

National University of Ireland, Galway

SAT Solver Recipe Book

by

Conor Cosnett

A project report submitted in partial fulfillment for the
degree of Higher Diploma in Applied Mathematics

in the
School of Mathematics, Statistics and Applied Mathematics, NUIG

April 2020

“Civilization advances by extending the number of important operations which we can perform without thinking of them.”

Alfred North Whitehead (1911)

Contents

1	Introduction	1
1.1	What is the Boolean Satisfiability Problem?	3
1.1.1	Example 1	3
1.2	What is a SAT Solver?	3
2	Game of Life	5
2.1	Conway's Game of Life	5
2.2	The Four Rules of GOL	6
2.2.1	Rule:1 Reproduction	7
2.2.1.1	Natural Language Description:	7
2.2.1.2	Diagrammatic Description:	7
2.2.1.3	First Order Logic Description:	7
2.2.2	Rule:2 Steady State	8
2.2.2.1	Natural Language Description:	8
2.2.2.2	Diagrammatic Description:	8
2.2.2.3	First Order Logic Description:	8
2.2.3	Rule:3 Survival	8
2.2.3.1	Natural Language Description:	8
2.2.3.2	Diagrammatic Description:	8
2.2.3.3	First Order Logic Description:	9
2.2.4	Rule:4 Underpopulation or Overcrowding	9
2.2.4.1	Natural Language Description:	9
2.2.4.2	Diagrammatic Description:	9
2.2.4.3	First Order Logic Description:	9
2.3	Encoding Rule 1 (Reproduction) into Boolean Logic	10
2.3.1	Unpacking the Predicate	10
2.4	Encoding Rule 2 (Steady State) into Boolean Logic	15
2.4.1	Unpacking the negated Predicate	15
2.5	Encoding Rule 3 (Survival) into Boolean Logic	21
2.5.1	Unpacking the predicates:	21
2.6	Encoding Rule 4 (Underpopulation or Overcrowding) into Boolean Logic	25
2.6.1	Unpacking the predicates	25
2.7	So What Do We Do Now?	30
2.8	Representing set S in computer code	30
2.9	Setting the states of cells in generation 6 such that they print "Götz"	31

2.10	Looping the Clauses Over the 7×15 Arrays	31
2.11	passing the $7 \times 15 \times 5 \times 190 = 99750$ clause set to a sat solver	31
2.12	Using the author's package	32
3	Conclusion	33
4	Appendix A	34
4.1	34
4.2	37
4.3	40
4.4	43
4.5	54
4.6	65
4.7	70
4.8	75
4.9	85
4.10	95
5	Appendix B	96
5.1	Author's package <code>wordLife.m</code>	96
5.2	Clause sets represented in Mathematica code and stored in package file: <code>clause_sets_R1_R2_R3_and_R4_.m</code>	98
	Bibliography	125

Chapter 1

Introduction

The famous NP-Complete class of problems contains many problems that a Software Engineer can encounter and would like to write software to solve but cannot because of their extreme complexity. These problems include vehicle routing, electronic circuit design, software testing, theorem proving, protein folding and Sudoku.

By definition NP-Complete problems are all the same problem. They can all be converted into each other such that a program that efficiently solves one problem is able to efficiently solve them all.

One of the most studied NP-Complete problems is the Boolean Satisfiability Problem (abbreviated SAT). SAT was the first problem proven to be NP-Complete (Cook 1971).

A program that solves the SAT problem is called a SAT Solver. When given a Boolean formula, a SAT Solver returns a truth assignment that satisfies the formula (if such an assignment is possible).

At the dawn of the millennium a series of remarkable breakthroughs took place that made SAT Solvers over a thousand times faster and become practical in real world applications. With speeds of up to $O(1.307^N)$ today's Solvers can handle millions of Boolean variables.

Made enthusiastic by reading about the potential of this new technology the author of this report went onto Stack Overflow and searched for questions with the tag [sat-solvers]. He expected to read hundreds of posts involving developers discussing the everyday niggles of working with SAT Solvers. Less than 60 questions appeared in his search results. It seems that the barrier to entry is that a Software Engineer has to first describe his or her problem using a particular subset of Boolean Logic called Conjunctive Normal Form (CNF) before he or she can solve it with a SAT solver. This

“knowledge engineering” process is labour intensive and time consuming. For instance the rules of chess require 100’000 pages of Boolean Logic to describe. The “Good Old Fashioned Artificial Intelligence” that was much hyped in the 1980s fell into a funding winter because of pessimism about the laborious process of translating knowledge into logic.

This author read one particular post asking if there existed a “recipe book of Boolean encodings”. This question got a lot of attention but no satisfactory answers. This HDip report hopes to address that enquiry by providing part of such a text. By being a “recipe book” this report aims to facilitate a concretization of the mathematical abstractions developed in more theoretical works (for instance Donald Knuth’s Fascicle 6 [1]). It contains practical recipes and step by step instructions for translating at least two problems into Boolean logic. The entire Boolean encodings themselves which can go on for several pages are provided. Also included, is executable code to get the software engineer up and running with SAT Solvers in an afternoon.

1.1 What is the Boolean Satisfiability Problem?

Given a Boolean formula $F(x_1, x_2, \dots, x_n)$ can we find an assignment of 1's and 0's for its variables such that F evaluates to 1.

1.1.1 Example 1

For instance consider:

$$F = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$$

If we compute out the truth table for this particular F :

x1	x2	x3	$(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	0

Then we can see that we can “satisfy” this particular F with if we assign $(x_1 = 0, x_2 = 0, x_3 = 1)$

1.2 What is a SAT Solver?

In this text we will treat a SAT Solver as a magical black-box. This black-box takes in a Boolean Formula like F above. This formula has to be in conjunctive normal form (CNF). That is as an AND of ORs. The black box also requires a list of all Boolean variables in the formula. Finally after a short duration of time the solver should output a list of assignments corresponding to the list of variables.

input (Mathematica Code)

```
booleanFormula = (x1 ∨ ¬ x2) ∧ (x2 ∨ x3) ∧ (¬ x1 ∨ ¬ x3) ∧ (¬ x1 ∨ ¬ x2 ∨ x3);  
listOfBooleanVariables = {x1, x2, x3};  
  
satSolver[listOfBooleanVariables, booleanFormula]
```

```
{0, 0, 1}
```

output

Chapter 2

Game of Life

“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.”

— Donald Knuth on John Conway and his Game of Life

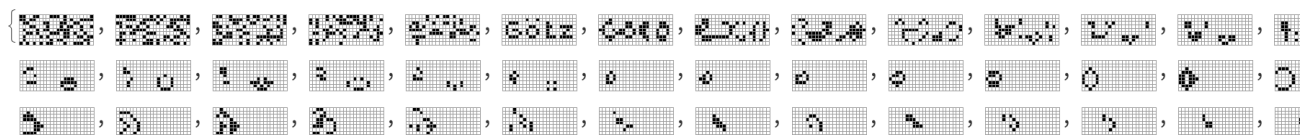


FIGURE 2.1: In this chapter we will use a SAT solver to discover the initial state which evolves into the word “Götz” after 5 successive transitions

2.1 Conway’s Game of Life

Conway’s **Game of Life** (a.k.a. GOL or Life) is a two dimensional cellular automata that is both interesting for its own sake as a mathematical object and because it sheds light on the nature of computation.

The game consists of a sequence of states. These states are grids of black and white square cells. Conventionally black cells are said to be alive and white cells dead.

To play the game one must define an initial state X_0 . This state then evolves into a “history” or set of subsequent states according to a handful of simple rules.

British mathematician John Conway developed Life in 1970 using a Go Board and pebbles after 2 years worth of experimentation during coffee breaks. He chose the rules so that the game would be simple to play but yet exhibit rich emergent behaviour.

The game has been discovered to be able to simulate a universal Turing machine and is hence undecidable by reduction from the halting problem.

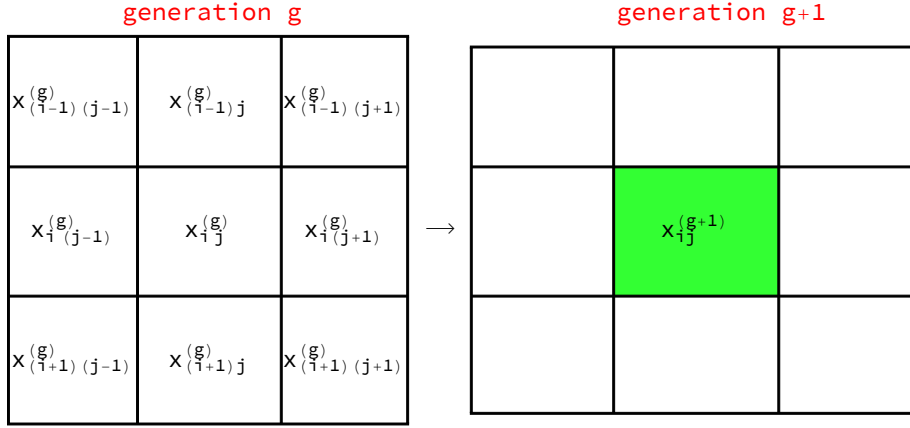
Even in 2020 the game is still being actively explored. An online community studies it as an object of pure mathematical enquiry. In 2013 the first true Von Neumann replicator was discovered in the game by Dave Green. This was a 219 trillion cell “lifeform” that can construct a replica of itself using a DNA like “instruction tape”. In 2018 a “lifeform” called an “elementary knightship” was discovered that moves around the grid universe like a knight in chess.

In Fascicle 6 Donald Knuth employs it to illustrate a practical application of SAT Solvers called Bounded Model Checking (BMC). Bounded Model Checking is a method to screen for bugs in mission critical computer programs and hardware. The basic idea of Bounded Model checking is to analyze sequences of state transitions. A computer program may tour a set of states during its normal operation. In BMC an initial state and an error state is defined and then a SAT solver is employed to check whether or not it is possible for the program to transition from the initial state to the error state. Essentially we want to make it impossible for your Nuclear Missile Guidance system or your Neuralink to be able to transition to undesirable states.

2.2 The Four Rules of GOL

In our description of the four rules of Game of Life (below) we will use the following notational conventions:

- The symbol $x_{ij}^{(g)}$ denotes that the cell in row i and column j in generation g is alive.
- The same symbol with an overline $\left(\overline{x_{ij}^{(g)}}\right)$ represents its negation (i.e. that that cell is dead).



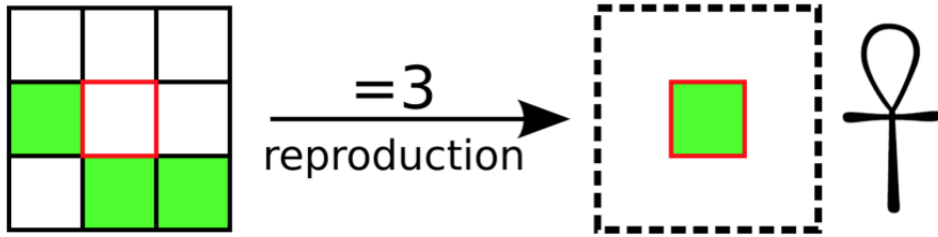
The game of life evolves from generation g to generation $g + 1$ according to the following four transition rules:

2.2.1 Rule:1 Reproduction

2.2.1.1 Natural Language Description:

If cell $x_{ij}^{(g)}$ is dead and it has exactly 3 alive neighbour cells then it will be alive in the next generation ($g + 1$)

2.2.1.2 Diagrammatic Description:



2.2.1.3 First Order Logic Description:

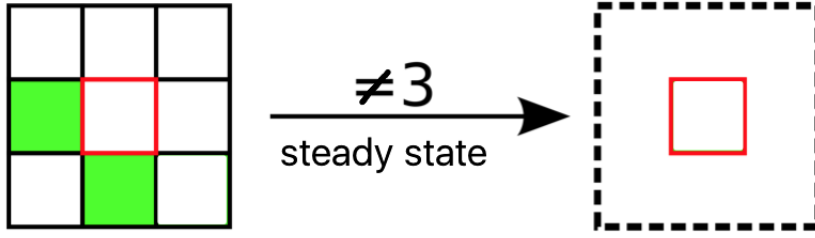
$$\overline{x_{i,j}^{(g)}} \wedge \text{aliveNeighbours}(x_{i,j}^{(g)}, 3) \Rightarrow x_{i,j}^{(g+1)}$$

2.2.2 Rule:2 Steady State

2.2.2.1 Natural Language Description:

If cell $x_{ij}^{(g)}$ is dead and it does **not** have exactly 3 alive neighbour cells then it will remain dead in the next generation ($g + 1$)

2.2.2.2 Diagrammatic Description:



2.2.2.3 First Order Logic Description:

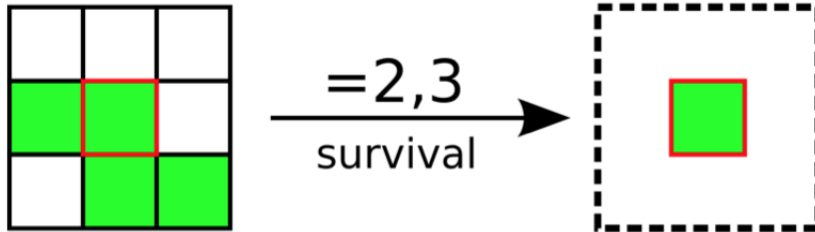
$$\overline{x_{i,j}^{(g)}} \wedge \overline{\text{aliveNeighbours}(x_{i,j}^{(g)}, 3)} \Rightarrow \overline{x_{i,j}^{(g+1)}}$$

2.2.3 Rule:3 Survival

2.2.3.1 Natural Language Description:

If cell $x_{ij}^{(g)}$ is alive and it has 2 or 3 live neighbour cells then it will remain alive in the next generation ($g + 1$)

2.2.3.2 Diagrammatic Description:



2.2.3.3 First Order Logic Description:

$$x_{i,j}^{(g)} \wedge \left\{ \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 2 \right) \vee \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 3 \right) \right\} \Rightarrow x_{i,j}^{(g+1)}$$

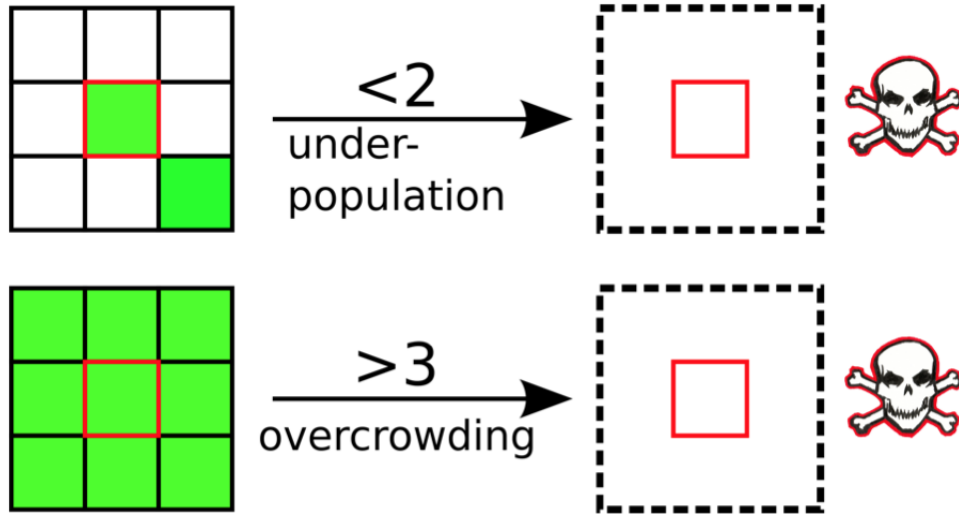
Where predicate $\text{aliveNeighbours} \left(x_{i,j}^{(g)}, 3 \right)$ returns 1 (True) if cell $x_{i,j}^{(g)}$ has 3 alive neighbour cells. Similarly the predicate $\text{aliveNeighbours} \left(x_{i,j}^{(g)}, 2 \right)$ returns True if cell $x_{i,j}^{(g)}$ has 2 alive neighbour cells and False otherwise.

2.2.4 Rule:4 Underpopulation or Overcrowding

2.2.4.1 Natural Language Description:

If cell $x_{ij}^{(g)}$ is alive and it does **not** have 2 or 3 live neighbour cells then it will be dead in the next generation $(g + 1)$

2.2.4.2 Diagrammatic Description:



2.2.4.3 First Order Logic Description:

$$x_{i,j}^{(g)} \wedge \overline{\left\{ \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 2 \right) \vee \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 3 \right) \right\}} \Rightarrow \overline{x_{i,j}^{(g+1)}}$$

We can apply De Morgan's law.

$$x_{i,j}^{(g)} \wedge \left\{ \overline{\text{aliveNeighbours} \left(x_{i,j}^{(g)}, 2 \right)} \wedge \overline{\text{aliveNeighbours} \left(x_{i,j}^{(g)}, 3 \right)} \right\} \Rightarrow \overline{x_{i,j}^{(g+1)}}$$

2.3 Encoding Rule 1 (Reproduction) into Boolean Logic

$$\overline{x_{i,j}^{(g)}} \wedge \text{aliveNeighbours}(x_{i,j}^{(g)}, 3) \Rightarrow x_{i,j}^{(g+1)} \quad (2.1)$$

In this section we shall derive the Boolean encoding for Conway's Game of Life from first principles (i.e. using the description of the rules in the last section as a starting point).

In the last section we translated the natural language description into the FOL formula above. Here we shall unpack the FOL formula and compile it down to Boolean logic.

2.3.1 Unpacking the Predicate

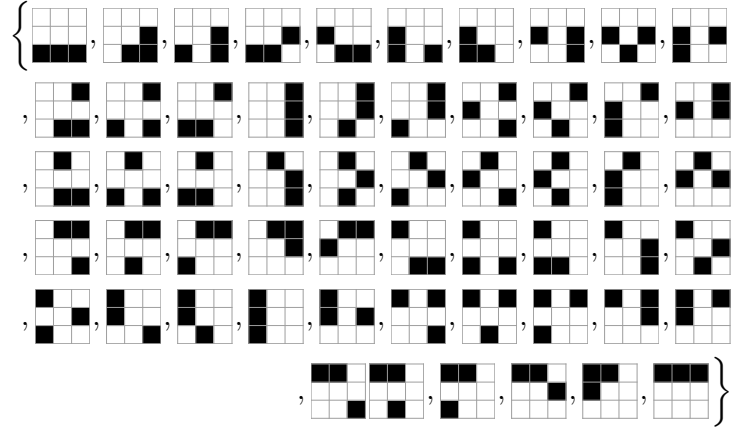
$$\text{aliveNeighbours}(x_{i,j}^{(g)}, 3) \quad (2.2)$$

The predicate $\text{aliveNeighbours}(x_{i,j}^{(g)}, 3)$ returns 1 if cell $x_{i,j}^{(g)}$ has precisely 3 live neighbours. Otherwise it returns 0.

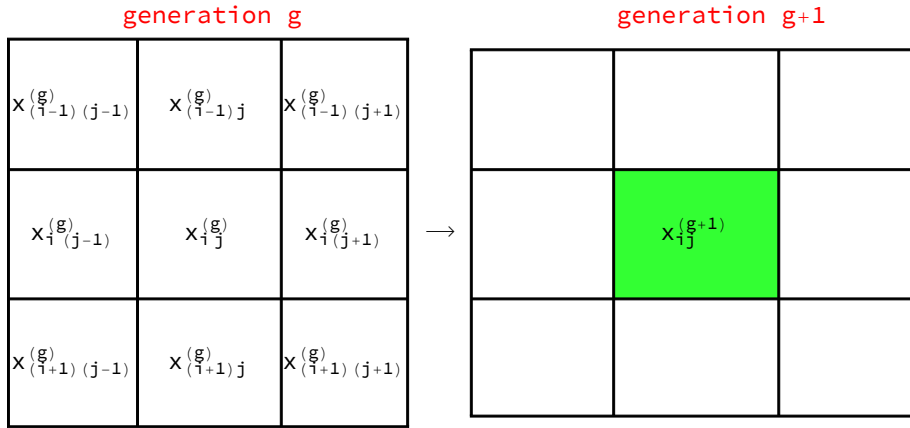
In the context of SAT Solvers, Boolean Logic (or Zeroth Order Logic) can be thought of as a lower level *declarative* programming language relative to First Order Logic (which can be thought of as a higher level declarative programming language). This is because we can *compile* First Order Logic down to Boolean Logic. The SAT Solver itself can be thought of as an interpreter (with which one can execute Boolean Logic formulae). First Order Logic is more abstract and thus more compact. Complexity is abstracted away (from Boolean Logic) by introducing *quantifiers* and *predicates*.

To represent this predicate in Boolean Logic we need to collect all the cases that this predicate describes.

There are $\binom{8}{3}$ different possible ways for a cell to have precisely 3 alive neighbour cells:



These neighbourhood cases can be intuitively described in Disjunctive Normal Form (DNF). One can then use some standard propositional equivalences to convert the resultant DNF formula into Conjunctive Normal Form (CNF). This needs to be done as SAT Solvers require their input to be preprocessed into CNF.



$$\begin{aligned}
1. \quad \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \\
2. \quad \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \\
3. \quad \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \\
&\vdots \\
56. \quad \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} &\equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right)
\end{aligned} \tag{2.3}$$

Full result in Appendix A. See [4.1](#).

We need to disjoin these 56 conjunctions into a disjunction and then *rewrite* the predicate in result (2.1) as this disjunction.

$$\begin{aligned}
&\overline{x_{i,j}^{(g)}} \wedge \text{aliveNeighbours}(x_{i,j}^{(g)}, 3) \Rightarrow x_{i,j}^{(g+1)} \\
&\overline{x_{i,j}^{(g)}} \wedge \left(\left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \right. \\
&\quad \vee \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \\
&\quad \vee \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \\
&\quad \vdots \\
&\quad \left. \vee \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \right) \Rightarrow x_{i,j}^{(g+1)}
\end{aligned} \tag{2.4}$$

Full result in Appendix A. See 4.2.

Then we need to convert this combined expression into CNF using propositional equivalences. This task is somewhat arduous as there are 1522 terms in this formula. The author will operate on an equivalent but much smaller formula below to demonstrate how this can be accomplished.

Let us represent 56 conjunctions with 2 conjunctions (of 3 Boolean variables each):

$$\mathbf{conjunction}_1 \equiv (a \wedge b \wedge c)$$

$$\mathbf{conjunction}_2 \equiv (d \wedge e \wedge f)$$

$$\begin{aligned}
& \overline{x_{i,j}^{(g)}} \wedge \text{aliveNeighbours}(, 3) \Rightarrow x_{i,j}^{(g+1)} \\
& \overline{x_{i,j}^{(g)}} \wedge (\mathbf{conjunction}_1 \vee \mathbf{conjunction}_2) \Rightarrow x_{i,j}^{(g+1)} \\
& \overline{x_{i,j}^{(g)}} \wedge ((a \wedge b \wedge c) \vee (d \wedge e \wedge f)) \Rightarrow x_{i,j}^{(g+1)} \\
& \left(\left(\overline{x_{i,j}^{(g)}} \wedge a \wedge b \wedge c \right) \vee \left(\overline{x_{i,j}^{(g)}} \wedge d \wedge e \wedge f \right) \right) \Rightarrow x_{i,j}^{(g+1)} \quad \text{distributing across } \overline{x_{i,j}^{(g)}} \wedge \\
& \overline{\left(\left(\overline{x_{i,j}^{(g)}} \wedge a \wedge b \wedge c \right) \vee \left(\overline{x_{i,j}^{(g)}} \wedge d \wedge e \wedge f \right) \right)} \vee x_{i,j}^{(g+1)} \quad p \Rightarrow q \equiv \bar{p} \vee q \\
& \left(\overline{\left(\overline{x_{i,j}^{(g)}} \wedge a \wedge b \wedge c \right)} \wedge \overline{\left(\overline{x_{i,j}^{(g)}} \wedge d \wedge e \wedge f \right)} \right) \vee x_{i,j}^{(g+1)} \quad \text{De Morgan's laws} \\
& \left(\left(\overline{\overline{x_{i,j}^{(g)}}} \vee \bar{a} \vee \bar{b} \vee \bar{c} \right) \wedge \left(\overline{\overline{x_{i,j}^{(g)}}} \vee \bar{d} \vee \bar{e} \vee \bar{f} \right) \right) \vee x_{i,j}^{(g+1)} \quad \text{De Morgan's laws} \\
& \left(\left(\overline{\overline{x_{i,j}^{(g)}}} \vee \bar{a} \vee \bar{b} \vee \bar{c} \vee x_{i,j}^{(g+1)} \right) \wedge \left(\overline{\overline{x_{i,j}^{(g)}}} \vee \bar{d} \vee \bar{e} \vee \bar{f} \vee x_{i,j}^{(g+1)} \right) \right) \quad \text{distributing across } \vee x_{i,j}^{(g+1)} \\
& \left(\left(x_{i,j}^{(g)} \vee \bar{a} \vee \bar{b} \vee \bar{c} \vee x_{i,j}^{(g+1)} \right) \wedge \left(x_{i,j}^{(g)} \vee \bar{d} \vee \bar{e} \vee \bar{f} \vee x_{i,j}^{(g+1)} \right) \right) \quad \text{double negation of } x_{i,j}^{(g)} \\
& \left\{ \left(x_{i,j}^{(g)} \vee \bar{a} \vee \bar{b} \vee \bar{c} \vee x_{i,j}^{(g+1)} \right), \left(x_{i,j}^{(g)} \vee \bar{d} \vee \bar{e} \vee \bar{f} \vee x_{i,j}^{(g+1)} \right) \right\} \quad \text{clausal form 1} \\
& \left\{ \left\{ x_{i,j}^{(g)}, \bar{a}, \bar{b}, \bar{c}, x_{i,j}^{(g+1)} \right\}, \left\{ x_{i,j}^{(g)}, \bar{d}, \bar{e}, \bar{f}, x_{i,j}^{(g+1)} \right\} \right\} \quad \text{clausal form 2} \\
& \left\{ x_{i,j}^{(g)} \bar{a} \bar{b} \bar{c} x_{i,j}^{(g+1)}, x_{i,j}^{(g)} \bar{d} \bar{e} \bar{f} x_{i,j}^{(g+1)} \right\} \quad \text{clausal form 3}
\end{aligned}$$

It is easy to see now how to convert the 1522 term formula into CNF. We just need to negate the 1500 gray and white terms, turn the conjunctions into clauses (disjunctions) and include the two blue terms in each clause.

Clausal form 2 is easiest to represent in Mathematica code (and Python 3 code) so we shall employ it here.

Here is result (2.1) fully encoded into Boolean CNF:

$$\begin{aligned}
 & \left\{ \left\{ x_{i,j}^{(g)}, x_{i-1,j-1}^{(g)}, x_{i-1,j}^{(g)}, x_{i-1,j+1}^{(g)}, x_{i,j-1}^{(g)}, x_{i,j+1}^{(g)}, \overline{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, x_{i,j}^{(g+1)} \right\} \right. \\
 & , \left\{ x_{i,j}^{(g)}, x_{i-1,j-1}^{(g)}, x_{i-1,j}^{(g)}, x_{i-1,j+1}^{(g)}, x_{i,j-1}^{(g)}, \overline{x_{i,j+1}^{(g)}}, \overline{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, x_{i,j}^{(g+1)} \right\} \\
 & , \left\{ x_{i,j}^{(g)}, x_{i-1,j-1}^{(g)}, x_{i-1,j}^{(g)}, x_{i-1,j+1}^{(g)}, x_{i,j-1}^{(g)}, \overline{x_{i,j+1}^{(g)}}, \overline{x_{i+1,j-1}^{(g)}}, x_{i+1,j}^{(g)}, \overline{x_{i+1,j+1}^{(g)}}, x_{i,j}^{(g+1)} \right\} \\
 & \quad \vdots \\
 & , \left\{ x_{i,j}^{(g)}, \overline{x_{i-1,j-1}^{(g)}}, \overline{x_{i-1,j}^{(g)}}, \overline{x_{i-1,j+1}^{(g)}}, x_{i,j-1}^{(g)}, x_{i,j+1}^{(g)}, \overline{x_{i+1,j-1}^{(g)}}, x_{i+1,j}^{(g)}, \overline{x_{i+1,j+1}^{(g)}}, x_{i,j}^{(g+1)} \right\} \\
 & \left. \right\} \\
 & \tag{2.5}
 \end{aligned}$$

Full result in Appendix A. See [4.3](#).

2.4 Encoding Rule 2 (Steady State) into Boolean Logic

$$\overline{x_{i,j}^{(g)}} \wedge \overline{aliveNeighbours(x_{i,j}^{(g)}, 3)} \Rightarrow \overline{x_{i,j}^{(g+1)}} \quad (2.6)$$

2.4.1 Unpacking the negated Predicate

$$\overline{aliveNeighbours(x_{i,j}^{(g)}, 3)} \quad (2.7)$$

This negated predicate $\overline{aliveNeighbours(x_{i,j}^{(g)}, 3)}$ returns 0 if cell $x_{i,j}^{(g)}$ has precisely 3 live neighbours. Otherwise it returns 1.

To represent this predicate in Boolean Logic we need to collect all the cases that this negated predicate describes.

There are $\binom{8}{0} + \binom{8}{1} + \binom{8}{2} + \binom{8}{4} + \binom{8}{5} + \binom{8}{6} + \binom{8}{7} + \binom{8}{8}$ different possible ways for a cell to **not** have 3 alive neighbour cells:

Replicating the treatment done in [section 2.3.1](#) we will describe these neighbourhood cases in Disjunctive Normal Form (DNF).

$$\begin{aligned}
1. \begin{array}{|c|c|} \hline \square & \square \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
2. \begin{array}{|c|c|} \hline \square & \blacksquare \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
3. \begin{array}{|c|c|} \hline \blacksquare & \square \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
&\vdots \\
200. \begin{array}{|c|c|} \hline \blacksquare & \blacksquare \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
&\hspace{15em} (2.8)
\end{aligned}$$

Full result in Appendix A. See 4.4.

Given the 200 conjunctions above one could easily join them together with ORs (as we did in [Section: 2.3 \(Encoding Rule 1 ... \)](#)) to make a large formula that is equivalent to the predicate. This resulting expression would be in Disjunctive Normal Form (DNF) (i.e. an OR of ANDs).

$$\overline{x_{i,j}^{(g)}} \wedge \overline{aliveNeighbours(x_{i,j}^{(g)}, 3)} \Rightarrow \overline{x_{i,j}^{(g+1)}}$$

$$\begin{aligned}
& \overline{x_{i,j}^{(g)}} \wedge \left(\left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \right. \\
& \quad \vee \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
& \quad \vee \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
& \quad \vdots \\
& \quad \left. \vee \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \right) \Rightarrow \overline{x_{i,j}^{(g+1)}}
\end{aligned} \tag{2.9}$$

In [Section: 2.3 \(Encoding Rule 1 ... \)](#) we showed (by inserting an equivalent but smaller DNF formula into Rule 1) how we could then transform Rule 1 into CNF. Here we will employ the same technique to transform Rule 2 into CNF. Recall that Rule 2 is $\overline{x_{i,j}^{(g)}} \wedge \overline{aliveNeighbours(x_{i,j}^{(g)}, 3)} \Rightarrow \overline{x_{i,j}^{(g+1)}}$.

Let us represent the 200 conjunctions we found above with 2 conjunctions (of 3 Boolean variables each):

$$\mathbf{conjunction_1} \equiv (a \wedge b \wedge c)$$

$$\mathbf{conjunction_2} \equiv (d \wedge e \wedge f)$$

$$\begin{aligned}
& \overline{x_{i,j}^{(g)}} \wedge \overline{\text{aliveNeighbours}(x_{i,j}^{(g)}, 3)} \Rightarrow \overline{x_{i,j}^{(g+1)}} \\
& \overline{x_{i,j}^{(g)}} \wedge \overline{\text{somePredicate}(x_{i,j}^{(g)})} \Rightarrow \overline{x_{i,j}^{(g+1)}} \\
& \overline{x_{i,j}^{(g)}} \wedge \overline{\text{massiveFormulaInDNF}} \Rightarrow \overline{x_{i,j}^{(g+1)}} \\
& \overline{x_{i,j}^{(g)}} \wedge (\mathbf{conjunction_1} \vee \mathbf{conjunction_2}) \Rightarrow \overline{x_{i,j}^{(g+1)}} \\
& \overline{x_{i,j}^{(g)}} \wedge ((a \wedge b \wedge c) \vee (d \wedge e \wedge f)) \Rightarrow \overline{x_{i,j}^{(g+1)}} \\
& \left(\left(\overline{x_{i,j}^{(g)}} \wedge a \wedge b \wedge c \right) \vee \left(\overline{x_{i,j}^{(g)}} \wedge d \wedge e \wedge f \right) \right) \Rightarrow \overline{x_{i,j}^{(g+1)}} \quad \text{distributing across } \overline{x_{i,j}^{(g)}} \wedge \\
& \overline{\left(\left(\overline{x_{i,j}^{(g)}} \wedge a \wedge b \wedge c \right) \vee \left(\overline{x_{i,j}^{(g)}} \wedge d \wedge e \wedge f \right) \right)} \vee \overline{x_{i,j}^{(g+1)}} \quad p \Rightarrow q \equiv \overline{p} \vee q \\
& \left(\overline{\left(\overline{x_{i,j}^{(g)}} \wedge a \wedge b \wedge c \right)} \wedge \overline{\left(\overline{x_{i,j}^{(g)}} \wedge d \wedge e \wedge f \right)} \right) \vee \overline{x_{i,j}^{(g+1)}} \quad \text{De Morgan's laws} \\
& \left(\left(\overline{x_{i,j}^{(g)}} \vee \overline{a} \vee \overline{b} \vee \overline{c} \right) \wedge \left(\overline{x_{i,j}^{(g)}} \vee \overline{d} \vee \overline{e} \vee \overline{f} \right) \right) \vee \overline{x_{i,j}^{(g+1)}} \quad \text{De Morgan's laws} \\
& \left(\left(\overline{x_{i,j}^{(g)}} \vee \overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{x_{i,j}^{(g+1)}} \right) \wedge \left(\overline{x_{i,j}^{(g)}} \vee \overline{d} \vee \overline{e} \vee \overline{f} \vee \overline{x_{i,j}^{(g+1)}} \right) \right) \quad \text{distributing across } \vee \overline{x_{i,j}^{(g+1)}} \\
& \left(\left(\overline{x_{i,j}^{(g)}} \vee \overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{x_{i,j}^{(g+1)}} \right) \wedge \left(\overline{x_{i,j}^{(g)}} \vee \overline{d} \vee \overline{e} \vee \overline{f} \vee \overline{x_{i,j}^{(g+1)}} \right) \right) \quad \text{double negation of } \overline{x_{i,j}^{(g)}} \\
& \left\{ \left(\overline{x_{i,j}^{(g)}} \vee \overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{x_{i,j}^{(g+1)}} \right), \left(\overline{x_{i,j}^{(g)}} \vee \overline{d} \vee \overline{e} \vee \overline{f} \vee \overline{x_{i,j}^{(g+1)}} \right) \right\} \quad \text{clausal form 1} \\
& \left\{ \left\{ \overline{x_{i,j}^{(g)}}, \overline{a}, \overline{b}, \overline{c}, \overline{x_{i,j}^{(g+1)}} \right\}, \left\{ \overline{x_{i,j}^{(g)}}, \overline{d}, \overline{e}, \overline{f}, \overline{x_{i,j}^{(g+1)}} \right\} \right\} \quad \text{clausal form 2}
\end{aligned}$$

It is easy to see now how to convert Rule 2 into CNF. We just need to negate the all the terms we discovered in the 200 conjunctions above, turn the conjunctions into clauses (disjunctions) (by replacing the ANDs with ORs) and include the two new blue terms in each clause. The first blue term ($\overline{x_{i,j}^{(g)}}$) is negated.

Here is result (2.3) fully encoded into Boolean CNF:

$$\left\{ \begin{aligned}
& \left\{ x_{i,j}^{(g)}, \boxed{x_{i-1,j-1}^{(g)}}, \boxed{x_{i-1,j}^{(g)}}, \boxed{x_{i-1,j+1}^{(g)}}, \boxed{x_{i,j-1}^{(g)}}, \boxed{x_{i,j+1}^{(g)}}, \boxed{x_{i+1,j-1}^{(g)}}, \boxed{x_{i+1,j}^{(g)}}, \boxed{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\}, \\
& \left\{ x_{i,j}^{(g)}, \boxed{x_{i-1,j-1}^{(g)}}, \boxed{x_{i-1,j}^{(g)}}, \boxed{x_{i-1,j+1}^{(g)}}, \boxed{x_{i,j-1}^{(g)}}, \boxed{x_{i,j+1}^{(g)}}, \boxed{x_{i+1,j-1}^{(g)}}, \boxed{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\}, \\
& \left\{ x_{i,j}^{(g)}, \boxed{x_{i-1,j-1}^{(g)}}, \boxed{x_{i-1,j}^{(g)}}, \boxed{x_{i-1,j+1}^{(g)}}, \boxed{x_{i,j-1}^{(g)}}, \boxed{x_{i,j+1}^{(g)}}, \boxed{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, \boxed{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\}, \\
& \vdots \\
& \left\{ x_{i,j}^{(g)}, \overline{x_{i-1,j-1}^{(g)}}, \overline{x_{i-1,j}^{(g)}}, \overline{x_{i-1,j+1}^{(g)}}, \overline{x_{i,j-1}^{(g)}}, \overline{x_{i,j+1}^{(g)}}, \overline{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\}
\end{aligned} \right\} \tag{2.10}$$

Full result in Appendix A. See [4.5](#).

$$\begin{aligned}
1. \begin{array}{|c|c|} \hline \blacksquare & \blacksquare \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \\
2. \begin{array}{|c|c|} \hline \blacksquare & \square \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \\
3. \begin{array}{|c|c|} \hline \square & \blacksquare \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
&\vdots \\
84. \begin{array}{|c|c|} \hline \square & \square \\ \hline \end{array} &\equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right)
\end{aligned} \tag{2.13}$$

Full result in Appendix A. See 4.6.

Given the 84 conjunctions above one could easily join them together with ORs (as we did in [Section: 2.3. Encoding Rule 1](#)) to make a large formula that is equivalent to the predicate. This resulting expression is in Disjunctive Normal Form (DNF) (i.e. an OR of ANDs).

In [Section:2.3. Encoding Rule 1](#)) we showed (by inserting an equivalent but smaller DNF formula into Rule 1) how we could then transform Rule 1 into CNF. Here we will employ the same technique to transform Rule 3 into CNF. Recall that Rule 3 is $x_{i,j}^{(g)} \wedge \left(\text{aliveNeighbours}(x_{i,j}^{(g)}, 2) \vee \text{aliveNeighbours}(x_{i,j}^{(g)}, 3) \right) \Rightarrow x_{i,j}^{(g+1)}$.

Let us represent the 84 conjunctions we found above with 2 conjunctions (of 3 Boolean variables each):

$$\mathbf{conjunction_1} \equiv (a \wedge b \wedge c)$$

$$\mathbf{conjunction_2} \equiv (d \wedge e \wedge f)$$

$$\begin{aligned}
& x_{i,j}^{(g)} \wedge \left(\text{aliveNeighbours} \left(x_{i,j}^{(g)}, 2 \right) \vee \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 3 \right) \right) \Rightarrow x_{i,j}^{(g+1)} \\
& x_{i,j}^{(g)} \wedge \text{somePredicate} \left(x_{i,j}^{(g)} \right) \Rightarrow x_{i,j}^{(g+1)} \\
& x_{i,j}^{(g)} \wedge \text{massiveFormulaInDNF} \Rightarrow x_{i,j}^{(g+1)} \\
& x_{i,j}^{(g)} \wedge (\mathbf{conjunction}_1 \vee \mathbf{conjunction}_2) \Rightarrow x_{i,j}^{(g+1)} \\
& x_{i,j}^{(g)} \wedge ((a \wedge b \wedge c) \vee (d \wedge e \wedge f)) \Rightarrow x_{i,j}^{(g+1)} \\
& \left(\left(x_{i,j}^{(g)} \wedge a \wedge b \wedge c \right) \vee \left(x_{i,j}^{(g)} \wedge d \wedge e \wedge f \right) \right) \Rightarrow x_{i,j}^{(g+1)} \quad \text{distributing across } x_{i,j}^{(g)} \wedge \\
& \overline{\left(\left(x_{i,j}^{(g)} \wedge a \wedge b \wedge c \right) \vee \left(x_{i,j}^{(g)} \wedge d \wedge e \wedge f \right) \right)} \vee x_{i,j}^{(g+1)} \quad p \Rightarrow q \equiv \bar{p} \vee q \\
& \left(\overline{\left(x_{i,j}^{(g)} \wedge a \wedge b \wedge c \right)} \wedge \overline{\left(x_{i,j}^{(g)} \wedge d \wedge e \wedge f \right)} \right) \vee x_{i,j}^{(g+1)} \quad \text{De Morgan's laws} \\
& \left(\left(\overline{x_{i,j}^{(g)}} \vee \bar{a} \vee \bar{b} \vee \bar{c} \right) \wedge \left(\overline{x_{i,j}^{(g)}} \vee \bar{d} \vee \bar{e} \vee \bar{f} \right) \right) \vee x_{i,j}^{(g+1)} \quad \text{De Morgan's laws} \\
& \left(\left(\overline{x_{i,j}^{(g)}} \vee \bar{a} \vee \bar{b} \vee \bar{c} \vee x_{i,j}^{(g+1)} \right) \wedge \left(\overline{x_{i,j}^{(g)}} \vee \bar{d} \vee \bar{e} \vee \bar{f} \vee x_{i,j}^{(g+1)} \right) \right) \quad \text{distributing across } \vee x_{i,j}^{(g+1)} \\
& \left(\left(\overline{x_{i,j}^{(g)}} \vee \bar{a} \vee \bar{b} \vee \bar{c} \vee x_{i,j}^{(g+1)} \right) \wedge \left(\overline{x_{i,j}^{(g)}} \vee \bar{d} \vee \bar{e} \vee \bar{f} \vee x_{i,j}^{(g+1)} \right) \right) \quad \text{negation of } x_{i,j}^{(g)} \\
& \left\{ \left(\overline{x_{i,j}^{(g)}} \vee \bar{a} \vee \bar{b} \vee \bar{c} \vee x_{i,j}^{(g+1)} \right), \left(\overline{x_{i,j}^{(g)}} \vee \bar{d} \vee \bar{e} \vee \bar{f} \vee x_{i,j}^{(g+1)} \right) \right\} \quad \text{clausal form 1} \\
& \left\{ \left\{ \overline{x_{i,j}^{(g)}}, \bar{a}, \bar{b}, \bar{c}, x_{i,j}^{(g+1)} \right\}, \left\{ \overline{x_{i,j}^{(g)}}, \bar{d}, \bar{e}, \bar{f}, x_{i,j}^{(g+1)} \right\} \right\} \quad \text{clausal form 2}
\end{aligned}$$

It is easy to see now how to convert Rule 3 into CNF. We just need to negate the all the terms we discovered in the 84 conjunctions above, turn the conjunctions into clauses (disjunctions) (by replacing the ANDs with ORs) and include the two new blue terms in each clause. The first blue term ($x_{i,j}^{(g)}$) is negated.

Here is result (2.5) fully encoded into Boolean CNF:

$$\left\{ \begin{aligned}
& \left\{ \overline{x_{i,j}^{(g)}}, x_{i-1,j-1}^{(g)}, x_{i-1,j}^{(g)}, x_{i-1,j+1}^{(g)}, x_{i,j-1}^{(g)}, x_{i,j+1}^{(g)}, x_{i+1,j-1}^{(g)}, \overline{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, x_{i,j}^{(g+1)} \right\}, \\
& \left\{ \overline{x_{i,j}^{(g)}}, x_{i-1,j-1}^{(g)}, x_{i-1,j}^{(g)}, x_{i-1,j+1}^{(g)}, x_{i,j-1}^{(g)}, x_{i,j+1}^{(g)}, \overline{x_{i+1,j-1}^{(g)}}, x_{i+1,j}^{(g)}, \overline{x_{i+1,j+1}^{(g)}}, x_{i,j}^{(g+1)} \right\}, \\
& \left\{ \overline{x_{i,j}^{(g)}}, x_{i-1,j-1}^{(g)}, x_{i-1,j}^{(g)}, x_{i-1,j+1}^{(g)}, x_{i,j-1}^{(g)}, x_{i,j+1}^{(g)}, \overline{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, x_{i+1,j+1}^{(g)}, x_{i,j}^{(g+1)} \right\}, \\
& \vdots \\
& \left\{ \overline{x_{i,j}^{(g)}}, \overline{x_{i-1,j-1}^{(g)}}, \overline{x_{i-1,j}^{(g)}}, \overline{x_{i-1,j+1}^{(g)}}, x_{i,j-1}^{(g)}, x_{i,j+1}^{(g)}, x_{i+1,j-1}^{(g)}, x_{i+1,j}^{(g)}, x_{i+1,j+1}^{(g)}, x_{i,j}^{(g+1)} \right\},
\end{aligned} \right\} \tag{2.14}$$

Full result in Appendix A. See 4.7.

2.6 Encoding Rule 4 (Underpopulation or Overcrowding) into Boolean Logic

$$x_{i,j}^{(g)} \wedge \overline{\left\{ \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 2 \right) \vee \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 3 \right) \right\}} \Rightarrow \overline{x_{i,j}^{(g+1)}} \quad (2.15)$$

2.6.1 Unpacking the predicates











$$\overline{\left(\text{aliveNeighbours} \left(x_{i,j}^{(g)}, 3 \right) \vee \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 2 \right) \right)} \quad (2.16)$$











The negated disjunction of 2 Predicates $\overline{\left(\text{aliveNeighbours} \left(x_{i,j}^{(g)}, 3 \right) \vee \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 2 \right) \right)}$ returns 0 if cell $x_{ij}^{(g)}$ has 2 or 3 live neighbours. Otherwise it returns 1.











To represent this predicate in Boolean Logic we need to collect all the cases that this expression describes.






There are $\binom{8}{0} + \binom{8}{1} + \binom{8}{4} + \binom{8}{5} + \binom{8}{6} + \binom{8}{7} + \binom{8}{8}$ different possible ways for a cell to **not** have (3 or 2) alive neighbour cells:

{

 ,  ,  ,  ,  ,  ,  ,  ,  , 

,  ,  ,  ,  ,  ,  ,  ,  ,  , 

,  ,  ,  ,  ,  ,  ,  ,  ,  , 

,  ,  ,  ,  ,  ,

Replicating the treatment done in [section 2.3.1](#) we will describe these neighbourhood cases in Disjunctive Normal Form (DNF).]

$$\begin{aligned}
1. \begin{array}{|c|c|} \hline \blacksquare & \blacksquare \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
2. \begin{array}{|c|c|} \hline \blacksquare & \blacksquare \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
3. \begin{array}{|c|c|} \hline \blacksquare & \blacksquare \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right) \\
&\vdots \\
172. \begin{array}{|c|c|} \hline \blacksquare & \blacksquare \\ \hline \end{array} &\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right)
\end{aligned} \tag{2.17}$$

Full result in Appendix A. See 4.8.

These neighbourhood cases can be intuitively described in Disjunctive Normal Form (DNF). One can then use some standard propositional equivalences to convert the resultant DNF formula into Conjunctive Normal Form (CNF). This needs to be done as SAT Solvers require their input to be preprocessed into CNF.

Given the 172 conjunctions above one could easily join them together with ORs (as we did in [Section: 2.3. Rule 1](#)) to make a large formula that is equivalent to the predicate. This resulting expression is in Disjunctive Normal Form (DNF) (i.e. an OR of ANDs).

In [Section: 2.3. Rule 1](#) we showed (by inserting an equivalent but smaller DNF formula into Rule 1) how we could then transform Rule 1 into CNF. Here we will employ the same technique to transform Rule 4 into CNF. Recall that Rule 4 is $\overline{x_{i,j}^{(g)}} \wedge \left(\text{aliveNeighbours}(x_{i,j}^{(g)}, 3) \vee \text{aliveNeighbours}(x_{i,j}^{(g)}, 2) \right) \Rightarrow \overline{x_{i,j}^{(g+1)}}$.

Let us represent the 172 conjunctions we found above with 2 conjunctions (of 3 Boolean variables each):

$$\text{conjunction}_1 \equiv (a \wedge b \wedge c)$$

$$\text{conjunction}_2 \equiv (d \wedge e \wedge f)$$

$$\begin{aligned}
x_{i,j}^{(g)} \wedge \overline{\left(\text{aliveNeighbours} \left(x_{i,j}^{(g)}, 3 \right) \vee \text{aliveNeighbours} \left(x_{i,j}^{(g)}, 2 \right) \right)} &\Rightarrow \overline{x_{i,j}^{(g+1)}} \\
x_{i,j}^{(g)} \wedge \text{somePredicate} \left(x_{i,j}^{(g)} \right) &\Rightarrow \overline{x_{i,j}^{(g+1)}} \\
x_{i,j}^{(g)} \wedge \text{massiveFormulaInDNF} &\Rightarrow \overline{x_{i,j}^{(g+1)}} \\
x_{i,j}^{(g)} \wedge (\mathbf{conjunction}_1 \vee \mathbf{conjunction}_2) &\Rightarrow \overline{x_{i,j}^{(g+1)}} \\
x_{i,j}^{(g)} \wedge ((a \wedge b \wedge c) \vee (d \wedge e \wedge f)) &\Rightarrow \overline{x_{i,j}^{(g+1)}} \\
\left(\left(x_{i,j}^{(g)} \wedge a \wedge b \wedge c \right) \vee \left(x_{i,j}^{(g)} \wedge d \wedge e \wedge f \right) \right) &\Rightarrow \overline{x_{i,j}^{(g+1)}} && \text{distributing across } x_{i,j}^{(g+1)} \wedge \\
\overline{\left(\left(x_{i,j}^{(g)} \wedge a \wedge b \wedge c \right) \vee \left(x_{i,j}^{(g)} \wedge d \wedge e \wedge f \right) \right)} \vee \overline{x_{i,j}^{(g+1)}} &&& p \Rightarrow q \equiv \bar{p} \vee q \\
\left(\overline{\left(x_{i,j}^{(g)} \wedge a \wedge b \wedge c \right)} \wedge \overline{\left(x_{i,j}^{(g)} \wedge d \wedge e \wedge f \right)} \right) \vee \overline{x_{i,j}^{(g+1)}} &&& \text{De Morgan's laws} \\
\left(\left(\overline{x_{i,j}^{(g)}} \vee \bar{a} \vee \bar{b} \vee \bar{c} \right) \wedge \left(\overline{x_{i,j}^{(g)}} \vee \bar{d} \vee \bar{e} \vee \bar{f} \right) \right) \vee \overline{x_{i,j}^{(g+1)}} &&& \text{De Morgan's laws} \\
\left(\left(\overline{x_{i,j}^{(g)}} \vee \bar{a} \vee \bar{b} \vee \bar{c} \vee \overline{x_{i,j}^{(g+1)}} \right) \wedge \left(\overline{x_{i,j}^{(g)}} \vee \bar{d} \vee \bar{e} \vee \bar{f} \vee \overline{x_{i,j}^{(g+1)}} \right) \right) &&& \text{distributing across } \vee \overline{x_{i,j}^{(g+1)}} \\
\left\{ \left(\overline{x_{i,j}^{(g)}} \vee \bar{a} \vee \bar{b} \vee \bar{c} \vee \overline{x_{i,j}^{(g+1)}} \right), \left(\overline{x_{i,j}^{(g)}} \vee \bar{d} \vee \bar{e} \vee \bar{f} \vee \overline{x_{i,j}^{(g+1)}} \right) \right\} &&& \text{clausal form 1} \\
\left\{ \left\{ \overline{x_{i,j}^{(g)}}, \bar{a}, \bar{b}, \bar{c}, \overline{x_{i,j}^{(g+1)}} \right\}, \left\{ \overline{x_{i,j}^{(g)}}, \bar{d}, \bar{e}, \bar{f}, \overline{x_{i,j}^{(g+1)}} \right\} \right\} &&& \text{clausal form 2}
\end{aligned}$$

It is easy to see now how to convert Rule 4 into CNF. We just need to negate the all the terms we discovered in the 172 conjunctions above, turn the conjunctions into clauses (disjunctions) (by replacing the ANDs with ORs) and include the two new blue terms in each clause. The second blue term ($x_{i,j}^{(g)}$) is negated.

Here is result (2.15) fully encoded into Boolean CNF:

$$\left\{ \begin{aligned}
 & \left\{ \overline{x_{i,j}^{(g)}}, x_{i-1,j-1}^{(g)}, x_{i-1,j}^{(g)}, x_{i-1,j+1}^{(g)}, x_{i,j-1}^{(g)}, x_{i,j+1}^{(g)}, x_{i+1,j-1}^{(g)}, x_{i+1,j}^{(g)}, x_{i+1,j+1}^{(g)}, \overline{x_{i,j}^{(g+1)}} \right\}, \\
 & \left\{ \overline{x_{i,j}^{(g)}}, x_{i-1,j-1}^{(g)}, x_{i-1,j}^{(g)}, x_{i-1,j+1}^{(g)}, x_{i,j-1}^{(g)}, x_{i,j+1}^{(g)}, x_{i+1,j-1}^{(g)}, x_{i+1,j}^{(g)}, \overline{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\}, \\
 & \left\{ \overline{x_{i,j}^{(g)}}, x_{i-1,j-1}^{(g)}, x_{i-1,j}^{(g)}, x_{i-1,j+1}^{(g)}, x_{i,j-1}^{(g)}, x_{i,j+1}^{(g)}, x_{i+1,j-1}^{(g)}, \overline{x_{i+1,j}^{(g)}}, x_{i+1,j+1}^{(g)}, \overline{x_{i,j}^{(g+1)}} \right\}, \\
 & \vdots \\
 & \left\{ \overline{x_{i,j}^{(g)}}, \overline{x_{i-1,j-1}^{(g)}}, \overline{x_{i-1,j}^{(g)}}, \overline{x_{i-1,j+1}^{(g)}}, \overline{x_{i,j-1}^{(g)}}, \overline{x_{i,j+1}^{(g)}}, \overline{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\}, \quad (2.18) \\
 & \left. \right\}
 \end{aligned} \right.$$

Full result in Appendix A. See [4.9](#).

2.12 Using the author's package

These above sketches are implemented in Mathematica code in the author's package `wordLife.m` which can be found in [Appendix B \(5.1\)](#) or downloaded at <https://github.com/ccosnett/wordLife>.

To use the package:

1. Download it from github
2. Install Mathematica (contact the author if you don't have a copy).
3. Open `DEMO1.nb`
4. Evaluate the cell by pressing Shift+Enter. Or click `Evaluation>Evaluate Notebook` in the menu bar.

input (mathematica code)

```

endState = 
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix};$$

```

```

endGeneration = 7;
Get[NotebookDirectory[] <> "wordLife.m"]

```

output

Chapter 3

Conclusion

In this recipe book, we encoded the rules of Conway's Game of Life into Boolean formulae (or sets of clauses). We showed how these formulae could be derived from first principles and generated programmatically. We then showed how these formulae could be solved using a SAT solver.

Finally we developed code which organized and visualized the output of the SAT solver.

In the recipe for the Game of Life we showed how to encode an end state into the clause set such that the SAT Solver returned assignments corresponding to an initial state that evolves into our end state. This is a seemingly whimsical example of a serious application of SAT solvers, called bounded model checking. An error state in a program can be specified and a SAT Solver can be employed to check whether or not different states of the program are able to transition to the error state.

In future work the author hopes to expand the number of recipes beyond one. He also hopes to translate all the Mathematica language code to Python 3 code.











Chapter 4

Appendix A

4.1

This result set is referenced as “4.1” in the text. Please generalise from this instance.

To go back to position in Chapter 2 please click [\(2.3\)](#).

1.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge \overline{x_{i,j+1}^{(g)}} \wedge x_{i+1,j-1}^{(g)} \wedge x_{i+1,j}^{(g)} \wedge x_{i+1,j+1}^{(g)} \right)$
2.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge x_{i,j+1}^{(g)} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge x_{i+1,j}^{(g)} \wedge x_{i+1,j+1}^{(g)} \right)$
3.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge x_{i,j+1}^{(g)} \wedge x_{i+1,j-1}^{(g)} \wedge \overline{x_{i+1,j}^{(g)}} \wedge x_{i+1,j+1}^{(g)} \right)$
4.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge \overline{x_{i,j-1}^{(g)}} \wedge x_{i,j+1}^{(g)} \wedge x_{i+1,j-1}^{(g)} \wedge x_{i+1,j}^{(g)} \wedge \overline{x_{i+1,j+1}^{(g)}} \right)$
5.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge x_{i,j-1}^{(g)} \wedge \overline{x_{i,j+1}^{(g)}} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge x_{i+1,j}^{(g)} \wedge x_{i+1,j+1}^{(g)} \right)$
6.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge x_{i,j-1}^{(g)} \wedge \overline{x_{i,j+1}^{(g)}} \wedge x_{i+1,j-1}^{(g)} \wedge x_{i+1,j}^{(g)} \wedge \overline{x_{i+1,j+1}^{(g)}} \right)$
7.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge x_{i,j-1}^{(g)} \wedge \overline{x_{i,j+1}^{(g)}} \wedge x_{i+1,j-1}^{(g)} \wedge \overline{x_{i+1,j}^{(g)}} \wedge x_{i+1,j+1}^{(g)} \right)$
8.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge x_{i,j-1}^{(g)} \wedge x_{i,j+1}^{(g)} \wedge \overline{x_{i+1,j-1}^{(g)}} \wedge \overline{x_{i+1,j}^{(g)}} \wedge x_{i+1,j+1}^{(g)} \right)$
9.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge x_{i,j-1}^{(g)} \wedge x_{i,j+1}^{(g)} \wedge x_{i+1,j-1}^{(g)} \wedge x_{i+1,j}^{(g)} \wedge \overline{x_{i+1,j+1}^{(g)}} \right)$
10.  $\equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \overline{x_{i-1,j}^{(g)}} \wedge \overline{x_{i-1,j+1}^{(g)}} \wedge x_{i,j-1}^{(g)} \wedge \overline{x_{i,j+1}^{(g)}} \wedge x_{i+1,j-1}^{(g)} \wedge \overline{x_{i+1,j}^{(g)}} \wedge \overline{x_{i+1,j+1}^{(g)}} \right)$

[illegible]

[illegible]

To go back to position in Chapter 2 please click [\(2.4\)](#).

[illegible]

[illegible]

To go back to position in Chapter 2 please click [\(2.5\)](#).

[illegible]

[illegible]

[illegible]

54. $\begin{smallmatrix} \blacksquare & \square \\ \square & \blacksquare \end{smallmatrix} \equiv \left(x_{i-1,j-1}^{\textcircled{S}} \wedge \boxed{x_{i-1,j}^{\textcircled{S}}} \wedge x_{i-1,j+1}^{\textcircled{S}} \wedge x_{i,j-1}^{\textcircled{S}} \wedge \boxed{x_{i,j+1}^{\textcircled{S}}} \wedge x_{i+1,j-1}^{\textcircled{S}} \wedge \boxed{x_{i+1,j}^{\textcircled{S}}} \wedge x_{i+1,j+1}^{\textcircled{S}} \right)$

[illegible]

[illegible]

[illegible]

[illegible]

$$188. \quad \blacksquare \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \overline{\boxed{x_{i,j+1}^{(g)}}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \overline{\boxed{x_{i+1,j+1}^{(g)}}} \right)$$

$$189. \begin{array}{|c|c|} \hline \blacksquare & \blacksquare \\ \hline \end{array} \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \overline{\boxed{x_{i+1,j-1}^{(g)}}} \wedge \overline{\boxed{x_{i+1,j}^{(g)}}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right)$$

$$190. \blacksquare \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \overline{\boxed{x_{i+1,j-1}^{(g)}}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \overline{\boxed{x_{i+1,j+1}^{(g)}}} \right)$$

$$191. \begin{array}{|c|c|} \hline \blacksquare & \blacksquare \\ \hline \square & \square \\ \hline \end{array} \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \overline{\boxed{x_{i+1,j}^{(g)}}} \wedge \overline{\boxed{x_{i+1,j+1}^{(g)}}} \right)$$

$$192. \blacksquare \equiv \left(\overline{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right)$$

$$193. \blacksquare \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \overline{\boxed{x_{i-1,j}^{(g)}}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right)$$

$$194. \blacksquare \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \overline{\boxed{x_{i-1,j+1}^{(g)}}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right)$$

195. $\begin{smallmatrix} \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare \end{smallmatrix} \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \overline{\boxed{x_{i,j-1}^{(g)}}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right)$

$$196. \begin{array}{|c|c|c|} \hline \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array} \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \overline{\boxed{x_{i,j+1}^{(g)}}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right)$$

$$197. \begin{array}{|c|} \hline \blacksquare \blacksquare \\ \hline \end{array} \equiv \left(\begin{array}{|c|} \hline x_{i-1,j-1}^{(g)} \\ \hline \end{array} \wedge \begin{array}{|c|} \hline x_{i-1,j}^{(g)} \\ \hline \end{array} \wedge \begin{array}{|c|} \hline x_{i-1,j+1}^{(g)} \\ \hline \end{array} \wedge \begin{array}{|c|} \hline x_{i,j-1}^{(g)} \\ \hline \end{array} \wedge \begin{array}{|c|} \hline x_{i,j+1}^{(g)} \\ \hline \end{array} \wedge \overline{\begin{array}{|c|} \hline x_{i+1,j-1}^{(g)} \\ \hline \end{array}} \wedge \begin{array}{|c|} \hline x_{i+1,j}^{(g)} \\ \hline \end{array} \wedge \begin{array}{|c|} \hline x_{i+1,j+1}^{(g)} \\ \hline \end{array} \right)$$

$$198. \blacksquare \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \overline{\boxed{x_{i+1,j}^{(g)}}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right)$$

199. $\blacksquare \equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \overline{\boxed{x_{i+1,j+1}^{(g)}}} \right)$

$$200. \blacksquare \equiv \left(x_{i-1,j-1}^{(g)} \wedge x_{i-1,j}^{(g)} \wedge x_{i-1,j+1}^{(g)} \wedge x_{i,j-1}^{(g)} \wedge x_{i,j+1}^{(g)} \wedge x_{i+1,j-1}^{(g)} \wedge x_{i+1,j}^{(g)} \wedge x_{i+1,j+1}^{(g)} \right)$$

4.5

To go back to position in Chapter 2 please click [\(2.10\)](#).

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

To go back to position in Chapter 2 please click [\(2.13\)](#).

[illegible]

[illegible]

[illegible]

To go back to position in Chapter 2 please click [\(2.14\)](#).

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

$$\begin{aligned}
169. \quad \blacksquare \blacksquare &\equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \overline{\boxed{x_{i+1,j-1}^{(g)}}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \\
170. \quad \blacksquare \blacksquare &\equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \overline{\boxed{x_{i+1,j}^{(g)}}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right) \\
171. \quad \blacksquare \blacksquare &\equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \overline{\boxed{x_{i+1,j+1}^{(g)}}} \right) \\
172. \quad \blacksquare \blacksquare &\equiv \left(\boxed{x_{i-1,j-1}^{(g)}} \wedge \boxed{x_{i-1,j}^{(g)}} \wedge \boxed{x_{i-1,j+1}^{(g)}} \wedge \boxed{x_{i,j-1}^{(g)}} \wedge \boxed{x_{i,j+1}^{(g)}} \wedge \boxed{x_{i+1,j-1}^{(g)}} \wedge \boxed{x_{i+1,j}^{(g)}} \wedge \boxed{x_{i+1,j+1}^{(g)}} \right)
\end{aligned}$$

To go back to position in Chapter 2 please click [\(2.18\)](#).

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

$$\left. \begin{aligned}
& \left\{ \overline{x_{i,j}^{(g)}}, \overline{x_{i-1,j-1}^{(g)}}, \overline{x_{i-1,j}^{(g)}}, \overline{x_{i-1,j+1}^{(g)}}, \overline{x_{i,j-1}^{(g)}}, \overline{x_{i,j+1}^{(g)}}, \overline{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\}, \\
& \left\{ \overline{x_{i,j}^{(g)}}, \overline{x_{i-1,j-1}^{(g)}}, \overline{x_{i-1,j}^{(g)}}, \overline{x_{i-1,j+1}^{(g)}}, \overline{x_{i,j-1}^{(g)}}, \overline{x_{i,j+1}^{(g)}}, \overline{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\}, \\
& \left\{ \overline{x_{i,j}^{(g)}}, \overline{x_{i-1,j-1}^{(g)}}, \overline{x_{i-1,j}^{(g)}}, \overline{x_{i-1,j+1}^{(g)}}, \overline{x_{i,j-1}^{(g)}}, \overline{x_{i,j+1}^{(g)}}, \overline{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\}, \\
& \left\{ \overline{x_{i,j}^{(g)}}, \overline{x_{i-1,j-1}^{(g)}}, \overline{x_{i-1,j}^{(g)}}, \overline{x_{i-1,j+1}^{(g)}}, \overline{x_{i,j-1}^{(g)}}, \overline{x_{i,j+1}^{(g)}}, \overline{x_{i+1,j-1}^{(g)}}, \overline{x_{i+1,j}^{(g)}}, \overline{x_{i+1,j+1}^{(g)}}, \overline{x_{i,j}^{(g+1)}} \right\},
\end{aligned} \right\}$$

4.10

To go back to position in Chapter 2 please click (??).

Chapter 5

Appendix B

5.1 Author's package `wordLife.m`

This package can be downloaded from <https://github.com/ccosnett/wordLife>. To go back to position in Chapter 2 please click [2.12](#).

```
(* wordLife.m *)

Clear[statePlot, lifePlot, satSolverOutput, x, S, updateLife,
R1, R2, R3, R4, S, bool, antiBool, or, and, join, CNF,
CNFToclausalForm2, clausalForm2ToCNF, array3Processor,
formula, varlist];

Get[NotebookDirectory[] <> "clause_sets_R1_R2_R3_and_R4_.m"];

(*HELPER FUNCTIONS*)

updateLife[stateXt_] := Module[
  {x, xNW, xN, xNE, xW, xE, xSW, xS, xSE, xPrime, pad, dim},
  ,
  dim = Dimensions[stateXt];
  pad = ArrayPad[#, 1] &;
  x[i_, j_] := pad[stateXt][[i, j]];
  xNW[i_, j_] := x[i + 1, j - 1];
  xN[i_, j_] := x[i + 1, j];
  xNE[i_, j_] := x[i + 1, j + 1];
  xW[i_, j_] := x[i, j - 1];
  xE[i_, j_] := x[i, j + 1];
  xSW[i_, j_] := x[i - 1, j - 1];
  xS[i_, j_] := x[i - 1, j];
  xSE[i_, j_] := x[i - 1, j + 1];
  xPrime[i_, j_] := Boole[
    2 < xNW[i, j] + xN[i, j] + xNE[i, j] + xW[i, j] + 1/2 x[i, j]
    + xE[i, j] + xSW[i, j] + xS[i, j] + xSE[i, j] < 4
```

```

]
; Table[xPrime[i, j] , {i, 2, dim[[1]]+1 }, {j, 2, dim[[2]]+1}]
];
statePlot = Magnify[ArrayPlot[#, Frame -> False, Mesh -> True] , .3] &;
lifePlot[seed_, steps_: 6] :=statePlot /@ NestList[updateLife[#] &, seed, steps];
bool=#/.{True->1,False->0}&;
antiBool=#/.{1->True,0->False}&;
or=Or@@Flatten[#]&;
and=And@@Flatten[#]&;
join=Join[##]//Flatten&;
CNF = BooleanConvert[#, "CNF"] &;
CNFToclausalForm2[clausesInCNF_] := (clausesInCNF) /. {Or -> List, And -> List };
clausalForm2ToCNF[clausesInClausalForm2_] := And@@(Or@@#&/(clausesInClausalForm2));
array3Processor[array3_] := And @@ (Or @@ # & /@ Flatten[array3 , 3]);

```

*(*MAIN PROGRAM*)*

*(*ARRAY SIZE DETERMINATION AND BOUNDARY CONDITIONS*)*

```

n=endGeneration;
{i, j} = Dimensions[endState];

Evaluate[Array[x[##,n]&, {i, j}]] = antiBool[endState];

x[_ , 0, g_/;(g!=n)] = False;
x[0, _ , g_/;(g!=n)] = False;
x[_ , j+1, g_/;(g!=n)] = False;
x[i+1, _ , g_/;(g!=n)] = False;

```

*(*PRINTING DESIRED END STATE*)*

```

MessageDialog[Evaluate[Array[x[##,n]&, {i, j}]]//bool//statePlot];

```

*(*TAKING THE UNION OF R1,R2,R3 and R4*)*

```

S[i_,j_,g_]:=Union[
    R1[i,j,g]
    , R2[i,j,g]
    , R3[i,j,g]
    , R4[i,j,g]
];

```

*(*LOOPING CLAUSE SETS*)*

```

formula = array3Processor@Array[S, {i, j, n-1} ];

(*GATHERING ALL VARIABLES USED INTO A LIST*)
varlist =
  Join[
    Flatten[Array[x, {i, j, n-1}]]
  ];

(*SAT SOLVER*)

satSolverOutput = SatisfiabilityInstances[formula, varlist];

(*PROCESSING OUTPUT OF SOLVER*)

initialState=If[Length[satSolverOutput] == 0, "unsatisfiable",
  out1 = Array[x[##,1]&, {i, j}] /. Thread[varlist -> RandomChoice[satSolverOutput]];
  bool@out1
];

(*FORMATTING OUTPUT*)

lifePlot[initialState , 10]

```

LISTING 5.1: wordLife.m

5.2 Clause sets represented in Mathematica code and stored in package file: `clause_sets_R1_R2_R3_and_R4.m`

To go back to position in Chapter 2 please click [2.8](#).

```

R1[i_,j_,g_]:={{x[i, j, g], x[-1 + i, -1 + j, g], x[-1 + i, j, g],
  x[-1 + i, 1 + j, g], x[i, -1 + j, g],
  x[i, 1 + j, g], ! x[1 + i, -1 + j, g], ! x[1 + i, j, g], !
  x[1 + i, 1 + j, g], x[i, j, 1 + g]}, {x[i, j, g],
  x[-1 + i, -1 + j, g], x[-1 + i, j, g], x[-1 + i, 1 + j, g],
  x[i, -1 + j, g], ! x[i, 1 + j, g],
  x[1 + i, -1 + j, g], ! x[1 + i, j, g], ! x[1 + i, 1 + j, g],
  x[i, j, 1 + g]}, {x[i, j, g], x[-1 + i, -1 + j, g], x[-1 + i, j, g],
  x[-1 + i, 1 + j, g],
  x[i, -1 + j, g], ! x[i, 1 + j, g], ! x[1 + i, -1 + j, g],
  x[1 + i, j, g], ! x[1 + i, 1 + j, g], x[i, j, 1 + g]}, {x[i, j, g],
  x[-1 + i, -1 + j, g], x[-1 + i, j, g], x[-1 + i, 1 + j, g],
  x[i, -1 + j, g], ! x[i, 1 + j, g], ! x[1 + i, -1 + j, g], !
  x[1 + i, j, g], x[1 + i, 1 + j, g], x[i, j, 1 + g]}, {x[i, j, g],

```



```

x[i, -1 + j, g], ! x[i, 1 + j, g], x[1 + i, -1 + j, g],
x[1 + i, j, g], x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {x[i, j, g], ! x[-1 + i, -1 + j, g],
x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], ! x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {x[i, j, g], ! x[-1 + i, -1 + j, g], !
  x[-1 + i, j, g], x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g],
x[1 + i, j, g], ! x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {x[i, j, g], ! x[-1 + i, -1 + j, g], !
  x[-1 + i, j, g], x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], ! x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {x[i, j, g], ! x[-1 + i, -1 + j, g], !
  x[-1 + i, j, g], x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], ! x[1 + i, -1 + j, g], x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {x[i, j, g], ! x[-1 + i, -1 + j, g], !
  x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {x[i, j, g], ! x[-1 + i, -1 + j, g], !
  x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], x[1 + i, j, g],
x[1 + i, 1 + j, g], x[i, j, 1 + g]}};

R2[i_, j_, g_] := {{x[i, j, g], x[-1 + i, -1 + j, g], x[-1 + i, j, g],
  x[-1 + i, 1 + j, g], x[i, -1 + j, g], x[i, 1 + j, g],
  x[1 + i, -1 + j, g], x[1 + i, j, g],
  x[1 + i, 1 + j, g], ! x[i, j, 1 + g]}, {x[i, j, g],
  x[-1 + i, -1 + j, g], x[-1 + i, j, g], x[-1 + i, 1 + j, g],
  x[i, -1 + j, g], x[i, 1 + j, g], x[1 + i, -1 + j, g],
  x[1 + i, j, g], ! x[1 + i, 1 + j, g], ! x[i, j, 1 + g]}, {x[i, j,
  g], x[-1 + i, -1 + j, g], x[-1 + i, j, g], x[-1 + i, 1 + j, g],
  x[i, -1 + j, g], x[i, 1 + j, g],
  x[1 + i, -1 + j, g], ! x[1 + i, j, g],
  x[1 + i, 1 + j, g], ! x[i, j, 1 + g]}, {x[i, j, g],
  x[-1 + i, -1 + j, g], x[-1 + i, j, g], x[-1 + i, 1 + j, g],
  x[i, -1 + j, g], x[i, 1 + j, g], ! x[1 + i, -1 + j, g],
  x[1 + i, j, g], x[1 + i, 1 + j, g], ! x[i, j, 1 + g]}, {x[i, j, g],
  x[-1 + i, -1 + j, g], x[-1 + i, j, g],
  x[-1 + i, 1 + j, g], ! x[i, -1 + j, g], x[i, 1 + j, g],
  x[1 + i, -1 + j, g], x[1 + i, j, g],
  x[1 + i, 1 + j, g], ! x[i, j, 1 + g]}, {x[i, j, g],
  x[-1 + i, -1 + j, g], x[-1 + i, j, g], ! x[-1 + i, 1 + j, g],
  x[i, -1 + j, g], x[i, 1 + j, g], x[1 + i, -1 + j, g],

```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]


```

x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g],
x[-1 + i, j, g],
x[-1 + i, 1 + j, g], ! x[i, -1 + j, g], ! x[i, 1 + j, g],
x[1 + i, -1 + j, g], x[1 + i, j, g], x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g],
x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], ! x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g],
x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], ! x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g],
x[-1 + i, j, g], ! x[-1 + i, 1 + j, g],
x[i, -1 + j, g], ! x[i, 1 + j, g], x[1 + i, -1 + j, g],
x[1 + i, j, g], x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g],
x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], ! x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g], !
x[-1 + i, j, g], x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g],
x[1 + i, j, g], ! x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g], !
x[-1 + i, j, g], x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], ! x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g], !
x[-1 + i, j, g], x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], ! x[1 + i, -1 + j, g], x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g], !
x[-1 + i, j, g], x[-1 + i, 1 + j, g], ! x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], x[1 + i, j, g],
x[1 + i, 1 + j, g],
x[i, j, 1 + g]}, {! x[i, j, g], ! x[-1 + i, -1 + j, g], !
x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], x[i, -1 + j, g],
x[i, 1 + j, g], x[1 + i, -1 + j, g], x[1 + i, j, g],
x[1 + i, 1 + j, g], x[i, j, 1 + g]}};

```

```

R4[i_, j_, g_] := {{! x[-1 + i, -1 + j, g], ! x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], !
x[i, -1 + j, g], ! x[i, j, g], ! x[i, j, 1 + g]}, {!
x[-1 + i, -1 + j, g], ! x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], !
x[i, j, g], ! x[i, j, 1 + g], ! x[i, 1 + j, g]}, {!
x[-1 + i, -1 + j, g], ! x[-1 + i, j, g], ! x[-1 + i, 1 + j, g], !

```

[illegible]

Bibliography

- [1] Donald E. Knuth. *The Art of Computer Programming*. 7 Bände. Addison-Wesley, 1968.

