to \{0,1\}^m$ and $k \in \mathbb{N}$ define

$$g^0 = g,\ g^k = g x g^{k-1}\ .$$

<u>Lemma 2</u>: There is a constant α such that for all $k, n \in \mathbb{N}$ and all $g:\{0,1\}^n \to \{0,1\}$ holds:

$$C(g^k) \leq \alpha \max\{n\, 2^n, nk \log^2 k\}$$

The idea of the proof is simple: to evaluate the finite function g on k inputs, we construct a Turing machine M, which sorts the inputs (using merge sort), determines the result of g on these inputs by table look up, then "unsorts" the results and prints them out. Applying to machine M the Fischer-Pippenger Conversion of Turing machines to circuits yields the desired circuit. The proof of Lemma 2 itself however is somewhat lengthy because we have to take care of some technical details.

We first describe a result from [4] [6] which we use later. Let M be a deterministic multitape Turing machine with input alphabet = output alphabet = Σ. Let f_M be the function computed by M, i.e. $f_M:\Sigma^* \to \Sigma^*$, $f_M(w)$ = the output produced by M on input w.

Let $D \subseteq \Sigma^n$ for some n. We say M is <u>length respecting</u> on D if for all $w, w' \in D$: $|f(w)| = |f(w')|$.

For $w \in \Sigma^*$, $i \in [1:|f(w)|]$ let $t(w,i)$ be the time at which M prints out $(f(w))_i$ when M was started with w. We say M is <u>uniform</u> on D if for all $w,w' \in D$: $t(w,i) = t(w'i)$ for all i for which the equation is defined.

Let $\sigma = \lceil \log(\text{cardinality of } \Sigma) \rceil$+ and let $\psi: \Sigma \xrightarrow{1-1} \{0,1\}^\sigma$ be an encoding of Σ. Extend $\psi: \Sigma^* \to \{0,1\}^*$ by defining: $\psi(w_1 \ldots w_n) = \psi(w_1) \ldots \psi(w_n)$

+$\lceil x \rceil = \min\{n | n \in \mathbb{N} \text{ and } n \geq x\}$

Fact 1 (M. J. Fischer and N. Pippenger): +

For each Turing machine M there is a constant β_M such that for all $n \in \mathbb{N}$ holds: If for any $D \subseteq \Sigma^n$ M is length respecting and uniform on D and if for all $w \in D$ M halts after at most T(D) steps, then one can construct from M a circuit with $\beta_M T(D) \log T(D)$ gates, which on input $w \in \psi(D)$ computes $\psi(f_M(w))$.

Define $\text{int}:\{0,1\}^* \to \mathbb{N}$ and $\text{bin}: \mathbb{N} \to \{0,1\}^*$ by

int(b) = the integer whose binary representation is b

bin(b) = binary representation of b.

We design a Turing machine M, which given a description of $g:\{0,1\}^n \to \{0,1\}$ and k arguments (i.e. binary n-tuples) computes g at these arguments. Thus for $n, k \in \mathbb{N}$ on inputs of the form

(3.1) $a_0 \# \ldots \# a_{2n-1} \#\# b_1 \# \ldots \# b_k$

where $a_i \in \{0,1\}$, $b_i \in \{0,1\}^n$ the Turing machine M prints out

$$a_{\text{int}(b_1)} \cdots a_{\text{int}(b_k)}.$$

M operates as follows:

(1) From the list

$L_0 = [b_1,\ldots,b_k]$

M computes the list

$L_1 = [(b_1,\text{bin}(1)),\ldots,(b_k,\text{bin}(k))]$.

+This theorem is stated in [4] only for $D = \Sigma^n$ but the proof of the theorem described here is implicit in the proof of the corresponding theorem in [4].

(2) Using merge sort (for details see e.g. [1 , pp 66,67]). M sorts the list L_1 according to the size of the numbers $int(b_i)$. So it produces the list

$$L_2 = [(b_{i_1}, bin(i_1)), \ldots, (b_{i_k}, bin(i_k))]$$

where $int(b_{i_j}) \leq int(b_{i_{j+1}})$ for all $j \in [1:k-1]$.

(3) Scanning $a_0 \# \ldots \# a_{2^n-1}$ once, M computes

$$L_3 = [(a_{int(b_{i_1})}, bin(i_1)), \ldots, (a_{int(b_{i_k})}, bin(i_k)]$$

(4) Again using merge sort, M sorts the list L_3 according to the size of the $bin(i_j)$. So M produces

$$L_4 = [(a_{int(b_1)}, bin(1)), \ldots, (a_{int(b_k)}, bin(k)) .$$

(5) Scanning L_4 once, M prints out

$$L_5 = a_{int(b_1)} \cdots a_{int(b_k)} .$$

<u>Time analysis</u>: each of the parts (1) and (5) takes time $O(n \cdot k)$. Each of part (2) and (4), which use merge sort of a list of k elements involve klogk comparisons, each comparison involving n steps on M. It turns out that merge sort can be implemented on M using time $O(nklogk)$. Part (3) takes time at most $O(2^n+kn)$.

Now one can modify M slightly such that for some constants k_1, k_2, k_3 and for all n and k: part (1) takes exactly $k_1 \cdot n \cdot k$ steps, each of part (2) and (4) take exactly time $k_2 nklogk$ and part (3) takes exactly time $k_3(2^n+kn)$.

Thus if part (5) is programmed in the most straightforward way and if for $n,k \in N$ we define $D_{n,k}$ to be the set of all inputs of the form (3.1), then for each n and k the modified machine M is lengthrespecting and uniform on $D_{n,k}$. Also there is a constant k_4 such that for all n and k: M halts after at most

$$(3.2)\quad T(D_{n,k}) = k_4 \max\{2^n + kn, nk\log^2 k\}$$

steps if started with any $w \in D_{n,k}$.

Hence by fact 1 for each n and k there is a circuit $G_{n,k}$ which on iputs

$$\psi(a_0 \# \ldots \# a_{2^n-1} \#\# b_1 \# \ldots \# b_2)$$

computes

$$\psi(a_{int(b_1)} \cdots a_{int(b_k)}\ ,$$ where ψ is a coding as defined above, and

$$(3.3)\quad C(G_{n,k}) \leq \beta_M T(D_{n,k}) \log T(D_{n,k}).$$

$$= \beta'_M \max\{(2^n+kn)\log(2^n+kn), nk\log k \cdot \log(nk\log k)\}.$$

Now observe, that in max{...,...} for $k < 2^n/n^4$ the first term dominates and can be estimated from above by $4n2^n$. If $k \geq 2^n/n^4$ the second term of max{...,...} can be estimated by $4nk \log^2 k$. Hence for all n and k:

$$C(G_{n,k}) \leq \beta''_M \max\{n\, 2^n, nk\log^2 k\} \text{ where } \beta''_M = 4\beta'_M.$$

By trivial coding and decoding networks the circuit $G_{n,k}$ can be transformed into a circuit $G'_{n,k}$, such that $G'_{n,k}$ has 2^n+kn

inputs and kn outputs, and on input $a_1 \ldots a_{2^n-1}\ b_1 \ldots b_k$ $a_i \in \{0,1\}, b_i \in \{0,1\}^n$, $G'_{n,k}$ produces the output $a_{int(b_1)} \ldots a_{int(b_k)}$. Moroever $C(G'_{n,k}) \leq \alpha \max\{n\,2^n, nk \log^2 k\}$. Setting the first 2^n inputs of $G'_{n,k}$ to the description of a function $g:\{0,1\}^n \to \{0,1\}^n$ -- more precisely setting $a_i = g(bin(i))$ for all $i \in [0:2^n-1]$ -- does not increase the cost and yields the desired circuit for g^k.

In exactly the same way one proves

Lemma 3: There is a constant γ such that for all $k, n \in \mathbb{N}$ and all $g:\{0,1\}^n \to \{0,1\}^n$

$$C(g^k) \leq \gamma \max\{n^2 2^n, nk\log^2 k\} \quad .$$

4. Functions on Disjoint Sets of Variables

We state some facts which we will use later:

Fact 2

(a) There is $N \in \mathbb{N}$ such that for all $n > N$ and all but a vanishing fraction of the functions $f:\{0,1\}^n \to \{0,1\}$ holds: $C(f) \geq 2^n/3n$.

(b) There is $N \in \mathbb{N}$ such that for all $n > N$ and all but a vanishing fraction of the functions $f:\{0,1\}^n \to \{0,1\}^n$ holds: $C(f) \geq 2^n/3$.

These facts are proven by the well-known Shannon counting argument [11] (See also [4] or [14]).

Theorem 1: For all $\varepsilon > 0$ there is $d \in \mathbb{N}$ such that for almost all $n \in \mathbb{N}$ there is $f:\{0,1\}^n \to \{0,1\}^n$ such that

(4.1) $C(f) \geq 2^{\sqrt[d]{n}}$

(4.2) $C(vo(f \times f)) \leq (1+\varepsilon)C(f)$

To prove Theorem 1 we first show

<u>Lemma 4</u>: for all $\varepsilon > 0$ there are $N, c \in \mathbb{N}$ such that for all $n > N$ there is $m \in [n:n^{c-2}]$ and $g:\{0,1\}^m \to \{0,1\}$ such that

(4.3) $\quad C(g) \geq 2^{\sqrt[c]{m}}$

(4.4) $\quad C(vo(gxg) \leq (1+\varepsilon)C(g)$

This implies Theorem 1 as follows: Given ε and given any $n > N^{c-2}$, where N, c are as in Lemma 4. Then by Lemma 4 there is an $m \in [\sqrt[c-2]{n}:n]$ and $g:\{0,1\}^m \to \{0,1\}$ such that (4.3) and (4.4) hold. Define $f:\{0,1\}^n \to \{0,1\}$ by

$$f(a) = g(a_1,\ldots,a_m) \text{ for all } a \in \{0,1\}^n .$$

Clearly $C(f) = C(g)$ and $C(vo(fxf)) = C(vo(gxg))$,

so

(4.5) $C(f) = C(g) \geq 2^{m^{1/c}} \geq 2^{n^{1/(c(c-2))}}$

(4.6) $C(vo(fxf)) = C(vo(gxg)) \leq (1+\varepsilon)C(g) = (1+\varepsilon)C(f)$.

Thus Theorem 1 holds with $d = c\cdot(c-2)$ if Lemma 4 is correct. Lemma 4 is proven by contradiction. Given $\varepsilon > 0$, choose c and N such that

(4.7) $(c-2)\log(1+\varepsilon) \geq 3$

(4.8) $N = \max\{N_1,N_2,N_3,N_4\}$

$N_1 = \min\{N \mid \forall n > N\ \exists\ g:\{0,1\}^n \to \{0,1\}$ such that $C(g) \geq 2^n/3n$

$N_2 = \min\{n \mid 2^{n^{1/(c-1)}}/3n^{1/(c-1)} \geq 2^{n^{1/c}}\}$

$N_3 = \min\{n \mid n\ 2^n \geq (c-2)^2 n^{c-1} \log^2 n\}$

$N_4 = \min\{n \mid n^2 2^n > 3\alpha n 2^n + c^n)\}$, α as in Lemma 2 .

(4.9) Let $n > N$ and assume that for all $m \in [n:n^{c-2}]$ and all $g:\{0,1\}^m \to \{0,1\}$ such that $C(g) \geq 2^{\sqrt[c]{m}}$ holds:
$C(v \circ g^2) \geq (1+\varepsilon)C(g)$.

Choose $g:\{0,1\}^n \to \{0,1\}$ such that

(4.10) $C(g) \geq 2^n/3n.$

For $i \in [0:(c-2)\log n]$ define

(4.11) $G(0) = g$

$G(i) = v \circ (G(i-1) \times G(i-1))$

For $p \in N$ define $v_p : \{0,1\}^p \to \{0,1\}$,

$v_p(a_1,\ldots,a_n) = a_1 v \ldots v a_n$.

Clearly for all $i \in [0:(c-2)\log n]$:

$G(i):\{0,1\}^{n2^i} \to \{0,1\}$

$G(i) = v_{2^i} o g^{2^i}$

Moreover for $i \leq (c-2)\log n$:

$n2^i \leq n^{c-1}$, thus $n \geq \sqrt[c-1]{2^i n}$ and

$C(G(i)) \geq C(g) \geq 2^n/3n$

$$\geq \frac{2^{\sqrt[c-1]{2^i n}}}{3\sqrt[c-1]{2^i n}} \geq 2^{\sqrt[c]{2^i n}} \qquad \text{by (4.8).}$$

Hence the assumption (4.9) applies to $G(i)$ for all $i \in [0:(c-2)\log n]$.

Thus using (4.10) and (4.11) this implies by induction on i:

$C(G(i)) \geq (1+\varepsilon)^i 2^n/3n$

and in particular for $i = (c-2)\log n$

$$C(v_{n^{c-2}} \circ g^{n^{c-2}}) \geq n^{(c-2)\log(1+\varepsilon)} \cdot 2^n/3n$$

$$\geq n^2\, 2^n/3 \qquad \text{by (4.7)}$$

But applying Lemma 2 with $k = n^{c-2}$ yields

$$C(g^{n^{c-2}}) \leq \alpha\max\{n2^n, (c-2)^2 n^{c-1}\log^2 n\}$$

$$= \alpha n 2^n \qquad \text{by (4.8)}$$

Thus

$$C(v_{n^{c-2}} \circ g^{n^{c-2}}) \leq \alpha n 2^n + n^{c-2}$$

$$< n^2 2^n/3 \qquad \text{by (4.8).}$$

This contradiction proves Lemma 4 and the theorem.

Corollary 1: For all $d > 0$ there are arbitrarily complex functions $f:\{0,1\}^n \to \{0,1\}$ such that $C(v\circ(f x f)) \leq 2\, C(f) - d$.

Theorem 2: For all $\varepsilon > 0$ there is $d \in N$ such that for almost all $n \in N$ there is $f:\{0,1\}^n \to \{0,1\}^n$ such that

(4.12) $\quad C(f) \geq 2^{\sqrt[d]{n}}$

(4.13) $\quad C(f x f) \leq (1+\varepsilon)C(f)$

Theorem 2 is proven in much the same way as Theorem 1. Instead of (4.11) one defines $G(i) = G(i-1) \times G(i-1)$.

Corollary 2: For all $d > 0$ there are arbitrarily complex functions $f:\{0,1\}^n \to \{0,1\}$ such that

$$C(f x f) \leq 2C(f)-d$$

We conclude this section by an alternative proof of Corollary 2, which was pointed out to us by an unknown referee:

There are 2^{n^2} different binary nxn-matrices D. Each matrix defines a function $f_D:\{0,1\}^n \to \{0,1\}^n$, $f_D(a) = Da$, where Da is the Boolean product+ of the matrix D and the vector a. $f_D \neq f_{D'}$ for $D \neq D'$, hence by a counting argument one concludes: For almost all n and all but a vanishing fraction of all binary nxn matrices: $C(f_D) \geq n^2/7 \log n$. Choose D such that $C(f_D) \geq n^2/7 \log n$. Assuming Corollary 2 is false for d, one proves by induction on i: $C(f_D^i) \geq i\, C(f_D) - id$. In particular for i = n:

$$C(f_D^n) \geq n^3/7\log n - nd.$$

But for (column) vectors $x_i \in \{0,1\}^n$ $f_D^n(x_1,\dots,x_n)$ is the boolean matrix product of D with the matrix $(x_1 \dots x_n)$, hence by Strassen's algorithm for matrix multiplication [12] this can be done with cost $O(n^{2.81})$, a contradiction.

5. Ashenhurst decomposition

Definition [2]: $f:\{0,1\}^n \to \{0,1\}$ has simple disjoint decomposition if there is $h:\{0,1\}^s \to \{0,1\}$, $2 \leq s \leq n-1$, $g:\{0,1\}^{n-s+1} \to \{0,1\}$ and a permutation $\binom{1 \dots n}{i_1 \dots i_n}$ such that

$$f(a) = g(h(a_{i_1},\dots,a_{i_s}), a_{i_{s+1}},\dots,a_{i_m}) \text{ for all } a \in \{0,1\}^n$$

+Replace in the usual definition of the product of a matrix with a vector "·" by "∧" and "+" by "∨".

If f has simple disjoint decomposition, this decomposition suggests in a natural way a realization of f in the form of Figure 1, where g and h are realized optimally. We say that such a realization makes use of a decomposition of f.

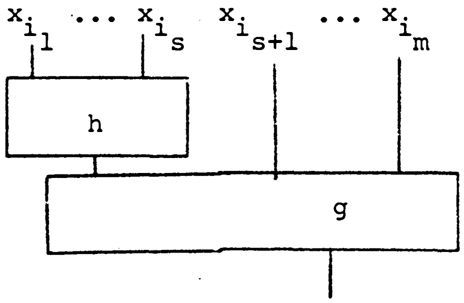

Figure 1

Theorem 3: A realization of a switching function f which makes use of a decomposition of f is not necessarily optimal.

Proof: For all $f:\{0,1\}^n \to \{0,1\}$, $v \circ (f \times f)$ can be decomposed in an obvious way. If realizations of f, which make use of a decomposition of f were always optimal, then $C(v \circ (f \times f)) \geq 2C(f)$. But this contradicts Theorem 1.

As pointed out by an unknown referee, Theorem 3 can also be proven directly.

Let $f:\{0,1\}^n \to \{0,1\}$ be such that $C(f) \geq 2^n/3n$ define $F:\{0,1\}^{2n+1} \to \{0,1\}$

$$F(x,y,z) = zf(x) \vee \bar{z}\, f(y) \quad \text{for } z \in \{0,1\},\ x,y \in \{0,1\}^n.$$

Making use of the obvious decomposition of F one gets a realization with cost at least 2C(f). But F can be realized as

$$F(x,y,z) = f(zx_1 \vee \bar{z}y_1, \ldots, zx_n \vee \bar{z}y_n),$$

hence $C(F) \leq C(f) + 3n$.

Although Theorem 3 shows that realizations which make use of a decomposition are not optimal in the general case, we can show the optimality of such realizations for a very restricted class of functions.

Theorem 4: If $f:\{0,1\}^n \to \{0,1\}$ has simple disjoint decomposition $f(x) = g(h(x_{i_1},\dots,x_{i_s}),x_{i_{s+1}},\dots,x_{i_n})$, $C(h) = s-1$ and f depends on all its arguments, then $C(f) = C(g) + s-1$, i.e. there is an optimal realization of f, which makes use of this decomposition.

Clearly $C(f) \leq C(g) + s-1$. It remains to show $C(f) \geq C(g) + s-1$. To prove this we first show

Lemma 5: If $f:\{0,1\}^n \to \{0,1\}$ has simple disjoint decomposition $f(x) = g(h(x_{i_1},x_{i_2}),x_{i_3},\dots,x_{i_n})$ and f depends on all its arguments, then $C(f) \geq C(g) + 1$.

Proof: For simplicity of notation we assume $i_j = j$ for $j \in [1:n]$.

For boolean functions or variables x and $a \in \{0,1\}$ let

$$x^a = \begin{cases} x & \text{if } a = 1 \\ \bar{x} & \text{if } a = 0 \end{cases} .$$

Because f depends on all arguments, so does h. There are two cases:

1) $h(x_1,x_2) = x_1^a \oplus x_2$ for some $a \in \{0,1\}$
2) $h(x_1,x_2) = (x_1^a \wedge x_2^b)^c$ for some $a,b,c \in \{0,1\}$

Let G be any optimal circuit for f. Transform G into another circuit G' as follows

in case 1: (i) Set the input x_2 to the constant 0

(ii) Set the input x_1 to x_1^a by negating it if a = 0, i.e. change the labels $\lambda(c)$ of all sons c of x_1 in an obvious way.

in case 2: (i) Set the input x_2 to the constant b

(ii) Set the input x_1 to $x_1^{a\equiv c}$ by negating it if $a\equiv c = 0$.

Because f depends on x_2, the input node x_2 has a successor node C in G. Transformation (i) makes one input of C constant, hence C can be eliminated.

Transformation (ii) does not change the cost of the circuit. Hence $C(G') \leq C(G) - 1$.

Let $q(y,x_3,\dots,x_n)$ be the function computed by G'. Then in case 1:

$$q(y,x_3,\dots,x_n) = f(y^a,0,x_3,\dots,x_n)$$
$$= g(h(y^a,0),\ x_3,\dots,\ x_n)$$
$$= g({y^a}^a \oplus 0,\ x_3,\dots,x_n)$$
$$= g(y,x_3,\dots,x_n)$$

In case 2

$$q(y,x_3,\dots,x_n) = f(y^{a\equiv c},\ b,\ x_3,\dots,x_n)$$
$$= g(h(y^{a\equiv c},b),x_3,\dots,x_n)$$
$$= g(((y^{a\equiv c})^a \wedge b^b)^c,\ x_3,\dots,x_n)$$
$$= g(y,x_3,\dots,x_n).$$

Hence in both cases $q = g$ and

$$C(g) \leq C(G') \leq C(G)-1 = C(f) - 1 \quad .$$

The theorem is now proven by induction on s. For $s = 2$ the theorem follows from Lemma 5. Suppose the theorem is proven for s-1. Because f (and hence h) depends on all its variables and $C(h) = s-1$, the graph of any optimal circuit G for h is a tree. There are at least two input nodes (leaves) x_i, x_j and a node c in H such that x_i and x_j are the fathers of c. Let $h'' = \lambda(c)$, i.e. h is the function computed by the node (gate) c.

For ease of notation let $i = 1$, $j = 2$. Transform G into a circuit G' by deleting x_1 and x_2 and making c a new input node. Let $h'(c,x_3,\ldots,x_s)$ be the function computed by G'. Then h' depends on all its arguments and $C(h') = s-2$.

Hence for $f':\{0,1\}^{n-1} \to \{0,1\}$,

$$f'(c_1,x_3,\ldots,x_n) = g(h'(c,x_3,\ldots,x_s),\ x_{s+1},\ldots,x_n)$$

the induction hypothesis implies $C(f') \geq C(g) + s-2$. But

$$f(x_1,\ldots,x_n) = f'(h''(x_1,x_2),x_3,\ldots,x_n) \quad ,$$

hence by lemma 5

$$C(f) \geq C(f') + 1 \geq C(g) + s-1.$$

6. <u>Monotone Circuits</u>:

<u>Definition</u>: for $a,b \in \{0,1\}^n$ define

$a \leq b$ if for all $i \in [1:n]$: $a_i \leq b_i$.

$f:\{0,1\}^n \to \{0,1\}^m$ is <u>monotone</u> if $a \leq b$ implies $f(a) \leq f(b)$.

A switching circuit is monotone if all nodes with indegree 2 are labelled with "∧" or "∨".

The monotone functions are exactly the functions computable by monotone circuits. For monotone f define the <u>monotone complexity</u> of f: $MC(f) = \min\{C(G) \mid G \text{ computes } f \text{ and } G \text{ is monotone}\}$.

The following theorem is due to M.J. Fischer [5]:

<u>Theorem 5</u>: for any monotone $f:\{0,1\}^n \to \{0,1\}^m$ and monotone $g:\{0,1\}^p \to \{0,1\}^q$ holds:

$$MC(f x g) = MC(f) + MC(y).$$

To prove Theorem 5 we use

<u>Fact 3 (Mehlhorn, Galil [8])</u>: Let G be a monotone circuit which computes a set of 1 - output functions $F = \{f_1,\ldots,f_k\}$. Let c be any gate in G and let $t_0, t_1, \ldots, t_n$ be the prime implicants of $res_G(c)$. If t_0 is not part+ of any prime implicant of any function in F, then c may be replaced by a node d with $res_G(d) = t_1 \vee \ldots \vee t_n$ and the new circuit still computes the same set of functions F.

<u>Proof of theorem 5</u>: Let G be a monotone circuit for fxg with input nodes $x_1,\ldots,x_n$, $y_1,\ldots,y_p$ and output nodes $u_1,\ldots,u_{m+q}$. We say a gate c is <u>shared</u> between f and g if there is a x_i and a y_j such that there is a path from x_i to c

---

+ $x_{i_1} \ldots x_{i_s}$ is <u>part</u> of $x_{j_1} \ldots x_{j_t}$ if

$$\{x_{i_1}, \ldots, x_{i_s}\} \subseteq \{x_{j_1}, \ldots, x_{j_t}\}.$$

and a path from y_j to c. To prove Theorem 5 it suffices to show that no optimal monotone circuit G for fxg contains a shared gate.

Suppose G contains one or more shared gates. Then there is at least one shared gate c in it such that no ancestor of c is a shared gate. Thus one input of c depends only on the x_i, the other only on the y_j. By Lemma 1 we can assume, that both inputs for c are non constant.

There are two cases to consider:

Case 1: c is labelled by $\wedge$.

Then $res_G(c)(x,y) = h_1(x) \wedge h_2(y)$ for some functions $h_1:\{0,1\}^n \to \{0,1\}$, $h_2:\{0,1\}^p \to \{0,1\}$. Because both h_1 and h_2 are non constant each prime implicant of h_1 is product of variables x_i and each prime implicant in h_2 is product of variables y_i. Hence each prime implicant of h contains both variables x_i and variables y_j.

It follows that no prime implicant of $res_G(c)$ is part of any prime implicant of any $res_G(u_i)$ because these depend or entirely on x or entirely on y. Hence by fact 3 $res_G(c)$ can be replaced by 0.

Case 2: If the label of c is $\vee$, $res_G(c)$ can be replaced by 1. This is proven as in Case 1, but one uses instead of fact 3 the dual+ of this fact.

+The dual of a theorem in boolean algebra is obtained by interchanging in the theorem and in all definitions preceding it $\wedge$ with $\vee$ and 0 with 1. That the dual of a theorem is also a theorem follows from the duality of the axioms of boolean algebra.

Corollary 3: For almost all n and all but a vanishing fraction of the binary nxn-matrices D holds: If G is a monotone circuit, which performs the boolean matrix product of D with an arbitrary nxn-matrix, then G has at least $n^3/(7\log n)$ gates.

Proof: By Theorem 5 and the direct proof of Corollary 2.

It has been shown, that any monotone circuit, which computes the boolean product of two arbitrary matrices requires $0(n^3)$ gates [8-10], but these proofs are more involved.

Theorem 6: There is a sequence of montone functions $f_n:\{0,1\}^n \to \{0,1\}^n$ such that:

(6.1) for almost all n: $MC(f_n) \geq n^{1.5}/(8\log n)$

(6.2) $C(f_n) \leq 0(n\log^2 n)$ for all n.

To prove Theorem 6 we prove

Lemma 6: There is a sequence of monotone functions $g_m:\{0,1\}^m \to \{0,1\}^m$, $m = 2^y$, $y \in \mathbb{N}$ such that,

(6.3) $MC(g_m) \geq m^{1.5}/(2\log m)$ for all large enough m

(6.4) $C(g_m) \leq 0(m\log^2 m)$ for all m.

This implies Theorem 6 in much the same way as Lemma 4 implies Theorem 1.

Proof of Lemma 6: For $y \in \mathbb{N}$ there are at least $2^{y\binom{y}{y/2}}$ + $\geq 2^{y2^{y/2}}$ different monotone functions $h:\{0,1\}^y \to \{0,1\}^y$. Hence by a counting argument for large enough y there is a monotone

---

+ $\binom{a}{b} = \frac{a!}{(a-b)!b!}$

$h:\{0,1\}^y \to \{0,1\}^y$ such that $MC(h) \geq C(h) \geq 2^{y/2}/2$.

Take such a function h and for $m = 2^y$ define: $g:\{0,1\}^m \to \{0,1\}^m$ by $g = h^{m/\log m}$,
By Theorem 5 and as $y = \log m$:

$$MC(g) \geq \frac{m}{\log m} \cdot \frac{2^{\log m/2}}{2} = \frac{m^{1.5}}{2\log m}$$

On the other hand, applying lemma 2 with $n = \log m$, $k = m/\log m$ yields

$$\begin{aligned} C(g) &\leq \gamma \max\{m\log^2 m, m\log^2(m/\log m)\} \\ &= \alpha\, m \log^2 m . \end{aligned}$$

This proves lemma 6 and the theorem.

7. Acknowledgements:

I wish to thank an unknown referee for bringing the alternative proofs of Corollary 2 and Theorem 3 to my attention and for simplifying the original proof of Lemma 5. For discussions I wish to thank Zvi Galil and Janos Simon. I am very thankful to Professor Albert Meyer for many helpful suggestions and comments.

8. References:

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman: The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[2] R.L. Ashenhurst: The Decomposition of Switching Functions, in Proc. Intem. Symp. Theory of Switching, Harvard University, 74-116, 1957.

[3] Curtis, H.A.: A New Approach to the Design of Switching Circuits, D. van Nostrand Comp., 1962.

[4] M.J. Fischer: Lectures on Network Complexity, University of Frankfort, Preprint, 1974.

[5] M.J. Fischer: private communication.

[6] M.J. Fischer and N. Pippenger: to appear.

[7] G. Hotz: Schaltkreistheorie, de Gruyter, 1974.

[8] K. Mehlhorn and Z. Galil, Monotone Switching Circuits and Boolean Matrix Product, to appear in Computing.

[9] M.S. Paterson: Complexity of Monotone Networks for Boolean Matrix Product, Technical Report, University of Warwick, 1974.

[10] V. Pratt: The Power of Negative Thinking in Multiplying Boolean Matrices, 6th ACM-STOC, 1974.

[11] C.E. Shannon, Bell Syst. Tech. J., No. 1, 59(1949).

[12] V. Strassen: Gaussian Elimination Is Not Optimal, Numer. Math. 13, 364-356 (1969).

[13] V. Strassen: Berechnung und Programm 1, Acta Informatica, 1, 320-335 (1972).

[14] V. Strassen: Berechnungen in partiallen Algebren endlichen Typs. Computing, 11, 181-196 (1973).