The Art of Computer Programming

VOLUME 4B

Combinatorial Algorithms
Part 2

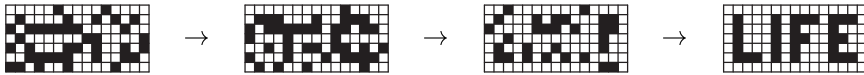# DONALD E. KNUTH

THE ART OF
COMPUTER PROGRAMMING

Fig. 78. 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. 78, 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, $\ldots$ 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. 78 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. 78, 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. 78 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 $\{x_{NW}, x_N, x_{SW}, x_S\}$ in common; so we need to compute $x_{NW} + x_N + x_{SW} + x_S$ 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 $X_4 = \mathsf{LIFE}$ 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. 78 was to try $X_3 = \mathsf{LIFE}$; 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

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

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 $1 \leq i, j \leq 8$, 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

$$\to\to\to\to\to\to\to\to\to\cdots,\quad(37)$$

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 $r = 8$:

$$\to\to\to\to\to\to\to\to;$$

$$\to\to\to\to\to\to\to\to;$$

$$\to\to\to\to\to\to\to\to;$$

$$\to\to\to\to\to\to\to\to;$$

$$\to\to\to\to\to\to\to\to.$$
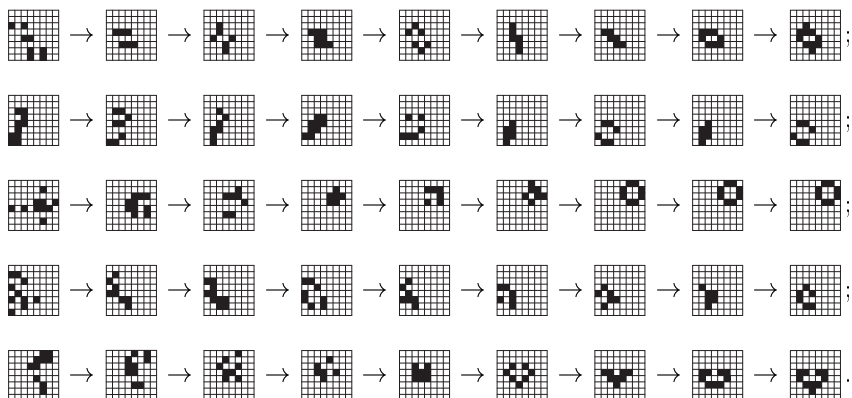
These paths illustrate the fact that symmetry can be gained, but never lost, as Life evolves deterministically. Marvelous designs are spawned in the process. In each of these sequences the next bitmap, X_9, would break our ground rules: The population immediately after X_8 grows to 12 in the first and 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 repetitive 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

$$X_0 = \;\; \rightarrow \;\; \rightarrow \;\; \rightarrow \;\; \rightarrow \;\; = \; X_5, \qquad (38)$$

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

$$\rightarrow \;\; \rightarrow \;\; \rightarrow \;\; \rightarrow \;\; \rightarrow \;\; , \qquad (39)$$

the four-step eater that was 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 $A0_t$, $A1_t$, $A2_t$, $A3_t$, $A4_t$, $B0_t$, $B1_t$, $B2_t$, $B3_t$, $B4_t$, l_t, together with ten binary exclusion clauses $(\overline{A0_t} \vee \overline{A1_t})$, $(\overline{A0_t} \vee \overline{A2_t})$, ..., $(\overline{A3_t} \vee \overline{A4_t})$ 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

$$A0_0 \wedge \overline{A1_0} \wedge \overline{A2_0} \wedge \overline{A3_0} \wedge \overline{A4_0} \wedge B0_0 \wedge \overline{B1_0} \wedge \overline{B2_0} \wedge \overline{B3_0} \wedge \overline{B4_0} \wedge \bar{l}_0, \quad (41)$$

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

$$
\begin{array}{lll}
(@_t \vee \overline{A0_t} \vee A0_{t+1}) & (\overline{@_t} \vee \overline{A0_t} \vee A0_{t+1} \vee A1_{t+1}) & (@_t \vee \overline{B0_t} \vee B0_{t+1} \vee B1_{t+1}) \\
(@_t \vee \overline{A1_t} \vee A1_{t+1}) & (\overline{@_t} \vee \overline{A1_t} \vee \bar{l}_t \vee A1_{t+1}) & (@_t \vee \overline{B1_t} \vee \bar{l}_t \vee B1_{t+1}) \\
(@_t \vee \overline{A2_t} \vee A2_{t+1}) & (\overline{@_t} \vee \overline{A1_t} \vee l_t \vee A2_{t+1}) & (@_t \vee \overline{B1_t} \vee l_t \vee B2_{t+1}) \\
(@_t \vee \overline{A3_t} \vee A3_{t+1}) & (\overline{@_t} \vee \overline{A2_t} \vee A3_{t+1}) & (@_t \vee \overline{B2_t} \vee B3_{t+1}) \\
(@_t \vee \overline{A4_t} \vee A4_{t+1}) & (\overline{@_t} \vee \overline{A2_t} \vee l_{t+1}) & (@_t \vee \overline{B2_t} \vee l_{t+1}) \\
(\overline{@_t} \vee \overline{B0_t} \vee B0_{t+1}) & (\overline{@_t} \vee \overline{A3_t} \vee A4_{t+1}) & (@_t \vee \overline{B3_t} \vee B4_{t+1}) \\
(\overline{@_t} \vee \overline{B1_t} \vee B1_{t+1}) & (\overline{@_t} \vee \overline{A4_t} \vee A0_{t+1}) & (@_t \vee \overline{B4_t} \vee B0_{t+1}) \\
(\overline{@_t} \vee \overline{B2_t} \vee B2_{t+1}) & (\overline{@_t} \vee \overline{A4_t} \vee \bar{l}_{t+1}) & (@_t \vee \overline{B4_t} \vee \bar{l}_{t+1}) \\
(\overline{@_t} \vee \overline{B3_t} \vee B3_{t+1}) & (\overline{@_t} \vee l_t \vee A2_t \vee A4_t \vee \bar{l}_{t+1}) & (@_t \vee l_t \vee B2_t \vee B4_t \vee \bar{l}_{t+1}) \\
(\overline{@_t} \vee \overline{B4_t} \vee B4_{t+1}) & (\overline{@_t} \vee \bar{l}_t \vee A2_t \vee A4_t \vee l_{t+1}) & (@_t \vee \bar{l}_t \vee B2_t \vee B4_t \vee l_{t+1})
\end{array}
\quad (42)
$$

If we now add the unit clauses $(A3_r)$ and $(B3_r)$, 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 $l = 0$:

$$
\begin{array}{ll}
\text{A0. Maybe go to A1.} & \text{B0. Maybe go to B1.} \\
\text{A1. If } l \text{ go to A2, else to A1.} & \text{B1. If } l \text{ go to B1, else to B2.} \\
\text{A2. Critical, go to A3.} & \text{B2. Critical, go to B3.} \\
\text{A3. Set } l \leftarrow 0, \text{ go to A0.} & \text{B3. Set } l \leftarrow 1, \text{ go to B0.}
\end{array}
\quad (43)
$$

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.
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.

$$(44)$$

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.
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.

$$(45)$$

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.
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.

$$(46)$$

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

$$X_0 \rightarrow X_1 \rightarrow \cdots \rightarrow X_p \rightarrow X_{p+1} \rightarrow \cdots \rightarrow X_r = X_p \qquad (47)$$

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.