NEWLY AVAILABLE SECTION OF THE CLASSIC WORK

The Art of Computer Programming

VOLUME 4

Satisfiability

FASCICLE 6

DONALD E. KNUTH

THE ART OF COMPUTER PROGRAMMING

VOLUME 4, FASCICLE 6

Satisfiability

DONALD E. KNUTH *Stanford University*

ADDISON–WESLEY

Boston · Columbus · Indianapolis · New York · San Francisco
Amsterdam · Cape Town · Dubai · London · Madrid · Milan
Munich · Paris · Montréal · Toronto · Mexico City · Saõ Paulo
Delhi · Sydney · Hong Kong · Seoul · Singapore · Taipei · Tokyo

Library of Congress Cataloging-in-Publication Data

```
Knuth, Donald Ervin, 1938-
  The art of computer programming / Donald Ervin Knuth.
  viii,310 p.  24 cm.
  Includes bibliographical references and index.
  Contents: v. 4, fascicle 6. Satisfiability.
  ISBN 978-0-134-39760-3 (pbk. : alk. papers : volume 4, fascicle 6)
1. Computer programming.  2. Computer algorithms.  I. Title.
  QA76.6.K64 2005
  005.1-dc22
                                                      2005041030
```

Internet page http://www-cs-faculty.stanford.edu/~knuth/taocp.html contains current information about this book and related books.

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.

And see http://www-cs-faculty.stanford.edu/~knuth/mmix.html for basic information about the MMIX computer.

Electronic version by Mathematical Sciences Publishers (MSP), http://msp.org

ISBN-13 978-0-13-439760-3
ISBN-10 0-13-439760-6

First digital release, February 2016

PREFACE

These unforeseen stoppages,
which I own I had no conception of when I first set out;
— but which, I am convinced now, will rather increase than diminish as I advance,
— have struck out a hint which I am resolved to follow;
— and that is, — not to be in a hurry;
— but to go on leisurely, writing and publishing two volumes of my life every year;
— which, if I am suffered to go on quietly, and can make a tolerable bargain
with my bookseller, I shall continue to do as long as I live.

— LAURENCE STERNE, *The Life and Opinions of Tristram Shandy, Gentleman* (1759)

THIS BOOKLET is Fascicle 6 of *The Art of Computer Programming*, Volume 4: *Combinatorial Algorithms*. As explained in the preface to Fascicle 1 of Volume 1, I'm circulating the material in this preliminary form because I know that the task of completing Volume 4 will take many years; I can't wait for people to begin reading what I've written so far and to provide valuable feedback.

To put the material in context, this lengthy fascicle contains Section 7.2.2.2 of a long, long chapter on combinatorial algorithms. Chapter 7 will eventually fill at least four volumes (namely Volumes 4A, 4B, 4C, and 4D), assuming that I'm able to remain healthy. It began in Volume 4A with a short review of graph theory and a longer discussion of "Zeros and Ones" (Section 7.1); that volume concluded with Section 7.2.1, "Generating Basic Combinatorial Patterns," which was the first part of Section 7.2, "Generating All Possibilities." Volume 4B will resume the story with Section 7.2.2, about backtracking in general; then Section 7.2.2.1 will discuss a family of methods called "dancing links," for updating data structures while backtracking. That sets the scene for the present section, which applies those ideas to the important problem of Boolean satisfiability, aka 'SAT'.

Wow — Section 7.2.2.2 has turned out to be the longest section, by far, in *The Art of Computer Programming*. The SAT problem is evidently a killer app, because it is key to the solution of so many other problems. Consequently I can only hope that my lengthy treatment does not also kill off my faithful readers! As I wrote this material, one topic always seemed to flow naturally into another, so there was no neat way to break this section up into separate subsections. (And anyway the format of *TAOCP* doesn't allow for a Section 7.2.2.2.1.)

I've tried to ameliorate the reader's navigation problem by adding subheadings at the top of each right-hand page. Furthermore, as in other sections, the exercises appear in an order that roughly parallels the order in which corresponding topics are taken up in the text. Numerous cross-references are provided between text, exercises, and illustrations, so that you have a fairly good chance of keeping in sync. I've also tried to make the index as comprehensive as possible.

Look, for example, at a "random" page — say page 80, which is part of the subsection about Monte Carlo algorithms. On that page you'll see that exercises 302, 303, 299, and 306 are mentioned. So you can guess that the main exercises about Monte Carlo algorithms are numbered in the early 300s. (Indeed, exercise 306 deals with the important special case of "Las Vegas algorithms"; and the next exercises explore a fascinating concept called "reluctant doubling.") This entire section is full of surprises and tie-ins to other aspects of computer science.

Satisfiability is important chiefly because Boolean algebra is so versatile. Almost any problem can be formulated in terms of basic logical operations, and the formulation is particularly simple in a great many cases. Section 7.2.2.2 begins with ten typical examples of widely different applications, and closes with detailed empirical results for a hundred different benchmarks. The great variety of these problems — all of which are special cases of SAT — is illustrated on pages 116 and 117 (which are my favorite pages in this book).

The story of satisfiability is the tale of a triumph of software engineering, blended with rich doses of beautiful mathematics. Thanks to elegant new data structures and other techniques, modern SAT solvers are able to deal routinely with practical problems that involve many thousands of variables, although such problems were regarded as hopeless just a few years ago.

Section 7.2.2.2 explains how such a miracle occurred, by presenting complete details of seven SAT solvers, ranging from the small-footprint methods of Algorithms A and B to the state-of-the-art methods in Algorithms W, L, and C. (Well I have to hedge a little: New techniques are continually being discovered, hence SAT technology is ever-growing and the story is ongoing. But I do think that Algorithms W, L, and C compare reasonably well with the best algorithms of their class that were known in 2010. They're no longer at the cutting edge, but they still are amazingly good.)

Although this fascicle contains more than 300 pages, I constantly had to "cut, cut, cut," because a great deal more is known. While writing the material I found that new and potentially interesting-yet-unexplored topics kept popping up, more than enough to fill a lifetime. Yet I knew that I must move on. So I hope that I've selected for treatment here a significant fraction of the concepts that will prove to be the most important as time passes.

I wrote more than three hundred computer programs while preparing this material, because I find that I don't understand things unless I try to program them. Most of those programs were quite short, of course; but several of them are rather substantial, and possibly of interest to others. Therefore I've made a selection available by listing some of them on the following webpage:

`http://www-cs-faculty.stanford.edu/~knuth/programs.html`

You can also download `SATexamples.tgz` from that page; it's a collection of programs that generate data for all 100 of the benchmark examples discussed in the text, and many more.

Special thanks are due to Armin Biere, Randy Bryant, Sam Buss, Niklas Eén, Ian Gent, Marijn Heule, Holger Hoos, Svante Janson, Peter Jeavons, Daniel Kroening, Oliver Kullmann, Massimo Lauria, Wes Pegden, Will Shortz, Carsten Sinz, Niklas Sörensson, Udo Wermuth, and Ryan Williams for their detailed comments on my early attempts at exposition, as well as to dozens and dozens of other correspondents who have contributed crucial corrections. Thanks also to Stanford's Information Systems Laboratory for providing extra computer power when my laptop machine was inadequate.

I happily offer 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 do my best to give you immortal glory, by publishing your name in the eventual book:–)

Volume 4B will begin with a special tutorial and review of probability theory, in an unnumbered section entitled "Mathematical Preliminaries Redux." References to its equations and exercises use the abbreviation 'MPR'. (Think of the word "improvement.") A preliminary version of that section can be found online, via the following compressed PostScript file:

`http://www-cs-faculty.stanford.edu/~knuth/fasc5a.ps.gz`

The illustrations in this fascicle currently begin with 'Fig. 33' and run through 'Fig. 56'. Those numbers will change, eventually, but I won't know the final numbers until fascicle 5 has been completed.

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 D. E. K.
23 September 2015

A note on notation. Several formulas in this booklet use the notation $\langle xyz \rangle$ for the median function, which is discussed extensively in Section 7.1.1. Other formulas use the notation $x \dot{-} y$ for the monus function (aka dot-minus or saturating subtraction), which was defined in Section 1.3.1´. Hexadecimal constants are preceded by a number sign or hash mark: $^{\#}$`123` means $(123)_{16}$.

If you run across other notations that appear strange, please look under the heading 'Notational conventions' in the index to the present fascicle, and/or at the Index to Notations at the end of Volume 4A (it is Appendix B on pages 822–827). Volume 4B will, of course, have its own Appendix B some day.

A note on references. References to *IEEE Transactions* include a letter code for the type of transactions, in boldface preceding the volume number. For example, '*IEEE Trans.* **C-35**' means the *IEEE Transactions on Computers*, volume 35. The IEEE no longer uses these convenient letter codes, but the codes aren't too hard to decipher: '**EC**' once stood for "Electronic Computers," '**IT**' for "Information Theory," '**SE**' for "Software Engineering," and '**SP**' for "Signal Processing," etc.; '**CAD**' meant "Computer-Aided Design of Integrated Circuits and Systems."

Other common abbreviations used in references appear on page x of Volume 1, or in the index below.

An external exercise. Here's an exercise for Section 7.2.2.1 that I plan to put eventually into fascicle 5:

00. [*20*] The problem of Langford pairs on $\{1, 1, \ldots, n, n\}$ can be represented as an exact cover problem using columns $\{d_1, \ldots, d_n\} \cup \{s_1, \ldots, s_{2n}\}$; the rows are $d_i\ s_j\ s_k$ for $1 \le i \le n$ and $1 \le j < k \le 2n$ and $k = i+j+1$, meaning "put digit i into slots j and k."

However, that construction essentially gives us every solution twice, because the left-right reversal of any solution is also a solution. Modify it so that we get only half as many solutions; the others will be the reversals of these.

And here's its cryptic answer (needed in exercise 7.2.2.2–13):

00. Omit the rows with $i = n - [n \text{ even}]$ and $j > n/2$.

(Other solutions are possible. For example, we could omit the rows with $i = 1$ and $j \ge n$; that would omit $n - 1$ rows instead of only $\lfloor n/2 \rfloor$. However, the suggested rule turns out to make the dancing links algorithm run about 10% faster.)

Now I saw, tho' too late, the Folly of
beginning a Work before we count the Cost,
and before we judge rightly of our own Strength to go through with it.
— DANIEL DEFOE, *Robinson Crusoe* (1719)

CONTENTS

Chapter 7 — Combinatorial Searching 0

7.2. Generating All Possibilities 0
 7.2.1. Generating Basic Combinatorial Patterns 0
 7.2.2. Basic Backtrack 0
  7.2.2.1. Dancing links 0
  7.2.2.2. Satisfiability 1
   Example applications 4
   Backtracking algorithms 27
   Random clauses 47
   Resolution of clauses 54
   Clause-learning algorithms 60
   Monte Carlo algorithms 77
   The Local Lemma 81
   *Message-passing algorithms 90
   *Preprocessing of clauses 95
   Encoding constraints into clauses 97
   Unit propagation and forcing 103
   Symmetry breaking 105
   Satisfiability-preserving maps 107
   One hundred test cases 113
   Tuning the parameters 124
   Exploiting parallelism 128
   History 129
   Exercises 133

Answers to Exercises 185

Index to Algorithms and Theorems 292

Index and Glossary 293

That your book has been delayed I am glad,
since you have gained an opportunity of being more exact.
— SAMUEL JOHNSON, letter to Charles Burney (1 November 1784)

He reaps no satisfaction but from low and sensual objects,
or from the indulgence of malignant passions.
— DAVID HUME, *The Sceptic* (1742)

I can't get no ...
— MICK JAGGER and KEITH RICHARDS, *Satisfaction* (1965)

7.2.2.2. Satisfiability. We turn now to one of the most fundamental problems of computer science: Given a Boolean formula $F(x_1, \ldots, x_n)$, expressed in so-called "conjunctive normal form" as an AND of ORs, can we "satisfy" F by assigning values to its variables in such a way that $F(x_1, \ldots, x_n) = 1$? For example, the formula

$$F(x_1, x_2, x_3) = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \qquad (1)$$

is satisfied when $x_1x_2x_3 = 001$. But if we rule that solution out, by defining

$$G(x_1, x_2, x_3) = F(x_1, x_2, x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3), \qquad (2)$$

then G is unsatisfiable: It has no satisfying assignment.

Section 7.1.1 discussed the embarrassing fact that nobody has ever been able to come up with an efficient algorithm to solve the general satisfiability problem, in the sense that the satisfiability of any given formula of size N could be decided in $N^{O(1)}$ steps. Indeed, the famous unsolved question "does P = NP?" is equivalent to asking whether such an algorithm exists. We will see in Section 7.9 that satisfiability is a natural progenitor of every NP-complete problem.*

On the other hand enormous technical breakthroughs in recent years have led to amazingly good ways to approach the satisfiability problem. We now have algorithms that are much more efficient than anyone had dared to believe possible before the year 2000. These so-called "SAT solvers" are able to handle industrial-strength problems, involving millions of variables, with relative ease, and they've had a profound impact on many areas of research such as computer-aided verification. In this section we shall study the principles that underlie modern SAT-solving procedures.

* At the present time very few people believe that P = NP [see *SIGACT News* **43**, 2 (June 2012), 53–77]. In other words, almost everybody who has studied the subject thinks that satisfiability cannot be decided in polynomial time. The author of this book, however, suspects that $N^{O(1)}$-step algorithms do exist, yet that they're unknowable. Almost all polynomial time algorithms are so complicated that they lie beyond human comprehension, and could never be programmed for an actual computer in the real world. Existence is different from embodiment.

To begin, let's define the problem carefully and simplify the notation, so that our discussion will be as efficient as the algorithms that we'll be considering. Throughout this section we shall deal with *variables*, which are elements of any convenient set. Variables are often denoted by x_1, x_2, x_3, …, as in (1); but any other symbols can also be used, like a, b, c, or even d'''_{74}. We will in fact often use the numerals 1, 2, 3, … to stand for variables; and in many cases we'll find it convenient to write just j instead of x_j, because it takes less time and less space if we don't have to write so many x's. Thus '2' and 'x_2' will mean the same thing in many of the discussions below.

A *literal* is either a variable or the complement of a variable. In other words, if v is a variable, both v and $\bar{v}$ are literals. If there are n possible variables in some problem, there are $2n$ possible literals. If l is the literal $\bar{x}_2$, which is also written $\bar{2}$, then the complement of l, $\bar{l}$, is x_2, which is also written 2.

The variable that corresponds to a literal l is denoted by $|l|$; thus we have $|v| = |\bar{v}| = v$ for every variable v. Sometimes we write $\pm v$ for a literal that is either v or $\bar{v}$. We might also denote such a literal by σv, where σ is ± 1. The literal l is called *positive* if $|l| = l$; otherwise $|l| = \bar{l}$, and l is said to be *negative*.

Two literals l and l' are *distinct* if $l \neq l'$. They are *strictly distinct* if $|l| \neq |l'|$. A set of literals $\{l_1, \ldots, l_k\}$ is strictly distinct if $|l_i| \neq |l_j|$ for $1 \leq i < j \leq k$.

The satisfiability problem, like all good problems, can be understood in many equivalent ways, and we will find it convenient to switch from one viewpoint to another as we deal with different aspects of the problem. Example (1) is an AND of clauses, where every clause is an OR of literals; but we might as well regard every clause as simply a *set* of literals, and a formula as a set of clauses. With that simplification, and with 'x_j' identical to 'j', Eq. (1) becomes

$$F \;=\; \bigl\{\{1,\bar{2}\}, \{2,3\}, \{\bar{1},\bar{3}\}, \{\bar{1},\bar{2},3\}\bigr\}.$$

And we needn't bother to represent the clauses with braces and commas either; we can simply write out the literals of each clause. With that shorthand we're able to perceive the real essence of (1) and (2):

$$F = \{1\bar{2}, 23, \bar{1}\bar{3}, \bar{1}\bar{2}3\}, \qquad G = F \cup \{12\bar{3}\}. \tag{3}$$

Here F is a set of four clauses, and G is a set of five.

In this guise, the satisfiability problem is equivalent to a *covering problem*, analogous to the exact cover problems that we considered in Section 7.2.2.1: Let

$$T_n = \bigl\{\{x_1,\bar{x}_1\}, \{x_2,\bar{x}_2\}, \ldots, \{x_n,\bar{x}_n\}\bigr\} = \{1\bar{1}, 2\bar{2}, \ldots, n\bar{n}\}. \tag{4}$$

"Given a set $F = \{C_1, \ldots, C_m\}$, where each C_i is a clause and each clause consists of literals based on the variables $\{x_1, \ldots, x_n\}$, find a set L of n literals that 'covers' $F \cup T_n$, in the sense that every clause contains at least one element of L." For example, the set F in (3) is covered by $L = \{\bar{1}, \bar{2}, 3\}$, and so is the set T_3; hence F is satisfiable. The set G is covered by $\{1, \bar{1}, 2\}$ or $\{1, \bar{1}, 3\}$ or $\cdots$ or $\{\bar{2}, 3, \bar{3}\}$, but not by any three literals that also cover T_3; so G is unsatisfiable.

Similarly, a family F of clauses is satisfiable if and only if it can be covered by a set L of *strictly distinct* literals.

If F' is any formula obtained from F by complementing one or more variables, it's clear that F' is satisfiable if and only if F is satisfiable. For example, if we replace 1 by $\bar{1}$ and 2 by $\bar{2}$ in (3) we obtain

$$F' = \{\bar{1}2, \bar{2}3, 1\bar{3}, 123\}, \qquad G' = F' \cup \{\bar{1}2\bar{3}\}.$$

In this case F' is *trivially* satisfiable, because each of its clauses contains a positive literal: Every such formula is satisfied by simply letting L be the set of positive literals. Thus the satisfiability problem is the same as the problem of switching signs (or "polarities") so that no all-negative clauses remain.

Another problem equivalent to satisfiability is obtained by going back to the Boolean interpretation in (1) and complementing both sides of the equation. By De Morgan's laws 7.1.1–(11) and (12) we have

$$\overline{F}(x_1, x_2, x_3) = (\bar{x}_1 \wedge x_2) \vee (\bar{x}_2 \wedge \bar{x}_3) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \bar{x}_3); \qquad (5)$$

and F is unsatisfiable $\iff F = 0 \iff \overline{F} = 1 \iff \overline{F}$ is a tautology. Consequently F is satisfiable if and only if $\overline{F}$ is not a tautology: The tautology problem and the satisfiability problem are essentially the same.*

Since the satisfiability problem is so important, we simply call it SAT. And instances of the problem such as (1), in which there are no clauses of length greater than 3, are called 3SAT. In general, kSAT is the satisfiability problem restricted to instances where no clause has more than k literals.

Clauses of length 1 are called *unit clauses*, or unary clauses. Binary clauses, similarly, have length 2; then come ternary clauses, quaternary clauses, and so forth. Going the other way, the *empty clause*, or nullary clause, has length 0 and is denoted by ϵ; it is always unsatisfiable. Short clauses are very important in algorithms for SAT, because they are easier to deal with than long clauses. But long clauses aren't necessarily bad; they're much easier to satisfy than the short ones.

A slight technicality arises when we consider clause length: The binary clause $(x_1 \vee \bar{x}_2)$ in (1) is equivalent to the ternary clause $(x_1 \vee x_1 \vee \bar{x}_2)$ as well as to $(x_1 \vee \bar{x}_2 \vee \bar{x}_2)$ and to longer clauses such as $(x_1 \vee x_1 \vee x_1 \vee \bar{x}_2)$; so we can regard it as a clause of *any* length ≥ 2. But when we think of clauses as *sets* of literals rather than ORs of literals, we usually rule out multisets such as $11\bar{2}$ or $1\bar{2}\bar{2}$ that aren't sets; in that sense a binary clause is *not* a special case of a ternary clause. On the other hand, every binary clause $(x \vee y)$ is equivalent to *two* ternary clauses, $(x \vee y \vee z) \wedge (x \vee y \vee \bar{z})$, if z is another variable; and every k-ary clause is equivalent to two $(k+1)$-ary clauses. Therefore we can assume, if we like, that kSAT deals only with clauses whose length is exactly k.

A clause is tautological (always satisfied) if it contains both v and $\bar{v}$ for some variable v. Tautological clauses can be denoted by $\wp$ (see exercise 7.1.4–222). They never affect a satisfiability problem; so we usually assume that the clauses input to a SAT-solving algorithm consist of strictly distinct literals.

When we discussed the 3SAT problem briefly in Section 7.1.1, we took a look at formula 7.1.1–(32), "the shortest interesting formula in 3CNF." In our

* Strictly speaking, TAUT is coNP-complete, while SAT is NP-complete; see Section 7.9.

new shorthand, it consists of the following eight unsatisfiable clauses:

$$R = \{12\bar{3}, 23\bar{4}, 341, 4\bar{1}2, \bar{1}\bar{2}3, \bar{2}\bar{3}4, \bar{3}\bar{4}\bar{1}, \bar{4}1\bar{2}\}. \qquad (6)$$

This set makes an excellent little test case, so we will refer to it frequently below. (The letter R reminds us that it is based on R. L. Rivest's associative block design 6.5–(13).) The first seven clauses of R, namely

$$R' = \{12\bar{3}, 23\bar{4}, 341, 4\bar{1}2, \bar{1}\bar{2}3, \bar{2}\bar{3}4, \bar{3}\bar{4}\bar{1}\}, \qquad (7)$$

also make nice test data; they are satisfied only by choosing the complements of the literals in the omitted clause, namely $\{4, \bar{1}, 2\}$. More precisely, the literals 4, $\bar{1}$, and 2 are necessary and sufficient to cover R'; we can also include either 3 or $\bar{3}$ in the solution. Notice that (6) is symmetric under the cyclic permutation $1 \to 2 \to 3 \to 4 \to \bar{1} \to \bar{2} \to \bar{3} \to \bar{4} \to 1$ of literals; thus, omitting *any* clause of (6) gives a satisfiability problem equivalent to (7).

A simple example. SAT solvers are important because an enormous variety of problems can readily be formulated Booleanwise as ANDs of ORs. Let's begin with a little puzzle that leads to an instructive family of example problems: *Find a binary sequence $x_1 \ldots x_8$ that has no three equally spaced* 0*s and no three equally spaced* 1*s.* For example, the sequence 01001011 almost works; but it doesn't qualify, because x_2, x_5, and x_8 are equally spaced 1s.

If we try to solve this puzzle by backtracking manually through all 8-bit sequences in lexicographic order, we see that $x_1x_2 = 00$ forces $x_3 = 1$. Then $x_1x_2x_3x_4x_5x_6x_7 = 0010011$ leaves us with no choice for x_8. A minute or two of further hand calculation reveals that the puzzle has just six solutions, namely

$$00110011,\ 01011010,\ 01100110,\ 10011001,\ 10100101,\ 11001100. \qquad (8)$$

Furthermore it's easy to see that none of these solutions can be extended to a suitable binary sequence of length 9. We conclude that *every binary sequence $x_1 \ldots x_9$ contains three equally spaced* 0*s or three equally spaced* 1*s.*

Notice now that the condition $x_2x_5x_8 \neq 111$ is the same as the Boolean clause $(\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_8)$, namely $\bar{2}\bar{5}\bar{8}$. Similarly $x_2x_5x_8 \neq 000$ is the same as 258. So we have just verified that the following 32 clauses are unsatisfiable:

$$\begin{gathered} 123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159, \\ \bar{1}\bar{2}\bar{3}, \bar{2}\bar{3}\bar{4}, \ldots, \bar{7}\bar{8}\bar{9}, \bar{1}\bar{3}\bar{5}, \bar{2}\bar{4}\bar{6}, \ldots, \bar{5}\bar{7}\bar{9}, \bar{1}\bar{4}\bar{7}, \bar{2}\bar{5}\bar{8}, \bar{3}\bar{6}\bar{9}, \bar{1}\bar{5}\bar{9}. \end{gathered} \qquad (9)$$

This result is a special case of a general fact that holds for any given positive integers j and k: *If n is sufficiently large, every binary sequence $x_1 \ldots x_n$ contains either j equally spaced* 0*s or k equally spaced* 1*s.* The smallest such n is denoted by $W(j,k)$ in honor of B. L. van der Waerden, who proved an even more general result (see exercise 2.3.4.3–6): *If n is sufficiently large, and if $k_0, \ldots, k_{b-1}$ are positive integers, every b-ary sequence $x_1 \ldots x_n$ contains k_a equally spaced a's for some digit a, $0 \leq a < b$.* The least such n is $W(k_0, \ldots, k_{b-1})$.

Let us accordingly define the following set of clauses when $j, k, n > 0$:

$$\begin{aligned} \mathit{waerden}(j,k;n) = {}& \bigl\{(x_i \vee x_{i+d} \vee \cdots \vee x_{i+(j-1)d}) \bigm| 1 \leq i \leq n-(j-1)d,\ d \geq 1\bigr\} \\ & \cup \bigl\{(\bar{x}_i \vee \bar{x}_{i+d} \vee \cdots \vee \bar{x}_{i+(k-1)d}) \bigm| 1 \leq i \leq n-(k-1)d,\ d \geq 1\bigr\}. \end{aligned} \qquad (10)$$

The 32 clauses in (9) are *waerden*$(3, 3; 9)$; and in general *waerden*$(j, k; n)$ is an appealing instance of SAT, satisfiable if and only if $n < W(j, k)$.

It's obvious that $W(1, k) = k$ and $W(2, k) = 2k - [k \text{ even}]$; but when j and k exceed 2 the numbers $W(j, k)$ are quite mysterious. We've seen that $W(3, 3) = 9$, and the following nontrivial values are currently known:

$k =$	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$W(3, k) =$	9	18	22	32	46	58	77	97	114	135	160	186	218	238	279	312	349
$W(4, k) =$	18	35	55	73	109	146	309	?	?	?	?	?	?	?	?	?	?
$W(5, k) =$	22	55	178	206	260	?	?	?	?	?	?	?	?	?	?	?	?
$W(6, k) =$	32	73	206	1132	?	?	?	?	?	?	?	?	?	?	?	?	?

V. Chvátal inaugurated the study of $W(j, k)$ by computing the values for $j+k \le 9$ as well as $W(3, 7)$ [*Combinatorial Structures and Their Applications* (1970), 31–33]. Most of the large values in this table have been calculated by state-of-the-art SAT solvers [see M. Kouril and J. L. Paul, *Experimental Math.* **17** (2008), 53–61; M. Kouril, *Integers* **12** (2012), A46:1–A46:13]. The table entries for $j = 3$ suggest that we might have $W(3, k) < k^2$ when $k > 4$, but that isn't true: SAT solvers have also been used to establish the lower bounds

$k =$	20	21	22	23	24	25	26	27	28	29	30
$W(3, k) \ge$	389	416	464	516	593	656	727	770	827	868	903

(which might in fact be the true values for this range of k); see T. Ahmed, O. Kullmann, and H. Snevily [*Discrete Applied Math.* **174** (2014), 27–51].

Notice that the literals in every clause of *waerden*$(j, k; n)$ have the same sign: They're either all positive or all negative. Does this "monotonic" property make the SAT problem any easier? Unfortunately, no: Exercise 10 proves that *any* set of clauses can be converted to an equivalent set of monotonic clauses.

Exact covering. The exact cover problems that we solved with "dancing links" in Section 7.2.2.1 can easily be reformulated as instances of SAT and handed off to SAT solvers. For example, let's look again at Langford pairs, the task of placing two 1s, two 2s, ..., two n's into $2n$ slots so that exactly k slots intervene between the two appearances of k, for each k. The corresponding exact cover problem when $n = 3$ has nine columns and eight rows (see 7.2.2.1–(00)):

$$d_1\, s_1\, s_3,\ d_1\, s_2\, s_4,\ d_1\, s_3\, s_5,\ d_1\, s_4\, s_6,\ d_2\, s_1\, s_4,\ d_2\, s_2\, s_5,\ d_2\, s_3\, s_6,\ d_3\, s_1\, s_5. \qquad (11)$$

The columns are d_i for $1 \le i \le 3$ and s_j for $1 \le j \le 6$; the row '$d_i\, s_j\, s_k$' means that digit i is placed in slots j and k. Left-right symmetry allows us to omit the row '$d_3\, s_2\, s_6$' from this specification.

We want to select rows of (11) so that each column appears just once. Let the Boolean variable x_j mean 'select row j', for $1 \le j \le 8$; the problem is then to satisfy the nine constraints

$$\begin{aligned} &S_1(x_1, x_2, x_3, x_4) \wedge S_1(x_5, x_6, x_7) \wedge S_1(x_8) \\ &\qquad \wedge S_1(x_1, x_5, x_8) \wedge S_1(x_2, x_6) \wedge S_1(x_1, x_3, x_7) \\ &\qquad\qquad \wedge S_1(x_2, x_4, x_5) \wedge S_1(x_3, x_6, x_8) \wedge S_1(x_4, x_7), \end{aligned} \qquad (12)$$

one for each column. (Here, as usual, $S_1(y_1,\ldots,y_p)$ denotes the symmetric function $[y_1+\cdots+y_p=1]$.) For example, we must have $x_5+x_6+x_7=1$, because column d_2 appears in rows 5, 6, and 7 of (11).

One of the simplest ways to express the symmetric Boolean function S_1 as an AND of ORs is to use $1+\binom{p}{2}$ clauses:

$$S_1(y_1,\ldots,y_p)=(y_1\vee\cdots\vee y_p)\wedge\bigwedge_{1\le j<k\le p}(\bar y_j\vee\bar y_k). \tag{13}$$

"At least one of the y's is true, but not two." Then (12) becomes, in shorthand,

$$\{1234,\bar1\bar2,\bar1\bar3,\bar1\bar4,\bar2\bar3,\bar2\bar4,\bar3\bar4,567,\bar5\bar6,\bar5\bar7,\bar6\bar7,8,$$
$$158,\bar1\bar5,\bar1\bar8,\bar5\bar8,26,\bar2\bar6,137,\bar1\bar3,\bar1\bar7,\bar3\bar7,$$
$$245,\bar2\bar4,\bar2\bar5,\bar4\bar5,368,\bar3\bar6,\bar3\bar8,\bar6\bar8,47,\bar4\bar7\}; \tag{14}$$

we shall call these clauses *langford*(3). (Notice that only 30 of them are actually distinct, because $\bar1\bar3$ and $\bar2\bar4$ appear twice.) Exercise 13 defines *langford*(n); we know from exercise 7–1 that *langford*(n) is satisfiable $\iff n \bmod 4=0$ or 3.

The unary clause 8 in (14) tells us immediately that $x_8=1$. Then from the binary clauses $\bar1\bar8$, $\bar5\bar8$, $\bar3\bar8$, $\bar6\bar8$ we have $x_1=x_5=x_3=x_6=0$. The ternary clause 137 then implies $x_7=1$; finally $x_4=0$ (from $\bar4\bar7$) and $x_2=1$ (from 1234). Rows 8, 7, and 2 of (11) now give us the desired Langford pairing 3 1 2 1 3 2.

Incidentally, the function $S_1(y_1,y_2,y_3,y_4,y_5)$ can also be expressed as

$$(y_1\vee y_2\vee y_3\vee y_4\vee y_5)\wedge(\bar y_1\vee\bar y_2)\wedge(\bar y_1\vee\bar y_3)\wedge(\bar y_1\vee\bar t)$$
$$\wedge(\bar y_2\vee\bar y_3)\wedge(\bar y_2\vee\bar t)\wedge(\bar y_3\vee\bar t)\wedge(t\vee\bar y_4)\wedge(t\vee\bar y_5)\wedge(\bar y_4\vee\bar y_5),$$

where t is a new variable. In general, if p gets big, it's possible to express $S_1(y_1,\ldots,y_p)$ with only $3p-5$ clauses instead of $\binom{p}{2}+1$, by using $\lfloor(p-3)/2\rfloor$ new variables as explained in exercise 12. When this alternative encoding is used to represent Langford pairs of order n, we'll call the resulting clauses *langford*$'(n)$.

Do SAT solvers do a better job with the clauses *langford*(n) or *langford*$'(n)$? Stay tuned: We'll find out later.

Coloring a graph. The classical problem of coloring a graph with at most d colors is another rich source of benchmark examples for SAT solvers. If the graph has n vertices V, we can introduce nd variables v_j, for $v\in V$ and $1\le j\le d$, signifying that v has color j; the resulting clauses are quite simple:

$$(v_1\vee v_2\vee\cdots\vee v_d)\text{ for }v\in V\text{ (“every vertex has at least one color”);} \tag{15}$$
$$(\bar u_j\vee\bar v_j)\text{ for }u\,\text{---}\,v,\ 1\le j\le d\text{ (“adjacent vertices have different colors”).} \tag{16}$$

We could also add $n\binom{d}{2}$ additional so-called *exclusion clauses*

$$(\bar v_i\vee\bar v_j)\text{ for }v\in V,\ 1\le i<j\le d\text{ (“every vertex has at most one color”);} \tag{17}$$

but they're optional, because vertices with more than one color are harmless. Indeed, if we find a solution with $v_1=v_2=1$, we'll be extra happy, because it gives us *two* legal ways to color vertex v. (See exercise 14.)

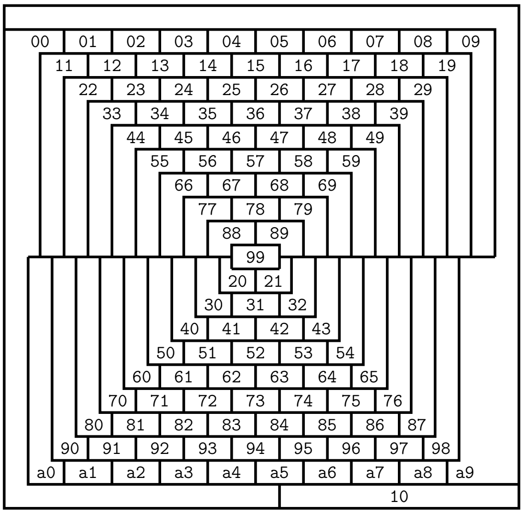


Fig. 33. The McGregor graph of order 10. Each region of this "map" is identified by a two-digit hexadecimal code. Can you color the regions with four colors, never using the same color for two adjacent regions?

Martin Gardner astonished the world in 1975 when he reported [*Scientific American* **232**, 4 (April 1975), 126–130] that a proper coloring of the planar map in Fig. 33 requires *five* distinct colors, thereby disproving the longstanding four-color conjecture. (In that same column he also cited several other "facts" supposedly discovered in 1974: (i) $e^{\pi\sqrt{163}}$ is an integer; (ii) pawn-to-king-rook-4 ('h4') is a winning first move in chess; (iii) the theory of special relativity is fatally flawed; (iv) Leonardo da Vinci invented the flush toilet; and (v) Robert Ripoff devised a motor that is powered entirely by psychic energy. Thousands of readers failed to notice that they had been April Fooled!)

The map in Fig. 33 actually *can* be 4-colored; you are hereby challenged to discover a suitable way to do this, before turning to the answer of exercise 18. Indeed, the four-color conjecture became the Four Color Theorem in 1976, as mentioned in Section 7. Fortunately that result was still unknown in April of 1975; otherwise this interesting graph would probably never have appeared in print. McGregor's graph has 110 vertices (regions) and 324 edges (adjacencies between regions); hence (15) and (16) yield $110 + 1296 = 1406$ clauses on 440 variables, which a modern SAT solver can polish off quickly.

We can also go much further and solve problems that would be extremely difficult by hand. For example, we can add constraints to limit the number of regions that receive a particular color. Randal Bryant exploited this idea in 2010 to discover that there's a four-coloring of Fig. 33 that uses one of the colors only 7 times (see exercise 17). His coloring is, in fact, unique, and it leads to an explicit way to 4-color the McGregor graphs of all orders $n \geq 3$ (exercise 18).

Such additional constraints can be generated in many ways. We could, for instance, append $\binom{110}{8}$ clauses, one for every choice of 8 regions, specifying that those 8 regions aren't all colored 1. But no, we'd better scratch that idea: $\binom{110}{8} = 409{,}705{,}619{,}895$. Even if we restricted ourselves to the 74,792,876,790 sets of 8 regions that are *independent*, we'd be dealing with far too many clauses.

An interesting SAT-oriented way to ensure that $x_1 + \cdots + x_n$ is at most r, which works well when n and r are rather large, was found by C. Sinz [*LNCS* **3709** (2005), 827–831]. His method introduces $(n-r)r$ new variables s_j^k for $1 \le j \le n-r$ and $1 \le k \le r$. If F is any satisfiability problem and if we add the $(n-r-1)r + (n-r)(r+1)$ clauses

$$(\bar{s}_j^k \vee s_{j+1}^k), \qquad \text{for } 1 \le j < n-r \text{ and } 1 \le k \le r, \qquad (18)$$
$$(\bar{x}_{j+k} \vee \bar{s}_j^k \vee s_j^{k+1}), \qquad \text{for } 1 \le j \le n-r \text{ and } 0 \le k \le r, \qquad (19)$$

where $\bar{s}_j^k$ is omitted when $k=0$ and s_j^{k+1} is omitted when $k=r$, then the new set of clauses is satisfiable if and only if F is satisfiable with $x_1+\cdots+x_n \le r$. (See exercise 26.) With this scheme we can limit the number of red-colored regions of McGregor's graph to at most 7 by appending 1538 clauses in 721 new variables.

Another way to achieve the same goal, which turns out to be even better, has been proposed by O. Bailleux and Y. Boufkhad [*LNCS* **2833** (2003), 108–122]. Their method is a bit more difficult to describe, but still easy to implement: Consider a complete binary tree that has $n-1$ internal nodes numbered 1 through $n-1$, and n leaves numbered n through $2n-1$; the children of node k, for $1 \le k < n$, are nodes $2k$ and $2k+1$ (see 2.3.4.5–(5)). We form new variables b_j^k for $1 < k < n$ and $1 \le j \le t_k$, where t_k is the minimum of r and the number of leaves below node k. Then the following clauses, explained in exercise 27, do the job:

$$(\bar{b}_i^{2k} \vee \bar{b}_j^{2k+1} \vee b_{i+j}^k), \quad \text{for } 0 \le i \le t_{2k},\ 0 \le j \le t_{2k+1},\ 1 \le i+j \le t_k+1,\ 1 < k < n; \quad (20)$$
$$(\bar{b}_i^2 \vee \bar{b}_j^3), \qquad \text{for } 0 \le i \le t_2,\ 0 \le j \le t_3,\ i+j = r+1. \qquad (21)$$

In these formulas we let $t_k = 1$ and $b_1^k = x_{k-n+1}$ for $n \le k < 2n$; all literals $\bar{b}_0^k$ and b_{r+1}^k are to be omitted. Applying (20) and (21) to McGregor's graph, with $n = 110$ and $r = 7$, yields just 1216 new clauses in 399 new variables.

The same ideas apply when we want to ensure that $x_1+\cdots+x_n$ is *at least* r, because of the identity $S_{\ge r}(x_1,\ldots,x_n) = S_{\le n-r}(\bar{x}_1,\ldots,\bar{x}_n)$. And exercise 30 considers the case of equality, when our goal is to make $x_1+\cdots+x_n = r$. We'll discuss other encodings of such cardinality constraints below.

Factoring integers. Next on our agenda is a family of SAT instances with quite a different flavor. *Given an $(m+n)$-bit binary integer $z = (z_{m+n}\ldots z_2 z_1)_2$, do there exist integers $x = (x_m \ldots x_1)_2$ and $y = (y_n \ldots y_1)_2$ such that $z = x \times y$?* For example, if $m = 2$ and $n = 3$, we want to invert the binary multiplication

$$\begin{array}{rrrrr} & & y_3 & y_2 & y_1 \\ & \times & & x_2 & x_1 \\ \hline & & a_3 & a_2 & a_1 \\ & b_3 & b_2 & b_1 & \\ \hline c_3 & c_2 & c_1 & & \\ \hline z_5 & z_4 & z_3 & z_2 & z_1 \end{array} \qquad \begin{aligned} (a_3a_2a_1)_2 &= (y_3y_2y_1)_2 \times x_1 \\ (b_3b_2b_1)_2 &= (y_3y_2y_1)_2 \times x_2 \end{aligned} \qquad \begin{aligned} z_1 &= a_1 \\ (c_1z_2)_2 &= a_2 + b_1 \\ (c_2z_3)_2 &= a_3 + b_2 + c_1 \\ (c_3z_4)_2 &= b_3 + c_2 \\ z_5 &= c_3 \end{aligned} \qquad (22)$$

when the z bits are given. This problem is satisfiable when $z = 21 = (10101)_2$, in the sense that suitable binary values x_1, x_2, y_1, y_2, y_3, a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3 do satisfy these equations. But it's unsatisfiable when $z = 19 = (10011)_2$.

Arithmetical calculations like (22) are easily expressed in terms of clauses that can be fed to a SAT solver: We first specify the computation by constructing a Boolean chain, then we encode each step of the chain in terms of a few clauses. One such chain, if we identify a_1 with z_1 and c_3 with z_5, is

$$\begin{array}{llllll} z_1\leftarrow x_1\wedge y_1, & b_1\leftarrow x_2\wedge y_1, & z_2\leftarrow a_2\oplus b_1, & s\leftarrow a_3\oplus b_2, & z_3\leftarrow s\oplus c_1, & z_4\leftarrow b_3\oplus c_2, \\ a_2\leftarrow x_1\wedge y_2, & b_2\leftarrow x_2\wedge y_2, & c_1\leftarrow a_2\wedge b_1, & p\leftarrow a_3\wedge b_2, & q\leftarrow s\wedge c_1, & z_5\leftarrow b_3\wedge c_2, \\ a_3\leftarrow x_1\wedge y_3, & b_3\leftarrow x_2\wedge y_3, & & & c_2\leftarrow p\vee q, & \end{array} \qquad (23)$$

using a "full adder" to compute c_2z_3 and "half adders" to compute c_1z_2 and c_3z_4 (see 7.1.2–(23) and (24)). And that chain is equivalent to the 49 clauses

$$(x_1\vee\bar z_1)\wedge(y_1\vee\bar z_1)\wedge(\bar x_1\vee\bar y_1\vee z_1)\wedge\cdots\wedge(\bar b_3\vee\bar c_2\vee\bar z_4)\wedge(b_3\vee\bar z_5)\wedge(c_2\vee\bar z_5)\wedge(\bar b_3\vee\bar c_2\vee z_5)$$

obtained by expanding the elementary computations according to simple rules:

$$\begin{aligned} &t\leftarrow u\wedge v \text{ becomes } (u\vee\bar t)\wedge(v\vee\bar t)\wedge(\bar u\vee\bar v\vee t);\\ &t\leftarrow u\vee v \text{ becomes } (\bar u\vee t)\wedge(\bar v\vee t)\wedge(u\vee v\vee\bar t); \qquad (24)\\ &t\leftarrow u\oplus v \text{ becomes } (\bar u\vee v\vee t)\wedge(u\vee\bar v\vee t)\wedge(u\vee v\vee\bar t)\wedge(\bar u\vee\bar v\vee\bar t). \end{aligned}$$

To complete the specification of this factoring problem when, say, $z=(10101)_2$, we simply append the unary clauses $(z_5)\wedge(\bar z_4)\wedge(z_3)\wedge(\bar z_2)\wedge(z_1)$.

Logicians have known for a long time that computational steps can readily be expressed as conjunctions of clauses. Rules such as (24) are now called *Tseytin encoding*, after Gregory Tseytin (1966). Our representation of a small five-bit factorization problem in 49+5 clauses may not seem very efficient; but we will see shortly that m-bit by n-bit factorization corresponds to a satisfiability problem with fewer than $6mn$ variables, and fewer than $20mn$ clauses of length 3 or less.

Even if the system has hundreds or thousands of formulas,
it can be put into conjunctive normal form "piece by piece,"
without any "multiplying out."
— MARTIN DAVIS and HILARY PUTNAM (1958)

Suppose $m\le n$. The easiest way to set up Boolean chains for multiplication is probably to use a scheme that goes back to John Napier's *Rabdologiæ* (Edinburgh, 1617), pages 137–143, as modernized by Luigi Dadda [*Alta Frequenza* **34** (1964), 349–356]: First we form all mn products $x_i\wedge y_j$, putting every such bit into $bin[i+j]$, which is one of $m+n$ "bins" that hold bits to be added for a particular power of 2 in the binary number system. The bins will contain respectively $(0,1,2,\ldots,m,m,\ldots,m,\ldots,2,1)$ bits at this point, with $n-m+1$ occurrences of "m" in the middle. Now we look at $bin[k]$ for $k=2,3,\ldots$. If $bin[k]$ contains a single bit b, we simply set $z_{k-1}\leftarrow b$. If it contains two bits $\{b,b'\}$, we use a half adder to compute $z_{k-1}\leftarrow b\oplus b'$, $c\leftarrow b\wedge b'$, and we put the carry bit c into $bin[k+1]$. Otherwise $bin[k]$ contains $t\ge 3$ bits; we choose any three of them, say $\{b,b',b''\}$, and remove them from the bin. With a full adder we then compute $r\leftarrow b\oplus b'\oplus b''$ and $c\leftarrow\langle bb'b''\rangle$, so that $b+b'+b''=r+2c$; and we put r into $bin[k]$, c into $bin[k+1]$. This decreases t by 2, so eventually we will have computed z_{k-1}. Exercise 41 quantifies the exact amount of calculation involved.

This method of encoding multiplication into clauses is quite flexible, since we're allowed to choose *any* three bits from $bin[k]$ whenever four or more bits are present. We could use a first-in-first-out strategy, always selecting bits from the "rear" and placing their sum at the "front"; or we could work last-in-first-out, essentially treating $bin[k]$ as a stack instead of a queue. We could also select the bits randomly, to see if this makes our SAT solver any happier. Later in this section we'll refer to the clauses that represent the factoring problem by calling them $\mathit{factor\_fifo}(m,n,z)$, $\mathit{factor\_lifo}(m,n,z)$, or $\mathit{factor\_rand}(m,n,z,s)$, respectively, where s is a seed for the random number generator used to generate them.

It's somewhat mind-boggling to realize that numbers can be factored without using any number theory! No greatest common divisors, no applications of Fermat's theorems, etc., are anywhere in sight. We're providing no hints to the solver except for a bunch of Boolean formulas that operate almost blindly at the bit level. Yet factors are found.

Of course we can't expect this method to compete with the sophisticated factorization algorithms of Section 4.5.4. But the problem of factoring does demonstrate the great versatility of clauses. And its clauses can be combined with *other* constraints that go well beyond any of the problems we've studied before.

Fault testing. Lots of things can go wrong when computer chips are manufactured in the "real world," so engineers have long been interested in constructing test patterns to check the validity of a particular circuit. For example, suppose that all but one of the logical elements are functioning properly in some chip; the bad one, however, is stuck: Its output is constant, always the same regardless of the inputs that it is given. Such a failure is called a *single-stuck-at fault.*

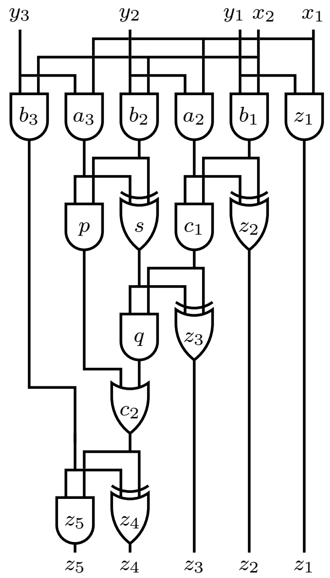


Fig. 34. A circuit that corresponds to (23).

Figure 34 illustrates a typical digital circuit in detail: It implements the 15 Boolean operations of (23) as a network that produces five output signals $z_5z_4z_3z_2z_1$ from the five inputs $y_3y_2y_1x_2x_1$. In addition to having 15 AND, OR, and XOR gates, each of which transforms two inputs into one output, it has 15 "fanout" gates (indicated by dots at junction points), each of which splits one input into two outputs. As a result it comprises 50 potentially distinct logical signals, one for each internal "wire." Exercise 47 shows that a circuit with m outputs, n inputs, and g conventional 2-to-1 gates will have $g+m-n$ fanout gates and $3g+2m-n$ wires. A circuit with w wires has $2w$ possible single-stuck-at faults, namely w faults in which the signal on a wire is stuck at 0 and w more on which it is stuck at 1.

Table 1 shows 101 scenarios that are possible when the 50 wires of Fig. 34 are activated by one particular sequence of inputs, assuming that at

Table 1

SINGLE-STUCK-AT FAULTS IN FIGURE 34 WHEN $x_2x_1 = 11$, $y_3y_2y_1 = 110$

	OK	x_1		x_1^1		x_1^2		x_1^3		x_1^4		x_2		x_2^1		x_2^2		x_2^3		x_2^4		y_1		y_1^1		y_1^2		y_2		y_2^1		y_2^2		y_3		y_3^1		y_3^2		z_1		a_2		a_2^1		a_2^2		a_3		a_3^1		a_3^2		b_1		b_1^1		b_1^2		b_2		b_2^1		b_2^2		b_3		b_3^1		b_3^2		z_2		c_1		c_1^1		c_1^2		s		s^1		s^2		p		z_3		q		c_2		c_2^1		c_2^2		z_4		z_5	
$x_1\leftarrow$input	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$x_1^1\leftarrow x_1$	1	0	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$x_1^2\leftarrow x_1$	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$x_1^3\leftarrow x_1^1$	1	0	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$x_1^4\leftarrow x_1^1$	1	0	1	0	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$x_2\leftarrow$input	1	1	1	1	1	1	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$x_2^1\leftarrow x_2$	1	1	1	1	1	1	1	1	1	1	1	0	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$x_2^2\leftarrow x_2$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$x_2^3\leftarrow x_2^1$	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$x_2^4\leftarrow x_2^1$	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$y_1\leftarrow$input	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$y_1^1\leftarrow y_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$y_1^2\leftarrow y_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$y_2\leftarrow$input	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$y_2^1\leftarrow y_2$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$y_2^2\leftarrow y_2$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$y_3\leftarrow$input	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$y_3^1\leftarrow y_3$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$y_3^2\leftarrow y_3$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$z_1\leftarrow x_1^2\wedge y_1^1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$a_2\leftarrow x_1^3\wedge y_2^1$	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$a_2^1\leftarrow a_2$	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$a_2^2\leftarrow a_2$	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$a_3\leftarrow x_1^4\wedge y_3^1$	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$a_3^1\leftarrow a_3$	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$a_3^2\leftarrow a_3$	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$b_1\leftarrow x_2^2\wedge y_1^2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$b_1^1\leftarrow b_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$b_1^2\leftarrow b_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$b_2\leftarrow x_2^3\wedge y_2^2$	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$b_2^1\leftarrow b_2$	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$b_2^2\leftarrow b_2$	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$b_3\leftarrow x_2^4\wedge y_3^2$	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$b_3^1\leftarrow b_3$	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$b_3^2\leftarrow b_3$	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$z_2\leftarrow a_2^1\oplus b_1^1$	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$c_1\leftarrow a_2^2\wedge b_1^2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$c_1^1\leftarrow c_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$c_1^2\leftarrow c_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$s\leftarrow a_3^1\oplus b_2^1$	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$s^1\leftarrow s$	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$s^2\leftarrow s$	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$p\leftarrow a_3^2\wedge b_2^2$	1	0	1	0	1	1	1	1	1	0	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$z_3\leftarrow s^1\oplus c_1^1$	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0
$q\leftarrow s^2\wedge c_1^2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**1**	0	0	0	0	0	0	0	0	0	0
$c_2\leftarrow p\vee q$	1	0	1	0	1	1	1	1	1	0	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	**0**	**1**	1	1	1	1	1	1	1	1
$c_2^1\leftarrow c_2$	1	0	1	0	1	1	1	1	1	0	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	**0**	**1**	1	1	1	1	1	1
$c_2^2\leftarrow c_2$	1	0	1	0	1	1	1	1	1	0	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	**0**	**1**	1	1	1	1
$z_4\leftarrow b_3^1\oplus c_2^1$	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	**0**	**1**	0	0
$z_5\leftarrow b_3^2\wedge c_2^2$	1	0	1	0	1	1	1	1	1	0	1	0	1	0	1	1	1	0	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	0	1	1	1	**0**	**1**

most one stuck-at fault is present. The column headed OK shows the correct behavior of the Boolean chain (which nicely multiplies $x = 3$ by $y = 6$ and obtains $z = 18$). We can call these the "default" values, because, well, they have no faults. The other 100 columns show what happens if all but one of the 50 wires have error-free signals; the two columns under b_2^1, for example, illustrate the results when the rightmost wire that fans out from gate b_2 is stuck at 0 or 1. Each row is obtained bitwise from previous rows or inputs, except that the boldface digits are forced. When a boldface value agrees with the default, its entire column is correct; otherwise errors might propagate. All values above the bold diagonal match the defaults.

If we want to test a chip that has n inputs and m outputs, we're allowed to apply test patterns to the inputs and see what outputs are produced. Close inspection shows, for instance, that the pattern considered in Table 1 doesn't detect an error when q is stuck at 1, even though q should be 0, because all five output bits $z_5z_4z_3z_2z_1$ are correct in spite of that error. In fact, the value of $c_2 \leftarrow p \vee q$ is unaffected by a bad q, because $p = 1$ in this example. Similarly, the fault "x_1^2 stuck at 0" doesn't propagate into $z_1 \leftarrow x_1^2 \wedge y_1^1$ because $y_1^1 = 0$. Altogether 44 faults, not 50, are discovered by this particular test pattern.

All of the relevant repeatable faults, whether they're single-stuck-at or wildly complicated, could obviously be discovered by testing all 2^n possible patterns. But that's out of the question unless n is quite small. Fortunately, testing isn't hopeless, because satisfactory results are usually obtained in practice if we do have enough test patterns to detect all of the detectable single-stuck-at faults. Exercise 49 shows that just five patterns suffice to certify Fig. 34 by this criterion.

The detailed analysis in exercise 49 also shows, surprisingly, that one of the faults, namely "s^2 stuck at 1," cannot be detected! Indeed, an erroneous s^2 can propagate to an erroneous q only if $c_1^2 = 1$, and that forces $x_1 = x_2 = y_1 = y_2 = 1$; only two possibilities remain, and neither $y_3 = 0$ nor $y_3 = 1$ reveals the fault. Consequently we can simplify the circuit by removing gate q; the chain (23) becomes shorter, with "$q \leftarrow s \wedge c_1$, $c_2 \leftarrow p \vee q$" replaced by "$c_2 \leftarrow p \vee c_1$."

Of course Fig. 34 is just a tiny little circuit, intended only to introduce the concept of stuck-at faults. Test patterns are needed for the much larger circuits that arise in real computers; and we will see that SAT solvers can help us to find them. Consider, for example, the generic multiplier circuit *prod*(m, n), which is part of the Stanford GraphBase. It multiplies an m-bit number x by an n-bit number y, producing an $(m + n)$-bit product z. Furthermore, it's a so-called "parallel multiplier," with delay time $O(\log(m+n))$; thus it's much more suited to hardware design than methods like the *factor_fifo* schemes that we considered above, because those circuits need $\Omega(m + n)$ time for carries to propagate.

Let's try to find test patterns that will smoke out all of the single-stuck-at faults in *prod*$(32, 32)$, which is a circuit of depth 33 that has 64 inputs, 64 outputs, 3660 AND gates, 1203 OR gates, 2145 XOR gates, and (therefore) 7008 fanout gates and 21,088 wires. How can we guard it against 42,176 different faults?

Before we construct clauses to facilitate that task, we should realize that most of the single-stuck-at faults are easily detected by choosing patterns at random, since faults usually cause big trouble and are hard to miss. Indeed, choosing $x =$ #3243F6A8 and $y =$ #885A308D more or less at random already eliminates 14,733 cases; and $(x, y) =$ (#2B7E1516, #28AED2A6) eliminates 6,918 more. We might as well keep doing this, because bitwise operations such as those in Table 1 are fast. Experience with the smaller multiplier in Fig. 34 suggests that we get more effective tests if we bias the inputs, choosing each bit to be 1 with probability .9 instead of .5 (see exercise 49). A million such random inputs will then generate, say, 243 patterns that detect all but 140 of the faults.

Our remaining job, then, is essentially to find 140 needles in a haystack of size 2^{64}, after having picked $42{,}176 - 140 = 42{,}036$ pieces of low-hanging fruit. And that's where a SAT solver is useful. Consider, for example, the analogous but simpler problem of finding a test pattern for "q stuck at 0" in Fig. 34.

We can use the 49 clauses F derived from (23) to represent the well-behaved circuit; and we can imagine corresponding clauses F' that represent the faulty computation, using "primed" variables z_1', a_2', $\ldots$, z_5'. Thus F' begins with $(x_1 \vee \bar{z}_1') \wedge (y_1 \vee \bar{z}_1')$ and ends with $(\bar{b}_3' \vee \bar{c}_2' \vee z_5')$; it's like F except that the clauses representing $q' \leftarrow s' \wedge c_1'$ in (23) are changed to simply $\bar{q}'$ (meaning that q' is stuck at 0). Then the clauses of F and F', together with a few more clauses to state that $z_1 \neq z_1'$ or $\cdots$ or $z_5 \neq z_5'$, will be satisfiable only by variables for which $(y_3y_2y_1)_2 \times (x_2x_1)_2$ is a suitable test pattern for the given fault.

This construction of F' can obviously be simplified, because z_1' is identical to z_1; any signal that differs from the correct value must be located "downstream" from the one-and-only fault. Let's say that a wire is *tarnished* if it is the faulty wire or if at least one of its input wires is tarnished. We introduce new variables g' only for wires g that are tarnished. Thus, in our example, the only clauses F' that are needed to extend F to a faulty companion circuit are $\bar{q}'$ and the clauses that correspond to $c_2' \leftarrow p \vee q'$, $z_4' \leftarrow b_3 \oplus c_2'$, $z_5' \leftarrow b_3 \wedge c_2'$.

Moreover, any fault that is revealed by a test pattern must have an *active path* of wires, leading from the fault to an output; all wires on this path must carry a faulty signal. Therefore Tracy Larrabee [*IEEE Trans.* **CAD-11** (1992), 4–15] decided to introduce additional "sharped" variables $g^\sharp$ for each tarnished wire, meaning that g lies on the active path. The two clauses

$$(\bar{g}^\sharp \vee g \vee g') \wedge (\bar{g}^\sharp \vee \bar{g} \vee \bar{g}') \tag{25}$$

ensure that $g \neq g'$ whenever g is part of that path. Furthermore we have $(\bar{v}^\sharp \vee g^\sharp)$ whenever g is an AND, OR, or XOR gate with tarnished input v. Fanout gates are slightly tricky in this regard: When wires g^1 and g^2 fan out from a tarnished wire g, we need variables $g^{1\sharp}$ and $g^{2\sharp}$ as well as $g^\sharp$; and we introduce the clause

$$(\bar{g}^\sharp \vee g^{1\sharp} \vee g^{2\sharp}) \tag{26}$$

to specify that the active path takes at least one of the two branches.

According to these rules, our example acquires the new variables $q^\sharp$, $c_2^\sharp$, $c_2^{1\sharp}$, $c_2^{2\sharp}$, $z_4^\sharp$, $z_5^\sharp$, and the new clauses

$$(\bar{q}^\sharp \vee q \vee q') \wedge (\bar{q}^\sharp \vee \bar{q} \vee \bar{q}') \wedge (\bar{q}^\sharp \vee c_2^\sharp) \wedge (\bar{c}_2^\sharp \vee c_2 \vee c_2') \wedge (\bar{c}_2^\sharp \vee \bar{c}_2 \vee \bar{c}_2') \wedge (\bar{c}_2^\sharp \vee c_2^{1\sharp} \vee c_2^{2\sharp}) \wedge$$
$$(\bar{c}_2^{1\sharp} \vee z_4^\sharp) \wedge (\bar{z}_4^\sharp \vee z_4 \vee z_4') \wedge (\bar{z}_4^\sharp \vee \bar{z}_4 \vee \bar{z}_4') \wedge (\bar{c}_2^{2\sharp} \vee z_5^\sharp) \wedge (\bar{z}_5^\sharp \vee z_5 \vee z_5') \wedge (\bar{z}_5^\sharp \vee \bar{z}_5 \vee \bar{z}_5').$$

The active path begins at q, so we assert the unit clause $(q^\sharp)$; it ends at a tarnished output, so we also assert $(z_4^\sharp \vee z_5^\sharp)$. The resulting set of clauses will find a test pattern for this fault if and only if the fault is detectable. Larrabee found that such active-path variables provide important clues to a SAT solver and significantly speed up the solution process.

Returning to the large circuit *prod*$(32, 32)$, one of the 140 hard-to-test faults is "W_{21}^{26} stuck at 1," where W_{21}^{26} denotes the 26th extra wire that fans out from the OR gate called W_{21} in §75 of the Stanford GraphBase program GB_GATES; W_{21}^{26} is an input to gate $b_{40}^{40} \leftarrow d_{40}^{19} \wedge W_{21}^{26}$ in §80 of that program. Test patterns for that fault can be characterized by a set of 23,194 clauses in 7,082 variables (of which only 4 variables are "primed" and 4 are "sharped"). Fortunately the solution $(x, y) = ($#7F13FEDD$,$ #5FE57FFE$)$ was found rather quickly in the author's experiments; and this pattern also killed off 13 of the other cases, so the score was now "14 down and 126 to go"!

The next fault sought was "$A_5^{36,2}$ stuck at 1," where $A_5^{36,2}$ is the second extra wire to fan out from the AND gate A_5^{36} in §72 of GB_GATES (an input to $R_{11}^{36} \leftarrow A_5^{36,2} \wedge R_1^{35,2}$). This fault corresponds to 26,131 clauses on 8,342 variables; but the SAT solver took a quick look at those clauses and decided almost instantly that they are *unsatisfiable*. Therefore the fault is undetectable, and the circuit *prod*$(32, 32)$ can be simplified by setting $R_{11}^{36} \leftarrow R_1^{35,2}$. A closer look showed, in fact, that clauses corresponding to the Boolean equations

$$x = y \wedge z, \quad y = v \wedge w, \quad z = t \wedge u, \quad u = v \oplus w$$

were present (where $t = R_{13}^{44}$, $u = A_{58}^{45}$, $v = R_4^{44}$, $w = A_{14}^{45}$, $x = R_{23}^{46}$, $y = R_{13}^{45}$, $z = R_{19}^{45}$); these clauses *force* $x = 0$. Therefore it was not surprising to find that the list of unresolved faults also included R_{23}^{46}, $R_{23}^{46,1}$ and $R_{23}^{46,2}$ stuck at 0. Altogether 26 of the 140 faults undetected by random inputs turned out to be *absolutely* undetectable; and only one of these, namely "Q_{26}^{46} stuck at 0," required a nontrivial proof of undetectability.

Some of the $126-26 = 100$ faults remaining on the to-do list turned out to be significant challenges for the SAT solver. While waiting, the author therefore had time to take a look at a few of the previously found solutions, and noticed that those patterns themselves were forming a pattern! Sure enough, the extreme portions of this large and complicated circuit actually have a fairly simple structure, stuck-at-fault-wise. Hence number theory came to the rescue: The factorization #87FBC059 $\times$ #F0F87817 $= 2^{63} - 1$ solved many of the toughest challenges, some of which occur with probability less than 2^{-34} when 32-bit numbers are multiplied; and the "Aurifeuillian" factorization $(2^{31} - 2^{16} + 1)(2^{31} + 2^{16} + 1) = 2^{62} + 1$, which the author had known for more than forty years (see Eq. 4.5.4–(15)), polished off most of the others.

The bottom line (see exercise 51) is that all 42,150 of the detectable single-stuck-at faults of the parallel multiplication circuit *prod*$(32, 32)$ can actually be detected with at most 196 well-chosen test patterns.

Learning a Boolean function. Sometimes we're given a "black box" that evaluates a Boolean function $f(x_1, \ldots, x_N)$. We have no way to open the box, but we suspect that the function is actually quite simple. By plugging in various values for $x = x_1 \ldots x_N$, we can observe the box's behavior and possibly learn the hidden rule that lies inside. For example, a secret function of $N = 20$ Boolean variables might take on the values shown in Table 2, which lists 16 cases where $f(x) = 1$ and 16 cases where $f(x) = 0$.

Suppose we assume that the function has a DNF (disjunctive normal form) with only a few terms. We'll see in a moment that it's easy to express such an assumption as a satisfiability problem. And when the author constructed clauses corresponding to Table 2 and presented them to a SAT solver, he did in fact learn

Table 2

VALUES TAKEN ON BY AN UNKNOWN FUNCTION

Cases where $f(x) = 1$	Cases where $f(x) = 0$
$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9$ … x_{20}	$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9$ … x_{20}
1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 1	1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 0 0 1 0 1
1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1	0 1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0 1 0
0 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1	1 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 0 1
0 1 0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0	1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 0
0 1 1 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0 0 0	0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0
0 0 0 0 1 1 0 1 1 1 0 0 0 0 0 1 1 1 0 0	0 1 1 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 0
1 1 0 1 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0	1 1 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 1 1
0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 0 0	1 0 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1
1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 1 1 1 0 0	1 1 0 0 1 1 1 0 0 0 1 0 1 1 0 1 0 0 1 1
1 1 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0	0 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 0 0 1
0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 0	1 1 1 0 0 0 0 1 0 0 1 1 0 1 1 0 0 1 0 0
0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 0 1 1	0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0 1 0 0
1 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 0 1 0 1	0 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 0
0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0	1 1 0 0 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1
0 1 1 1 1 0 0 1 1 0 0 0 1 1 1 0 0 0 1 1	1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 0 1 0 0 1
0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 1 1 1 0 1	1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 0 1

almost immediately that a very simple formula is consistent with all of the data:

$$f(x_1,\ldots,x_{20}) = \bar{x}_2\bar{x}_3\bar{x}_{10} \vee \bar{x}_6\bar{x}_{10}\bar{x}_{12} \vee x_8\bar{x}_{13}\bar{x}_{15} \vee \bar{x}_8 x_{10}\bar{x}_{12}. \qquad (27)$$

This formula was discovered by constructing clauses in $2MN$ variables $p_{i,j}$ and $q_{i,j}$ for $1 \le i \le M$ and $1 \le j \le N$, where M is the maximum number of terms allowed in the DNF (here $M = 4$) and where

$$p_{i,j} \;=\; [\text{term } i \text{ contains } x_j], \qquad q_{i,j} \;=\; [\text{term } i \text{ contains } \bar{x}_j]. \qquad (28)$$

If the function is constrained to equal 1 at P specified points, we also use auxiliary variables $z_{i,k}$ for $1 \le i \le M$ and $1 \le k \le P$, one for each term at every such point.

Table 2 says that $f(1,1,0,0,\ldots,1) = 1$, and we can capture this specification by constructing the clause

$$(z_{1,1} \vee z_{2,1} \vee \cdots \vee z_{M,1}) \qquad (29)$$

together with the clauses

$$(\bar{z}_{i,1} \vee \bar{q}_{i,1}) \wedge (\bar{z}_{i,1} \vee \bar{q}_{i,2}) \wedge (\bar{z}_{i,1} \vee \bar{p}_{i,3}) \wedge (\bar{z}_{i,1} \vee \bar{p}_{i,4}) \wedge \cdots \wedge (\bar{z}_{i,1} \vee \bar{q}_{i,20}) \qquad (30)$$

for $1 \le i \le M$. Translation: (29) says that at least one of the terms in the DNF must evaluate to true; and (30) says that, if term i is true at the point $1100\ldots1$, it cannot contain $\bar{x}_1$ or $\bar{x}_2$ or x_3 or x_4 or $\cdots$ or $\bar{x}_{20}$.

Table 2 also tells us that $f(1,0,1,0,\ldots,1) = 0$. This specification corresponds to the clauses

$$(q_{i,1} \vee p_{i,2} \vee q_{i,3} \vee p_{i,4} \vee \cdots \vee q_{i,20}) \qquad (31)$$

for $1 \le i \le M$. (Each term of the DNF must be zero at the given point; thus either $\bar{x}_1$ or x_2 or $\bar{x}_3$ or x_4 or $\cdots$ or $\bar{x}_{20}$ must be present for each value of i.)

In general, every case where $f(x) = 1$ yields one clause like (29) of length M, plus MN clauses like (30) of length 2. Every case where $f(x) = 0$ yields M clauses like (31) of length N. We use $q_{i,j}$ when $x_j = 1$ at the point in question, and $p_{i,j}$ when $x_j = 0$, for both (30) and (31). This construction is due to A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende [*Mathematical Programming* **57** (1992), 215–238], who presented many examples. From Table 2, with $M = 4$, $N = 20$, and $P = 16$, it generates 1360 clauses of total length 3904 in 224 variables; a SAT solver then finds a solution with $p_{1,1} = q_{1,1} = p_{1,2} = 0$, $q_{1,2} = 1$, ..., leading to (27).

The simplicity of (27) makes it plausible that the SAT solver has indeed psyched out the true nature of the hidden function $f(x)$. The chance of agreeing with the correct value 32 times out of 32 is only 1 in 2^{32}, so we seem to have overwhelming evidence in favor of that equation.

But no: Such reasoning is fallacious. The numbers in Table 2 actually arose in a completely different way, and Eq. (27) has essentially *no* credibility as a predictor of $f(x)$ for any other values of x! (See exercise 53.) The fallacy comes from the fact that short-DNF Boolean functions of 20 variables are not at all rare; there are many more than 2^{32} of them.

On the other hand, when we *do* know that the hidden function $f(x)$ has a DNF with at most M terms (although we know nothing else about it), the clauses (29)–(31) give us a nice way to discover those terms, provided that we also have a sufficiently large and unbiased "training set" of observed values.

For example, let's assume that (27) actually *is* the function in the box. If we examine $f(x)$ at 32 random points x, we don't have enough data to make any deductions. But 100 random training points will almost always home in on the correct solution (27). This calculation typically involves 3942 clauses in 344 variables; yet it goes quickly, needing only about 100 million accesses to memory.

One of the author's experiments with a 100-element training set yielded

$$\hat{f}(x_1,\ldots,x_{20}) = \bar{x}_2\bar{x}_3\bar{x}_{10} \lor x_3\bar{x}_6\bar{x}_{10}\bar{x}_{12} \lor x_8\bar{x}_{13}\bar{x}_{15} \lor \bar{x}_8x_{10}\bar{x}_{12}, \qquad (32)$$

which is close to the truth but not quite exact. (Exercise 59 proves that $\hat{f}(x)$ is equal to $f(x)$ more than 97% of the time.) Further study of this example showed that another nine training points were enough to deduce $f(x)$ uniquely, thus obtaining 100% confidence (see exercise 61).

Bounded model checking. Some of the most important applications of SAT solvers in practice are related to the verification of hardware or software, because designers generally want some kind of assurance that particular implementations correctly meet their specifications.

A typical design can usually be modeled as a *transition relation* between Boolean vectors $X = x_1 \ldots x_n$ that represent the possible states of a system. We write $X \to X'$ if state X at time t can be followed by state X' at time $t + 1$. The task in general is to study sequences of state transitions

$$X_0 \to X_1 \to X_2 \to \cdots \to X_r, \qquad (33)$$

and to decide whether or not there are sequences that have special properties. For example, we hope that there's no such sequence for which X_0 is an "initial state" and X_r is an "error state"; otherwise there'd be a bug in the design.

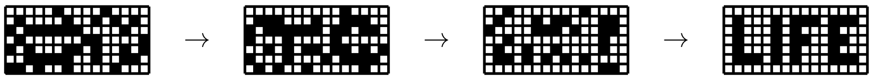


Fig. 35. Conway's rule (35) defines these three successive transitions.

Questions like this are readily expressed as satisfiability problems: Each state X_t is a vector of Boolean variables $x_{t1} \ldots x_{tn}$, and each transition relation can be represented by a set of m clauses $T(X_t, X_{t+1})$ that must be satisfied. These clauses $T(X, X')$ involve $2n$ variables $\{x_1, \ldots, x_n, x'_1, \ldots, x'_n\}$, together with q auxiliary variables $\{y_1, \ldots, y_q\}$ that might be needed to express Boolean formulas in clause form as we did with the Tseytin encodings in (24). Then the existence of sequence (33) is equivalent to the satisfiability of mr clauses

$$T(X_0, X_1) \wedge T(X_1, X_2) \wedge \cdots \wedge T(X_{r-1}, X_r) \tag{34}$$

in the $n(r+1)+qr$ variables $\{x_{tj} \mid 0 \le t \le r,\ 1 \le j \le n\} \cup \{y_{tk} \mid 0 \le t < r,\ 1 \le k \le q\}$. We've essentially "unrolled" the sequence (33) into r copies of the transition relation, using variables x_{tj} for state X_t and y_{tk} for the auxiliary quantities in $T(X_t, X_{t+1})$. Additional clauses can now be added to specify constraints on the initial state X_0 and/or the final state X_r, as well as any other conditions that we want to impose on the sequence.

This general setup is called "bounded model checking," because we're using it to check properties of a model (a transition relation), and because we're considering only sequences that have a bounded number of transitions, r.

John Conway's fascinating *Game of Life* provides a particularly instructive set of examples that illustrate basic principles of bounded model checking. The states X of this game are two-dimensional bitmaps, corresponding to arrays of square cells that are either alive (1) or dead (0). Every bitmap X has a unique successor X', determined by the action of a simple 3×3 cellular automaton: Suppose cell x has the eight neighbors $\{x_{\rm NW}, x_{\rm N}, x_{\rm NE}, x_{\rm W}, x_{\rm E}, x_{\rm SW}, x_{\rm S}, x_{\rm SE}\}$, and let $\nu = x_{\rm NW} + x_{\rm N} + x_{\rm NE} + x_{\rm W} + x_{\rm E} + x_{\rm SW} + x_{\rm S} + x_{\rm SE}$ be the number of neighbors that are alive at time t. Then x is alive at time $t+1$ if and only if either (a) $\nu = 3$, or (b) $\nu = 2$ and x is alive at time t. Equivalently, the transition rule

$$x' \;=\; [2 < x_{\rm NW} + x_{\rm N} + x_{\rm NE} + x_{\rm W} + \tfrac{1}{2}x + x_{\rm E} + x_{\rm SW} + x_{\rm S} + x_{\rm SE} < 4] \tag{35}$$

holds at every cell x. (See, for example, Fig. 35, where the live cells are black.)

Conway called Life a "no-player game," because it involves no strategy: Once an initial state X_0 has been set up, all subsequent states X_1, X_2, ... are completely determined. Yet, in spite of the simple rules, he also proved that Life is inherently complicated and unpredictable, indeed beyond human comprehension, in the sense that it is universal: *Every finite, discrete, deterministic system, however complex, can be simulated faithfully by some finite initial state X_0 of Life.* [See Berlekamp, Conway, and Guy, *Winning Ways* (2004), Chapter 25.]

In exercises 7.1.4–160 through 162, we've already seen some of the amazing Life histories that are possible, using BDD methods. And many further aspects of Life can be explored with SAT methods, because SAT solvers can often deal with many more variables. For example, Fig. 35 was discovered by using $7\times 15 = 105$ variables for each state X_0, X_1, X_2, X_3. The values of X_3 were obviously predetermined; but the other $105\times 3 = 315$ variables had to be computed, and BDDs can't handle that many. Moreover, additional variables were introduced to ensure that the initial state X_0 would have as few live cells as possible.

Here's the story behind Fig. 35, in more detail: Since Life is two-dimensional, we use variables x_{ij} instead of x_j to indicate the states of individual cells, and x_{tij} instead of x_{tj} to indicate the states of cells at time t. We generally assume that $x_{tij} = 0$ for all cells outside of a given finite region, although the transition rule (35) can allow cells that are arbitrarily far away to become alive as Life goes on. In Fig. 35 the region was specified to be a 7×15 rectangle at each unit of time. Furthermore, configurations with three consecutive live cells on a boundary edge were forbidden, so that cells "outside the box" wouldn't be activated.

The transitions $T(X_t, X_{t+1})$ can be encoded without introducing additional variables, but only if we introduce 190 rather long clauses for each cell not on the boundary. There's a better way, based on the binary tree approach underlying (20) and (21) above, which requires only about 63 clauses of size ≤ 3, together with about 14 auxiliary variables per cell. This approach (see exercise 65) takes advantage of the fact that many intermediate calculations can be shared. For example, cells x and x_{W} have four neighbors $\{x_{\mathrm{NW}}, x_{\mathrm{N}}, x_{\mathrm{SW}}, x_{\mathrm{S}}\}$ in common; so we need to compute $x_{\mathrm{NW}} + x_{\mathrm{N}} + x_{\mathrm{SW}} + x_{\mathrm{S}}$ only once, not twice.

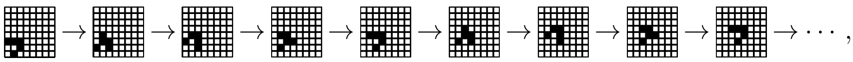
The clauses that correspond to a four-step sequence $X_0 \to X_1 \to X_2 \to X_3 \to X_4$ leading to $X_4 =$ **LIFE** turn out to be unsatisfiable without going outside of the 7×15 frame. (Only 10 gigamems of calculation were needed to establish this fact, using Algorithm C below, even though roughly 34000 clauses in 9000 variables needed to be examined!) So the next step in the preparation of Fig. 35 was to try $X_3 =$ **LIFE**; and this trial succeeded. Additional clauses, which permitted X_0 to have at most 39 live cells, led to the solution shown, at a cost of about 17 gigamems; and that solution is optimum, because a further run (costing 12 gigamems) proved that there's no solution with at most 38.

Let's look for a moment at some of the patterns that can occur on a chessboard, an 8×8 grid. Human beings will never be able to contemplate more than a tiny fraction of the 2^{64} states that are possible; so we can be fairly sure that "Lifenthusiasts" haven't already explored every tantalizing configuration that exists, even on such a small playing field.

One nice way to look for a sequence of interesting Life transitions is to assert that no cell stays alive more than four steps in a row. Let us therefore say that a *mobile* Life path is a sequence of transitions $X_0 \to X_1 \to \cdots \to X_r$ with the additional property that we have

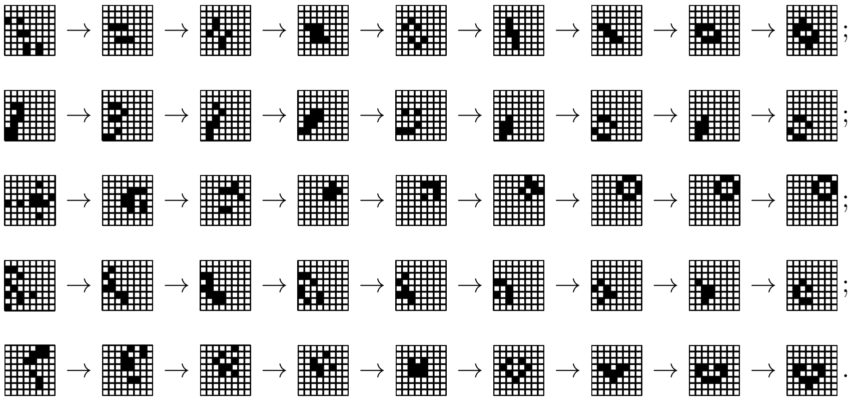
$$(\bar{x}_{tij} \vee \bar{x}_{(t+1)ij} \vee \bar{x}_{(t+2)ij} \vee \bar{x}_{(t+3)ij} \vee \bar{x}_{(t+4)ij}), \qquad \text{for } 0 \le t \le r-4. \qquad (36)$$

To avoid trivial solutions we also insist that X_r is not entirely dead. For example, if we impose rule (36) on a chessboard, with x_{tij} permitted to be alive only if $1 \le i, j \le 8$, and with the further condition that at most five cells are alive in each generation, a SAT solver can quickly discover interesting mobile paths such as


(37)

which last quite awhile before leaving the board. And indeed, the five-celled object that moves so gracefully in this path is R. K. Guy's famous *glider* (1970), which is surely the most interesting small creature in Life's universe. The glider moves diagonally, recreating a shifted copy of itself after every four steps.

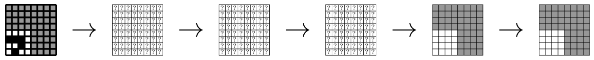
Interesting mobile paths appear also if we restrict the population at each time to $\{6,7,8,9,10\}$ instead of $\{1,2,3,4,5\}$. For example, here are some of the first such paths that the author's solver came up with, having length $r = 8$:



These paths illustrate the fact that symmetry can be gained, but never lost, as Life evolves deterministically. Marvelous designs are spawned in the process. In each of these sequences the next bitmap, X_9, would break our ground rules: The population immediately after X_8 grows to 12 in the first and last examples, but shrinks to 5 in the second-from-last; and the path becomes immobile in the other two. Indeed, we have $X_5 = X_7$ in the second example, hence $X_6 = X_8$ and $X_7 = X_9$, etc. Such a repeating pattern is called an *oscillator* of period 2. The third example ends with an oscillator of period 1, known as a "still life."

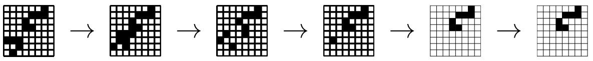
What are the ultimate destinations of these paths? The first one becomes still, with $X_{69} = X_{70}$; and the fourth becomes *very* still, with $X_{12} = 0$! The fifth is the most fascinating of the group, because it continues to produce ever more elaborate valentine shapes, then proceeds to dance and sparkle, until finally beginning to twinkle with period 2 starting at time 177. Thus its members X_2 through X_7 qualify as "Methuselahs," defined by Martin Gardner as "Life patterns of population less than 10 that do not become stable within 50 generations." (A predictable pattern, like the glider or an oscillator, is called *stable*.)

SAT solvers are basically useless for the study of Methuselahs, because the state space becomes too large. But they are quite helpful when we want to illuminate many other aspects of Life, and exercises 66–85 discuss some notable instances. We will consider one more instructive example before moving on, namely an application to “eaters.” Consider a Life path of the form

$X_0 =$  $= X_5,$ (38)

where the gray cells form a still life and the cells of X_1, X_2, X_3 are unknown. Thus $X_4 = X_5$ and $X_0 = X_5 +$ glider. Furthermore we require that the still life X_5 does not interact with the glider’s parent, ; see exercise 77. The idea is that a glider will be gobbled up if it happens to glide into this particular still life, and the still life will rapidly reconstitute itself as if nothing had happened.

Algorithm C almost instantaneously (well, after about 100 megamems) finds

 (39)

the four-step eater first observed in action by R. W. Gosper in 1971.

Applications to mutual exclusion. Let’s look now at how bounded model checking can help us to prove that algorithms are correct. (Or incorrect.) Some of the most challenging issues of verification arise when we consider parallel processes that need to synchronize their concurrent behavior. To simplify our discussion it will be convenient to tell a little story about Alice and Bob.

Alice and Bob are casual friends who share an apartment. One of their joint rooms is special: When they’re in that critical room, which has two doors, they don’t want the other person to be present. Furthermore, being busy people, they don’t want to interrupt each other needlessly. So they agree to control access to the room by using an indicator light, which can be switched on or off.

The first protocol they tried can be characterized by symmetrical algorithms:

```
A0. Maybe go to A1.                    B0. Maybe go to B1.
A1. If $l$ go to A1, else to A2.       B1. If $l$ go to B1, else to B2.
A2. Set $l \leftarrow 1$, go to A3.    B2. Set $l \leftarrow 1$, go to B3.    (40)
A3. Critical, go to A4.                B3. Critical, go to B4.
A4. Set $l \leftarrow 0$, go to A0.    B4. Set $l \leftarrow 0$, go to B0.
```

At any instant of time, Alice is in one of five states, $\{A0, A1, A2, A3, A4\}$, and the rules of her program show how that state might change. In state A0 she isn’t interested in the critical room; but she goes to A1 when she does wish to use it. She reaches that objective in state A3. Similar remarks apply to Bob. When the indicator light is on ($l = 1$), they wait until the other person has exited the room and switched the light back off ($l = 0$).

Alice and Bob don’t necessarily operate at the same speed. But they’re allowed to dawdle only when in the “maybe” state A0 or B0. More precisely, we model the situation by converting every relevant scenario into a discrete sequence of state transitions. At every time $t = 0, 1, 2, \ldots,$ either Alice or Bob (but not both) will perform the command associated with their current state, thereby perhaps changing to a different state at time $t + 1$. This choice is nondeterministic.

Only four kinds of primitive commands are permitted in the procedures we shall study, all of which are illustrated in (40): (1) “Maybe go to s”; (2) “Critical, go to s”; (3) “Set $v \leftarrow b$, go to s”; and (4) “If v go to s_1, else to s_0”. Here s denotes a state name, v denotes a shared Boolean variable, and b is 0 or 1.

Unfortunately, Alice and Bob soon learned that protocol (40) is unreliable: One day she went from A1 to A2 and he went from B1 to B2, before either of them had switched the indicator on. Embarrassment (A3 and B3) followed.

They could have discovered this problem in advance, if they’d converted the state transitions of (40) into clauses for bounded model checking, as in (33), then applied a SAT solver. In this case the vector X_t that corresponds to time t consists of Boolean variables that encode each of their current states, as well as the current value of l. We can, for example, have eleven variables $\mathrm{A0}_t$, $\mathrm{A1}_t$, $\mathrm{A2}_t$, $\mathrm{A3}_t$, $\mathrm{A4}_t$, $\mathrm{B0}_t$, $\mathrm{B1}_t$, $\mathrm{B2}_t$, $\mathrm{B3}_t$, $\mathrm{B4}_t$, l_t, together with ten binary exclusion clauses $(\overline{\mathrm{A0}_t} \vee \overline{\mathrm{A1}_t})$, $(\overline{\mathrm{A0}_t} \vee \overline{\mathrm{A2}_t})$, $\ldots$, $(\overline{\mathrm{A3}_t} \vee \overline{\mathrm{A4}_t})$ to ensure that Alice is in at most one state, and with ten similar clauses for Bob. There’s also a variable $@_t$, which is true or false depending on whether Alice or Bob executes their program step at time t. (We say that Alice was “bumped” if $@_t = 1$, and Bob was bumped if $@_t = 0$.)

If we start with the initial state X_0 defined by unit clauses

$$\mathrm{A0}_0 \wedge \overline{\mathrm{A1}_0} \wedge \overline{\mathrm{A2}_0} \wedge \overline{\mathrm{A3}_0} \wedge \overline{\mathrm{A4}_0} \wedge \mathrm{B0}_0 \wedge \overline{\mathrm{B1}_0} \wedge \overline{\mathrm{B2}_0} \wedge \overline{\mathrm{B3}_0} \wedge \overline{\mathrm{B4}_0} \wedge \bar{l}_0, \qquad (41)$$

the following clauses for $0 \le t < r$ (discussed in exercise 87) will emulate the first r steps of every legitimate scenario defined by (40):

$$\begin{array}{lll}
(@_t \vee \overline{\mathrm{A0}_t} \vee \mathrm{A0}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A0}_t} \vee \mathrm{A0}_{t+1} \vee \mathrm{A1}_{t+1}) & (@_t \vee \overline{\mathrm{B0}_t} \vee \mathrm{B0}_{t+1} \vee \mathrm{B1}_{t+1}) \\
(@_t \vee \overline{\mathrm{A1}_t} \vee \mathrm{A1}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A1}_t} \vee \bar{l}_t \vee \mathrm{A1}_{t+1}) & (@_t \vee \overline{\mathrm{B1}_t} \vee \bar{l}_t \vee \mathrm{B1}_{t+1}) \\
(@_t \vee \overline{\mathrm{A2}_t} \vee \mathrm{A2}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A1}_t} \vee l_t \vee \mathrm{A2}_{t+1}) & (@_t \vee \overline{\mathrm{B1}_t} \vee l_t \vee \mathrm{B2}_{t+1}) \\
(@_t \vee \overline{\mathrm{A3}_t} \vee \mathrm{A3}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A2}_t} \vee \mathrm{A3}_{t+1}) & (@_t \vee \overline{\mathrm{B2}_t} \vee \mathrm{B3}_{t+1}) \\
(@_t \vee \overline{\mathrm{A4}_t} \vee \mathrm{A4}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A2}_t} \vee l_{t+1}) & (@_t \vee \overline{\mathrm{B2}_t} \vee l_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B0}_t} \vee \mathrm{B0}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A3}_t} \vee \mathrm{A4}_{t+1}) & (@_t \vee \overline{\mathrm{B3}_t} \vee \mathrm{B4}_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B1}_t} \vee \mathrm{B1}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A4}_t} \vee \mathrm{A0}_{t+1}) & (@_t \vee \overline{\mathrm{B4}_t} \vee \mathrm{B0}_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B2}_t} \vee \mathrm{B2}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A4}_t} \vee \bar{l}_{t+1}) & (@_t \vee \overline{\mathrm{B4}_t} \vee \bar{l}_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B3}_t} \vee \mathrm{B3}_{t+1}) & (\overline{@_t} \vee l_t \vee \mathrm{A2}_t \vee \mathrm{A4}_t \vee \bar{l}_{t+1}) & (@_t \vee l_t \vee \mathrm{B2}_t \vee \mathrm{B4}_t \vee \bar{l}_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B4}_t} \vee \mathrm{B4}_{t+1}) & (\overline{@_t} \vee \bar{l}_t \vee \mathrm{A2}_t \vee \mathrm{A4}_t \vee l_{t+1}) & (@_t \vee \bar{l}_t \vee \mathrm{B2}_t \vee \mathrm{B4}_t \vee l_{t+1})
\end{array} \qquad (42)$$

If we now add the unit clauses $(\mathrm{A3}_r)$ and $(\mathrm{B3}_r)$, the resulting set of $13 + 50r$ clauses in $11+12r$ variables is readily satisfiable when $r = 6$, thereby proving that the critical room might indeed be jointly occupied. (Incidentally, standard terminology for mutual exclusion protocols would say that “two threads concurrently execute a *critical section*”; but we shall continue with our roommate metaphor.)

Back at the drawing board, one idea is to modify (40) by letting Alice use the room only when $l = 1$, but letting Bob in when $l = 0$:

A0. Maybe go to A1.	B0. Maybe go to B1.	
A1. If l go to A2, else to A1.	B1. If l go to B1, else to B2.	(43)
A2. Critical, go to A3.	B2. Critical, go to B3.	
A3. Set $l \leftarrow 0$, go to A0.	B3. Set $l \leftarrow 1$, go to B0.	

Computer tests with $r = 100$ show that the corresponding clauses are unsatisfiable; thus mutual exclusion is apparently guaranteed by (43).

But (43) is a nonstarter, because it imposes an intolerable cost: Alice can't use the room k times until Bob has already done so! Scrap that.

How about installing another light, so that each person controls one of them?

```
A0. Maybe go to A1.                      B0. Maybe go to B1.
A1. If $b$ go to A1, else to A2.         B1. If $a$ go to B1, else to B2.
A2. Set $a \leftarrow 1$, go to A3.      B2. Set $b \leftarrow 1$, go to B3.        (44)
A3. Critical, go to A4.                  B3. Critical, go to B4.
A4. Set $a \leftarrow 0$, go to A0.      B4. Set $b \leftarrow 0$, go to B0.
```

No; this suffers from the same defect as (40). But maybe we can cleverly switch the order of steps 1 and 2:

```
A0. Maybe go to A1.                      B0. Maybe go to B1.
A1. Set $a \leftarrow 1$, go to A2.      B1. Set $b \leftarrow 1$, go to B2.
A2. If $b$ go to A2, else to A3.         B2. If $a$ go to B2, else to B3.           (45)
A3. Critical, go to A4.                  B3. Critical, go to B4.
A4. Set $a \leftarrow 0$, go to A0.      B4. Set $b \leftarrow 0$, go to B0.
```

Yes! Exercise 95 proves easily that this protocol does achieve mutual exclusion.

Alas, however, a new problem now arises, namely the problem known as "deadlock" or "livelock." Alice and Bob can get into states A2 and B2, after which they're stuck — each waiting for the other to go critical.

In such cases they could agree to "reboot" somehow. But that would be a cop-out; they really seek a better solution. And they aren't alone: Many people have struggled with this surprisingly delicate problem over the years, and several solutions (both good and bad) appear in the exercises below. Edsger Dijkstra, in some pioneering lecture notes entitled *Cooperating Sequential Processes* [Technological University Eindhoven (September 1965), §2.1], thought of an instructive way to improve on (45):

```
A0. Maybe go to A1.                      B0. Maybe go to B1.
A1. Set $a \leftarrow 1$, go to A2.      B1. Set $b \leftarrow 1$, go to B2.
A2. If $b$ go to A3, else to A4.         B2. If $a$ go to B3, else to B4.
A3. Set $a \leftarrow 0$, go to A1.      B3. Set $b \leftarrow 0$, go to B1.        (46)
A4. Critical, go to A5.                  B4. Critical, go to B5.
A5. Set $a \leftarrow 0$, go to A0.      B5. Set $b \leftarrow 0$, go to B0.
```

But he realized that this too is unsatisfactory, because it permits scenarios in which Alice, say, might wait forever while Bob repeatedly uses the critical room. (Indeed, if Alice and Bob are in states A1 and B2, she might go to A2, A3, then A1, thereby letting him run to B4, B5, B0, B1, and B2; they're back where they started, yet she's made no progress.)

The existence of this problem, called *starvation*, can also be detected via bounded model checking. The basic idea (see exercise 91) is that starvation occurs if and only if there is a loop of transitions

$$X_0 \to X_1 \to \cdots \to X_p \to X_{p+1} \to \cdots \to X_r = X_p \tag{47}$$

such that (i) Alice and Bob each are bumped at least once during the loop; and (ii) at least one of them is never in a "maybe" or "critical" state during the loop.

And those conditions are easily encoded into clauses, because we can identify the variables for time r with the variables for time p, and we can append the clauses

$$(\overline{@_p} \vee \overline{@_{p+1}} \vee \cdots \vee \overline{@_{r-1}}) \;\wedge\; (@_p \vee @_{p+1} \vee \cdots \vee @_{r-1}) \qquad (48)$$

to guarantee (i). Condition (ii) is simply a matter of appending unit clauses; for example, to test whether Alice can be starved by (46), the relevant clauses are $\overline{\mathrm{A0}_p} \wedge \overline{\mathrm{A0}_{p+1}} \wedge \cdots \wedge \overline{\mathrm{A0}_{r-1}} \wedge \overline{\mathrm{A4}_p} \wedge \overline{\mathrm{A4}_{p+1}} \wedge \cdots \wedge \overline{\mathrm{A4}_{r-1}}$.

The deficiencies of (43), (45), and (46) can all be viewed as instances of starvation, because (47) and (48) are satisfiable (see exercise 90). Thus we can use bounded model checking to find counterexamples to *any* unsatisfactory protocol for mutual exclusion, either by exhibiting a scenario in which Alice and Bob are both in the critical room or by exhibiting a feasible starvation cycle (47).

Of course we'd like to go the other way, too: If a protocol has no counterexamples for, say, $r = 100$, we still might not know that it is really reliable; a counterexample might exist only when r is extremely large. Fortunately there are ways to obtain decent upper bounds on r, so that bounded model checking can be used to prove correctness as well as to demonstrate incorrectness. For example, we can verify the simplest known correct solution to Alice and Bob's problem, a protocol by G. L. Peterson [*Information Proc. Letters* **12** (1981), 115–116], who noticed that a careful combination of (43) and (45) actually suffices:

```
A0. Maybe go to A1.                  B0. Maybe go to B1.
A1. Set $a \leftarrow 1$, go to A2.          B1. Set $b \leftarrow 1$, go to B2.
A2. Set $l \leftarrow 0$, go to A3.          B2. Set $l \leftarrow 1$, go to B3.
A3. If $b$ go to A4, else to A5.      B3. If $a$ go to B4, else to B5.     (49)
A4. If $l$ go to A5, else to A3.      B4. If $l$ go to B3, else to B5.
A5. Critical, go to A6.              B5. Critical, go to B6.
A6. Set $a \leftarrow 0$, go to A0.          B6. Set $b \leftarrow 0$, go to B0.
```

Now there are *three* signal lights, a, b, and l—one controlled by Alice, one controlled by Bob, and one switchable by both.

To show that states A5 and B5 can't be concurrent, we can observe that the shortest counterexample will not repeat any state twice; in other words, it will be a *simple* path of transitions (33). Thus we can assume that r is at most the total number of states. However, (49) has $7 \times 7 \times 2 \times 2 \times 2 = 392$ states; that's a finite bound, not really out of reach for a good SAT solver on this particular problem, but we can do much better. For example, it's not hard to devise clauses that are satisfiable if and only if there's a simple path of length $\leq r$ (see exercise 92), and in this particular case the longest simple path turns out to have only 54 steps.

We can in fact do better yet by using the important notion of *invariants*, which we encountered in Section 1.2.1 and have seen repeatedly throughout this series of books. Invariant assertions are the key to most proofs of correctness, so it's not surprising that they also give a significant boost to bounded model checking. Formally speaking, if $\Phi(X)$ is a Boolean function of the state vector X, we say that Φ is invariant if $\Phi(X)$ implies $\Phi(X')$ whenever $X \to X'$. For example, it's not hard to see that the following clauses are invariant with respect to (49):

$$\begin{aligned}\Phi(X) \;=\; &(\mathrm{A0}\vee\mathrm{A1}\vee\mathrm{A2}\vee\mathrm{A3}\vee\mathrm{A4}\vee\mathrm{A5}\vee\mathrm{A6}) \wedge (\mathrm{B0}\vee\mathrm{B1}\vee\mathrm{B2}\vee\mathrm{B3}\vee\mathrm{B4}\vee\mathrm{B5}\vee\mathrm{B6})\\ &\wedge (\overline{\mathrm{A0}}\vee\bar a)\wedge(\overline{\mathrm{A1}}\vee\bar a)\wedge(\overline{\mathrm{A2}}\vee a)\wedge(\overline{\mathrm{A3}}\vee a)\wedge(\overline{\mathrm{A4}}\vee a)\wedge(\overline{\mathrm{A5}}\vee a)\wedge(\overline{\mathrm{A6}}\vee a)\\ &\wedge (\overline{\mathrm{B0}}\vee\bar b)\wedge(\overline{\mathrm{B1}}\vee\bar b)\wedge(\overline{\mathrm{B2}}\vee b)\wedge(\overline{\mathrm{B3}}\vee b)\wedge(\overline{\mathrm{B4}}\vee b)\wedge(\overline{\mathrm{B5}}\vee b)\wedge(\overline{\mathrm{B6}}\vee b).\end{aligned} \tag{50}$$

(The clause $\overline{\mathrm{A0}} \vee \bar a$ says that $a = 0$ when Alice is in state A0, etc.) And we can use a SAT solver to *prove* that Φ is invariant, by showing that the clauses

$$\Phi(X) \wedge (X \to X') \wedge \neg\Phi(X') \tag{51}$$

are *unsatisfiable*. Furthermore $\Phi(X_0)$ holds for the initial state X_0, because $\neg\Phi(X_0)$ is unsatisfiable. (See exercise 93.) Therefore $\Phi(X_t)$ is true for all $t \geq 0$, by induction, and we may add these helpful clauses to all of our formulas.

The invariant (50) reduces the total number of states by a factor of 4. And the real clincher is the fact that the clauses

$$(X_0 \to X_1 \to \cdots \to X_r) \wedge \Phi(X_0) \wedge \Phi(X_1) \wedge \cdots \wedge \Phi(X_r) \wedge \mathrm{A5}_r \wedge \mathrm{B5}_r, \tag{52}$$

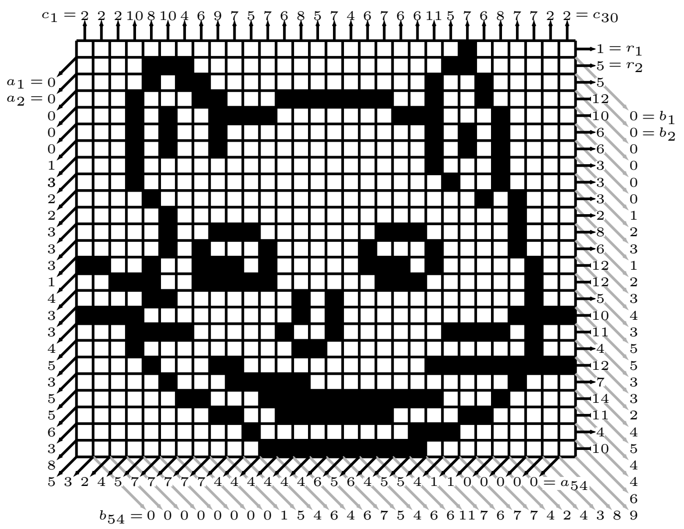
where X_0 is *not* required to be the initial state, turn out to be unsatisfiable when $r = 3$. In other words, there's no way to go back more than two steps from a bad state, without violating the invariant. We can conclude that mutual exclusion needs to be verified for (49) only by considering paths of length 2(!). Furthermore, similar ideas (exercise 98) show that (49) is starvation-free.

Caveat: Although (49) is a correct protocol for mutual exclusion according to Alice and Bob's ground rules, it *cannot* be used safely on most modern computers unless special care is taken to synchronize cache memories and write buffers. The reason is that hardware designers use all sorts of trickery to gain speed, and those tricks might allow one process to see $a = 0$ at time $t + 1$ even though another process has set $a \leftarrow 1$ at time t. We have developed the algorithms above by assuming a model of parallel computation that Leslie Lamport has called *sequential consistency* [*IEEE Trans.* **C-28** (1979), 690–691].

Digital tomography. Another set of appealing questions amenable to SAT solving comes from the study of binary images for which partial information is given. Consider, for example, Fig. 36, which shows the "Cheshire cat" of Section 7.1.3 in a new light. This image is an $m \times n$ array of Boolean variables $(x_{i,j})$, with $m = 25$ rows and $n = 30$ columns: The upper left corner element, $x_{1,1}$, is 0, representing white; and $x_{1,24} = 1$ corresponds to the lone black pixel in the top row. We are given the row sums $r_i = \sum_{j=1}^{n} x_{i,j}$ for $1 \leq i \leq m$ and the column sums $c_j = \sum_{i=1}^{m} x_{i,j}$ for $1 \leq j \leq n$, as well as both sets of sums in the 45° diagonal directions, namely

$$a_d = \sum_{i+j=d+1} x_{i,j} \quad\text{and}\quad b_d = \sum_{i-j=d-n} x_{i,j} \quad\text{for } 0 < d < m+n. \tag{53}$$

To what extent can such an image be reconstructed from its sums r_i, c_j, a_d, and b_d? Small examples are often uniquely determined by these Xray-like projections (see exercise 103). But the discrete nature of pixel images makes the reconstruction problem considerably more difficult than the corresponding

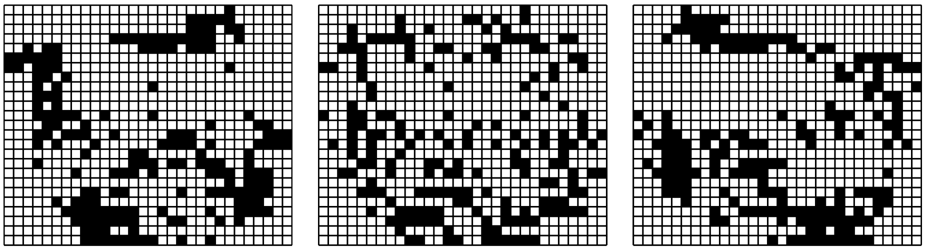


$c_1 = 2\ 2\ 2\ 10\ 8\ 10\ 4\ 6\ 9\ 7\ 5\ 7\ 6\ 8\ 5\ 7\ 4\ 6\ 7\ 6\ 6\ 11\ 5\ 7\ 6\ 8\ 7\ 7\ 2\ 2 = c_{30}$
$1 = r_1$
$5 = r_2$
5 12 10 6 6 3 3 3 2 8 6 12 12 5 10 11 4 12 7 14 11 4 10
$a_1 = 0$
$a_2 = 0$
0 0 0 1 3 2 2 3 3 3 1 4 3 3 4 5 3 5 5 6 3 8
5 3 2 4 5 7 7 7 7 7 4 4 4 4 4 6 5 6 4 5 5 4 1 1 0 0 0 0 0 $= a_{54}$
$0 = b_1$
$0 = b_2$
0 0 0 0 1 2 3 1 2 3 4 3 5 5 3 3 2 4 5 4 4 6 9
$b_{54} = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 5\ 4\ 6\ 4\ 6\ 7\ 5\ 4\ 6\ 6\ 11\ 7\ 6\ 7\ 7\ 4\ 2\ 4\ 3\ 8\ 9$

Fig. 36. An array of black and white pixels together with its row sums r_i, column sums c_j, and diagonal sums a_d, b_d.

continuous problem, in which projections from many different angles are available. Notice, for example, that the classical "8 queens problem" — to place eight nonattacking queens on a chessboard — is equivalent to solving an 8×8 digital tomography problem with the constraints $r_i = 1$, $c_j = 1$, $a_d \le 1$, and $b_d \le 1$.

The constraints of Fig. 36 appear to be quite strict, so we might expect that most of the pixels $x_{i,j}$ are determined uniquely by the given sums. For instance, the fact that $a_1 = \cdots = a_5 = 0$ tells us that $x_{i,j} = 0$ whenever $i + j \le 6$; and similar deductions are possible at all four corners of the image. A crude "ballpark estimate" suggests that we're given a few more than 150 sums, most of which occupy 5 bits each; hence we have roughly $150 \times 5 = 750$ bits of data, from which we wish to reconstruct $25 \times 30 = 750$ pixels $x_{i,j}$. Actually, however, this problem turns out to have many billions of solutions (see Fig. 37), most of which aren't catlike! Exercise 106 provides a less crude estimate, which shows that this abundance of solutions isn't really surprising.



(a) lexicographically first; (b) maximally different; (c) lexicographically last.

Fig. 37. Extreme solutions to the constraints of Fig. 36.

A digital tomography problem such as Fig. 36 is readily represented as a sequence of clauses to be satisfied, because each of the individual requirements is just a special case of the cardinality constraints that we've already considered in the clauses of (18)–(21). This problem differs from the other instances of SAT that we've been discussing, primarily because it consists *entirely* of cardinality constraints: It is a question of solving $25 + 30 + 54 + 54 = 163$ simultaneous linear equations in 750 variables $x_{i,j}$, where each variable must be either 0 or 1. So it's essentially an instance of *integer programming* (IP), not an instance of satisfiability (SAT). On the other hand, Bailleux and Boufkhad devised clauses (20) and (21) precisely because they wanted to apply SAT solvers, not IP solvers, to digital tomography. In the case of Fig. 36, their method yields approximately 40,000 clauses in 9,000 variables, containing about 100,000 literals altogether.

Figure 37(b) illustrates a solution that differs as much as possible from Fig. 36. Thus it minimizes the sum $x_{1,24} + x_{2,5} + x_{2,6} + \cdots + x_{25,21}$ of the 182 variables that correspond to black pixels, over all 0-or-1-valued solutions to the linear equations. If we use linear programming to minimize that sum over $0 \le x_{i,j} \le 1$, *without* requiring the variables to be integers, we find almost instantly that the minimum value is ≈ 31.38 under these relaxed conditions; hence every black-and-white image must have at least 32 black pixels in common with Fig. 36. Furthermore, Fig. 37(b) — which can be computed in a few seconds by widely available IP solvers such as CPLEX — actually achieves this minimum. By contrast, state-of-the-art SAT solvers as of 2013 had great difficulty finding such an image, even when told that a 32-in-common solution is possible.

Parts (a) and (c) of Fig. 37 are, similarly, quite relevant to the current state of the SAT-solving art: They represent hundreds of individual SAT instances, where the first k variables are set to particular known values and we try to find a solution with the next variable either 0 or 1, respectively. Several of the subproblems that arose while computing rows 6 and 7 of Fig. 37(c) turned out to be quite challenging, although resolvable in a few hours; and similar problems, which correspond to different kinds of lexicographic order, apparently still lie beyond the reach of contemporary SAT-oriented methods. Yet IP solvers polish these problems off with ease. (See exercises 109 and 111.)

If we provide more information about an image, our chances of being able to reconstruct it uniquely are naturally enhanced. For example, suppose we also compute the numbers r'_i, c'_j, a'_d, and b'_d, which count the *runs of 1s* that occur in each row, column, and diagonal. (We have $r'_1 = 1$, $r'_2 = 2$, $r'_3 = 4$, and so on.) Given this additional data, we can show that Fig. 36 is the only solution, because a suitable set of clauses turns out to be unsatisfiable. Exercise 117 explains one way by which (20) and (21) can be modified so that they provide constraints based on the run counts. Furthermore, it isn't difficult to express even more detailed constraints, such as the assertion that "column 4 contains runs of respective lengths $(6, 1, 3)$," as a sequence of clauses; see exercise 438.

SAT examples — summary. We've now seen convincing evidence that simple Boolean clauses — ANDs of ORs of literals — are enormously versatile. Among

INDEX AND GLOSSARY

The republic of letters is at present divided into three classes.
One writer, for instance, excels at a plan or a title-page,
another works away the body of the book,
and a third is a dab at an index.
— OLIVER GOLDSMITH, in *The Bee* (1759)

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.

∂S (boundary set), 58, 154, 180, 188.
0–1 matrices, 106–109, 151, 176–177, 181, *see also* Grid patterns.
1SAT, 49, 148.
2-colorability of hypergraphs, 185.
2SAT, 49, 51–54, 77–78, 80, 101, 144, 147, 149, 157, 159, 266.
3-regular graphs, 147, 154, 231.
3CNF, 3, 148.
3D MATCHING problem, 134, 225, 290–291.
3D visualizations, 116–118.
3SAT, 3–4, 47–51, 59, 60, 78–80, 93–94, 131, 135, 146, 148–151, 153, 182–184, 231.
4-cycles, 109–110, 178, 225, 274.
4-regular graphs, 290.
4SAT, 49, 51, 150, 290.
5SAT, 51, 58, 224, 291.
6SAT, 51.
7SAT, 51, 151.
8 queens problem, 25, 282.
90°-rotational symmetry, 138, 202, 275.
100 test cases, 113–124, 127, 182, 184.
$\emptyset$ (the empty set), 185.
ϵ (the empty clause), 3, 27, 185, 291.
ϵ (the empty string), 3, 85.
ϵ (the tolerance for convergence), 93–94.
ε (offset in heuristic scores), 126, 213.
νx (1s count), *see* Sideways sum.
π (circle ratio), *see* Pi.
ρ (damping factor for variable activity), 67, 125–127, 155, 286.
ρ (damping factor for reinforcement), 93–94.
ϱ (damping factor for clause activity), 74, 125–127, 286.
τ parameter, 125–127, 235, 286.
$\tau(a,b)$ function, 147.
ϕ (golden ratio), 146, 147, 160, 251.
ψ (agility threshold), 76–77, 124–127, 240, 286.
ψ (confidence level), 93, 255.

a.s.: almost surely, 149, 153.
AAAI: American Association for Artificial Intelligence (founded in 1979); Association for the Advancement of Artificial Intelligence (since 2007), 67.
Absorbed clauses, 168.
Accordion solitaire, 282.
Achlioptas, Dimitris (Αχλιόπτας, Δημήτρης), 221.
`ACT(c)`, 74, 125.
`ACT(k)`, 66–68, 75, 125, 132.
Active path, 13.
Active ring, 32.
Activity scores, 67, 74–76, 125, 132, 155, 239.
Acyclic orientation, 161.
Adams, Douglas Noel (42), 126.
Adaptive control, 46, 126.
Addition, encoding of, 100–101, 114; *see also* Full adders, Half adders.
Adjacency matrix, 281.
Adjacent pairs of letters, avoiding, 248.
`AGILITY`, 76, 158, 240.
Agility level, 76, 124, 158.
Agility threshold (ψ), 76–77, 124–127, 240, 286.
Ahmed, Tanbir (তানবীর আহমেদ), 5, 147.
Alava, Mikko Juhani, 80.
Aldous, David John, 219.
Algorithm L^0, 39, 147.
Alice, 20–24, 139–141.
All-different constraint, 171.
All solutions, 143, 266.
Alon, Noga (נוגה אלון), 174, 254, 260.
Aloul, Fadi Ahmed (فادي أحمد العالول), 112, 281, 284.
Analysis of algorithms, 146–152, 158–160, 164.
Ancestors, 43.
AND operation, 9, 10, 13.
  bitwise (x & y), 28, 29, 31, 37, 38, 66, 68, 76, 81, 196, 209–211, 220, 241.
André, Pascal, 131.
Anisimov, Anatoly Vasilievich (Анисимов, Анатолий Васильевич), 249.
Annexstein, Fred Saul, 273.
Anti-maximal-element clauses, 56, 62, 97, 115, 153, 155, 157, 167.

Antisymmetry, 178.
Appier dit Hanzelet, Jean, 57.
April Fool, 7.
Ardila Mantilla, Federico, 256.
Arithmetic progressions, 4, 114.
 avoiding, 135.
Armies of queens, 180.
Asín Achá, Roberto Javier, 267.
Asserting clause, *see* Forcing clause.
Associative block design, 4.
Associative law, 227.
Asymmetric Boolean functions, 178.
Asymmetric elimination, 260.
Asymmetric tautology, *see* Certifiable clauses.
Asymptotic methods, 53–54, 147–151, 164, 210, 226, 230, 283.
At-least-one constraint, 171, 265.
At-most-one constraint, 6, 97–99, 103, 104, 120, 134, 149, 170, 171, 238, 265, 266, 289.
ATPG: Automatic test pattern generation, *see* Fault testing.
Atserias, Albert Perí, 262.
Audemard, Gilles, 72.
Aurifeuille, Léon François Antoine, factors, 14.
Autarkies, 44, 71, 146, 152, 177, 214, 215, 217.
 testing for, 146, 214.
Autarky principle, 44.
Automatic test pattern generation, *see* Fault testing.
Automaton, 272.
Automorphisms, 108, 111, 180, 197, 236, 277.
Autosifting, 220.
Auxiliary variables, 6, 8, 15, 17, 60, 97, 101, 104, 105, 109, 135, 136, 148, 170–174, 186, 262, 268, 276–279, 280–281, 287.
`AVAIL` stack, 257.
Averages, 120.
Avoiding submatrices, 106–107.
Awkward trees, 227.
Axiom clauses, 54, 59, 100, 264, 266.

Bacchus, Fahiem, 73, 271.
Backjumping, 64, 68, 74, 132, 233, 236, 239.
Backtrack trees, *see* Search trees.
Backtracking, 4, 27–34, 38–39, 64, 105, 128, 129, 132, 151, 176, 190, 204, 219, 231, 236.
Bailleux, Olivier, 8, 26, 135, 137, 143, 174, 272.
Baker, Andrew Baer, 98.
Balas, Egon, 206.
Baldassi, Carlo, 93.
Ball, Walter William Rouse, 180.
Ballot numbers, 78.
Balls and urns, 221.
Banbara, Mutsunori (番原睦則), 264, 267, 268.
Bartley, William Warren, III, 129.
Basket weavers, 141.
Batcher, Kenneth Edward, 266.
Baumert, Leonard Daniel, 265.
Bayardo, Roberto Xavier, Jr., 132.
Bayes, Thomas, networks, 95.
BCP: Boolean constraint propagation, *see* Unit propagation.
BDD: A reduced, ordered binary decision diagram, 17–18, 102, 103, 132, 137, 148, 174, 181, 188, 193, 194, 197, 202, 220.
BDD base, 219.
Belief propagation, 95.
Ben-Dor, Amir (אמיר בן-דור), 289.
Ben-Sasson, Eli (אלי בן-ששון), 57–58, 153, 231.
Benchmark tests, 35, 131–133, 139, 147, 190, 206.
 100 test cases, iv, 113–124, 127, 182, 184.
Bender, Edward Anton, 250.
Beresin, May, 275.
Berghammer, Rudolf, 204.
BerkMin solver, 132.
Berlekamp, Elwyn Ralph, 17.
Berman, Piotr, 224.
Bernhart, Frank Reiff, 188.
Bernoulli, Jacques (= Jakob = James), distribution, multivariate, 89.
Bethe, Hans Albrecht, 95.
Better reasons, 157.
Bias messages, 92.
Biased random bits, 12, 241.
Biere, Armin, v, 66, 76, 96, 129, 132, 166, 188, 258, 260, 261, 269, 280.
Big clauses, 145.
`BIMP` tables, 36–41, 43, 45, 124, 144, 235.
Binary addition, 114.
Binary clauses, 3, 6, 36, 124, 133, 155–156.
Binary constraints, 171.
Binary decoder, 179.
Binary implication graph, *see* Dependency digraph, 41.
Binary matrices, 106–109, 151, 176–177, 181, *see also* Grid patterns.
Binary multiplication, 8.
Binary number system, 9, 98.
Binary recurrence relations, 189.
Binary relations, 56.
Binary search, 187.
Binary strings, 181.
Binary tensor contingency problem, 142, 151.
Binomial coefficients, 149.
Binomial convolutions, 250.
Bipartite graphs, 58, 177, 290.
Bipartite matching, 150.
Bipartite structure, 90.

Birthday paradox, 49.
Bishops, 141.
Bitmaps, 17, 139.
Bitwise operations, 11, 12, 81, 158, 161, 241, 246, 258–259.
Black and blue principle, 146, 216.
Black and white principle, 146.
Blake, Archie, 130.
blit, 234, 236.
Block decomposition, 275.
Block designs, 106.
Block diagonal matrices, 177.
Blocked clauses, 102, 215, 260, 261, 269.
  binary, 146.
  elimination of, 167.
Blocked self-subsumption, 167.
Blocking digraph, 215.
Blocks in Life, 197, 200.
Bloom, Burton Howard, coding, 258.
Bloom, Thomas Frederick, 185.
Bob, 20–24, 115, 139–141.
Böhm, Max Joachim, 131.
Bollobás, Béla, 54, 220.
Bonacina, Maria Paola, 129.
book graphs, 126, 179.
Boole, George, 129.
Boolean chains, 9, 11, 12, 102, 114, 173.
  optimum, 178–179.
Boolean formulas, 1.
Boolean functions, 14–16.
  expressible in kCNF, 220.
  synthesis of, 178–179.
Boppana, Ravi Babu, 174.
Borgs, Christian, 54.
Bottom-up algorithms, 252.
Boufkhad, Yacine (ياسين بوفخد), 8, 26, 131, 135, 137, 143, 174, 272.
Boundary sets, 58, 154, 180, 188.
Boundary variables, 230.
Bounded model checking, 16–24, 132, 137–141, 157, 179–180.
Branching heuristics, 105, 144, *see also* Decision literals.
Branching programs, 102, 173, 174.
Branchless computation, 242.
Braunstein, Alfredo, 90, 91, 256.
Breadth-first search, 37, 43, 68, 130, 235.
Break count, 79.
Breaking symmetries, vii, 5, 19, 105–114, 138, 176–181, 187, 188, 190–192, 238, 267, 281–283, 285, 288–290.
  in graph coloring, 99–100, 114, 171, 179, 187.
Broadcasting, 170.
Broadword computations, 11, 12, 158, 161, 246, 258.
Brown, Cynthia Ann Blocher, 30, 32, 131, 151, 226.
Brown, Thomas Craig, 185.
Brummayer, Robert Daniel, 269.
Brunetti, Sara, 206.
Bryant, Randal Everitt, v, 7, 187.
BST(l), 211.
BSTAMP counter, 211.
Buckingham, David John, 197, 200.
Buddy system, 36, 144, 235.
Bugrara, Khaled Mohamed (خالد محمد ابوغراره), 226.
Bugs, 16, 69, 77, 133, 240.
Bulnes-Rozas, Juan Bautista, 215.
Bumped processes, 21–22, 140, 202.
Bundala, Daniel, 196.
Burney, Charles, viii.
Burns, James Edward, 204.
Buro, Michael, 131.
Buss, Samuel Rudolph, v, 153, 270.
Bystanders, *see* Easy clauses.

C-SAT solver, 131.
Cache memories, 24.
Calabro, Chris, 288.
Candidate variables, 40–44, 131, 214.
Canonical forms, 138, 248.
Cardinality constraints, 7–8, 26, 104, 106, 113, 114, 121, 135, 143, 187, 188, 193, 194, 196, 204, 285.
  for intervals, 100, 190, 280.
Carlier, Jacques, 131.
Carlitz, Leonard, 162.
Carriers in Life, 197, 200.
Carroll, Lewis (= Dodgson, Charles Lutwidge), 129–130.
Carry bits, 9, 12, 101, 192, 193.
Cartier, Pierre Emile, 83, 86, 163.
Case analysis, 27, 130.
CDCL (conflict driven clause learning) solvers, 62–71, 103, 121, 132–133, 155.
  combined with lookahead solvers, 129.
  compared to lookahead solvers, 98–100, 118–121, 182, 290.
Cells of memory, 28, 122–124.
Cellular automata, 17, 202.
Certifiable clauses, 168, 260.
Certificates of unsatisfiability, 69–71, 157, 169, 176, 178.
Chaff solver, 67, 132.
Chain rule for conditional probability, 254.
Chains, 276, *see also* Boolean chains, Resolution chains, s-chains.
Channel assignment, 136.
Channeling clauses, 264.
Characteristic polynomial of a matrix, 163, 218.
Chavas, Joël, 91.
Chayes, Jennifer Tour, 54.

Chebyshev (= Tschebyscheff), Pafnutii Lvovich (Чебышевъ, Пафнутій Львовичъ = Чебышев, Пафнутий Львович), inequality, 221.
  polynomials, 247.
Cheshire Tom, 24–26, 115, 142–143.
Chess, 7, 170.
Chessboards, 18, 25, 99, 106, 115, 138, 180.
Chiral symmetry (rotation but not reflection), 138, 202, 275.
Chordal graphs, 163–164.
Chromatic number $\chi(G)$, 99, 135–136, 147, 174, 281.
Chung Graham, Fan Rong King (鍾金芳蓉), 283.
Chvátal, Václav (= Vašek), 5, 52, 59, 185.
Cimatti, Alessandro, 132.
Circuits, Boolean, 10, 101–103, 114, *see also* Boolean chains.
Circular lists, 32.
Clarke, Edmund Melson, Jr., 132.
Clashing pairs of letters, 84.
Clausal proofs, *see* Certificates of unsatisfiability.
Clause activity scores, 74, 239.
Clause-learning algorithms, 61–62, 103, 118, 121, 132–133, 154–155.
Clauses per literal, 150, 231; *see also* Density of clauses.
Claw graph, 249.
Clichés, 76.
Clique Local Lemma, 165.
Cliques, 81, 100, 162, 167, 169, 171, 179, 190.
  covering by, 165.
Closest strings, 114, 181, 182.
Clusters, 166.
CNF: Conjunctive normal form, 9, 101, 154, 173, 193, 196.
Cocomparability graphs, 249, 250.
Coe, Timothy Vance, 201.
Coexisting armies of queens, 180.
Cographs, 163, 250.
Cohen, Bram, 79, 246.
Coja-Oghlan, Amin, 221.
Colexicographic order, 206, 278.
Coloring a graph, 6–7, 99–100, 153, 179, 260.
  fractional, 135.
  multiple, 135.
  of queens, 99–100, 114–115, 171.
  radio, 136.
Column sums, 151.
Commutative law, 27, 180, 227.
  partial, 83, 250–251.
Comparator modules, 115, 137.
Comparison, lexicographic, 101, 111–113.
Comparison of running times, 34–35, 39, 69, 97–100, 105–107, 110, 112, 118–128, 182, 184, 218, 237, 264, 281, 290.
Compensation resolvents, 39, 144, 147.
Competitions, 131–133, 291.
Complement of a graph, 134.
Complementation of unary representations, 100.
Complemented literals, 2–4, 37, 62–64, 78, 111, 210, 266.
Complete binary trees, 8, 135, 230.
Complete bipartite graphs $K_{m,n}$, 250, 254.
Complete graphs K_n, 153, 178, 186, 262.
Complete k-partite graphs, 250, 262.
Complete t-ary trees, 160.
Compressing, *see* Purging unhelpful clauses.
Conditional autarkies, 215.
Conditional expectation inequality, 150.
Conditional symmetries, 107, *see* Endomorphisms.
Conditioning operations ($F\,|\,l$ and $F\,|\,L$), 27, 96, 143, 157, *see* Unit conditioning.
Cones in trace theory, 87.
Confidence level (ψ), 93, 255.
Conflict clauses, 63, 70, 171; *see also* Preclusion clauses.
Conflict driven clause learning, 62–69, 103, 121, 132–133, 155.
Conflicts, 62, 124, 132.
Conjunctive normal form, 1, 9, 101, 154, 173, 193, 196.
  irredundant, 257.
Conjunctive prime form, 104.
Connected graphs, 177.
Connectedness testing, 169–170.
Connection puzzles, 170.
CoNP-complete problems, 3, 207.
Consecutive 1s, 88, 175, 254.
Consensus of implicants, 130.
Consistent Boolean formulas, *see* Satisfiable formulas.
Consistent partial assignments, 30, 165.
Constrained variables in partial assignments, 165–166.
Contests, 131–132.
Context free languages, 175.
Contiguous United States, 136.
Contingency tables, binary, 142.
  3D, 151.
Convex functions, 216.
Convex hulls, 247.
Convolution principle, 250.
Conway, John Horton, 17, 139, 201.
Cook, Stephen Arthur, 61, 62, 130–131, 154, 229, 237.
cook clauses, 157.
Cooper, Alec Steven, 285.
Core assignments, 166.
Core of Horn clauses, 174, 216.
Coupon collector's test, 220.
Covering assignments, 166, 221, 255.
Covering problems, 2, 193, 194, *see also* Domino coverings.

Covering strings, 181.
CPLEX system, 26, 289.
CPU: Central Processing Unit (one computer thread), 121.
Crawford, James Melton, Jr., 98, 113.
Cray 2 computer, 137.
Critical sections, 21–23, 140–141.
Crossover point, *see* Threshold of satisfiability.
Crusoe (= Kreutznaer), Robinson, vii.
CSP: The constraint satisfaction problem, 283.
Cube and conquer method, 129.
Cubic graphs (3-regular, trivalent), 147, 154, 231.
Cufflink pattern, 255.
Culver, Clayton Lee, 185.
Cut rule, 59.
Cutoff parameters, 41, 145.
Cutting planes, 184, 206.
Cycle detection problem, 260.
Cycle graphs C_n, 135, 160, 262.
Cycle structure of a permutation, 108, 112–113, 163, 178, 277.
Cyclic DPLL algorithm, 33.
Cyclic patterns, 19.
Cyclic permutations, 163.

da Vinci, Leonardo di ser Piero, 7.
Dadda, Luigi, 9, 114, 136, 173.
Dags: Directed acyclic graphs, 54.
 of resolutions, 54–56, 70.
Damping factors, 46, 67, 74, 76, 93–94, 125, 126, 155.
Dancing links, 5, 121, 134, 208, 288, 291.
Dantchev, Stefan Stoyanov (Данчев, Стефан Стоянов), 110.
Darwiche, Adnan Youssef (عدنان يوسف درويش), 67, 262.
Data structures, 28–34, 36–38, 43, 66–67, 80, 95–96, 143–145, 155–156, 159, 167, 238, 273.
Davis, Martin David, 9, 31–32, 130, 298.
Dawson, Thomas Rayner, 170.
De Morgan, Augustus, laws, 3.
de Vries, Sven, 206.
de Wilde, Boris, 213.
Deadlock, 22–23.
Debugging, 69, 77.
Dechter, Rina Kahana (רינה כהנא דכטר), 67.
Decision literals, 62, 69, 124, 132.
Decision trees, *see* Search trees.
Decomposable matrices, 177.
Default parameters, 93, 125–126.
Default values of gates, 11.
Definite Horn clauses, 174.
Defoe, Daniel (= Daniel Foe), vii.
Degree of a vertex, 191.
Degrees of truth, 37–39, 42–43, 45–46, 216.
Dekker, Theodorus Jozef, 140.
Del Lungo, Alberto, 206.
Delayer, 55–56, 152–153.
Deletion from a heap, 234.
Delta sequence, 290.
Demenkov, Evgeny Alexandrovich (Деменков, Евгений Александрович), 280.
Density of clauses: The number of clauses per variable, 50–51, 150, 231, 288.
Dependence graph in trace theory, 248.
Dependence of literals, 63.
Dependency digraph (of literals), 41, 131, 168, 215, 237, 260.
Dependency-directed backtracking, *see* Backjumping.
Dependency graph (of events), 82, 164, 165.
Dependency on a variable, 137.
Depth-first search, 130.
Dequen, Gilles Maurice Marceau, 131.
Determinants, 162, 163, 251.
Deterministic algorithm, 17, 120.
Deventer, Mattijs Oskar van, 290.
`DFAIL`, 46, 147.
Dfalse literals, 45.
Diagonals of a matrix, 24–25, 141–142.
Diagram of a trace, 84.
Díaz Cort, José Maria (= Josep), 51.
Dick, William Brisbane, 180.
Difficult instances of SAT, 5, 14, 26, 51, 55–59, 118–121, 153–154, 184, 190, 192, 197, 206, 280.
Digital tomography, 24–26, 115, 141–143, 167, 285.
Digraphs, 54, 108, 161, 162, 263, *see also* Blocking digraph, Dependency digraph, Implication digraph.
Dijkstra, Edsger Wybe, 22, 202, 204.
DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, 131.
DIMACS: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, inaugurated in 1990.
Ding, Jian (丁剑), 51.
Direct encoding, 98, 114, 171, 186, 264, 265, 281.
Direct sum of graphs or matrices, 162, 177.
Directed acyclic graphs of resolutions, 54–56, 70.
Directed graphs, *see* Digraphs.
Discarding the previous learned clause, 72, 156.
Discrepancy patterns, 114, 182.
Disjoint shortest paths, 276.
Disjunctive normal forms, 14, 115, 130, 195, 257.
Distance $d(u, v)$ in a graph, 262.
Distinct literals, 2.
Division of traces, 85, 161, 250.

DNF: Disjunctive normal form, 14, 115, 130, 195, 257.
Dodgson, Charles Lutwidge, 129–130.
Domino coverings, 110, 114, 115, 143, 177, 178.
Don't-cares, 194, 280.
Double clique hints, 100, 114, 171.
Double coloring, 115, 135.
Double lookahead, 45–46, 126, 131, 286.
Double order, 214.
Double truth, 45.
Doubly linked lists, 28, 257, 259.
Downhill resolution, 96, 166.
Downhill transformations, 95.
Doyle, Arthur Ignatius Conan, 72.
DPLL (Davis, Putnam, Logemann, Loveland) algorithm, 32–33, 62.
  with lookahead, 38, 131.
DT (double truth), 45.
Dtrue literals, 45.
Dual of a Boolean function, 130, 174.
Dubois, Olivier, 131.
Dudeney, Henry Ernest, 114, 263.
Dufour, Mark, 37.
Dull, Brutus Cyclops, 181.
Durfee, William Pitt, square, 276.
Dynamic storage allocation, 144.
Dynamical system, discrete, 16.

e, as source of "random" data, 12, 193.
Eager data structures, 30, 36, 156.
Easy clauses, 149.
Eaters in Life, 20, 139.
Eén, Niklas Göran, v, 67, 96, 166, 203, 260, 268.
Ehlers, Thorsten, 196.
Eightfold symmetry, 138, 198.
Elegance, 35–36, 196.
Elimination of clauses, 167–168; *see also* Purging unhelpful clauses.
Elimination of variables, 60–61, 95–97, 101, 102, 129, 130, 154–155, 166–168, 173, 174, 256–257, 259–260, 270, 271.
Embedded graphs, 169, 262.
Empilements, 84, 161, 248.
Empirical performance measurements, 122–124.
Empty clause (ϵ), 3, 27, 185.
Empty list, representation of, 33, 210.
Empty partial assignment, 166.
Empty set ($\emptyset$), 185.
Empty string (ϵ), 3, 85.
Encoding into clauses, 6, 18, 97–105, 120, 134, 170, 179, 198, 202.
  ternary data, 100, 141, 179.
Endomorphisms, 107–111, 177–178, 181, 281, 290.
Equal sums, encoding of, 174.
Equally spaced 1s, 4, 114, 135; *see also waerden*.
Equivalence classes in trace theory, 84.
Equivalence of Boolean functions, 178.
Erdős, Pál (= Paul), 81, 107, 190, 281.
  discrepancy patterns, 114, 179, 182.
Erp rules, 95–96, 166–168, 259.
Evaluation of Boolean functions, 137, 178–179, 194.
Even-length cycles, 277.
Even-odd endomorphisms, 110, 177–178.
Exact cover problems, vii, 2, 5–6, 28, 134, 183, 186, 219, 225, 257, 291.
  by pairs (perfect matchings), 109–110, *see also* Domino coverings.
  by triples (3D MATCHING), 134, 225, 290–291.
  fractional, 135–136.
Exclusion clauses, 6, 21, 99, 114, 134, 149, 153, 238, 260, 289.
Exclusive or, ternary, 136.
Existential quantifiers, 60.
Expander graphs, 58, 231.
Exploitation stack, 259.
Exploration phase of lookahead, 40, 43–44.
Exponential time, 144.
  hypothesis, 288.
Extended resolution, 60, 71, 133, 154, 168, 215.
Extreme distribution, 87, 89, 163.

factor_fifo(m,n,z), 10, 12, 114, 184, 192.
factor_lifo(m,n,z), 10, 114, 184, 192.
factor_rand(m,n,z,s), 10, 184.
Factorization, 8–10, 136, 184, 192.
  of traces, 86, 162, 250.
Failed literals, 97, 167, 175, 269.
Fallacious reasoning, 16, 284.
False hits, 258.
False literals preferred, 31, 33, 67, 125–127, 286.
Fanout gates, 10–14, 136.
Fat clauses, 58.
Fault testing, 10–14, 114, 126, 136–137, 167, 260.
Feedback mechanism, 46, 104.
Fermat, Pierre de, 10.
Fernandez de la Vega, Wenceslas, 52.
Fibonacci, Leonardo, of Pisa (= Leonardo filio Bonacii Pisano), numbers, 160, 215, 254.
  ruler function, 246.
Fichte, Johannes Klaus, 262.
Field of a variable, 91, 165.
FIFO: first in, first out, 10.
Finite-state automata, 175.
First in, first out, 10.
First moment principle, 53, 148, 150.
First order logic, 59, 130.
Fischetti, Matteo, 206.
Fixed point of endomorphisms, 177.

Fixed values of literals, 37, 42–46.
FKG inequality, 89.
Flag bits, 235.
Flammenkamp, Achim, 198.
Flexibility coefficients, 91.
Flickering state variables, 141.
Flipflops in Life, 138, 143.
Floating point arithmetic, 91–92, 217, 239.
 overflow, 67.
Floor tiling, 115, 143, 199.
Flower snarks, 69, 147, 153, 157.
Flushing literals and restarting, 68, 75–77, 124, 132, 157, 158, 169, 234, 246.
Foata, Dominique Cyprien, 83, 86, 163.
Focus of attention, 41, 67, 91, 132.
Foe, Daniel (= Daniel Defoe), vii.
Footprint heuristic, 279.
Forced literals, 45.
Forcing clause, 62, *see* Unit propagation.
Forcing representations, 104–105, 174, 175, 274.
Forests, 43, 87, 163.
Forgetting clauses, 168, 169, *see* Purging unhelpful clauses.
Four bit protocol, 115.
Four Color Theorem, 7.
Fourfold symmetry, 138.
Fractional coloring number, 135.
Fractional exact cover, 135–136.
Frances, Moti (מודי פרנסס), 285.
Franco, John Vincent, 131, 148, 225, 273.
Free literals and free variables, 38, 66, 165–166.
Freeman, Jon William, 131.
Friedgut, Ehud (אהוד פרידגוט), 51.
Frontier ($\partial_{\rm in}$), 180, 188.
Frost, Daniel Hunter, 67.
fsnark clauses, 69, 71, 114, 147–148, 153, 157.
Full adders, 9, 136, 179, 192, 268, 278.
 modified, 114, 280.
Full runs, 73, 158, 235.
Furtlehner, Cyril, 91.

Gμ: Gigamems = billions of memory accesses, 97, 118.
 per minute, 289.
Gadgets, 134, 183.
Gallager, Robert Gray, 95.
Game of Life, 17–20, 97, 114, 137–139, 143, 167.
Garden of Eden, 139.
Gardner, Martin, 7, 19, 181, 188, 283.
Gates, 10–13, 101–103, 121, 136.
GB_GATES, 13–14.
Gebauer, Heidi Maria, 224.
Geek art, 116–117.
Generalization of resolution, 226.
Generating functions, 85, 89–90, 151, 158–159, 164, 188, 194, 219, 222, 230, 284.
 exponential, 162.
Generic graph, 174.
Gent, Ian Philip, v, 265, 283.
Gentzen, Gerhard Karl Erich, 59.
Geometric distribution, 244, 262.
Georges, John Pericles, 192.
Gerdes, Paulus Pierre Joseph, 205.
Gessel, Ira Martin, 248.
Giant strong component, 52.
Gigamem (Gμ): One billion memory accesses, 35, 39, 97, 188.
 per minute, 289.
Ginsberg, Matthew Leigh, 113.
Gipatsi patterns, 141.
Girth of a graph, 176, 290.
Given literals ($F\,|\,l$ or $F\,|\,L$), 27, 96, 143, 157; *see also* Unit conditioning.
Gliders in Life, 19, 138–139, 197, 201.
 symmetry of, 200.
Global ordering, 284.
Glucose measure, *see* Literal block distance.
Goerdt, Andreas, 52.
Goldberg, Allen Terry, 131, 225.
Goldberg, Eugene Isaacovich (Гольдберг, Евгений Исаакович), 70, 132.
Goldman, Jay Robert, 250.
Goldsmith, Oliver, 293.
Golomb, Solomon Wolf, 265.
González-Arce, Teófilo Francisco, 268.
Gosper, Ralph William, Jr., 20, 198.
Goultiaeva, Alexandra Borisovna (Гультяева, Александра Борисовна), 73.
Grabarchuk, Peter Serhiyevich (Грабарчук, Петро Сергійович), 263.
Grabarchuk, Serhiy Alexeevich (Грабарчук, Сергій Олексійович), 263.
Grabarchuk, Serhiy Serhiyevich (Грабарчук, Сергій Сергійович), 263.
Graham, Ronald Lewis (葛立恒), 185, 283.
Graph-based axioms, 59, 154, 178, 290.
Graph coloring problems, *see* Coloring a graph.
Graph embedding, 169, 262.
Graph layout, 116–118.
Graph quenching, 114, 179–180, 281.
Gray, Frank, codes, 201, 282.
Greedy algorithms, 80, 136, 172.
Greenbaum, Steven Fine, 102.
Greene, Curtis, 276.
Grid graphs, 110, 136, 151, 162–163.
 list coloring of, 151.
Grid patterns, 17–20, 24–26, 137–139, 142.
 rotated 45°, 141–142.
Griggs, Jerrold Robinson, 192.
Grinbergs, Emanuels Donats Frīdrihs Jānis, 218.

Gritzman, Peter, 206.
Grötschel, Martin, 264.
Gu, Jun (顾钧), 77.
Guéret-Jussien, Christelle, 267.
Guilherme De Carvalho Resende, Mauricio, 16.
GUROBI system, 289.
Guy, Richard Kenneth, 17, 19, 107.
Gwynne, Matthew Simon, 105, 270, 273.

Hackers, 199.
Haken, Armin, 57, 58.
Halevi, Shai (שי הלוי), 289.
Half adders, 9, 192, 268.
Halting problem, 130.
Hamadi, Youssef (ⵃ·ⵎ·ⴷⵉ ⵢⵓⵙ·ⴼ, حمادي يوسف), 236.
Hamilton, William Rowan, cycles, 169.
  paths, 184.
Hamming, Richard Wesley, distance, 285.
Han, Hyojung (한효정), 236.
Handwaving, 89.
Hanzelet, *see* Appier dit Hanzelet.
Hard clauses, 168.
Hard sudoku, 183.
Hartman, Christiaan, 202.
Haven, G Neil, 131.
Head of the list, 28.
Header elements, 225.
`HEAP` array, 67, 158, 233–235, 240.
Heap data structure, 67, 214.
  insertion and deletion from, 234.
Heaps of pieces, 83.
Height of a literal, 214.
Height of a trace, 85.
Heilmann, Ole Jan, 251.
Helpful rounds, 169.
Heule, Marienus (= Marijn) Johannes Hendrikus, v, 37, 40, 46, 71, 75, 97–98, 104, 129, 134, 147, 182, 186, 202, 213, 239, 260, 261, 263.
Heuristic scores, 90–95.
  for clauses, 72–74, 125–127, 158, 239, 286.
  for variables, 40–44, 61, 67, 80, 126, 145–147, 214.
Hidden weighted bit function, 173.
Hierarchy of hardness, 176, 178.
Hilton, Anthony John William, 191.
Hint clauses, 100, 114, 171.
Hirsch, Edward Alekseevich (Гирш, Эдуард Алексеевич), 215.
Historical notes, 32, 59–60, 105, 129–133, 231.
Hollow mazes, 290.
Holmes, Thomas Sherlock Scott, 72.
Homomorphic embedding, 169, 262.
Honest representations, 105, 270.
Hoory, Shlomo (שלמה חורי), 224.
Hoos, Holger Hendrik, v, 125–127, 133, 160.
Horn, Alfred, clauses, 132, 166, 176, 216, 263.
  core of, 174, 216.
  renamed, 176, 263.
Horsley, Daniel James, 274.
Horton, Robert Elmer, numbers, 152.
Hsiang, Jieh (項傑), 129.
Hume, David, 1.
Hunt, Warren Alva, Jr., 71, 239.
Hutter, Frank Roman, 125, 133.
Hypergraph 2-colorability, 185.
Hyperresolution, 56, 228, 257.

Idle Year solitaire, 180.
IEEE Transactions, vi.
If-then-else operation (u? v: w), 81, 102, 152, 173, 219.
ILS: Iterated Local search, 125.
Impagliazzo, Russell Graham, 55, 288.
Implication digraph, 52, 144.
in situ deletion, 156.
Inclusion and exclusion, 221–222, 256.
Inconsistent clauses, *see* Unsatisfiable formulas.
Indecomposable matrices, 177.
Independent vertices, 7, 147.
Induced graph, 262.
Induction proofs by machine, 24, 203.
Infinite loop, 244.
Initial guess for literals, 31, 33, 66, 125–127, 286.
Initial state X_0, 16–17, 21, 24, 140, 202.
Inprocessing, 95, 168.
Input and output, 120.
Input states, 175.
Insertion into a heap, 234.
Integer programming, 26, 184, 206, 285.
Interactive SAT solving, 142.
Interlaced roots of polynomials, 163.
Internet, ii, iv, v, 118.
Intersection graphs, 84, 161.
Interval graphs, 87, 163.
Intervals, cardinality constrained to, 100, 190, 280.
Invariant assertions, 23–24, 43, 115, 140, 203, 216, 217, 255, 261.
Inverse permutations, 112, 265.
Inversions of a permutation, 213.
Involution polynomial of a set, 163.
Involutions, signed, 112–113, 180, 277–278.
`INX` array, 38, 211, 286.
IP: Integer programming, 26, 184, 206, 285.
Irredundant CNF, 257.
Irreflexive relation, 56.
Irving, Robert Wylie, 151.
Isaacs, Rufus Philip, 218.
Isolated vertices, 262.
`IST` array, 38.
`ISTACK` array, 38, 145.
`ISTAMP` counter, 37–38, 46, 145.

Iterated local search, 125.
Iwama, Kazuo (岩間一雄), 224.

Jabbour, Saïd (ⵊⴰⴱⵓⵔ ⵙⴰⵄⴷ, سعيد جبور), 236, 289.
Jacquet, Philippe Pierre, 225.
Jagger, Michael Philip “Mick”, 1.
Janson, Carl Svante, v.
Järvisalo, Matti Juhani, 105, 132, 260, 261.
Jeannicot, Serge, 215.
Jeavons, Peter George, v.
Jerrum, Mark Richard, 151.
Job shop problems, 172.
Johnson, David Stifler, 184, 191.
Johnson, Samuel, viii.
Join of graphs, 162.

k-induction, 203.
Kμ: Kilomems = thousands of memory accesses, 98.
$K_{m,n}$ (complete bipartite graph), 176.
Kamath, Anil Prabhakar (अनिल प्रभाकर कामथ), 16.
Kaporis, Alexis Constantine Flora (Καπόρης, Αλέξιος Κωνσταντίνου Φλώρας), 51.
Karmarkar, Narendra Krishna (नरेन्द्र कृष्ण करमरकर), 16.
Karp, Richard Manning, 52, 268.
Karpiński (= Karpinski), Marek Mieczysław, 224.
Karpovsky, Mark Girsh, 285.
Kasif, Simon (שמעון כסיף), 265.
Katona, Gyula (Optimális Halmaz), 107.
Kautz, Henry Alexander, 79, 132.
Kaye, Richard William, 207.
Keller, Robert Marion, 83.
Kernel of a graph, 99, 134, 186, 188, 218.
  clauses for, 114, 134.
Kilomem (Kμ): One thousand memory accesses, 39, 98.
Kim, Jeong Han (김정한), 54.
King moves, 134, 169.
Kingwise connected cells, 170.
Kirousis, Lefteris Miltiades (Κυρούσης, Ελευθέριος Μιλτιάδης), 51.
Kitagawa, Satoshi (北川哲), 264, 267.
Kleine Büning (= Kleine-Büning), Hans Gerhard, 131, 185.
Kleitman, Daniel J (Isaiah Solomon), 276.
Knapsack problem with a partial ordering, 158.
Knessl, Charles, 225.
Knight moves, 115, 169.
Knuth, Donald Ervin (高德纳), i, ii, v, 1, 14, 16, 19, 51–52, 72, 74, 93, 94, 118, 125–127, 192, 193, 195, 197, 202, 210, 212, 213, 216, 227, 235–237, 240, 242, 249, 260, 264–267, 274, 278, 279, 282, 286, 290.
Knuth, John Martin (高小强), *see* Truth.
Kojevnikov, Arist Alexandrovich (Кожевников, Арист Александрович), 280.
Kolipaka, Kashyap Babu Rao (కొలిపాక కశ్యప్ బాబు రావు), 90, 161, 255.
Konev, Boris Yurevich (Конев, Борис Юрьевич), 281.
Kouřil, Michal, 5, 185.
Kroening, Daniel Heinrich Friedrich Emil, v, 203, 288.
Krom, Melven Robert, clauses, *see* 2SAT.
kSAT, 3, 49–51, 146, 148, 150, 183.
Kulikov, Alexander Sergeevich (Куликов, Александр Сергеевич), 280.
Kullmann, Oliver, v, 5, 105, 129, 147, 152, 215, 216, 218, 228, 260, 270, 273.
Kwekkeboom, Cornelis (= Kees) Samuël, 202.

$L(2,1)$ labeling of graphs, 136.
L_7 lattice, 255.
Labeled pyramids, 162.
Labeled traces, 162.
Lalas, Efthimios George (Λάλας, Ευθύμιος Γεωργίου), 51.
Lamport, Leslie B., 24, 204.
Land mines, 142.
Landman, Bruce Michael, 185.
Langford, Charles Dudley, problem of pairs, vii, 5–6, 34, 98, 121, 125, 134, 170, 186, 289.
$\mathit{langford}(n)$, 6, 34–35, 39, 97, 98, 114, 121, 134, 210, 236, 289.
$\mathit{langford}'(n)$, 6, 98, 114, 134, 289.
$\mathit{langford}''(n)$, 98.
$\mathit{langford}'''(n)$, 264.
Larrabee, Tracy Lynn, 13, 137.
Las Vegas algorithms, iv, 159–160.
Last in, first out, 10.
Late Binding Solitaire, 114, 180.
Latin rectangle construction, 151.
Lattices, 255, 276.
  of partial assignments, 165–166.
Lauria, Massimo, v, 56.
Laurier, Jean-Louis, 187.
Lazy data structures, 30–34, 36, 65, 156, 234.
Le Berre, Daniel Claude Yves, 132.
Learned clauses, 63–65, 70–71, 124, 132, 168.
  sequence of, 70, 156.
Learning a Boolean function, 14–16, 115, 137.
Least common ancestor, 253.
Left division of traces, 85, 161.
Left factor of a trace, 161–162.
Lemaire, Bernard François Camille, 282.
Lemma generation, *see* Clause-learning algorithms.
Length of a trace, 85.

Lettmann, Theodor August, 185.
Level 0, 62, 66, 124, 156, 207, 233.
Levels of values, 62–66, 156, 233.
Levesque, Hector Joseph, 50.
Levine, Eugene, 275.
Lewis, Jerome Luther, 275.
Lex-leader: The lexicographically smallest element, 111, 283.
Lexicographic order, 4, 25, 26, 30, 101, 105, 107, 109, 111–113, 115, 197, 282–283.
  encoded in clauses, 101, 173, 174.
Lexicographic row/column symmetry, 106–107, 177, 181, 274.
Lexicographically smallest (or largest) solution, 25–26, 111–113, 142, 157, 282, 283.
Lexicographically smallest traces, 84, 161, 162, 250.
Leyton-Brown, Kevin Eric, 125, 133.
Li, Chu Min (李初民), 131.
Li, Wei (李未), 149.
Lieb, Elliott Hershel, 251.
Life, Game of, 17–20, 97, 114, 137–139, 143, 167.
Light speed in Life, 139.
Line graph of a graph, 147, 249.
Linear equations, 26, 231.
Linear extensions, *see* Topological sorting.
Linear inequalities, 184.
  encoding of, 100–101, 172, 173.
Linear programming relaxation, 26.
Lines, abstracted, 106.
Links, dancing, 5, 121, 134, 208, 288.
Lisitsa, Alexei Petrovich (Лісіца, Аляксей Пятровіч), 281.
List coloring of graphs, 135, 151.
List merging, 231, 258.
Literal block distance, 72, 74, 158.
Literals, 2, 111.
  flushing, 76.
  internal representation, 28, 37, 66, 208, 209, 242, 257.
Litman, Ami (עמי ליטמן), 285.
Livelock, 22–23.
Llunell, Albert Oliveras i, 267.
LNCS: *Lecture Notes in Computer Science*, inaugurated in 1973.
Local Lemma, 81–90, 133, 151, 160–165.
Log encodings, 98, 114–115, 173.
Logemann, George Wahl, 31–32, 130, 298.
Longest simple path, 23, 203.
Lonlac, Jerry, 289.
Look-back, *see* Backjumping.
Lookahead autarky clauses, *see* Black and blue principle.
Lookahead forest, 42–44, 145–147, 168.
Lookahead solvers, 38–46, 55, 97, 129, 131, 176.
  combined with CDCL solvers, 129.
  compared to CDCL solvers, 98–100, 118–121, 182, 290.
Loopless shadows, 184.
Lopsidependency graphs, 82, 83, 160, 164, 165, 185, 224.
Lovász, László, 81, 82, 185, 191.
Loveland, Donald William, 32, 130, 298.
Lower semimodular lattices, 255–256.
Loyd, Samuel, 263.
Luby, Michael George, 80, 159.
Luks, Eugene Michael, 113.

Mμ: Megamems = millions of memory accesses, 69, 98.
Maaren, Hans van, 37, 46.
MacColl (= McColl), Hugh, 227.
MacMahon, Percy Alexander, Master Theorem, 250, 251.
Mader, Adolf, 275.
Madigan, Conor Francis, 132.
Magic, 193.
Magic sequences, 285.
Magnetic tape, 32.
Makespan, 172–173.
Mallach, Sven, 289.
Malik, Sharad (शरद मलिक), 132.
Maneva, Elitza Nikolaeva (Манева, Елица Николаева), 166, 256.
Mapping three items into two-bit codes, 179.
`march` solver, 40, 216.
Marek, Victor Wiktor, 216.
Markov (= Markoff), Andrei Andreevich (Марков, Андрей Андреевич), the elder, inequality, 158, 241.
Markov, Igor Leonidovich (Марков, Ігор Леонідович), 112, 281, 284.
Markström, Klas Jonas, 290.
Marques da Silva (= Marques-Silva), João Paulo, 132.
Marriage theorem, 224.
Martin, Alexander, 264.
Matching polynomial of a graph, 249.
Matchings in a graph: Sets of disjoint edges, 150, 230, 249.
  perfect, 109–110, 177.
Mathews, Edwin Lee (41), 67.
Matrix multiplication, 260.
Matroids, 276.
Mauro, David Whittlesey, 192.
Maximal elements, 56, 62, 97, 115, 153, 157, 167.
Maximal planar graphs, 186.
Maximum independent sets, 87, 136, 187, 188.
Maximum number of 1s, 106–109, 135, 136, 177.
MAXSAT lower bound, 184.
"Maybe" state, 20.
Mazurkiewicz, Antoni Wiesław, 83.
McColl (= MacColl), Hugh, 227.

McGregor, William Charles, 7, 188.
  graphs, 7–8, 114–115, 134, 135, 188.
Mean running time, 120.
Median operation, 9, 136, 179.
Median running times, 99, 120–124, 127.
Megamem (Mμ): One million memory accesses, 201.
Méjean, Henri-Michel, 226.
Mellin, Robert Hjalmar, transforms, 151.
Mem (μ): One 64-bit memory access, 34.
`MEM` array, 66, 68, 124, 156.
Memo cache, 60, 233.
Memoization technique, 233.
Memoryless property, 244.
Menagerie, 116–117.
Merge networks, 266.
Merging lists, 231, 258.
Mertens, Stephan, 51.
Message passing, 90–95, 165–166.
Method I, 61.
Method IA, 61, 154.
Method of Trees, 129.
Methuselah solitaire, 282.
Methuselahs in Life, 19.
Mézard, Marc Jean Marcel, 51, 90, 91, 95.
Midpoint inequalities, 266.
Mijnders, Sid, 213.
Mills, Burton Everett, 130.
Minesweeper, 142–143.
Minimally unsatisfiable clauses, 150, 153.
Minimum covers, 193.
MiniSAT, 67.
Minterms, 179.
Mitchell, David Geoffrey, 50.
Miters, 121, 182.
Mitsche, Dieter Wilhelm, 51.
Mixed metaphors, 76.
Mixed-radix number systems, 268.
`MMIX` computer, ii, 158.
Mobile Life paths, 18–19, 138–139.
  flipflops, 138.
Möbius, Augustus Ferdinand, functions, 86.
  series, 86, 160, 162–163, 165, 247, 249.
Mod 3 addition, 114, 179.
Mod 3 parity, 179.
Mod 4 parity, 179.
Model checking, 16–17, 137–141, 179–180.
Model RB, 149.
Modified full adders, 114, 280.
Modular lattices, 255.
`mone` (-1), 242.
Monien, Burkhard, 215.
Monkey wrench principle, 113, 181.
Monotone functions, 163.
  Boolean, 137, 281.
Monotonic clauses, 5, 133, 157.
Monotonic paths, 108, 276.
Montanari, Andrea, 95.
Monte Carlo algorithms, iv, 77–83, 158–160.
Monus operation ($x\dot{-}y=\max\{0,x-y\}$), vi, 92, 247, 268.
Moore, Edward Forrest, 202.
Morehead, Albert Hodges, 282.
Morel, Henri, 226.
Moser, Robin Alexander, 82, 254.
Moskewicz, Matthew Walter, 132.
Mossel, Elchanan (אלחנן מוסל), 166.
Mott-Smith, Geoffrey Arthur, 282.
Move codes, 29–31, 34, 144, 145, 155, 210.
MPR: Mathematical Preliminaries Redux, v.
Mueller, Rolf Karl, 60, 130.
Müller, Mike, 196.
Multicommodity flows, 170.
Multigraphs, 231.
Multilinear function, 86.
Multiplication of binary numbers, 8–9, 12–14, 114, 136, 173.
Multiplication of traces, 85, 161.
Multisets, 3, 214, 224, 250.
Multivalued graph colorings, 99.
Mutilated chessboard, 110, 114, 177–178, 286.
Mutual exclusion protocols, 20–24, 115, 139–141.
Mutzbauer, Otto, 275.
Mux operation (u? v: w), 81, 102, 152, 173, 219.
Mysterians, 290.

n-cube, 79, 136, 148, 184.
n.f.: not falsified, 271.
n queens problem, 25, 115, 171, 282.
NAND operation, 60.
Napier, John, Laird of Merchiston, 9, 173.
Near truth, 37–39.
Necessary assignments, 45, 146.
Negated auxiliary variables, 105.
Negative k-clauses, 157.
Negative literals, 2, 132, 153.
Nesting phase of lookahead, 40, 42–43, 145–147.
Newbie variables, 41.
Newton, Isaac, method for rootfinding, 216–217.
Niborski, Rodolfo, 190.
Niemelä, Ilkka Niilo Fredrik, 105.
Nieuwenhuis, Robert Lukas Mario, 267.
Nightingale, Peter William, 265.
No-player game, 17.
Nodes of a search tree, 34–35, 69, 124.
Noels, Alain, 202.
Noisy data, 181.
Nonattacking queens, 25, 115, 171, 282.
Nonaveraging sets, 114, 135.
Nonchromatic rectangles, 176–177.
Nonchronological backtracking, *see* Backjumping.
Noncommutative variables, 162.
Nonconstructive proofs, 57, 58, 81, 202.

Nondeterministic finite-state automata, 175.
Nondeterministic polynomial time, 131.
Nondeterministic processes, 20, 141, 182.
Nonintersecting paths, 170.
Nonnegative coefficients, 164.
Nonprimary columns, 186.
Nonterminal symbols, 175.
Normal chains, 278.
Normal functions, 279.
Not-all-equal SAT, 185.
Notational conventions, vi.
  ∂S (boundary set), 58, 154, 180, 188.
  $C' \diamond C''$ (resolvent), 54, 152.
  $C \subseteq C'$ (subsumption), 61, 152.
  $F\,|\,l$ (F given l), 27, 96, 291.
  $F\,|\,L$ (F given L), 27, 103, 157.
  $F \vdash C$ (F implies C), 59, 152, 153.
  $F \vdash_1 \epsilon$, 70, 157, 175.
  $F \vdash_1 l$, 103–104, 176.
  $F \vdash_k \epsilon$,
  $F \vdash_k l$, 175–176.
  $G \oplus H$ (direct sum), 162, 177.
  $|l|$ (a literal's variable), 2.
  $\pm v$ (v or $\bar{v}$), 2.
  $\langle xyz \rangle$ (median), vi, 9.
  $x \mathbin{\&} y$ (bitwise AND), 28, 29, 31, 37, 38, 66, 68, 76, 81, 196, 209–211, 220, 241.
  $x \mid y$ (bitwise OR), 43, 196, 241, 258–259.
  $x \oplus y$ (bitwise XOR), 28, 137, 196, 208, 220, 241.
  $x \dot{-} y$ (monus), 92, 247, 268.
  $x?\ y\colon z$ (if-then-else), 81, 102, 152, 173, 219.
  $w(\alpha)$, 57.
  $w(\alpha \vdash \epsilon)$, 57.
  $\|\alpha \vdash C\|$, 57.
  $\mu(C)$, 59, 153.
Novikov, Yakov Andreevich (Новиков, Яков Андреевич), 70, 132.
Nowakowski, Richard Joseph, 107, 275.
NP-complete problems, 1, 3, 27, 87, 130–131, 134, 142, 151, 181–183, 207, 268, *see also* CoNP-complete problems.
`NT` (near truth), 37–39.
Null clause (ϵ), 3, 27, 185, 291.
Null list, representation of, 33, 210.
Null partial assignment, 166.
Null set ($\emptyset$), 185.
Null string (ϵ), 85.
Nullary clause (ϵ), 3, 27, 185, 291.
Number theory, 14, 137, 192.

Occurrence threshold of a graph, 160.
Odd permutations, 218.
Odd-even merge network, 266.
Odd-even transposition sort, 263.
Oliveras i Llunell, Albert, 267.
On-the-fly subsumptions, 124, 156.
One-in-three satisfiability, 183.
One-per-clause satisfiability, 183.
Open shop scheduling problems, 115, 172–173.
OR operation, 9, 10, 13, 258.
  bitwise ($x \mid y$), 43, 196, 241, 258–259.
Orbits of a permutation group, 108, 277.
Order encoding, 98–101, 114, 120, 170–173, 190, 268, 281.
Order of a permutation, 111.
Organ-pipe permutations, 171.
Oriented cycle detection, 260.
Oriented trees, 108.
Orphan patterns in Life, 139.
Orponen, Olli Pekka, 80.
Oscillators in Life, 19, 138–139.
Output states, 175.
`OVAL` array, 74, 125, 237, 240.
Overfitting, 182.
Overflow in arithmetic, 67, 240.
Oxusoff, Laurent, 215.

$\wp$ (tautology, the always-true clause), 3, 58, 60, 152, 180, 215, 226–228, 258.
P = NP, 1.
Palindromes, 136.
Panagiotou, Konstantinos (Παναγιώτου, Κονσταντίνος), 221.
Papadimitriou, Christos Harilaos (Παπαδημητρίου, Χρίστος Χαριλάου), 77, 240, 241.
Parallel multiplication circuits, 12–14, 137.
Parallel processes, 20, 24, 121, 128–129.
Parameters, tuning of, 80–81, 93–94, 124–128.
ParamILS, 125, 286–287.
Parity-related clauses, 153–154, 172, 178, 231–232, 290.
Partial assignments, 30, 61, 62, 165, 176.
Partial backtracking, 208.
Partial latin square construction, 151.
Partial orderings, 56, 85, 115, 248, 276.
  of dimension ≤ 2, 213.
Participants, 41, 44, 145.
Path detection, 169.
Path graphs P_n, 84, 160, 253.
Patience, *see* Solitaire games.
Paturi, Ramamohan (రామ హన్ పాటూరి), 288.
Paul, Jerome Larson, 5.
Paull, Marvin Cohen, 148.
PC$_k$, 176, 178.
Pearl, Judea (יהודה פרל), 95.
Pegden, Wesley Alden, v, 164, 253.
Peierls, Rudolf Ernst, 95.
Peres, Yuval (יובל פרס), 221.
Pérez Giménez, Xavier, 51.
Perfect matchings in a graph, 109–110, 177.
Permanent of a matrix, 183, 251.
Permutation polynomial of a set, 163.
Permutation posets, 213.

Permutations, 105, 265.
  signed, *see* Signed permutations.
  weighted, 163.
Permuting variables and/or complementing them, *see* Signed permutations.
Peterson, Gary Lynn, 23, 115, 140, 204.
Petrie, Karen Elizabeth Jefferson, 283.
Phase saving, 67, 75.
Phase transitions, 50–52, 149–150.
Phi (ϕ), 146, 147, 160, 251.
Phoenix in Life, 198, 207.
Pi (π), as source of “random” data, 12, 46, 108, 115, 147, 184, 193, 286; *see also* Pi function.
Pi function, 102, 174.
Pieces, in trace theory, 84–87.
Pigeonhole principle, 57.
  clauses for, 57–59, 105–106, 113, 153, 176, 181, 186, 265.
Pikhurko, Oleg Bohdan (Піхурко, Олег Богданович), 285.
Pile sums, 151.
Pincusians, 133.
Pipatsrisawat, Thammanit (= Knot) (พิพัฒน์ศรีสวัสดิ์, ธรรมนิติ์ (= นอต)), 67, 262.
Pixel images, 200; *see also* Grid patterns.
Plaisted, David Alan, 102.
Planning, 132.
Playing cards, 114, 180, 282.
Points, abstracted, 106.
Poison cards, 282.
Poisson, Siméon Denis, probability, 225.
Polarities, 3, 67, 76, 207, 237.
Pólya, György (= George), theorem, 284.
Polynomials in trace theory, 85.
Population in Life, 19.
Portfolio solvers, 133.
Posets, *see* Partial orderings.
Positive autarkies, 146.
Positive j-clauses, 157.
Positive literals, 2, 132, 146.
Posthoff, Christian, 275.
Postorder, 42–43, 214.
Postprocessor, 96.
PostScript language, v.
Preclusion clauses, 99, 171, 186.
Preorder, 42–43, 214.
Preprocessing of clauses, 95–97, 103, 166–168, 182, 269, 271, 278.
Preselection phase of lookahead, 40–42, 147.
Prestwich, Steven David, 264.
Primary variables, 104, 105.
Prime clauses, 174, 270, 273.
Prime implicants, 281.
Pringsheim, Alfred Israel, 88, 164.
Prins, Jan Fokko, 267.
Probabilistic method, 81.
Probability of satisfiability, 47–54.
$prod(m,n)$, 12–14, 114, 137.
Production rules, 175.
Profile of a search tree, 151.
Progress, display of, 30, 145, 155.
Progress saving, 67, *see* Phase saving.
Projection of a path, 184.
Projective plane, 274.
Propagation, kth order, 175–176, 273.
Propagation completeness (UC_1), 176.
Proper ancestors, 164.
Proto truth, 37, 42.
Prover–Delayer game, 55–56, 152–153.
PSATO solver, 159.
Pseudo-Boolean constraints, *see* Threshold functions.
`PT` (proto truth), 37, 42.
Pudlák, Pavel, 55.
Puget, Jean-François, 113.
Purdom, Paul Walton, Jr., 30, 32, 131, 151, 226.
Pure cycles, 140.
Pure literals, 29, 31, 32, 34, 44, 60, 130, 135, 146, 152, 208, 215, 227, 256, 259, 268, 269, 275.
Purging unhelpful clauses, 68, 71–75, 124, 132, 157, 158, 168, 182, 184, 235.
  threshold for, 74, 125, 127.
Putnam, Hilary, 9, 32, 130, 298.
Pyramids in trace theory, 87, 162.

q.s.: quite surely, 149, 153, 169.
QDD: A quasi-BDD, 188.
Quad-free matrices, 106–107, 113, 176–177, 274, 284.
Quantified formulas, 60, 154.
Queen graphs, 25, 99–100, 114–115, 120, 171, 180, 181.
Quenchable graphs, 179–180, 281.
Quick, Jonathan Horatio, 181.
Quilt patterns, 198.
Quimper, Claude-Guy, 272.
Quine, Willard Van Orman, 129, 130.

$\mathcal{R}(G)$ (Local Lemma bounds), 82, 87–90, 160, 163–165.
Radio colorings, 136.
Radix-d representation, 173.
Rado, Richard, 191.
Ramakrishnan, Kajamalai Gopalaswamy, 16.
raman graphs, 231.
Ramani, Arathi (ஆர்த்தி ரமணி), 112, 281, 284.
Ramanujan Iyengar, Srinivasa (ஸ்ரீனிவாஸ ராமானுஜன் ஐயங்கார்), graphs, 154; *see also raman* graphs.
Ramos, Antonio, 75.
Ramsey, Frank Plumpton, theorem, 81.
rand, 39–40, 46, 50, 115, 147, 182.
Random bits, biased, 12, 241.
Random choices, 12.

Random decision variables, 125–127, 155, 286.
Random graphs, 81.
Random permutations, 233.
Random satisfiability problems, 47–54, 91, 151.
  2SAT, 51–54, 149.
  3SAT, 39–40, 46–51, 59–60, 80, 93–94, 147–149, 153, 242.
  kSAT, 49–51, 146, 148.
Random walks, 77–81, 125, 243.
Random words, 149.
Randomized methods, 77, 129, 182, 210.
RANGE scores, 74, 125–127, 158, 239.
RAT, *see* Resolution certifiable clauses.
Rauzy, Antoine Bertrand, 131, 215.
Reachability in a graph, 169.
Ready list, 32.
Real roots of polynomials, 163, 249.
Real truth, 37–39.
Reasons, 62, 72, 157, 165, 233.
Rebooting, 22.
Reckhow, Robert Allen, 61.
Recurrence relations, 151, 177, 189, 215, 243.
Recursive procedures, 27, 130, 172, 186, 233.
Recycling of clauses, 66, 124.
Reduction of clauses, 27, 143; *see also* Simplification of clauses.
Redundant clauses, 257.
Redundant literals, 65, 155–156, 232, 234.
Redundant representations, 171.
Reed, Bruce Alan, 52.
Reflected ternary code, 290.
Reflection symmetries, 112, 138, 156.
Refutation chains, 57, 227.
Refutation trees, 152.
Refutations, 54–60, 70, 110, 152; *see also* Certificates of unsatisfiability.
Regular expressions, 174–175.
Regular resolution, 55, 152, 231.
Reinforcement messages, 91–93.
Reliability polynomials, 83.
Reluctant doubling, iv, 77, 80–81, 159–160.
Reluctant Fibonacci sequence, 160.
Renamed Horn clauses, 176, 263.
Repeated clauses, 49.
Replacement principle, 96.
Representation of Boolean functions, 104, *see* Encoding into clauses.
Representing three states with two bits, 179.
Rescaled activity scores, 67.
Resende, *see* Guilherme De Carvalho Resende.
Resizing of data structures, 210.
Resolution certifiable clauses, 261.
Resolution chains, 57–59, 152, 153, 227.
Resolution of clauses, 54–65, 70, 101, 129, 130, 144, 167, 185, 215, 224, 256.
  implementation of, 167.
Resolution refutations, 54–60, 70, 110, 152; *see also* Certificates of unsatisfiability.
  extended, 60, 71, 133, 154, 168, 215.
  regular, 55, 152, 231.
  treelike, 55–56, 152–153.
Resolvable clauses, 164.
Resolvent ($C' \diamond C''$), 54, 130, 152.
Restarting, 80–81, 95, 125, 132.
  and flushing literals, 68, 75–77, 124, 132, 157, 158, 169, 234, 246.
Restricted growth strings, 179.
Restricted pigeonhole principle, 58.
Reusing the trail, 75.
Reverse unit propagation, 71.
Revolving door Gray code, 282.
Reynaud, Gérard, 226.
Richards, Keith, 1.
Rickard, John, 290.
Right division of traces, 85, 161.
Right factor of a trace, 161.
Riis, Søren Møller, 110.
Ripoff, Robert Iosifovich (Рипов, Роберт Иосифович), 7.
Rivest, Ronald Linn, clauses, 4, 55, 70, 134, 144, 182.
Roberts, Fred Stephen, 136.
Robertson, Aaron Jon, 185.
Robinson, Gilbert de Beauregard, 275.
Robinson, John Alan, 59, 96, 227.
Rodríguez Carbonell, Enric, 267.
Rokicki, Tomas Gerhard, 200.
Rooij, Iris van, 207.
Rook paths, 206.
Rookwise connected cells, 170.
Ross, Kenneth Andrew, 282.
Rotational symmetry, 138, 202, 275.
Rotors in Life, 138.
Roussel, Olivier Michel Joseph, 132, 272.
Routing, disjoint, 170.
Row sums, 151.
Roy, Amitabha (অমিতাভ রায়), 113.
RT (real truth), 37–39, 43.
Ruler doubling, 160.
Ruler of Fibonaccis, 246.
Running times, 89–90.
  comparison of, 34–35, 39, 69, 97–100, 105–107, 110, 112, 118–128, 182, 184, 218, 237, 264, 281, 290.
  mean versus median, 120.
  worst case, 144, 146, 154.
Runs of 1s, 26, 143, 175.

s-chains, 52–53, 149.
s-snares, 53, 149.
$S_1(y_1, \ldots, y_p)$, 6.
$S_k(m, n)$, 50–54.
$S_{k,n}$, 49–51, 148, 149.
$S_{\leq r}(x_1, \ldots, x_n)$ and $S_{\geq r}(x_1, \ldots, x_n)$, 8, *see* Cardinality constraints.
Saddle point method, 226.

Sahni, Sartaj Kumar (सरताज कुमार साहनी), 268.
Saïs, Lakhdar (ⵙⴰⵉⵙ ⵍⴰⵅⴹⴰⵔ, سايس لخضر), 236, 289.
Sakallah, Karem Ahmad (أحمد ساقالله كارم), 112, 132, 281, 284.
Salhi, Yakoub (يعقوب صالحي), 289.
Sampling with and without replacement, 49–50, 132, 226.
Samson, Edward Walter, 60, 130.
SAT: The satisfiability problem, 3.
SAT solvers, 1, 131–133.
SATexamples.tgz, iv, 118.
Satisfiable formulas, 1.
  variability in performance, 35, 120–121, 128, 287.
Satisfiability, 1–184.
  history, 32, 59–60, 105, 129–133.
  thresholds for, 50–54, 91, 148–149, 221.
Satisfiability-preserving transformations, 107–113.
Satisfying assignments, 1, 30, 143–144, 166, 214, 219.
SATzilla solver, 132–133.
Schaefer, Thomas Jerome, 289.
Schensted, Craige Eugene (= Ea Ea), 275.
Schlipf, John Stewart, 273.
Schmitt, John Roger, 285.
Schoenfield, Jon Ellis, 192.
Schöning, Uwe, 78.
Schrag, Robert Carl, 132.
Schroeppel, Richard Crabtree, 197.
Schwarzkopf, Bernd, 282.
Scott, Alexander David, 224, 251, 252.
Scott, Allan Edward Jolicoeur, 207.
Scott, Sidney Harbron, 191.
Scoville, Richard Arthur, 162.
Search trees, 28–29, 32–34, 124, 152.
  expected size, 151–152.
  optimum, 144.
Second moment principle, 54, 221, 222.
Seitz, Simo Sakari, 80.
Self-subsumption, 96, 167, 168, 257.
Selman, Bart, 50, 79, 132.
Semimodular lattices, 255–256.
Sentinel values, 259.
Sequential consistency, 24.
Sequential lists, 36–37, 144.
Sequents, 59.
Serial correlation coefficients, 143.
Set partitions, 220.
SGB, *see* Stanford GraphBase.
Shadows of paths, 184.
Shandy, Tristram, iii.
Sharp thresholds, 51–52, 149.
Shearer, James Bergheim, 82, 87, 160.
Sheeran, Mary, 203.
Shlyakhter, Ilya Alexander (Шляхтер, Илья Александрович), 284.
Shmoys, David Bernard, 267.
Shortest paths, 262.
Shortz, William Frederic, v.
SIAM: The Society for Industrial and Applied Mathematics, 204.
Sideways sum (νx): Sum of binary digits, 114, 143, 179, 195, 279.
  second order ($\nu^{(2)}x$), 143.
Sifting, 219, 220.
Siftup in a heap, 234.
Signature of a clause, 72, 158.
Signature of a literal, 258.
Signed mappings, 180–181.
Signed permutations, 4, 111, 178.
  involutions, 112–113, 180, 277–278.
Silva, *see* Marques da Silva.
Silver, Stephen Andrew, 138, 200.
Simmons, Gustavus James, 192.
Simon, Laurent Dominique, 72, 132.
Simple cycles and paths, 23–24, 140.
simplex graphs, 136.
Simplification of clauses, 65, 155, 232; *see also* Preprocessing of clauses.
Sims, Charles Coffin, tables, 283.
Simultaneous read/write, 141.
Simultaneous write/write, 141.
Sinclair, Alistair, 80, 159, 256.
Singh, Satnam, 203.
Single lookahead unit resolution, 105, 176.
Single-stuck-at faults, 10–14, 114, 136–137.
Sink: A vertex with no successor, 87, 214.
  components, 108–110.
Sinz, Carsten Michael, v, 8, 117, 118, 135, 174, 189, 280.
Skip Two solitaire, 282.
Slack, in trace theory, 88, 251.
Slisenko (= Slissenko), Anatol Olesievitch (Слисенко, Анатоль Олесьевич), 59.
SLS: Stochastic local search, 77.
SLUR algorithm, 105, 176.
Sly, Allan Murray, 51.
Smile, 207.
Smith, Barbara Mary, 283.
Snake dance, 138.
Snakes, 52–54, 149.
Snares, 52–54, 149.
Snark graphs, 69, 147, 153, 157.
Snevily, Hunter Saint Clair, 5.
Socrates, son of Sophroniscus of Alopece (Σωκράτης Σωφρονίσκου Ἀλωπεκῆθεν), 129.
Soft clauses, 168.
Sokal, Alan David, 251, 252.
Solitaire games, 180, 282.
Solutions, number of, 48, 219.
Somenzi, Fabio, 236.
Sörensson, Niklas Kristofer, v, 67, 155, 203, 268.
Sorting networks, 115, 137, 203, 263, 266.

Source: A vertex with no predecessor, 87, 252.
Spaceships in Life, 139, 201.
Spanning trees, 281, 290.
Sparse encoding, *see* Direct encoding.
Speckenmeyer, Ewald, 131, 215.
Spence, Ivor Thomas Arthur, 290.
Spencer, Joel Harold, 81, 82, 254.
Sperner, Emanuel, k-families, 276.
Spiral order, 206.
Stable Life configurations, 19, 197.
Stable partial assignments, 165–166.
Stacks, 37–39, 43.
Stacking the pieces, 84–85.
Stålmarck, Gunnar Martin Natanael, 56, 132, 153, 203, 232, 238.
Stamm-Wilbrandt, Hermann, 131.
`STAMP(l)`, 258.
Stamping of data, 37–38, 64, 66, 145, 155, 211, 236, 258–260.
Standard deviation, 48, 240.
Stanford GraphBase, ii, 12, 13, 126, 179, 214, 231.
Stanford Information Systems Laboratory, v.
Stanford University, 282.
Stanley, Richard Peter, 275.
Starfish graphs, 249.
Starvation, 22–24, 115, 140, 141.
Statistical mechanics, 90.
Stators in Life, 138.
Stege, Ulrike, 207.
Stein, Clifford Seth, 267.
Steinbach, Heinz Bernd, 275.
Steiner, Jacob, tree packing, 264.
  triple systems, 106, 274.
Sterne, Laurence, iii.
Stickel, Mark Edward, 132.
Sticking values, 67, *see* Phase saving.
Still Life, 19, 138, 200.
Stirling, James, approximation, 221, 240.
  subset numbers, 149, 220, 226.
Stochastic local search, 77.
Stopping time, 48–50, 148.
Strahler, Arthur Newell, numbers, 152.
Strengthening a clause, 96, 156, 259–260.
Stříbrná, Jitka, 224.
Strichman, Ofer (עופר שטריכמן), 203.
Strictly distinct literals, 2–3, 52, 165.
Strings generalized to traces, 83.
Strong components: Strongly connected components, 41–42, 52–53, 108, 131, 215, 221, 263.
Strong exponential time hypothesis, 183.
Strong product of graphs, 134.
Strongly balanced sequences, 179.
Stuck-at faults, single, 10–14, 114, 136–137.
Stützle, Thomas Günter, 125.
Subadditive law, 59.
Subcubes, 148.
Subforests, 42.
Subinterval constraints, 190.
Submatrices, 106–109, 177.
Subset sum problem, 268.
Substitution, 257.
Subsumption of clauses, 61, 96, 124, 152, 155, 156, 166–168, 181, 269.
  implementation, 167, 259.
  on-the-fly, 124, 156.
Subtraction, encoding of, 100.
Sudoku, 183, 225.
Summation by parts, 48.
Summers, Jason Edward, 200.
Sun, Nike (孙妮克), 51.
Support clauses, 99, 114, 171.
Survey propagation, 51, 90–95, 165–166, 213.
Swaminathan, Ramasubramanian (= Ram) Pattu (ராமசுப்ரமணியன் பட்டு சுவாமிநாதன்), 273.
Swapping to the front, 211, 242.
Sweep of a matrix, 108–109, 177.
Swoop of a matrix problem, 109.
Syllogisms, 129.
Symeater in Life, 200.
Symmetric Boolean functions, 179, 207, 219, 270; *see also* Cardinality constraints.
  $S_{\leq 1}$, *see* At-most-one constraint.
  S_1, 6, 220.
  $S_{\geq 1}$, *see* At-least-one constraint.
  S_r, 135, 179, 256.
Symmetric threshold functions, *see* Cardinality constraints.
Symmetrical clauses, 105–106, 156.
Symmetrical solutions, 138, 183, 274.
Symmetries of Boolean functions, 178.
Symmetry breaking, vii, 5, 105–114, 138, 176–181, 187, 188, 190–192, 238, 267, 281–283, 285, 288–290.
  in graph coloring, 99–100, 114, 171, 179, 187.
Symmetry from asymmetry, 19, 201.
Synthesis of Boolean functions, 137, 178–179, 194.
Szabó, Tibor András, 224.
Szegedy, Márió, 90, 161, 255.
Szeider, Stefan Hans, 224, 284.
Szemerédi, Endre, 59.
Szpankowski, Wojciech, 225.

t-snakes, 53, 54, 149.
Tμ: teramems = trillions of memory accesses, 110, 121, 126, 265, 281.
Tableaux, 275.
Taga, Akiko (多賀明子), 264, 267.
Tajima, Hiroshi (田島宏史), 100.
Tak, Peter van der, 75.
Takaki, Kazuya (高木和哉), 224.
Tamura, Naoyuki (田村直之), 100, 171, 264, 267, 268.
"Take account," 37, 43, 45–46, 217, 235.

Tanjo, Tomoya (丹生智也), 268.
TAOCP: *The Art of Computer Programming*, problem, 115, 169.
Tape records, 32.
Tardos, Gábor, 82, 224, 254.
Tarjan, Robert Endre, 41, 42, 214, 217.
Tarnished wires, 13, 193.
Tatami tilings, 115, 143.
TAUT: The tautology problem, 3, 129, 130.
Tautological clause ($\wp$), 3, 58, 60, 152, 180, 215, 226–228, 258.
Tensors, 151.
Teramem (Tμ): One trillion memory accesses, 40, 106, 107, 110, 217, 218, 286.
Ternary clauses, 3–6, 36, 118, 131, 183; *see also* 3SAT.
Ternary numbers, 100, 141, 179.
Ternary operations, 9, 136.
Territory sets, 84, 161, 163.
Test cases, 113–124.
  capsule summaries, 114–115.
Test patterns, *see* Fault testing.
Tetris, 84.
Theobald, Gavin Alexander, 190.
Theory and practice, 109.
Three-coloring problems, *see* Flower snarks.
Threshold functions, 100–101, 175.
Threshold of satisfiability, 50–54, 91, 148–149, 221.
Threshold parameter Θ, 126, 213, 286.
Thurley, Marc, 262.
Tie-breakers, 74, 239.
Tiling a floor, 115, 138, 143, 199.
Time stamps, *see* Stamping of data.
Timeouts, 120.
TIMP tables, 36–40, 43, 45, 144–145.
To-do stack, 259.
Tomographically balanced matrices, 141.
Tomography, 24–26, 115, 141–143, 167, 285.
Top-down algorithms, 252.
Topological sorting, 85, 248.
Toruses, 134, 138, 200.
Touched clauses, 44.
Touched variables, 259.
Tovey, Craig Aaron, 150, 223.
Tower of Babel solitaire, 282.
Tower of London solitaire, 282.
Trace of a matrix: The sum of its diagonal elements, 108, 218.
Traces (generalized strings), 83–90, 161–162, 252, 254.
Tradeoffs, 125–126.
Trail (a basic data structure for Algorithm C), 62–65, 68, 72, 124, 166, 236, 238.
  reusing, 75.
Training sets, 15–16, 115, 125–127, 133, 137, 182, 286.
Transitions between states, 16–24, 175, 202, 218.
Transitive law, 56, 228.
Tree-based lookahead, *see* Lookahead forest.
Tree function, 230.
Tree-ordered graphs, 163–164.
Treelike resolution, 55–56, 152–153.
Treengeling solver, 121.
Triangle-free graphs, 167.
Triangles (3-cliques), 167, 238, 264.
Triangular grids, 136.
Tribonacci numbers, 216.
Triggers, 46, 126.
Trivalent graphs, 147, 154, 231.
Trivial clauses, 124–127, 156, 236, 239.
Trivially satisfiable clauses, 3.
Truemper, Klaus, 273.
Truszczyński, Mirosław (= Mirek) Janusz, 216.
Truth, degrees of, 37–39, 42–43, 45–46, 216.
Truth tables, 129–130, 179, 194, 220, 277.
Tseytin, Gregory Samuelovich (Цейтин, Григорий Цамуилович), 9, 59–60, 71, 133, 152, 154, 168, 178, 215, 231, 290.
  encodings, 9, 17, 101–102, 136, 173, 195.
  encodings, half of, 192, 268.
Tsimelzon, Mark Boris, 134.
Tuning of parameters, 124–128, 133, 182.
Turán, Pál (= Paul), 190.
Turton, William Harry, 180.
Two-level circuit minimization, 257.

UC$_k$, 176, 273.
UIP: Unique implication point, 132, 233.
Unary clauses, *see* Unit clauses.
Unary representation (= order encoding), 98–101, 114, 120, 170–173, 190, 268, 281.
Undoing, 28–31, 37–39, 95–96, 143–145, 208, 212, 217–218.
Uniform distribution, 159.
Unique implication points, 132, 233.
Uniquely satisfiable clauses, 48, 219.
Unit clauses (= unary clauses), 3, 6, 9, 13, 21, 23, 30, 31, 33, 35, 36, 66, 70, 130, 144, 151, 157, 192, 205, 210, 238, 290.
Unit conditioning, 27, 96, 166, 259, 261.
Unit propagation ($\vdash_1$), 31–34, 36, 62, 65, 68, 70–71, 93, 97–99, 103–104, 132, 155, 157, 165, 171, 174, 236, 269, 272, 276.
  generalized to $\vdash_k$, 175.
Universality of Life, 17.
Unnecessary branches, 55, 227.
Unsatisfiable core, 185.
Unsatisfiable formulas, 1.
  implications of, 104, 175–176.
Unsolvable problems, 130.
Urns and balls, 221.
Urquhart, Alisdair Ian Fenton, 231.

VAL array, in Algorithm C, 66–68, 73–76, 233–236, 238, 240.
  in Algorithm L, 37–39, 43, 216.

Valid partial assignments, 165–166.
Van de Graaff, Robert Jemison, 198.
van der Tak, Peter, 75.
van der Waerden, Bartel Leendert, 4.
numbers, 5, *see* $W(k_0, \ldots, k_{b-1})$.
van Deventer, Mattijs Oskar, 290.
Van Gelder, Allen, 71, 233, 237, 263.
van Maaren, Hans, 37, 46.
van Rooij, Iris, 207.
van Zwieten, Joris Edward, 37.
`VAR` array, in Algorithm L, 38, 182, 211.
Variability in performance on satisfiable problems, 35, 120–121, 128, 287.
on unsatisfiable problems, 69, 121, 128, 287.
Variable elimination, 96–97, 101, 102, 129, 154–155, 166–168, 173, 174, 256–257, 259–260, 270, 272.
Variable interaction graphs, 116–118, 182.
Variables, 2.
introducing new, 3, 6, 8, 9, 13, 60; *see* Auxiliary variables, Extended resolution.
Variance, 49, 158, 164, 240, 243.
Vassilevska Williams, Virginia Panayotova (Василевска, Виргиния Панайотова), 167.
Vaughan, Theresa Phillips, 162.
Verification, 16, 157; *see also* Certificates of unsatisfiability.
Viennot, Gérard Michel François Xavier, 83, 84, 87, 162, 249.
Vinci, Leonardo di ser Piero da, 7.
Virtual unswapping, 211.
Visualizations, 116–118.
Vitushinskiy, Pavel Viktorovich (Витушинский, Павел Викторович), 282.
Vries, Sven de, 206.
VSIDS, 132.

$W(k_0, \ldots, k_{b-1})$ (van der Waerden numbers), 4–5, 127, 133.
waerden$(j, k; n)$, 4–5, 32, 35, 37, 39–42, 45, 63–66, 69, 71–75, 97, 112, 115, 121, 127–129, 133, 142–145, 156, 157, 166, 167, 181, 210, 236, 256.
Waerden, Bartel Leendert van der, 4.
numbers, 5, *see* $W(k_0, \ldots, k_{b-1})$.
Wagstaff, Samuel Standfield, Jr., 190.
Wainwright, Robert Thomas, 138, 166, 197, 198.
Walks in a graph, 260.
WalkSAT algorithm, 79–81, 93–94, 118, 125, 159–160, 182, 191, 265, 281.
Walsh, Toby, 272.
Warmup runs, 125, 239.
Warners, Johannes (= Joost) Pieter, 268.
Warrington, Gregory Saunders, 285.
Watched literals, 30–34, 65–66, 68, 132, 144, 155, 233–236.
Weakly forcing, 174.
Websites, ii, iii, v, 118.
Weighted permutations, 163.
Wein, Joel Martin, 267.
Weismantel, Robert, 264.
Welzl, Emmerich Oskar Roman (= Emo), 158.
Wermuth, Udo Wilhelm Emil, v.
Wetzler, Nathan David, 71, 239.
Wheel graphs (W_n), 191.
Whittlesey, Marshall Andrew, 192.
Width of a resolution chain, 57–59, 153–154.
Wieringa, Siert, 129.
Wigderson, Avi (אבי ויגדרזון), 57–58, 153, 231.
Wilde, Boris de, 213.
Williams, Richard Ryan, v, 270.
Williams, Virginia Panayotova Vassilevska (Виргиния Панайотова Василевска), 167.
Wilson, David Bruce, 54, 149, 221.
Windfalls, 43, 147, 182, 217.
Winkler, Peter Mann, 290.
Winn, John Arthur, Jr., 275.
Wires of a circuit, 10–14, 136.
Wobble function, 51, 151.
Worst case, 144, 146, 154, 239, 244.
Write buffers, 24.

Xeon computer, 289.
XOR operation, 9, 10, 13, 136.
bitwise ($x \oplus y$), 28, 137, 196, 208, 220, 241.
Xray-like projections, 24.
Xu, Ke (许可), 149.
Xu, Lin (徐林), 133.
Xu, Yixin (徐一新), 255.

Yaroslavtsev, Grigory Nikolaevich (Ярославцев, Григорий Николаевич), 280.
Yeh, Roger Kwan-Ching (葉光清), 192.
Yuster, Raphael (רפאל יוסטר), 260.

$Z(m, n)$ (Zarankewicz numbers), 106–107, 176.
Zanette, Arrigo, 206.
Zarankiewicz, Kazimierz, 106.
quad-free problem, 106–107, 113, 176.
Závodný, Jakub, 196.
Zecchina, Riccardo, 51, 90, 91, 256.
Zhang, Hantao (张瀚涛), 129, 132.
Zhang, Lintao (张霖涛), 132.
Zhao, Ying (赵颖), 132.
Zhu, Yunshan (朱允山), 132.
`ZSEV` (zero or set if even), 242.
Zuckerman, David Isaac, 80, 159.
Zwick, Uri (אורי צוויק), 260.
Zwieten, Joris Edward van, 37.