THE ART OF
COMPUTER PROGRAMMING

VOLUME 4       PRE-FASCICLE 0C

A DRAFT OF SECTION 7.1.2:
BOOLEAN EVALUATION

DONALD E. KNUTH  Stanford University

ADDISON–WESLEY

See also http://www-cs-faculty.stanford.edu/~knuth/sgb.html for information about The Stanford GraphBase, including downloadable software for dealing with the graphs used in many of the examples in Chapter 7.

See also http://www-cs-faculty.stanford.edu/~knuth/smixware.html for downloadable software to simulate the MMIX computer.

Copyright © 2006 by Addison–Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, except that the official electronic file may be used to print single copies for personal (not commercial) use.

Zeroth printing (revision 8), 30 January 2008
PREFACE

Your mind should break free of custom, furiously seizing the bit
and recklessly choosing its own path,
where you would fear to ascend by yourself.
— SENEC A, De Tranquillitate Animi (c. 50)

This booklet contains draft material that I'm circulating to experts in the
field, in hopes that they can help remove its most egregious errors before too
many other people see it. I am also, however, posting it on the Internet for
courageous and/or random readers who don't mind the risk of reading a few
pages that have not yet reached a very mature state. Beware: This material has
not yet been proofread as thoroughly as the manuscripts of Volumes 1, 2, and 3
were at the time of their first printings. And those carefully-checked volumes,
 alas, were subsequently found to contain thousands of mistakes.

Given this caveat, I hope that my errors this time will not be so numerous
and/or obtrusive that you will be discouraged from reading the material carefully.
I did try to make the text both interesting and authoritative, as far as it goes.
But the field is so vast, I cannot hope to have surrounded it enough to corral it
completely. Therefore I beg you to let me know about any deficiencies that you
discover.

To put the material in context, this pre-fascicle contains Section 7.1.2 of a
long, long chapter on combinatorial algorithms. Chapter 7 will eventually fill
at least three volumes (namely Volumes 4A, 4B, and 4C), assuming that I'm
able to remain healthy. It will begin with a short review of graph theory, with
emphasis on some highlights of significant graphs in the Stanford GraphBase,
from which I will be drawing many examples. Then comes Section 7.1: Zeros
and Ones, beginning with basic material in Section 7.1.1 (see pre-fascicle 6b).
Section 7.1.2, which you're about to read here, is concerned with the study of
efficient Boolean function evaluation. Section 7.1.3 will deal with tricks and
techniques of bitwise calculation; Section 7.1.4 will discuss data structures for
Boolean functions.

The next section, 7.2, is about generating all possibilities, and it begins
with Section 7.2.1: Generating Basic Combinatorial Patterns. Fascicles for this
section have already appeared on the Web and/or in print. Section 7.2.2 will
deal with backtracking in general. And so it will go on, if all goes well; an outline
of the entire Chapter 7 as currently envisaged appears on the tao cp webpage
that is cited on page ii.
The topic of Boolean functions and bit manipulation can of course be interpreted so broadly that it encompasses the entire subject of computer programming. My original title for Section 7.1 — “Bit Fiddling” — was much more modest; I decided, however, that those words were a bit too low-brow. The real goal of this fascicle is to focus on concepts that appear at the lowest levels, on which we can erect significant superstructures. And even these apparently lowly notions turn out to be surprisingly rich, with explicit ties to Sections 2.3.4.4, 4.3.1, 4.6.4, and 5.3.4 of the first three volumes. I strongly believe in building up a firm foundation, so I have discussed Boolean topics much more thoroughly than I will be able to do with material that is newer or less basic. After typing the manuscript I was astonished to discover that I had come up with 88 exercises, even though — believe it or not — I had to eliminate quite a lot of the interesting material that appears in my files.

My notes on combinatorial algorithms have been accumulating for more than forty years, so I fear that in several respects my knowledge is woefully behind the times. Please look, for example, at the exercises that I've classed as research problems (rated with difficulty level 46 or higher), namely exercises 21 and 24; I've also implicitly mentioned or posed additional unsolved questions in the answers to exercises 17, 40, 55, 61, 63, 70, and 80. Are those problems still open? Please let me know if you know of a solution to any of these intriguing questions. And of course if no solution is known today but you do make progress on any of them in the future, I hope you'll let me know.

I urgently need your help also with respect to some exercises that I made up as I was preparing this material. I certainly don't like to receive credit for things that have already been published by others, and most of these results are quite natural “fruits” that were just waiting to be “plucked.” Therefore please tell me if you know who deserves to be credited, with respect to the ideas found in exercises 11, 14, 16, 27, 29, 30, 34, 40, and 88, and/or the answer to exercise 38. Furthermore I've cited unpublished results of Frank Liang, Mike Paterson, and Rich Schroeppe!; do you know of any related publications?

The text presents an approach to synthesis based on so-called “footprints” of Boolean functions, which I haven't seen in the literature. Is this method new, or did I overlook some relevant papers?

I shall happily pay a finder's fee of $2.56 for each error in this draft when it is first reported to me, whether that error be typographical, technical, or historical. The same reward holds for items that I forgot to put in the index. And valuable suggestions for improvements to the text are worth 32¢ each. (Furthermore, if you find a better solution to an exercise, I'll actually reward you with immortal glory instead of mere money, by publishing your name in the eventual book:—)

Cross references to yet-unwritten material sometimes appear as ‘00'; this impossible value is a placeholder for the actual numbers to be supplied later.

Happy reading!

Stanford, California
14 March 2006

D. E. K.
Helpful hints. Readers of Section 7.1.2 should ideally have already read (or at least skimmed) Section 7.1.1. In particular, they should not be shocked or puzzled by notations such as

i) $\nu(x_1x_2\ldots x_n)$ or $\nu((x_1x_2\ldots x_n)_2)$ for the sum $x_1 + x_2 + \cdots + x_n$;

ii) $S_{k_1,k_2,\ldots,k_l}(x)$ for the symmetric function that is true when $\nu x = k_1$ or $k_2$ or $\ldots$ or $k_l$;

iii) $(x_1x_2\ldots x_{2k-1})$ for the threshold function $S_{\geq k}$ that equals the median value of $\{x_1,x_2,\ldots,x_{2k-1}\}$ (which is also the majority value).

I thank the Stanford University InfoLab and Sun Microsystems Laboratories for generously donating many hours of computer time on large, fast machines, thereby allowing me to investigate the Boolean functions of five variables.
By and large the minimization of switching components
outweighs all other engineering considerations
in designing economical logic circuits.


He must be a great calculator indeed who succeeds.
Simplify, simplify.

— HENRY D. THOREAU, Walden; or, Life in the Woods (1854)

7.1.2. Boolean Evaluation

Our next goal is to study the efficient evaluation of Boolean functions, much as
we studied the evaluation of polynomials in Section 4.6.4. One natural way to
investigate this topic is to consider chains of basic operations, analogous to the
polynomial chains discussed in that section.

A Boolean chain, for functions of $n$ variables ($x_1, \ldots, x_n$), is a sequence
$(x_{n+1}, \ldots, x_{n+r})$ with the property that each step combines two of the preceding
steps:

$$x_i = x_{j(i)} \circ_i x_{k(i)}, \quad \text{for } n + 1 \leq i \leq n + r,$$

where $1 \leq j(i) < i$ and $1 \leq k(i) < i$, and where $\circ_i$ is one of the sixteen binary
operators of Table 7.1.1–1. For example, when $n = 3$ the two chains

$$
\begin{align*}
x_4 &= x_1 \land x_2 & x_4 &= x_2 \oplus x_3 \\
x_5 &= \overline{x_1} \land x_3 & x_5 &= x_1 \land x_4 \\
x_6 &= x_4 \lor x_5 & x_6 &= x_3 \oplus x_5
\end{align*}
$$

both evaluate the “mux” or “if-then-else” function $x_6 = (x_1 ? x_2 : x_3)$, which
takes the value $x_2$ or $x_3$ depending on whether $x_1$ is 1 (true) or 0 (false).
(Notice that the left-hand example in (2) uses the simplified notation \( x_5 = \bar{x}_1 \land x_3 \) to specify the NOT-BUT operation, instead of the form \( x_5 = x_1 \bar{x}_3 \) that appears in Table 7.1.1. The main point is that, regardless of notation, every step of a Boolean chain is a Boolean combination of two prior results.)

Boolean chains correspond naturally to electronic circuits, with each step in the chain corresponding to a “gate” that has two inputs and one output. Electrical engineers traditionally represent the Boolean chains of (2) by circuit diagrams such as

![Circuit Diagram](https://example.com/circuit-diagram)

They need to design economical circuits that are subject to various technological constraints; for example, some gates might be more expensive than others, some outputs might need to be amplified if reused, the layout might need to be planar or nearly so, some paths might need to be short. But our chief concern in this book is software, not hardware, so we don’t have to worry about such things. For our purposes, all gates have equal cost, and all outputs can be reused as often as desired. (Jargonwise, our Boolean chains boil down to circuits in which all gates have fan-in 2 and unlimited fan-out.)

Furthermore we shall depict Boolean chains as binary trees such as

![Binary Tree](https://example.com/binary-tree)

instead of using circuit diagrams like (3). Such binary trees will have overlapping subtrees when intermediate steps of the chain are used more than once. Every internal node is labeled with a binary operator; external nodes are labeled with an integer \( k \), representing the variable \( x_k \). The label ‘\( \ominus \)’ in the left tree of (4) stands for the NOT-BUT operator, since \( \bar{x} \land y = [x < y] \); similarly, the BUT-NOT operator, \( x \land \bar{y} \), can be represented by the node label ‘\( \ominus \)’.

Several different Boolean chains might have the same tree diagram. For example, the left-hand tree of (4) also represents the chain

\[ x_4 = \bar{x}_1 \land x_3, \quad x_5 = x_1 \land x_2, \quad x_6 = x_5 \lor x_4. \]

Any topological sorting of the tree nodes yields an equivalent chain.

Given a Boolean function \( f \) of \( n \) variables, we often want to find a Boolean chain such that \( x_{n+r} = f(x_1, \ldots, x_n) \), where \( r \) is as small as possible. The *combinational complexity* \( C(f) \) of a function \( f \) is the length of the shortest chain that computes it. To save excess verbiage, we will simply call \( C(f) \) the “cost of \( f \)”.

The Boolean function in our examples above has cost 3, because one can show by exhaustive trials that it can’t be produced by any Boolean chain of length 2.

The DNF and CNF representations of \( f \), which we studied in Section 7.1.1, rarely tell us much about \( C(f) \), since substantially more efficient schemes of
calculation are usually possible. For example, in the discussion following 7.1.1-
(30) we found that the more-or-less random function of four variables whose
truth table is 1100 1001 0000 1111 has no DNF expression shorter than
\[(\bar{x}_1 \land \bar{x}_2 \land \bar{x}_3) \lor (\bar{x}_1 \land \bar{x}_3 \land \bar{x}_4) \lor (x_2 \land x_3 \land x_4) \lor (x_1 \land x_2).\] 
(5)
This formula corresponds to a Boolean chain of 10 steps. But that function can
also be expressed more cleverly as
\[((x_2 \land \bar{x}_4) \lor \bar{x}_3) \land \bar{x}_1) \lor x_2,\]
so its complexity is at most 4.

How can nonobvious formulas like (6) be discovered? We will see that a
computer can find the best chains for functions of four variables without doing an
enormous amount of work. Still, the results can be quite startling, even for people
who have had considerable experience with Boolean algebra. Typical examples
of this phenomenon can be seen in Fig. 9, which illustrates the four-variable
functions that are perhaps of greatest general interest, namely the functions
that are symmetric under all permutations of their variables.

Consider, for example, the function \(S_2(x_1, x_2, x_3, x_4)\), for which we have
\[
\begin{align*}
x_1 & = 0000 0000 1111 1111 \\
x_2 & = 0000 1111 0000 1111 \\
x_3 & = 0011 0011 0011 0011 \\
x_4 & = 0101 0101 0101 0101 \\
x_5 &= x_1 \oplus x_3 \quad 0011 0011 1100 1100 \\
x_6 &= x_1 \oplus x_2 \quad 0000 1111 1111 0000 \\
x_7 &= x_3 \oplus x_4 \quad 0110 0110 0110 0110 \\
x_8 &= x_5 \lor x_6 \quad 0011 1111 1111 1100 \\
x_9 &= x_6 \oplus x_7 \quad 0110 1001 1001 0110 \\
x_{10} &= x_9 \land \bar{x}_9 \quad 0001 0110 0110 1000 \\
\end{align*}
\] 
(7)
according to Fig. 9. Truth tables are shown here so that we can easily verify
each step of the calculation. Step \(x_8\) yields a function that is true whenever
\(x_1 \neq x_2\) or \(x_1 \neq x_3\); and \(x_8 = x_1 \oplus x_2 \oplus x_3 \oplus x_4\) is the parity function \((x_1 + x_2 + x_3 + x_4) \mod 2\).

Therefore the final result, \(x_{10}\), is true precisely when exactly two
of \(\{x_1, x_2, x_3, x_4\}\) are 1; these are the cases that satisfy \(x_8\) and have even parity.

Several of the other computational schemes of Fig. 9 can also be justified
intuitively. But some of the chains, like the one for \(S_{1,4}\), are quite amazing.

Notice that the intermediate result \(x_6\) is used twice in (7). In fact, no six-
step chain for the function \(S_2(x_1, x_2, x_3, x_4)\) is possible without making double
use of some intermediate subexpression; the shortest algebraic formulas for \(S_2\),
including nice symmetrical ones like
\[
((x_1 \land x_2) \lor (x_3 \land x_4)) \oplus ((x_1 \lor x_2) \land (x_3 \lor x_4)),
\] 
(8)
all have cost 7. But Fig. 9 shows that the other symmetric functions of four vari-
ables can all be evaluated optimally via "pure" binary trees, without overlapping
subtrees except at external nodes (which represent the variables).
In general, if \( f(x_1, \ldots, x_n) \) is any Boolean function, we say that its \textit{length} \( L(f) \) is the number of binary operators in the shortest formula for \( f \). Obviously \( L(f) \geq C(f) \); and we can easily verify that \( L(f) = C(f) \) whenever \( n \leq 3 \), by considering the fourteen basic types of 3-variable functions in 7.1.1-(95). But we have just seen that \( L(S_2) = 7 \) exceeds \( C(S_2) = 6 \) when \( n = 4 \), and in fact \( L(f) \) is almost always substantially larger than \( C(f) \) when \( n \) is large (see exercise 49).

The \textit{depth} \( D(f) \) of a Boolean function \( f \) is another important measure of its inherent complexity: We say that the depth of a Boolean chain is the length of the longest downward path in its tree diagram, and \( D(f) \) is the minimum achievable depth when all Boolean chains for \( f \) are considered. All of the chains illustrated in Fig. 9 have not only the minimum cost but also the minimum depth—except in the cases \( S_{2,3} \) and \( S_{1,2} \), where we cannot simultaneously achieve cost 6 and depth 3. The formula

\[
S_{2,3}(x_1, x_2, x_3, x_4) = ((x_1 \land x_2) \oplus (x_3 \land x_4)) \lor ((x_1 \lor x_2) \land (x_3 \oplus x_4))
\]

shows that \( D(S_{2,3}) = 3 \), and a similar formula works for \( S_{1,2} \).

**Optimum chains for \( n = 4 \).** Exhaustive computations for 4-variable functions are feasible because such functions have only \( 2^{16} = 65,536 \) possible truth tables. In fact we need only consider half of those truth tables, because the complement \( \bar{f} \) of any function \( f \) has the same cost, length, and depth as \( f \) itself.
Let's say that \( f(x_1, \ldots, x_n) \) is **normal** if \( f(0, \ldots, 0) = 0 \), and in general that
\[
f(x_1, \ldots, x_n) \oplus f(0, \ldots, 0)
\]
is the “normalization” of \( f \). Any Boolean chain can be normalized by normalizing each of its steps and by making appropriate changes to the operators; for if \((\hat{x}_1, \ldots, \hat{x}_{i-1})\) are the normalizations of \((x_1, \ldots, x_{i-1})\) and if \( x_i = \hat{x}_{j(i)} \circ \hat{x}_{k(i)} \) as in (1), then \( \hat{x}_i \) is clearly a binary function of \( \hat{x}_{j(i)} \) and \( \hat{x}_{k(i)} \). (Exercise 7 presents an example.) Therefore we can restrict consideration to normal Boolean chains, without loss of generality.

Notice that a Boolean chain is normal if and only if each of its binary operators \( \circ_i \) is normal. And there are only eight normal binary operators — three of which, namely \( \perp \), \( \perp \), and \( \ominus \), are trivial. So we can assume that all Boolean chains of interest are formed from the five operators \( \land, \lor, \land, \lor, \lor \), and \( \circ \), which are denoted respectively by \( \oplus \), \( \odot \), \( \ominus \), \( \ominus \), and \( \ominus \) in Fig. 9. Furthermore we can assume that \( j(i) < k(i) \) in each step.

There are \( 2^{15} = 32,768 \) normal functions of four variables, and we can compute their lengths without difficulty by systematically enumerating all functions of length 0, 1, 2, etc. Indeed, \( L(f) = r \) implies that \( f = g \circ h \) for some \( g \) and \( h \), where \( L(g) + L(h) = r - 1 \) and \( \circ \) is one of the five nontrivial normal operators; so we can proceed as follows:

**Algorithm L (Find normal lengths).** This algorithm determines \( L(f) \) for all normal truth tables \( 0 \leq f < 2^{2^{n-1}} \), by building lists of all nonzero normal functions of length \( r \) for \( r \geq 0 \).

**L.1.** [Initialize.] Let \( L(0) \leftarrow 0 \) and \( L(f) \leftarrow \infty \) for \( 1 \leq f < 2^{2^{n-1}} \). Then, for \( 1 \leq k \leq n \), set \( L(x_k) \leftarrow 0 \) and put \( x_k \) into list 0, where
\[
x_k = (2^{2^n} - 1)/(2^{2^n-k} + 1)
\]
is the truth table for \( x_k \). (See exercise 8.) Finally, set \( c \leftarrow 2^{2^n-1} - n - 1 \); \( c \) is the number of places where \( L(f) = \infty \).

**L.2.** [Loop on \( r \).] Do step L3 for \( r = 1, 2, \ldots \); eventually the algorithm will terminate when \( c \) becomes 0.

**L.3.** [Loop on \( j \) and \( k \).] Do step L4 for \( j = 0, 1, \ldots, \) and \( k = r - 1 - j \), while \( j \leq k \).

**L.4.** [Loop on \( g \) and \( h \).] Do step L5 for all \( g \) in list \( j \) and all \( h \) in list \( k \). (If \( j = k \), it suffices to restrict \( h \) to functions that follow \( g \) in list \( k \).)

**L.5.** [Loop on \( f \).] Do step L6 for \( f = g \land h \), \( f = \overline{g} \land h \), \( f = g \lor h \), \( f = \overline{g} \lor h \), and \( f = g \oplus h \). (Here \( g \land h \) denotes the bitwise AND of the integers \( g \) and \( h \); we are representing truth tables by integers in binary notation.)

**L.6.** [Is \( f \) new?] If \( L(f) = \infty \), set \( L(f) \leftarrow r \), \( c \leftarrow c - 1 \), and put \( f \) in list \( r \).
- Terminate the algorithm if \( c = 0 \). \( \blacksquare \)

Exercise 10 shows that a similar procedure will compute all depths \( D(f) \).

With a little more work, we can in fact modify Algorithm L so that it finds better upper bounds on \( C(f) \), by computing a heuristic bit vector \( \phi(f) \) called...
the “footprint” of \( f \). A normal Boolean chain can begin in only \( 5^\binom{n}{2} \) different ways, since the first step \( x_{n+1} \) must be either \( x_1 \land x_2 \) or \( \neg x_1 \land x_2 \) or \( x_1 \lor x_2 \) or \( x_1 \oplus x_2 \) or \( x_1 \land x_3 \) or \( \ldots \) or \( x_{n-1} \oplus x_n \). Suppose \( \phi(f) \) is a bit vector of length \( 5^\binom{n}{2} \) and \( U(f) \) is an upper bound on \( C(f) \), with the following property: Every 1 bit in \( \phi(f) \) corresponds to the first step of some Boolean chain that computes \( f \) in \( U(f) \) steps.

Such pairs \((U(f), \phi(f))\) can be computed by extending the basic strategy of Algorithm L. Initially we set \( U(f) \leftarrow 1 \) and we set \( \phi(f) \) to an appropriate vector 0...010...0, for all functions \( f \) of cost 1. Then, for \( r = 2, 3, \ldots \), we proceed to look for functions \( f = g \circ h \) where \( U(g) + U(h) = r - 1 \), as before, but with two changes: (1) If the footprints of \( g \) and \( h \) have at least one element in common, namely if \( \phi(g) \cap \phi(h) \neq 0 \), then we know that \( C(f) \leq r - 1 \), so we can decrease \( U(f) \) if it was \( \geq r \). (2) If the cost of \( g \circ h \) is equal to (but not less than) our current upper bound \( U(f) \), we can set \( \phi(f) \leftarrow \phi(f) \cap (\phi(g) \cap \phi(h)) \) if \( U(f) = r \), \( \phi(f) \leftarrow \phi(f) \cap (\phi(g) \cap \phi(h)) \) if \( U(f) = r - 1 \). Exercise 11 works out the details.

It turns out that this footprint heuristic is powerful enough to find chains of optimum cost \( U(f) = C(f) \) for all functions \( f \), when \( n = 4 \). Moreover, we'll see later that footprints also help us solve more complicated evaluation problems.

According to Table 7.1.1-5, the \( 2^{16} = 65,536 \) functions of four variables belong to only 222 distinct classes when we ignore minor differences due to permutation of variables and/or complementation of values. Algorithm L and its variants lead to the overall statistics shown in Table 1.

**Evaluation with minimum memory.** Suppose the Boolean values \( x_1, \ldots, x_n \) appear in \( n \) registers, and we want to evaluate a function by performing a sequence of operations having the form

\[
x_{j(i)} \leftarrow x_{j(i)} \circ_i x_{k(i)}, \quad \text{for } 1 \leq i \leq r,
\]

(12)

where \( 1 \leq j(i) \leq n \) and \( 1 \leq k(i) \leq n \) and \( \circ_i \) is a binary operator. At the end of the computation, the desired function value should appear in one of the registers.

When \( n = 3 \), for example, the four-step sequence

\[
\begin{align*}
x_1 & \leftarrow x_1 \oplus x_2 \\
x_3 & \leftarrow x_3 \land x_1 \\
x_2 & \leftarrow x_2 \land \neg x_1 \\
x_3 & \leftarrow x_3 \lor x_2
\end{align*}
\]

(13)
computes the median $\langle x_1, x_2, x_3 \rangle$ and puts it into the original position of $x_3$. (All eight possibilities for the register contents are shown here as truth tables, before and after each operation.)

In fact we can check the calculation by working with only one truth table at a time, instead of keeping track of all three, if we analyze the situation backwards. Let $f_l(x_1, \ldots, x_n)$ denote the function computed by steps $l$, $l+1$, \ldots, $r$ of the sequence, omitting the first $l-1$ steps; thus, in our example, $f_2(x_1, x_2, x_3)$ would be the result in $x_3$ after the three steps $x_3 \leftarrow x_3 \land x_1$, $x_2 \leftarrow x_2 \land \overline{x_1}$, $x_3 \leftarrow x_3 \lor x_2$. Then the function computed in register $x_3$ by all four steps is

$$f_1(x_1, x_2, x_3) = f_2(x_1 \lor x_2, x_2, x_3).$$

Similarly $f_2(x_1, x_2, x_3) = f_3(x_1, x_2, x_3 \land x_1)$, $f_3(x_1, x_2, x_3) = f_4(x_1, x_2 \land \overline{x_1}, x_3)$, $f_4(x_1, x_2, x_3) = f_5(x_1, x_2, x_2 \land x_2), \text{ and } f_5(x_1, x_2, x_3) = x_3$. We can therefore go back from $f_5$ to $f_4$ to \ldots to $f_1$ by operating on truth tables in an appropriate way.

For example, suppose $f(x_1, x_2, x_3)$ is a function whose truth table is

$$t = a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7;$$

then the truth table for $g(x_1, x_2, x_3) = f(x_1 \lor x_2, x_2, x_3)$ is

$$u = a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_3, \text{ obtained by replacing } a_7 \text{ by } a_3,$$

obtained by replacing $a_7$ by $a_3$, where

$$x = (x_1 x_2 x_3)_2 \text{ implies } x' = ((x_1 \lor x_2) x_2 x_3)_2.$$  

Similarly the truth table for, say, $h(x_1, x_2, x_3) = f(x_1, x_2, x_3 \land x_1)$ is

$$v = a_0 a_0 a_2 a_4 a_5 a_6 a_7.$$

And we can use bitwise operations to compute $u$ and $v$ from $t$:

$$u = t \oplus ((t \gg 4) \oplus (t \ll 4)) \& (00110011)_2, \quad (15)$$

$$v = t \oplus ((t \gg 1) \& (01010000)_2). \quad (16)$$

Let $C_m(f)$ be the length of a shortest minimum-memory computation for $f$. The backward-computation principle tells us that, if we know the truth tables of all functions $f$ with $C_m(f) < r$, we can readily find all the truth tables of functions with $C_m(f) = r$. Namely, we can restrict consideration to normal functions as before. Then, for all normal $g$ such that $C_m(g) = r - 1$, we can construct the $5n(n - 1)$ truth tables for

$$g(x_1, \ldots, x_{j-1}, x_j \circ x_k, x_{j+1}, \ldots, x_n) \quad (17)$$

and mark them with cost $r$ if they haven’t previously been marked. Exercise 14 shows that those truth tables can all be computed by performing simple bitwise operations on the truth table for $g$.

When $n = 4$, all but 13 of the 222 basic function types turn out to have $C_m(f) = C(f)$, so they can be evaluated in minimum memory without increasing the cost. In particular, all of the symmetric functions have this property — although that fact is not at all obvious from Fig. 9. Five classes of functions
have \( C(f) = 5 \) but \( C_m(f) = 6 \); eight classes have \( C(f) = 6 \) but \( C_m(f) = 7 \). The most interesting example of the latter type is probably the function \( (x_1 \lor x_2) \lor (x_3 \lor x_4) \lor (x_1 \land x_2 \land x_3 \land x_4) \), which has cost 6 because of the formula

\[
x_1 \lor (x_3 \lor x_4) \lor (x_2 \land (x_1 \lor (x_3 \land x_4))),
\]

(18)

but it has no minimum-memory chain of length less than 7. (See exercise 15.)

*Determining the minimum cost. The exact value of \( C(f) \) can be found by observing that all optimum Boolean chains \( (x_{n+1}, \ldots, x_{n+r}) \) for \( f \) obviously satisfy at least one of three conditions:

i) \( x_{n+r} = x_j \circ x_k \), where \( x_j \) and \( x_k \) use no common intermediate results;

ii) \( x_{n+1} = x_j \circ x_k \), where either \( x_j \) or \( x_k \) is not used in steps \( x_{n+2}, \ldots, x_{n+r} \);

iii) Neither of the above, even when the intermediate steps are renumbered.

In case (i) we have \( f = g \circ h \), where \( C(g) + C(h) = r - 1 \), and we can call this a “top-down” construction. In case (ii) we have \( f(x_1, \ldots, x_n) = g(x_1, \ldots, x_{j-1}, x_j \circ x_k, x_{j+1}, \ldots, x_n) \), where \( C(g) = r - 1 \); we call this construction “bottom-up.”

The best chains that recursively use only top-down constructions correspond to minimum formula length, \( L(f) \). The best chains that recursively use only bottom-up constructions correspond to minimum-memory calculations, of length \( C_m(f) \). We can do better yet, by mixing top-down constructions with bottom-up constructions; but we still won’t know that we’ve found \( C(f) \), because a special chain belonging to case (iii) might be shorter.

Fortunately such special chains are rare, because they must satisfy rather strong conditions, and they can be exhaustively listed when \( n \) and \( r \) aren’t too large. For example, exercise 19 proves that no special chains exist when \( r < n + 2 \); and when \( n = 4, r = 6 \), there are only 25 essentially different special chains that cannot obviously be shortened:

By systematically trying \( 5^r \) possibilities in every special chain, one for each way to assign a normal operator to the internal nodes of the tree, we will find at least
one function $f$ in every equivalence class for which the minimum cost $C(f)$ is achievable only in case (iii).

In fact, when $n = 4$ and $r = 6$, these $25 \cdot 5^6 = 390,625$ trials yield only one class of functions that can't be computed in 6 steps by any top-down-plus-bottom-up chain. The missing class, typified by the partially symmetric function $(x_1 x_2 x_3) \lor x_4 \oplus (x_1 \land x_2 \land x_3)$, can be reached in six steps by appropriately specializing any of the first five chains illustrated above; for example, one way is

$$
\begin{align*}
x_5 &= x_1 \land x_2, \\
x_6 &= x_1 \lor x_2, \\
x_7 &= x_3 \oplus x_0, \\
x_8 &= x_4 \land \bar{x}_5, \\
x_9 &= x_6 \land x_7, \\
x_{10} &= x_8 \lor x_9,
\end{align*}
$$

(19)
corresponding to the first special chain. Since all other functions have $L(f) \leq 7$, these trial calculations have established the true minimum cost in all cases.

**Historical notes:** The first concerted attempts to evaluate all Boolean functions $f(w, x, y, z)$ optimally were reported in *Annals of the Computation Laboratory of Harvard University* 27 (1951), where Howard Aiken's staff presented heuristic methods and extensive tables of the best switching circuits they were able to construct. Their cost measure $V(f)$ was different from the cost $C(f)$ that we've been considering, because it was based on "control grids" of vacuum tubes: They had three kinds of gates, NOR, OR, and NAND, each of which could take $k$ inputs with cost $k$. Every input to such a gate could be either a variable, or the complement of a variable, or the result of a previous gate. Furthermore the function being evaluated was represented at the top level as an AND of any number of gates, with no additional cost.

With these cost criteria, a function might not have the same cost as its complement, because AND gates were possible only at the top level. One could evaluate $x \land y$ as NOR($\bar{x}, \bar{y}$), with cost 2; but the cost of $\bar{x} \lor (\bar{y} \land z)$ = NAND($x, OR(y, z)$) was 4 while its complement $x \land (y \lor z) = AND(NOR(x), OR(y, z))$ cost only 3. Therefore the Harvard researchers needed to consider 402 essentially different classes of 4-variable functions instead of 222 (see the answer to exercise 7.1.1-125). Of course in those days they were working by hand. They found $V(f) < 20$ in all cases, except for the 64 functions equivalent to $S_{0,1}(w, x, y, z) \lor (S_{2}(w, x, y) \land z)$, which they evaluated with 20 control grids as follows:

$$
g_1 = \text{NOR}(\bar{y}, \bar{z}), \\
g_2 = \text{NAND}(\bar{y}, z), \\
g_3 = \text{NOR}(w, x), \\
f = \text{AND}(\text{NAND}(g_1, g_2), \text{NAND}(g_3, \text{NOR}(\bar{y}, \bar{z}))), \\
\text{NOR}(\text{NOR}(g_3, z), \text{NOR}(g_1, g_2, g_3)).
$$

(20)

The first computer program to find provably optimum circuits was written by Leo Hellerman [*IEEE Transactions EC-12* (1963), 198-223], who determined the fewest NOR gates needed to evaluate any given function $f(x, y, z)$. He required every input of every gate to be either an uncomplemented variable or the output of a previous gate; fan-in and fan-out were limited to at most 3. When two circuits had the same gate count, he preferred the one with smallest sum-of-inputs. For example, he computed $\bar{z} = \text{NOR}(x)$ with cost 1; $x \lor y \lor z = \text{NOR}(\text{NOR}(x, y, z))$ with cost 2; $(xy) = \text{NOR}(\text{NOR}(x, y), \text{NOR}(x, z), \text{NOR}(y, z))$.
with cost 4; \( S_1(x, y, z) = \text{NOR}(\text{NOR}(x, y, z), (xyz)) \) with cost 6; etc. Since he limited the fan-out to 3, he found that every function of three variables could be evaluated with cost 7 or less, except for the parity function \( x \oplus y \oplus z = (x \equiv y) \equiv z \), where \( x \equiv y \) has cost 4 because it is \( \text{NOR}(\text{NOR}(x, \text{NOR}(x, y)), \text{NOR}(y, \text{NOR}(x, y))) \).

Electrical engineers continued to explore other cost criteria; but four-variable functions seemed out of reach until 1977, when Frank M. Liang established the values of \( C(f) \) shown in Table 1. Liang’s unpublished derivation was based on a study of all chains that cannot be reduced by the bottom-up construction.

**The case \( n = 5 \).** There are 616,126 classes of essentially different functions \( f(x_1, x_2, x_3, x_4, x_5) \), according to Table 7.1.1–5. Computers are now fast enough that this number is no longer frightening; so the author decided while writing this section to investigate \( C(f) \) for all Boolean functions of five variables. Thanks to a bit of good luck, complete results could indeed be obtained, leading to the statistics shown in Table 2.

For this calculation Algorithm L and its variants were modified to deal with class representatives, instead of with the entire set of \( 2^{31} \) normal truth tables. The method of exercise 7.2.1.2–20 made it easy to generate all functions of a class, given any one of them, resulting in a thousand-fold speedup. The bottom-up method was enhanced slightly, allowing it to deduce for example that \( f(x_1 \land x_2, x_1 \lor x_2, x_3, x_4, x_5) \) has cost \( \leq r \) if \( C(f) = r - 2 \). After all classes of cost 10 had been found, the top-down and bottom-up methods were able to find chains of length \( \leq 11 \) for all but seven classes of functions. Then the time-consuming part of the computation began, in which approximately 53 million special chains with \( n = 5 \) and \( r = 11 \) were generated; every such chain led to \( 5^{11} = 48,828,125 \) functions, some of which would hopefully fall into the seven remaining mystery classes. But only six of those classes were found to have 11-step solutions. The lone survivor, whose truth table is \( 169\text{ae}4\text{43} \) in hexadecimal notation, is the unique class for which \( C(f) = 12 \), and it also has \( L(f) = 12 \).

The resulting constructions of symmetric functions are shown in Fig. 10. Some of them are astonishingly beautiful; some of them are beautifully simple;
and others are simply astonishing. (Look, for example, at the 8-step computation of \( S_{2,3}(x_1, x_2, x_3, x_4, x_5) \), or the elegant formula for \( S_{2,3,4} \), or the nonmonotonic chains for \( S_{3,5} \) and \( S_{3,4,5} \).) Incidentally, Table 2 shows that all 5-variable functions have depth \( \leq 4 \), but no attempt to minimize depth has been made in Fig. 10.

It turns out that all of these symmetric functions can be evaluated in minimum memory without increasing the cost. But no simple proof of that fact is known.
Multiple outputs. We often want to evaluate several different Boolean functions $f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)$ at the same input values $x_1, \ldots, x_n$; in other words, we often want to evaluate a multibit function $y = f(x)$, where $y = f_1 \ldots f_m$ is a binary vector of length $m$ and $x = x_1 \ldots x_n$ is a binary vector of length $n$. With luck, much of the work involved in the computation of one component value $f_j(x_1, \ldots, x_n)$ can be shared with the operations that are needed to evaluate the other component values $f_k(x_1, \ldots, x_n)$.

Let $C(f) = C(f_1 \ldots f_m)$ be the length of a shortest Boolean chain that computes all of the nontrivial functions $f_j$. More precisely, the chain $(x_{n+1}, \ldots, x_{n+r})$ should have the property that, for $1 \leq j \leq m$, either $f_j(x_1, \ldots, x_n) = x_{j(j)}$ or $f_j(x_1, \ldots, x_n) = x_{l(j)}$, for some $l(j)$ with $0 \leq l(j) \leq n + r$, where $x_0 = 0$. Clearly $C(f) \leq C(f_1) + \cdots + C(f_m)$, but we might be able to do much better.

For example, suppose we want to compute the functions $z_1$ and $z_0$ defined by

$$(z_1 z_0)_2 = x_1 + x_2 + x_3,$$  \hspace{1cm} (21)

the two-bit binary sum of three Boolean variables. We have

$$z_1 = (x_1 x_2 x_3), \quad z_0 = x_1 + x_2 + x_3,$$  \hspace{1cm} (22)

so the individual costs are $C(z_1) = 4$ and $C(z_0) = 2$. But it’s easy to see that the combined cost $C(z_1 z_0)$ is at most 5, because $x_1 \oplus x_2$ is a suitable first step in the evaluation of each bit $z_j$:

$$x_4 = x_1 + x_2, \quad x_5 = x_3 \oplus x_4;$$

$$x_6 = x_3 \land x_4, \quad x_7 = x_1 \land x_2, \quad z_1 = x_8 = x_6 \lor x_7.$$  \hspace{1cm} (23)

Furthermore, exhaustive calculations show that $C(z_1 z_0) > 4$; hence $C(z_1 z_0) = 5$.

Electrical engineers traditionally call a circuit for (21) a full adder, because $n$ such building blocks can be hooked together to add two $n$-bit numbers. The special case of (22) in which $x_3 = 0$ is also important, although it boils down simply to

$$z_1 = x_1 \land x_2 \quad \text{and} \quad z_0 = x_1 + x_2$$  \hspace{1cm} (24)

and has complexity 2; engineers call it a “half adder” in spite of the fact that the cost of a full adder exceeds the cost of two half adders.

The general problem of radix-2 addition

$$\begin{align*}
(x_{n-1} \ldots x_1 x_0)_2 \\
(y_{n-1} \ldots y_1 y_0)_2 \\
(z_{n-1} \ldots z_1 z_0)_2
\end{align*}$$

is to compute $n + 1$ Boolean outputs $z_n \ldots z_1 z_0$ from the $2n$ Boolean inputs $x_{n-1} \ldots x_0 y_{n-1} \ldots y_0$; and it is readily solved by the formulas

$$c_{j+1} = (x_j y_j c_j), \quad z_j = x_j \oplus y_j \oplus c_j, \quad \text{for } 0 \leq j < n,$$  \hspace{1cm} (26)

where the $c_j$ are “carry bits” and we have $c_0 = 0$, $z_n = c_n$. Therefore we can use a half adder to compute $c_1$ and $z_1$, followed by $n - 1$ full adders to compute the other $c$’s and $z$’s, accumulating a total cost of $5n - 3$. And in fact N. P. Red’kin [Problemy Kibernetiki 38 (1981), 181–216] has proved that $5n - 3$ steps
are actually necessary, by constructing an elaborate 35-page proof by induction, which concludes with Case 2.2.3.1.2.3.2.4.3(1). But the depth of this circuit, \(2n - 1\), is far too large for practical parallel computation, so a great deal of effort has gone into the task of devising circuits for addition that have depth \(O(\log n)\) as well as reasonable cost. (See exercises 41–44.)

Now let’s extend (21) and try to compute a general “sideways sum”

\[
(z_{[\log n]} \cdots z_1 z_0)_2 = x_1 + x_2 + \cdots + x_n. 
\]

(27)

If \(n = 2k + 1\), we can use \(k\) full adders to reduce the sum to \((x_1 + \cdots + x_n) \mod 2\) plus \(k\) bits of weight 2, because each full adder decreases the number of weight-1 bits by 2. For example, if \(n = 9\) and \(k = 4\) the computation is

\[
x_{10} = x_1 \oplus x_2 \oplus x_3, \quad x_{11} = x_4 \oplus x_5 \oplus x_6, \quad x_{12} = x_7 \oplus x_8 \oplus x_9, \quad x_{13} = x_{10} \oplus x_{11} \oplus x_{12},
\]

\(y_1 = (x_1 x_2 x_3), \quad y_2 = (x_4 x_5 x_6), \quad y_3 = (x_7 x_8 x_9), \quad y_4 = (x_{10} x_{11} x_{12}),\)

and we have \(x_1 + \cdots + x_9 = x_{13} + 2(y_1 + y_2 + y_3 + y_4)\). If \(n = 2k\) is even, a similar reduction applies but with a half adder at the end. The bits of weight 2 can then be summed in the same way; so we obtain the recurrence

\[
s(n) = 5\lceil n/2 \rceil - 3 \lfloor n/2 \rfloor + s\lceil n/2 \rceil, \quad s(0) = 0,
\]

(28)
for the total number of gates needed to compute \(z_{[\log n]} \cdots z_1 z_0\). (A closed formula for \(s(n)\) appears in exercise 30.) We have \(s(n) < 5n\), and the first values

\[
n = 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18 \ 19 \ 20\]

\[
s(n) = 0 \ 2 \ 5 \ 9 \ 12 \ 17 \ 20 \ 26 \ 29 \ 34 \ 37 \ 44 \ 47 \ 52 \ 55 \ 63 \ 66 \ 71 \ 74 \ 81
\]

show that the method is quite efficient even for small \(n\). For example, when \(n = 5\) it produces

\[
S_{4,5} = z_2 = z_1 = S_{2,3} = z_0 = S_{1,3,5}
\]

which computes three different symmetric functions \(z_2 = S_{4,5}(x_1, \ldots, x_5), \ z_1 = S_{2,3}(x_1, \ldots, x_3), \ z_0 = S_{1,3,5}(x_1, \ldots, x_5)\) in just 12 steps. The 10-step computation of \(S_{4,5}\) is optimum, according to Fig. 10; of course the 4-step computation of \(S_{1,3,5}\) is also optimum. Furthermore, although \(C(S_{2,3}) = 8\), the function \(S_{2,3}\) is computed here in a clever 10-step way that shares all but one gate with \(S_{4,5}\).

Notice that we can now compute any symmetric function efficiently, because every symmetric function of \(\{x_1, \ldots, x_n\}\) is a Boolean function of \(z_{[\log n]} \cdots z_1 z_0\). We know, for example, that any Boolean function of four variables has complexity \(\leq 7\); therefore any symmetric function \(S_{k_1, \ldots, k_4}(x_1, \ldots, x_{15})\) costs at most \(s(15) + 7 = 62\). Surprise: The symmetric functions of \(n\) variables were among the hardest of all to evaluate; when \(n\) was small, but they’re among the easiest when \(n \geq 10\).
7.1.2 BOOLEAN EVALUATION

We can also compute sets of symmetric functions efficiently. If we want, say, to evaluate all \( n + 1 \) symmetric functions \( S_k(x_1, \ldots, x_n) \) for \( 0 \leq k \leq n \) with a single Boolean chain, we simply need to evaluate the first \( n+1 \) minterms of \( z_0, z_1, \ldots, z_{\lceil \log n \rceil} \). For example, when \( n = 5 \) the minterms that give us all functions \( S_k \) are respectively \( S_0 = \bar{z}_0 \land \bar{z}_1 \land \bar{z}_2 \), \( S_1 = \bar{z}_0 \land \bar{z}_1 \land \bar{z}_2 \), \ldots, \( S_5 = z_0 \land \bar{z}_1 \land z_2 \).

How hard is it to compute all \( 2^n \) minterms of \( n \) variables? Electrical engineers call this function an \( n \)-to-\( 2^n \) binary decoder, because it converts \( n \) bits \( x_1 \ldots x_n \) into a sequence of \( 2^n \) bits \( d_0 d_1 \ldots d_{2^n-1} \), exactly one of which is 1. The principle of “divide and conquer” suggests that we first evaluate all minterms on the first \( \lceil n/2 \rceil \) variables, as well as all minterms on the last \( \lceil n/2 \rceil \); then \( 2^n \) AND gates will finish the job. The cost of this method is \( t(n) \), where

\[
t(0) = t(1) = 0; \quad t(n) = 2^n + t(\lfloor n/2 \rfloor) + t(n/2) \quad \text{for } n \geq 2. \tag{30}
\]

So \( t(n) = 2^n + O(2^{n/2}) \); there’s roughly one gate per minterm. (See exercise 32.)

Functions with multiple outputs often help us build larger functions with single outputs. For example, we’ve seen that the sideways adder (27) allows us to compute symmetric functions; and an \( n \)-to-\( 2^n \) decoder also has many applications, in spite of the fact that \( 2^n \) can be huge when \( n \) is large. A case in point is the \( 2^m \)-way multiplexer \( M_m(x_1, \ldots, x_m; y_0, y_1, \ldots, y_{2^m-1}) \), also known as the \( m \)-bit storage access function, which has \( n = m + 2^m \) inputs and takes the value \( y_k \) when \( (x_1 \ldots x_m)_2 = k \). By definition we have

\[
M_m(x_1, \ldots, x_m; y_0, y_1, \ldots, y_{2^m-1}) = \bigvee_{k=0}^{2^m-1} (d_k \land y_k), \tag{31}
\]

where \( d_k \) is the \( k \)th output of an \( m \)-to-\( 2^m \) binary decoder; thus, by (30), we can evaluate \( M_m \) with \( 2^n + (2^m-1) + t(m) = 3n + O(\sqrt{n}) \) gates. But exercise 39 shows that we can actually reduce the cost to only \( 2n + O(\sqrt{n}) \). (See also exercise 79.)

Asymptotic facts. When the number of variables is small, our exhaustive-search methods have turned up lots of cases where Boolean functions can be evaluated with stunning efficiency. So it’s natural to expect that, when more variables are present, even more opportunities for ingenious evaluations will arise. But the truth is exactly the opposite, at least from a statistical standpoint:

**Theorem S.** The cost of almost every Boolean function \( f(x_1, \ldots, x_n) \) exceeds \( 2^n/n \). More precisely, if \( c(n, r) \) Boolean functions have complexity \( \leq r \), we have

\[
(r - 1)!c(n, r) \leq 2^{2r+1}(n + r - 1)^{2r}. \tag{32}
\]

**Proof.** If a function can be computed in \( r-1 \) steps, it is also computable by an \( r \)-step chain. (This statement is obvious when \( r = 1 \); otherwise we can let \( x_{n+r} = x_{n+r-1} \land x_{n+r-1} \).) We will show that there aren’t very many \( r \)-step chains, hence we can’t compute very many different functions with cost \( \leq r \).

Let \( \pi \) be a permutation of \( \{1, \ldots, n + r\} \) that takes \( 1 \mapsto 1, \ldots, n \mapsto n \), and \( n + r \mapsto n + r \); there are \( (r-1)! \) such permutations. Suppose \( (x_{n+1}, \ldots, x_{n+r}) \) is a
Boolean chain in which each of the intermediate steps $x_{n+1}, \ldots, x_{n+r-1}$ is used in at least one subsequent step. Then the permuted chains defined by the rule

$$x_{i} = x_{j(i)}^\prime \circ_i x_{k(i)}^\prime = x_{j(i)\pi^{-1}} \circ_{i\pi} x_{k(i)\pi^{-1}}, \quad \text{for } n < i \leq n + r,$$

(33) are distinct for different $\pi$. (If $\pi$ takes $a \mapsto b$, we write $b = a\pi$ and $a = b\pi^{-1}$.) For example, if $\pi$ takes $5 \mapsto 6 \mapsto 7 \mapsto 8 \mapsto 9 \mapsto 5$, the chain (7) becomes

<table>
<thead>
<tr>
<th>Original</th>
<th>Permutted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_5 = x_1 \oplus x_3,$</td>
<td>$x_5 = x_1 \oplus x_2,$</td>
</tr>
<tr>
<td>$x_6 = x_1 \oplus x_2,$</td>
<td>$x_6 = x_3 \oplus x_4,$</td>
</tr>
<tr>
<td>$x_7 = x_3 \oplus x_4,$</td>
<td>$x_7 = x_9 \lor x_5,$</td>
</tr>
<tr>
<td>$x_8 = x_5 \lor x_6,$</td>
<td>$x_8 = x_5 \oplus x_6,$</td>
</tr>
<tr>
<td>$x_9 = x_5 \oplus x_7,$</td>
<td>$x_9 = x_1 \oplus x_3,$</td>
</tr>
<tr>
<td>$x_{10} = x_9 \land x_3;$</td>
<td>$x_{10} = x_7 \land x_8.$</td>
</tr>
</tbody>
</table>

(34)

Notice that we might have $j^\prime(i) \geq k^\prime(i)$ or $j^\prime(i) > i$ or $k^\prime(i) > i$, contrary to our usual rules. But the permuted chain computes the same function $x_{n+r}$ as before, and it doesn’t have any cycles by which an entry is defined indirectly in terms of itself, because the permuted $x_i$ is the original $x_{i\pi}$.

We can restrict consideration to normal Boolean chains, as remarked earlier. So the $c(n, r)/2$ normal Boolean functions of cost $\leq r$ lead to $(r - 1)!c(n, r)/2$ different permuted chains, where the operator $\circ_i$ in each step is either $\land$, $\lor$, $\ominus$, or $\oplus$. And there are at most $4^r(n+r-1)^{2r}$ such chains, because there are four choices for $\circ_i$ and $n+r-1$ choices for each of $j(i)$ and $k(i)$, for $n < i \leq n + r$. Equation (32) follows; and we obtain the opening statement of the theorem by setting $r = \lceil 2^n/n \rceil$. (See exercise 46.)

On the other hand, there’s also good news for infinity-minded people: We can actually evaluate every Boolean function of $n$ variables with only slightly more than $2^n/n$ steps of computation, even if we avoid $\ominus$ and $\oplus$, using a technique devised by C. E. Shannon and improved by V. B. Lupanov [Bell System Tech. J. 28 (1949), 59–98, Theorem 6; Izvestiia VUZov, Radiofizika 1 (1958), 120–140]. In fact, the Shannon–Lupanov approach leads to useful results even when $n$ is small, so let’s get acquainted with it by studying a small example. Consider

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = [(x_1 x_2 x_3 x_4 x_5 x_6)_{2} \text{ is prime}],$$

(35)
a function that identifies all 6-bit prime numbers. Its truth table has $2^6 = 64$ bits, and we can work with it conveniently by using a $4 \times 16$ array to look at those bits instead of conforming ourselves to one dimension:

$$
\begin{array}{cccccccccccccccc}
& x_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
& x_4 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
& x_5 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
& x_6 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
$$

$$
\begin{array}{cccccccccccccccc}
& x_1 x_2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
& x_1 x_2 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
& x_1 x_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
& x_1 x_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
$$

(36)

Group 1

| Group 2 |

$$
\begin{array}{cccccccccccccccc}
& x_1 x_2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
& x_1 x_2 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
& x_1 x_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
& x_1 x_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
$$

Normal upper bound++
Shannon
Lupanov
Prime numbers
Truth table
The rows have been divided into two groups of two rows each; and each group of rows has 16 columns, which are of four basic types, namely $0$, $1$, $\frac{1}{2}$, or $\frac{3}{4}$. Thus we see that the function can be expressed as

$$
f(x_1, \ldots, x_6) = \big((x_1, x_2 \in \{00\}) \land (x_3, x_4, x_5 x_6 \in \{0010, 0101, 1011\})\big) \\
\lor \big((x_1, x_2 \in \{01\}) \land (x_3, x_4, x_5 x_6 \in \{0001, 1111\})\big) \\
\lor \big((x_1, x_2 \in \{00, 01\}) \land (x_3, x_4, x_5 x_6 \in \{0011, 0111, 1101\})\big) \\
\lor \big((x_1, x_2 \in \{10\}) \land (x_3, x_4, x_5 x_6 \in \{1001, 1111\})\big) \\
\lor \big((x_1, x_2 \in \{11\}) \land (x_3, x_4, x_5 x_6 \in \{1100, 1111\})\big) \\
\lor \big((x_1, x_2 \in \{10, 11\}) \land (x_3, x_4, x_5 x_6 \in \{1010, 1011\})\big). \quad (37)
$$

(The first line corresponds to group 1, type $\frac{1}{2}$, then comes group 1, type $\frac{3}{4}$, etc.; the last line corresponds to group 2 and type $\frac{3}{2}$.) A function like $[x_3, x_4, x_5, x_6 \in \{0010, 0101, 1011\}]$ is the OR of three minterms of $\{x_3, x_4, x_5, x_6\}$.

In general we can view the truth table as a $2^k \times 2^{n-k}$ array, with $l$ groups of rows having either $[2^k/l]$ or $[2^k/l]$ rows in each group. A group of size $m$ will have rows of $2^m$ basic types. We form a conjunction $(g_i(x_1, \ldots, x_k) \land h_i(x_{k+1}, \ldots, x_n))$ for each group $i$ and each nonzero type $t$, where $g_i$ is the OR of all minterms of $\{x_1, \ldots, x_k\}$ for the rows of the group where $t$ has a 1, while $h_i$ is the OR of all minterms of $\{x_{k+1}, \ldots, x_n\}$ for the columns having type $t$ in group $i$. The OR of all these conjunctions $(g_i \land h_i)$ gives $f(x_1, \ldots, x_n)$.

Once we’ve chosen the parameters $k$ and $l$, with $1 \leq k \leq n-l$ and $1 \leq l \leq 2^k$, the computation starts by computing all the minterms of $\{x_1, \ldots, x_k\}$ and all the minterms of $\{x_{k+1}, \ldots, x_n\}$, in $t(k) + t(n-k)$ steps (see (30)). Then, for $1 \leq i \leq l$, we let group $i$ consist of rows for which $(x_1, \ldots, x_k)$ such that $(i-1)2^k/l \leq (x_1, \ldots, x_k) < i2^k/l$; it contains $m_i = \lceil i2^k/l \rceil - \lfloor (i-1)2^k/l \rfloor$ rows. We form all functions $g_i$ for $t \in S_i$, the family of $2^{m_1} - 1$ nonempty subsets of those rows; $2^{m_i} - m_i - 1$ ORs of previously computed minterms will accomplish that task. We also form all functions $h_i$ representing columns of nonzero type $t$; for this purpose we’ll need at most $2^{n-k}$ OR operations in each group $i$, since we can OR each minterm into the $h$ function of the appropriate type $t$. Finally we compute $t = \bigvee_{i=1}^{l} \bigvee_{t \in S_i} (g_i \land h_i)$; each AND operation is compensated by an unnecessary first OR into $h_i$. So the total cost is at most

$$
t(k) + t(n-k) + (l-1) + \sum_{i=1}^{l} \left(\frac{2^{m_1} - m_1 - 1}{2^{m_1}} + 2^{n-k} + 2^{m_i} - 2\right); \quad (38)
$$

we want to choose $k$ and $l$ so that this upper bound is minimized. Exercise 52 discusses the best choice when $n$ is small. And when $n$ is large, a good choice yields a provably near-optimum chain, at least for most functions:

**Theorem L.** Let $C(n)$ denote the cost of the most expensive Boolean functions of $n$ variables. Then as $n \to \infty$ we have

$$
C(n) \geq \frac{2^n}{n} \left(1 + \frac{\log n}{n} + O\left(\frac{1}{n}\right)\right); \quad (39)
$$

$$
C(n) \leq \frac{2^n}{n} \left(1 + \frac{3\log n}{n} + O\left(\frac{1}{n}\right)\right). \quad (40)
$$
7.1.2

**Proof.** Exercise 48 shows that the lower bound (39) is a consequence of Theorem S. For the upper bound, we set \( k = \lfloor 2 \log n \rfloor \) and \( l = \lfloor 2^k/(n - 3 \log n) \rfloor \) in Lupanov’s method; see exercise 53.  

**Synthesizing a good chain.** Formula (37) isn’t the best way to implement a 6-bit prime detector, but it does suggest a decent strategy. For example, we needn’t let variables \( x_1 \) and \( x_2 \) govern the rows: Exercise 51 shows that a better chain results if the rows are based on \( x_5 \) \( x_6 \) while the columns come from \( x_1 x_2 x_3 x_4 \), and in general there are many ways to partition a truth table by playing \( k \) of the variables against the other \( n - k \).

Furthermore, we can improve on (37) by using our complete knowledge of all 4-variable functions; there’s no need to evaluate a function like \( x_3 x_4 x_5 x_6 \in \{0010, 0101, 1011\} \) by first computing the minterms of \( \{x_3, x_4, x_5, x_6\} \), if we know the best way to evaluate every such function from scratch. On the other hand, we do need to evaluate several 4-variable functions simultaneously, so the minterm approach might not be such a bad idea after all. Can we really improve on it?

Let’s try to find a good way to synthesize a Boolean chain that computes a given set of 4-variable functions. The six functions of \( x_5 x_4 x_3 x_2 x_1 \) in (37) are rather tame (see exercise 54), so we’ll learn more by considering a more interesting example chosen from everyday life.

A **seven-segment display** is a now-ubiquitous way to represent a 4-bit number \( (x_1 x_2 x_3 x_4)_2 \) in terms of seven cleverly positioned segments that are either visible or invisible. The segments are traditionally named \((a, b, c, d, e, f, g)\) as shown; we get a ‘0’ by turning on segments \((a, b, c, d, e, f)\), but a ‘1’ uses only segments \((b, c)\). (Incidentally, the idea for such displays was invented by F. W. Wood, U.S. Patent 974943 (1910), although Wood’s original design used eight segments because he thought that a ‘4’ requires a diagonal stroke.) Seven-segment displays usually support only the decimal digits ‘0’, ‘1’, \ldots, ‘9’; but of course a computer scientist’s wrist digital watch should display also hexadecimal digits. So we shall design seven-segment logic that displays the sixteen digits

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
8 & 9 & A & B & C & D & E & F
\end{array}
\]

(41)

when given the respective inputs \( x_1 x_2 x_3 x_4 = 0000, 0001, 0010, \ldots, 1111 \).

In other words, we want to evaluate seven Boolean functions whose truth tables are respectively

\[
\begin{align*}
a &= 1011\ 0111\ 1110\ 0011, \\
b &= 1111\ 1001\ 1110\ 0100, \\
c &= 1101\ 1111\ 1111\ 0100, \\
d &= 1011\ 0110\ 1101\ 1110, \\
e &= 1010\ 0010\ 1011\ 1111, \\
f &= 1000\ 1111\ 1111\ 0011, \\
g &= 0011\ 1110\ 1111\ 1111.
\end{align*}
\] (42)
If we simply wanted to evaluate each function separately, several methods that we’ve already discussed would tell us how to do it with minimum costs \( C(a) = 5, C(b) = C(c) = C(d) = 6, C(e) = C(f) = 5, \) and \( C(g) = 4; \) the total cost for all seven functions would then be 37. But we want to find a single Boolean chain that contains them all, and the shortest such chain is presumably much more efficient. How can we discover it?

Well, the task of finding a truly optimum chain for \( \{a, b, c, d, e, f, g\} \) is probably infeasible from a computational standpoint. But a surprisingly good solution can be found with the help of the “footprint” idea explained earlier. Namely, we know how to compute not only a function’s minimum cost, but also the set of all first steps consistent with that minimum cost in a normal chain. Function \( e \), for example, has cost 5, but only if we evaluate it by starting with one of the instructions

\[
x_5 = x_1 \oplus x_4 \quad \text{or} \quad x_5 = x_2 \land \overline{x}_3 \quad \text{or} \quad x_5 = x_2 \lor x_3.
\]

Fortunately, one of the desirable first steps belongs to four of the seven footprints: Functions \( c, d, f, \) and \( g \) can all be evaluated optimally by starting with \( x_5 = x_2 \oplus x_3 \). So that is a natural choice; it essentially saves us three steps, because we know that at most 33 of the original 37 steps will be needed to finish.

Now we can recompute the costs and footprints of all \( 2^{16} \) functions, proceeding as before but also initializing the cost of the new function \( x_5 \) to zero. The costs of functions \( c, d, f, \) and \( g \) decrease by 1 as a result, and the footprints change too. For example, function \( a \) still has cost 5, but its footprint has increased from \( \{ x_1 \oplus x_3, x_2 \land x_3 \} \) to \( \{ x_1 \oplus x_3, x_1 \land x_4, x_1 \land x_4, x_2 \land x_3, \overline{x}_2 \land x_4, x_2 \land x_4, x_4 \land x_5, x_4 \land x_5 \} \) when the function \( x_5 = x_2 \oplus x_3 \) is available for free.

In fact, \( x_6 = \overline{x}_1 \land x_4 \) is common to four of the new footprints, so again we have a natural way to proceed. And when everything is recalculated with zero cost given to both \( x_5 \) and \( x_6 \), the subsequent step \( x_7 = x_3 \land \overline{x}_6 \) turns out to be desirable in five of the newest footprints. Continuing in this “greedy” fashion, we aren’t always so lucky, but a remarkable chain of only 22 steps does emerge:

\[
\begin{align*}
x_5 &= x_2 \oplus x_3, & x_{13} &= x_1 \oplus x_7, & \overline{a} = x_{20} = x_{14} \land \overline{x}_{19}, \\
x_6 &= \overline{x}_1 \land x_4, & x_{14} &= x_5 \oplus x_6, & \overline{b} = x_{21} = x_7 \oplus x_{12}, \\
x_7 &= x_3 \land \overline{x}_6, & x_{15} &= x_7 \lor x_{12}, & \overline{c} = x_{22} = \overline{x}_8 \land x_{15}, \\
x_8 &= x_1 \oplus x_2, & x_{16} &= x_1 \lor x_5, & \overline{d} = x_{23} = x_9 \land \overline{x}_{13}, \\
x_9 &= x_4 \oplus x_5, & x_{17} &= x_5 \lor x_6, & \overline{e} = x_{24} = x_6 \lor x_{18}, \\
x_{10} &= \overline{x}_7 \land x_8, & x_{18} &= x_9 \land x_{10}, & \overline{f} = x_{25} = \overline{x}_8 \land x_{17}, \\
x_{11} &= x_9 \oplus x_{10}, & x_{19} &= x_3 \land x_9, & g = x_{26} = x_7 \lor x_{16}. \\
x_{12} &= x_5 \land x_{11},
\end{align*}
\]

(This is a normal chain, so it contains the normalizations \( \{ \overline{a}, \overline{b}, \overline{c}, \overline{d}, \overline{e}, \overline{f}, \overline{g} \} \) instead of \( \{ a, b, c, d, e, f, g \} \). Simple changes will produce the unnormalized functions without changing the cost.)

**Partial functions.** In practice the output value of a Boolean function is often specified only at certain inputs \( x_1 \ldots x_n \), and the outputs in other cases don’t really matter. We might know, for example, that some of the input combinations
will never arise. In such cases, we place an asterisk into the corresponding positions of the truth table, instead of specifying 0 or 1 everywhere.

The seven-segment display provides a case in point, because most of its applications involve only the ten binary-coded decimal inputs for which we have \((x_1 x_2 x_3 x_4)\leq 9\). We don’t care what segments are visible in the other six cases. So the truth tables of \((42)\) actually become

\[
\begin{align*}
a &= 1011 0111 11** ****, \\
b &= 1111 1001 11** ****, \\
c &= 1101 1111 11** ****, \\
d &= 1011 0110 11** ****, \\
e &= 1010 0010 10** ****, \\
f &= 1000 111* 11** ****, \\
g &= 0011 1110 11** ****.
\end{align*}
\]

Function \(f\) here has an asterisk also in position \(x_1 x_2 x_3 x_4 = 0111\), because a ‘7’ can be displayed as either \(7\) or \(\tilde{7}\). Both of these styles appeared about equally often in the display units available to the author when this section was written. Truncated variants of the \(5\) and the \(9\) were sometimes seen in olden days, but they have thankfully disappeared.

Asterisks in truth tables are generally known as don’t-cares—a quaint term that could only have been invented by an electrical engineer. Table 3 shows that the freedom to choose arbitrary outputs is advantageous. For example, there are \((\frac{16}{3})^{2^{13}} = 4,587,320\) truth tables with 3 don’t-cares; 69% of them cost 4 or less, even though only 21% of the asterisk-free truth tables permit such economy. On the other hand, don’t-cares don’t save us as much as we might hope; exercise 63 proves that a random function with, say, 30% don’t-cares in its truth table tends to save only about 30% of the cost of a fully specified function.

What is the shortest Boolean chain that evaluates the seven partially specified functions in \((44)\)? Our greedy-footprint method adapts itself readily to the presence of don’t-cares, because we can OR together the footprints of all \(d\) functions that match a pattern with \(d\) asterisks. The initial costs to evaluate each function separately are now reduced to \(C(a) = 3, C(b) = C(c) = 2, C(d) = 5, C(e) = 2, C(f) = 3, C(g) = 4,\) totalling just 21 instead of 37. Function \(g\) hasn’t gotten cheaper, but it does have a larger footprint. Proceeding as before, but taking advantage of the don’t-cares, we now can find a suitable chain of length only 13—a chain with fewer than two operations per output(!):

\[
\begin{align*}
x_5 &= x_1 \oplus x_2, & \tilde{c} &= x_{10} = x_4 \vee x_8, & \bar{b} &= x_{15} = x_2 \wedge \bar{x}_{13}, \\
x_6 &= x_3 \wedge \bar{x}_4, & g &= x_{11} = x_7 \oplus x_8, & \bar{c} &= x_{16} = \bar{x}_2 \wedge x_6, \\
x_7 &= x_1 \oplus x_3, & x_{12} = x_4 \oplus x_{11}, & \tilde{f} &= x_{17} = \bar{x}_5 \wedge x_9, \\
x_8 &= x_2 \wedge \bar{x}_6, & \bar{d} &= x_{13} = x_{10} \wedge x_{12}, \\
x_9 &= x_3 \vee x_4, & \bar{a} &= x_{14} = \bar{x}_3 \wedge x_{13},
\end{align*}
\]

**Tic-tac-toe.** Let’s turn now to a slightly larger problem, based on a popular children’s game. Two players take turns filling the cells of a \(3 \times 3\) grid. One player writes \(X\)’s and the other writes \(O\)’s, continuing until there either are three
The number of 4-variable functions with $d$ don't-cares and cost $c$

<table>
<thead>
<tr>
<th>$d$</th>
<th>$c = 0$</th>
<th>$c = 1$</th>
<th>$c = 2$</th>
<th>$c = 3$</th>
<th>$c = 4$</th>
<th>$c = 5$</th>
<th>$c = 6$</th>
<th>$c = 7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>60</td>
<td>456</td>
<td>2474</td>
<td>10624</td>
<td>24184</td>
<td>25008</td>
<td>2720</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>960</td>
<td>7296</td>
<td>35040</td>
<td>131904</td>
<td>227296</td>
<td>119072</td>
<td>2560</td>
</tr>
<tr>
<td>2</td>
<td>1200</td>
<td>7200</td>
<td>52730</td>
<td>221840</td>
<td>700512</td>
<td>816448</td>
<td>106144</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5600</td>
<td>33600</td>
<td>228092</td>
<td>831232</td>
<td>204592</td>
<td>1381952</td>
<td>60102</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>18200</td>
<td>108816</td>
<td>666628</td>
<td>2034408</td>
<td>3505344</td>
<td>1011828</td>
<td>8326</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>43680</td>
<td>257472</td>
<td>1367776</td>
<td>3351488</td>
<td>3491648</td>
<td>433566</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>80080</td>
<td>455616</td>
<td>2015072</td>
<td>3648608</td>
<td>1914800</td>
<td>80616</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>114400</td>
<td>606944</td>
<td>2115438</td>
<td>2474888</td>
<td>535568</td>
<td>12032</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>128860</td>
<td>640704</td>
<td>1528808</td>
<td>960080</td>
<td>71520</td>
<td>896</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>114080</td>
<td>440660</td>
<td>707488</td>
<td>197632</td>
<td>4160</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>78960</td>
<td>224144</td>
<td>189248</td>
<td>20160</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>41440</td>
<td>72064</td>
<td>25472</td>
<td>800</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>15480</td>
<td>12300</td>
<td>1280</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3680</td>
<td>800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>480</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

X's or three O's in a straight line (in which case that player wins) or all nine cells are filled without a winner (in which case it's a "cat's game" or tie). For example, the game might proceed thus:

$$
\begin{array}{cccccccc}
\# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# \\
\end{array}
$$

(46)

X has won. Our goal is to design a machine that plays tic-tac-toe optimally—making a winning move from each position in which a forced victory is possible, and never making a losing move from a position in which defeat is avoidable.

More precisely, we will set things up so that there are 18 Boolean variables $x_1, \ldots, x_9, o_1, \ldots, o_9$, which govern lamps to illuminate cells of the current position. The cells are numbered as on a telephone dial. Cell $j$ displays an X if $x_j = 1$, an O if $o_j = 1$, or remains blank if $x_j = o_j = 0$.* We never have $x_j = o_j = 1$, because that would display ‘\$‘. We shall assume that the variables $x_1, \ldots, x_9, o_1, \ldots, o_9$ have been set to indicate a legal position in which nobody has won; the computer plays the X's, and it is the computer's turn to move. For this purpose we want to define nine functions $y_1, \ldots, y_9$, where $y_j$ means “change $x_j$ from 0 to 1.” If the current position is a cat's game, we should make $y_1 = \cdots = y_9 = 0$; otherwise exactly one $y_j$ should be equal to 1, and of course the output value $y_j = 1$ should occur only if $x_j = o_j = 0$.

With 18 variables, each of our nine functions $y_j$ will have a truth table of size $2^{18} = 262,144$. It turns out that only 4520 legal inputs $x_1, x_2, o_1, \ldots, o_9$ are

---

* This setup is based on an exhibit from the early 1960s at the Museum of Science and Industry in Chicago, where the author was first introduced to the magic of switching circuits. The machine in Chicago, designed circa 1940 by W. Keister at Bell Telephone Laboratories, allowed me to go first; yet I soon discovered that there was no way to defeat it. Therefore I decided to move as stupidly as possible, hoping that the designer had not anticipated such bizarre behavior. In fact I allowed the machine to reach a position where it had two winning moves and it seized both of them! Moving twice is of course a flagrant violation of the rules, so I had won a moral victory even though the machine announced that I had lost.
possible, so those truth tables are 98.3% filled with don’t-cares. Still, 4520 is uncomfortably large if we hope to design and understand a Boolean chain that makes sense intuitively. Section 7.1.4 will discuss alternative ways to represent Boolean functions, by which it is often possible to deal with hundreds of variables even though the associated truth tables are impossibly large.

Most functions of 18 variables require more than $2^{18}/18$ gates, but let’s hope we can do better. Indeed, a plausible strategy for making suitable moves in tic-tac-toe suggests itself immediately, in terms of several conditions that aren’t hard to recognize:

- $w_j$, an $\times$ in cell $j$ will win, completing a line of $\times$’s;
- $b_j$, an $\oslash$ in cell $j$ would lose, completing a line of $\oslash$’s;
- $f_j$, an $\times$ in cell $j$ will give $\times$ two ways to win;
- $d_j$, an $\oslash$ in cell $j$ would give $\oslash$ two ways to win.

For example, $\times$’s move to the center in (46) was needed to block $\oslash$, so it was of type $b_5$; fortunately it was also of type $f_5$, forcing a win on the next move.

Let $L = \{\{1,2,3\}, \{4,5,6\}, \{7,8,9\}, \{1,4,7\}, \{2,5,8\}, \{3,6,9\}, \{1,5,9\}, \{3,5,7\}\}$ be the set of winning lines. Then we have

- $m_j = x_j \land b_j$; \hspace{1cm} [moving in cell $j$ is legal] \hspace{1cm} (47)
- $w_j = m_j \land \bigvee_{(i,j,k) \in L} (x_i \land x_k)$; \hspace{1cm} [moving in cell $j$ wins] \hspace{1cm} (48)
- $b_j = m_j \land \bigvee_{(i,j,k) \in L} (x_i \land x_k)$; \hspace{1cm} [moving in cell $j$ blocks] \hspace{1cm} (49)
- $f_j = m_j \land S_2(\{\alpha_{ik} | (i,j,k) \in L\})$; \hspace{1cm} [moving in cell $j$ forks] \hspace{1cm} (50)
- $d_j = m_j \land S_2(\{\beta_{ik} | (i,j,k) \in L\})$; \hspace{1cm} [moving in cell $j$ defends] \hspace{1cm} (51)

here $\alpha_{ik}$ and $\beta_{ik}$ denote a single $\times$ or $\oslash$ together with a blank, namely

$$\alpha_{ik} = (x_i \land m_k) \lor (m_i \land x_k), \quad \beta_{ik} = (x_i \land m_k) \lor (m_i \land x_k). \hspace{1cm} (52)$$

For example, $b_1 = m_1 \land ((x_2 \land x_3) \lor (x_4 \land x_7) \lor (x_5 \land x_9))$; $f_2 = m_2 \land S_2(\alpha_{13}, \alpha_{58}) = m_2 \land \alpha_{13} \land \alpha_{58}$; $d_5 = m_5 \land S_2(\beta_{19}, \beta_{28}, \beta_{57}, \beta_{36})$.

With these definitions we might try rank-ordering our moves thus:

$$\{w_1, \ldots, w_9\} > \{b_1, \ldots, b_9\} > \{f_1, \ldots, f_9\} > \{d_1, \ldots, d_9\} > \{m_1, \ldots, m_9\}. \hspace{1cm} (53)$$

"Win if you can; otherwise block if you can; otherwise fork if you can; otherwise defend if you can; otherwise make a legal move." Furthermore, when choosing
between legal moves it seems sensible to use the ordering

\[ m_6 > m_1 > m_3 > m_9 > m_2 > m_6 > m_8 > m_4, \]

(54)
because 5, the middle cell, occurs in four winning lines, while a corner move to 1, 3, 9, or 7 occurs in three, and a side cell 2, 6, 8, or 4 occurs in only two. We might as well adopt this ordering of subscripts within all five groups of moves \( \{w_j\}, \{b_j\}, \{f_j\}, \{d_j\}, \) and \( \{m_j\} \) in (53).

To ensure that at most one move is chosen, we define \( \mu'_j, \nu'_j, \ell'_j, d'_j, m'_j \) to mean “a prior choice is better.” Thus, \( \mu'_5 = 0, \mu'_1 = w_5, \mu'_3 = w_1 \lor w'_1, \ldots, \mu'_4 = w_8 \lor w'_8, \nu'_5 = w_4 \lor w'_4, \ell'_1 = b_5 \lor b'_5, \ell'_3 = m_4 \lor m'_4 \). Then we can complete the definition of a tic-tac-toe automaton by letting

\[ y_j = (w_j \land \bar{w}'_j) \lor (b_j \land \bar{b}'_j) \lor (f_j \land \bar{f}'_j) \lor (d_j \land \bar{d}'_j) \lor (m_j \land \bar{m}'_j), \quad \text{for } 1 \leq j \leq 9. \]

(55)
So we’ve constructed 9 gates for the \( m \)’s, 48 for the \( w \)’s, 48 for the \( b \)’s, 144 for the \( \alpha \)’s and \( \beta \)’s, 35 for the \( f \)’s (with the help of Fig. 9), 35 for the \( d \)’s, 43 for the primed variables, and 80 for the \( y \)’s. Furthermore we can use our knowledge of partial 4-variable functions to reduce the six operations in (52) to only four,

\[ \alpha_{ik} = (x_i \lor \bar{x}_k) \lor (a_i \lor \bar{a}_k), \quad \beta_{ik} = (x_i \lor \bar{x}_k) \lor (a_i \lor \bar{a}_k). \]

(56)
This trick saves 48 gates; so our design has cost 396 gates altogether.

The strategy for tic-tac-toe in (47)-(56) works fine in most cases, but it also has some glaring glitches. For example, it loses ignominiously in the game

\[ \begin{array}{cccccccc}
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\end{array} \]

(57)
the second \( \times \) move is \( d_9 \), defending against a fork by \( \bigcirc \), yet it actually forces \( \bigcirc \) to fork in the opposite corner! Another failure arises, for example, after position \[ \begin{array}{cccccccc}
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\end{array} \]

when move \( m_9 \) leads to the cat’s game \[ \begin{array}{cccccccc}
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\end{array} \]

instead of to the victory for \( \times \) that appeared in (46). Exercise 65 patches things up and obtains a fully correct Boolean tic-tac-toe player that needs just 445 gates.

*Functional decomposition. If the function \( f(x_1, \ldots, x_n) \) can be written in the form \( g(x_1, \ldots, x_k, h(x_{k+1}, \ldots, x_n)) \), it’s usually a good idea to evaluate \( y = h(x_{k+1}, \ldots, x_n) \) first and then to compute \( g(x_1, \ldots, x_k, y) \). Robert L. Ashenhurst inaugurated the study of such decompositions in 1952 [see Annals Computation Lab. Harvard University 29 (1957), 74–116], and observed that there’s an easy way to recognize when \( f \) has this special property: If we write the truth table for \( f \) in a \( 2^k \times 2^{n-k} \) array as in (36), with rows for each setting of \( x_1 \ldots x_k \) and columns for each setting of \( x_{k+1} \ldots x_n \), then the desired subfunctions \( g \) and \( h \) exist if and only if the columns of this array have at most two different values. For example, the truth table for the function \( \langle x_1, x_2, \langle x_3 x_4 x_5 \rangle \rangle \) is

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]
when expressed in this two-dimensional form. One type of column corresponds to the case \( h(x_{k+1}, \ldots, x_n) = 0 \); the other corresponds to \( h(x_{k+1}, \ldots, x_n) = 1 \).

In general the variables \( X = \{ x_1, \ldots, x_n \} \) might be partitioned into any two disjoint subsets \( Y = \{ y_1, \ldots, y_k \} \) and \( Z = \{ z_1, \ldots, z_{n-k} \} \), and we might have \( f(x) = g(y, h(z)) \). We could test for a \((Y,Z)\) decomposition by looking at the columns of the \( 2^k \times 2^{n-k} \) truth table whose rows correspond to values of \( y \). But there are \( 2^n \) such ways to partition \( X \); and all of them are potential winners, except for trivial cases when \(|Y| = 0\) or \(|Z| \leq 1\). How can we avoid examining such a humongous number of possibilities?

A practical way to proceed was discovered by V. Y.-S. Shen, A. C. McKellar, and P. Weiner [IEEE Transactions C-20 (1971), 304–309], whose method usually needs only \( O(n^2) \) steps to identify any potentially useful partition \((Y,Z)\) that may exist. The basic idea is simple: Suppose \( x_i \in Z \), \( x_j \in Z \), and \( x_m \in Y \). Define eight binary vectors \( \delta_l \) for \( l = (l_1 l_2 l_3) \), where \( \delta_l \) has \((l_1, l_2, l_3)\) respectively in components \((i, j, m)\), and zeros elsewhere. Consider any randomly chosen vector \( x = x_1 \ldots x_n \), and evaluate \( f_l = f(x \oplus \delta_l) \) for \( 0 \leq l \leq 7 \). Then the four pairs

\[
\begin{pmatrix}
  f_0 \\
  f_1 \\
  f_2 \\
  f_3 \\
  f_4 \\
  f_5 \\
  f_6 \\
  f_7
\end{pmatrix}
\]

will appear in a \( 2 \times 4 \) submatrix of the \( 2^k \times 2^{n-k} \) truth table. So a decomposition is impossible if these pairs are distinct, or if they contain three different values.

Let’s call the pairs “good” if they’re all equal, or if they have only two different values. Otherwise they’re “bad.” If \( f \) has essentially random behavior, we’ll soon find bad pairs if we do this experiment with several different randomly chosen vectors \( x \), because only 88 of the 256 possibilities for \( f_0 f_1 \ldots f_7 \) correspond to a good set of pairs; the probability of finding good pairs ten times in a row is only \((\frac{88}{256})^{10} \approx .00002\). And when we do discover bad pairs, we can conclude that

\[
x_i \in Z \quad \text{and} \quad x_j \in Z \quad \implies \quad x_m \in Z,
\]

because the alternative \( x_m \in Y \) is impossible.

Suppose, for example, that \( n = 9 \) and that \( f \) is the function whose truth table 1100100100011...00101 consists of the 512 most significant bits of \( \pi \), in binary notation. (This is the “more-or-less random function” that we studied for \( n = 4 \) in (5) and (6) above.) Bad pairs for this \( \pi \) function are quickly found in each of the cases \((i,j,m)\) for which \( m \neq i < j \neq m \). Indeed, in the author’s experiments, 170 of those 252 cases were decided immediately; the average number of random \( x \) vectors per case was only 1.52; and only one case needed as many as eight \( x \)’s before bad pairs appeared. Thus (59) holds for all relevant \((i,j,m)\), and the function is clearly indecomposable. In fact, exercise 73 points out that we needn’t make 252 tests to establish the indecomposability of this \( \pi \) function; only \( \binom{9}{2} = 36 \) of them would have been sufficient.

Turning to a less random function, let \( f(x_1, \ldots, x_9) = (\det X) \mod 2 \), where

\[
X = \begin{pmatrix}
  x_1 & x_2 & x_3 \\
  x_4 & x_5 & x_6 \\
  x_7 & x_8 & x_9
\end{pmatrix}
\]
This function does not satisfy condition (59) when \( i = 1, j = 2, \) and \( m = 3, \) because there are no bad pairs in that case. But it does satisfy (59) for \( 4 \leq m \leq 9 \) when \( \{i, j\} = \{1, 2\}. \) We can denote this behavior by the convenient abbreviation \( 12 \to 456789 \); the full set of implications, for all pairs \( \{i, j\}, \) is

\[
\begin{align*}
12 &\to 456789 & 18 &\to 34569 & 27 &\to 34569 & 37 &\to 24568 & 48 &\to 12369 & 67 &\to 12358 \\
13 &\to 456789 & 19 &\to 24568 & 28 &\to 134679 & 38 &\to 14567 & 49 &\to 12358 & 68 &\to 12347 \\
14 &\to 235689 & 23 &\to 456789 & 29 &\to 14567 & 39 &\to 124578 & 56 &\to 123789 & 69 &\to 124578 \\
15 &\to 36789 & 24 &\to 36789 & 34 &\to 25789 & 45 &\to 123789 & 57 &\to 12369 & 78 &\to 123456 \\
16 &\to 25789 & 25 &\to 134679 & 35 &\to 14789 & 46 &\to 123789 & 58 &\to 134679 & 79 &\to 123456 \\
17 &\to 235689 & 26 &\to 14789 & 36 &\to 124578 & 47 &\to 235689 & 59 &\to 12347 & 89 &\to 123456
\end{align*}
\]

(see exercise 69). Bad pairs are a little more difficult to find when we probe this function at random: The average number of \( x \)'s needed in the author's experiments rose to about 3.6, when bad pairs did exist. And of course there was a need to limit the testing, by choosing a tolerance threshold \( t \) and then giving up when \( t \) consecutive trials failed to find any bad pairs. Choosing \( t = 10 \) would have found all but 8 of the 198 implications listed above.

Implications like (59) are Horn clauses, and we know from Section 7.1.1 that it's easy to make further deductions from Horn clauses. Indeed, the method of exercise 74 will deduce that the only possible partition with \( |Z| > 1 \) is the trivial one \( (Y = \emptyset, Z = \{x_1, \ldots, x_9\}), \) after looking at fewer than 50 cases \((i, j, m)\).

Similar results occur when \( f(x_1, \ldots, x_9) = [\text{per } X > 0], \) where per denotes the permanent function. (In this case \( f \) tells us if there is a matching in the bipartite subgraph of \( K_{3,3} \) whose edges are specified by the variables \( x_1 \ldots x_9). \)

Now there are just 180 implications,

\[
\begin{align*}
12 &\to 456789 & 18 &\to 3459 & 27 &\to 3459 & 37 &\to 2468 & 48 &\to 1269 & 67 &\to 1358 \\
13 &\to 456789 & 19 &\to 2468 & 28 &\to 134679 & 38 &\to 1567 & 49 &\to 1358 & 68 &\to 2347 \\
14 &\to 235689 & 23 &\to 456789 & 29 &\to 1567 & 39 &\to 124578 & 56 &\to 123789 & 69 &\to 124578 \\
15 &\to 36789 & 24 &\to 36789 & 34 &\to 25789 & 45 &\to 123789 & 57 &\to 12369 & 78 &\to 123456 \\
16 &\to 25789 & 25 &\to 134679 & 35 &\to 1489 & 46 &\to 123789 & 58 &\to 134679 & 79 &\to 123456 \\
17 &\to 235689 & 26 &\to 1489 & 36 &\to 124578 & 47 &\to 235689 & 59 &\to 2347 & 89 &\to 123456
\end{align*}
\]

only 122 of which would have been discovered with \( t = 10 \) as the cutoff threshold. (The best choice of \( t \) is not clear; perhaps it should vary dynamically.) Still, those 122 Horn clauses were more than enough to establish indecomposability.

What about a decomposable function? With \( f = \langle x_2x_3x_6x_9, x_1x_4x_5x_7x_8 \rangle \) we get \( i \land j \Rightarrow m \) for all \( m \notin \{i, j\}, \) except when \( \{i, j\} \subseteq \{1, 4, 5, 7, 8\}; \) in the latter case, \( m \) must also belong to \( \{1, 4, 5, 7, 8\}. \) Although only 185 of these 212 implications were discovered with tolerance \( t = 10, \) the partition \( Y = \{x_2, x_3, x_6, x_9\}, \)

\( Z = \{x_1, x_4, x_5, x_7, x_8\} \) emerged quickly as a strong possibility.

Whenever a potential decomposition is supported by the evidence, we need to verify that the corresponding \( 2^k \times 2^n \) truth table does indeed have only one or two distinct columns. But we're happy to spend \( 2^n \) units of time on that verification, because we've greatly simplified the evaluation of \( f. \)
The comparison function \( f = [(x_1 x_2 x_3 x_4)_2 \geq (x_5 x_6 x_7 x_8)_2 + x_9] \) is another interesting case. Its 184 potentially deducible implications are

\[
\begin{align*}
12 &\rightarrow 3456789 & 18 &\rightarrow 2345679 & 27 &\rightarrow 34689 & 37 &\rightarrow 489 & 48 &\rightarrow 9 & 67 &\rightarrow 23489 \\
13 &\rightarrow 2456789 & 19 &\rightarrow 2345678 & 28 &\rightarrow 34679 & 38 &\rightarrow 479 & 49 &\rightarrow 8 & 68 &\rightarrow 23479 \\
14 &\rightarrow 2356789 & 23 &\rightarrow 46789 & 29 &\rightarrow 34678 & 39 &\rightarrow 478 & 56 &\rightarrow 123478 & 69 &\rightarrow 23478 \\
15 &\rightarrow 2346789 & 24 &\rightarrow 36789 & 34 &\rightarrow 789 & 45 &\rightarrow 1236789 & 57 &\rightarrow 123468 & 78 &\rightarrow 349 \\
16 &\rightarrow 2345789 & 25 &\rightarrow 1346789 & 35 &\rightarrow 1246789 & 46 &\rightarrow 23789 & 58 &\rightarrow 1234679 & 79 &\rightarrow 348 \\
17 &\rightarrow 2345678 & 26 &\rightarrow 34789 & 36 &\rightarrow 24789 & 47 &\rightarrow 389 & 59 &\rightarrow 1234678 & 89 &\rightarrow 4, \\
\end{align*}
\]

and 145 of them were found when \( t = 10 \). Three decompositions reveal themselves in this case, having \( Z = \{x_4, x_8, x_9\} \), \( Z = \{x_3, x_4, x_7, x_9\} \), and \( Z = \{x_2, x_3, x_4, x_6, x_7, x_8, x_9\} \), respectively. Ashenhurst proved that we can reduce \( f \) immediately as soon as we find a nontrivial decomposition; the other decompositions will show up later, when we try to reduce the simpler functions \( g \) and \( h \).

*Decomposition of partial functions.* When the function \( f \) is only partially specified, a decomposition with partition \((Y, Z)\) hinges on being able to assign values to the don't-cares so that at most two different columns appear in the corresponding \( 2^k \times 2^{n-k} \) truth table.

Two vectors \( u_1 \ldots u_m \) and \( v_1 \ldots v_m \) consisting of 0s, 1s, and \( * \)s are said to be *incompatible* if either \( u_j = 0 \) and \( v_j = 1 \) or \( u_j = 1 \) and \( v_j = 0 \), for some \( j \)—equivalently, if the subcubes of the \( n \)-cube specified by \( u \) and \( v \) have no points in common. Consider the graph whose vertices are the columns of a truth table with don't-cares, where \( u \) — \( v \) if and only if \( u \) and \( v \) are incompatible. We can assign values to the \( * \)'s to achieve at most two distinct columns if and only if this graph is bipartite. For if \( u_1, \ldots, u_t \) are mutually compatible, their generalized consensus \( u_1 \uplus \cdots \uplus u_t \), defined in exercise 7.1.1-32, is compatible with all of them. [See S. L. Hight, *IEEE Trans. C-22* (1973), 103-110; E. Boros, V. Gurvich, P. L. Hammer, T. Ibaraki, and A. Kogan, *Discrete Applied Math.* 62 (1995), 51-75.]

Since a graph is bipartite if and only if it contains no odd cycles, we can easily test this condition with a depth-first search (see Section 7.4.1).

Consequently the method of Shen, McKellar, and Weiner works also when don't-cares are present: The four pairs in (g8) are considered bad if and only if three of them are mutually incompatible. We can operate almost as before, although bad pairs will naturally be harder to find when there are lots of \( * \)'s (see exercise 72). However, Ashenhurst's theorem no longer applies. When several decompositions exist, they all should be explored further, because they might use different settings of the don't-cares, and some might be better than the others.

Although most functions \( f(x) \) have no simple decomposition \( g(y, h(z)) \), we needn't give up hope too quickly, because other forms like \( g(y, h_1(z), h_2(z)) \) might well lead to an efficient chain. If, for example, \( f \) is symmetric in three of its variables \( \{z_1, z_2, z_3\} \), we can always write \( f(x) = g(y, S_{1,2}(z_1, z_2, z_3), S_{1,3}(z_1, z_2, z_3)) \), since \( S_{1,2}(z_1, z_2, z_3) \) and \( S_{1,3}(z_1, z_2, z_3) \) characterize the value of \( z_1 + z_2 + z_3 \). (Notice that just four steps will suffice to compute both \( S_{1,2} \) and \( S_{1,3} \)).

In general, as observed by H. A. Curtis [JACM 8 (1961), 484-496], \( f(x) \) can be expressed in the form \( g(y, h_1(z), \ldots, h_n(z)) \) if and only if the \( 2^k \times 2^{n-k} \) truth
table corresponding to $Y$ and $Z$ has at most $2^r$ different columns. And when don't-cares are present, the same result holds if and only if the incompatibility graph for $Y$ and $Z$ can be colored with at most $2^r$ colors.

For example, the function $f(x) = (\det X) \bmod 2$ considered above turns out to have eight distinct columns when $Z = \{x_4, x_5, x_6, x_7, x_8, x_9\}$; that's a surprisingly small number, considering that the truth table has 8 rows and 64 columns. From this fact we might be led to discover how to expand a determinant by cofactors of the first row,

$$f(x) = x_1 \wedge h_1(x_4, \ldots, x_9) \oplus x_2 \wedge h_2(x_4, \ldots, x_9) \oplus x_3 \wedge h_3(x_4, \ldots, x_9),$$

if we didn't already know such a rule.

When there are $d \leq 2^r$ different columns, we can think of $f(x)$ as a function of $y$ and $h(z)$, where $h$ takes each binary vector $z_1 \ldots z_{n-k}$ into one of the values \{0, 1, \ldots, d-1\}. Thus $(h_1, \ldots, h_r)$ is essentially an encoding of the different column types, and we hope to find very simple functions $h_1, \ldots, h_r$ that provide such an encoding. Moreover, if $d$ is strictly less than $2^r$, the function $g(y, h_1, \ldots, h_r)$ will have many don't-cares that may well decrease its cost.

The distinct columns might also suggest a function $g$ for which the $h$'s have don't-cares. For example, we can use $g(y_1, y_2, h_1, h_2) = (y_1 \oplus (h_1 \wedge y_2)) \wedge h_2$ when all columns are either $(0, 0, 0, 0)^T$ or $(0, 0, 1, 1)^T$ or $(0, 1, 1, 0)^T$; then the value of $h_1(z)$ is arbitrary when $z$ corresponds to an all-zero column. H. A. Curtis has explained how to exploit this idea when $|Y| = 1$ and $|Z| = n - 1$ [see IEEE Transactions C-25 (1976), 1033–1044].


**Larger values of $n$.** We've been considering only rather tiny examples of Boolean functions. Theorem S tells us that large, random examples are inherently difficult; but practical examples might well be highly nonrandom. So it makes sense to search for simplifications using heuristic methods.

When $n$ grows, the best ways currently known for dealing with Boolean functions generally start with a Boolean chain—not with a huge truth table—and they try to improve that chain via "local changes." The chain can be specified by a set of equations. Then, if an intermediate result is used in comparatively few subsequent steps, we can try to eliminate it, temporarily making those subsequent steps into functions of three variables, and reformulating those functions in order to make a better chain when possible.

For example, suppose the gate $x_i = x_j \cdot x_k$ is used only once, in the gate $x_l = x_i \cdot x_m$, so that $x_l = (x_j \cdot x_k) \cdot x_m$. Other gates might already exist, by which we have computed other functions of $x_j, x_k, x_l$, and $x_m$; and the definitions of $x_j, x_k$, and $x_m$ may imply that some of the joint values of $(x_j, x_k, x_m)$ are impossible. Thus we might be able to compute $x_l$ from other gates by doing just one further operation. For example, if $x_l = x_j \land x_k$ and $x_l = x_k \lor x_m$, and if the values $x_j \lor x_m$ and $x_k \lor x_m$ appear elsewhere in the chain, we can set $x_l = (x_j \lor x_m) \land (x_k \lor x_m)$; this eliminates $x_i$ and reduces the cost by 1. Or if,
say, \( x_j \land (x_k \oplus x_m) \) appears elsewhere and we know that \( x_j x_k x_m \neq 101 \), we can set \( x_i = x_m \oplus (x_j \land (x_k \oplus x_m)) \).

If \( x_i \) is used only in \( x_i \) and \( x_i \) is used only in \( x_p \), then gate \( x_p \) depends on four variables, and we might be able to reduce the cost by using our total knowledge of four-variable functions, obtaining \( x_p \) in a better way while eliminating \( x_i \) and \( x_i \).

Similarly, if \( x_i \) appears only in \( x_i \) and \( x_p \), we can eliminate \( x_i \) if we find a better way to evaluate two different functions of four variables, possibly with don’t-cares and with other functions of those four variables available for free. Again, we know how to solve such problems, using the footprint method discussed above.

When no local changes are able to decrease the cost, we can also try local changes that preserve or even increase the cost, in order to discover different kinds of chains that might simplify in other ways. We shall discuss such local search methods extensively in Section 7.10.


**Lower bounds.** Theorem 5 tells us that nearly every Boolean function of \( n \geq 12 \) variables is hard to evaluate, requiring a chain whose length exceeds \( 2^n/n \).

Yet modern computers, which are built from logic circuits involving electric signals that represent thousands of Boolean variables, happily evaluate billions of Boolean functions every microsecond. Evidently there are plenty of important functions that can be evaluated quickly, in spite of Theorem 5. Indeed, the proof of that theorem was indirect; we simply counted the cases of low cost, so we learned absolutely nothing about any particular examples that might arise in practice. When we want to compute a given function and we can only think of a laborious way to do the job, how can we be sure that there’s no tricky shortcut?

The answer to that question is almost scandalous: After decades of concentrated research, computer scientists have been unable to find any explicit family of functions \( f(x_1, \ldots, x_n) \) whose cost is inherently nonlinear, as \( n \) increases. The true behavior is \( 2^n/n \), but no lower bound as strong as \( n \log \log \log n \) has yet been proved! Of course we could rig up artificial examples, such as “the lexicographically smallest truth table of length \( 2^n \)” that isn’t achievable by any Boolean chain of length \( [2^n/n] - 1 \); but such functions are surely not explicit. The truth table of an explicit function \( f(x_1, \ldots, x_n) \) should be computable in at most, say, \( 2^n \) units of time for some constant \( c \); that is, the time needed to specify all of the function values should be polynomial in the length of the truth table. Under those ground rules, no family of single-output functions is currently known to have a combinational complexity that exceeds \( 3n + O(1) \) as \( n \to \infty \).

[See N. Blum, Theoretical Computer Science 28 (1984), 337–345.]

The picture is not totally bleak, because several interesting linear lower bounds have been proved for functions of practical importance. A basic way to obtain such results was introduced by N. P. Red’kin in 1970: Suppose we have
an optimum chain of cost $r$ for $f(x_1, \ldots, x_n)$. By setting $x_n \leftarrow 0$ or $x_n \leftarrow 1$, we obtain reduced chains for the functions $g(x_1, \ldots, x_{n-1}) = f(x_1, \ldots, x_{n-1}, 0)$ and $h(x_1, \ldots, x_{n-1}) = f(x_1, \ldots, x_{n-1}, 1)$, having cost $r - u$ if $x_n$ was used as an input to $u$ different gates. Moreover, if $x_n$ is used in a “canalizing” gate $x_i = x_n \circ x_k$, where the operator $\circ$ is neither $\oplus$ nor $\equiv$, some setting of $x_n$ will force $x_i$ to be constant, thereby further reducing the chain for $g$ or $h$. Lower bounds on $g$ and/or $h$ therefore lead to a lower bound on $f$. (See exercises 77-81.)

But where are the proofs of nonlinear lower bounds? Almost every problem with a yes-no answer can be formulated as a Boolean function, so there's no shortage of explicit functions that we don't know how to evaluate in linear time, or even in polynomial time. For example, any directed graph $G$ with vertices $\{v_1, \ldots, v_n\}$ can be represented by its adjacency matrix $X$, where $x_{ij} = [v_i \rightarrow v_j]$; then

$$f(x_1, x_2, \ldots, x_m, \ldots, x_{m(m-1)}) = [G \text{ has a Hamiltonian path}]$$

is a Boolean function of $n = m(m-1)$ variables. We would dearly love to be able to evaluate this function in, say, $n^4$ steps. We do know how to compute the truth table for $f$ in $O(m!2^m) = 2^{n+O(\sqrt{n}\log n)}$ steps, since only $m!$ potential Hamiltonian paths exist; thus $f$ is indeed “explicit.” But nobody knows how to evaluate $f$ in polynomial time, or how to prove that there isn’t a $4n$-step chain.

For all we know, short Boolean chains for $f$ might exist, for each $n$. After all, Figs. 9 and 10 reveal the existence of fiendishly clever chains even in the cases of 4 and 5 variables. Efficient chains for all of the larger problems that we ever will need to solve might well be “out there” — yet totally beyond our grasp, because we don’t have time to find them. Even if an omniscient being revealed the simple chains to us, we might find them incomprehensible, because the shortest proof of their correctness might be longer than the number of cells in our brains.

Theorem S rules out such a scenario for most Boolean functions. But fewer than $2^{100}$ Boolean functions will ever be of practical importance in the entire history of the world, and Theorem S tells us zilch about them.

In 1974, Larry Stockmeyer and Albert Meyer were, however, able to construct a Boolean function $f$ whose complexity is provably huge. Their $f$ isn’t “explicit,” in the precise sense described above, but it isn’t artificial either; it arises naturally in mathematical logic. Consider symbolic statements such as

\[
\begin{align*}
O48+n015\#1063; \\
\forall m3n(m<n+1); \\
\forall n3m(m+1<n); \\
\forall w\forall b\forall a(2^3ab(a<ab\land\neg(ab)b)); \\
\forall \forall A(\forall B\forall C(\forall eA\land B\land C \land A)); \\
\forall (3n(n\in A) \land \exists m(m\in A \land m\in n));
\end{align*}
\]

(62) (63) (64) (65) (66) (67) (68) (69) (70)
Stockmeyer and Meyer defined a language \( L \) by using the 63-character alphabet
\[
\forall \exists \neg(\exists \in \mathbb{A} + \mathbb{V} + \mathbb{S} \leq 9 > abcdedefghijklmnopqABCD\ldots KLMNOPQ}\]
and giving conventional meanings to these symbols. Strings of lowercase letters within the sentences of \( L \), like \( \text{ab'} \) in (65), represent numeric variables, restricted to nonnegative integers; strings of uppercase letters represent set variables, restricted to finite sets of such numbers. For example, (66) means, “For all finite sets \( A \) and \( B \), we have \( A = B \) if and only if there doesn’t exist a number \( n \) that is in \( A \) but not in \( B \), or in \( B \) but not in \( A \).” Some of these statements are true; others are false. (See exercise 82.)

All of the strings (62)–(70) belong to \( L \), but the language is actually quite restricted: The only arithmetic operation allowed on a number is to add a constant; we can write ‘\( a + 13 \)’ but not ‘\( a + b \)’. The only relation allowed between a number and a set is elementhood (\( \in \) or \( \notin \)). The only relation allowed between sets is equality (\( = \)). Furthermore all variables must be quantified by \( \exists \) or \( \forall \).*

Every sentence of \( L \) that has length \( k \leq n \) can be represented by a binary vector of length \( 6n \), with zeros in the last \( 6(n - k) \) bits. Let \( f(x) \) be a Boolean function of \( 6n \) variables such that \( f(x) = 1 \) whenever \( x \) represents a true sentence of \( L \), and \( f(x) = 0 \) whenever \( x \) represents a sentence that is false; the value of \( f(x) \) is unspecified when \( x \) doesn’t represent a meaningful sentence. The truth table for such a function \( f \) can be constructed in a finite number of steps, according to theorems of Büchi and Elgot [Zeitschrift für math. Logik und Grundlagen der Math. 6 (1960), 66–92; Transactions of the Amer. Math. Soc. 98 (1961), 21–51]. But “finite” does not mean “feasible”: Stockmeyer and Meyer proved that
\[
C(f) > 2^{7 - 5} \quad \text{whenever} \quad n \geq 460 + 302r + 5.08 \ln r \quad \text{and} \quad r > 36. \tag{71}
\]
In particular, we have \( C(f) > 2^{426} > 10^{128} \) when \( n = 621 \). A Boolean chain with that many gates could never be built, since \( 10^{28} \) is a generous upper bound on the number of protons in the universe. So this is a fairly small, finite problem that will never be solved.

Details of Stockmeyer and Meyer’s proof appear in JACM 49 (2002), 753–784. The basic idea is that the language \( L \), though limited, is rich enough to describe truth tables and the complexity of Boolean chains, using fairly short sentences; hence \( f \) has to deal with inputs that essentially refer to themselves.

*For further reading. Thousands of significant papers have been written about networks of Boolean gates, because such networks underlie so many aspects of theory and practice. We have focused in this section chiefly on topics that are relevant to computer programming for sequential machines. But other topics have also been extensively investigated, of primary relevance to parallel computation, such as the study of small-depth circuits in which gates can have any number of inputs (“unlimited fan-in”). Ingo Wegener’s book The Complexity of

---

* Technically speaking, the sentences of \( L \) belong to “weak second-order monadic logic with one successor.” Weak second-order logic allows quantification over finite sets; monadic logic with \( k \) successors is the theory of unlabeled \( k \)-ary trees.
**Boolean Functions** (Teubner and Wiley, 1987) provides a good introduction to the entire subject.

We have mostly considered Boolean chains in which all binary operators have equal importance. For our purposes, gates such as $\oplus$ or $\subseteq$ are neither more nor less desirable than gates such as $\land$ or $\lor$. But it's natural to wonder if we can get by with only the monotone operators $\land$ and $\lor$ when we are computing a monotone function. Alexander Razborov has developed striking proof techniques to show that, in fact, monotone operators by themselves have inherently limited capabilities. He proved, for example, that all AND-OR chains to determine whether the permanent of an $n \times n$ matrix of 0s and 1s is zero or nonzero must have cost $n^{O(\log n)}$. [See Doklady Akademii Nauk SSSR 281 (1985), 798–801; Matematicheskie Zametki 37 (1985), 887–900.] By contrast, we will see in Section 7.5.1 that this problem, equivalent to “bipartite matching,” is solvable in only $O(n^{2.5})$ steps. Furthermore, the efficient methods in that section can be implemented as Boolean chains of only slightly larger cost, when we allow negation or other Boolean operations in addition to $\land$ and $\lor$. (Vaughan Pratt has called this “the power of negative thinking.”) An introduction to Razborov’s methods appears in exercises 85 and 86.

**EXERCISES**

1. [24] The “random” function in formula (6) corresponds to a Boolean chain of cost 4 and depth 4. Find a formula of depth 3 that has the same cost.

2. [27] Show how to compute (a) $w \oplus \langle xyz \rangle$ and (b) $w \land \langle xyz \rangle$ with formulas that have depth 3 and cost 5.

3. [M23] (B. I. Finikov, 1957.) If the Boolean function $f(x_1, \ldots, x_n)$ is true at exactly $k$ points, prove that $L(f) < 2n + (k-2)2^{k-1}$. Hint: Think of $k = 3$ and $n = 10^6$.

4. [M26] (P. M. Spirn, 1971.) Prove that the minimum depth and formula length of a Boolean function satisfy $\log L(f) < D(f) \leq \alpha \log L(f) + 1$, where $\alpha = 2/\log(2) \approx 3.419$. Hint: Every binary tree with $r \geq 3$ internal nodes contains a subtree with $s$ internal nodes, where $\frac{1}{3} r \leq s < \frac{2}{3} r$.

5. [27] The Fibonacci threshold function $\bar{F}_n(x_1, \ldots, x_n) = \langle x_1^{F_1} \land \cdots \land x_n^{F_n-2} \rangle$ was analyzed in exercise 7.1.1-101, when $n \geq 3$. Is there an efficient way to evaluate it?

6. [20] True or false A Boolean function $f(x_1, \ldots, x_n)$ is normal if and only if it satisfies the general distributive law $f(x_1, \ldots, x_n) \land y = f(x_1 \land y, \ldots, x_n \land y)$.

7. [20] Convert the Boolean chain `$x_0 = x_1 \lor x_4$, $x_6 = \bar{x}_2 \lor x_5$, $x_7 = \bar{x}_3 \land x_7$, $x_8 = x_6 \equiv x_7$' to an equivalent chain `$\bar{x}_0, \bar{x}_6, \bar{x}_7, \bar{x}_8$' in which every step is normal.

8. [20] Explain why (11) is the truth table of variable $x_k$.

9. [20] Algorithm L determines the lengths of shortest formulas for all functions $f$, but it gives no further information. Extend the algorithm so that it also provides actual minimum-length formulas like (6).

10. [20] Modify Algorithm L so that it computes $D(f)$ instead of $L(f)$.

11. [22] Modify Algorithm L so that, instead of lengths $L(f)$, it computes upper bounds $U(f)$ and footprints $\phi(f)$ as described in the text.

12. [15] What Boolean chain is equivalent to the minimum-memory scheme (13)?
13. [16] What are the truth tables of $f_1$, $f_2$, $f_3$, $f_4$, and $f_5$ in example (13)?

14. [22] What's a convenient way to compute the $5n(n-1)$ truth tables of (17), given the truth table of $g$? (Use bitwise operations as in (15) and (16).)

15. [28] Find short-as-possible ways to evaluate the following functions using minimum memory: (a) $S_2(x_1, x_2, x_3, x_4)$; (b) $S_1(x_1, x_2, x_3, x_4)$; (c) the function in (18).

16. [HM33] Prove that fewer than $2^{18}$ of the $2^{28}$ Boolean functions $f(x_1, \ldots, x_7)$ are computable in minimum memory.

17. [25] (M. S. Paterson, 1977.) Although Boolean functions $f(x_1, \ldots, x_n)$ cannot always be evaluated in $n$ registers, prove that $n+1$ registers are always sufficient. In other words, show that there is always a sequence of operations like (13) to compute $f(x_1, \ldots, x_n)$ if we allow $0 \leq j(i), k(i) \leq n$.

18. [35] Investigate optimum minimum-memory computations for $f(x_1, x_2, x_3, x_4, x_5)$: How many classes of five-variable functions have $C_m(f) = r$, for $r = 0, 1, 2, \ldots$?

19. [M22] If a Boolean chain uses $n$ variables and has length $r < n + 2$, prove that it must be either a "top-down" or a "bottom-up" construction.

20. [40] (R. Schroeppele, 2004.) A Boolean chain is canalisizing if it does not use the operators $\oplus$ or $\equiv$. Find the optimum cost, length, and depth of all 4-variable functions under this constraint. Does the footprint heuristic still give optimum results?

21. [46] For how many four-variable functions did the Harvard researchers discover an optimum vacuum-tube circuit in 1951?

22. [21] Explain the chain for $S_3$ in Fig. 10, by noting that it incorporates the chain for $S_{2,3}$ in Fig. 9. Find a similar chain for $S_2(x_1, x_2, x_3, x_4, x_5)$.

23. [25] Figure 10 illustrates only 16 of the 64 symmetric functions on five elements. Explain how to write down optimum chains for the others.

24. [47] Does every symmetric function $f$ have $C_m(f) = C(f)$?

25. [17] Suppose we want a Boolean chain that includes all functions of $n$ variables: Let $f_k(x_1, \ldots, x_n)$ be the function whose truth table is the binary representation of $k$, for $0 \leq k < m = 2^n$. What is $C(f_0, f_1, \ldots, f_{m-1})$?

26. [25] True or false: If $f(x_0, \ldots, x_n) = (x_0 \land g(x_1, \ldots, x_n)) \lor h(x_1, \ldots, x_n)$, where $g$ and $h$ are nontrivial Boolean functions whose joint cost is $C(gh)$, then $C(f) = 2 + C(gh)$.

27. [25] Can a full adder (22) be implemented in five steps using only minimum memory (that is, completely inside three one-bit registers)?

28. [26] Prove that $C(u \lor v) = C(u' \lor v') = 5$ for the two-output functions defined by

$$u \lor v = (x + y - (uv)) \mod 4,$$
$$u' \lor v' = (-x - y - (uv)) \mod 4.$$ Use these functions to evaluate $[(x_1 + \ldots + x_n) \mod 4]$ in fewer than $2.5n$ steps.

29. [M28] Prove that the text's circuit for sideways addition (27) has depth $O(\log n)$.

30. [M28] Solve the binary recurrence (28) for the cost $s(n)$ of sideways addition.

31. [21] If $f(x_1, \ldots, x_n)$ is symmetric, prove that $C(f) \leq 5n + O(n/\log n)$.

32. [HM16] Why does the solution to (30) satisfy $t(n) = 2^n + O(2n/2)$?

33. [HM22] True or false: If $1 \leq N \leq 2^n$, the first $N$ minterms of $\{x_1, \ldots, x_n\}$ can all be evaluated in $N + O(\sqrt{N})$ steps, as $n \to \infty$ and $N \to \infty$. 

bitwise operations
symmetric functions
minimum memory
Paterson
five-variable functions
top-down
bottom-up
Schroeppele
canalisizing functions
AND/OR/NOT circuits
footprint
four-variable functions
Harvard
vacuum-tube circuit
symmetric functions
full adder
minimum memory
sideways addition
binary recurrence
minterms
34. [22] A priority encoder has \( n = 2^m - 1 \) inputs \( x_1 \ldots x_n \) and \( m \) outputs \( y_1 \ldots y_m \), where \( (y_1 \ldots y_m)_2 = k \) if and only if \( k = \max \{ j \mid j = 0 \text{ or } x_j = 1 \} \). Design a priority encoder that has cost \( O(n) \) and depth \( O(m) \).

35. [23] If \( n > 1 \), show that the conjunctions \( x_1 \land \cdots \land x_{k-1} \land x_{k+1} \land \cdots \land x_n \) for \( 1 \leq k \leq n \) can all be computed from \( (x_1, \ldots, x_n) \) with total cost \( \leq 3n - 6 \).

36. [M28] (R. E. Ladner and M. J. Fischer, 1980.) Let \( y_k \) be the "prefix" \( x_1 \land \cdots \land x_k \) for \( 1 \leq k \leq n \). Clearly \( C(y_1 \ldots y_n) = n - 1 \) and \( D(y_1 \ldots y_n) = \lceil \lg n \rceil \); but we can't simultaneously minimize both cost and depth. Find a chain of optimum depth \( \lceil \lg n \rceil \) that has cost \( < 4n \).

37. [M28] (Marc Snir, 1986.) Given \( n \geq m \geq 1 \), consider the following algorithm:

S1. [Upward loop.] For \( t \leftarrow 1, 2, \ldots, \lfloor \lg m \rfloor \), set 
\[ x_{\min(m, 2^{t+1})} = x_{2t(k-1/2)} \land x_{\min(m, 2^{t+1})} \]
for \( k \geq 1 \) and \( 2^t < m < 2^{t+1} \).

S2. [Downward loop.] For \( t \leftarrow \lfloor \lg m \rfloor - 1, \lfloor \lg m \rfloor - 2, \ldots, 1 \), set 
\[ x_{2t} \land x_{2t+1} = x_{2t(1/2) - 1} \land x_{2t(1/2)} \]
for \( k \geq 1 \) and \( 2^t < m < 2^{t+1} \).

S3. [Extension.] For \( k \leftarrow m + 1, m + 2, \ldots, n \), set 
\[ x_k = x_{k-1} \land x_k \].

a) Prove that this algorithm solves the prefix problem of exercise 36. It transforms 
\( (x_1, x_2, \ldots, x_n) \) into \( (x_1, x_1 \land x_2, \ldots, x_1 \land x_2 \land \cdots \land x_n) \).

b) Let \( c(m, n) \) and \( d(m, n) \) be the cost and depth of the corresponding Boolean chain.

Prove that, if \( n \) is sufficiently large, \( c(m, n) + d(m, n) = 2n - 2 \).

c) Given \( n \), what is \( d(n) = \min_{m \leq n} d(m, n) \)? Show that \( d(n) < 2\lg n \).

d) Prove that there's a Boolean chain of cost \( 2n - 2 \) and depth \( d \) for the prefix problem whenever \( d(n) \leq d < n \). (This cost is optimum, by exercise 81.)

38. [25] In Section 5.3.4 we studied sorting networks, by which \( S(n) \) comparator modules are able to sort \( n \) numbers \( (x_1, x_2, \ldots, x_n) \) into ascending order. If the inputs \( x_j \) are 0 and 1, each comparator module is equivalent to two gates \( (x \land y, x \lor y) \); so a sorting network corresponds to a certain kind of Boolean chain, which evaluates \( n \) particular functions of \( (x_1, x_2, \ldots, x_n) \).

a) What are the \( n \) functions \( f_1f_2 \ldots f_n \) that a sorting network computes?

b) Show that those functions \( \{f_1, f_2, \ldots, f_n\} \) can be computed in \( O(n) \) steps with a chain of depth \( O(\log n) \). (Hence sorting networks aren't asymptotically optimal, Booleanwise.)

39. [M21] (M. S. Paterson and P. Klein, 1980.) Implement the \( 2^m \)-way multiplexer \( M_m(x_1, \ldots, x_m; y_0, y_1, \ldots, y_{2^m-1}) \) of (31) with an efficient chain that simultaneously establishes the upper bound \( C(M_m) \leq 2n + O(\sqrt{n}) \) and \( D(M_m) \leq m + O(\log m) \).

40. [25] If \( n \geq k \geq 1 \), let \( f_{n,k}(x_1, \ldots, x_n) \) be the "k in a row" function,
\[ (x_1 \land \cdots \land x_k) \lor (x_2 \land \cdots \land x_{k+1}) \lor \cdots \lor (x_{n+1-k} \land \cdots \land x_n) \].

Show that the cost \( C(f_{n,k}) \) of this function is less than \( 4n - 3k \).

41. [M22] (Conditional-sum adders.) One way to accomplish binary addition (25) with depth \( O(\log n) \) is based on the multiplexer trick of exercise 4: If \( (xz' + yz')_2 = (z' + y')_2 \), where \( [z'] = [y'] = [z] \), we have either \( xz + yz = (z)_2 \) and \( (x' + y')_2 = (z')_2 \), or \( xz + yz + 1 = (z)_2 \) and \( (x' + y')_2 = (z')_2 \). To save time, we can compute both \( (x + y)_2 \) and \( (z + y)_2 + 1 \) simultaneously as we compute \( (x' + y')_2 \). Afterwards, when we know whether or not the less significant part \( (x')_2 + (y')_2 \) produces a carry, we can use multiplexers to select the correct bits for the most significant part.
If this method is used recursively to build $2n$-bit adders from $n$-bit adders, how many gates are needed when $n = 2^m$? What is the corresponding depth?

42. [25] In the binary addition (25), let $u_k = x_k \land y_k$ and $v_k = x_k \lor y_k$ for $0 \leq k < n$.

a) Show that $z_k = v_k \lor c_k$, where the carry bits $c_k$ satisfy

$$c_k = u_{k-1} \lor (v_{k-2} \lor (v_{k-2} \land (\cdots (u_1 \land v_0) \cdots ))).$$

b) Let $U^k = 0$, $V^k = 1$, and $U^{k+1} = u_k \lor (v_k \land U^k)$, $V^{k+1} = v_k \land V^k$, for $k \geq j$.

Prove that $c_k = U^k$, and that $U^k = U^j \lor (V^j \land U^j)$, $V^k = V^j \land V^1$ for $i \leq j \leq k$.

c) Let $h(m) = 2^{m(m-1)/2}$. Show that when $n = h(m)$, the carries $c_1, \ldots, c_n$ can all be evaluated with depth $(m+1)n/2 \approx \lg n + \sqrt{2\lg n}$ and with total cost $O(2^m n)$.

43. [28] A finite state transducer is an abstract machine with a finite input alphabet $A$, a finite output alphabet $B$, and a finite set of internal states $Q$. One of those states, $q_0$, is called the “initial state.” Given a string $\alpha = a_1 \ldots a_n$, where each $a_j \in A$, the machine computes a string $\beta = b_1 \ldots b_n$, where each $b_j \in B$, as follows:

T1. [Initialize.] Set $j \leftarrow 1$ and $q \leftarrow q_0$.

T2. [Done?] Terminate the algorithm if $j > n$.

T3. [Output $b_j$.] Set $b_j \leftarrow c(q,a_j)$.

T4. [Advance $j$.] Set $q \leftarrow d(q,a_j)$, $j \leftarrow j + 1$, and return to step T2.

The machine has built-in instructions that specify $c(q,a) \in B$ and $d(q,a) \in Q$ for every state $q \in Q$ and every character $a \in A$. The purpose of this exercise is to show that, if the alphabets $A$ and $B$ of any finite state transducer are encoded in binary, the string $\beta$ can be computed from $\alpha$ by a Boolean chain of size $O(n)$ and depth $O(\log n)$.

a) Consider the problem of changing a binary vector $a_1 \ldots a_n$ to $b_1 \ldots b_n$ by setting

$$b_j \leftarrow a_j \oplus [a_{j-k-1} = \cdots = a_{j-k} = 1 \land a_{j-k-1} = 0],$$

assuming that $a_0 = 0$. For example, $\alpha = 11000100100011111101101010$ $\iff \beta = 1000100100010101001001010.$

Prove that this transformation can be carried out by a finite state transducer with $|A| = |B| = |Q| = 2$.

b) Suppose a finite state transducer is in state $q_j$ after reading $a_1 \ldots a_{j-1}$. Explain how to compute the sequence $q_{j+1} \ldots q_n$ with a Boolean chain of cost $O(n)$ and depth $O(\log n)$, using the construction of Ladner and Fischer in exercise 36. (From this sequence $q_1 \ldots q_n$, it is easy to compute $b_1 \ldots b_n$, since $b_j = c(q_j,a_j)$.)

c) Apply the method of (b) to the problem in (a).

44. [26] (H. E. Ladner and M. J. Fischer, 1980.) Show that the problem of binary addition (25) can be viewed as a finite state transduction. Describe the Boolean chain that results from the construction of exercise 43 when $n = 2^m$, and compare it to the conditional-sum adder of exercise 41.

45. [HM20] Why doesn’t the proof of Theorem S simply argue that the number of ways to choose $j(i)$ and $k(i)$ so that $1 \leq j(i), k(i) < i$ is $n^2(n+1)^2 \cdots (n+r-1)^2$?

46. [HM21] Let $\alpha(n) = \binom{n}{2n/n}/2^{2^n}$ be the fraction of $n$-variable Boolean functions $f(x_1, \ldots, x_n)$ for which $C(f) \leq 2^n/n$. Prove that $\alpha(n) \to 0$ rapidly as $n \to \infty$.

47. [M23] Extend Theorem S to functions with $n$ inputs and $m$ outputs.

48. [HM23] Find the smallest integer $r = r(n)$ such that $(r-1)!2^{2^n} \leq 2^{2^{n+1}}(n+r-1)^2$, (a) exactly when $1 \leq n \leq 16$; (b) asymptotically when $n \to \infty$. 
49. [HM85] Prove that, as \( n \rightarrow \infty \), almost all Boolean functions \( f(x_1, \ldots, x_n) \) have minimum formula length \( L(f) > 2^n/\log n - 2^{n+2}/(\log n)^2 \).

50. [24] What are the prime implicants and prime clauses of the prime-number function \( (35) \)? Express that function in (a) DNF (b) CNF of minimum length.

51. [20] What representation of the prime-number detector replaces \( (37) \), if rows of the truth table are based on \( x_2x_6 \) instead of \( x_1x_2 \)?

52. [23] What choices of \( k \) and \( l \) minimize the upper bound \( (38) \) when \( 5 \leq n \leq 16 \)?

53. [HM82] Estimate \( (38) \) when \( k = \lceil 2 \sqrt{n} \rceil \) and \( l = \lfloor 2^{n/3} - 3 \sqrt{n} \rfloor \) and \( n \rightarrow \infty \).

54. [29] Find a short Boolean chain to evaluate all six of the functions \( f_j(x) = \prod_{i=1}^4 x_i x_{i+1} x_{i+2} x_{i+3} \in A_j \), where \( A_1 = \{0010, 0101, 1011\} \), \( A_2 = \{0001, 1111\} \), \( A_3 = \{0111, 0111, 1101\} \), \( A_4 = \{1011, 1101\} \), \( A_5 = \{1101\} \). Compare your chain to the minimum-first evaluation scheme of Lupanov’s general method.

55. [34] Show that the cost of the 6-bit prime-detecting function is at most 14.

56. [16] Explain why all functions with 14 or more don’t-cares in Table 3 have cost 0.

57. [19] What seven-segment “digits” are displayed when \( x_1x_2x_3x_4 \geq 9 \) in \( (45) \)?

58. [30] A \( 4 \times 4 \)-bit S-box is a permutation of the 4-bit vectors \( \{0000, 0001, \ldots, 1111\} \); such permutations are used as components of well-known cryptographic systems. Every \( 4 \times 4 \)-bit S-box corresponds to a sequence of four functions \( f_1(x_1, x_2, x_3, x_4), \ldots, f_4(x_1, x_2, x_3, x_4) \), which transform \( x_1x_2x_3x_4 \mapsto f_1f_2f_3f_4 \).

Find all \( 4 \times 4 \)-bit S-boxes for which \( C(f_1) = C(f_2) = C(f_3) = C(f_4) = 7 \).

59. [29] One of the S-boxes satisfying the conditions of exercise 58 takes \( (0, \ldots, f) \mapsto (0, 6, 5, b, 3, 9, f, e, c, 4, 7, 8, d, 2, a, 1) \); in other words, the truth tables of \( (f_1, f_2, f_3, f_4) \) are respectively \( (179a, 63e8, 5b26, 3e29) \). Find a Boolean chain that evaluates these four “maximally difficult” functions in fewer than 20 steps.

60. [23] (Frank Ruskey.) Suppose \( z = (x + y) \mod 3 \), where \( x = (x_1, x_2) \), \( y = (y_1, y_2) \), \( z = (z_1, z_2) \), and each two-bit value is required to be either 00, 01, or 10. Compute \( z_1 \) and \( z_2 \) from \( x_1 \), \( x_2 \), \( y_1 \), and \( y_2 \) in \( 6 \) Boolean steps.

61. [34] Continuing exercise 60, find a good way to compute \( z = (x + y) \mod 5 \), using the three-bit values 000, 001, 010, 011, 100.

62. [HM22] Consider a random Boolean partial function of \( n \) variables that has \( 2^c \) “cares” and \( 2^d \) “don’t-cares,” where \( c + d = 1 \). Prove that the cost of almost all such partial functions exceeds \( 2^c/n \).

63. [HM35] (L. A. Sholomov, 1969.) Continuing exercise 62, prove that all such functions have cost \( \leq 2^c/n\lceil 1 + O(n^{-1} \log n) \rceil \). Hint: There is a set of \( 2^m(1 + k) \) vectors \( x_1 \ldots x_k \) that intersects every \( (k - m) \)-dimensional subspace of the \( k \)-cube.

64. [25] (Magic Fifteen.) Two players alternately select digits from 1 to 9, using no digit twice; the winner, if any, is the first to get three digits that sum to 15. What’s a good strategy for playing this game?

65. [35] Modify the tic-tac-toe strategy of \( (47) - (56) \) so that it always plays correctly.

66. [20] Criticize the moves chosen in exercise 65. Are they always optimum?
67. [40] Instead of simply finding one correct move for each position in tic-tac-toe, we might prefer to find them all. In other words, given \( x_1 \ldots x_{90} \ldots x_9 \), we could try to compute nine outputs \( g_1 \ldots g_9 \), where \( g_j = 1 \) if and only if a move into cell \( j \) is among \( X \)'s best. For example, exclamation marks indicate all of the right moves for \( X \) in the following typical positions:

\[ \begin{array}{cccccccc}
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\end{array} \]

A machine that chooses randomly among these possibilities is more fun to play against than a machine that has only one fixed strategy.

One attractive way to solve the all-good-moves problem is to use the fact that tic-tac-toe has eight symmetries. Imagine a chip that has 18 inputs \( x_1 \ldots x_{90} \ldots x_9 \) and three outputs \( (c, s, m) \), for "corner," "side," and "middle," with the property that the desired functions \( g_j \) can be computed by hooking together eight of the chips appropriately:

\[
g_1 = c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1) \\
\vee c(x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\
\end{array} \]

and \( g_5 \) is the OR of the \( m \) outputs from all eight chips.

Design such a chip, using fewer than 2000 gates.

68. [M25] Consider the \( n \)-bit \( \pi \) function \( \pi_n(x_1 \ldots x_n) \), whose value is the \( (x_1 \ldots x_n)_2 \)th bit to the right of the most significant bit in the binary representation of \( \pi \). Does the method of exercise 4.3.1-39, which describes an efficient way to compute arbitrary bits of \( \pi \), prove that \( C(\pi_n) < 2^n/n \) for sufficiently large \( n \)?

69. [M24] Let the multilinear representation of \( f \) be

\[
c_{00} x_{m} \oplus c_{001} x_{m} + c_{001} x_{j} x_{m} + c_{001} x_{j} x_{m} + c_{001} x_{j} x_{m} + c_{001} x_{j} x_{m} + c_{001} x_{j} x_{m},
\]

where each coefficient \( c_{ij} \) is a function of the variables \( \{x_1, \ldots, x_n\} \setminus \{x_i, x_j, x_m\} \).

a) Prove that the pairs \( (g_8) \) are "good" if and only if the coefficients satisfy

\[
c_{00} c_{01} = c_{011} c_{010}, \quad c_{01} c_{111} = c_{010} c_{111}, \quad \text{and} \quad c_{110} c_{110} = c_{111} c_{010}.
\]

b) For which values \( (i, j, m) \) are the pairs bad, when \( f = (\text{det } X) \mod 2 ? \) (See (60).)

70. [M27] Let \( X \) be the \( 3 \times 3 \) Boolean matrix (60). Find efficient chains for the Boolean functions (a) \( (\text{det } X) \mod 2 \); (b) \( \|X > 0 \| \); (c) \( \|\text{det } X > 0 \| \).

71. [M28] Suppose \( f(x) \) is equal to 0 with probability \( p \) at each point \( x = x_1 \ldots x_n \), independent of its value at other points.

a) What is the probability that the pairs \( (g_8) \) are good?

b) What is the probability that bad pairs \( (g_8) \) exist?

c) What is the probability that bad pairs \( (g_8) \) are found in at most \( t \) random trials?

d) What is the expected time to test case \( (i, j, m) \), as a function of \( p, t, \) and \( n \)?
72. [M24] Extend the previous exercise to the case of partial functions, where \( f(x) = 0 \) with probability \( p \), \( f(x) = 1 \) with probability \( q \), and \( f(x) = \ast \) with probability \( r \).

73. [20] If bad pairs (58) exist for all \((i, j, m)\) with \( m \neq i \neq j \), show that the indecomposability of \( f \) can be deduced after testing only \( \binom{n}{2} \) well-chosen triples \((i, j, m)\).

74. [25] Extend the idea in the previous exercise, suggesting a strategy for choosing successive triples \((i, j, m)\) when using the method of Shen, McKellar, and Weiner.

75. [26] What happens when the text’s decomposition procedure is applied to the “all-equal” function \( S_{0,n}(x_1, \ldots, x_n) \)?

76. [M25] (D. Ulbig, 1974.) The purpose of this exercise is to prove the amazing fact that, for certain functions \( f \), the best chain to evaluate the Boolean function

\[
F(u_1, \ldots, u_n, v_1, \ldots, v_n) = f(u_1, \ldots, u_n) \vee f(v_1, \ldots, v_n)
\]

costs less than \( 2C(f) \); hence functional decomposition is not always a good idea.

We let \( n = m + 2^m \) and write \( f(i_1, \ldots, i_m, x_0, \ldots, x_{2^m-1}) = f_i(x) \), where \( i \) is regarded as the number \((i_1 \ldots i_m)_{2}\). Then \((u_1, \ldots, u_n) = (i_1, \ldots, i_m, x_0, \ldots, x_{2^m-1})\), \((v_1, \ldots, v_n) = (j_1, \ldots, j_m, y_0, \ldots, y_{2^m-1})\), and \( F(u, v) = f_i(x) \vee f_j(y) \).

a) Prove that a chain of cost \( O(n \log n)^2 \) suffices to evaluate the \( 2m + 1 \) functions

\[
z_i = x \oplus ((i \leq i) \oplus [i \leq j]) \wedge (x \oplus y), \quad 0 \leq i \leq 2^m,
\]

from given vectors \( i, j, x, \) and \( y \). Each \( z_i \) is a vector of length \( 2^m \).

b) Let \( g_i(x) = f_i(x) \oplus f_{i-1}(x) \) for \( 0 \leq i \leq 2^m \), where \( f_{i-1}(x) = f_{2^m}(x) = 0 \). Estimate the cost of computing the \( 2m + 1 \) values \( c_i = g_i(z_i) \), given the vectors \( z_i \), for \( 0 \leq i \leq 2^m \).

c) Let \( c'_i = c_i \wedge ([i \leq j] \equiv [i \leq i]) \) and \( c''_i = c_i \wedge ([i \leq j] \equiv [j > i]) \). Prove that

\[
f_i(x) = c'_0 \oplus c'_1 \oplus \cdots \oplus c'_{2^m}, \quad f_j(y) = c''_0 \oplus c''_1 \oplus \cdots \oplus c''_{2^m}.
\]

b) Conclude that \( C(F) \leq 2^m/n + O(2^m \log n)/n^2 \). (When \( n \) is sufficiently large, this cost is definitely less than \( 2^{m+1}/n \), but functions \( f \) exist with \( C(f) > 2^m/n \).)

e) For clarity, write out the chain for \( F \) when \( m = 1 \) and \( f(i, x_0, x_1) = (i \wedge x_0) \vee x_1 \).

77. [35] (N. P. Red’kin, 1970.) Suppose a Boolean chain uses only the operations AND, OR, or NOT; thus, every step is either \( x = x_j(x) \wedge x_k(x) \) or \( x = x_j(x) \vee x_k(x) \) or \( x = \overline{x}_j(a) \). Prove that if such a chain computes either the “odd parity” function \( f_n(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n \) or the “even parity” function \( f_n(x_1, \ldots, x_n) = 1 \oplus x_1 \oplus \cdots \oplus \overline{x}_n \), where \( n \geq 2 \), the length of the chain is at least \( 4(n - 1) \).

78. [26] (W. J. Paul, 1977.) Let \( f(x_1, \ldots, x_m, y_0, \ldots, y_{2^m-1}) \) be any Boolean function that equals \( y_k \) whenever \( (x_1, \ldots, x_m) = k \in S \), for some given set \( S \subseteq \{0, 1, \ldots, 2^m-1\} \); we don’t care about the value of \( f \) at other points. Show that \( C(f) \geq 2|S| - 2 \) whenever \( S \) is nonempty. (In particular, when \( S = \{0, 1, \ldots, 2^m - 1\} \), the multiplexer chain of exercise 39 is asymptotically optimum.)

79. [32] (C. P. Schnorr, 1976.) Say that variables \( u \) and \( v \) are “mates” in a Boolean chain if there is exactly one simple path between them in the corresponding binary tree diagram. Two variables can be mates only if they are each used only once in the chain; but this necessary condition is not sufficient. For example, variables 2 and 4 are mates in the chain for \( S_{1,2,3} \) in Fig. 9, but they are not mates in the chain for \( S_2 \).

a) Prove that a Boolean chain on \( n \) variables with no mates has cost \( \geq 2n - 2 \).

b) Prove that \( C(f) = 2n - 3 \) when \( f \) is the all-equal function \( S_{0,n}(x_1, \ldots, x_n) \).
80. [M27] (L. J. Stockmeyer, 1977.) Another notation for symmetric functions is sometimes convenient: If \( \alpha = a_0 a_1 \ldots a_n \) is any binary string, let \( S_\alpha(x) = a_{\gamma(x)} \). For example, \( \langle x_1 x_2 x_3 \rangle = S_{001} \) and \( x_1 \oplus x_2 \oplus x_3 = S_{010} \) in this notation. Notice that \( S_\alpha(0, x_2, \ldots, x_n) = S_{\alpha'}(x_2, \ldots, x_n) \) and \( S_\alpha(1, x_2, \ldots, x_n) = S_{\alpha'}(x_2, \ldots, x_n) \), where \( \alpha' \) and \( \gamma \) stand respectively for \( \alpha \) with its last or first element deleted. Also,
\[
S_\alpha(f(x_3, \ldots, x_n), f(x_3, \ldots, x_n), x_3, \ldots, x_n) = S_{\alpha'}(x_3, \ldots, x_n)
\]
when \( f \) is any Boolean function of \( n-2 \) variables.

a) A parity function has \( a_0 \neq a_1 \neq a_2 \neq \cdots \neq a_n \). Assume that \( n \geq 2 \). Prove that if \( S_\alpha \) is not a parity function and \( S_{\alpha'} \) isn't constant, then
\[
C(S_\alpha) \geq \max(C(S_{\alpha'}), 2 + C(S_{\alpha}) + 2, \min(C(S_{\alpha'}) + 3, C(S_{\alpha}) + 3, C(S_{\alpha'}) + 5))
\]
b) What lower bounds on \( C(S_\alpha) \) and \( C(S_{\geq k}) \) follow from this result, when \( 0 \leq k \leq n \)?

81. [23] (M. Snir, 1986.) Show that any chain of cost \( c \) and depth \( d \) for the prefix problem of exercise 36 has \( c + d \geq 2n - 2 \).

82. [M23] Explain the logical sentences (62)-(70). Which of them are true?

83. [21] If there's a Boolean chain for \( f(x_1, \ldots, x_n) \) that contains \( p \) canalizing operations, show that \( C(f) < (p + 1)(n + p/2) \).

84. [M20] A monotone Boolean chain is a Boolean chain in which every operator \( \alpha \) is monotone. The length of a shortest monotone chain for \( f \) is denoted by \( C^+(f) \). If there's a monotone Boolean chain for \( f(x_1, \ldots, x_n) \) that contains \( p \) occurrences of \( \land \) and \( q \) occurrences of \( \lor \), show that \( C^+(f) < p + 1 + (n + q/2) \).

85. [M28] Let \( M_n \) be the set of all monotone functions of \( n \) variables. If \( L \) is a family of functions contained in \( M_n \), let
\[
x \sqcup y = \bigwedge \{ z \in L \mid z \supseteq x \lor y \} \quad \text{and} \quad x \sqcap y = \bigvee \{ z \in L \mid z \subseteq x \land y \}.
\]
We call \( L \) "legitimate" if it includes the constant functions \( 0 \) and \( 1 \) as well as the projection functions \( x_j \), for \( 1 \leq j \leq n \), and if \( x \sqcup y \in L \), \( x \sqcap y \in L \) whenever \( x, y \in L \).

a) When \( n = 3 \) we can write \( M_3 = \{00, 01, 03, 05, 11, 13, 15, 0f, 33, 55, 17, 1f, 37, 57, 3f, 5f, 77, 7f, ff \} \), representing each function by its hexadecimal truth table. There are \( 2^{15} \) families \( L \) such that \( \{00, 0f, 33, 55, ff \} \subseteq L \subseteq M_3 \); how many of them are legitimate?

b) If \( A \) is a subset of \( \{1, \ldots, n\} \), let \( [A] = \bigvee_{a \in A} a \); also let \( [\infty] = 1 \). Suppose \( A \) is a family of subsets of \( \{1, \ldots, n\} \) that contains all sets of size \( \leq 1 \) and is closed under intersection; in other words, \( A \cap B \in A \) whenever \( A \in A \) and \( B \in A \). Prove that the family \( L = \{ [A] \mid A \in A \cup \{\infty\} \} \) is legitimate.

c) Let \( (x_{n+1}, \ldots, x_{n+r}) \) be a monotone Boolean chain \( \alpha \). Suppose \( (\hat{x}_{n+1}, \ldots, \hat{x}_{n+r}) \) is obtained from the same Boolean chain, but with every operator \( \land \) changed to \( \lor \) and every operator \( \lor \) changed to \( \land \), with respect to some legitimate family \( L \). Prove that, for \( n + 1 \leq l \leq n + r \), we must have
\[
\hat{x}_l \subseteq x_l \lor \bigvee_{i=n+1}^{l} \{ \hat{x}_i \land (\hat{x}_{j(i)} \lor \hat{x}_{k(i)}) \mid \alpha_i = \lor \};
\]
\[
x_l \subseteq \hat{x}_l \lor \bigvee_{i=n+1}^{l} \{ \hat{x}_i \land (\hat{x}_{j(i)} \lor \hat{x}_{k(i)}) \mid \alpha_i = \land \}.\]
86. [HM37] A graph $G$ on vertices $\{1, \ldots, n\}$ can be defined by $N = \binom{n}{2}$ Boolean variables $x_{uv}$ for $1 \leq u < v \leq n$, where $x_{uv} = [u-v\in G]$. Let $f$ be the function $f(x) = |G\text{ contains a triangle}|$; for example, when $n = 4$, $f(x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34}) = (x_{12} \wedge x_{13} \wedge x_{23}) \lor (x_{12} \wedge x_{14} \wedge x_{24}) \lor (x_{13} \wedge x_{14} \wedge x_{34}) \lor (x_{23} \wedge x_{24} \wedge x_{34})$. The purpose of this exercise is to prove that the monotone complexity $C^+(f)$ is $\Omega(n/\log n)^3$.

a) If $u_j \rightarrow v_j$ for $1 \leq j \leq r$ in a graph $G$, call $S = \{(u_1, v_1), \ldots, (u_r, v_r)\}$ an $r$-family, and let $\Delta(S) = \bigcup_{1 \leq i < j \leq r} (\{u_i, u_j\} \cap \{v_i, v_j\})$ be the elements of its pairwise intersections. Say that $G$ is $r$-closed if we have $u \rightarrow v$ whenever $\Delta(S) \subseteq \{u, v\}$ for some $r$-family $S$. It is strongly $r$-closed if, in addition, we have $|\Delta(S)| \geq 2$ for all $r$-families $S$. Prove that a strongly $r$-closed graph is also strongly $(r+1)$-closed.

b) Prove that the complete bigraph $K_{m,n}$ is strongly $r$-closed when $r > \max(m, n)$.

c) Prove that a strongly $r$-closed graph has at most $(r-1)^2$ edges.

d) Let $L$ be the family of functions $\{1\} \cup \{|G| \mid G$ is a strongly $r$-closed graph on $\{1, \ldots, n\}\}$. (See exercise 85(b), we regard $G$ as a set of edges. For example, when the edges are $1 \rightarrow 3$, $1 \rightarrow 4$, $2 \rightarrow 3$, $2 \rightarrow 4$, we have $|G| = x_{13} \lor x_{14} \lor x_{23} \lor x_{24}$.) Is $L$ legitimate?

e) Let $x_{N+1}, \ldots, x_{N+p+q} = f$ be a monotone Boolean chain with $p \wedge$-steps and $q \lor$-steps, and consider the modified chain $\tilde{x}_{N+1}, \ldots, \tilde{x}_{N+p+q} = \tilde{f}$ based on the family $L$ in (d). If $\tilde{f} \neq 1$, show that $2^r(r-1)^2 + 2^q(r-1)^2(n-2) = \binom{n}{2}$. Hint: Use the second formula in exercise 85(c).

f) Furthermore, if $\tilde{f} = 1$ we must have $r^2q \geq 2^{r-1}$.

g) Therefore $p = \Omega(n/\log n)^3$. Hint: Let $r \approx 61 \log n$ and apply exercise 84.

87. [M20] Show that when nonmonotonic operations are permitted, the triangle function of exercise 86 has cost $C(f) = O(n^{0.7} (\log n)^7) = O(n^{0.81})$. Hint: A graph has a triangle if and only if the cube of its adjacency matrix has a nonzero diagonal.

88. [40] A median chain is analogous to a Boolean chain, but it uses median-of-three steps $x_i = \langle x_{j(1)}x_{j(2)}x_{j(3)} \rangle$ for $n+1 \leq i \leq n+r$, instead of the binary operations in (i). Study the optimum length, depth, and cost of median chains, for all self-dual monotone Boolean functions of 7 variables. What is the shortest chain for $x_1x_2x_3x_4x_5x_6x_7$?
SECTION 7.1.2

1. \((x_1 \lor x_4) \land x_2 \equiv (x_1 \lor x_3)\).
2. (a) \((w \oplus (x \land y)) \oplus ((x \land y) \land z)\); (b) \((w \land (x \lor y)) \land ((x \land y) \lor z)\).
3. [Doklady Akademii Nauk SSSR 115 (1957), 247-248.] Construct a \(k \times n\) matrix whose rows are the vectors \(x\) where \(f(x) = 1\). By permuting and/or complementing variables, we may assume that the top row is \(1 \ldots 1\) and that the columns are sorted. Suppose there are \(l\) distinct columns. Then \(f = g \land h\), where \(g\) is the AND of the expressions \((x_{j-1} \equiv x_j)\) over all \(1 < j \leq n\) such that column \(j - 1\) equals column \(j\), and \(h\) is the OR of \(k\) minterms of length \(l\), using one variable from each group of equal columns. For example, if \(n = 8\) and if \(f\) is 1 at the \(k = 3\) points \(11111111\), \(00011111\), \(00110111\), then \(l = 4\) and \(f(x) = (x_1 \equiv x_2) \land (x_3 \equiv x_4) \land (x_6 \equiv x_7) \land (x_7 \equiv x_8) \land ((x_1 \land x_3 \land x_5 \land x_6) \lor (x_1 \land x_3 \land x_5 \land x_6) \lor (x_1 \land x_3 \land x_5 \land x_6))\). The length of this formula in general is \(2n + (k - 2)l - 1\), and we have \(l \leq 2^{k-1}\).

Notice that, if \(k\) is large, we get shorter formulas by writing \(f(x)\) as a disjunction \(f_1(x) \lor \cdots \lor f_r(x)\), where each \(f_j\) has at most \([k/r]\) 1s. Thus
\[
L(f) \leq \min_{r \geq 1} (r - 1 + (2n + [k/r - 2]2^{k/r-1}))r.
\]

4. The first inequality is obvious, because a binary tree of depth \(d\) has at most \(1 + 2 + \cdots + 2^{d-1} = 2^d - 1\) internal nodes.

The hint follows because we can find a minimal subtree of size \(\geq \lceil r/3 \rceil\). Its size \(s\) is at most \(1 + 2\lceil r/3 \rceil - 1\). Therefore we can write \(f = (g \land f_1 \land f_0)\), where \(g\) is a subformula of size \(s\); \(f_0\) and \(f_1\) are the formulas of size \(r - s - 1\) obtained when that subformula is replaced by 0 and 1, respectively.

Let \(d(r) = \max\{D(f) : L(f) = r\}\). Since the mux function has depth 2, and since \(\max(s, r - s - 1) < \lceil 2^r/2 \rceil\), we have \(d(r) \leq 2 + d(\lceil 2^r/2 \rceil - 1)\) for \(r \geq 3\), and the result follows by induction on \(r\). [Hawaii International Conf. System Sci. 4 (1971), 525-527.]

5. Let \(g_0 = 0\), \(g_1 = x_1\), and \(g_j = x_j \land (x_{j-1} \lor g_{j-2})\) for \(j \geq 2\). Then \(F_n = g_n \lor g_{n-1}\), with cost \(2n - 2\) and depth \(n\). [These functions \(g_j\) also play a prominent role in binary addition; see exercises 42 and 44 for ways to compute them with depth \(O(\log n)\).]

6. True: Consider the cases \(y = 0\) and \(y = 1\).

7. \(\hat{x}_3 = x_1 \lor x_4, \hat{x}_6 = x_2 \land \hat{x}_5, \hat{x}_7 = x_1 \land x_3, \hat{x}_8 = \hat{x}_6 \lor \hat{x}_7\). (The original chain computes the “random” function (6); see exercise 1. The new chain computes the normalization of that function, namely its complement.)

8. The desired truth table consists of blocks of \(2^{n-k}\) 0s alternating with blocks of \(2^{n-k}\) 1s, as in (7). Therefore, if we multiply by \(2^{n-k} + 1\) we get \(x_k + (x_k \ll 2^{n-k})\), which is all 1s.

9. When finding \(L(f) = \infty\) in step L6, we can store \(g\) and \(h\) in a record associated with \(f\). Then a recursive procedure will be able to construct a minimum-length formula for \(f\) from the respective formulas for \(g\) and \(h\).

10. In step L3, use \(k = r - 1\) instead of \(k = r - 1 - j\). Also change \(L\) to \(D\) everywhere.

11. The only subtle point is that \(j\) should decrease in step U3; then we'll never have \(\phi(g) \& \phi(h) = 0\) when \(j = 0\), so all cases of cost \(r - 1\) will be discovered before we begin to look at list \(r - 1\).

U1. [Initialize.] Set \(U(0) \leftarrow \phi(0) \leftarrow 0\) and \(U(f) \leftarrow \infty\) for \(1 \leq f < 2^n - 1\). Then set \(U(x_k) \leftarrow \phi(x_k) \leftarrow 0\) and put \(x_k\) into list 0, as in step L1. Also set
7.1.2  ANSWERS TO EXERCISES  39

U(x_j \circ x_k) \leftarrow 1, set \phi(x_j \circ x_k) to an appropriate bit vector of weight 1, and
put x_j \circ x_k into list 1, for 1 \leq j < k \leq n and all five normal operators \circ.
Finally set c \leftarrow 2^n - 1 - 5(n^2) - n - 1.

U2. [Loop on r.] Do step U3 for r = 2, 3, \ldots, while c > 0.

U3. [Loop on j and k.] Do step U4 for j = [(r - 1)/2], [(r - 1)/2] - 1, \ldots, and
k = r - j - 1, while k \geq 0.

U4. [Loop on g and h.] Do step U5 for all g in list j and all h in list k; if j = k,
restrict h to functions that follow g in list k.

U5. [Loop on f.] If \phi(g) \& \phi(h) \neq 0, set u \leftarrow r - 1 and v \leftarrow \phi(g) \& \phi(h); otherwise
set u \leftarrow r and v \leftarrow \phi(g) | \phi(h). Then do step U6 for f = g \& h, f = g \& h,
f = g \& h, f = g \& h, and f = g \& h.

U6. [Update \U(f) and \phi(f).] If U(f) = \infty, set c \leftarrow c - 1, \phi(f) \leftarrow v, and put f
into list u. Otherwise if U(f) > u, set \phi(f) \leftarrow v and move f from list U(f)
to list u. Otherwise if U(f) = u, set \phi(f) \leftarrow \phi(f) | v.]

12. \quad x_1 = x_1 \oplus x_2, \quad x_2 = x_3 \wedge x_2, x_5 = x_2 \wedge z_4, x_7 = x_5 \vee x_6.

13. \quad f_0 = 01010101 (x_3); f_4 = 01110111 (x_2 \lor x_3); f_3 = 01110101 ((x_1 \wedge x_2) \lor x_3);
f_2 = 00101011 (x_1 \wedge x_2); f_1 = 00011011 ((x_1 x_2 x_3).

14. \quad For 1 \leq j \leq n, first compute t \leftarrow (g \or (g \gg 2^{n-1}) \land x_j), t \leftarrow t \oplus (t \ll 2^{n-1}),
where x_j is the truth table (11); then for 1 \leq k \leq n and k \neq j, the desired truth table
corresponding to x_j \leftarrow x_j \circ x_k is \overline{g \or (t \land ((x_j \land x_k) \oplus x_j))}.

(The 5n(n - 1) masks (x_j \circ x_k) \oplus x_j are independent of g and can be computed
in advance. The same idea applies if we allow more general computations of the form
x_j(t) \leftarrow x_k(t) \circ x(t), with \overline{5n(n - 1)} masks (x_k \circ x_j) \oplus x_j.)

15. Remarkably asymmetrical ways to compute symmetrical functions:
   \quad (a) x_1 \leftarrow x_1 \circ x_2, \quad (b) x_1 \leftarrow x_1 \circ x_2, \quad (c) x_1 \leftarrow x_1 \circ x_2,
   x_3 \leftarrow x_3 \circ x_3, \quad x_3 \leftarrow x_3 \circ x_3, \quad x_1 \leftarrow x_1 \circ x_3,
   x_1 \leftarrow x_1 \circ x_3, \quad x_3 \leftarrow x_3 \circ x_1, \quad x_4 \leftarrow x_4 \circ x_4,
   x_4 \leftarrow x_4 \circ x_4, \quad x_1 \leftarrow x_1 \circ x_4,
   x_3 \leftarrow x_3 \lor x_2, \quad x_2 \leftarrow x_2 \lor x_3, \quad x_2 \leftarrow x_2 \lor x_1,
   x_3 \leftarrow x_3 \lor x_2, \quad x_2 \leftarrow x_2 \lor x_1, \quad x_2 \leftarrow x_2 \lor x_4,
   x_3 \leftarrow x_3 \lor x_2, \quad x_2 \leftarrow x_2 \lor x_4, \quad x_2 \leftarrow x_2 \lor x_4.

16. A computation that uses only \lor and complementation produces nothing but
affine functions (see exercise 7.1.1–132). Suppose \U(x) = f(x_1, \ldots, x_n) is a non-affine
function computable in minimum memory. Then \U(x) has the form g(Ax + c) where
\quad g(y_1, y_2, \ldots, y_n) = g(y_1 \land y_2, y_2, \ldots, y_n), for some nonsingular n \times n matrix A of 0s
and 1s, where x and c are column vectors and the vector operations are performed
modulo 2; in this formula the matrix A and vector c account for all operations x_i \leftarrow
x_i \oplus x_i and/or permutations and complementations of coordinates that occur after
the most recent non-affine operation that was performed. We will exploit the fact that
\quad g(0, 0, y_2, \ldots, y_n) = g(1, 0, y_2, \ldots, y_n).

Let \alpha and \beta be the first two rows of A; also let a and b be the first two elements
of c. Then if Ax + c \equiv y (modulo 2) we will have y_2 = y_2 = 0 if and only if \alpha \cdot x \equiv a and
\beta \cdot x \equiv b. Exactly 2^n - 2 functions x satisfy this condition, and for all such vectors we
we have \U(x) = f(x \oplus w), where \quad Ax \equiv (1, 0, \ldots, 0)^T.

Let \alpha, \beta, a, b, and w, with \alpha \neq (0, \ldots, 0), \beta \neq (0, \ldots, 0), \alpha \neq \beta, and \alpha \cdot w \equiv 1
(modulo 2), there are 2^n - 2 functions f with the property that \U(x) = f(x \oplus w).
whenever \(a \cdot x \mod 2 = a\) and \(b \cdot x \mod 2 = b\). Therefore the total number of functions computable in minimum memory is at most \(2^{n+1}\) (for affine functions) plus
\[
(2^n - 1)(2^n - 2)2^n(2^{n-1})(2^{2^n-2^{n-1}}) < 2^{2^n-2^{n-2}+3n+1}.
\]

17. Let \(f(x_1, \ldots, x_n) = g(x_1, \ldots, x_{n-1}) \oplus (h(x_1, \ldots, x_{n-1}) \land x_n)\) as in 7.1.1-(16). Representing \(h\) in CNF, form the clauses one by one in \(x_0\) and AND them into \(x_n\), obtaining \(h \land x_n\). Representing \(g\) as a sum (mod 2) of conjunctions, form the successive conjunctions in \(x_0\) and OR them into \(x_n\) when ready.

(It appears to be impossible to evaluate all functions inside of \(n + 1\) registers if we disallow the non-canalizing operators \(\oplus\) and \(\equiv\). But \(n + 2\) registers clearly do suffice, even if we restrict ourselves to the single operator \(\land\).)

18. As mentioned in answer 14, we should extend the text’s definition of minimum-memory computation to allow also steps like \(x_{j(i)} \leftarrow x_{k(i)} \land x_{l(i)}\), with \(k(i) \neq j(i)\) and \(l(i) \neq j(i)\), because that will give better results for certain functions that depend on only four of the five variables. Then we find \(C_m(f) = (0, 1, \ldots, 13, 14)\) for respectively (2, 2, 5, 30, 93, 389, 1960, 10459, 47604, 135990, 198092, 123590, 21450, 472, 0) classes of functions (and 75,908 classes and 575,963,136 functions) for which \(C_m(f) = \infty\) because they cannot be evaluated at all in minimum memory. The most interesting function of that kind is probably \((x_1 \land x_2) \lor (x_2 \land x_3) \lor (x_3 \land x_4) \lor (x_4 \land x_5) \lor (x_5 \land x_1)\), which has \(C(f) = 7\) but \(C_m(f) = \infty\). Another interesting case is \(((x_1 \land x_2) \lor x_3) \land ((x_2 \land x_4) \land x_5) \land (x_1 \land x_2) \lor x_3 \lor x_4)\), for which \(C(f) = 8\) and \(C_m(f) = 13\). One way to evaluate that function in eight steps is \(x_0 = x_1 \lor x_2, x_3 = x_1 \lor x_4, x_5 = x_2 \lor x_7, x_9 = x_3 \land x_6, x_{10} = x_4 \lor x_9, x_{11} = x_5 \lor x_9, x_{12} = x_8 \land x_{10}, x_{13} = x_1 \land \bar{x}_{12}\).

19. If not, the left and right subtrees of the root must overlap, since case (i) fails. Each variable must occur at least once as a leaf, by hypothesis. At least two variables must occur at least twice, as leaves, since case (ii) fails. But we can’t have \(n + 2\) leaves with \(r \leq n + 1\) internal nodes, unless the subtrees fail to overlap.

20. Now Algorithm L (with \(f = g \oplus h\) omitted in step L5) shows that some formulas must have length 15; and even the footprint method of exercise 11 does no better than 14. To get truly minimum chains, the 25 special chains for \(r = 6\) in the text must be supplemented by five others that can no longer be ruled out, namely

\[
\begin{array}{ccccccccc}
\text{Class} & \text{Functions} & \text{Class} & \text{Functions} & \text{Class} & \text{Functions} & \text{Class} & \text{Functions} & \text{Class} & \text{Functions} \\
C_1(f) & C_2(f) & C_3(f) & C_4(f) & C_5(f) & C_6(f) & C_7(f) & C_8(f) & C_9(f)
\end{array}
\]

and when \(r = (7, 8, 9)\) we must also consider respectively (653, 12387, 225660) additional potential chains that are not special cases of the top-down and bottom-up constructions. Here are the resulting statistics, for comparison with Table 1:
7.1.2

ANSWERS TO EXERCISES

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>61</th>
<th>18984</th>
<th>7</th>
<th>45</th>
<th>15128</th>
<th>7</th>
<th>37</th>
<th>11768</th>
<th>7</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>45</td>
<td>17680</td>
<td>8</td>
<td>40</td>
<td>14296</td>
<td>8</td>
<td>27</td>
<td>10592</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>37</td>
<td>7882</td>
<td>9</td>
<td>23</td>
<td>8568</td>
<td>9</td>
<td>33</td>
<td>11536</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>320</td>
<td>10</td>
<td>28</td>
<td>5920</td>
<td>10</td>
<td>16</td>
<td>5472</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>6</td>
<td>1504</td>
<td>11</td>
<td>30</td>
<td>6304</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>5</td>
<td>576</td>
<td>12</td>
<td>3</td>
<td>900</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>3</td>
<td>144</td>
<td>13</td>
<td>8</td>
<td>1472</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>2</td>
<td>34</td>
<td>14</td>
<td>2</td>
<td>96</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>15</td>
<td>4</td>
<td>114</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The two function classes of depth 5 are represented by \( S_{2,4}(x_1, x_2, x_3, x_4) \) and \( x_1 \oplus S_2(x_2, x_3, x_4) \). And those two functions, together with \( S_2(x_1, x_2, x_3, x_4) \) and the parity function \( S_{1,3}(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \), have length 15. Also \( U_c(S_{2,4}) = U_c(S_{1,3}) = 14 \). The four classes of cost 10 are represented by \( S_{1,4}(x_1, x_2, x_3, x_4) \), \( S_{2,4}(x_1, x_2, x_3, x_4) \), \( S_4(x_1, x_2, x_3, x_4) = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \land (x_1 x_2 x_3 x_4) \), and \( \{ (x_1 x_2 x_3 x_4): x \in \{0, 1, 4, 7, 10, 13\} \} \).

21. (The authors stated that their table entries “should be regarded only as the most economical operators known to the present writers.”)

22. \( \nu(x_1 x_2 x_3 x_4 x_5) = 3 \) if and only if \( \nu(x_1 x_2 x_3 x_4) \in \{2, 3\} \) and \( \nu(x_1 x_2 x_3 x_4 x_5) \) is odd.

Similarly, \( S_2(x_1, x_2, x_3, x_4, x_5) = S_3(x_1, x_2, x_3, x_4, x_5) \) incorporates \( S_{1,2}(x_1, x_2, x_3, x_4) \):

\[
S_2 = \begin{array}{c}
1 \swarrow 2 \searrow 3 \\
\downarrow & \downarrow \\
4 & 5
\end{array}
\]

23. We need only consider the 32 normal cases, as in Fig. 9, since the complement of a symmetric function is symmetric. Then we can use reflection, like \( S_{1,2}(x) = S_{0,4}(x) \), possibly together with complementation, like \( S_{2,3,4,5}(x) = S_{0,1}(x) = S_{0,5}(x) \), to deduce most of the remaining cases. Of course \( S_i, S_{1,3,5}, \) and \( S_{1,2,3,4,5} \) trivially have cost 4.

That leaves only \( S_{1,2,3,4}(x_1, x_2, x_3, x_4, x_5) = (x_1 \oplus x_2) \lor (x_2 \oplus x_3) \lor (x_3 \oplus x_4) \lor (x_4 \oplus x_5), \) which is discussed for general \( n \) in exercise 79.

24. As noted in the text, this conjecture holds for \( n \leq 5 \).

25. It is \( 2^{n-1} - n - 1 \), the number of nontrivial normal functions. (In any normal chain of length \( r \) that doesn’t include all of these functions, \( x_j \circ x_k \) will be a new function for some \( j \) and \( k \) in the range \( 1 \leq j, k \leq n + r \) and some normal binary operator \( \circ \); so we can compute a new function with every new step, until we’ve got them all.)

26. False. For example, if \( g = S_{1,3}(x_1, x_2, x_3) \) and \( h = S_{2,3}(x_1, x_2, x_3) \), then \( C(gh) = 5 \) is the cost of a full adder; but \( f = S_{2,3}(x_0, x_1, x_2, x_3) \) has cost 6 by Fig. 9.

27. Yes: The operators \( x_2 \leftarrow x_2 \oplus x_1, x_1 \leftarrow x_1 \oplus x_3, x_1 \leftarrow x_1 \land \bar{x}_2, x_1 \leftarrow x_1 \oplus x_3, x_2 \leftarrow x_2 \oplus x_3 \) transform \( (x_1, x_2, x_3) \) into \( (x_1, x_0, x_3) \).

28. Let \( u' = u' = (v \oplus y) \cap (x \oplus y) \oplus u, u'' = ((v \oplus y) \lor (x \oplus y)) \lor u \). Thus we can set \( u_0 = 0, u_1 = x_1, u_j = ((v_{j-1} \oplus x_{j-1}) \lor (x_0 \oplus x_{j-1})) \lor u_{j-1} \) if \( j \) is odd, \( u_j = ((v_{j-1} \oplus x_{j-1}) \lor (x_0 \oplus x_{j-1})) \lor u_{j-1} \) if \( j \) is even, and \( v_{j-1} = v_{j-1} \oplus (x_0 \oplus x_{j-1}) \), obtaining \( (v_j)_{12} = (x_1 + \cdots + x_{2j+1}) \mod 4 \) for \( 1 \leq j \leq \lfloor n/2 \rfloor \). Set \( x_{n+1} = 0 \) if \( n \) is even. The function \( \{ (x_1 + \cdots + x_n) \mod 4 = 0 \} = 6_{n/2} \land 6_{n/2} \) is thereby computed in \( \lfloor 5n/2 \rfloor - 2 \) steps.
This construction is due to L. J. Stockmeyer, who proved that it is nearly optimal. In fact, the result of exercise 80 together with Figs. 9 and 10 shows that it is at most one step longer than a best possible chain, for all $n \geq 5$.

Incidentally, the analogous formula $u''' = ((v \oplus y) \land (x \oplus y)) \oplus u$ yields $(u'''')_2 = ((uv)_2 + x - y) \mod 4$. The simpler-looking function $((uv)_2 + x + y) \mod 4$ costs 6, not 5.

29. To get an upper bound, assume that each full adder or half adder increases the depth by 3. If there are $a_{jd}$ bits of weight $2^j$ and depth $3d$, we schedule at most $[a_{jd}/3]$ subsequent bits of weights $\{2^j, 2^{j+1}\}$ and depth $3(d + 1)$. It follows by induction that $a_{jd} \leq \left(\frac{3}{2}\right)^{3d} - n + 4$. Hence $a_{jd} \leq 5$ when $d \geq \log_3(n)$, and the overall depth is at most $3 \log_3(n) + 3$. (Curiously, the actual turnout turns out to be exactly 100 when $n = 10^3$.)

30. As usual, let $\nu n$ denote the sideways addition of the bits in the binary representation of $n$ itself. Then $s(n) = 5n - 2m - 3 \lfloor \log n \rfloor - 3$.

31. After sideways addition in $s(n) < 5n$ steps, an arbitrary function of $(z_{[\log n]}, \ldots, z_0)$ can be evaluated in $\sim 2n/\log n$ steps at most, by Theorem L. [See O. B. Lupanov, *Doklady Akademii Nauk SSSR* 140 (1961), 322–325.]

32. Bootstrap: First prove by induction on $n$ that $t(n) \leq 2^{n+1}$.

33. False, on a technicality: If, say, $N = \sqrt{n}$, at least $n$ steps are needed. A correct asymptotic formula $N + O(\sqrt{N}) + O(n)$ can, however, be proved by first noting that the text’s method gives $N + O(\sqrt{N})$ when $N \geq 2^{n-1}$; otherwise, if $|\log N| = n - k - 1$, we can use $O(n)$ operations to AND the quantity $z_1 \land \cdots \land z_k$ to the other variables $x_{k+1}, \ldots, x_n$, then proceed with $n$ reduced by $k$.

(One consequence is that we can compute the symmetric functions $\{S_1, S_2, \ldots, S_n\}$ with cost $s(n) + n + O(\sqrt{n}) = 6n + O(\sqrt{n})$ and depth $O(\log n)$.)

34. Say that an extended priority encoder has $n + 1 = 2^m$ inputs $x_0x_1 \ldots x_n$ and $m + 1$ outputs $y_0y_1 \ldots y_m$, where $y_0 = x_0 \lor x_1 \lor \cdots \lor x_n$. If $Q_m$ and $Q'_m$ are extended encoders for $x'_0, x'_1, \ldots, x'_m$ and $x'_0, x'_1, \ldots, x'_m$ if we define $y_0 = y_0' \lor y_1', y_1 = y_0' \land y_1', y_2 = y_0' \land y_1' \land y_2'$, and so on, then $Q_{m+1}$ works for $x'_0, x'_1, \ldots, x'_n$ in a similar way.

Starting with $m = 2$ and $y_2 = x_2 \lor (x_1 \land x_2)$, $y_3 = x_3 \lor x_2, y_0 = x_0 \lor x_1 \lor y_1$, this construction yields $P_m$ and $Q_m$ of costs $p_m$ and $q_m$, where $p_2 = 3$, $q_2 = 5$, and $p_{m+1} = 3m + p_m + q_m, q_{m+1} = 3m + 1 + 2q_m$ for $m \geq 2$. Consequently $p_m = q_m = m + 15.2^{m-2} - 3m - 4 \approx 3.75n$.

35. If $n = 2m$, compute $x_1 \land x_2, \ldots, x_{2k-1} \land x_n$, then recursively form $x_1 \land \cdots \land x_{2k-2} \land x_{2k+1} \land \cdots \land x_n$ for $1 \leq k \leq m$, and finish in $m$ more steps. If $n = 2m - 1$, use this chain for $n + 1$ elements; three steps can be eliminated by setting $x_{n+1} \leftarrow 1$. [I. Wegener, *The Complexity of Boolean Functions* (1987), exercise 3.25. The same idea can be used with any associative and commutative operator in place of $\land$.]

36. Recursively construct $P_n(x_1, \ldots, x_n)$ and $Q_n(x_1, \ldots, x_n)$ as follows, where $P_n$ has optimum depth and $Q_n$ has depth $\leq \lfloor \log n \rfloor + 1$: The case $n = 1$ is trivial; otherwise $P_n$ is obtained from $P'_n(x_1, \ldots, x_r)$ and $P'_n(x_{r+1}, \ldots, x_n)$, where $r = \lfloor n/2 \rfloor$ and $s = \lfloor n/2 \rfloor$, by setting $y_j = y'_j$ for $1 \leq j \leq r$, $y_j = y'_j \land y'_{j-r}$ for $r < j \leq n$. And $Q_n$ is obtained from either $P'_n(x_1 \land x_2, \ldots, x_{s-1} \lor x_n)$ or $P'_n(x_1 \land x_2, \ldots, x_{s-2} \land x_{s-1}, x_n)$ by setting $y_1 = x_1, y_2 = y'_j, y_{j+1} = y'_j \land y_{j-1}$ for $1 \leq j < s$, and $y_{n} = y'_j, y_{n} = y'_{j-n}$. (In doing so, we must show that output $y_n$ of $Q_n$ has depth $\lfloor \log n \rfloor$; notice that $Q_{2m+1}$ would fail if we began it with $P'_m(x_1 \land x_2, \ldots, x_{2m-1} \land x_{2m})$ instead of with $P'_{m+1}(x_1 \land x_2, \ldots, x_{2m-1} \land x_{2m}, x_{2m+1})$, except when $m$ is a power of 2.)
7.1.2 Answers to Exercises

These calculations can be performed in minimum memory, setting \( x_k(i) \leftarrow x_j(i) \land x_k(i) \) at step \( i \) for some indices \( j(i) < k(i) \). Thus we can illustrate the construction with diagrams analogous to the diagrams for sorting networks. For example,

\[
P_k = \begin{array}{c}
\text{(delay 0)} \\
\text{(delay 1)} \\
\text{(delay 2)} \\
\text{(delay 3)} \\
\text{(delay 4)} \\
\end{array}
\quad
Q_k = \begin{array}{c}
\text{(delay 0)} \\
\text{(delay 1)} \\
\text{(delay 2)} \\
\text{(delay 3)} \\
\text{(delay 4)} \\
\end{array}
\]

The costs \( p_n \) and \( q_n \) satisfy \( p_n = \lfloor n/2 \rfloor + q_{\lfloor n/2 \rfloor} + q_{\lfloor n/2 \rfloor} \) when \( n > 1 \); for example, \((p_1, \ldots, p_7) = (q_1, \ldots, q_7) = (0, 1, 2, 4, 5, 7, 9)\). Setting \( p_0 = 4n - p_n \) and \( q_0 = 3n - q_n \) leads to simpler formulas, which prove that \( p_n < 4n \) and \( q_n < 3n \). Setting \( p_0 = p_{2n} + p_n + 1, p_{4n} + 1 = p_{2n} + p_{n+1} + 1, p_{4n+2} = p_{2n+1} + p_{n+1}, p_{4n+3} = p_{2n+2} + 2 \) in particular, \( 1 + p_2 = F_{m+5} \) is a Fibonacci number.

[See JACM 27 (1980), 831–834. Slightly better chains are obtained if we use the otherwise-forbidden \( P_{n/2} \) construction for \( Q_n \) when \( n = 2^n + 1 \), if we replace \( P_k \) by \( Q_k \), and \( Q_k \), and if we then replace \( (P_0, P_10, P_{11}, P_{17}) \) by \( (Q_0, Q_{10}, Q_{11}, Q_{17}) \).]

Notice that this construction works in general if we replace \( \wedge \) by any associative operator. In particular, the sequence of prefixes \( x_1 \oplus \cdots \oplus x_k \) for \( 1 \leq k \leq n \) defines the conversion from Gray binary code to radix-2 integers, Eq. 7.2.1.1–(10).

37. The case \( m = 15, n = 16 \) is illustrated at the right.

(a) Let \( x_{i,j} \) denote the original value of \( x_i \land \cdots \land x_j \). Whenever the algorithm sets \( x_{j+1,k} \leftarrow x_j \land x_k \), one can show that the previous value of \( x_k \) was \( x_{j+1,k} \). After step S1, \( x_k \) is \( x_{j+1,k} \) where \( f(k) = k \land (k - 1) \) for \( 1 \leq k < m \) and \( f(m) = 0 \). After step S2, \( x_k \) is \( x_1,k \) for \( 1 \leq k \leq m \).

(b) The cost of S1 is \( m - 1 \), the cost of S2 is \( m - 1 - \lfloor \log m \rfloor \), and the cost of S3 is \( m - n \). The final delay of \( x_k \) is \( \lfloor \log k \rfloor + (k - 1) - 1 \) for \( 1 \leq k < m \), and it is \( \lfloor \log m \rfloor + m - k \) for \( m \leq k \leq n \). So the maximum delay for \( \{x_1, \ldots, x_{m-1}\} \) turns out to be \( g(m) = m - 1 \) for \( m < 4 \), \( g(m) = \lfloor \log m \rfloor + \lfloor \log m \rfloor \) for \( m \geq 4 \). We have \( c(m, n) = m + n - 2 - \lfloor \log m \rfloor \), \( d(m, n) = \max(g(m), \lfloor \log m \rfloor + n - m) \). Hence \( c(m, n) + d(m, n) = 2n - 2 \) whenever \( n \geq m + g(m) - \lfloor \log m \rfloor \).

(c) A table of values reveals that \( d(n) = \lfloor \log n \rfloor \) for \( n < 8 \), and \( d(n) = \lfloor \log (n - \lfloor \log n \rfloor + 3) \rfloor + \lfloor \log (n - \lfloor \log n \rfloor + 3) \rfloor - 1 \) for \( n \geq 8 \). Stating this another way, we have \( d(n) > d(n - 1) > 0 \) if and only if \( n = 2^m + k - 3 \) or \( 2^m + 2^{m-1} + k - 3 \) for some \( k > 1 \). The minimum occurs for \( m = n \) when \( n < 8 \); otherwise it occurs for \( m = n - \lfloor \log (n - \lfloor \log n \rfloor + 3) \rfloor + 2 - [n = 2^m + k - 3 \text{ for some } k] \).

(d) Set \( m \leftarrow m(n, d) \), where \( m(n, d) \) is defined in the previous sentence and \( m(n, d) = m(n - 1, d - 1) \) when \( d > d(n) \). [See J. Algorithms 7 (1986), 185–201.]

38. (a) From top to bottom, \( f_k(x_1, \ldots, x_n) \) is an elementary symmetric function also called the threshold function \( S_{2k}(x_1, \ldots, x_n) \). (See Exercise 5.3.4–28, Eq. 7.1.1–(90).)

(b) After calculating \( \{S_1, \ldots, S_k\} \) in \( \approx 6n \) steps as in answer 33, we can apply the method of exercise 37 to finish in \( 2n \) further steps.

But it is more interesting to design a Boolean circuit specifically for the computation of the \( 2^m + 1 \) threshold functions \( g_k(x_1, \ldots, x_n) = [x_1, \ldots, x_m] \geq j \) for \( 0 \leq k \leq 2^m \). Since \( [x_1] \geq [y_1] \) \( \lor \) \( ([x_1] \geq [y_2]) \land ([x_1] \geq [y_2])) \), a divide-and-conquer construction analogous to a binary decoder solves this problem with a cost at most \( 2t(m) \).
Furthermore, if $2^{m-1} \leq n < 2^m$, the cost $u(n)$ of computing $\{g_1, \ldots, g_n\}$ by this method turns out to be $2n + O(\sqrt{n})$, and it is quite reasonable when $n$ is small:

\[
n = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
u(n) = 0 1 2 4 7 8 12 15 17 19 20 21 22 27 32 34 36 36
\]

Starting with sideways addition, we can sort $n$ Boolean values in $s(n) + u(n) \approx 7n$ steps. A sorting network, which costs $2S(n)$, is better when $n = 4$ but loses when $n \geq 8$. [See 5.3.4.1 (11); D. E. Muller and F. P. Preparata, JACM 22 (1975), 195–201.]


\[
M_{r+s}(x_1, \ldots, x_r, x_{r+1}, \ldots, x_{r+s}; y_0, \ldots, y_{2^{r+s}-1}) = M_r(x_1, \ldots, x_r; y_0, \ldots, y_{2^r-1})
\]

where $y_j = \bigvee_{k=0}^{2^r-1} (y_0 \land \ldots \land y_{2^r-1})$ and $d_k$ is the $k$th output of an $s$-to-$2^s$ decoder applied to $(x_{r+1}, \ldots, x_{r+s})$, shows that $C(M_{r+s}) \leq C(M_r) + 2^{r+s} + 2^r(2^r - 1) + t(s)$, where $t(s)$ is the depth of the decoder. The depth is $D(M_{r+s}) = \max(D_x(M_r), D_y(M_r))$, where $D_x$ and $D_y$ denote the maximum depth of the $x$ and $y$ variables; we have $D_x(M_{r+s}) \leq \max(D_x(M_r), 1 + s + \lfloor \log s \rfloor + D_y(M_r))$ and $D_y(M_{r+s}) \leq 1 + s + D_x(M_r)$.

Taking $r = \lfloor m/2 \rfloor$ and $s = \lfloor m/2 \rfloor$ yields $C(M_m) \leq 2^{m+1} + O(2^{m/2})$, $D_y(M_m) \leq m + 1 + \lfloor \log m \rfloor$, and $D_x(M_m) \leq D_y(M_m) + \lfloor \log m \rfloor$.

40. We can, for example, let $f_{nk}(x) = \bigvee_{j=0}^{k-1} (l_j(x) \land r_j(x))$, where

\[
l_j(x) = \begin{cases} x_j, & \text{if } j \mod k = 0, \\
               x_j \land l_{j+1}(x), & \text{if } j \mod k \neq 0, \text{ for } 1 \leq j \leq n - (n \mod k),
\end{cases}
\]

\[
r_j(x) = \begin{cases} 1, & \text{if } j \mod k = 0, \\
               x_j \land r_{j-1}(x), & \text{if } j \mod k \neq 0, \text{ for } k \leq j \leq n.
\end{cases}
\]

The cost is $4n - 3k - 3\left\lfloor \frac{n}{k} \right\rfloor - \left\lfloor \frac{n+1}{k} \right\rfloor + 2 - (n \mod k)$.

A recursive solution is preferable when $n$ is small or $k$ is small: Observe that

\[
f_{nk}(x) = \begin{cases} x_{n-k+1} \land \cdots \land x_k \land f_{2(n-k)}(x_1, \ldots, x_{n-k}, x_{k+1}, \ldots, x_n), & \text{for } k < n < 2k;
               f_{\lfloor (n+k)/2 \rfloor}(x_1, \ldots, x_{\lfloor (n+k)/2 \rfloor}) \land \bigvee_{i=1}^{\lfloor (n+k)/2 \rfloor} f_{\lfloor (n+k)/2 \rfloor}(x_1, \ldots, x_{\lfloor (n-k)/2 \rfloor} + 1, \ldots, x_n), & \text{for } n \geq 2k.
\end{cases}
\]

The cost of this solution can be shown to equal $n - 1 + \sum_{j=1}^{n-k} \lfloor \log j \rfloor$ when $k \leq n < 2k$, and it lies asymptotically between $(m + c_k - 1)n + O(km)$ and $(m + 2 - 2/c_k)n + O(km)$ as $n \to \infty$, where $m = \lfloor \log k \rfloor$ and $1 < c_k = (k + 1)/2^m \leq 2$.

A marriage of these methods is better yet; the optimum cost is unknown.

41. Let $c(m)$ be the cost of computing both $(x) + (y) + (y)$ and $(x) + (y) + (y)$ by the conditional-sum method when $x$ and $y$ have $n = 2^m$ bits, and let $c'(m)$ be the cost of the simpler problem of computing just $(x) + (y)$. Then $c(m+1) = 2c(m) + 6 \cdot 2^m + 2$, $c'(m) = c(m) + 3 \cdot 2^m + 1$. (Bit $z_n$ of the sum costs 1; but bits $z_k$ for $n < k \leq 2n$ cost 3, because they have the form $c? a_k$: $f_k$ where $c$ is a carry bit.) If we start with $n = 1$ and $c(0) = 3$, $c'(0) = 2$, the solution is $c(m) = (3m + 5)2^m - 2$, $c'(m) = (3m + 2)2^m - m$. But improved constructions for the case $n = 2$ allow us to start with $c(1) = 11$ and $c'(1) = 7$; then the solution is $c(m) = (3m + \frac{7}{2})2^m - 2$, $c'(m) = (3m + \frac{11}{2})2^m - m + 1$. In either case the depth is $2m + 1$. [See J. Sklansky, IRE Transactions EC-9 (1960), 226–231.]
42. (a) Since \( <x_3, y_3, c_3> = u_k \lor (v_k \land c_k) \), we can use (26) and induction.

(b) Notice that \( U_{k+1}^k = u_k \) and \( V_{k+1}^k = v_k \); use induction on \( j - i \). [See A. Weinberger and J. L. Smith, IRE Transactions EC-5 (1956), 65–73; R. P. Brent and H. T. Kung, IEEE Transactions C-31 (1982), 260–264.]

(c) First, for \( l = 1, 2, \ldots, m - 1 \), and for \( 1 \leq k \leq n \), compute \( V_i^k \) for all multiples \( i \) of \( h(l) \) in the range \( k_i \geq i \geq k_{i+1} \), where \( k = h(l) \left( k - 1 \right) / h(l) \) denotes the largest multiple of \( h(l) \) that is less than \( k \). For example, when \( l = 2 \) and \( k = 99 \), we compute \( V_9^9, V_8^8 = V_9^9 \land V_8^8, V_6^8 = V_9^9 \land V_8^8, \ldots, V_4^8 = V_9^9 \land V_6^8 \); this is a prefix computation using the values \( V_9^9, V_8^8, V_6^8, \ldots, V_2^2 \) that were computed when \( l = 2 \).

Using the method of exercise 36, step \( l \) adds at most \( l \) levels to the depth, and it requires a total of \( (p_1 + p_2 + \cdots + p_{2^l}) n / 2^l = O(2^l n) \) gates.

Then, again for \( l = 1, 2, \ldots, m - 1 \), and for \( 1 \leq k \leq n \), compute \( U_i^k \) for \( i = k_{i+1} \), using the “unrolled” formula

\[
U_{k+1}^k = u_k^k \lor \bigvee_{k_i \geq j \geq k_{i+1}} (V_j^k \land U_j^k).
\]

For example, the unrolled formula when \( l = 3 \) and \( k = 99 \) is

\[
U_{64}^9 = U_{66}^9 \lor (V_{96}^9 \land U_{66}^9) \lor (V_{95}^9 \land U_{86}^9) \lor (V_{96}^8 \land U_{90}^8) \lor (V_{96}^8 \land U_{72}^8).
\]

Every such \( U_i^k \) is a union of at most \( 2^l \) terms, so it can be computed with depth \( \leq l \) in addition to the depth of each term. The total cost of this phase for \( 1 \leq k \leq n \) is \((6 + 2 + 4 + \cdots + (2^l - 2)) n / 2^l = O(2^l n)\).

The overall cost to compute all necessary \( U \)'s and \( V \)'s is therefore \( \sum_{i=1}^{m-1} O(2^l n) = O(2^m n) \). (Furthermore the quantities \( V_0^0 \) aren't actually needed, so we save the cost of \( \sum_{i=1}^{m-1} h(l) p_{2^l} \) gates.) For example, when \( m = (2,3,4,5) \) we obtain Boolean chains for the addition of \((2,8,64,1024)\)-bit numbers, respectively, with overall depths \((3,7,11,16)\) and costs \((7,64,1254,4870)\).

This construction is due to V. M. Khrapchenko, Problemy Kibernetiki 19 (1967), 107–122, who also showed how to combine it with other methods so that the overall cost will be \( O(n) \) while still achieving depth \( \lg n + O(\sqrt{\log n}) \). However, his combined method is purely of theoretical interest, because it requires \( n > 2^{64} \) before the depth becomes less than \( 2 \lg n \). Another way to achieve small depth using the recurrences in (b) can be based on the Fibonacci numbers: The Fibonacci method computes the carries with depth \( \log_2 n + O(1) \approx 1.44 \lg n \) and cost \( O(n \log n) \). For example, it yields chains for binary addition with the following characteristics:

\[
\begin{array}{cccccccccccc}
\text{n} & 4 & 8 & 16 & 32 & 64 & 128 & 256 & 512 & 1024 \\
\text{depth} & 6 & 7 & 9 & 10 & 12 & 13 & 15 & 16 & 18 \\
\text{cost} & 24 & 71 & 186 & 467 & 1125 & 2648 & 6102 & 13775 & 30861
\end{array}
\]


Charles Babbage found an ingenious mechanical solution to the analogous problem for addition in radix 10, claiming that his design would be able to add numbers of arbitrary precision in constant time; for this to work he would have needed idealized, rigid components with vanishing clearances. See H. P. Babbage, Babbage’s Calculating Engines (1889), 334–335. Curiously, an equivalent idea works fine with physical transistors, although it cannot be expressed in terms of Boolean chains; see P. M. Fenwick, Comp. J. 30 (1987), 77–79.]
43. (a) Let $A = B = Q = \{0,1\}$ and $q_0 = 0$. Define $c(q, a) = d(q, a) = \bar{q} \land a$.

(b) The key idea is to construct the functions $d_1(q) \ldots d_{n-1}(q)$, where $d_1(q) = d(q, a_1)$ and $d_{j+1}(q) = d_j(q, a_j)$. In other words, $d_1 = d^{(a_1)}$ and $d_{j+1} = d_j \circ d^{(a_j)}$, where $d^{(a)}$ is the function that takes $q \mapsto d(q, a)$ and where $\circ$ denotes composition of functions. Each function $d_j$ can be encoded in binary notation, and $\circ$ is an associative operation on these binary representations. Hence the functions $d_1 d_2 \ldots d_{n-1}$ are the prefixes $d^{(a_1)}(d^{(a_2)}(\ldots(d^{(a_{n-1})} q)\ldots))$, and $q \circ q' \ldots q_n = q_0 d_1(q_0) \ldots d_{n-1}(q_n)$.

(c) Represent a function $f(q)$ by its truth table $f_0 f_1$. Then the composition $f_0 f_1 \circ g_0 g_1$ is $h_0 h_1$, where the functions $h_0 = f_0 \cdot g_0 \cdot \text{gate}$ and $h_1 = f_1 \cdot g_1 \cdot \text{gate}$ are muxes that can each be computed with cost 3 and depth 2. (The combined cost $C(h_0 h_1)$ is only 5, but we are trying to keep the depth small.) The truth table for $d^{(a)}$ is $a0$. Using exercise 36, we can therefore compute the truth tables $d_0 d_1 d_2 d_3 \ldots d_{n-1}$ with cost $\leq 6p_{n-1} < 24n$ and depth $\leq 2\lfloor \log(n-1) \rfloor$; then $b_j = q_j \land a_j = d^{(a_j)} q_0 \land a_j$. (These cost estimates are quite conservative; substantial simplifications arise because of the 0s in the initial truth table of $d^{(a)}$ and because many of the intermediate values $d_j$ are never used. For example, when $n = 5$ the actual cost is only 10, not $6p_{4}+4 = 28$; the actual depth is 4, not $1 + 2\lfloor \log 4 \rfloor = 5$.)

44. The inputs may be regarded as the string $x_0 y_0 x_1 y_1 \ldots x_{n-1} y_{n-1}$ whose elements belong to the four-letter alphabet $A = \{00, 01, 10, 11\}$; there are two states $Q = \{0, 1\}$, representing a possible carry bit, with $q_0 = 0$; the output alphabet is $B = \{0, 1\}$, and we have $c(q, x) = q \oplus x \oplus y$, $d(q, x y) = \langle x y \rangle$. In this case, therefore, the finite state transducer is essentially described by a full adder.

Only three of the four possible functions of $q$ occur when we compose the mappings $d^{(x y)}$. We can encode them as $u \lor (q \land v)$. The initial functions $d^{(x y)}$ have $u = x \land y$, $v = x \lor y$, and the composition $(uv) \circ (u'v')$ is $u''v''$, where $u'' = u' \lor (v' \land u)$ and $v'' = v \lor u'$. When $n = 4$, for example, the chain has the following form, using the notation of exercise 42.

$U^{k+1} = x_k \land y_k$, $V^{k+1} = x_k \lor y_k$, for $0 \leq k < 4$; $U^2 = U^3 \lor (V^2 \land U^3)$,

$U^2 = U^3 \lor (V^3 \land U^3)$, $V^2 = V^3 \land V^3$; $U^3 = U^2 \lor (V^2 \land U^2)$, $U^3 = U^2 \lor (V^2 \land U^2)$;

$z_0 = V^0, z_1 = U^0 \lor V^1, z_2 = U^0 \lor V^2, z_3 = U^0 \lor V^3, z_4 = U^0$. The total cost is 20, and the maximum depth is 5.

In general the cost will be $2n + 3p_n$ in the notation of exercise 36, because we need $2n$ gates for the initial $u$’s and $v$’s, then $3p_n$ gates for the prefix computation; the $n - 1$ additional gates needed to form $z_j$ for $0 < j < n$ are compensated by the fact that we need not compute $V^j$ for $1 < j < n$. Therefore the total cost is $14 \cdot 2^n - 3F_{m+5} + 3$, clearly superior to the conditional-sum method (which has the same depth $2m + 1$):

$n = 2 \quad 4 \quad 8 \quad 16 \quad 32 \quad 64 \quad 128 \quad 256 \quad 512 \quad 1024$

cost of conditional-sum chain 7 25 74 197 492 1179 2746 6265 14072 31223

cost of Ladner–Fischer chain 7 20 52 125 286 632 1363 2888 6040 12509

[George Boole introduced his Algebra in order to show that logic can be understood in terms of arithmetic. Eventually logic became so well understood, the situation was reversed: People like Shannon and Zuse began in the 1930s to design circuits for arithmetic in terms of logic, and since then many approaches to the problem of parallel addition have been discovered. The first Boolean chains of cost $O(n)$ and depth $O(\log n)$ were devised by Yu. P. Ofman, Doklady Akademii Nauk SSSR 145 (1962), 48–51. His chains were similar to the construction above, but the depth was approximately $4m$.

45. That argument would indeed be simpler, but it wouldn’t be strong enough to prove the desired result. (Many chains with steps of fan-out 0 inflate the simpler estimate.)

composition of functions

associative

truth table

mux

multiplex operation

carry

full adder

Boole

Shannon

Zuse

parallel addition

Ofman

fan-out
7.1.2 ANSWERS TO EXERCISES

The text’s permutation-enhanced proof technique was introduced by J. E. Savage in his book *The Complexity of Computing* (New York: Wiley, 1976), Theorem 3.4.1.

46. When \( r = 2^n/n + O(1) \) we have \( \ln(2^{2r+1}(n+r-1)/r-1)) = r \ln r + (1 + \ln 4) r + O(n) = (2^n/n)(n \ln 2 - \ln n + 1 + \ln 4) + O(n) \). So \( \alpha(n) \leq (n/(4e))^{2^n/n + O(n)} \), which approaches zero quite rapidly indeed when \( n > 4e \).

(In fact, (32) gives \( \alpha(11) < 7.6 \times 10^7, \alpha(12) < 4.2 \times 10^{-6}, \alpha(13) < 1.2 \times 10^{-38} \).

47. Restrict permutations to the \((r - m)\) cases where \( kr = i \) for \( 1 \leq i \leq n \) and \((n+r-1-k)k\) is the \(k\)th output. Then we get \((r-m)!c(m,n,r) \leq 2^{2r+1}(n+r-1)^2r\) in place of (32). Hence, as in exercise 46, almost all such functions have cost exceeding \(2^n/m(n + \lg m)\) when \( m = O(2^n/n^2)\).

48. (a) Not surprisingly, this lower bound on \( C(n) \) is rather crude when \( n \) is small:

\[
\begin{align*}
n & = 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \quad 13 \quad 14 \quad 15 \quad 16 \\
r(n) & = 1 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 9 \quad 16 \quad 29 \quad 54 \quad 99 \quad 184 \quad 343 \quad 639 \quad 1196 \quad 2246 \quad 4229
\end{align*}
\]

(b) The bootstrap method (see *Concrete Mathematics* §9.4) yields

\[
r(n) = \frac{2^n}{n} \left( 1 + \frac{\ln n - 2 - 1/\ln 2}{n} + O\left(\frac{\ln n}{n^2}\right)\right).
\]

49. The number of normal Boolean functions that can be represented by a formula of length \( \leq r \) is at most \( 5^r r^{r+1} g_r \), where \( g_r \) is the number of oriented binary trees with \( r \) internal nodes. Set \( r = 2^r/\ln n - 2^{r+2}/(\ln n)^2 \) in this formula and divide by \( 2^{n-r} \) to get an upper bound on the fraction of functions with \( L(f) \leq r \). The result rapidly approaches zero, by exercise 2.3.4.4–7, because it is \( O((5a/16)^{2^{n/8}}) \) where \( a \approx 2.483 \).


50. (a) Using subcube notation as in exercise 7.1.1–30, the prime implicants are \( 00001*, (0001*1), 0100*1, 0111*1, 1010*1, 101*11, 00*011, 00*101, (01*111), 11*101, (0*1011), (1*0011), 1*1011, 0*0111, 0*0101, (*01011), (**1101), (**1110) \), where the parenthesized subcubes are omitted in a shortest DNF. (b) Similarly, the prime clauses and a shortest CNF are given by \( 00111*, 01010*, 01101*, 00100*, 00000*, 11000*, 11101*, 01000*, (1*000*), 1*001*, (1***0*), **0000*, (**1**000*, **1***0**, **1**00**, and (*****00*. (Thus the CNF is \( (x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5) \land (x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5) \land \cdots \land (x_4 \lor x_5)\).

51. \( f = ([x_5 \lor x_6 \in \{01\}] \land [(x_1 \land x_2 \land x_3 \land x_4) \in \{1,3,4,7,9,10,13,15\})] \lor ([x_5 \lor x_6 \in \{10,11\}] \land [x_1 \land x_2 = \{0000\}] \lor ([x_5 \in \{11\}] \land [(x_1 \land x_2 = \{0000\}] \lor ([x_1 \land x_2 \lor x_3] = \{1\})] \lor ([x_1 \land x_2 \lor x_3 = \{1,2,4,5,7,10,11,14\}]])\).

52. The small-\(n\) results are quite different from those that work asymptotically:

\[
\begin{array}{cccc}
n & k & l & (38) \\
5 & 2 & 2 & 39 \\
6 & 2 & 2 & 67 \\
7 & 2 & 1 & 109 \\
\end{array}
\]

\[
\begin{array}{cccc}
5 & 2 & 2 & 175 \\
6 & 2 & 2 & 279 \\
7 & 2 & 1 & 471 \\
\end{array}
\]

(Optimizations like the fact that \( [x_1 \lor x_2] = \{00,01\} \Rightarrow \delta \) usually reduce the cost further.)

53. First note that \( 2^n/l \leq n - 3 \lg n, \) hence \( m_i \leq n - 3 \lg n + l \) and \( 2^{m_i} = O(2^n/n^2) \). Also \( l = O(n) \) and \( t(n - k) = O(2^n/n^2) \). So \( \text{(38)} \) reduces to \( l \cdot 2^{n-k} + O(2^n/n^2) = 2^n/(n - 3 \lg n) + O(2^n/n^2) \).
54. The greedy-footprint heuristic gives a chain of length 15:

\[
\begin{align*}
x_0 &= x_1 \lor x_2, & x_{10} &= x_4 \lor x_5, & f_2 &= x_{15} = x_5 \land x_8, \\
x_6 &= x_2 \lor x_3, & x_{11} &= x_7 \lor x_{10}, & f_3 &= x_{16} = x_4 \land x_{12}, \\
x_7 &= x_1 \land \overline{x}_3, & x_{12} &= x_6 \land x_{11}, & f_4 &= x_{17} = x_1 \land x_8, \\
x_8 &= x_4 \land x_6, & x_{13} &= x_{10} \land x_{12}, & f_5 &= x_{18} = x_7 \land x_9, \\
x_9 &= x_4 \land \overline{x}_5, & f_1 &= x_{14} = x_6 \land \overline{x}_{11}, & f_6 &= x_{19} = \overline{x}_3 \land x_{13}.
\end{align*}
\]

The minterm-first method corresponds to a chain of length 22, after we remove steps that are never used:

\[
\begin{align*}
x_5 &= \overline{x}_1 \land \overline{x}_2, & x_{13} &= x_5 \land x_{10}, & x_{20} &= x_8 \land x_{11}, \\
x_6 &= \overline{x}_1 \land x_2, & x_{14} &= x_5 \land x_{11}, & f_6 &= x_{21} = x_{15} \lor x_{18}, \\
x_7 &= x_1 \land \overline{x}_2, & x_{15} &= x_6 \land x_9, & f_1 &= x_{22} = x_{13} \lor x_{21}, \\
x_8 &= x_1 \land x_2, & x_{16} &= x_6 \land x_{11}, & f_2 &= x_{23} = x_{12} \lor x_{20}, \\
x_9 &= \overline{x}_3 \land x_4, & x_{17} &= x_7 \land x_9, & x_{24} &= x_{14} \lor x_{16}, \\
x_{10} &= x_3 \land x_4, & x_{18} &= x_7 \land x_{11}, & f_3 &= x_{25} = x_{24} \lor x_{19}, \\
x_{11} &= x_3 \land x_4, & f_5 &= x_{19} = x_8 \land x_9, & f_4 &= x_{26} = x_{17} \lor x_{20}, \\
x_{12} &= x_5 \land x_9.
\end{align*}
\]

(The distributive law could replace the computation of \(x_{14}, x_{16}\), and \(x_{24}\) by two steps.) Incidentally, the three functions in the answer to exercise 51 can be computed in only ten steps:

\[
\begin{align*}
x_5 &= x_2 \lor x_4, & f_3 &= x_9 = x_6 \lor x_8, & x_{12} &= x_2 \lor x_3, \\
x_6 &= \overline{x}_1 \land x_5, & x_{10} &= x_1 \land x_8, & x_{13} &= \overline{x}_1 \land x_{12}, \\
x_7 &= x_2 \land x_4, & \overline{f}_2 &= x_{11} = x_9 \lor x_{10}, & f_1 &= x_{14} = x_4 \lor x_{13}, \\
x_8 &= x_3 \land \overline{x}_7.
\end{align*}
\]

55. The optimum two-level DNF and CNF representations in answer 50 cost 53 and 43, respectively. Formula (37) costs 30, when optimized as in exercise 54. The alternative in exercise 51 costs only 17. But the catalog of optimum five-variable chains suggests

\[
\begin{align*}
x_7 &= \overline{x}_1 \land x_2, & x_{11} &= x_5 \land x_{10}, & x_{15} &= x_{13} \lor x_{14}, & x_{18} &= \overline{x}_4 \land x_{17}, \\
x_8 &= x_3 \land x_7, & x_{12} &= x_5 \lor x_{10}, & x_{16} &= x_5 \land \overline{x}_{10}, & x_{19} &= x_6 \land x_{15}, \\
x_9 &= x_2 \land x_8, & x_{13} &= x_4 \land \overline{x}_{11}, & x_{17} &= \overline{x}_3 \land x_{16}, & x_{20} &= x_{18} \lor x_{19}, \\
x_{10} &= x_1 \land x_9, & x_{14} &= x_7 \land x_{12}.
\end{align*}
\]

for this six-variable function. Is there a better way?

56. If we care about at most two values, the function can be either constant or \(x_j\) or \(\bar{x}_j\).

57. The truth tables for \(x_5\) through \(x_{17}\), in hexadecimal notation, are respectively \(0ff\), \(2222\), \(33cc\), \(0dd\), \(7777\), \(55dd\), \(3ec\), \(6b94\), \(4914\), \(4804\), \(060b\), \(2020\), \(7007\). So we get

\[
\begin{align*}
1010 &= \circ, & 1011 &= \bullet, & 1012 &= \bar{f}, & 1013 &= f, & 1014 &= \bar{u}, & 1015 &= u.
\end{align*}
\]

58. The truth tables of all cost-7 functions with exactly eight 1s in their truth tables are equivalent to either \(0779\), \(169b\), or \(179a\). Combining these in all possible ways yields 9656 solutions that are distinct under permutation and/or complementation of \(\{x_1, x_2, x_3, x_4\}\) as well as under permutation and/or complementation of \(\{f_1, f_2, f_3, f_4\}\).
59. The greedy-footprint heuristic produces the following 17-step chain:

\[
\begin{align*}
\varphi_0 &= \varphi_1 \lor \varphi_2, \\
\varphi_1 &= \varphi_9 \lor \varphi_7, \\
\varphi_2 &= \varphi_7 \lor \varphi_6, \\
\varphi_3 &= \varphi_8 \lor \varphi_5, \\
\varphi_4 &= \varphi_4, \\
\varphi_5 &= \varphi_6 \lor \varphi_3, \\
\varphi_6 &= \varphi_7 \lor \varphi_2, \\
\varphi_7 &= \varphi_8 \lor \varphi_1, \\
\varphi_8 &= \varphi_9, \\
\varphi_9 &= \varphi_9.
\end{align*}
\]

The initial functions all have large footprints, so we can’t achieve \( C(f_1 f_2 f_3 f_4) = 28 \); but a slightly more difficult S-box probably does exist.

60. One way is \( u_1 = x_1 \oplus y_1, u_2 = x_2 \oplus y_2, v_1 = y_2 \oplus u_1, v_2 = y_1 \oplus u_2, z_1 = v_1 \oplus u_2, z_2 = v_2 \oplus u_1 \).

61. Viewing these partial functions of six variables as \( 4 \times 16 \) truth tables, with rows governed by \( x_1 y_1 \), our knowledge of 4-bit functions suggests good ways to compute the rows and leads to the following 25-step solution: \( t_1 = x_2 \land y_2, t_2 = x_3 \land y_3, t_3 = x_2 \lor y_2, t_4 = x_3 \lor y_3, t_5 = x_2 \lor y_2, t_6 = x_3 \lor y_3, t_7 = x_1 \land y_1, t_8 = x_1 \land y_1, t_9 = x_1 \land y_1, t_{10} = x_1 \land y_1, t_{11} = x_1 \land y_1, t_{12} = x_1 \land y_1, t_{13} = x_1 \land y_1, t_{14} = x_1 \land y_1, t_{15} = x_1 \land y_1, t_{16} = x_1 \land y_1, t_{17} = x_1 \land y_1, t_{18} = x_1 \land y_1, t_{19} = x_1 \land y_1, t_{20} = x_1 \land y_1, t_{21} = x_1 \land y_1, t_{22} = x_1 \land y_1, z_1 = x_1 \land y_1, z_2 = x_1 \land y_1, z_3 = x_1 \land y_1. \) (Is there a better way?) Gilbert Lee has found a 17-step solution if the inputs are represented by 000, 001, 011, 101, and 111.

62. There are \( \binom{2^n}{2^k} \) such functions, at most \( \binom{2^n}{2^k} \) of which have cost \( \leq r \). So we can argue as in exercise 46 to conclude from (32) that the fraction with cost \( \leq r = \frac{2^n}{2^k} \) is at most \( \frac{2^{k+1}}{2^k (n + r - 1)^2} (r - 1) \leq 2^{-r \ln n + \Theta(r)}. \)

63. [Problem 16.21] Put the truth table in a \( 2^k \times 2^{n-k} \) array as in Lupanov’s method, and suppose there are \( c_j \) cares in column \( j \), for \( 0 \leq j < 2^{n-k} \). Break that column into \( \lfloor c_j / m \rfloor \) subcolumns that each have \( m \) cares, plus a possibly empty subcolumn at the bottom that contains fewer than \( m \) of them. The hint tells us that at most \( 2^{n+k} \) column vectors suffice to match the 0s and 1s of every subcolumn that has a specified top row \( i_0 \) and bottom row \( i_1 \). With \( O(m 2^{n+k}) \) operations we can therefore construct \( O(2^{n+3k}) \) functions \( g(x_1, \ldots, x_k) \) from the minterms of \( \{ x_1, \ldots, x_k \} \), so that every subcolumn matches some type \( t \). And for every type \( t \) we can construct functions \( h_t(x_{k+1}, \ldots, x_n) \) from the minterms of \( \{ x_{k+1}, \ldots, x_n \} \), specifying the columns that match \( t \); the cost is at most \( \frac{1}{2} \sum_t \lfloor c_j / m \rfloor + 1 \leq 2^n c/m + 2^{n-k} \). Finally, \( f = \bigvee_t (g_t \land h_t) \) requires \( O(2^{n+3k}) \) additional steps. Choosing \( k = \lfloor 2 \ln n \rfloor \) and \( m = \lfloor n - 9 \ln n \rfloor \) makes the total cost at most \( O(2^n/n)(1 + 9 n^{-1} \ln n + O(n^{-2})) \).

Of course we need to prove the hint, which is due to E. N. Nechiporuk [Doklady Akad. Nauk SSSR 163 (1965), 40–42]. In fact, \( 2^{n+1}[1 + (k \ln 2)] \) vectors suffice (see S. K. Stein, J. Combinatorial Theory A16 (1974), 391–397): If we choose \( q = 2^{m+k} \) vectors at random, not necessarily distinct, the expected number of untouched subcubes is \( \binom{2^m}{2^k} (1 - 2^{-m})^2 < \binom{2^m}{2^k} e^{-c2^{-m}} < 2^{2m} \). (An explicit construction would be nicer.)

For extensive generalizations—tolerating a percentage of errors and specifying the density of 1s—we see N. Pippenger, Mathematical Systems Theory 10 (1977), 129–167.

64. It’s exactly the game of tic-tac-toe, if we number the cells \( \square \) as in an ancient Chinese magic square. Berlekamp, Conway, and Guy use this numbering scheme to present a complete analysis of tic-tac-toe in their book Winning Ways 3 (2003), 732–736.

65. One solution is to replace the “defending” moves \( d_j \) by “attacking” moves \( a_j \) and “counterattacking” moves \( c_j \), and to include them only for corner cells \( j \in \{1,3,9,7\} \).
Let \( j \cdot k = (j k) \mod 10 \); then
\[
\begin{align*}
&j \cdot 1 \quad j \cdot 2 \quad j \cdot 3 \\
&j \cdot 4 \quad j \cdot 5 \quad j \cdot 6 \\
&j \cdot 7 \quad j \cdot 8 \quad j \cdot 9
\end{align*}
\]
gives us another way to look at the tic-tac-toe diagram, when \( j \) is a corner, because \( j \perp 10 \). The precise definition of \( a_j \) and \( c_j \) is then
\[
a_j = m_j \land ((x_{j3} \land \beta_{(j, 3)}(j, 0) \land (a_{j, 3} \oplus o_{j, 3})) \lor (x_{j, 7} \land \beta_{(j, 7)}(j, 0) \land (a_{j, 2} \oplus o_{j, 2}))) \\
\lor (m_{j, 9} \land ((x_{j, 8} \land \theta_{(j, 8)}(j, 0) \land (a_{j, 8} \oplus o_{j, 8})) \lor (m_{j, 8} \land x_{j, 4} \land (a_{j, 7} \oplus o_{j, 7}))))
\]
\[
c_j = d_j \land (x_{j, 3} \land a_{j, 7} \lor (x_{j, 8} \land a_{j, 3})) \land d_{j, 9};
\]
here \( d_j = m_j \land \beta_{(j, 2)}(j, 3) \land \beta_{(j, 6)}(j, 7) \) takes the place of (51). We also define
\[
u = (a_1 \oplus o_3) \oplus (x_7 \oplus x_9), \\
z_j = \begin{cases} 
  m_j \land \bar{r}, & \text{if } j = 5, \\
  m_j \land \bar{d}_{j, 9}, & \text{if } j \in \{1, 3, 9, 7\}, \\
  m_j, & \text{if } j \in \{2, 6, 8, 4\},
\end{cases}
\]
in order to cover a few more exceptional cases. Finally the sequence of rank-ordered moves \( d_0 d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 m_1 m_2 m_3 m_4 \) in (53) is replaced by the sequence \( a_1 a_2 a_3 a_7 a_9 c_1 c_2 c_3 c_7 c_9 z_2 z_3 z_7 z_2 z_6 z_8 z_4 \); and we replace \((d_j \land \bar{d}_j) \lor (m_j \land \bar{m}_j)\) in (55) by \((a_j \land \bar{a}_j) \lor (c_j \land \bar{c}_j) \lor (z_j \land \bar{z}_j)\) when \( j \) is a corner cell, otherwise simply by \((z_j \land \bar{z}_j)\).

(Notice that this machine is required to move correctly from all legal positions, even when those positions couldn’t arise after the machine had made \( X \)’s earlier moves. We essentially allow humans to play the game until they ask the machine for advice. Otherwise great simplifications would be possible. For example, if \( X \) always goes first, it could grab the center cell and eliminate a huge number of future possibilities; fewer than \( 8 \times 6 \times 4 = 384 \) games could arise. Even if \( O \) goes first, there are fewer than \( 9 \times 7 \times 5 \times 3 = 945 \) possible scenarios against a fixed strategy. In fact, the actual number of different games with the strategy defined here turns out to be 76 + 457, of which 72 + 328 are won by the machine and the rest belong to the cat.)

66. The Boolean chain in the previous answer fulfills its mission of making correct moves from all 4520 legal positions, where correctness was essentially defined to mean that the worst-case final outcome is maximized. But a truly great tic-tac-toe player would do things differently. For example, from position \( \text{X} \) the machine takes the center, \( \Box \), and \( O \) probably draws by playing in a corner. But moving to \( \text{X} \) would give \( O \) only two chances to avoid defeat. [See Martin Gardner, *Hexaflexagons and Other Mathematical Diversions*, Chapter 4.]

Furthermore the best move from a position like \( \Box \) is to \( \text{X} \) instead of winning immediately; then if the reply is \( \text{X} \) move to \( \text{X} \). That way you still win, but without humiliating your opponent so badly.

Finally, even the concept of a single “best move” is flawed, because a good player will choose different moves in different games (as Babbage observed).

*It might be thought that programing a digital computer to play ticktacktoe, or designing special circuits for a ticktacktoe machine, would be simple. This is true unless your aim is to construct a master robot that will win the maximum number of games against inexperienced players.*

67. The author's best effort, with 1734 gates, was constructed by adapting the method of Sholomov in answer 63: First divide the truth tables into 64 rows for \( a_5 x_5 \) and 4096 columns for the other 12 input variables. Then place appropriate Is into "care" positions, in such a way that the columns have relatively few Is. Then find a small number of column types that match the cares in all columns; 23 types suffice for the \( c \) function, 20 types for \( s \), and 6 for \( m \). We can then compute each output as \( \sqrt{(g_1 \land h_2)} \), sharing much of the work of the minterm calculations within \( g_1 \) and \( h_2 \).

[This exercise was inspired by a discussion in John Wakerly's book Digital Design (Prentice-Hall, 3rd edition, 2000), §6.2.7. Incidentally, Babbage planned to choose among \( k \) possible moves by looking at \( N \mod k \), where \( N \) was the number of games won so far; he didn't realize that successive moves would tend to be highly correlated until \( N \) changed. Much better would have been to let \( N \) be the number of moves made so far.]

68. No. That method yields a "uniform" chain with a comprehensible structure, but its cost is \( 2^n \) times a polynomial in \( n \). A circuit with approximately \( 2^n/n \) gates, constructed by Theorem L, exists but is more difficult to fabricate. (Incidentally, \( C(\pi) = 10 \).)

69. (a) One can, for example, verify this result by trying all 64 cases.

(b) If \( x_m \) lies in the same row or column as \( x_i \), and also in the same row or column as \( x_j \), we have \( a_{11} = a_{10} = a_{10} = 0 \), so the pairs are good. Otherwise there are essentially three different possibilities, all bad: If \( (i, j, m) = (1, 2, 4) \) then \( a_{10} = 0 \), \( a_{10} = x_2 x_3 \oplus x_2 x_3 \), \( a_{10} = x_3 \); if \( (i, j, m) = (1, 2, 6) \) then \( a_{10} = x_2 x_3 \), \( a_{10} = x_2 x_3 \); if \( (i, j, m) = (1, 5, 9) \) then \( a_{11} = 1 \), \( a_{10} = 0 \), \( a_{10} = x_3 x_7 \).

70. (a) \( x_1 \land ((x_5 \land x_9) \lor (x_8 \land x_6)) \lor x_9 \land ((x_4 \land x_1) \lor x_3 \land ((x_4 \land x_3) \land (x_5 \land x_7))).

(b) \( x_1 \land ((x_5 \land x_9) \lor (x_8 \land x_6)) \lor x_9 \land ((x_4 \land x_1) \lor x_3 \land ((x_4 \land x_3) \land (x_5 \land x_7))).

(c) Let \( y_1 = x_1 \land x_5 \land x_9, y_2 = x_1 \land x_6 \land x_8, y_3 = x_2 \land x_3 \land x_7, y_4 = x_2 \land x_4 \land x_9, y_5 = x_3 \land x_4 \land x_8, y_6 = x_3 \land x_5 \land x_7 \). The function \( f(y_1, \ldots, y_6) = \lnot y_1 + y_2 + y_3 + y_4 + y_5 + y_6 \) can be evaluated in 15 further steps with two full adders and a comparator; but there is a 14-step solution: Let \( z_1 = (y_1 \lor y_2) \land y_3, z_2 = (y_1 \lor y_2) \land (y_3 \lor y_4), z_3 = (y_4 \lor y_5) \land y_6 \). The function \( f(y_1, \ldots, y_6) = \lnot y_1 + y_2 + y_3 + y_4 + y_5 + y_6 \) can be evaluated in 15 further steps with two full adders and a comparator; but there is a 14-step solution: Let \( z_1 = (y_1 \lor y_2) \land y_3, z_2 = (y_1 \lor y_2) \land (y_3 \lor y_4), z_3 = (y_4 \lor y_5) \land y_6 \). Then \( f = (z_1 \land (z_2 \land (z_3 \lor z_4)))) \land (z_5 \lor z_4) \). Furthermore \( y_1 y_2 y_3 = 111 \iff y_4 y_5 y_6 = 111 \), so there are don't-cares, leading to an 11-step solution: \( f = ((z_1 \land z_2) \lor z_2) \). The total cost is \( 12 + 11 = 23 \).

(There's no way by which a computer could discover such an efficient chain in a reasonable amount of time, given only the truth table of \( f \). But perhaps an even better chain exists.)

71. (a) \( P(p) = 1 - 12p^3 + 24p^4 + 12p^5 + 96p^6 - 96p^7 + 24p^8 \), which is \( \frac{11}{32} + \frac{9p^3 - 3p^4 - 24p^5 + 24p^6}{p^7} \) when \( p = \frac{1}{2} + x \).

(b) There are \( N = 2^{2^n} \) sets of eight values \( (f_0, \ldots, f_7) \), each of which yields good pairs with probability \( P(p) \). So the answer is \( 1 - P(p)^N \).

(c) The probability is \( \binom{N}{t} P(p)^t (1 - P(p))^{N-t} \) that exactly \( t \) sets succeed; and in such a case \( t \) trials will find good pairs with probability \( (r/N)^t \). The answer is therefore \( 1 - \sum_{t=0}^{N} \binom{N}{t} P(p)^t (1 - P(p))^{N-t} (r/N)^t = 1 - P(p) + O(t^2/N) \).

(d) \( \sum_{t=0}^{N} \binom{N}{t} P(p)^t (1 - P(p))^{N-t} \sum_{i=0}^{r} (r/N)^t = (1 - P(p)^t)/(1 - P(p)) + O(t^2/N) \).

72. The probability in exercise 71(a) becomes \( P(p) = (12p^3 - 24p^4 + 12p^5 - 96p^6 + 96p^7 - 12p^8) + (60p^2 - 240p^3 + 456p^4 - 432p^5 + 144p^6) r^2 + (48p^2 - 216p^3 + 432p^5 + 144p^6) r^3 + (36p^2 + 24p^3 + 12p^5) r^4 + (48p^2 - 24p^3 + 12p^5) r^5 - 12p^2 r^6 \). If \( p = q = (1 - r)/2 \), this is \( (1 + 48p^2 + 36p^2 + 144p^3 - 30p^4 + 336p^5 - 348p^6 + 144p^7 - 21p^8)/32 \); for example, it's \( 7739/8192 \approx 0.94 \) when \( r = 1/2 \).
73. Consider the Horn clauses $1 \land 2 \Rightarrow 3, 1 \land 3 \Rightarrow 4, \ldots, 1 \land (n - 1) \Rightarrow n, 1 \land n \Rightarrow 2,$ and $i \land j \Rightarrow 1$ for $1 < i < j \leq n$. Suppose $|Z| > 1$ in a decomposition, and let $i$ be minimum such that $x_i \in Z$. Also let $j$ be minimum such that $j > i$ and $x_j \in Z$. We cannot have $i > 1$, since $i \land j \Rightarrow 1$ in that case. Thus $i = 1$, and $x_j \in Z$ for $2 \leq j \leq n$.

74. Suppose we know that no nontrivial decomposition exists with $x_1 \in Z$ or $\cdots$ or $x_{i-1} \in Z$; initially $i = 1$. We hope to rule out $x_i \in Z$ too, by choosing $j$ and $m$ cleverly. The Horn clauses $i \land j \Rightarrow m$ reduce to Krom clauses $j \Rightarrow m$ when $i$ is asserted. So we essentially want to use Tarjan's depth-first search for strong components, in a digraph with arcs $j \Rightarrow m$ that may or may not exist.

When exploring from vertex $j$, first try $m = 1, \ldots, m = i - 1$; if any such implication $i \land j \Rightarrow m$ succeeds, we can eliminate $j$ and all its predecessors from the digraph for $i$. Otherwise, test unexplored vertices $m$. Otherwise test vertices that have already been seen, favoring those near the root of the depth-first tree.

In the example $f(x) = (\det X) \mod 2$, we would successively find $1 \land 2 \neq 3, 1 \land 2 \Rightarrow 4, 1 \land 4 \Rightarrow 3, 1 \land 3 \Rightarrow 5, 1 \land 5 \Rightarrow 6, 1 \land 6 \Rightarrow 7, 1 \land 7 \Rightarrow 8, 1 \land 8 \Rightarrow 9, 1 \land 9 \Rightarrow 2$ (now $i \leftarrow 2$); $2 \land 3 \neq 1, 2 \land 3 \Rightarrow 4, 2 \land 4 \Rightarrow 5, 2 \land 4 \Rightarrow 6, 2 \land 6 \Rightarrow 1$ (now 3, 4, and 6 are eliminated from the digraph for 2), $2 \land 5 \Rightarrow 1$ (and 5 is eliminated), $2 \land 7 \neq 1, 2 \land 7 \Rightarrow 3$ (7 is eliminated), $2 \land 8 \Rightarrow 1, 2 \land 9 \Rightarrow 1$ (now $i \leftarrow 3$); $3 \land 4 \neq 1, 3 \land 4 \Rightarrow 2, 3 \land 5 \Rightarrow 1$, etc.

75. This function is 1 at only two points, which are complementary. So it is indecomposable yet the pairs $(g_8)$ are never bad when $n > 3$. Every partition $(Y, Z)$ will therefore be a candidate for decomposition.

Similarly, if $f$ is decomposable with respect to $(Y, Z)$, the indecomposable function $f(x) \oplus S_0(x)$ will act essentially like $f$ in the tests. (A method to deal with approximately decomposable functions should probably be provided in a general-purpose decomposability test.)

76. (a) Let $a_i = [i \geq n]$ for $0 \leq i \leq 2^n$. The cost is $\leq 2t(m)$, as observed in answer 38(b); and in fact, the cost can be reduced to $2^{m+1} - 2m - 2$ with $\Theta(m)$ depth. Furthermore the function $[i \leq j] = (t_1 \land j_1) \lor ((i_1 \equiv j_1) \land [i_2 \cdots i_m \leq j_2 \cdots j_m])$ can be evaluated with $4m-3$ gates. After computing $x \lor y$, each $z_i$ costs $2^{m+1} + 1 = O(n/\log n)$.

(b) Here the cost is at most $C(g_0) + \cdots + C(g_{2^n}) \leq (2^n + 1)(2^{2^n}/(2^n + O(m)))$ by Theorem L, because each $g_i$ is a function of $2^n$ inputs.

(c) If $i \leq j$ we have $z_i = x$ for $i \leq i$ and $z_i = y$ for $i > i$; hence $f_i(x) = c_0 \oplus \cdots \oplus c_i$ and $f_j(y) = c_{j+1} \oplus \cdots \oplus c_{2^n}$. If $i > j$ we have $z_i = y$ for $i \leq i$ and $z_i = x$ for $i > i$; hence $f_j(y) = c_0 \oplus \cdots \oplus c_i$ and $f_i(x) = c_{i+1} \oplus \cdots \oplus c_{2^n}$.

(d) The functions $b_i = [j \leq l]$ can be computed for $0 \leq j \leq 2^n$ in $O(2^n)$ steps, as in (a). So we can compute $F$ from $(c_0, \ldots, c_{2^n})$ with $O(2^n)$ further steps. Step (b) therefore dominates the cost, for large $m$.

(e) $a_0 = 1, a_1 = i, a_2 = 0, b_0 = 0, b_1 = j, b_2 = 1; d = [i \leq j] = r \lor j; m_i = a_i \oplus d, z_{i0} = x_0 \oplus (m_i \land (x_0 \oplus y_0)), z_{i1} = x_1 \lor (m_i \land (x_1 \lor y_1)), \text{ for } l = 0, 1, 2; c_0 = z_{i0}; c_1 = z_{i1} \land \delta_1; c_2 = x_{i2} \lor \delta_2; c_3 = c_1 \land (d \equiv a_1), c_4 = c_1 \land (d \equiv b_1), \text{ for } l = 0, 1, 2; \text{ and finally } F = (d \lor c_3 \lor c_4) \lor (c_0 \lor c_3 \lor c_4)$.

The net cost (29 after obvious simplifications) is, of course, outrageous in such a small example. But one wonders if a state-of-the-art automatic optimizer would be able to reduce this chain to just 5 gates.

77. Given a shortest such chain for \( f_n \) or \( \overline{f}_n \), let \( U_l = \{ i \mid l = j(i) \) or \( l = k(i) \} \) be the “uses” of \( x_l \), and let \( u_i = |U_l| \). Let \( t_i = 1 \) if \( x_i = x_{j(i)} \) or \( x_{k(i)} \), otherwise \( t_i = 0 \). We will show that there’s a chain of length \( \leq r - 4 \) that computes either \( f_{n-1} \) or \( \overline{f}_{n-1} \), by using the following idea: If variable \( x_m \) is set to 0 or 1, for any \( m \), we can obtain a chain for \( f_{n-1} \) or \( \overline{f}_{n-1} \) by deleting all steps of \( U_m \) and modifying other steps appropriately. Furthermore, if \( x_i = x_{j(i)} \) or \( x_{k(i)} \) is known to equal \( t_i \) when \( x_m \) has been set to 0 or 1, then we can also delete the steps \( U_k \) (Throughout this argument, the letter \( m \) will stand for an index in the range \( 1 \leq m \leq n \).

Case 1: \( u_m = 1 \) for some \( m \). This case cannot occur in a shortest chain. For if the only use of \( x_m \) is \( x_i = x_m \), eliminating this step would change \( f_n \leftrightarrow \overline{f}_n \), and otherwise we could set the values of \( x_1, \ldots, x_{m-1}, x_{m+1}, \ldots, x_n \) to make \( x_i \) independent of \( x_m \), contradicting \( x_{n+r} = f_n \) or \( \overline{f}_n \). Thus every variable must be used at least twice.

Case 2: \( x_i = \bar{x}_m \) for some \( i \) and \( m \), where \( u_m > 1 \). Then \( x_i = x_i \land x_k \) for some \( i \) and \( k \), and we can set \( x_m \leftarrow t_i \) to make \( x_i \) independent of \( x_k \). Eliminating steps \( U_m \), \( U_i \), and \( U_k \) then removes at least 3 steps, except when \( u_i = u_k = 1 \) and \( u_m = 2 \) and \( x_i = \overline{x}_m \) or \( \overline{x}_i \); but in that case we can also eliminate \( U_i \).

Case 3: \( u_m \geq 3 \) for some \( m \), and not Case 2. If \( i, j, k \in U_m \) and \( i < j < k \), set \( x_m \leftarrow t_k \) and remove steps \( i, j, k, U_k \).

Case 4: \( u_i = u_2 = \cdots = u_n = 2 \), and not Case 2. We may assume that the first step is \( x_1 = x_1 \land x_2 \), and that \( x_k = x_i \land x_k \) for some \( k < l \).

Case 4.1: \( k > 0 \). Then \( k > 1 \). If \( u_k = 1 \), set \( x_k \leftarrow t_i \) and remove steps \( 1, k, l, U_i \). Otherwise set \( x_2 \leftarrow t_i \); this forces \( x_k = \overline{t}_i \), and we can remove steps \( 1, k, l, U_k \).

Case 4.2: \( x_i = x_i \land x_m \). Then we must have \( m = 2 \); for if \( m > 2 \) we could set \( x_2 \leftarrow t_1 \), \( x_m \leftarrow t_1 \), and make \( x_k \) independent of \( x_1 \). Hence we may assume that \( x_1 = x_1 \land x_2, x_2 = x_1 \lor x_2 \). Setting \( x_1 \leftarrow 0 \) allows us to remove \( U_0 \) and \( U_1 \); setting \( x_1 \leftarrow 1 \) allows us to remove \( U_0 \) and \( U_2 \). Thus we’re done unless \( u_k = u_2 = 1 \).

If \( x_2 = \bar{x}_i \), set \( x_i \leftarrow 0 \) and remove \( 1, 2, p, U_p \); if \( x_2 = x_2 \), set \( x_1 \leftarrow 1 \) and remove \( 1, 2, q, U_q \). Otherwise \( x_p = x_1 \land x_u \) and \( x_q = x_2 \lor x_v \), where \( x_u \) and \( x_v \) do not depend on \( x_1 \) or \( x_2 \). But that’s impossible; it would allow us to set \( x_3, \ldots, x_n \) to make \( x_u = t_p \), then \( x_2 \leftarrow 1 \) to make \( x_k \) independent of \( x_1 \).

[Problem 2 in Kibernetiki 23 (1970), 83–101; 28 (1974), 4. With similar proofs, Red’kin showed that the shortest AND-OR-NOT chains for the functions ‘\( x_1 \ldots x_n < y_1 \ldots y_n \)’ have lengths \( 5n - 3 \) and \( 5n - 1 \), respectively.]

78. [SICOMP 6 (1977), 427–430.] Say that \( y_k \) is active if \( k \in S \). We may assume that the chain is normal and that \( |S| > 1 \); the proof is like Red’kin’s in answer 77:

Case 1: Some active \( y_k \) is used more than once. Setting \( y_k \leftarrow 0 \) saves at least two steps and yields a chain for a function with \( |S| - 1 \) active values.

Case 2: Some active \( y_k \) appears only in an AND gate. Setting \( y_k \leftarrow 0 \) eliminates at least two steps, unless this AND is the final step. But it can’t be the final step, because \( y_k = 0 \) makes the result independent of every other active \( y_j \).

Case 3: Like Case 2 but with an OR or NOT-BUT or BUT-NOT gate. Setting \( y_k \leftarrow c \) for some appropriate constant \( c \) has the desired effect.

Case 4: Like Case 2 but with XOR. The gate can’t be final, since the result should be independent of \( y_k \) when \( x_1 \ldots x_m \) addresses a different active value \( y_j \). So we can eliminate two steps by setting \( y_k \) to the function defined by the other input to XOR.

79. (a) Suppose the cost is \( r < 2n - 2 \); then \( n > 1 \). If each variable is used exactly once, two leaves must be nodes. Therefore some variable is used at least twice. Pruning it away produces a chain of cost \( \leq r - 2 \) on \( n - 1 \) variables, having no mates.
(Incidentally, the cost is at least $2n-1$ if every variable is used at least twice, because at least $2n$ uses of variables must be connected together in the chain.)

(b) Notice that $S_{0,n} = \bigwedge u \in \{x_1, \ldots, x_n\}$ whenever the edges $u \iff v$ form a free tree on $\{x_1, \ldots, x_n\}$. So there are many ways to achieve cost $2n-3$.

Any chain of cost $r < 2n-3$ must have $n > 2$ and must contain mates $u$ and $v$. By renaming and possibly complementing intermediate results, we can assume that $u = 1$, $v = 2$, and that $f(x_1, \ldots, x_n) = g(x_1 \circ h(x_3, \ldots, x_n), x_2, \ldots, x_n)$, where $\circ$ is $\land$ or $\lor$.

Case 1: $\circ$ is AND. We must have $h(0, \ldots, 0) = h(1, \ldots, 1) = 1$, for otherwise $f(x_1, x_2, y, \ldots, y)$ wouldn’t depend on $x_1$. Therefore $f(x_1, \ldots, x_n) = h(x_3, \ldots, x_n) \land g(x_1, x_2, \ldots, x_n)$ can be computed by a chain of the same cost in which 1 and 2 are mates and in which the path between them has gotten shorter.

Case 2: $\circ$ is XOR. Then $f = f_0 \lor f_1$, where $f_0(x_1, \ldots, x_n) = (x_1 \equiv h(x_3, \ldots, x_n)) \land g(0, x_2, \ldots, x_n)$ and $f_1(x_1, \ldots, x_n) = (x_1 \oplus h(x_3, \ldots, x_n)) \land g(1, x_2, \ldots, x_n)$. But $f = S_{0,n}$ has only two prime implicants; so there are only four possibilities:

Case 2a: $f_0 = f$. Then we can replace $x_1 \oplus h$ by 0, to get a chain of cost $\leq r - 2$ for the function $g(0, x_2, \ldots, x_n) = S_{0,n-1}(x_2, \ldots, x_n)$.

Case 2b: $f_1 = f$. Similar to Case 2a.

Case 2c: $f_0(x) = x_1 \land \cdots \land x_n$ and $f_1(x) = x_1 \land \cdots \land x_n$. In this case we must have $g(0, x_2, \ldots, x_n) = x_2 \land \cdots \land x_n$ and $g(1, x_2, \ldots, x_n) = x_2 \land \cdots \land x_n$. Replacing $h$ by 1 therefore yields a chain that computes $f$ in $< r$ steps.

Case 2d: $f_0(x) = x_1 \land \cdots \land x_n$ and $f_1(x) = x_1 \land \cdots \land x_n$. Similar to Case 2c.

Applying these reductions repeatedly will lead to a contradiction. Similarly, one can show that $C(S_{0,n}) = 2n - 2$. [Theoretical Computer Science 1 (1976), 289-295.]

80. [Mathematical Systems Theory 10 (1977), 323-336.] Without loss of generality, $a_0 = 0$ and the chain is normal. Define $U_i$ and $u_i$ as in answer 77. We may assume by symmetry that $u_i = \max(u_1, \ldots, u_n)$.

We must have $u_1 \geq 2$. For if $u_1 = 1$, we could assume further that $x_{n+1} = x_1 \circ x_2$; hence two of the three functions $S_n(0,0, x_3, \ldots, x_n) = S_n'$, $S_n(0,1, x_3, \ldots, x_n) = S_n'$, $S_n(1,1, x_3, \ldots, x_n) = S_n'$ would be equal. But then $S_n$ would be a parity function, or $S_n'$, would be constant.

Therefore setting $x_1 = 0$ allows us to eliminate the gates of $U_1$, giving a chain for $S_n'$ with at least 2 fewer gates. It follows that $C(S_n) \geq C(S_n') + 2$. Similarly, setting $x_1 = 1$ proves that $C(S_n) \geq C(S_n') + 2$.

Three cases arise when we explore the situation further:

Case 1: $u_1 \geq 3$. Setting $x_1 = 0$ proves that $C(S_n) \geq C(S_n') + 3$.

Case 2: $U_1 = \{i, j\}$ and operator $o_i$ is cananlizing (namely, AND, BUT-NOT, NOT-BUT, or OR). Setting $x_1$ to an appropriate constant forces the value of $x_j$ and allows us to eliminate $U_1 \cup U_j$; notice that $i \notin U_j$ in an optimum chain. So either $C(S_n) \geq C(S_n') + 3$ or $C(S_n) \geq C(S_n') + 3$.

Case 3: $U_1 = \{i, j\}$ and $o_i = o_j = \oplus$. We may assume that $x_1 = x_1 \oplus x_2$ and $x_j = x_1 \oplus x_2$. If $u_1 = 1$ and $x_1 = x_j \oplus x_2$, we can restructure the chain by letting $x_j = x_2 \oplus x_2$, $x_1 = x_1 \oplus x_2$; therefore we can assume that either $u_j = 1$ or $x_1 = x_j \oplus x_2$ for some canalisng operator $o$. If $U_2 = \{i, j\}$, we can assume similarly that $x_j = x_2 \oplus x_2$ and that either $u_j = 1$ or $x_2 = x_j \oplus x_2$ for some canalizing operator $o'$. Furthermore we can assume by symmetry that $x_j$ does not depend on $x_j$.

If $x_2$ does not depend on $x_i$, let $f(x_3, \ldots, x_n) = x_2$; otherwise let $f(x_3, \ldots, x_n)$ be the value of $x_2$ when $x_i = 1$. By setting $x_1 = f(x_3, \ldots, x_n)$ and $x_2 = f(x_3, \ldots, x_n)$, or vice versa, we make $x_1$ and $x_2$ constant, and we obtain a chain for the nonconstant
function $S_{u,i}$. We can, in fact, ensure that $x_i$ is constant in the case $w_j = 1$. We claim that at least five gates of this chain (including $x_1$ and $x_{j_1}$) can be eliminated, hence $C(S_a) \geq C(S_{u,i}) + 5$. The claim is clearly true if $|U_i \cup U_j| \geq 3$.

We must have $|U_i \cup U_j| \geq 1$. Otherwise we'd have $p = i$, and $x_k$ would not depend on $x_i$, so $S_a$ would be independent of $x_1$ with our choice of $x_2$. Therefore $|U_i \cup U_j| \geq 2$.

Case 3a: $U_j \neq \emptyset$. Then $x_i$ is constant; we can eliminate $x_1$, $x_j$, and $U_i \cup U_j$. If the latter set contains only two elements, then $x_q = x_i \circ x_j$ is also constant and we eliminate $U_q$. Since $S_{u,i}$ isn’t constant, we want to eliminate the output gate.

Case 3b: $U_i \subseteq U_j$, $|U_j| = 2$. Then $x_q = x_i \circ x_j$ for some $q$; we can eliminate $x_1$, $x_j$, and $U_i \cup U_q$. The claim has been proved.

(b) By induction, $C(S_k) \geq 2n + \min(k,n - k) - 3 - \lfloor n = 2k\rfloor$, for $0 < k < n$; $C(S_{\geq k}) \geq 2n + \min(k,n + 1 - k) - 4$, for $1 < k < n$. The easy cases are $C(S_0) = C(S_1) = C(S_{\geq 2}) = n - 1$; $C(S_{\leq 2}) = n - 2$. For $k \geq 3$, $C(S_k) = C(S_{k-1}) + C(S_{k-2})$, where the bound is optimum for $k = \lfloor n/2\rfloor$ when $n \leq 5$. All known results are consistent with the conjecture that $C(S_k) = n - k$ for $k \geq n/2$.

81. If some variable is used more than once, we can set it to a constant, decreasing $n$ by 1 and decreasing $c$ by 2. Otherwise the first operation must involve $x_1$, because $y_1 = x_1$ is the only output that doesn’t need computation; making $x_1$ constant decreases $n$ by 1, $c$ by 1, and $d$ by 1. [J. Algorithms 7 (1986), 185-201.]

82. (62) is false. (63) reads, “For all numbers $m$ there’s a number $n$ such that $m < n + 1$; it is true because we can take $m = n$.

(64) fails when $n = 0$ or $n = 1$, because the numbers in these formulas are required to be nonnegative integers.

(65) says that, if $b$ exceeds $a$ by 2 or more, there’s a number $ab$ between them. Of course it’s true, because we can let $ab = a + 1$.

(66) was explained in the text, and it too is true. Notice that ‘$\land$’ takes precedence over ‘$\lor$’ and ‘$\leq$’ takes precedence over ‘$\geq$’ just as ‘$+$’ takes precedence over ‘$\geq$’ in (65); these conventions reduce the need for parentheses in sentences of $L$.

(67) says that, if $A$ contains at least one element $n$, it must contain a minimum element $m$ (an element that’s less than or equal to all of its elements). True.

(68) is similar, but $m$ is now a maximum element. Again true, because all sets are assumed to be finite.

(69) asks for a set $P$ with the property that $[0 \in P] = [3 \not\in P]$, $[1 \in P] = [4 \not\in P]$, 

$\ldots, [999 \in P] = [1000 \not\in P]$, $[1000 \in P] = [1003 \not\in P]$, $[1001 \in P] \neq [1004 \not\in P]$, etc. It’s true if (and only if) $P = \{x \mid x \text{ mod } 6 \in \{1, 2, 3\} \text{ and } 0 \leq x < 1000\}.$

Finally, the subformula $\forall m (n \in C \Rightarrow n + 1 \in C)$ in (70) is another way of saying that $C = \emptyset$, because $C$ is finite. Hence the parenthesized formula $\forall A \forall B$ is a tricky way to say that $A = \emptyset$ and $B \neq \emptyset$. (Stockmeyer and Meyer used this trick to abbreviate statements in $L$ that involve long subformulas more than once.) Statement (70) is true because an empty set doesn’t equal a nonempty set.

83. We can assume that the chain is normal. Let the canalizing steps be $y_1, \ldots, y_p$. Then $y_k = \alpha_k \circ \beta_k$ and $f = \alpha_{p+1}$, where $\alpha_k$ and $\beta_k$ are $\oplus$’s of some subsets of $\{x_1, \ldots, x_n, y_1, \ldots, y_p\}$; at most $n + k - 2 \oplus$’s are needed to compute them, combining common terms first. Hence $C(f) \leq p + \sum_{k=1}^{p+1} (n + k - 2) = (p + 1)(n + p/2) - 1$.

84. Argue as in the previous answer, with $\lor$ or $\land$ in place of $\oplus$. [N. Alon and R. B. Boppana, Combinatorica 7 (1987), 15-16.]
85. (a) A simple computer program shows that 13744 are legitimate and 19024 aren’t. (An illegitimate family of this kind has at least 8 members; one such is \( \{00, 0f, 33, 55, \quad 15, 3f, 77\} \). Indeed, if the functions \( x_1 \lor x_2 \) \((3f)\), \( x_1 \lor x_3 \) \((77)\), and \( (x_1 \lor x_2) \land x_3 \) \((15)\) are present in a legitimate family \( L \), then \( x_2 \cup 15 = 33 \mid 15 = 37 \) must also be in \( L \).

(b) The projection and constant functions are obviously present. Define \( A^* = \bigcap \{B \mid B \supseteq A \land B \in A\} \), or \( A^* = \infty \) if no such set \( B \) exists. Then we have \( |A| \cap |B| = |A \cap B| \) and \( |A| \cup |B| = |(A \cup B)^*| \).

(c) Abbreviate the formulas as \( x_i \subseteq x_i \lor \bigvee_{i=1}^{n} \delta_i \), \( x_i \subseteq x_i \lor \bigvee_{i=1}^{n} \epsilon_i \), and argue by induction: If step \( i \) is an AND step, \( x_i = x_j \land x_k \subseteq x_j \lor \bigvee_{i=1}^{n} \delta_i \land (x_k \lor \bigvee_{i=1}^{n} \epsilon_i) = x_j \lor \bigvee_{i=1}^{n} \delta_i \land x_k \lor \bigvee_{i=1}^{n} \epsilon_i = (x_j \lor \bigvee_{i=1}^{n} \delta_i) \lor (x_k \lor \bigvee_{i=1}^{n} \epsilon_i) \). Argue similarly if step \( i \) is an OR step.

86. (a) If \( S \) is an \( r \)-family contained in the \((r+1)\)-family \( S' \), clearly \( \Delta(S) \subseteq \Delta(S') \).

(b) By the pigeonhole principle, \( \Delta(S) \) contains elements \( u \) and \( v \) of each part, whenever \( S \) is an \( r \)-family. And if \( \Delta(S) = \{u, v\} \), we certainly have \( u \not\rightarrow v \).

(c) The result is obvious when \( r = 1 \). There are at most \( r - 1 \) edges containing any given vertex \( u \), by the “strong” property. And if \( u \not\rightarrow v \), the edges \( \{u, v\} \) are strongly \((r - 1)\)-closed; so there are at most \( (r - 2)^2 \) of them, by induction. Thus there are at most \( 1 + 2(r - 2) + (r - 2)^2 \) edges altogether.

(d) Yes, by exercise 85(b), if \( r > 1 \), because strongly \( r \)-closed graphs are closed under intersection. All graphs with \( \leq 1 \) edges are strongly \( r \)-closed when \( r > 1 \), because they have no \( r \)-families containing distinct edges.

(e) There are \( 3 \) triangles \( x_{ij} \land x_{ik} \land x_{jk} \), only \( n - 2 \) of which are contained in any term \( x_{uv} \) of \( f \). Hence the minterms for at most \((r - 1)^2(n - 2) \) triangles are contained in \( f \), and the others must be contained in one of the functions \( \epsilon_i = x_i \lor (x_j \land x_k) \). Some term has the form \( (G \setminus H) \lor (G \lor H) \), where \( G \) and \( H \) are strongly \( r \)-closed; we will prove that \( T \) contains at most \( 2(r - 1)^3 \) triangles.

A triangle \( x_{ij} \land x_{ik} \land x_{jk} \) in \( T \) involves some variable (say \( x_{ij} \)) of \( G \) and some variable (say \( x_{uv} \)) of \( H \), but no variable of \( G \lor H \). There are at most \((r - 1)^2 \) choices for \( ij \); and then there are at most \( 2(r - 1) \) choices for \( k \), since \( H \) has at most \( r - 1 \) edges touching \( i \) and at most \( r - 1 \) edges touching \( j \).

(f) There are \( 2^n - r^2 \) complete bigraphs obtained by coloring 1 red, coloring other vertices either yellow or blue, and letting \( u \not\rightarrow v \) if and only if \( u \) and \( v \) have opposite colors. By the first formula in exercise 85(c), the minterms \( B \) for every such graph must be contained in one of the terms \( \delta_i = x_i \lor (x_j \land x_k) \). (For example, if \( n = 4 \) and vertices \( (2, 3, 4) \) are red, blue, blue), then \( B = x_{12} \land x_{13} \land x_{14} \land x_{23} \land x_{24} \land x_{34} \).) A minterm \( B \) is contained in \( T \) if and only if, in the coloring for \( B \), some edge of \((G \lor H)^* \) has vertices of opposite colors, but all edges of \( G \lor H \) are monochromatic. We will prove that \( T \) includes at most \( 2^n - r^2 \) such \( B \).

Let \( G \) be any graph, and \( T = G^* \lor G \). The following (inefficient) algorithm can be used to find \( G^* \)’s: If there’s an \( r \)-family \( S \) with \( |\Delta(S)| < 2 \), stop with \( G^* = \infty \). Otherwise, if \( \Delta(S) = \{u, v\} \), add the edge \( u \not\rightarrow v \) to \( G \) and repeat.

At most \( 2^n - r^2 \) bipartite minterms \( B \) have monochromatic \( \{u_j, v_j\} \) for \( 1 \leq j \leq r \) when \( |\Delta(S)| < 2 \). And when \( \Delta(S) = \{u, v\} \) there are \( 2^n - r^2 \) with monochromatic \( \{u_j, v_j\} \) and bichromatic \( \{u, v\} \). So we want to show that the algorithm for \( G \) takes fewer than \( 2r^2 \) iterations when \( G \) is strongly \( r \)-closed.

For \( k \geq 1 \), let \( u_k \not\rightarrow v_k \) be the first new edge added to \( G \) that is disjoint from \( \{u_j, v_j\} \) for \( 1 \leq j < k \). At most \( r \) such edges exist, by “strongness”; and each of them
is followed by at most $2r - 3$ new edges that touch $u_j$ or $v_j$. So the total number of steps to find $G'$ is at most $r(2r - 2) + 1 < 2r^2$.

Exercise 84 tells us that $q < \left( \frac{2}{5} \right) + (p + 1)\left( \frac{3}{5} \right)$. Thus we have either $2(r - 1)^3 p \geq (\binom{n}{2} - (r - 1)^2(n - 2)$ or $(\frac{2}{5}) + (p + 1)\left( \frac{3}{5} \right) > 2r - r^2$. Both lower bounds for $p$ are

$$\frac{1}{12} \left( \frac{n}{6 \log n} \right)^3 \left( 1 + O \left( \frac{\log \log n}{\log n} \right) \right) \quad \text{when} \quad r = \left\lfloor \frac{n^6}{186624(\log n)^4} \right\rfloor.$$  

[Noga Alon and Ravi B. Boppana, *Combinatorica* 7 (1987), 1–22, proceeded in this way to prove, among other things, the lower bound $\Omega(n/\log n)^2$ for the number of $\land$s in any monotone chain that decides whether or not $G$ has a clique of fixed size $s \geq 3$.]

87. The entries of $X^3$ are at most $n^2$ when $X$ is a $0$–$1$ matrix. A Boolean chain with $O(n^{\log_2 7}(\log n)^2)$ gates can implement Strassen’s matrix multiplication algorithm 4.6.4–(36), on integers modulo $2^{[\log n^3]} + 1$.

88. There are $1,422,564$ such functions, in $716$ classes with respect to permutation of variables. Algorithm L and the other methods of this section extend readily to ternary operations, and we obtain the following results for optimum median-only computation:

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
C(f) & C_m(f) & L(f) & D(f) \\
\hline
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 35 & 35 & 35 & 35 & 35 \\
2 & 2 & 360 & 360 & 360 & 360 & 360 \\
3 & 39 & 3885 & 3885 & 3885 & 3885 & 3885 \\
4 & 48 & 42483 & 42483 & 42483 & 42483 & 42483 \\
5 & 201 & 406945 & 406945 & 406945 & 406945 & 406945 \\
6 & 353 & 709866 & 709866 & 709866 & 709866 & 709866 \\
7 & 99 & 169891 & 169891 & 169891 & 169891 & 169891 \\
8 & 2 & 252 & 252 & 252 & 252 & 252 \\
9 & 0 & 9 & 9 & 9 & 9 & 9 \\
10 & 0 & 10 & 10 & 10 & 10 & 10 \\
11 & 0 & 113 & 113 & 113 & 113 & 113 \\
\hline
\end{array}
\]

S. Amarel, G. E. Cooke, and R. O. Winder [IEEE Trans. EC-13 (1964), 4–13, Fig. 5b] conjectured that the 9-operation formula

\[
\langle x_1 x_2 x_3 x_4 x_5 x_6 x_7 \rangle = \langle x_1 (\langle x_2 x_3 x_4 \langle x_5 x_6 x_7 \rangle \rangle) (\langle x_5 x_6 x_7 \rangle) (\langle x_4 x_5 x_7 \rangle) \rangle
\]

is the best way to compute medians of 7 via medians of 3. But the “magic” formula

\[
\langle x_1 (\langle x_2 x_3 x_4 \langle x_5 x_6 x_7 \rangle \rangle) (\langle x_4 x_5 x_7 \rangle) (\langle x_3 x_5 x_7 \rangle) \rangle
\]

needs only 8 operations, and in fact the shortest chain needs just seven steps:

\[
\langle x_1 x_2 x_3 x_4 x_5 x_6 x_7 \rangle = \langle x_1 (\langle x_2 x_5 x_6 \rangle) (\langle x_3 x_4 x_7 \rangle) \rangle (\langle x_5 x_6 x_7 \rangle) (\langle x_4 x_5 x_7 \rangle) \rangle
\]

The interesting function $f(x_1, \ldots, x_7) = (x_1 \land x_2 \land x_3) \lor (x_2 \land x_3 \land x_4) \lor (x_4 \land x_5 \land x_7) \lor (x_5 \land x_6 \land x_7) \lor (x_6 \land x_7 \land x_2) \lor (x_7 \land x_1 \land x_2)$, whose prime implicants correspond to the projective plane with 7 points, is the toughest of all: Its minimum length $L(f) = 11$ and minimum depth $D(f) = 4$ are achieved by the remarkable formula

\[
\langle (\langle x_1 x_4 x_5 x_6 \rangle) (\langle x_3 x_6 \langle x_1 x_2 x_3 x_7 \rangle \rangle) (\langle x_2 x_7 x_1 \langle x_5 x_3 x_7 \rangle) \rangle
\]

And the following even more astonishing chain computes it optimally:

\[
x_8 = \langle x_1 x_2 x_3 \rangle, \quad x_9 = \langle x_1 x_4 x_6 \rangle, \quad x_{10} = \langle x_1 x_5 x_8 \rangle, \quad x_{11} = \langle x_2 x_7 x_9 \rangle, \quad x_{12} = \langle x_3 x_9 x_{10} \rangle, \quad x_{13} = \langle x_4 x_5 x_{12} \rangle, \quad x_{14} = \langle x_6 x_{11} x_{12} \rangle, \quad x_{15} = \langle x_7 x_{13} x_{14} \rangle.
\]
INDEX AND GLOSSARY

When an index entry refers to a page containing a relevant exercise, see also the answer to that exercise for further information. An answer page is not indexed here unless it refers to a topic not included in the statement of the exercise.

0–1 matrices, 29, 34, 57.
0-preserving functions, see Normal
    Boolean functions.
2^n-way multiplexer, 13, 31.
3-variable functions, 8–9.
4-variable functions, 2–9, 16–18, 26, 30, 33.
5-variable functions, v, 9–10, 30, 48.
\nu x (sideways sum), v, 12, 30, 42, 44.
\pi (circle ratio), as “random” example, 2, 22, 32, 38.
\pi function, 2, 22, 29, 34, 38.

Addition, binary, 11–12, 31–32.
Addition modulo 3 and 5, 33.
Addition modulo 4, 30.
Adjacency matrix, 27, 37.
Affine functions, 39–40.
Aiken, Howard Hathaway, 8.
All-equal function \( (S_0, n) \), 35.
Alon, Noga \( (\nu k \text{n} \text{n}) \), 55, 57.
Amarel, Saul, 57.
Analysis of algorithms, 34.
AND gates \( (\wedge) \), 1.
with vacuum tubes, 8.
AND-OR-NOT chains, 35, see also
    Canalizing chains.
Approximately decomposable functions, 52.
Ashenburt, Robert Lovett, 21, 24.
Associative operators, 42, 43, 46.
Asymptotic methods, 13–16, 32–33.

Babbage, Charles, 20, 45, 50, 51.
Babbage, Henry Moore, 45.
Backward-computation principle, 6.
Bad pairs, 22–24, 34–35.
Bell Telephone Laboratories, 19.
Berlekamp, Elwyn Ralph, 49.
Bigraph: A bipartite graph.
Binary addition, 11–12, 31–32.
Binary comparison function, 24, 51–53.
Binary decoder, 13, 43.
Binary recurrences, 12, 13, 30, 43.
Binary trees, 1, 2, 47.
Binary-coded decimal digits, 18.
Bipartite graphs, 24, 37, 51.
Bipartite matching, 29.
Bitwise operations, 4–6, 30, 39, 56.
Blum, Norbert Karl, 26.
Boole, George, 46.

Boolean chains, 0–57.
    AND-OR, 29, 36–37.
    AND-OR-NOT, 30, 35, 47.
    Canalizing, 30, 36, 47.
    Definition of, 0.
    Monotone, 29, 36–37.
    of 3 variables, 8–9.
    of 4 variables, 2–9, 16–18, 26, 30, 33.
    of 5 variables, v, 9–10, 30, 48.
    of many variables, 13–16, 21–37.
    Optimization of, 25–26, 32.
    with several outputs, 11–13, 16–21, 25–26, 30–34.
Bootstrapping, 42, 47.
Boppama, Ravi Babu, 55, 57.
Bores, Endre, 24.
Bottom-up synthesis, 7–9, 30, 40.
Brayton, Robert King, 26.
Breit, Richard Peirce, 45.
Büchi, Julius Richard, 28.
BUT-NOT gates \( (\oplus) \), 1, 4, 14.

\( C(f) \), 1, 15–16, see Cost of a Boolean
    function.
\( C(f_1 \ldots f_m) \), 11.
\( C^+(f) \), 36, 37.
\( C_m(f) \), 6–7, 30, 57.
Canalizing chains, 30, 36, 47.
Canalizing operators, 27, 30, 40, 54.
Care, 33, 51, see also Don’t-cares.
Carry bits, 11, 31, 32, 46.
Cat’s game, 19, 21, 50.
Circuits, Boolean, 1, see Boolean chains.
Clique, 57.
Closure under intersection, 36.
CNF: Conjunctive normal form, 1, 40, 48.
    Coloring of graphs, 24–25.
    Combinatorial complexity, 1, 15–16, see
    Cost of a Boolean function.
Comparator modules, 31.
Comparison function, binary, 24, 51–53.
Complement of a Boolean function, 3–4, 11, 41.

Complete bipartite graphs, 37, 56.
Composition of functions, 46.
Conditional-sum adders, 31–32.
Consecutive Is, 31, 32.
Control grids, 8.
Conway, John Horton, 49.
Cooke, George Erskine, 57.
Cost of a Boolean function, 1, 11, 15–16, 30–36.
    Statistics, 5, 9, 40–41.
INDEX AND GLOSSARY  59

Curtis, Herbert Allen, 0, 24-25.

\(D(f)\), 3, see Depth of a Boolean function.

De Micheli, Giovanni, 26.

Decoder, binary, 13, 43.

Decomposition of functions, 21-25, 34-36.

Decomposition of partial functions, 24-25.

Depth of a Boolean function, 3, 4, 28-29, 36.

Statistics, 5, 9, 40-41.

Depth-first search, 52.

Determinants, 25, 34.

Diagonalization, 28.

Disjoint decomposition, 21-24.

Distributive law, 29.

Divide and conquer, 13, 42-45.

DNP: Disjunctive normal form, 1, 48.

Don’t-cares, 18, 20, 24-26, 33, 35, 51.

Electrical engineers, 0, 1, 9, 11, 13, 18, 26.

Elgot, Calvin Creston, 28.

Evaluation of Boolean functions, 0-57, see Boolean chains.

Fan-in: The number of inputs to a gate, 1, 8, 28.

Fan-out: The number of uses of a gate, 1, 8, 46.

Fenwick, Peter McAlaway, 45.

Fibonacci, Leonardo, of Pisa (= Leonardo
filo Bonacì Pisano), numbers, 43, 45, threshold functions, 29.

Finikov, Boris Ivanovich (Фиников, 
Борис Иванович), 29.

Finite state transducers, 32.

Fischer, Michael John, 31, 32.

Five-variable functions, v, 9-10, 30, 48.

Footprints, iv, 4-5, 17, 18, 26, 29, 30, 48-49.

Formula complexity, see Length of a 
Boolean function.

Four-variable functions, 2-9, 16-18,
26, 30, 33.

Free trees, 54.

Full adders, 11-12, 30, 41, 46, 51.

Functional decomposition, 21-25, 34-35.

Games, see Tic-tac-toe.

Gardner, Martin, 50.

Gates, 1, 8.

Generalized consensus, 24.

Good pairs, 22, 34-35.

GOST cipher, 33.

Gray, Frank, binary code, 43.

Greedy algorithm, 17.

Greedy-footprint heuristic, 17-18, 26, 48-49.

Gurvich, Vladimir Alexander (Гурвич, 
Владимир Александрович), 24.

Guy, Richard Kenneth, 49.

Hachitel, Gary Deane, 26.

Half adders, 11-12.

Hammer, Péter László (= Peter Lenie = 
Ivánescu, Petru Ladislav), 24.

Hardware versus software, 1, 45.

Harvard University Computation Laboratory, 30, 41.

Hellerman, Leo, 8-9.

Hexadecimal notation for truth tables, 9, 36, 48.

Highs, Stuart Lee, 24.

Horn, Alfred, clauses, 23, 52.

Ibaraki, Toshihide (杉木隆秀), 24.

If-then-else function (mux), 0, 38, 42, 44, 46.

Incompatible columns, 24.

Internet, ii, iii.

ITE, see If-then-else function.

\(k\)-ary trees, 28.

\(k\)-cubes, 33.

\(k\)-in-a-row function, 31.


Kleister, William, 19.

Khramchenko, Valerii Mikhailovich 
(Храмченко, Валерий Михайлович), 45.

Klein, Peter, 31.

Knuth, Donald Ervin (高德納), i, iv, 9, 19, 45.

Kogan, Alexander (Коган, Александр 
Юрьевич), 24.

Krichenkov, Rafail Evseevich (Кричековский, 
Рафаил Евсеевич), 47.

Krom, Melven Robert, clauses, 52.

Kung, Hsiang Tsung (孔祥庭), 45.

\(L(f)\), 3, see Length of a Boolean function.

Ladner, Richard Emil, 31, 32.

Lattice theory, 36.

Legalities of functions, 36, 37.

Length of a Boolean function, 3, 7, 29, 49.

Statistics, 5, 9, 40-41.

Li, Franklin Jack, iv, 9.

Lo Shu square magic, 49.

Local optimizations, 25.

Logic, 27-28, 36.

Lower bounds on combinational complexity, 7-8, 13-16, 26-28, 35-36.

Lupanov, Oleg Borisovich (Лупанов, Олег 
Борисович), 14, 16, 33, 42, 47, 49.

Magic Fifteen, 33.

Magic square, 49.

Majority operation, v, 8.

Matching, 23.

Mates in a Boolean chain, 35.

Matrix multiplication, 57.

McKellar, Archie Charles, 22, 24, 35.

Median chains, 37.

Median of seven, 37.

Median operation, v, 6, 29, 37.
Meyer, Albert Ronald da Silva, 27, 28, 55.
Minimum-memory evaluation, 5-7, 10, 29, 30, 43.
Minterms, 13, 15, 30.
MON, ii.
Monadic logic: Logic with only unary operators, 27-28, 36.
Monotone Boolean chains, 29, 36-37.
Monotone Boolean functions, self-dual, 37.
Monotone complexity, 10, 29, 36-37.
Müller, David Eugene, 44.
Multilevel logic synthesis, 26.
Multilinear representation of a Boolean function, 34, 40.
Multiple outputs, 11-13, 16-21, 25-26, 30-34.
Multiplexer, 2-way, see Mux.
2-way, 13, 31, 35.
Museum of Science and Industry, 19.
Mux (multiplexer) operation, 0, 38, 42, 44, 46.

NAND (7), 40.
with vacuum tubes, 8.
Nechiporuk, Eduard Ivanovich (Нечипорук, 
Эдуард Иванович), 49.
Normal Boolean functions, 4, 6, 14, 17, 29, 41, 53.
Normalization, 4, 38.
NOT-BOT gates (C), 1, 4.
Notational conventions, v, 36.
(xvz), v, see Median operation.
Noughts and crosses, see Tic-tac-toe.

Ogasawara, Yuri Petrovich (Огасавара, 
Юри Петрович), 46.
Optimization of Boolean chains, 25-26, 52.
Optimum Boolean evaluation, 7-9.
Or gates (v), 1.
with vacuum tubes, 8.
Oriented binary trees, 47.
Overlapping subtrees, 1.

Parallel addition, 12, 31-32, 46.
Parallel computation, 12, 28.
Parity function, 2, 9, 35, 36, 41.
Partial functions, 17-18, 35.
Paterson, Michael Stewart, iv, 30, 31.
Paul, Wolfgang Jakob, 35.
Permanence of a matrix, 23, 29.
Pi (x), as "random" example, 2, 22, 32, 38.
Pi function, 2, 22, 29, 34, 38.
Pigeonhole principle, 56.
Pippenger, Nicholas John, 49.
Pratt, Vaughan Ronald, 29.
Prefix problem, 31, 36.
Prefixes of strings, 36.

Preparata, Franco Paolo, 44.
Prime clauses, 33.
Prime implicants, 33, 54, 57.
Prime-number function, 14, 33.
Priority encoders, 31.
Projective plane, finite, 57.
Quantified formulas, 27-28, 55.
r-closed graphs, 37.
r-families of edges, 37.
Randomization, 22, 34.
Razborov, Alexander Alexandrovich (Разборов, Александр 
Александрович), 29.
Recurrence relations, 42, 44.
binary, 12, 15, 30, 43.
Recursive procedures, 38, 42.
Red’kin, Nikolai Petrovich (Редкин, 
Николай Петрович), 11, 26, 35, 53.
Registers, 5-7, 30.
Reordan, John, 47.
Ruskey, Frank, 33.

S-boxes, 33.
$S_{k_1,\ldots, k_n}(x)$ notation v, see Symmetric 
functions.
Sangianni-Vincenelli, Alberto Luigi, 26.
Savage, John E. Hornd, 47.
Schneier, Claus-Peter, 35.
Schroeppel, Richard Crabtree, iv, 30.
Second-order logic, 28.
Selection function, see Mux.
Self-dual monotone functions, 37.
Self-reference, 28, 60.
Seneca, Lucius Annaeus, iii.
Series-parallel switching networks, 47.
Seven-segment display, 16-18, 33.
Shannon, Claude Elwood, Jr., 14, 46, 47.
Shen, Vincent Yun-Shen (沈遵申), 
22, 24, 35.
Shoikov, Lev Abramovich (Шоиков, 
Лев Абрамович), 33, 51.
Sideways addition function (uv), v, 
12, 30, 42, 44.
Sklansky, Jack, 44.
Smith, John Lynn, 45.
Snr, Marc (Марк Снр), 31, 36.
Sorting networks, 31, 43.
Spira, Philip Martin, 29.
Stanford GraphBase, ii, iii, 45.
Stanford University InfoLab, v.
Stein, Sherman Kopald, 49.
Stockmeyer, Larry Joseph, 27, 28, 36, 42, 55.
Storage access function, 13.
Straight-line computation, see Boolean 
chains.
Strassen, Volker, 57.
Strong components, 52.
Subcubes, 33, 47.
INDEX AND GLOSSARY 61

Suffixes of strings, 36.
Sun Microsystems Laboratories, v.
Symmetric functions, v, 2–3, 8–10, 12, 13, 20, 30, 35, 36, 41, 42.

Tarjan, Robert Endre, 52.
Ternary Boolean functions, 8–9, 57.
Thoreau, David Henry (= Henry David), 0.
Threshold functions, 43.
Tic-tac-toe, 18–21, 33–34, 49.
Top-down synthesis, 7–9, 30, 40.
Topological sorting, 1.
Triangle function, 37.
Trivial functions, 41.

Truth tables, 2, 4, 6, 9, 46.
in hexadecimal notation, 9, 36, 48.
of partial functions, 18–20.
two-dimensional, 14–15, 21–25, 49, 51.

Uhlig, Dietmar, 35.
Upper bounds on combinational complexity, 14–16, 33, see also Footprints.

Vacuum-tube circuits, 8, 30.
Vector-valued Boolean functions, 11, 25.

Wakerly, John Francis, 51.
Weak second-order logic, 27–28, 36.
Wegener, Ingo Werner, 28, 42.
Weinberger, Arnold, 45.
Weiser, Peter Gallegos, 22, 24, 35.
Winder, Robert Owen, 57.
Wood, Frank W., 16.
WS1S, 28.

XOR gates (⊕), 1.

Zuse, Konrad, 46.