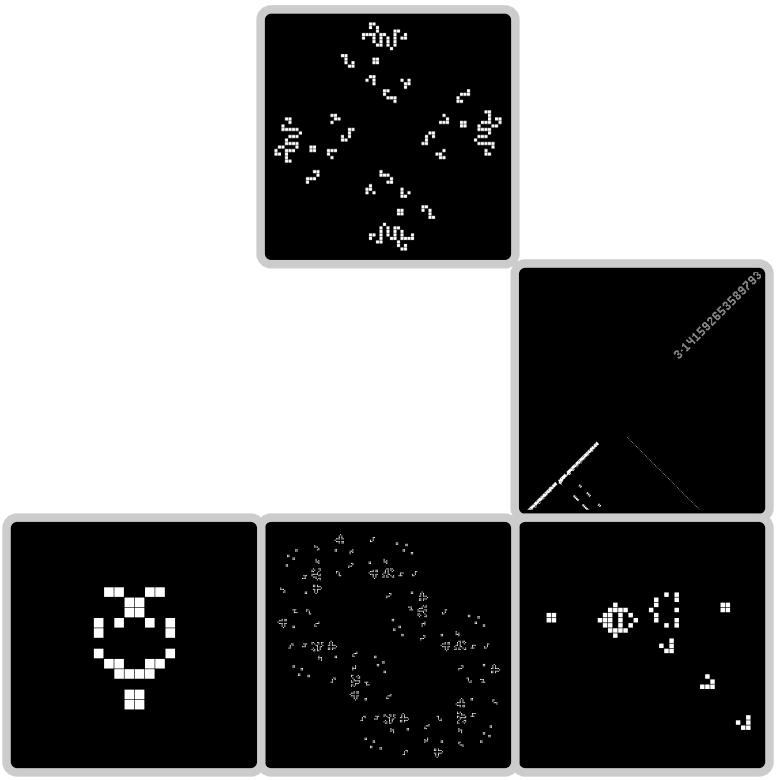
Conway's Game of Life

Mathematics and Construction



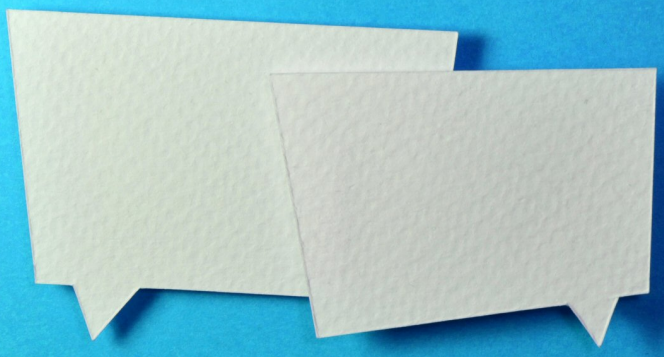
Nathaniel Johnston and **Dave Greene**

CONWAYLIFE.COM/BOOK

To John Horton Conway
For giving us 50 years of Life.

Contents

Appendices and Supplements



Preface

The Goal

In this book, we provide an introduction to Conway's Game of Life, the mathematics behind it, and the methods used to construct many of its most interesting patterns. We generally try to avoid presenting patterns in isolation or as historical notes, but instead we try to guide the reader through the thought processes that went into creating them in the first place.

While we largely follow the history of the Game of Life as we go through the book, we emphasize that this is *not* its primary purpose. Rather, it is a by-product of the fact that most recently-discovered patterns build upon patterns and techniques that were developed earlier. The goal of this book is to demystify the Game of Life by breaking down the complex patterns that have been developed in it into bite-size chunks that can be understood individually.

Intended Audience

While this book does not have any formal mathematical or computer science prerequisites, it is aimed at the level of a first-year undergraduate university student. Some high-school-level topics like logarithms, the "floor" function $f(x) = \lfloor x \rfloor$ for rounding numbers down, the binary representation of a positive integer, and summation notation like $\Sigma_{k=1}^{n} k^2$ for adding numbers, are used frequently and without much explanation. Perhaps the most sophisticated mathematical machinery we use is in Section **??**, where we multiply some 2×2 matrices (but only upper triangular 2×2 matrices, and we provide the matrix multiplication formula).

A basic understanding of how mathematical proofs and computer programming work is also expected. That is, we expect a certain level of mathematical and computer science maturity from the reader, but no specialized knowledge of either topic. Throughout the book, we prove some simple theorems about the Game of Life. These proofs do not use any specialized proof techniques or expect the reader to have any specific university-level mathematical knowledge, but rather just expect the reader to be able to follow a logical argument. Similarly, we introduce a programming language for building computer programs out of Life circuits in Chapter **??**, so that material will be easier to master if the reader has had prior exposure to computer programming.

Somewhat more advanced mathematical topics that we make use of are summarized in Appendix **??**, though they are typically introduced very gently in the main text as well, and we only

require a very surface-level understanding of them. We make use of the greatest common divisor, the least common multiple, and Bézout's identity when discussing oscillator periods in Chapter **??**, so we introduce these tools in Appendix **??**. Infinite series make brief appearances at the end of Chapter **??** and in Section **??**, though the reader is not expected to really have any familiarity with them or to understand convergence issues. Finally, big-Θ notation is used to discuss the size and growth rate of patterns in Sections **??**, **??**, and **??**, so we introduce this concept in Appendix **??**.

How to Use

Conway's Game of Life is an extremely visual game, so this book makes very liberal use of figures throughout, particularly when new patterns or techniques for creating patterns are introduced. However, the Game of Life is best observed in motion, which makes static images in a textbook less than ideal as learning tools. In order to help present the motion of patterns a bit better, we do five things:

- Figures are presented with alive cells in black and dead cells in white, but furthermore we use a gradient from blue to orange to denote cells that were alive in past generations of the pattern. Bright blue cells were just alive, whereas cells that are orange were alive in the more distant past (roughly 75 generations or more—see Figure 1).

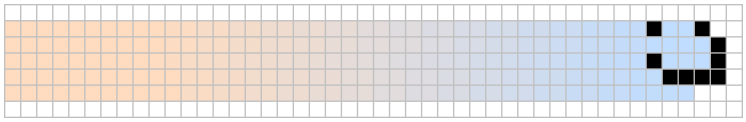


Figure 1: An object moving to the right with a gradient behind it indicating how long ago it was in each location. White cells were never alive, black cells are currently alive, blue cells were alive recently, and orange cells were alive long ago.

- If we really wish to emphasize what a pattern looks like in different generations, all generations of interest will be displayed, along with arrows that specify how many generations have passed. For example, if we want to clarify exactly what the pattern in Figure 1 does as it moves from left to right, we might display it as in Figure 2.

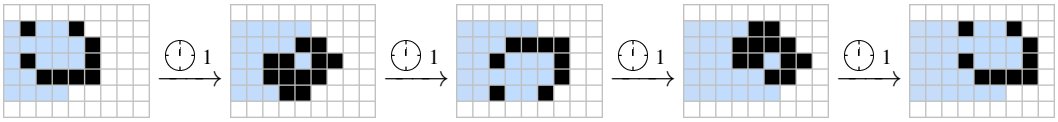


Figure 2: The same object as in Figure 1, but displayed in a more explicit fashion. This image shows how the pattern changes every time 1 generation elapses.

- We use colors to highlight various pieces of patterns, and we maintain a consistent coloring scheme throughout the book. Light pastel colors like aqua, magenta, light green, yellow, and light orange are used to highlight *around* objects—cells in these colors are dead, and they indicate a region in the Life plane consisting of cells that all serve some common purpose or logically make up one "object" (see Figure 3). On the other hand, darker colors like dark green, dark orange, and dark red are used to highlight certain live cells. Dark green is typically used to highlight the input to some reaction, dark orange is typically used for the output of the reaction, and dark red is used otherwise (see Figure 4).

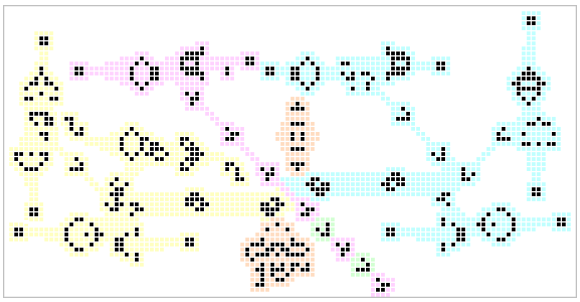


Figure 3: A glider gun made up of several different component reactions that are highlighted in different colors. In particular, gliders are created by a gun highlighted in magenta, lightweight spaceships are created by guns highlighted in aqua and yellow, and those lightweight spaceships are merged into the original glider stream by oscillators highlighted in light orange.

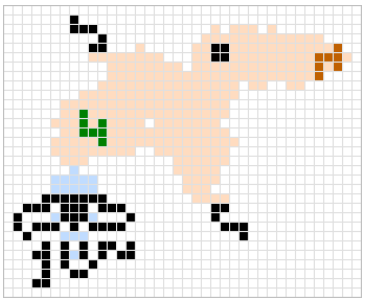


Figure 4: A dark green Herschel comes in from the left and creates the dark orange Herschel on the right. Cells highlighted in blue are constantly changing due to being part of an oscillator, whereas cells that are light orange are usually dead, except when the Herschel passes through.

- In the electronic version of this book, almost every figure is actually a clickable link that will open a text file containing RLE or Macrocell code for the displayed pattern (go ahead, click on any of the figures above, or even one of the five images on the cover page). This code can be copy and pasted into Life simulation software like Golly (`golly.sourceforge.net`) so that it can be explored and manipulated.[1] Note that clicking on figures may not work in certain PDF viewers (such as the viewers built into web browsers), so we recommend using Adobe Acrobat Reader (`get.adobe.com/reader`) to read this book digitally.
- RLE, LifeHistory, or Macrocell codes for all of the patterns displayed in the book are also available at the book's website (`conwaylife.com/book`). Furthermore, the patterns can be viewed and manipulated right on that website as well via any modern web browser, without downloading any additional software.

Exercises

We strongly encourage the reader to work through this book's many exercises that can be found at the end of each chapter. For this reason, it is extremely important to either download Life software or use the book's website to view and edit patterns while making your way through the book—Life is meant to be played, not just watched, and many of the exercises simply cannot be solved without the assistance of Life simulation software.

Roughly half of the exercises are marked with an asterisk (∗), which means that they have a solution provided in Appendix **??**. Exercises also begin with a numeric difficulty estimate on a scale of 1 to 5, displayed within square brackets like [3/5] or [5/5]. Difficulty-1 exercises follow fairly immediately from the relevant definitions and results in the text, up to difficulty-4 exercises that involve either an extensive and delicate calculation, or a complicated multi-step construction. Difficulty-5 exercises are more like projects that could take a day or so of work to complete.

Acknowledgments

The authors are indebted to dozens of people who opened the world of Conway's Game of Life to them. Rather than acknowledging the people who discovered the reactions and patterns that we discuss here, they are credited in footnotes and figures throughout the book.

We extend thanks to Adam P. Goucher for numerous contributions to the writing of this book, especially in Chapter 1. Thanks to Nicolay Beluchenko, Steven Eker, Tanner Jacobi, Mark Niemiec, Michael Simkin, and Satoshi Tanaka for various helpful conversations and corrections. Thanks

[1] For an explanation of how RLE or Macrocell code works, see `conwaylife.com/wiki/Run_Length_Encoded` or `conwaylife.com/wiki/Macrocell`.

to Andrew Trevorrow and Tomas Rokicki for creating the open-source cross-platform CA editor and simulator Golly (golly.sourceforge.net), without which many of the patterns discussed in this book would not have been discovered. Thanks to Chris Rowett for creating LifeViewer (lazyslug.com/lifeviewer), which is used on this book's website (conwaylife.com/book) to display patterns and make them interactive. Thanks to Velimir Gayevskiy and Mathias Legrand for the *Legrand Orange Book* LaTeX template from LaTeXTemplates.com.

Finally, the authors would like to thank their wives Kathryn and Melanie for tolerating them during their years of mental absence glued to this book, and their parents for encouraging them to care about both learning and teaching. Many thanks also to Mount Allison University for giving the first author the academic freedom to pursue a project like this one.

I

Classical Topics

II

Circuitry and Logic

III Constructions

1. The 0E0P Metacell

> If you couldn't predict what [Life] did then probably that's because it's capable of doing anything.
>
> John H. Conway

In the previous chapter, we introduced universal construction and presented several adjustable spaceships based on it. In this chapter, we explore a more complex application of universal construction: a self-reproducing pattern that interacts with nearby copies of itself so as to emulate Conway's Game of Life, or a different cellular automaton, on a much larger and slower scale.

Specifically, we construct a pattern called the **0E0P metacell**,[1] which has the property that if we arrange copies of it on the Life plane then, at a zoomed-out macroscopic scale, it evolves in the same way that the corresponding arrangement of cells would evolve. For example, if we place five copies of this metacell in the Life plane in the formation of a glider, as illustrated in Figure 1.1, then we indeed get a spaceship that travels like a glider (but much more slowly).

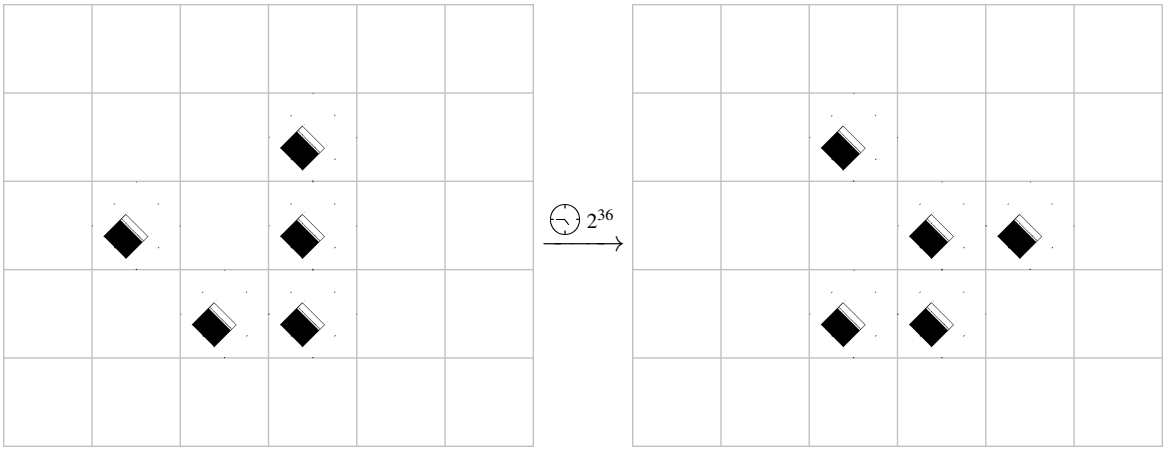


Figure 1.1: A **metaglider** made up of five copies of the 0E0P metacell, each of which is $2^{18} \times 2^{18}$ and logically makes up one "cell". It travels in much the same way as a glider, but 2^{36} times as slowly.

[1] Constructed by Adam P. Goucher from 2014 to November 2018. The acronym "0E0P" stands for "(State) 0 Encoded by 0 Population", in reference to the fact that its "off" state is just a metacell-sized region of empty space—its most remarkable property, which is not shared by any metacell that came before it (see Section 1.9).

A meta-fied pattern like this is roughly $2^{18} = 262\,144$ times as large as the original pattern that it is based on, and it runs $2^{36} = 68\,719\,476\,736$ times as slowly. Indeed, the 0E0P metacell is so much larger and slower than any other pattern that we have seen in this book that evolving the metaglider from Figure 1.1 through four **metagenerations** (i.e., 4×2^{36} generations), to see that it really is a spaceship, would take a couple of years on a modern desktop computer via standard Life simulation algorithms.[2]

While emulating Life within Life perhaps does not seem as remarkable a feat as some of the other things we have achieved over the past three chapters, what really makes the 0E0P metacell shine is that it can actually emulate a huge variety of 2D cellular automata besides Life as well. As a result, any pattern from one of those other cellular automata can be straightforwardly "imported" into Life simply by meta-fying it, thus giving us exotic new types of patterns that were previously not known how to construct.

To illustrate this phenomenon, consider the cellular automaton that has all the same rules as Life, plus the additional rule that a dead cell comes to life if it has exactly 6 live neighbors (i.e., the Life-like cellular automaton with rulestring `B36/S23`). This cellular automaton is called **HighLife**, and it is interesting for the fact that it has a simple **replicator**: a pattern that produces arbitrarily many copies of itself, as illustrated in Figure 1.2.

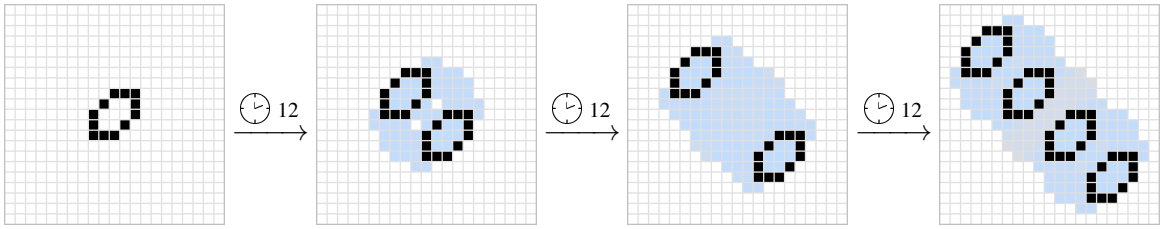


Figure 1.2: A **replicator** in the HighLife (`B36/S23`) Life-like cellular automaton that duplicates itself every 12 generations. Found by Nathan Thompson in February 1994.

After this replicator copies itself the first time, each of its copies attempt to do the same. However, since these copies are right next to each other, two of *their* copies attempt to occupy the same space, and instead annihilate each other (while their other copies are successfully created farther away). This behavior repeats forever: every 12 generations, if there is space for a copy of this replicator then it is made, but two copies being made at the same spot at the same time mutually annihilate. The result is that after n replication cycles (i.e., at generation $12n$), the number of replicators is exactly $2^{B(n)}$, where $B(n)$ is the **binary weight** of n: the number of "1" bits in its binary representation.

Since we're using HighLife's rules that require birth when a dead cell has 6 live neighbors, this is not a valid replicator in Life: when run, it merely degenerates into a configuration of eight blinkers. However, we can "import" its behavior into regular Life by programming the 0E0P metacell to emulate HighLife and then arranging 12 copies of that metacell in the same formation as the 12 cells that make up the replicator from Figure 1.2. This meta-fied replicator is displayed in Figure 1.3.[3] While it is not the first replicator to be constructed in Life (that honor goes to the linear propagator that we mentioned in Section **??**), this is just the tip of the monumental iceberg of what can be done with the 0E0P metacell.

In this chapter, we first explore a few other cellular automata that have exotic patterns that can be implemented in this way in Life via the 0E0P metacell, and then we describe the inner workings of the 0E0P metacell itself.

[2] The fastest "standard" Life simulation algorithm is **HashLife** [Gos84], which is fast enough to run any of the patterns that we saw earlier in this book—even huge ones like the π calculator and the Gemini spaceship. To run patterns that use huge streams of gliders (like the 0E0P metacell) more efficiently, Adam P. Goucher developed a new **StreamLife** algorithm, but even it would require several months to evolve a metaglider through four metagenerations.

[3] Slsparse (see `conwaylife.com/wiki/Slsparse`) comes with a Python script called `isotropic_metafier.py` that can metafy patterns like this automatically.

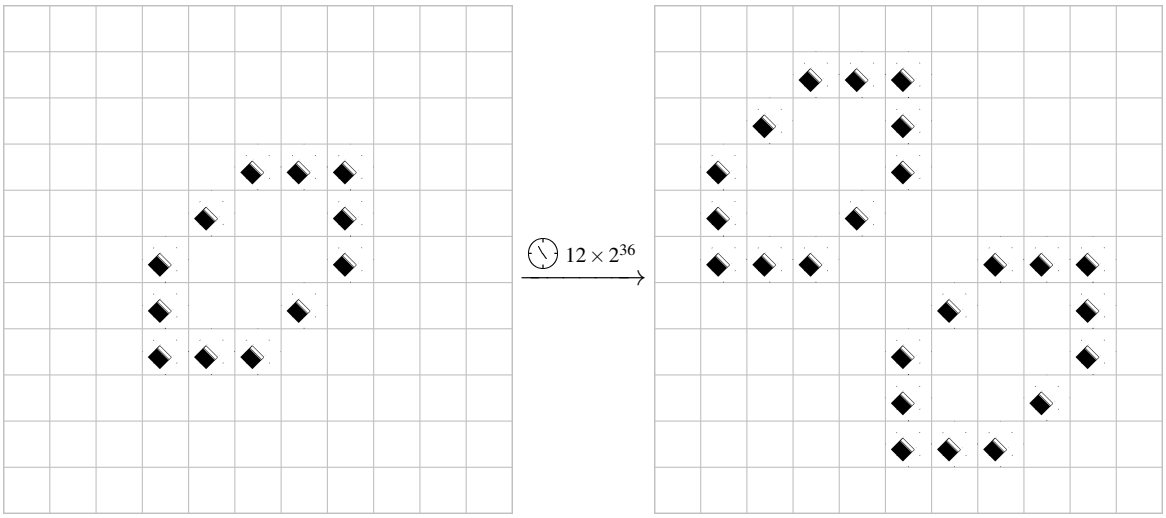


Figure 1.3: A replicator in Conway's Game of Life that is made up of twelve 0E0P metacells, each emulating the HighLife rule (B36/S23) from Figure 1.2.

1.1 Other 2D Cellular Automata

While cellular automata (CA) can act on grids of any dimension and of a variety of different shapes, all of the ones that we consider (and all of the ones that can be emulated by the 0E0P metacell) act on a 2-dimensional square grid. Furthermore, in this section we only consider cellular automata that use two states, which we still refer to as "alive" and "dead", and the Moore neighborhood of Figure **??**, though we will see in Section 1.2 that the 0E0P metacell can emulate some cellular automata without these two restrictions.

1.1.1 Life-Like (i.e., Outer-Totalistic) Cellular Automata

A 2-state cellular automaton is called **outer-totalistic** if the birth and death rules depend only on the state of the current cell, as well the number of live neighbors that it has.[4] That is, they are exactly the cellular automata that can be described by the Bx/Sy rulestring notation that was introduced earlier. An outer-totalistic cellular automaton is said to be **Life-like** if it furthermore satisfies all of the properties that we assumed at the start of this section (i.e., it acts on a 2-dimensional square grid and neighbors are counted according to the Moore neighborhood).

Some Life-like cellular automata have simple patterns that behave unlike any simple patterns that are known in Life itself, as evidenced by the replicator that we saw in Figure 1.2. While that pattern replicates along a single line, there are also Life-like rules that give rise to simple replicators that replicate in multiple directions and fill the whole plane. In fact, in the appropriately-named **replicator** rule (B1357/S1357), *every* pattern is a replicator that repeatedly produces copies of itself in all 8 orthogonal and diagonal directions—see Figure 1.4 and Exercise 1.1.

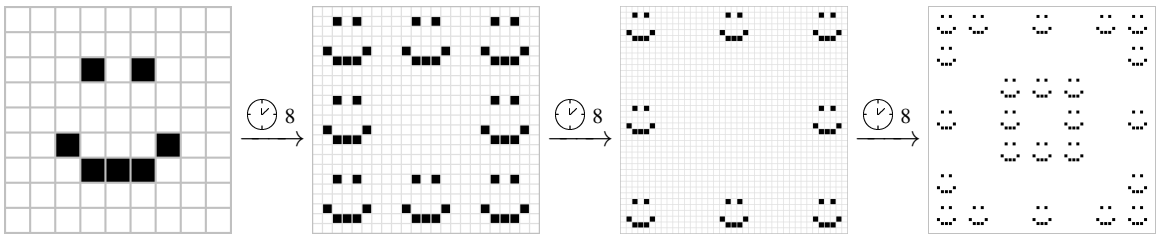


Figure 1.4: In the **replicator** rule (B1357/S1357), every pattern is a replicator that creates copies of itself in the 8 standard directions.

[4] In contrast with **totalistic** cellular automata, in which the birth and death rules depend only on the number of live neighbors including the cell itself.

While the patterns of Figures 1.2 and 1.4 replicate in a sawtooth-like fashion—whenever two copies would be created in the same place, they cleanly destroy each other instead, so their populations repeatedly reach new heights and then jump back down below some fixed value—replicators need not behave in this way. For example, consider the rule `B12345678/S012345678` in which a cell is born if it ever has at least one live neighbor, and then lives forever. In this rule, a single cell acts as a replicator that repeatedly births all cells in its Moore neighborhood, as illustrated in Figure 1.5.

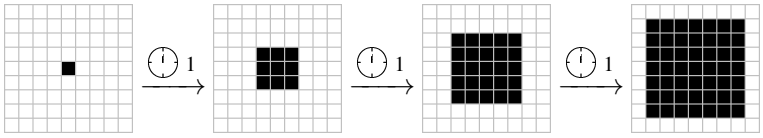


Figure 1.5: A single cell acts as a space-filling replicator in the rule `B12345678/S012345678`.

In fact, elementary replicators of various shapes and speeds are known in a few dozen different Life-like cellular automata,[5] giving us a wide variety of patterns of this type in Life now, thanks to the 0E0P metacell. Somewhat less common are patterns that grow in other ways, like the spiral-growth pattern for the rule `B34568/S15678` that is displayed in Figure 1.6 (compare with the spiral growth pattern that we constructed in Life back in Figure **??**).

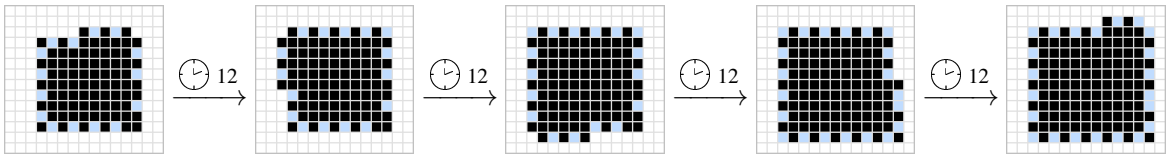


Figure 1.6: A pattern that exhibits spiral growth in the `B34568/S15678` rule. Found by Dean Hickerson in June 2006.

1.1.2 Isotropic (but Non-Outer-Totalistic) Cellular Automata

A cellular automaton is called **isotropic** if the cell transition rules are invariant under rotations and reflections, and it is called **non-isotropic** otherwise. That is, an isotropic cellular automaton may take into account the *relative* positions of neighboring cells, but not their *absolute* positions. Every outer-totalistic cellular automaton is isotropic, but the converse is not true, as shown in Figure 1.7.

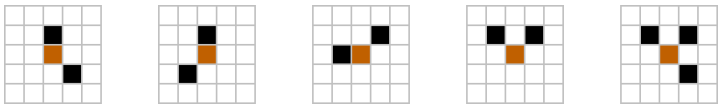


Figure 1.7: In an outer-totalistic cellular automaton, the central cells (displayed in orange) in the leftmost four configurations must all evolve in the same way, since they have the same number of live neighbors (2). In an isotropic cellular automaton, the central cells must evolve in the same way in only the *three* leftmost configurations, since their arrangements of neighbors are rotations and/or reflections of each other. In a non-isotropic cellular automaton, the central cells can evolve in different ways in all five configurations.

Because of the extra flexibility afforded by isotropic cellular automata (there are a whopping 2^{102} isotropic 2-state CA on a 2D square grid, versus "just" 2^{18} outer-totalistic ones), they often contain elementary patterns that appear completely alien when compared to those from Life. For example, consider the CA that has the same rules as Life, but with four isotropic modifications: two to its birth conditions and two to its survival conditions, as illustrated in Figure 1.8. This CA's rulestring is `B3-j6i/S23-c4i`, where the substrings "`-j`", "`6i`", "`-c`", and "`4i`" correspond to these four new rules (in the same order as they are displayed in Figure 1.8). The details of how to construct rulestrings for arbitrary isotropic cellular automata are somewhat involved, so we leave them to Appendix **??**.

[5] See `www.ics.uci.edu/~eppstein/ca/replicators/index.html` for a partial list.

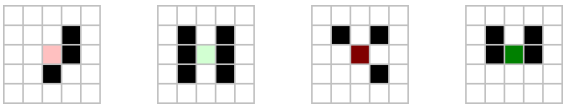
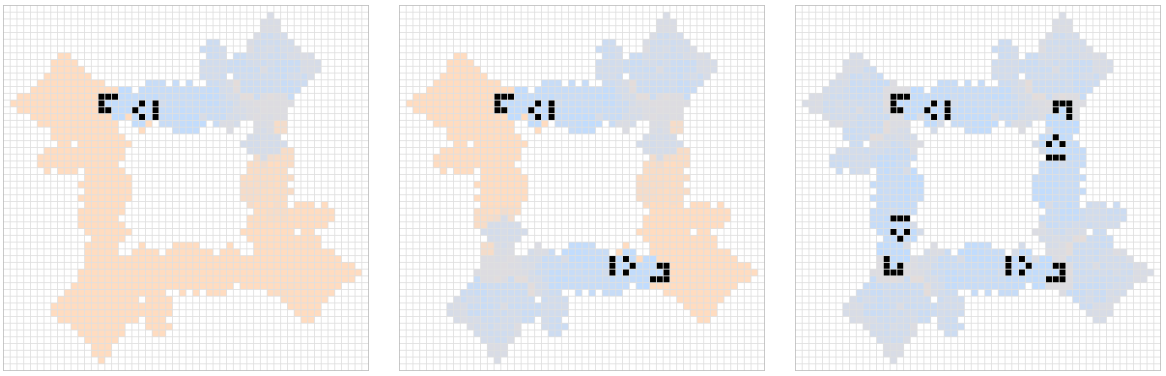


Figure 1.8: The four differences (up to rotation and reflection) between Life (`B3/S23`) and the isotropic cellular automaton `B3-j6i/S23-c4i`. The central cell in the first (i.e., leftmost) configuration is *not* born (whereas it would be in Life), the central cell is born in the next configuration, the central cell dies in the third, and the central cell survives in the fourth.

Our interest in this cellular automaton comes from the pattern displayed in Figure 1.9, which is an example of a **reflectorless rotating oscillator** (or **RRO** for short): an oscillator with the property that one of its phases is a rotation of another one, and two non-interacting copies of the oscillator can combine so as to produce an oscillator with period half as large.[6] Informally, RROs travel around in a loop like a spaceship that periodically turns a corner (they are even sometimes called **looping spaceships**). This is very unlike anything that we have seen in Life—all oscillators that we have seen either stay mostly stationary (like all of the oscillators in the first 8 or 9 pages of Chapter **??**), or move around a path with the help of other stationary pieces (like glider loops, Herschel tracks, and the pi-heptomino hassler from Figure **??**).



(a) One copy, period 200. **(b)** Two copies, period 100. **(c)** Four copies, period 50.

Figure 1.9: A **reflectorless rotating oscillator** in the isotropic cellular automaton `B3-j6i/S23-c4i`. Either one, two, or four copies of the oscillator can be placed along the same path, producing oscillators with periods 200, 100, and 50. Found by ConwayLife.com forums user "Hdjensofjfnen" in January 2020.

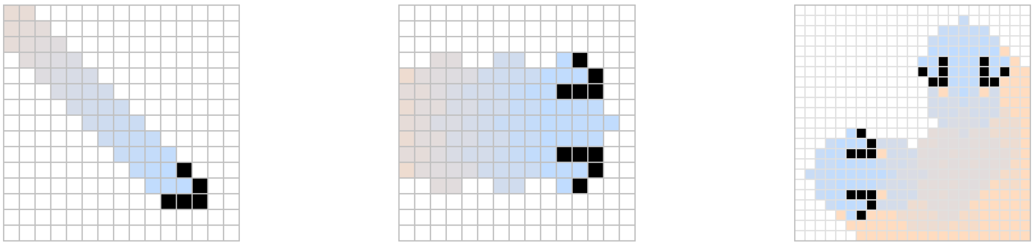
Another exotic type of pattern that appears in isotropic cellular automata, but for which no elementary example is known in their Life-like brethren, is a **spaceship made of spaceships** (or **SMOS** for short): a spaceship that works by colliding two or more other spaceships with each other.[7] One of the simplest such patterns occurs in the (admittedly very *not* simple) cellular automaton

`B3acijn4jktwyz5ijr6-en7c/S2aen3-aceq4acijqty5cikr6ak7c`.

Under this rule, a glider functions as a $c/4$ diagonal spaceship in the exact same way as in Life, but if two gliders collide together just right then they push each other in such as way as to produce a $4c/17$ orthogonal spaceship (see Figure 1.10). Even more remarkably, if two of these $4c/17$ spaceships collide together just right then they produce a $c/44$ diagonal spaceship—a **spaceship made of spaceships made of spaceships** (**SMOSMOS**).

[6] This final "two copies..." requirement enforces the "reflectorless" part of the name: a single glider in a square Snark loop is an oscillator whose phases are rotations of other phases, but two copies of this loop only produce an oscillator with half the period if we overlap the Snarks (thus violating the "non-interacting" part of the definition of an RRO).

[7] The Demonoids that we constructed in the previous chapter are *mostly* made up of gliders. However, they are not spaceships made of spaceships since there is no phase in which they consist *entirely* of gliders.



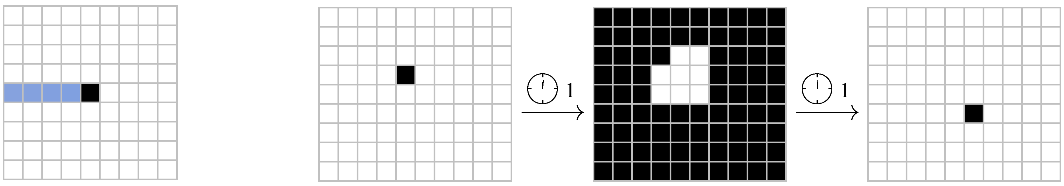
(a) A $(c/4$ diagonal) glider. **(b)** A $4c/17$ orthogonal SMOS. **(c)** A $c/44$ diagonal SMOSMOS.

Figure 1.10: In the rule `B3acijn4jktwyz5ijr6-en7c/S2aen3-aceq4acijqty5cikr6ak7c`, (a) a glider functions as a $c/4$ diagonal spaceship in the usual way. However, when it collides with itself it turns into (b) a $4c/17$ orthogonal spaceship (found by ConwayLife.com forums user "Saka" in August 2017), and when *that* spaceship collides with itself it turns into (c) a $c/44$ diagonal spaceship (found by ConwayLife.com forums user "FWKnightship" in August 2019).

1.1.3 Non-Isotropic Cellular Automata

If we go one step further and consider non-isotropic cellular automata, we can make even just a single cell behave in an extraordinary variety of ways. For example, we can define a rule with the property that cells never survive, and they are only born if they have a single live cell directly to their left.[8] In this rule, a single cell is a spaceship that travels orthogonally at lightspeed (see Figure 1.11(a)).

We can also consider the slightly more exotic cellular automaton in which cells are only born if they have no neighbors or a single neighbor to their southeast, and they only survive if they have neighbors everywhere *except* for to their north and northwest.[9] In this rule, a single cell acts as a $(1,2)c/2$ knightship, as illustrated in Figure 1.11(b). It is similarly possible to construct rules in which a single cell is a spaceship that travels at many other speeds and slopes—see Exercise 1.2.



(a) A single-cell lightspeed orthogonal spaceship.

(b) A single-cell knightship travelling at $(1,2)c/2$. In this rule, a dead cell is born when it has no live neighbors, so the background array of cells alternates between dead and alive in even and odd generations.

Figure 1.11: Non-isotropic rules are general enough that they can turn a single cell into a spaceship with a wide variety of speeds and slopes.

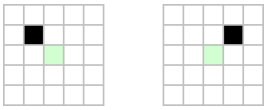
By changing the evolution rule even more, a single cell can also act in a variety of other exotic ways. For example, consider the cellular automaton that is defined by all cells surviving, and cells being born if and only if they have exactly one live cell directly to their northeast or northwest, as in Figure 1.12(a).[10] In this rule, a single cell generates better and better approximations of the Sierpiński triangle, as illustrated in Figure 1.12(b).

Indeed, if we restrict non-isotropic cellular automata to a single dimension (e.g., by having every cell survive and having births only occur to the south of live cells, as we did in Figure 1.12(b)) then we get exactly what are called **elementary cellular automata** [Wol02]. There are $2^8 = 256$ automata of this type, and the Sierpiński-triangle-generating rule works by emulating the one called **Rule 18** (see Exercise **??**).

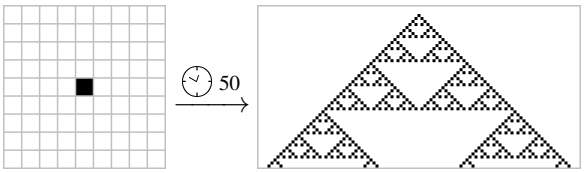
[8] This cellular automaton is described by the rulestring
`MAPAAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`.
For an explanation of rulestrings of non-isotropic cellular automata, see Appendix **??**.

[9] This cellular automaton is described by the rulestring
`MAPwAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`.

[10] This cellular automaton is described by the rulestring
`MAPAAD//wAA//+AAP//AAD//wAA//8AAP//AAD//wAA//+AAP//AAD//wAA//8AAP//AAD//wAA//8AAP//AAD//w`.



(a) Cells always survive, but they are only born in the two configurations shown here (in the central location marked in green).



(b) A single-cell pattern creating the Sierpiński triangle.

Figure 1.12: In the non-isotropic rule in which all cells survive, but are only born as in (a), a single cell produces the Sierpiński triangle as in (b).

There are 2^{512} different 2D not-necessarily-isotropic cellular automata, and the 0E0P metacell can emulate the 2^{511} of them that send a dead cell with no live neighbors to a dead cell. It can thus be used to embed all of the patterns from this section into Life, except for the knightship from Figure 1.11(b), though at a much larger and slower scale. Of particular note is the fact that this method gave the first (and to date, only) explicit construction of a spaceships made of spaceships in Life (and thus the first SMOSMOS in Life as well). This method also gave the first (and to date, only) reflectorless rotating oscillator in Life, but there are minor technicalities that must be overcome in this case (see Exercise 1.3).

1.2 Rule Emulation

While the 0E0P metacell can emulate any 2-state Moore-neighborhood cellular automaton in which a dead cell with all dead neighbors stays dead, it does not do so directly. Indeed, building a 0E0P metacell that directly emulates Life would be highly nontrivial for (at least) two reasons:

- **Quantity of neighbors**: each metacell would need to be able to construct a copy of itself in up to eight different locations. If both the north and east neighbours are present, for example, it may be difficult to maneuver a construction arm to build the northeast neighbour.
- **Survival**: a cell can live for multiple generations, so its logic circuitry would need to be reusable. Reusable circuitry is considerably more expensive than single-use circuitry: compare, for instance, the size of a boat and a Snark—the smallest known one-time turner[11] and reusable stable reflector, respectively.

To get around these two problems, the 0E0P metacell instead emulates an 8-state von-Neumann-neighborhood CA in which every cell dies in every generation (but new cells may be born, so the pattern as a whole need not die).[12] In a cellular automaton of this type, patterns evolve according to a checkerboard pattern—if they live entirely on one color of the checkerboard in one generation, then they will live entirely on its other color in the next (see Figure 1.13 and note that we rotate the square grid by 45 degrees so that its cells match the orientation of the diamond-shaped 0E0P metacell). This checkerboard pattern is important, as it ensures that the 0E0P metacell's four diagonal neighbors are dead (i.e., empty), giving it lots of space to copy itself if necessary.

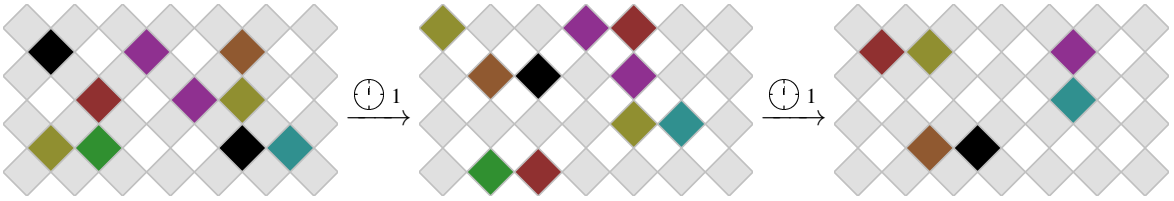


Figure 1.13: An 8-state cellular automaton using the von Neumann neighborhood that is of the type that the 0E0P metacell can emulate. The "dead" state is displayed in a white and light grey checkerboard pattern. The other 7 states are displayed in black and various other colors, which alternate being on the white and light grey portions of the checkerboard pattern from one generation to the next.

[11] We originally showed that a boat is a one-time turner in Exercise **??**(a).

[12] In other words, every pattern is a phoenix in these cellular automata.

Despite the seemingly limited nature of these cellular automata, the fact that they have 8 states makes them general enough to emulate arbitrary 2-state Moore-neighborhood cellular automata at half speed. To see how this works, suppose we are given a particular 2-state CA that we want to emulate. If we represent the "dead" and "alive" cells of that CA by the numbers 0 and 7, respectively,[13] then we can encode the CA by a function

$$M : \{0,7\}^9 \to \{0,7\}$$

that describes how a cell changes from alive to dead, or vice-versa, depending on the state of itself and its 8 neighbors.

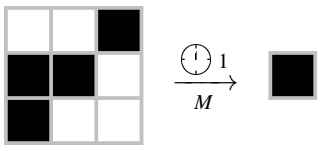
Our goal is to construct an 8-state von-Neumann-neighborhood CA, in which the central cell always dies, that emulates M at half speed. If we label the states of this cellular automaton by $\{0,1,2,3,4,5,6,7\}$ (where 0 is the background "dead" state, as usual), then this is equivalent to finding a function

$$N : \{0,1,2,3,4,5,6,7\}^4 \to \{0,1,2,3,4,5,6,7\}$$

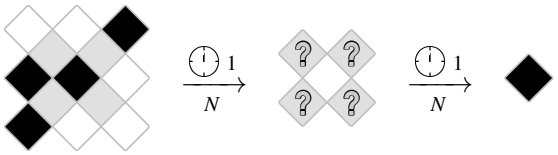
with the property that

$$M\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} = N\begin{pmatrix} N\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} & N\begin{pmatrix} a_{1,2} & a_{1,3} \\ a_{2,2} & a_{2,3} \end{pmatrix} \\ N\begin{pmatrix} a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix} & N\begin{pmatrix} a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \end{pmatrix} \end{pmatrix} \tag{1.1}$$

whenever $a_{1,1}, a_{1,2}, \ldots, a_{3,3} \in \{0,7\}$. That is, we want this 8-state CA to have the property that, if we apply it to 2×2 grids of cells (i.e., rotated von Neumann neighborhoods) independently, then after 2 iterations we get exactly the same result as if we were to apply the original 2-state CA rule to 3×3 grids of cells (i.e., Moore neighborhoods), as in Figure 1.14.



(a) A 3×3 configuration that leads to the central cell surviving in Life.



(b) In the emulating CA, each 3×3 grid evolves in the same way over the course of 2 generations, by evolving 2×2 grids individually.

Figure 1.14: The 8-state CA that we construct will work by acting on 2×2 grids of cells in such a way that applying it twice results in the same evolution (at half speed) as applying the original 2-state CA to 3×3 grids of cells.

While there are many ways that we could construct N, one reasonably simply approach is to ensure that if all four of its inputs comes from $\{0,7\}$ then its output is in $\{0,1,2,3,4,5,6\}$, and vice-versa. That is, we design the 8-state CA so that patterns in even generations consist entirely of cells in the states 0 and 7, thus emulating the "dead" and "live" cells in a 2-state CA, whereas in odd generations they consist entirely of cells in the states 0 through 6, which are just used as helper states for reconstructing the proper configuration in even generations.

We first specify how N should act when all of the inputs that it receives are either 0 or 7. Since there are only $2^4 = 16$ possible combinations of inputs in this case, we can list how N acts on them explicitly:

$$\begin{array}{llll} N(0,0,0,0) = 0, & N(7,0,0,0) = 1, & N(0,7,0,0) = 2, & N(7,7,0,0) = 3, \\ N(0,0,7,0) = 4, & N(7,0,7,0) = 5, & N(0,7,7,0) = 6, & N(7,7,7,0) = 4, \\ N(0,0,0,7) = 6, & N(7,0,0,7) = 4, & N(0,7,0,7) = 3, & N(7,7,0,7) = 0, \\ N(0,0,7,7) = 1, & N(7,0,7,7) = 6, & N(0,7,7,7) = 5, & N(7,7,7,7) = 2. \end{array} \tag{1.2}$$

[13] Yes, 7 seems like a weird choice. It will make more sense shortly.

These 16 evolution rules are displayed in Figure 1.15, and they do not depend at all on the rule that we are trying to emulate (i.e., these input-output combinations of N are the same no matter what M is).

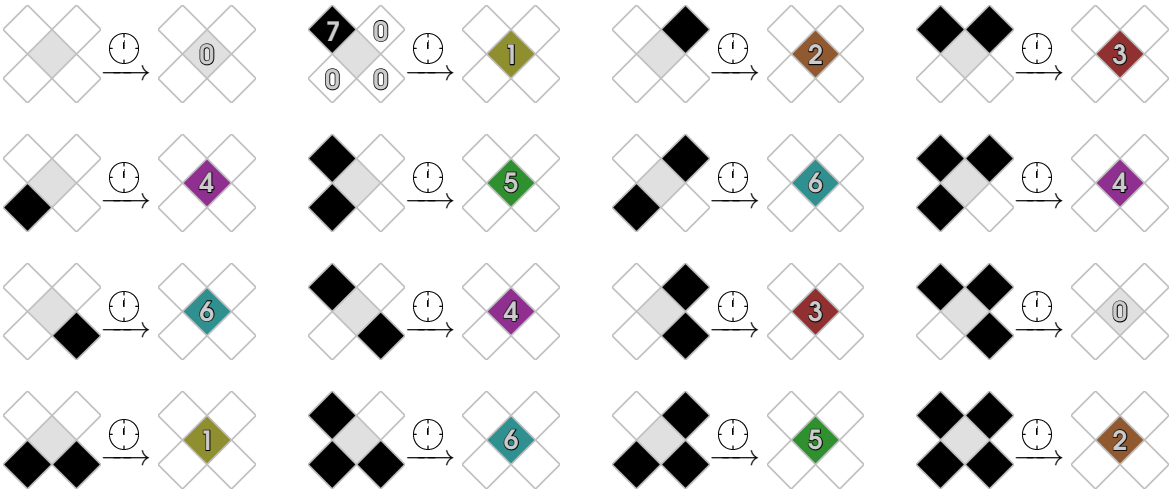


Figure 1.15: The rule for transitioning from even to odd generations in the 8-state von-Neumann-neighborhood CA from Equation (1.2). The states 0 and 7 correspond to "dead" and "alive" in Life and are thus displayed in white and black, respectively. The states 1, 2, 3, 4, 5, and 6 are displayed in yellow, orange, red, magenta, green, and aqua, respectively.

The reason for choosing these seemingly-random transition rules is that they lead to the function

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \mapsto \begin{pmatrix} N\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} & N\begin{pmatrix} a_{1,2} & a_{1,3} \\ a_{2,2} & a_{2,3} \end{pmatrix} \\ N\begin{pmatrix} a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix} & N\begin{pmatrix} a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \end{pmatrix} \end{pmatrix}$$

from $\{0,7\}^9$ to $\{0,1,2,3,4,5,6\}^4$ being injective (i.e., each output of the function corresponds to a *unique* input).[14] That is, given any 2×2 arrangement of states 0, 1, ..., 6, we can figure out which 3×3 arrangement of 0 and 7 states (if any) gave rise to it. We can thus define N on inputs from $\{0,1,2,3,4,5,6\}^4$ so that Equation (1.1) holds (i.e., the cellular automaton defined by N emulates the cellular automaton defined by M at half speed).

The resulting function N is typically quite messy to write out in full detail (as it would just be an explicit list of how the $2^9 = 512$ possible arrangements of 2×2 grids of states 0, 1, ..., 6 should evolve), even if the rule M describes a relatively simple cellular automaton like Life. However, it can be constructed straightforwardly by a computer script,[15] and the emulation of a glider in this way is illustrated in Figure 1.16.

The 0E0P metacell can be programmed to emulate any of the 8^{8^4-1} possible zero-preserving 8-state von-Neumann-neighborhood cellular automata in which every cell dies every generation (not just the ones that emulate one of the 2^{2^9-1} zero-preserving 2-state Moore-neighborhood CAs that we are actually interested in). It takes 2^{35} generations for the 0E0P metacell to run one generation of the 8-state rule (emulated at a 45-degree angle, as in the figures of this section), and therefore 2^{36} generations to emulate one generation of the corresponding 2-state Moore-neighborhood rule (in the usual orientation).

[14] However, the choices we made in Equation (1.2) (or equivalently, Figure 1.15) are not the only ones that lead to this function being injective. This collection of outputs, and many others, can be found by computer search—see Exercise ??.

[15] One such script is available at `conwaylife.com/forums/viewtopic.php?p=38032`.

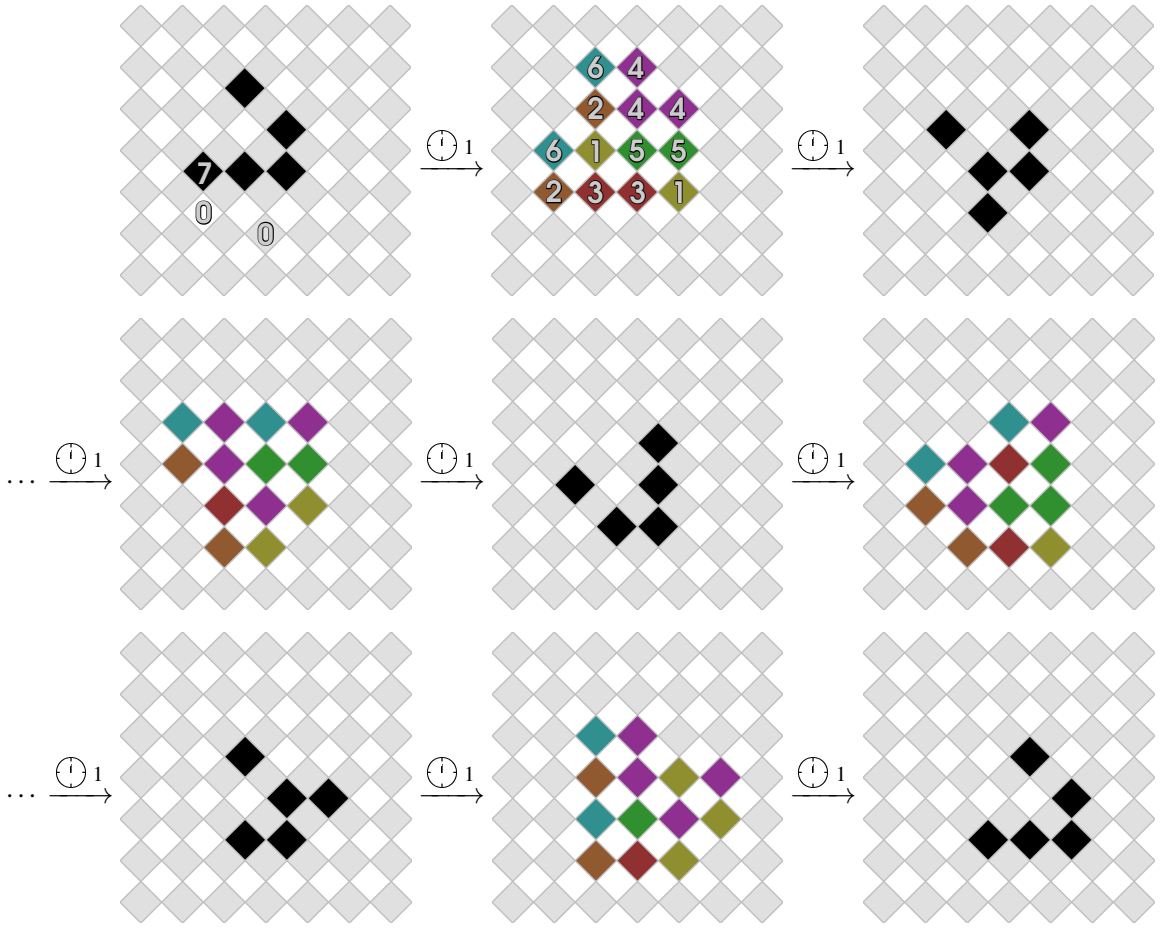


Figure 1.16: A glider from Life being emulated at half speed via an 8-state cellular automaton using the von Neumann neighborhood (rotated by 45 degrees). Uses the same state coloring as in Figure 1.15, which describes the transitions from even to odd generations. The transitions from odd to even generations are encoded as an explicit list of which 2×2 arrangements of states $0, 1, \ldots, 6$ should lead to their central cell being born.

1.3 Overview of the Metacell

The 0E0P metacell works by using the single-channel glider synthesis techniques of Chapter **??** to construct copies of itself in neighboring locations. It starts by constructing a copy to the southeast, northeast, northwest, and southwest (in that order, and only if no metacells are already present in those locations), and then it sends information about its current state to those neighboring metacells so that the proper rule is emulated.

In order to implement this behavior, the 0E0P metacell is made up of three components (see Figures 1.17 and 1.18), which are constructed by parent metacells one at a time:

- The outermost part is the **shell**, which directs information about the states of neighboring metacells. It has exact 90-degree rotational symmetry, consisting of four spiral arms which propagate gliders inwards. Only one of these arms is actually used; the other four exist only to enforce the rotational symmetry.
- Inside the shell is the **kernel**, which does not have any symmetry constraints. The south corner of the kernel contains a clock gun for regulating the metacell's lifecycle and logic circuitry for computing the state of the metacell based on the states of its four neighboring cells. The kernel also contains an output path, shown in magenta in Figure 1.17, which can connect to one of the four construction arms (dashed) or to the input shell of one of the four neighbours.
- At its center is the largest region of the metacell: its **nucleus**, which is a huge boustrophedonic

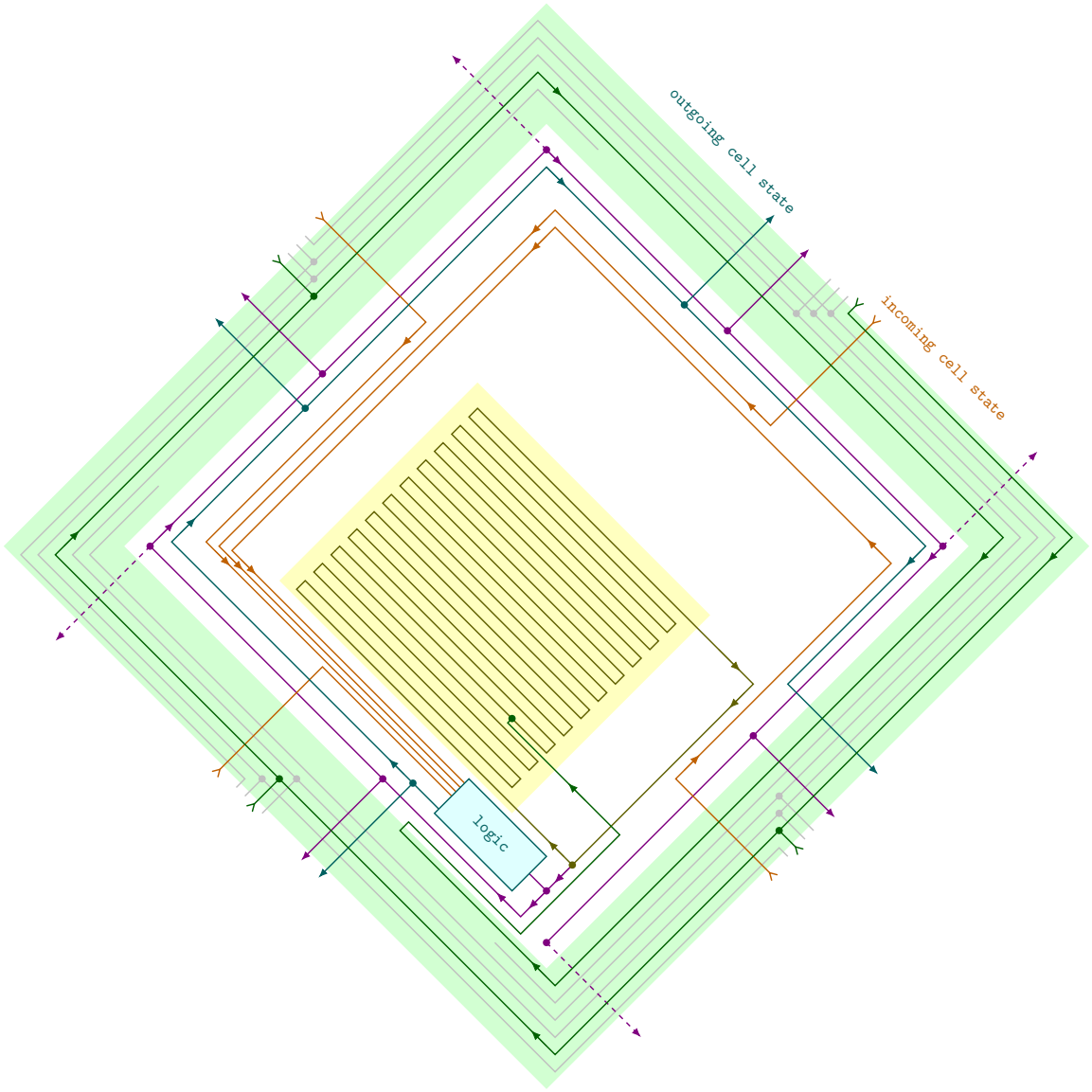


Figure 1.17: A schematic of the 0E0P metacell. The **nucleus**, highlighted in yellow, houses a single-channel glider sequence containing over 3 million gliders, which encode the rule being emulated as well as the construction of the 0E0P metacell itself. When the 0E0P is being constructed, its symmetrical outer **shell** (highlighted in light green) takes in the single-channel glider sequence from any of its neighboring parents (along the path displayed in dark green) and injects it into the nucleus. The unhighlighted region in the middle is the **kernel**, which contains an output path for a copy of the single-channel glider sequence (displayed in magenta), as well as input and output paths along which parent metacells can tell child metacells what state they are in (displayed in orange and aqua, respectively).

glider loop[16] with a period of exactly 2^{29}. It is populated by over three million gliders, which together encode a complete single-channel glider synthesis of the 0E0P metacell, along with a lookup table for the rule that the metacell is emulating.

We describe how these three components of the metacell function throughout its lifecycle in more detail in the next three sections, and we then talk about how they are constructed by parent metacells in Section 1.7. Due to the complexity and size of the 0E0P metacell, it is extremely important that the reader does not just read through the next few sections, but also explores the metacell's different

[16]That is, a glider loop in which the gliders travel back and forth along antiparallel lanes.

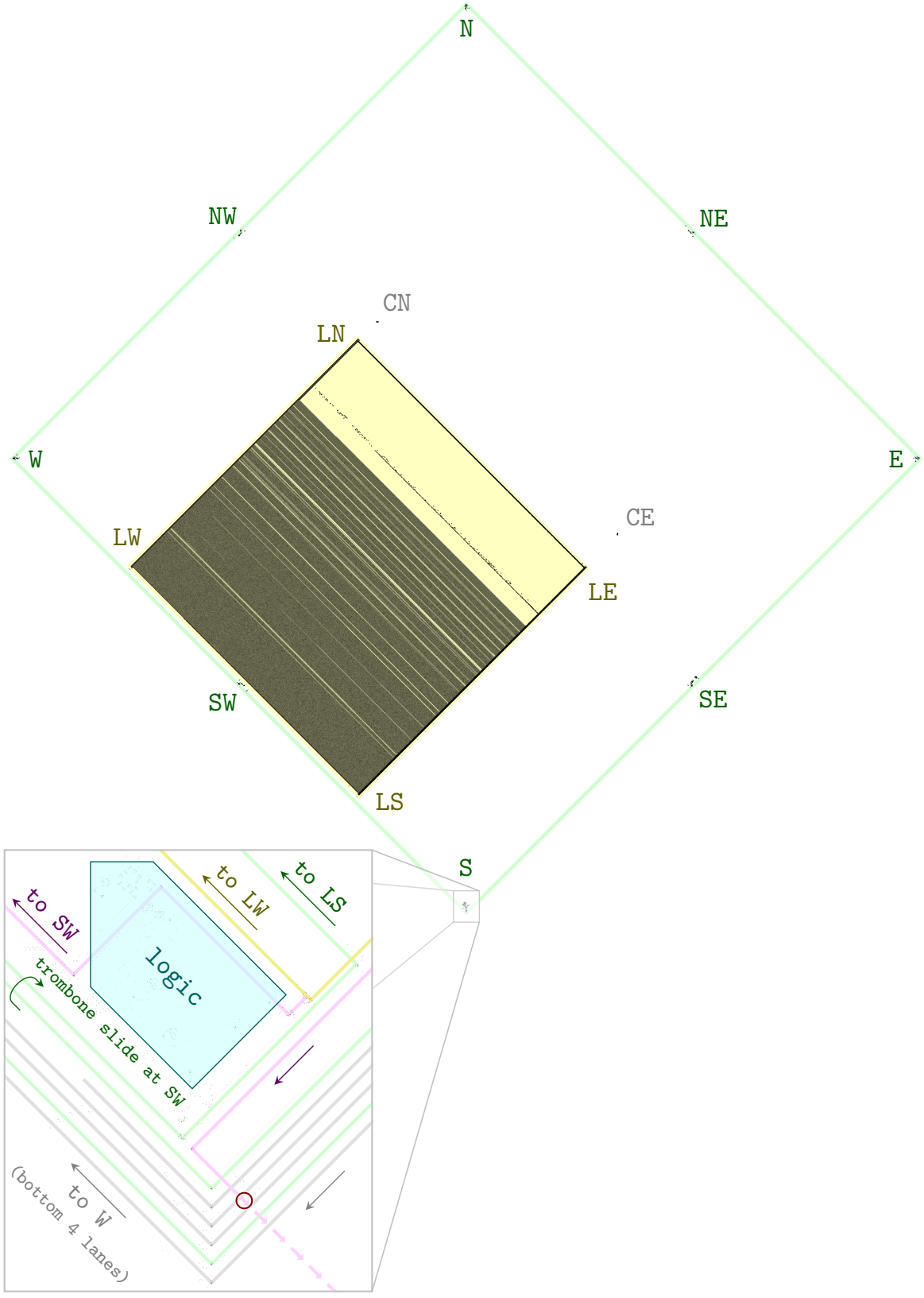


Figure 1.18: The 0E0P metacell itself, which implements the schematic of Figure 1.17. It contains a significant amount of empty space between 14 points of interest. The shell (highlighted in green) goes through points SW, W, NW, N, NE, E, SE, and S, as do the points where the glider paths are reflected in the kernel. The logic circuitry is also located at point S, and an elbow block that is used to construct a new metacell to the southeast is circled in red. The nucleus (highlighted in yellow) has corners at points LW, LN, LE, and LS. The sides between LW and LN, and between LE and LS, are each made up of 1024 two-Snark (i.e., 180-degree) reflectors that create a path long enough to house the nucleus's 3.6 million gliders. The points CN and CE are not displayed in the schematic of Figure 1.17—they contain temporary circuitry that is used to construct the sides of the nucleus.

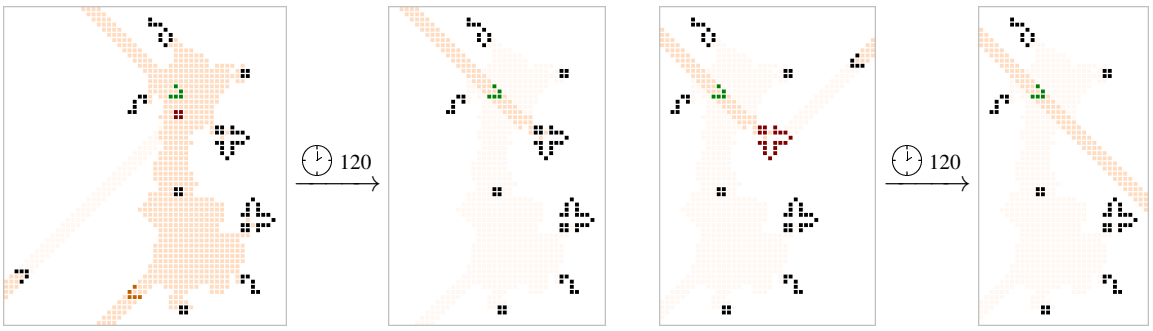
components as we describe them via Life simulation software.[17]

1.4 The Shell and Lane-Switching Circuitry

The rotationally symmetric shell that makes up the outermost edges of the 0E0P metacell is the simplest of its three components. Each of its four sides has four input lanes (in green and grey in Figure 1.17). However, only one of these four input lanes is ever actually used (in green)—the one that connects to the output lane coming in from the kernel of the neighboring metacell (in solid magenta), if it exists.[18] The remaining input lanes (in grey) are unused, and only exist to enforce the shell's fourfold rotational symmetry.

After the metacell has been constructed, these input lanes read in a copy of the 0E0P's single-channel construction recipe and rule lookup table from its neighbors. That (extremely long) sequence of gliders revolves clockwise all the way around the spiral arm of the shell, until it is finally injected into the nucleus at the 0E0P's center.

To make sure that just one copy of the single-channel glider sequence is allowed to enter the shell, regardless of how many neighbors try to send in that glider sequence, the 0E0P strategically deletes part of its own circuitry after the nucleus is populated. By using a stray glider to delete a single block from a syringe, as demonstrated in Figure 1.19(a), the 0E0P can change a syringe-based-reflector into a device that simply eats a glider stream. Another stray glider could also be used to destroy the syringe's eater 2, thus unblocking the stream as demonstrated in Figure 1.19(b), if desired.[19] These extra stray gliders are provided by its clock gun, which we discuss in the upcoming Section 1.5.1.



(a) A single glider coming in from the southwest destroys a block, thus "shutting off" the syringe-based reflector.

(b) A glider coming in from the northeast destroys the eater 2, thus unblocking the glider stream.

Figure 1.19: Some ways of deleting pieces of a syringe (displayed in red) so as to change the path of a glider stream.

1.5 The Kernel

The asymmetric kernel contains the paths that are used to feed the 0E0P's single-channel recipe into its neighbors (in magenta in Figure 1.17). Along these paths, gliders spiral clockwise-and-outwards, and then bypass the shell and do one of two things: go into a construction lane (in dashed magenta) to construct a neighboring metacell, or go into an output lane (in solid magenta) that connects to an input lane of an already-constructed neighboring metacell.

One important aspect of how the output and input lanes (not the construction lanes) of neighboring metacells connect to each other is that, irrespective of the orientation of the neighbors that are communicating, the single-channel glider sequence performs exactly 6 spiral quarter-turns to get from

[17] Unfortunately, no Life simulation software is currently fast enough to run the entire 0E0P metacell "quickly". However, bits and pieces of it can be evolved to see how they behave, and numerous snapshots that show the metacell's state at different timestamps are provided in Section 1.8 and at ??.

[18] This design is similar to that of a circuit on a motherboard—you can think of each side of the 0E0P metacell as having 8 "pins" (4 inputs pins and 4 output pins) that connect to the 8 pins of neighboring metacells.

[19] This is not desired here, but we will see shortly that it is desired at another location in the 0E0P's circuitry.

the south corner of the parent metacell to the south corner of the child metacell. This is how child metacells always end up in the correct phase and orientation after they are constructed. There are also some trombone slides of different lengths on different edges of the kernel, designed to compensate for the "Olympic running track" effect—the fact that outer lanes of the spiral slightly longer than inner lanes. These trombone slides ensure that the communication time from the parent to each of the four children is exactly the same.

In order to control which lane the single-channel glider sequence enters, the 0E0P periodically destroys some of the kernel's Snarks. We saw one method of doing this back in Figure **??**—a single glider can be used to destroy a Snark (with the help of 3 extra pre-placed still lifes). The 0E0P metacell makes use of Snark-destroying reactions like this one[20] so as to redirect copies of its single-channel glider sequence to eight different locations over the course of its lifespan. In order,[21] they are:

1) Along the southeast construction lane (in dashed magenta in Figure 1.17), to construct the child metacell to the southeast.
2) Along the southeast output lane (in solid magenta), to copy the glider sequence into the (now constructed) southeast child's nucleus.
3) Along the northeast construction lane (in dashed magenta), to construct the child metacell to the northeast.

4) – 8) Along the northeast output lane, then the northwest construction lane, and so on counterclockwise.

1.5.1 The Clock Gun

The lane-switching mechanisms described earlier are implemented by a **clock gun** that is located at the top-left corner of the "`logic`" component in Figures 1.17 and 1.18. This gun sends out a single glider every 2^{29} generations, thus matching exactly the period of the glider loop contained in the nucleus. It is made up of a p256 gun attached to a sequence of 21 semi-Snarks, much like the clock gun of Figure **??** that was used by Chapter **??**'s universal computer (though that gun used quadri-Snarks instead).[22]

The gliders that are released by the clock gun travel through sequences of one-time turners and splitters that are interspersed throughout the rest of the 0E0P's circuitry, so as to trigger the appropriate lane-switching mechanisms at each step of its lifecycle. This method is illustrated at a much smaller scale in Figure 1.20, where a clock gun is used to change what happens to the path of a single-channel glider sequence three times.

In the actual 0E0P, the clock gun...

1.5.2 Logic Circuitry

The kernel also contains, at its southern corner, the logic circuitry that is used to compute the metacell's state from the states of its four neighboring parents.

1.6 The Nucleus and Memory Tape

This is one of the few optimisations I deemed necessary. (Without this idea of subroutines and parallel construction, the metacell's side-length would need to be increased by a factor of 8.)

The 0E0P memory unit is a boustrophedonic tape loop composed of 180-degree reflectors (each of which is a pair of Snarks together with some nearby seeds of destruction). Each of those 180-degree reflectors is quite expensive; the single-channel recipe is somewhere between 1 and 2 million ticks.

[20] But not *exactly* this one—see Exercise 1.7.

[21] The single-channel glider sequence really does have to be sent along these 8 paths one after another. If we just used glider duplicators to send it along all 8 paths at the same time, there could be unwanted collisions when constructing child metacells (e.g., if two parent metacells tried to create a child in the same location at the same time).

[22] The 0E0P's clock gun uses the color-changing semi-Snark of Figure **??**, since it is Spartan and is thus easier to construct via a single-channel glider recipe than quadri-Snarks.

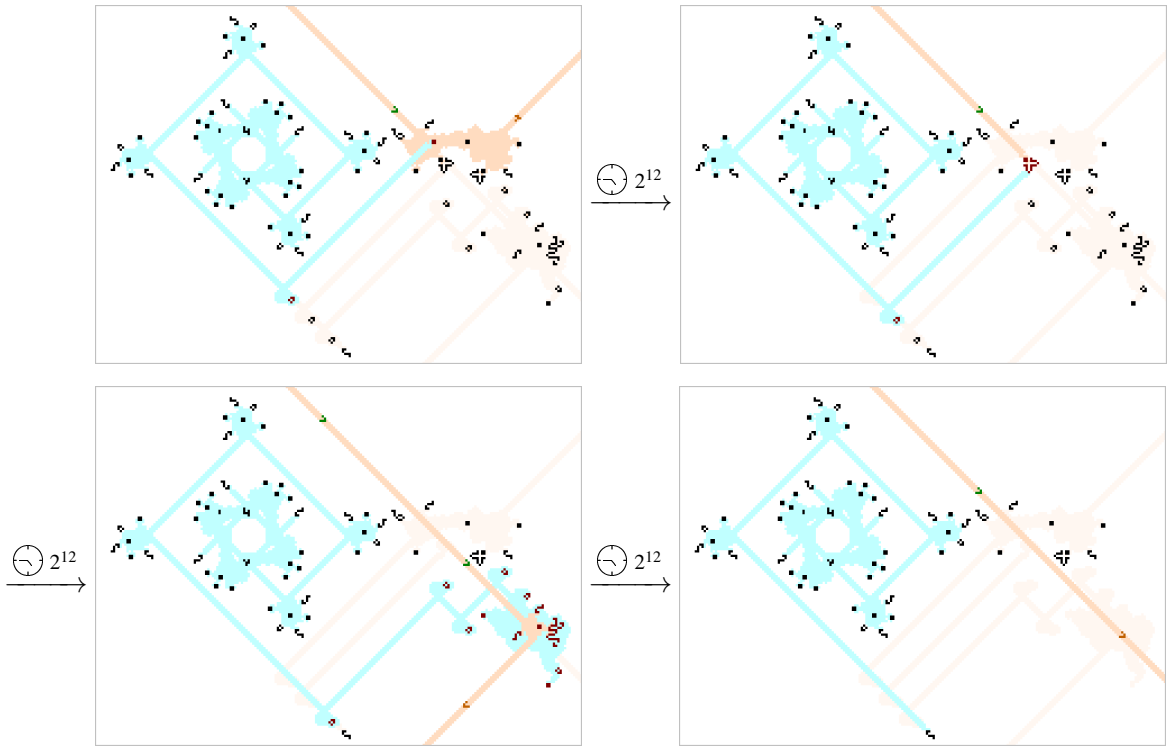


Figure 1.20: A period $2^{12} = 4096$ clock gun (highlighted in aqua) using one-time turners and the reactions from Figures **??** and 1.19 to change the path of the single-channel glider stream coming from the northwest. Initially, that stream is reflected to the northeast, but the first glider from the clock gun blocks the reflector. Its second glider unblocks it and redirects the stream to the southwest via a Snark. Its third glider destroys the Snark, so the stream passes straight through to the southeast.

If the recipes for these reflectors were all individually stored in the memory unit, there would be a significant problem: adding more rows to the memory unit would linearly increase the total length of the recipe that needs to be stored in the memory unit. To break even, it would be necessary to increase the width of each row of the memory unit so that it takes 2 million ticks for gliders to travel from one 180-degree reflector to the next, making the memory unit 524288fd wide!

A solution is to only store the recipe for a single 180-degree reflector in the tape, and to repeatedly invoke this 'subroutine' in a loop. Due to parity constraints, the north-west and south-east 180-degree reflectors have a different shape, so it is necessary to have two 2-million-tick subroutines: one for the north-west reflectors, and one for the south-east reflectors. In the last phase of constructing the daughter metacell, these subroutines are injected into two small temporary memory loops. The two subroutines run in parallel (giving a 2x speedup compared with constructing the reflectors sequentially), and each loop has a fourfold fanout to construct four copies of the same reflector simultaneously. The effect is that the memory loop is constructed in one-eighth of the time necessary for sequential synthesis, so the memory unit can be a mere 65536fd wide.

This reduces the problem of constructing an immense (2^{29}-tick) memory loop to the problem of constructing two smaller memory loops (2^{21} ticks each) to handle the subroutines. These smaller memory loops are small enough that they can be built in a naive sequential manner along with the remainder of the kernel of the 0E0P metacell. The small memory loops are each equipped with a binary counter which halts the subroutine after it has run for exactly 256 iterations and built 1024 reflectors (i.e. 2048 Snarks).

These subroutine storage units are among the most complicated mechanisms in the 0E0P metacell. A lot of the rest of the 0E0P recipe is just simple single-channel recipes, building one thing after another after another. But for these two storage units it's necessary to extract two small pieces of the full recipe, and execute each one 512 times to build the Snark chains that make up the "nucleus". Adam mentioned that the 0E0P would have had to be a lot bigger if the recipes for all of those Snarks

were included inline with no subroutines

Incoming gliders entering the child metacell get a 'head start', being injected several rows beyond the start of the tape. This 'head start' exactly compensates for the delay incurred by the gliders spiralling through the circuitry to pass from the parent metacell to the child, so the tapes of the parent and child metacell end up in exactly the same phase. This injection point is what you called 'Area 51' on Ch91's thread:

1.7 Construction

Ensuring all four metacells are built in the same orientation is by building the outer shell first (which is impossible to get wrong, because it's rotationally symmetric!) and then sending the remainder of the recipe through the correct pin to build the kernel in the correct orientation. If we'd counterfactually sent gliders down one of the other three pins, they'd end up in a different one of the four spiral arms and the resulting metacell would be in the wrong orientation.

1.8 A Complete Metageneration

So far, we have described how the 0E0P metacell works at a fairly high level: every 2^{29} generations, the clock gun fires a glider that potentially changes what happens throughout the metacell. Some of these gliders direct the single-channel glider sequence from the nucleus to start constructing a neighboring metacell, some of them redirect that glider sequence to be duplicated into a neighbor's nucleus, and some of them trigger the self-destruction of the entire metacell.

If we break the 2^{35}-generation lifecycle of the metacell down into $2^6 = 64$ "cycles" of 2^{29} generations each, then they behave roughly as follows:

- Cycle 0: The metacell constructs its southeast neighbor.
- Cycle 1: It duplicates the contents of its nucleus into that of the southeast neighbor.
- Cycle 2: It constructs its northeast neighbor.
- Cycle 3: It duplicates the contents of its nucleus into that of the northeast neighbor.
- Cycles 4–7: Repeat cycles 0–3 for the northwest and then southwest neighbors.
- Cycles 8–12: Do nothing (just spend some time "looking like a cell").
- Cycle 13: Empty the single-channel glider stream out of the metacell's nucleus.
- Cycle 14: The parent metacell sends its state to all four of its children.
- Cycle 15: The parent metacell self-destructs while the child metacells compute their states.
- Cycles 16–63: Child metacells either self-destruct (if their state is 0) or do nothing (if their state is non-zero, they just spend time "looking like a cell").

A bit more specifically, ...

We now describe what happens throughout the 0E0P's lifecycle in much more specific detail, and highlight key timestamps where interesting things happen or change in its circuitry.

1.8.1 The First 0.26×2^{29} Generations: Construction of the Southeast Child's Shell

We start with an 0E0P metacell that is ready to start constructing its neighbors. We say that generation $t = 0$ is the one displayed in Figure 1.22: the parent metacell has a Herschel in its clock gun that will create a glider that escapes past all of the semi-Snarks and removes an eater that is along another one of the clock gun's output paths (a path that receives a glider every 2^{25} generations, rather than just every 2^{29}).

A glider escapes via that new path 2^{25} generations later, at $t = 33\,554\,432 = 0.0625 \times 2^{29}$, and removes an eater 2 from a syringe (see Figure 1.23). This unblocks a copy of the single-channel glider recipe, which now spirals outward through the kernel until it reaches the construction elbow at the metacell's south corner. That construction elbow is then pushed southeast, and is used to begin constructing the southeast child metacell's symmetric shell.

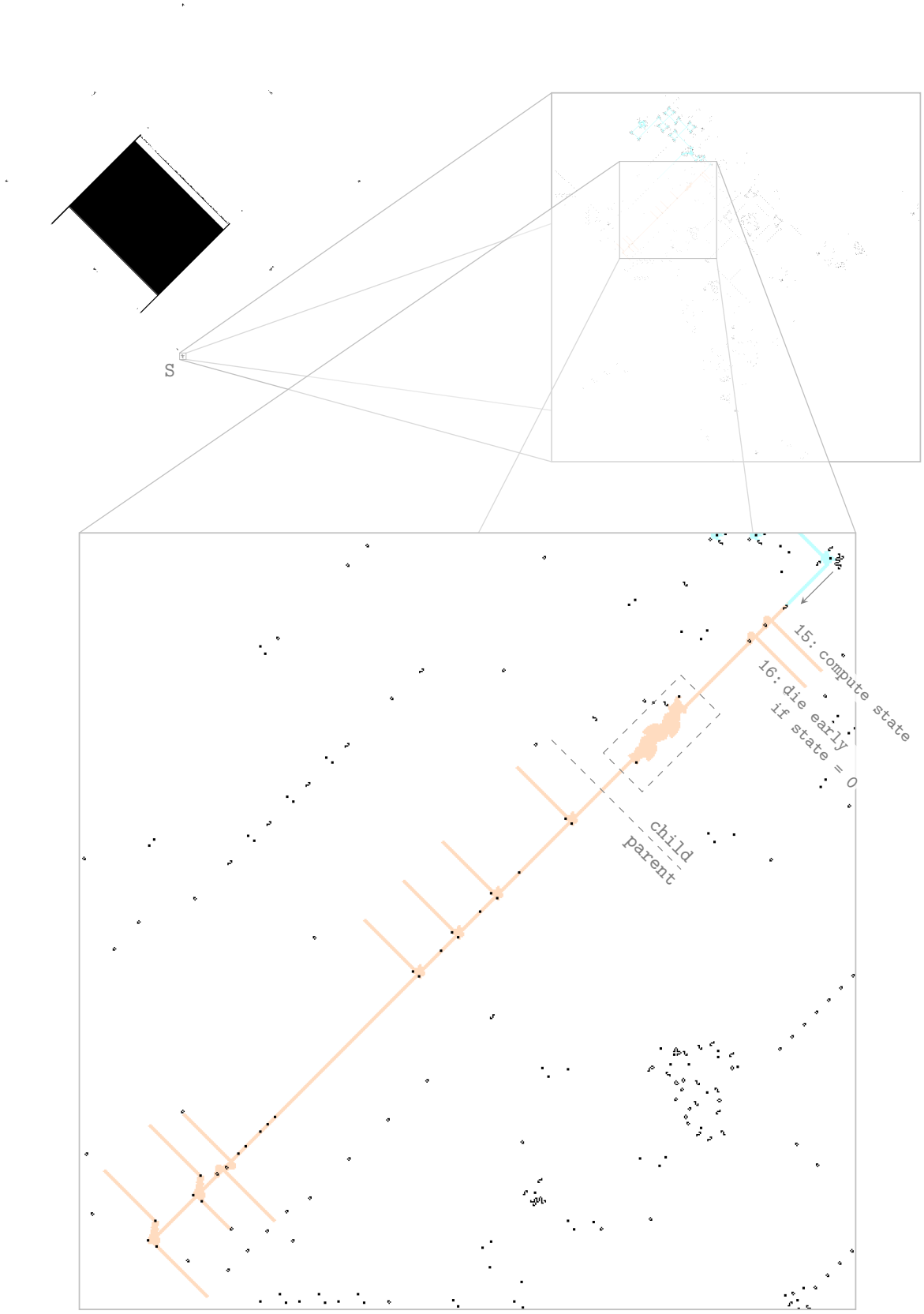


Figure 1.21: The clock gun.

Construction of the child's symmetric shell continues in several stages. A stage often begins with the construction of a Cordership, followed by a slow salvo to shoot down the Cordership once it has

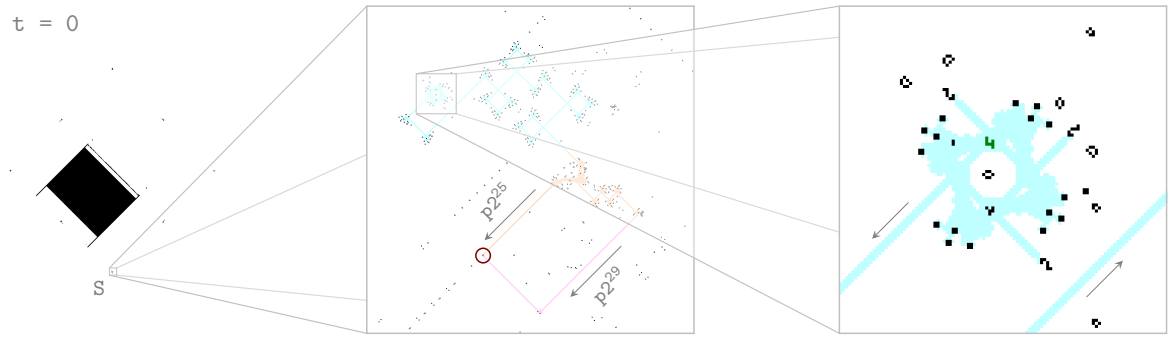


Figure 1.22: A Herschel in the parent metacell's clock gun (highlighted in aqua) is about to create a glider that will make it out of its period 2^{29} branch. That glider will destroy an eater (circled in red) so that the next glider can escape along that path 2^{25} generations later.

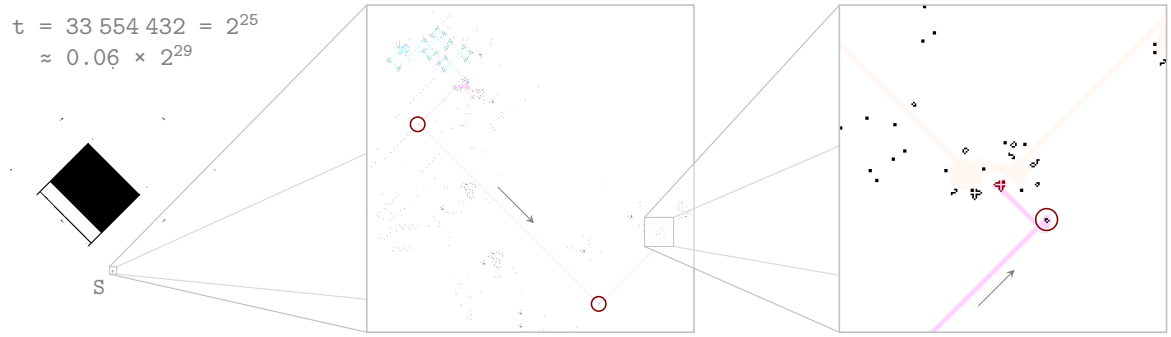


Figure 1.23: A Herschel in the parent metacell's clock gun (highlighted in aqua) is about to create a glider that will escape via the path that was cleared in Figure 1.22. It will use some one-time turners (circled in red) and then destroy an eater 2 from a syringe, which will allow a copy of the single-channel glider recipe to escape from the nucleus and start constructing the southeast child metacell, starting with its shell.

traveled the right distance and turn it into a new target "hand" block at the correct location.[23] Another slow salvo recipe then follows, to build whatever is needed at the location of the new hand block. There are many visible gaps in the 0E0P's single-channel recipe, and most of them correspond to the flight time of a Cordership, while the recipe waits to shoot it down after it has travelled the right distance.

The first such Cordership launch happens at generation $t = 50\,921\,682 \approx 0.0948 \times 2^{29}$ (see Figure 1.24), where a Cordership is constructed at the child's south corner and is launched to its east corner.

Figure 1.24: A glider (circled in red) at the south corner of the child metacell's shell, about to synthesize a Cordership that does a long-distance elbow push to its east corner.

[23] The 0E0P metacell uses 3-engine Corderships, since the single-channel recipe for the 2-engine Cordership from Figure **??** and Appendix **??** was not known at the time of its completion in 2018.

By generation $t = 139\,291\,154 \approx 0.2594 \times 2^{29}$, construction of the final south corner of the symmetric shell is just about complete—the final construction glider has been produced, and a long-range backwards elbow move (**LRBEM**) recipe is processed immediately after this. At $t = 140\,354\,185 \approx 0.2614 \times 2^{29}$, the return glider from this LRBEM recipe has made a new elbow, and the single-channel recipe then shoots a single glider sideways from the new elbow position (at $t = 140\,355\,701$), removes the elbow (at $t = 140\,356\,370$), and then uses five gliders to clean up some leftover junk from the LRBEM (at $t = 140\,880\,132$), far away in the northeast corner.[24]

That single extra stray glider that was shot sideways by the temporary elbow will be used shortly to trigger construction of the southeast child metacell's kernel.

1.8.2 Generations 0.26×2^{29} to 0.80×2^{29}: Construction of the Southeast Child's Kernel

At generation $t = 141\,139\,967 \approx 0.2629 \times 2^{29}$ the extra stray glider has traveled through several one-time turners to the circuitry at the parent's southeast edge. It then cleanly destroys a Snark from the single-channel glider recipe's path (see Figure 1.25), so that the recipe is directed into the child's newly-constructed symmetric shell. The design of the shell allows the interior part of the metacell to always be constructed in the same orientation, no matter which direction the construction recipe is coming from.

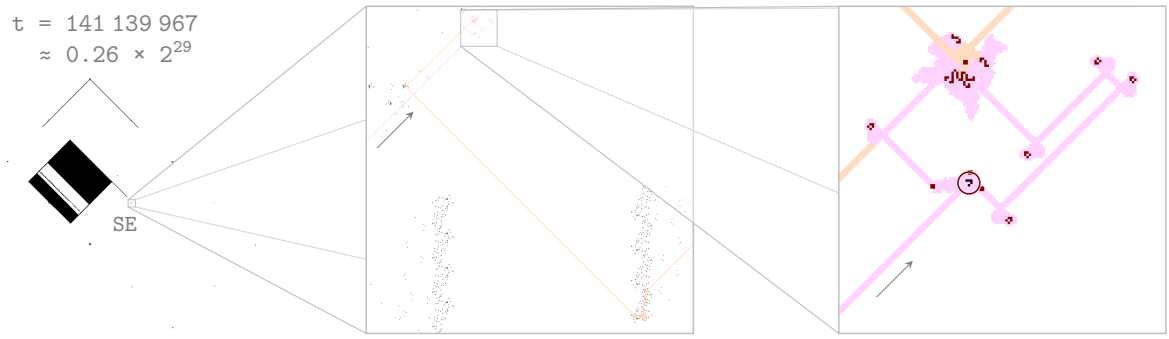


Figure 1.25: The southeast child's shell has been constructed. A single stray glider (circled in red) is about to delete a Snark, allowing the rest of the construction recipe to enter the correct lane of the child's shell. This ensures that the rest of that child is constructed in the proper orientation.

The kernel takes a bit more than twice as long to construct as the shell, and makes use of numerous long-distance Cordership-based elbow pushes. This construction is heavily unoptimized, as it was generated automatically by slsparse using known standard methods—fire a Cordership towards a location, build something there, fire a glider back, and repeat.

However, this could be made much less expensive. For example, instead of using a long-distance Cordership-based elbow push to start construction of the CN construction site (which is done at $t = 181\,162\,958 \approx 0.3374 \times 2^{29}$), it would have been possible to aim gliders at an outlying still life in the unused circuitry near the construction point along the northwest edge, produce a new target object, then rebuild the temporarily sacrificed still life. The 0E0P metacell could be reduced in size and made to run several times faster with these kinds of optimizations—but it would have been far more difficult to complete a working pattern.

The last part of the kernel to be constructed is its south corner, which begins at generation $t = 367\,748\,078 \approx 0.6850 \times 2^{29}$ (see Figure 1.26). The entirety of the metacell's clock gun and logic circuitry has been constructed at this point. The final glider used in the south corner's construction is emitted at $t = 430\,473\,285 \approx 0.8018 \times 2^{29}$, followed by a Snarkmaker recipe that is used to redirect the rest of the single-channel glider recipe towards the nucleus.

[24] This out-of-order cleanup is just the way that slsparse automatically compiled these LRBEMs in 2018. There are more efficient recipes known now.

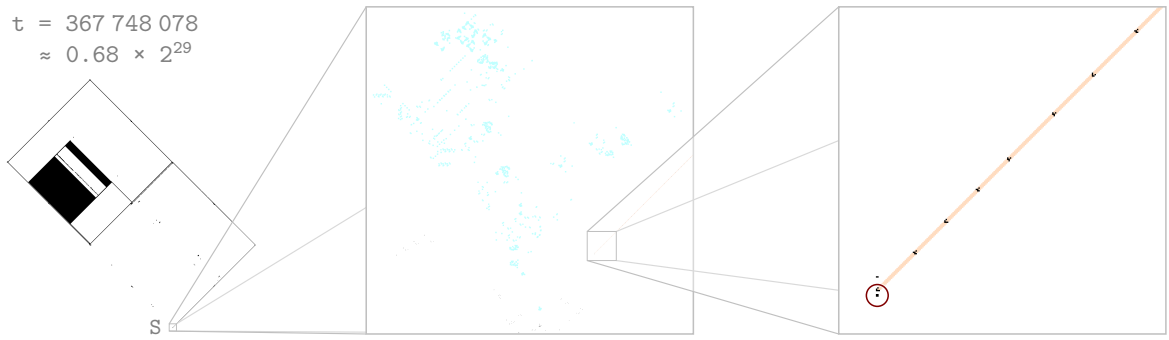


Figure 1.26: Construction of the child's south corner begins. The logic circuit (highlighted in aqua, including the clock gun) will now be constructed by the elbow block circled in red.

1.8.3 Generations 0.80×2^{29} to 0.82×2^{29}: CN and CE Loop Initialization

A single "trigger" glider passes through the Snark that was just created at the south end of the kernel, and is redirected towards a temporary Snark that it passes through at generation $t = 431\,312\,741 \approx 0.8034 \times 2^{29}$ (see Figure 1.27). That glider then travels northeast and then northwest so as to initialize the CN subroutine loop. The single-channel glider recipe that will fill that loop will follow the same path shortly afterwards.

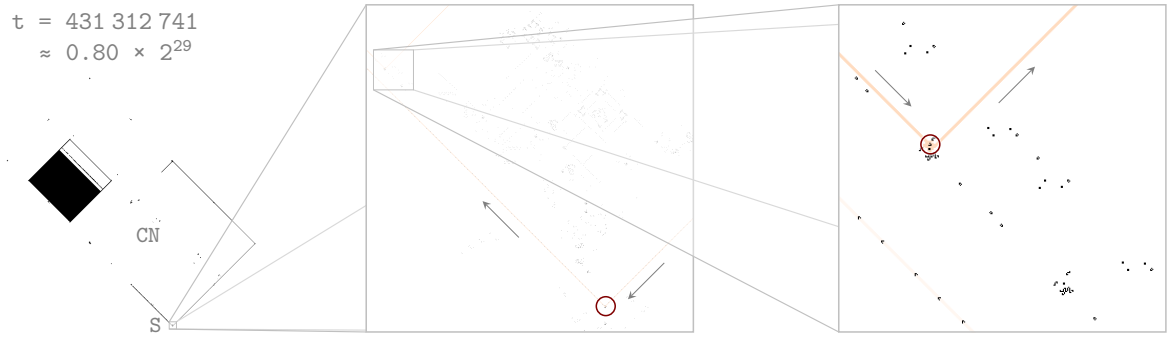


Figure 1.27: The child's kernel has been constructed. A glider goes through the final Snark that was constructed at the south corner of its kernel (circled in red in the middle) and then is redirected through another temporary Snark (circled in red on the right) that leads it to CN, which will trigger construction of the nucleus's northwest wall.

By $t = 431\,969\,616 \approx 0.8046 \times 2^{29}$, the leading trigger glider reaches a temporary Snark in the CN subroutine storage loop (see Figure 1.28). It then goes through a splitter at $t = 431\,969\,830$ and initializes the period 2^{29} clock gun there at $t = 431\,971\,416$, which will destroy the period 2^{21} storage loop after it has been used $2^{29}/2^{21} = 256$ times.

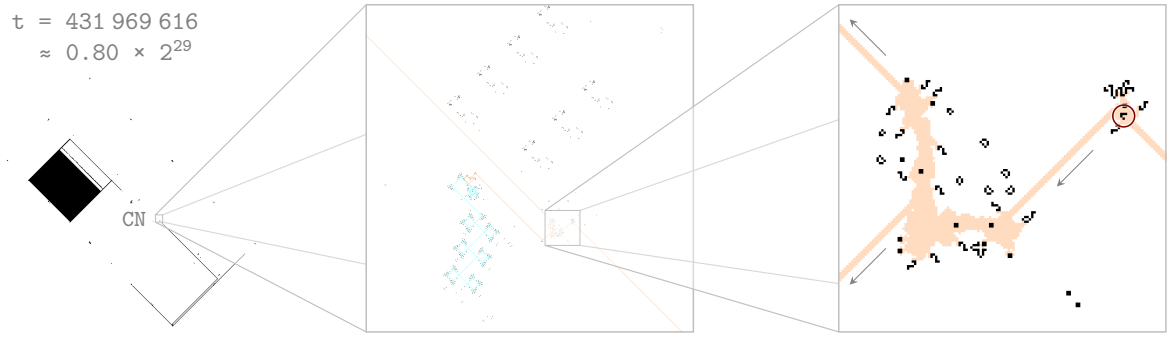


Figure 1.28: The glider from Figure 1.27 (circled in red) has reached CN and is about to start the clock there (highlighted in aqua). That clock will trigger CN's self-destruction 2^{29} generations later, after the period 2^{21} CN subroutine loop has been used $2^8 = 256$ times.

The single-channel recipe following the trigger glider then goes through the same splitter, to be duplicated several times into four streams heading southwest that construct the northwest wall of the nucleus, and also to be copied into the storage loop where it will eventually re-enter this same splitter again, 255 more times. The final glider in that portion of the single-channel recipe enters the loop at $t = 433\,582\,348 \approx 0.8076 \times 2^{29}$ (see Figure 1.29).

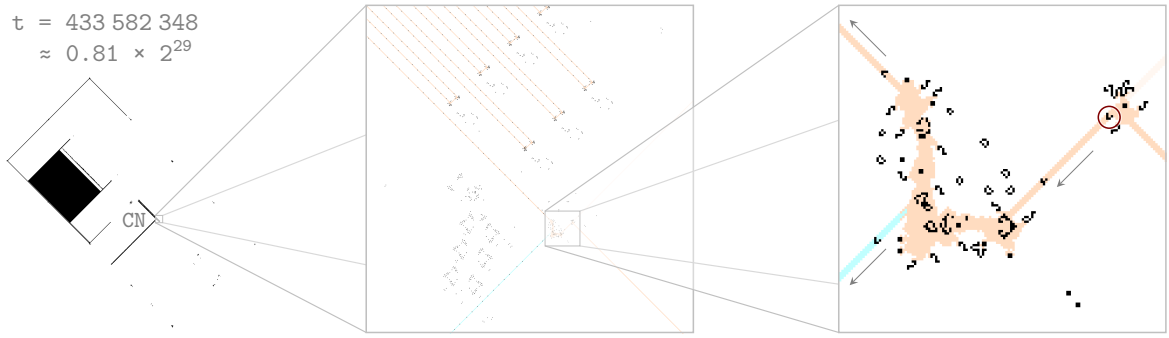


Figure 1.29: The subroutine glider loop at CN is now filled, since its final glider (circled in red) just passed through the same temporary Snark as in Figure 1.28, which will soon be destroyed by the glider from Figures 1.27 and 1.28. The gliders leaving from the aqua lane of the splitter are duplicated three more times and then head to LW to construct the northwest wall of the nucleus.

That loop of gliders can be seen having constructed the first set of 8 Snarks along the northwest wall of the nucleus by generation $t = 434\,341\,913 \approx 0.8090 \times 2^{29}$ in Figure 1.30. This 8-Snark construction is repeated a total of 256 times along this northwest wall (for a total 1 024 reflectors made up of 2 Snarks each).

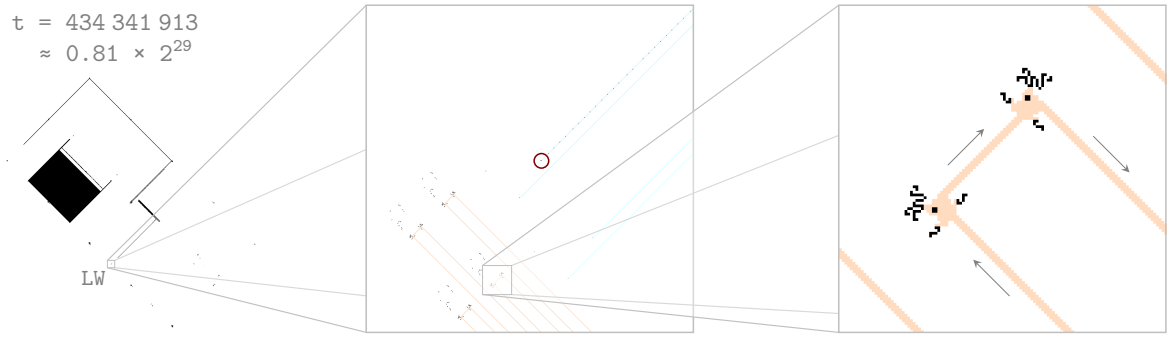


Figure 1.30: The subroutine glider loop has been used once at LW to construct the first set of Snarks along the nucleus's northwest wall (the last elbow operation from that loop is about to pull the elbow block circled in red by 14fd). The glider from Figure 1.27 is between CE and SE, on its way to S.

After the subroutine storage loop is filled back in Figure 1.29, the input Snark from that figure must be removed because it's directly in the way of the circulating recipe in the storage loop. The trigger glider does this, and the Snark is completely erased as of $t = 434\,725\,275 \approx 0.8097 \times 2^{29}$. The trigger glider then travels southeast, and then southwest, back to the child metacell's south corner, where it will destroy and be destroyed by the Snark from Figure 1.27 that sent it to the CN storage loop in the first place. This final act of the trigger glider takes place at $t = 434\,724\,270 \approx 0.8097 \times 2^{29}$ (see Figure 1.31).

The entire procedure used to initialize the CN subroutine loop, as described by Figures 1.27–1.31, then repeats for the CE subroutine loop. At $t = 435\,925\,955 \approx 0.8120 \times 2^{29}$, the trigger glider for the CE subroutine recipe passes through the location where the Snark from Figure 1.31 used to be, and instead goes into a Snark that directs it to CE (see Figure 1.32).

That trigger glider will then activate the clock gun located at CE at $t = 436\,382\,282 \approx 0.8128 \times 2^{29}$, in the exact same way as happened at CN in Figure 1.28. The subroutine portion of the single-channel recipe then fills the loop, with its final glider entering the loop at $t = 437\,805\,347 \approx 0.8155 \times 2^{29}$.

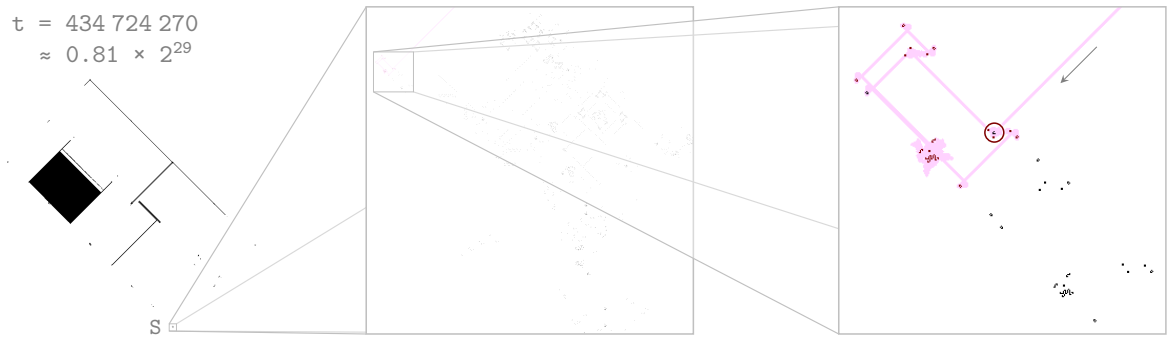


Figure 1.31: The glider from Figure 1.27 makes it back to S, where it destroys (and is destroyed by) the Snark from that figure. This opens a path toward the CE construction location, which will be used soon.

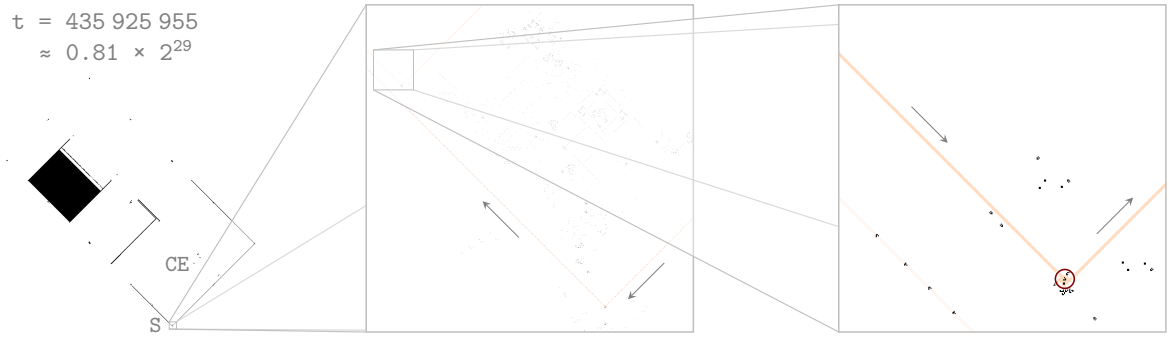


Figure 1.32: A glider goes through a Snark at the child's south corner that leads it to CE, which will trigger construction of the nucleus's southeast wall (just like the glider from Figure 1.27). The same things as in Figures 1.28–1.30 then happen, but along that southeast wall instead of the northwest one.

Once this glider passes through this temporary Snark, the full construction recipe of the 0E0P metacell has been completely processed.

The trigger glider then destroys the CE subroutine loop's input Snark, and is then redirected along a self-destruct chain toward the child's south corner, removing reflectors in four more locations along the way. Finally, it destroys (and is destroyed by) the Snark that directed it to CE in the first place. This final usage of the trigger glider occurs at $t = 438\,935\,522 \approx 0.8176 \times 2^{29}$ (see Figure 1.33).

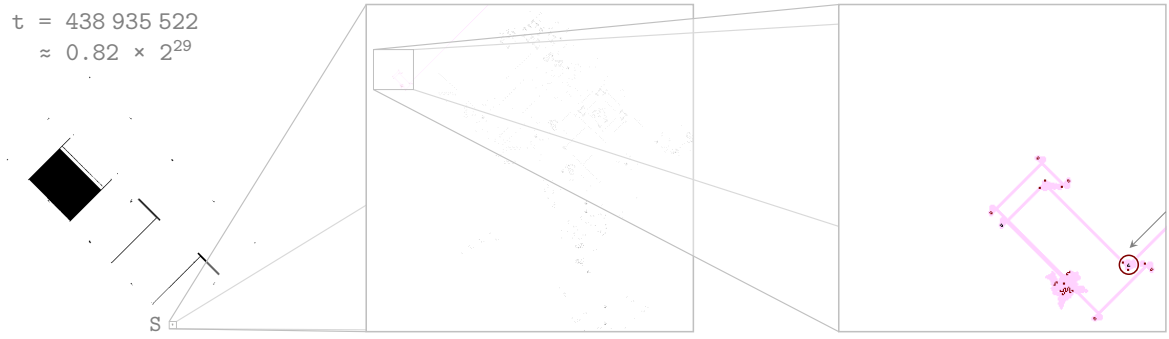


Figure 1.33: The glider from Figure 1.32 makes it back to S, where it destroys (and is destroyed by) the Snark from that figure. A complete copy of the construction recipe from the parent metacell has now been used—the next copy will be directed into the nucleus of this child.

Thanks to the removal of this Snark, the *next* copy of the single-channel glider recipe that comes into this child metacell will be fed into the Snarks that make up the nucleus's walls, rather than to the CN or CE subroutine loops. Those nucleus walls are still being constructed by the subroutine loops, which won't complete for roughly 2^{29} more generations, but the nucleus construction process moves at roughly the same speed as the approaching recipe, so the construction will safely complete before

the recipe gets there.

1.8.4 Generations 0.82×2^{29} to 2×2^{29}: Activation of the Southeast Child

Now both subroutine loops of the child metacell are filled and are building the repetitive Snark chains that will hold its copy of the single-channel glider recipe. When construction is done there will be 256 copies of 8 Snarks in long chains along the northwest and southeast walls of the nucleus. The sets of 8 Snarks and their associated self-destruct circuitry are not quite mirror images of each other, so different recipes are needed in the CN and CE subroutine loops.

This construction and filling of the nucleus goes on for quite a while. The subroutine storage loops are period 2^{21}, and they have to cycle 256 times each before being shut down by the period $256 \times 2^{21} = 2^{29}$ clock guns. Splitters near the storage units duplicate the recipe several times, so that construction actually proceeds on four Snark pairs simultaneously. The unit of repetition that's constructed 256 times is made up of eight gliders, offset in pairs from each other so that the simultaneous constructions don't get in each other's way.

At generation $t = 968\,843\,520 \approx 1.8046 \times 2^{29}$, the northwest wall of Snarks is complete, and a Herschel is present in the CN subroutine clock's gun that will produce the glider that makes it through all of the twenty-one semi-Snarks in the loops below (see Figure 1.34). That signal destroys a block in a syringe in that subroutine loop, so that all further recipe gliders in the loop will be harmlessly absorbed by the syringe's eater 2—no more building will occur.

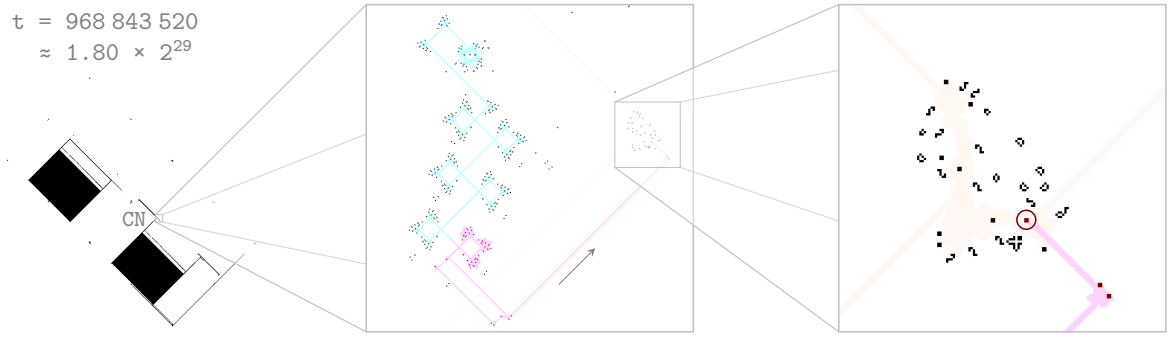


Figure 1.34: A Herschel is present in the CN clock gun that will produce the glider that makes it through all of the twenty-one semi-Snarks in the loops below. That signal destroys a block in a syringe (circled in red), so that the CN storage loop stops building the northwest wall of the nucleus. This same glider also destroys the three outermost semi-Snarks, making the clock gun 8 times faster.

A side branch of that same signal also starts the gradual process of dismantling the subroutine clock gun. This first output glider destroys three of the clock gun's semi-Snarks, making it 8 times faster, and every subsequent output glider acts similarly. The result of this is that the clock gun did nothing for a long time (2^{29} generations), and then its destruction happens faster and faster.

For example, the *next* three semi-Snarks are destroyed just 2^{26} generations later, by the Herschel that appears at $t = 1\,035\,952\,384 \approx 1.9296 \times 2^{29}$, and the clock gun vanishes completely just a bit more than 2^{24} generations after that, at $t = 1\,054\,065\,485 \approx 1.9633 \times 2^{29}$. The entirety of the CN storage loop is similarly gone just a few thousand generations later, at $t = 1\,054\,073\,568$. The CE subroutine clock and storage loop self-destruct in the exact same way shortly thereafter, and are gone as of $t = 1\,058\,486\,020 \approx 1.9716 \times 2^{29}$.

Back in time slightly at $t = 974\,368\,517 \approx 1.8149 \times 2^{29}$, after construction of the Snark walls finished but before the subroutine storage loops self-destructed, the final glider in the single-channel recipe is about to enter the child's nucleus at the location just northwest of LS (see Figure 1.35). Once this insertion is complete, the child's nucleus will be fully populated with a copy of the parent's DNA.

After the first glider in the child's nucleus makes it all the way through the long back-and-forth Snark chains for the first time, it arrives at a splitter at the south corner at $t = 1\,073\,734\,584 \approx 2.0000 \times 2^{29}$. It then uses one-time circuitry to start up the child's clock gun (see Figure 1.36), aligned

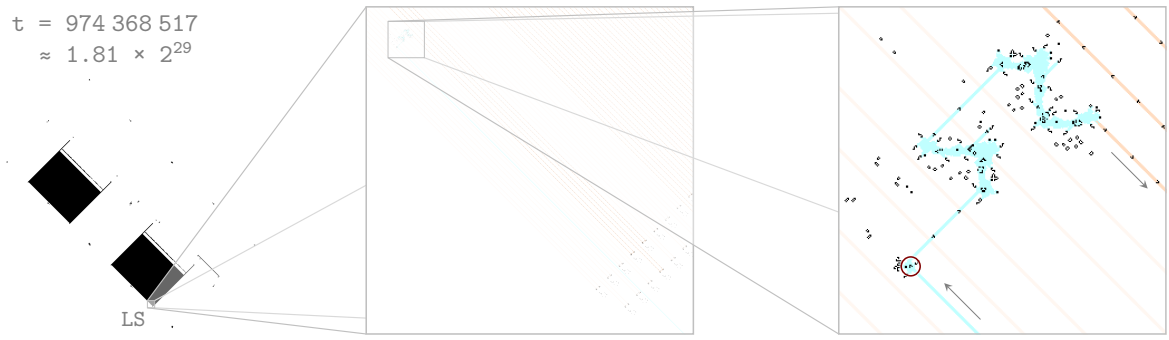


Figure 1.35: The final glider in the 0E0P's single-channel glider recipe is injected into the child's nucleus, at the location circled in red.

exactly with the timing of the parent's clock gun. However, the clock gun does not actually do anything until much later, when an eater is removed from its path.

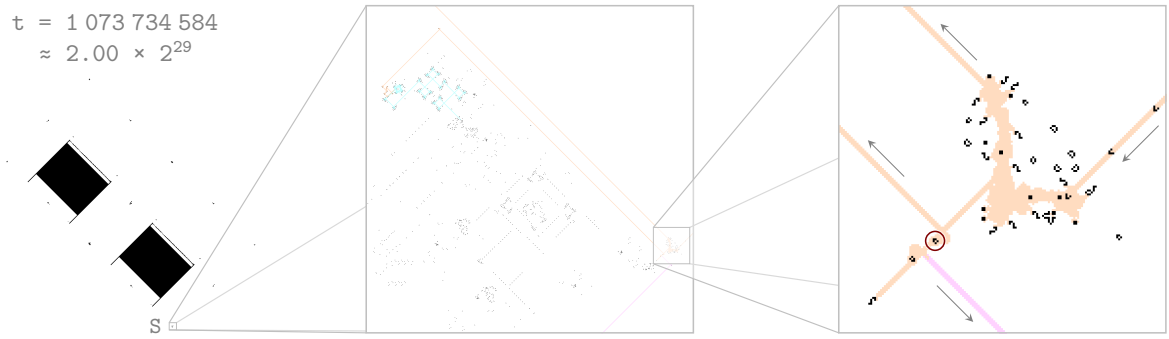


Figure 1.36: The first gliders in the child's single-channel glider recipe complete their first journey through its nucleus. The very first of these gliders hits a one-time reflector (circled in red) that redirects it to start up that child metacell's clock gun (highlighted in aqua).

At this point, the southeast child is fully constructed and simply waits until the parent metacell's other neighbors are constructed before really doing anything. The single-channel glider recipe will cycle through its nucleus several times before they are actually made to do anything of note.

1.8.5 Generations 2×2^{29} to 8×2^{29}: Construction of the Other Children

Stuff.

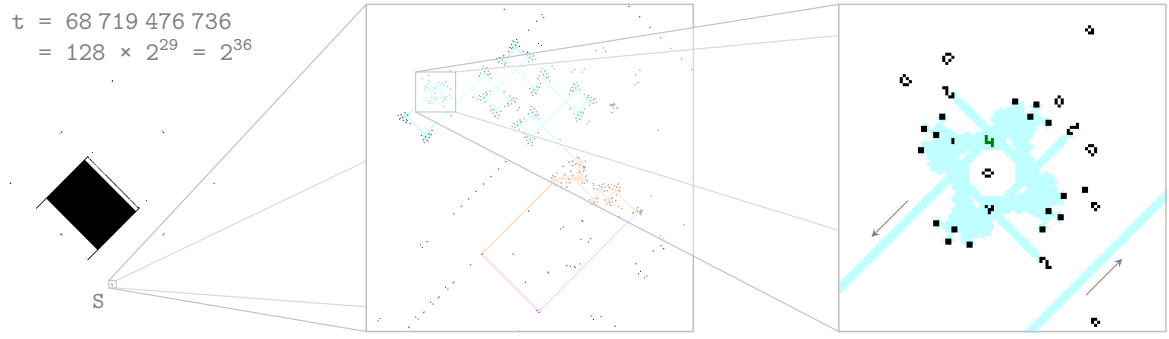


Figure 1.37: A full metageneration of the 2-state Moore-neighborhood rule has passed. In the particular rule being emulated here, a single cell lives from one generation to the next, so this snapshot is identical to the one from Figure 1.22: a Herschel is present in the clock gun which will release a glider that will destroy an eater, eventually leading to the construction of its southeast neighbor.

1.9 Notes and Historical Remarks

Many metacells—patterns of size larger than 1×1 that emulate the behavior of a single cell—were constructed prior to 0E0P. The first one was the **p5760 metacell**, which was constructed by David Bell in January 1996. This metacell is much smaller and faster than 0E0P, with a period of just 5 760 generations and a bounding box size of just 500×500 (see Figure 1.38). However, there are numerous trade-offs that make this metacell less useful:

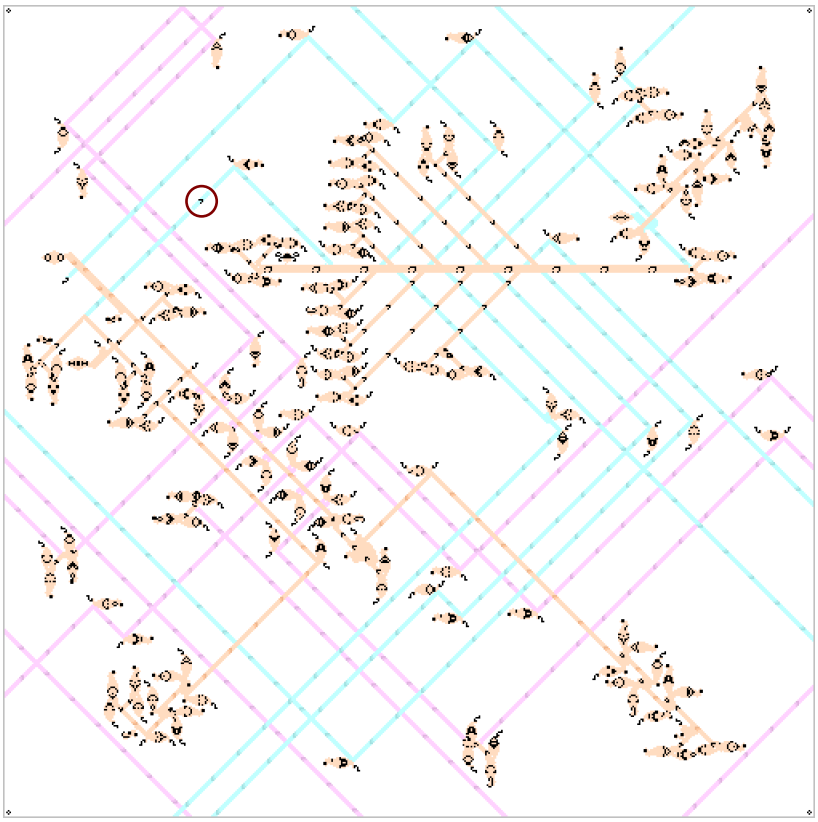


Figure 1.38: The *p5760 metacell*. Whether the cell is considered "alive" or "dead" is determined by the presence or absence of a glider at the location circled in red. That glider, if present, is duplicated 8 times and sent to its neighbors along the 8 output paths highlighted in aqua, signaling to them that it is alive. Similarly, neighboring alive cells send their signals to this one along the input paths highlighted in magenta.

1) It is hard-wired to emulate Life, and cannot easily be modified to emulate most of the 2^{512} different non-isotropic Life-like cellular automata.
2) It is not easy to tell at a glance which "cells" are alive and which are dead—it is determined by the presence or absence of a single extra glider in the cell—and thus is not interesting to look at from a far-out zoom level.
3) "Dead" cells must be placed on the Life plane, which means that, for example, spaceships cannot be emulated by this metacell unless the pattern is infinitely large.

The first two of these problems were solved by the *OTCA metapixel*, which was constructed by Brice Due from late 2005 to mid-2006. While this metacell is a bit larger and slower than the first metacell (it is 2048×2048 and has period 35 328), it can be used to emulate any of the 2^{18} different outer-totalistic Life-like cellular automata. Indeed, built into its circuitry is an easily-adjustable array of eaters that determine how many live neighboring OTCA metapixels should lead to the birth or survival of the current metapixel (see Figure **??**).

The OTCA metapixel also has the remarkable feature that its alive and dead states look, from a distance, like alive and dead cells. This feature is achieved by the "alive" version of the cell releasing 43 pairs of perpendicular lightweight spaceship streams that mutually annihilate each other, thus

partially filling in the otherwise empty center of the metacell. For example, arranging a 1×3 row of "alive" metapixels (and a suitably large "dead" array of metapixels around its edges) results in a pattern that looks and evolves like a blinker, but 2048 times as long and wide and 35328 times as slow (see Figure 1.39).

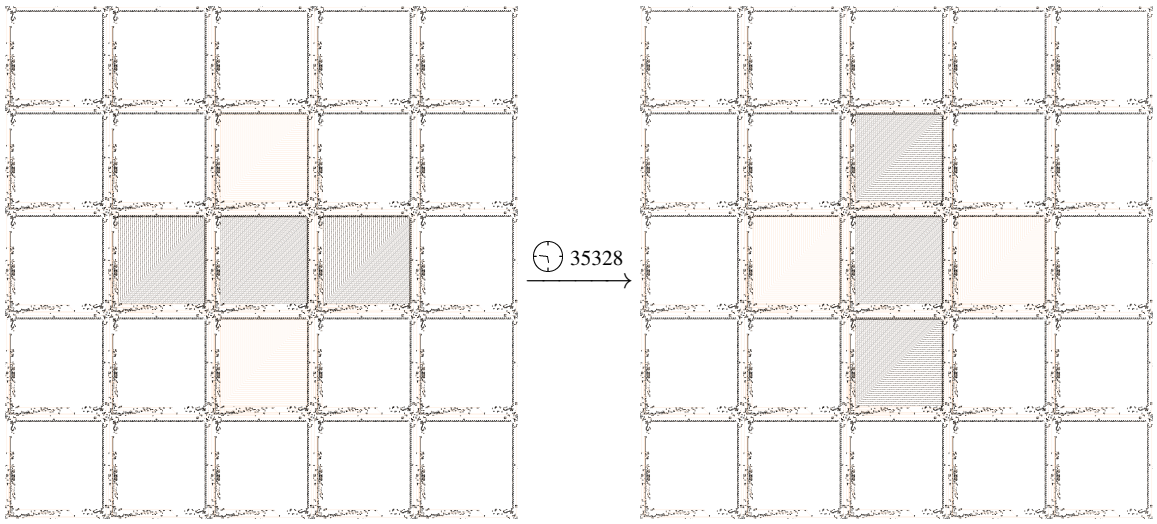


Figure 1.39: A *metablinker*: an arrangement of OTCA metapixels that emulates a blinker, but with period 2×35328.

The next notable metacell to be constructed was the *p1 megacell*, by Adam P. Goucher in 2008. This metacell was, again, larger and slower than the metacells that came before it, with a bounding box of $2^{15} \times 2^{15}$ and a period of 2^{24}. The new features this time that warranted the extra size and delay were twofold:

- This metacell was built entirely out of stable (p1) components like Herschel tracks, with the exception of a single period 2^{24} gun used to regulate its timing.[25]
- All 2^{512} non-outer-totalistic Life-like cellular automata can be emulated by this metacell, versus the 2^{18} outer-totalistic Life-like cellular automata that can be emulated by the OTCA metapixel.

Finally, Adam P. Goucher spent 2014–2018 constructing the 0E0P metapixel, which solved problem (3) described earlier—it does not require a background grid of "dead" cells to be placed on the Life plane.

Exercises

solutions to starred exercises on page ??

1.1 Recall the replicator rule `B1357/S1357` that was illustrated in Figure 1.4.

(a) [2/5] If the longest side of a pattern's bounding box is n cells long, how many generations would it take to copy itself for the first time in this rule?

(b) [5/5] Prove that every pattern in this rule really does replicate, as we claimed.
[Hint: First, prove (via induction, perhaps) that a single cell replicates. Then show that evolving a multi-cell pattern in this rule is equivalent to evolving each cell individually and XOR-ing the results together.].

1.2 Modify the rule used in Figure 1.11(b) so as to construct a (non-isotropic) 2-state cellular automaton on a square grid in which a single cell is a spaceship with speed...

(a) [2/5] $c/2$ orthogonal;

(b) [2/5] c diagonal;

(c) [2/5] $c/2$ diagonal; and

(d) [4/5] $(1,3)c/4$ or any other non-orthogonal, non-diagonal, and non-knightship slope.
[Hint: Use two generations to transform the cell into a domino, and two more generations to transform it back into a single cell.]

[25] This gun could be swapped out for a gun of another period, but periods that are multiples of 2 help the pattern run quicker under the HashLife algorithm in Life simulation software like Golly.

1.3 [3/5] Copies of the 0E0P metacell can be placed on the Life plane so as to evolve in the same way as the pattern from Figure 1.9(a).

(a) Explain why this pattern is *not* a reflectorless rotating oscillator in Life, despite being a RRO in the rule that the 0E0P is emulating.

(b) Explain how you could use even more 0E0P metacells to emulate multiple copies of the pattern from Figure 1.9(a) so as to create an actual RRO in Life.

1.4 The 0E0P metacell from Figures 1.22–1.37 is emulating the isotropic rule

```
B2-an3-eiky4aiqw5ijnr6ak
  /S012ik3-acir4kqw5acq6ack7c8.
```

(a) [2/5] Find an RRO in this rule.

(b) [3/5] Use multiple copies of the 0E0P metacell (each emulating this rule) so as to create an RRO in Life. [Hint: Be careful and recall the solution of Exercise 1.3.]

1.5 [3/5] Create a von-Neumann-neighbourhood cellular automaton on a 2D square grid with at most 8 states, where every cell dies in every generation (so it is of the type described in Section 1.2 that can be emulated by the 0E0P metacell), in which two single-cell spaceships can collide so as to create a 2-cell SMOS.
[Hint: Make a cell in one state move south, a cell in another state move east, and something else happen when they collide.]

1.6 [3/5] What is the non-isotropic rulestring (in the sense of Appendix **??**) for Conway's Game of Life?

∗**1.7** [2/5] Recall that a single glider can destroy a Snark, as long as we place some extra still lifes near it as in Figure **??**.

(a) Find a similar configuration of still lifes that is used in the 0E0P metacell so as to allow a single glider to destroy a Snark.

(b) Explain why the configuration of still lifes from part (a) might be preferable to the one from Figure **??**, despite being larger.

∗**1.8** [2/5] In Figure **??**, there is a one-time-turner that emits a glider to the south (first to the southeast, but it is then redirected to the southwest). What does this glider do, and what purpose does it serve?

Appendices and Supplements

Bibliography

[Gos84] R. Wm. Gosper. Exploiting regularities in large cellular spaces. *Physica*, 10D:75–80, 1984.

[Wol02] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

Index

R

S

T

V