

Efficient Universal Constructors in Conway’s Game of Life

Adam P. Goucher, Pavel Grankovskiy, Dave Greene, Vladan Majerech¹,
and Daniel A. N. Vargas

¹Department of Theoretical Computer Science and Mathematical Logic (KTIML), Charles University, Malostranské náměstí 25, Prague 118 00, Czech Republic

September 15, 2024

Abstract

We formalise a notion of ‘constructibility’ of configurations in Conway’s Game of Life, and prove the equivalence of several alternative definitions for constructibility. Several examples of efficient universal constructors are presented, each of which can be instructed to construct any constructible configuration by providing a description in a binary encoding. This is applied to show that every constructible configuration has a predecessor consisting of 15 gliders.

1 Introduction

The theory of universal construction was developed by John von Neumann in the late 1940s and published posthumously in [vNB66]. He introduced a 29-state cellular automaton for the explicit purpose of demonstrating universal construction. In [Cod68], Codd introduced an 8-state cellular automaton with rotational in symmetry which also supports universal construction; an explicit example of a self-replicating computer in Codd’s rule was built by Hutton in [Hut10]. By expanding the neighbourhood of cells from the 5-neighbour *von Neumann neighbourhood* to the 9-neighbour *Moore neighbourhood*, John H. Conway introduced a 2-state rule with rotational and reflectional symmetry; for the remainder of this paper, we shall operate entirely within this rule, commonly known as *Conway’s Game of Life* (CGoL in short). We refer the reader to [JG22] for a comprehensive survey of discoveries in this rule.

Moore [Moo62] and Myhill [Myh63] proved that any irreversible cellular automaton necessarily contains *Gardens of Eden*: configurations of cells which have no predecessors, and therefore cannot possibly be built by a universal constructor. In [ST22], Salo and Törmä discovered a stable configuration containing a ‘self-forcing’ subpattern which, if present at time $t + 1$, must also be present at time t ; by induction, it must have existed from time 0, and therefore cannot be built by a universal constructor.

These examples show that we must be careful in how we define ‘universal constructor’. Informally, we say that a pattern is *constructible* if and only if it can be built by colliding copies of the *glider*, the smallest spaceship in the rule; we provide a rigorous definition

in this paper. A universal constructor is then a configuration capable of building any constructible configuration; this definition excludes configurations such as the Salo-Törmä still-life whilst being very broad: every still-life of ≤ 21 cells is constructible, as are many oscillators, spaceships, and other patterns.

2 Formal definitions

Here we mathematically formalise several notions.

Definition 1. A pattern is a finite subset $P \subseteq \mathbb{Z}^2$. The elements of P are called cells.

Definition 2. Two elements $u, v \in \mathbb{Z}^2$ are adjacent if $\|u - v\|_1 = 1$.

Definition 3. The connected components of a pattern are the equivalence classes of cells under the reflexive transitive closure of the adjacency relation. A pattern is connected if and only if it has exactly one connected component.

Definition 4. Conway's Game of Life is the function $\mathcal{L}$ from the set of patterns to itself defined by the following rule:

$$\mathcal{L}(P) = \{u \in \mathbb{Z}^2 : 5 \leq \sum_{v \in P} f(u, v) \leq 7\}$$

where we define:

$$f(u, v) = \begin{cases} 1 & \text{if } u = v, \\ 2 & \text{if } u \text{ and } v \text{ are adjacent,} \\ 0 & \text{if } u \text{ and } v \text{ are non-adjacent} \end{cases}$$

Definition 5. A glider is a connected pattern G such that $|G| = 5$ and there is a nonzero vector v such that $\mathcal{L}^4(G) = \{u + v : u \in G\}$.

3 Glider syntheses

A glider synthesis consists of an infinitely-rewindable¹ arrangement of two or more gliders colliding to produce (or *synthesize*) a desired object.

Definition 6. Formally, we say that a pattern P has a glider synthesis G if the following properties hold:

- G is an ancestor of P , meaning that there exists a nonnegative integer n such that P is the n th generation of G .
- There exists an infinite sequence of patterns $G = G_0, G_1, G_2, \dots$ such that, for all $i \geq 0$, G_i is the first generation of G_{i+1} and every connected component of G_i is a glider.

¹i.e., no gliders would have previously crashed into each other

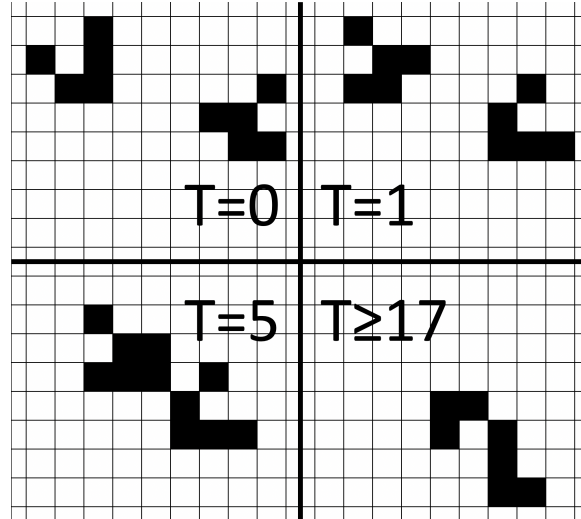


Figure 1: Example of a two-glider synthesis of a still-life called a fishhook or eater 1

A fishhook has the useful property that, when hit with a glider in a certain way, it creates another glider travelling in a perpendicular direction, leaving no residue behind. It is also possible to synthesize with gliders collections of nearby still-lives that, when hit with a glider in the right way, create two, three, or more gliders with known synchronization. These allow any glider synthesis to be given a "one-glider seed": a collection of still-lives that, when hit with a single glider in the right way, turn it into an assortment of gliders with the proper directions, relative positions, and relative timings necessary to recreate the glider synthesis in question.

The minimum number of gliders necessary to create a given object is known as its glider cost. Larger and more complicated patterns tend to have larger glider costs, with some large patterns requiring over one thousand gliders—except for the fact that it is possible to, with a fixed cost, construct (with gliders) a pattern that can recreate any glider synthesis, even a very expensive one. The mechanism, called a universal constructor, creates a one-glider seed for the given synthesis (with the instructions to create it encoded in the initial distance between the universal constructor's components), triggers it, and cleans up itself, leaving only the desired pattern; therefore, the cost of any constructible object is capped by the cost of the cheapest universal constructor.

4 Bootstrapping and self-modification

We leverage the idea of *partial construction* introduced by Buckley in [Buc08].

4.1 Binary encoding method and optimization

The sequence of arm inputs is encoded by the initial distance between two components, which allows an infinite number of glider recipes to be encoded with finite initial population and glider count. This is converted into binary by bouncing a glider-speed signal, which can be either: one or more gliders; the absence of one or more gliders in one or more (otherwise) periodic glider streams; a combination consisting of both missing gliders and extra gliders; or alternate between one or more forms, between two signal-reflectors (patterns that, upon receiving a glider signal, send another glider signal in the direction that

the incoming glider signal originated from), at least one of which is moving towards the other. For all universal constructors glider-constructed so far created in Conway’s Game of Life, one of the signal-reflectors moves at one-third of the speed of a glider, while the other signal-reflector is stationary. Some simple Doppler shift calculations inform us that if the moving signal-reflector is delayed by $2N$ ticks, any glider-speed signals sent towards it will come back N ticks later compared to if the moving signal-reflector were not delayed—assuming that the time offset is compatible with the period of the moving signal-reflector. (“Compatible with the period of the moving signal-reflector” means at minimum that N must be a multiple of 64, but certain moving signal-reflectors impose more stringent requirements.)

Without accounting for the periods of the reflectors, repeating the process indefinitely would lead to glider-speed signal delays (compared to without delaying the moving signal-reflector) of N ticks, $\frac{3}{2}N$ ticks, $\dots$, $\sum_{j=0}^{k-1} \frac{N}{2^j} = \frac{2^k-1}{2^{k-1}}N$ ticks, $\dots$ —but of course, this delay would eventually cease to be compatible with the period of the signal-reflectors, allowing them to have a different response. This different response includes both employing a different signal-reflection timing so that the timing of the glider-speed signal remains compatible with the other signal-reflector and sending different construction inputs. Now, suppose that the moving signal-reflector is delayed by $4N$ ticks instead of $2N$ ticks. This would cause the relative delay to become incompatible with a signal-reflector one reflection-pair later, which would alter the next construction input but not any construction inputs prior to it. The construction input after that could be altered without altering any prior construction inputs by delaying the moving signal-reflector by $8N$ ticks, and so on. Thus, assigning the two construction inputs to the binary digits 0 and 1 (where the appropriate assignment depends on how the glider-speed signal’s timing is adjusted to remain compatible with the signal-reflectors’ periods).

Much of the improvement in glider cost has come from improving the signal-reflectors. The original (329-glider) universal constructor used a two-engine Cordership as the moving signal-reflector ($64 \mid N$), and the stationary signal reflector ($256 \mid N$) used four p8 bouncers (one of which also helps with the construction inputs), three p8 bumpers (all of which also help with the construction inputs), two outputs of a Herschel-conduit-based glider tripler, and three barrels of a four-barreled p256 glider gun. The 15-, 16-, and 17-glider universal constructors use a glider-producing switch engine as the moving signal-reflector ($256 \mid N$) and three p256 glider streams (two of which double as the construction streams) from three more glider-producing switch engines as the stationary signal-reflector ($256 \mid N$). The relative cheapness of glider-producing switch engines (four gliders individually—but some pairs require only seven) compared to a two-engine Cordership (ten² gliders), p8 bumpers (nine gliders each plus glider-redirection costs), p8 bouncers (nine gliders each plus glider-redirection costs), the four-barreled p256 glider gun used (28 gliders), and the glider tripler in question (47 gliders) accounts for much of the glider cost improvement of universal construction.

²The 329-glider synthesis uses what is essentially an eight-glider synthesis for a two-engine Cordership with an extra boat left over, but two of those gliders could not come from infinity and therefore must be made from two infinitely-rewindable gliders each

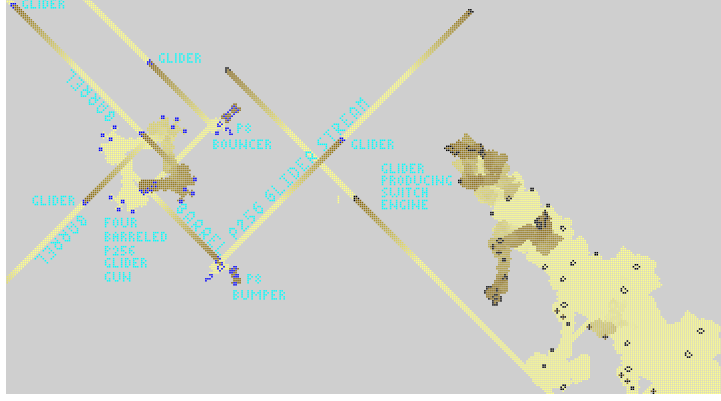


Figure 2: Some formations shown in debugging format showing how long a cell is dead and if it was alive at given moment

Total glider cost	Cost of moving signal-reflector	Cost exclusive to stationary signal-reflector	Cost shared between stationary signal-reflector and construction input	Cost exclusive to construction input	Cost of target
329	10	30	115	168	6
59	8	8	16	24	3
58	7	8	16	24	3
44	7	10	8	16	3
43	7	9	8	16	3
35	7	9	8	8	3
33	7	9	8	8	1
32	6	9	8	8	1
17	4	4	8	0	1
16	4	4	7	0	1
15	4	4	7	0	0

As the above table shows, the only larger source of glider cost decrease is the construction input, which has been reduced from a three-barrelled p256 gun with additional circuitry to a pair of glider-producing switch engines with no additional circuitry required. Part of this was due to replacing p256 guns with cheaper glider-producing switch engines, which also eliminated the need for periodic reflectors. In addition, the number of p256 streams of construction gliders was decreased from three³ to two by improvements in slow construction operations. These also allowed the cost of the target to be cheapened to one glider. These construction operations are sequences of construction inputs that emits gliders, split a target, destroy one or more extra targets, or do nothing, each beginning and ending at one of a limited number of "anchor configurations" for easy chaining, as described in the following section.

The 17-, 16-, and 15-glider versions all use the same basic circuitry: The moving signal reflector creates an (after Doppler-shifting) p256 glider stream that collides with one of the p256 construction lanes to create stationary objects (called blinkers) that then destroy the other p256 construction lane in a reaction (called a phi spark) that also destroys the p256 glider stream from the "stationary" signal reflector. When a single

³with the 44- and 43-glider versions using four

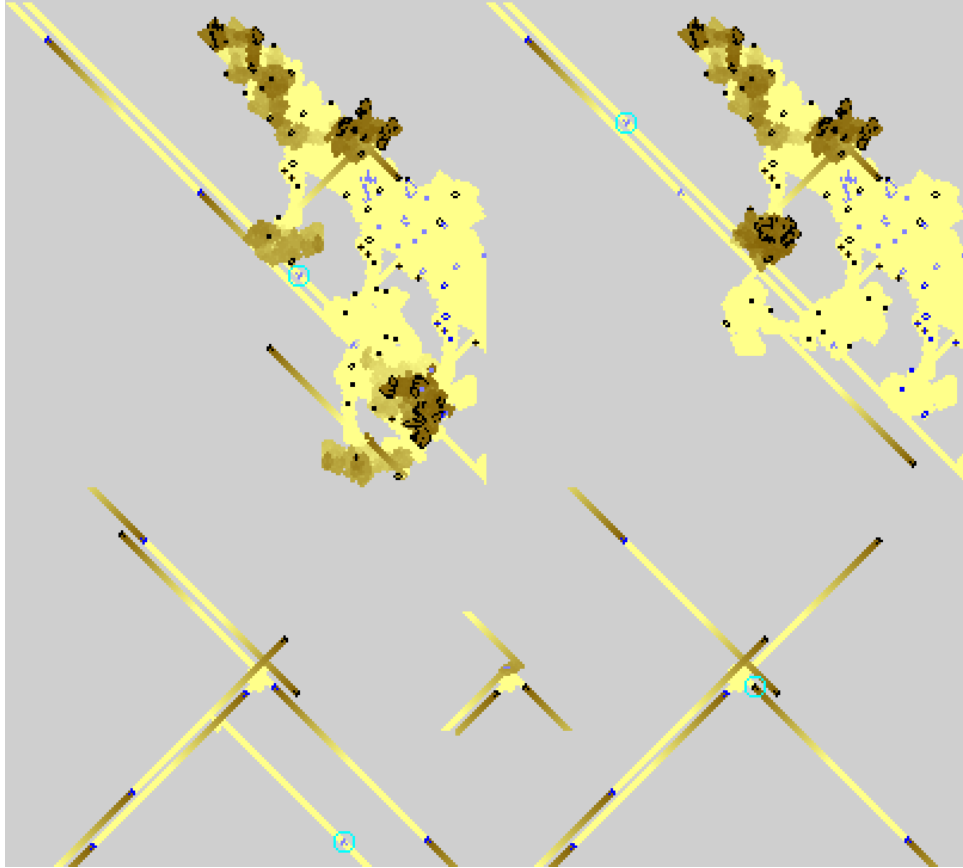


Figure 3: Signal transformations in 15-glider version: Glider signal, Missing glider signal, Glider $\rightarrow$ '1', No Signal, Missing glider $\rightarrow$ '2'. Actually signal '1' in center issues glider to NW GPSE, if the next distance corresponds to signal '1' it is reflected as a 2 bits delayed echo signal coming close to signal from the center (otherwise it shifts a block in NW trash). This echo signal modifies some trash left by SE GPSE without releasing further signals (not shown on the picture). Similarly the distance creation of missing glider releases glider SE. It just destroys one block of previous reaction (neighbour of '2' modification, and this is why first bit of the recipe must be '1').

glider from the p256 glider stream is allowed through, one of two possibilities happens, depending on the modular distance: In either case, one of the two gliders creating the blinker is destroyed, preventing said blinker from being created and thereby allowing a glider from the second construction lane to go to the construction target and allowing a single glider from the "stationary" signal reflector through to repeat the cycle. In one case, this happens because the moving signal reflector sends an extra glider that annihilates a glider from the first construction lane. In the other case, this happens due to a gap in the p256 glider stream from the moving signal reflector. In this case, instead of helping to create a blinker, the glider from the first construction lane is also allowed to proceed to the construction target (slightly ahead of the glider from the second construction lane). Thus, depending on the modular distance, either a single glider or a glider pair goes to the construction target (see figure 3). By sending certain patterns of single gliders and glider pairs, one can move the construction target, turn it into multiple targets, or do several other useful operations allowing for universal construction. The process of finding this sequences is described in the next section.

5 Binary arm

The least efficient construction arm used in universal constructor project is the arm built by glider collisions with simple still lives. The input to the arm is one of two cases (see figure 3):

'1'

- one glider directed NE. Let us denote minimal $x+y$ where an active cell of the glider appears lane 0.

'2'

- two gliders directed NE with one of them with exactly the same parameters as in the case 1 and the other one crossing the same SE lane 178 ticks earlier, using lane which is 3 full diagonals SE from the first.

The inputs are coming in the same ticks (mod256).

Universality of the arm was proven using half of traffic light (further from center on lane $x + y = 10$) as a base still life for the arm. Finding sequence of inputs

- duplicating the pattern 4 fd closer,
- pulling the pattern 3 fd,
- pushing the pattern 4 fd,
- emitting NW glider in each combination of phase (mod2) and color
- creating block sufficiently far from the input lanes
- destroying the pattern not affecting pattern 4 fd further away

Later just a blinker with center on lane $x + y = 13$ (quarter of the traffic lights) was chosen as base still life for the arm and broader set of basic sequences was found and used in the construction.

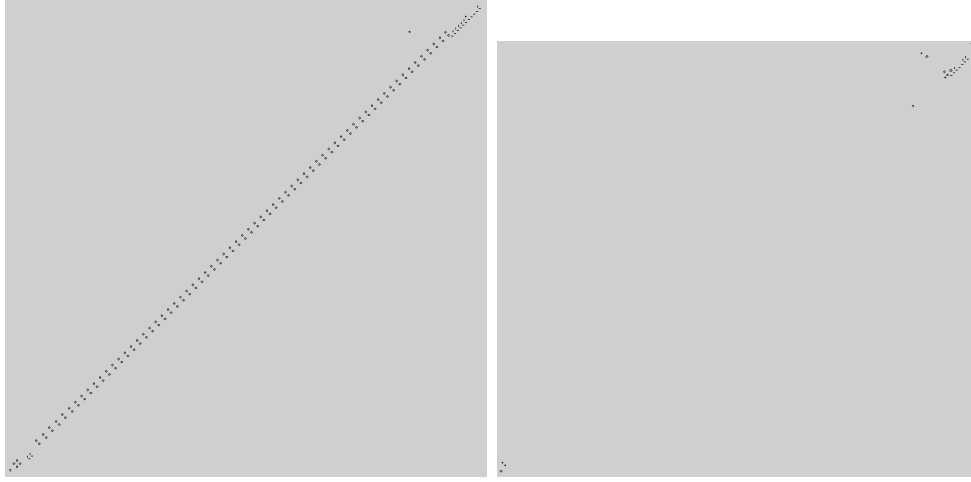


Figure 4: '1'H6T22D²26(t20T18)³(T18)⁵⁹T16T6SD⁵BS³D – stack after 19659 bits with boat as a glider turner

Figure 5: '2'T6SPDS²D²26(t20T18)³(T18)⁵⁹T16T6SD⁵BS³D – top of stack after 19823 bits, glider emission ready to hit glider splitter, NE glider will hit boat glider turner (on previous picture) and SW glider will hit the LWSS seed. LWSS-glider collision creates far enough target boat.

6 Binary arm optimization

6.1 Grammar

Doing search on CGoL configurations would be extremely inefficient, so we have defined anchored configurations (checked at time congruent to 8) and we computed sufficiently large subset of anchor to anchor bit sequences. In the original, so called blinker binary arm, the CGoL configuration was considered anchored when it contained only horizontal blinkers with centres at lane $x + y = 13$. As the arm has to create slow NW glider salvo, NW gliders do not violate CGoL configuration being anchor. First blinker arm distinguished glider phases to odd/even, but as the chosen sleeping pattern to be build was 8 periodic the need to distinguish phases of gliders (mod8) arised. The lane relative to the position of the nearest blinker is the other characteristic information of the created glider. But as duplicating the anchor 4 fd (8 hd) closer was very cheap operation and destroyal of one such anchor as well, there was importance of having all glider emission lanes (mod8) hd.

So the goal of the search of anchor to anchor configurations was to find binary sequences positioning the nearest blinker (moves) and binary sequences emitting gliders filling as many phase (mod8), lane characteristics as possible. Unfortunately some of the slots are much more expensive than others, sometimes moving the anchor to another position (mod8) hd is cheaper or necessary.

The main opportunity to improve the blinker arm was not to return to blinker only anchor after each "step", but rather to continue with naturally appearing extended set of anchors. Four anchors are closely related to the blinker one, which was named S. Those are D (the half of traffic light we started with), P consisting of S and pond with NE corner of inner block 4 fd NW from the centre of S and B consisting of S and block with SE corner 4 fd NW from the centre of S. Closely related to D is d, which is D

in opposite phase, but its standard position is shifted 1 fd NE. We have used 5 other anchors, H, h representing nearer half of a honeyfarm, (their position differs by 1 hd) and T, t representing further half of a honeyfarm (with 1 hd difference in position). Last used anchor l consists of vertical blinker with centre on lane $x + y = 1$ (relative $y - x$ position for H, h, T, t, l items can be deduced from source codes or the pictures). As a special case were chosen anchors R resp. L representing intended block emissions NW resp. SE from the input lanes, but they are not considered to be stack items.

We have decomposed all the recipes used by the blinker arm into recipes based on the new set of anchors. Recipes can be applied only conditionally as the reaction must not interact with the farther anchors. This lead to defining for each sequence minimal $y - x$ distances individual anchor types should be separated.

There is a rare case where distance to two next anchors should be checked. This most extreme case is following piece of grammar emitting glider in phase 4 and lane 18 hd above the last item of the pre-stack (first square brackets), after all the bits reached the destination the post-stack (second square brackets) replaces the pre-stack. The conditions in the curly braces show that if there are other items in natural distance 8 under the working top of the reaction, they must be SS. (This is rather efficient emission, but it fixes S's on the stack, what could become expensive to destroy.)

```
[B6d6DDSSDDB]22(18 4)21121121111111111111111112222121222[]
{:_,21T,19t,19I,23H,25h,21B,23P,21d,21D,5S}
```

Probably the most efficient emission is

```
[B6d6DSSDS]22(10 3)[]{:_,3T,3t,3I,7H,7h,5B,7P,5d,5D,5S}
```

Special care requires S anchor with respect to '1' signal as the glider misses the anchor entirely and nearest non S item is affected. In that case another condition must be met and that is required distance between last S and the first non S anchor. Similarly the other glider misses the l anchor. There are cases when '2' input gliders interact independently. When glider '1' annihilates independently, signal '2' is changed to signal '0' and some recipes accept such a signal (Similarly when glider '0' annihilates independently, signal '2' is changed to signal '1' what is natural input of a lot of recipes.)

Portion of the grammar dealing with conditioning these independent reactions (-127 means essentially the item can be ignored at all):

```
[B]0[] {I9:_,5T,5t,5I,9H,9h,7B,9P,7d,7D,7S}
[B]2[1] {:_,5T,5t,5I,9H,9h,7B,9P,7d,7D,7S}
[I]1[] {S-127:_,7T,5t,7I,9H,11h,7B,9P,7d,7D,-127S}
[I]2[0] {:_,11T,15t,7I,9H,11h,9B,19P,9d,13D,11S}
[S]0[D6d-2] {I5:_,7T,5t,3I,9H,11h,7B,9P,7d,7D,7S}
[T]0[] {I11:_,7T,5t,7I,11H,11h,7B,9P,9d,7D,7S}
```

Advantage of this conditional decomposition was that several hundred of original recipes (and others) were generated by only 230 new recipes.

6.2 Input for the search program

We want to create a pattern using a binary arm, but the input to the search program is not the final pattern. First we need to convert the pattern to a slow NW salvo. The salvo

computing program expects higher construction costs when consecutive glider lanes are far apart, so it optimizes the order of glider lanes to minimize the sum of consecutive lane distances (switching order of independent subsalvos, whenever it reduces the L0 metrics). As we are building p8 pattern (pattern with period of 8), the glider phases (mod8) are specified. But all the cells affected by the emitted glider could be p2 or even p1 and in that case using expensive (mod8) specific emission would be waste of resources. This is why we created for each glider emission alternatives. First step was to find minimal period the glider target has. But generalisation of this is not to consider the lane-phase program be the goal, but rather the effect of the given lane-phase program. Alternatives including gliders in different lanes when the collision effect is the same were precomputed and specified in the input file for the search.

There could be even generalisations allowing p2 targets in different phase when later these targets are converted to p1, but describing the dependencies would be complicated task and that would multiply the number of vertices in the implicit search graph. (We did it manually for a special 18 lanes program, but doing it for program of thousands of lanes would be unfeasible.)

As each glider of binary arm is very expensive, we have added one additional transformation to the input. The pattern to be build has some functional parts and requires some squares empty. But there can be places where the contents can be arbitrary. The computed lane-phase program during building the functional parts builds an additional contents and uses some gliders to destroy the additional contents. We have detected which additional content is not in the way of the construction and marked such cells with the same label as the lane in the lane-phase program. We then manually checked which additional cells interfered with the function of the resulting pattern and we have commented out from the input the lane-phases whose result do not interfere.

```
(g0%1) [674] (g0%1) [679]
(g1%2) [679]
(g0%2) [667] (g0%2) [668] (g1%2) [671] (g1%2) [672]
(g1%2) [667]
(g0%2) [670]
(g1%2) [667]
(g6%8) [684]
(g0%1) [729]
(g0%1) [719]
(g1%2) [720]
(g0%1) [726] -- 3123 blocks DBCA add lane
--(g0%1) [739] (g0%1) [744] -- 3124 116,491,117,491,115,492,118,492,...
```

Part of the lane program corresponding to emissions of 3213 - 3124 gliders of the nonag-nosticised salvo. Glider 3123 is responsible for destroying trash not coliding with the rest of recipe, but it is incompatible with the functional pattern (manually commented). Trash generated by glider 3124 (SE half of the honey farm with center (111,491) comment shortened) could remain in the result. See figure 9.

6.3 Templated search

Searching restricted to moves provided by the anchor to anchor grammars is more efficient than CGoL configuration search, but a lot of early decisions become important much later.

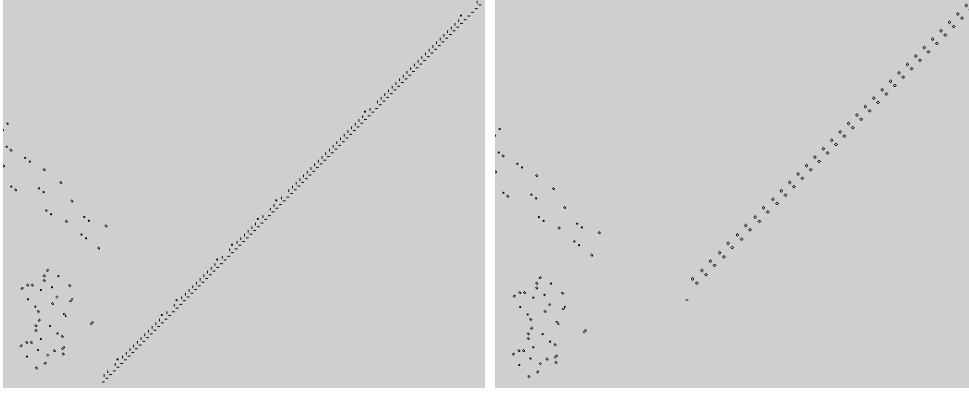


Figure 6: Stack destroyal and long pull of stack bottom (long sequence of signals '1' would follow) — almost ready to start more efficient arm

Usually for a glider emission we need few exact anchors followed by several arbitrary anchors from the $\{D, P, B, S\}$ set with distances 8 hd. Later when we require emissions with lower $y - x$, we have to reduce number of anchors in "the stack" and the exact choice of anchors become important. If we would maintain all possible choices during the search the number of states would grow exponentially quickly making the search unfeasible. Solution is to delay the choices and maintain sets of possible anchors. Each subset of the $\{D, P, B, S\}$ set gains its own identifier and it represents the set in the symbolic representation of the stack context. Whenever there is a reason to restrict the set (sequence is impossible in some cases, anchor processing is less efficient for given choice, ...), the set and therefore its representative is changed in the stack description. At the end all set sizes are reduced to 1 and position in the recipe where we have option to select the anchor is located and changed accordingly. (Input signal '1' changes D to P, P to B, B to S, and sequence '2112' changes S through P6d6, B6d6, and S6d6 to PD (we use numbers to denote exact hd distances except the default distance 8), unfortunately this fails when P is the second anchor on the stack). There is no recipe in our grammar creating anchor P immediately after B, D or P anchor. This is the only implicit constraint on the template string.

For portions of salvo, where glider lanes differ only slightly the stack based on $\{D, P, B, S\}$ set works more efficiently than using anchors from the $\{H, h, T, t, I\}$ set for arm positioning. The main problem is inefficiency of conversion of the stack to the situation with $\{D, P, B, S\}$ on top. (Except glider emissions whenever stack starts on either $\{T, t, I\}$, the portion of stack consisting of only $\{H, h, T, t, I\}$ items could be only removed. Only stacks starting either H or h could be converted to $\{D, P, B, S\}$ on top (removing H resp. h from the stack), but the corresponding bit sequence is very long (162, resp. 259 bits). On the other side, transformation from H to H6T12 is much more efficient in pulling the nearest anchor than the $\{D, P, B, S\}$ cycle, that if there is a region of above about 200 lanes where no glider emission is needed, honey farm anchors could become more efficient for the pull to the other side of the region (see figures 4 and 5). Similarly the honey farm anchors are efficient when we would need arm only much closer to the input source. We could make a long stack of honey farm anchors, create S anchor at front and destroy the honey farm anchors using only '1' signals (see figure 6).

The search builds an implicit directed leveled graph whose vertices on level ℓ are templated stack contexts after an emission of ℓ -th glider with phase-lane allowed by the input. The vertices are named by stack context with top corresponding to the grammar

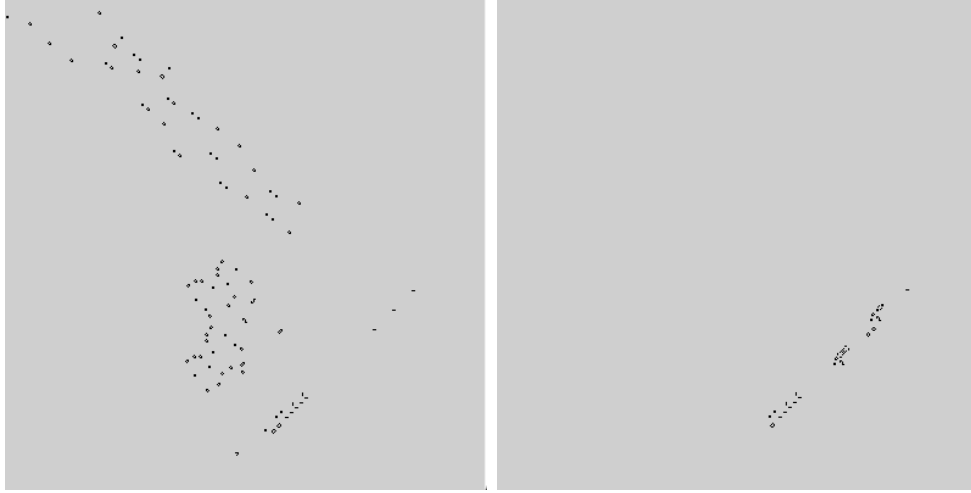


Figure 7: Redirect to more effective arm, just to destroy the stack, note the half of the honeyfarm (even the lane program we made sleeping pattern for was leave-innocent-trash optimized)

stack after the emission with bottom allowing arbitrary combination of S, B, P, D anchors. The vertex could be reached by several edges from the vertices of previous level and the cheapest path (smallest number of bits) is chosen. This could (partially) specify some anchors. The set of vertices for the level is easily obtained from the grammar where both the lane and phase (mod8) filters possible emissions the post emission stack defines the vertex, the pre emission stack defines what the subsearch should find for edges.

The most complex task in the search is to find the transformation of the stack from previous level vertex to the pre emission stack. Search using the grammar is used, with lower bounds estimating required number of bits to finish the transformation are used. For several cases the heuristics is sure with the best solution.

The first version of the program required an enthusiastic user to finish the job. The main problem was the set of emissions in the grammar did not contain all 8^2 possible combinations of phase and lane (mod8). Therefore portions of the task had to be divided manually to those compatible with the (mod8) position of the stack items. Manual planning of movements of stack top changing (mod8) position was required. The search for edge costs ended when the cost was known, but only the portion of bits leading to position heuristics said the cost of optimal emission were stored. The bits the heuristics would fill were filled later (when the edge appeared in the final recipe).

The set of emissions when phase (mod1), or (mod2) is specified contains all lanes (mod8), but emissions where exact phase (mod4) or (mod8) is required are for some lanes (mod8) unknown or not optimal and starting by an anchor pull by nonzero lanes (mod8) could be required/more efficient. Extending the set of emissions by these composite emissions improves the efficiency. Unfortunately there is still an opportunity for improvement as these composite emissions return to the original anchor position (mod8) even when starting next emission from the other position (mod8) could be more efficient.

The second version of the program used set of emissions extended by composite emissions combining stack top changing (mod8) move primary emission and stack top removal leaving stack items in the original (mod8) position. The number of considered emissions grown significantly and therefore the number of considered edges. The composite emissions typically work from bigger distance, therefore bigger portion of the stack has to be

destroyed during the stack transformation. Second version of the program was not very effective in the stack destroyal and trying these edges was time consuming. Often the optimal transformation was found during first second or two, but the search had no proof better solution does not exist even after several minutes. According to logs improvement after first minute happened so rarely we have decided to interrupt the search for the best edge after one minute, using the best result find (if any). With this workaround it seemed the result could be computed in few days. Next speedup was obtained by reducing of the set of emissions used. From emissions with the same pre-stack and post-stack and emitted glider only the shortest was selected (filtered in program) (that was definitely bug of the data obtained by compositions), but that was just a part of huge manual emission data reductions. Finally we have decided to split emission files to (mod1), (mod2), (mod4), and (mod8) portions when (mod i) portion is considered for emissions where phase is specified mod at least i . The most efficient emissions (for each lane (mod8)) are held in (mod1) portion and we are considering edges to very expensive emissions phase (mod8) specific only when phase (mod8) specific output is required. We were trying to remove from (mod i) portion only emissions dominated by other emission presented in (mod $\leq i$) files. Statistics of ever used emissions were consulted, the risk of worsened bit efficiency was outweighed by the gain in computational efficiency. The risky parts of input are those where the requested lane numbers drop significantly and therefore emissions with long prestack could become surprisingly efficient. It could be interesting experiment to use extended set of emissions for these (lane drop) cases and see the difference. Reducing the number of vertices considered on each level speeded up the computation so the result for our task could be computed within half a day. Just the final stack destroyal required a lot of manual help. Carefull debugging shown the heuristics sometimes missed better destroyal during the stack transformation (SS $\rightarrow$ S case).

In the third version of the algorithm in its first phase the longest stack bottom is found which could be transformed by additional '1' bits without further destroyal to match the target stack, then the first task is to destroy top of the stack to reach this position and following task is to build the top of the final stack. Fortunately there are only 4 really efficient transformations reducing the stack. B'2' $\rightarrow$ '1', SDBDS'212' $\rightarrow$ λ , SSPS'212' $\rightarrow$ λ , and SS'2111211121121' $\rightarrow$ S. Very often for the stack consisting of only S, B, P, D anchors (templates using them as subsets) very efficient stack reduction could be find by combination of these transformations. As each of them reduces the stack depth, the search is very narrow (each vertex is determined by remaining stack length and number of bits '1' which were additionally passed to the stack). Transposition tables are used to remember found stack top destroyals (they convert the search to dynamic programming). The stack destroyal is a bit complicated by the fact there are templates on the stack so we have to choose the most efficient substitution. The stack build is even easier using already mentioned set of transitions '1' passing through S (allowing template introduction) and transforming D $\rightarrow$ P, P $\rightarrow$ B, and B $\rightarrow$ S and '2112' transforming S $\rightarrow$ PD (through P6d6, B6d6, and S6d6) increasing the stack depth. All the emission pre stacks consists of S, B, P, D anchors (templates using them as subsets) and at most one of the prefixes T6SPD or b6d6 on the top, where b represents one of the S, B anchors. We can use SSDD'121112111111211121121' $\rightarrow$ T6SPD. As the post emission stacks in grammar could have on top of S, B, P, D anchors portion only l4, l38, l42, T28, T14, T6, H6T22, S6d6, D6d6, l16T6, h4t14, h4t22 tops, we still have to do some search using the grammar to find stack we can reduce (and as we want as short reduction as possible, we continue searching for up to a minute (shorter if we know this is the best solution)).

The third version of the program finishes its task in about 3 hours including the stack destroyals, producing a tiny bit shorter results than the second version, (but careful comparison have shown in some sections second version given better subresults).

6.4 Opportunities to reduce the final number of bits

Current stack destroyal code has a partially templated stack as the input. It tries to choose optimal anchors from the provided options, and the chosen variant is forced later to exist on input (by modifying the building portion responsible for template creation). The templates are on places we know we can select them for the minimal cost. But we could have specified some anchor what causes destroyal cost increase higher then if we would changed the specified anchor in the appropriate time. (... $S_4S_3S_2S_1$... portion destroyal is rather expensive (S's are numbered just to simplify talking about them). It could happen it originated in the stack $S_3S_2S_1$... and D instead of S_2 would be compatible with the rest of the emissions. Than replacing S_2 by D at this time ($S_3S_2 \rightarrow S \rightarrow S_3D$ would lead to ... S_4S_3DS ... portion destroyal. The additional costs to create SSDS is much smaller than additional cost of destroying SSSS rather to SSDS. Considering well timed rebuild cost might prove usefull.

Another opportunity for the improvement is in changing the bottom (mod8) for longer time then one (composite) emission. How to deal with it without exponentially increasing the size of searched graph is a question. Maybe the search processing input from the first to the last emission is not the optimal choice. Searching from the emissions requiring the longest stack (nearest emission lanes) could probably allow considering different stack bottom (mod8) positions and choose best options by some sort of dynamic programming.

Current implementation as a side effect of hash table usage with randomization of a hash function generates different results. (Emissions are considered in order depending on the hash function, and in the case of ties in the bit length different anchors could be fixed in the stack what affects stack destroyal efficiency in far future.) Often one output performs better in some section and is outperformed in another. Combining several solutions taking better sections from portions where they differ could save few bits as well. This solution crossing can be used even on places the bottom (under the effective portion of the stacks) differ. Crossing in order of decreasing stack lengths could be advantage ... and it could happen the "dreamed dynamic programming" would eliminate the usability of the crossing technique.)

Average cost of one emitted glider by the Binary arm is slightly above 84 bits. Average cost of the arm using only S anchors is slightly above 103 bits.

7 Decoder and Better Construction Arm

We need a lot of gliders to clean what glider producing switch engines (GPSE in short) left behind. This is one reason why creating a cheap and more efficient arm is important.

The plan for Decoder and Better Construction Arm (DBCA in short) in this universal constructor is to prepare Far South East cleaning, create another arm responsible for rest of the cleaning (mainly Far South West and North West) but DBCA as well. The other construction arm is responsible for building the final pattern. DBCA should also prepare central region for collision of GPSE's and redirect and delay remaining incoming bits to the other construction arm.

The idea is to create a circuit, translating incoming bits or rather their sequences as a higher level commands. The arm typically uses target block in a fixed $x - y$ coordinate but variable $x + y$ to determine $x + y$ coordinate the glider emitted SW would have. The emission mechanism must differ depending on the parity of the glider lane (color). The glider lanes differ from the block coordinate by an offset corresponding to the emission mechanism.

As our lane recipes are mostly based on p2 patterns, typically the arm allows to send a glider in each lane bigger than a constant depending on the arms position in both parities of phases. We have to be careful when we want to build a p8 pattern. The lane program should contain only phases (mod8) the arm could provide.

The instruction set of a DBCA should also contain moves of the arm target block allowing its arbitrary positioning. Natural encoding of such higher level command sets is a fixed number of instructions repeated forever where one incoming signal informs there is time to make a step and the other signal determines whether the instruction should be executed or skipped.

7.1 Encoding efficiency

We have used a lane-phase program required in earlier version of the project as a test data for the encoding efficiencies of different encodings.

Encoding	Total bits required	Bits per emission	Cycle length
$e0, o1, e1, o0, i16, d8, i4, d2, i1$	375846	9.24	9
$e0, o1, e1, o0, i16, d8, d4, d2, d1$	367008	9.02	9
$e0, o1, e1, o0, i8, d4, d2, d1$	392516	9.65	8
$e0, o1, e1, o0, i4, d2, d1$	537666	13.22	7
$d1, d2, i4, f + c, c + p_h(p)$	426444	10.48	5
$d1, d2, i4, f + c, c + p(p)$	443804	10.91	5
$d1, d2, i4, f + c, c + p, p$	467092	11.48	6
$d1, d2, i4, f + c(c + p_h)$	480924	11.82	4
$d3, i4, f + c(c + p_h)$	458920	11.28	3
$d3, i4, f + c, c + p_h(p)$	463268	11.38	4
$d3, i4, f + c, c + p(p)$	491976	12.09	4
$d3, i4, f + c, c + p, p$	558229	13.72	5

In the encoding name the individual instructions are separated by commas, action done after last command in the cycle is processed (executed or skipped) is noted in the parentheses. Block moves increasing block distance from the arm are encoded by i , moves in the opposite direction by d . The number near i resp. d denotes distance in fd. Glider emission in even resp. odd parity lane are encoded by e resp. o , the bit near e resp. o denotes parity of the glider phase (all emissions do $d1$ as a side-effect). If the instruction consists of several micro-instructions, $+$ is used to separate the micro-instructions. Glider emission corresponding to the arm inner state are encoded by f . Color inner state change is encoded by c . Parity inner state change is encoded by p . Modifier $_h$ means maintain another inner state to halve frequency of doing the micro-instruction. Order of the instructions does not affect the DBCA efficiency.

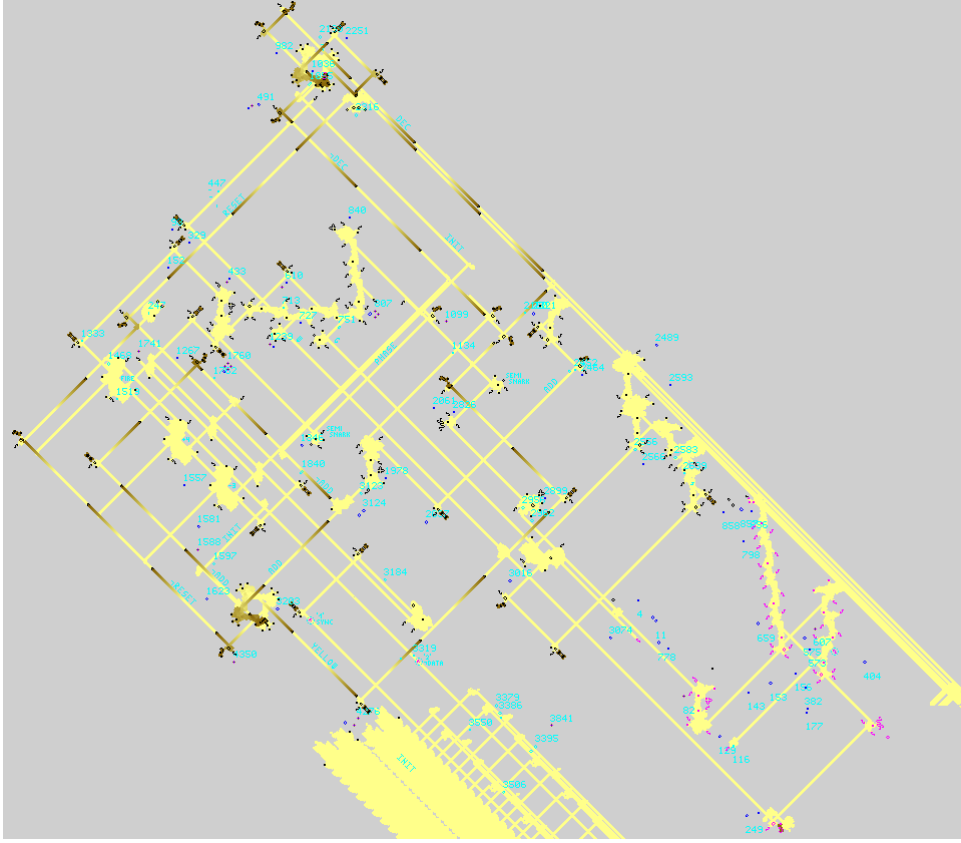


Figure 8: RiskDBCA with trash marks

Encodings with other movesets were tested as well and winners⁴ for different maximal instruction sets lengths were.

Encoding	Maximal cycle length	Bits per glider
$d3, i4, f + c(c + p_h)$	3-4	11.28
$d1, d2, i4, f + c, c + p_h(p)$	5-6	10.48
$d1, d2, d4, i8, f + c, c + p, p$	7-9	8.64

The construction costs of the DBCA is much more important than the small differences in the arm efficiency. Construction cost (number of bits) of the DBCA in the previous version was 8 times bigger then the number of bits used by the arm. The most promising encoding considering expected number of bits required together by both Binary and DBCA arms is $d3, i4, f + c(c + p_h)$. This RiskDBCA uses only 3 instructions in the cycle, but the compilation is a bit complicated by the need to consider its inner states. Preparing alternatives for the line-phase program are very important as exact phase requirement when the phase is not important could mean unnecessary delay by 2 cycles (6 bits). The alternatives in the lane choice could help as well especially with the limited move set⁵.

⁴ $i1, i2, i4, i8, d16, f + c, c + p, p$ was not tested

⁵ $d3, i4, f + c(c + p_h)$ used in portion required in the RCT15 actually used 85827 bits for 8002(8001 required) gliders (version not generaing ECCA SoD) so less then 10.73 per glider

7.2 RiskDBCA design

Sharing parts of circuit in DBCA using f instruction is must. The main heart of the circuit creates $g0$ (the same lane) pair of gliders in different parities of phases, and it emits gliders to both color emission paths. There is a frequency halver (semisnark) on all these 3 lanes meaning according to its state only one parity of glider goes through, and as the semisnarks on the color lanes are in opposite states, just one color lane would be passed. The color lane prepares the emission sleeping pattern (boat) and makes sure the glider in the phase lane reaches the correct destination (hits the boat appropriately).

There is slight complication in this design as we have to change the states of the semisnarks to prepare required emission. This requires sending glider to the semisnark, but not allowing it to continue in doing further tasks.

The solution is to use stream of gliders (we can use a p256 barrel) and let them annihilate with the gliders after they went through the semisnarks. We can reuse the heart of our circuit to hit all these semisnarks. We just need to be time (mod256) shifted from the emission glider to allow selective filtering by the 'yellow' stream of gliders. (Using p8 reflectors creates another synchronisation restriction). The only problem is we do not want to hit phase determining semisnark twice so we have to yellow filter one of the $g0$ gliders before the semisnark. As we want to change phase each other time, we have to destroy the other glider (each other time) as well, so we annihilate it through another semisnark.

The movements are realised by hitting p256 streams in a way 3 resp. 4 gliders are removed from the stream (that creates a lot of constraints to relative phases of gliders).

We have managed such design (and also thanks to improvements program transforming patterns to glider lane-phases) which required slow salvo of only 4462 gliders to build a working RiskDBCA (including positioning of starting block from starting boat). Actually the DBCA was only half-working. It was able to produce gliders of only one color (lane parity). But this was sufficient for finishing DBCA construction by self modification.

7.3 Selfmodification

We have decided to build the other color using one color restricted DBCA. That required modification of the program transforming patterns to lane salvos to support color restricted salvos. This required rebuild of primary salvo databases as filtering of color unrestricted salvos would cost a lot of time and the set of remaining salvos would rarely be able to build required pattern.

Moreover, the DBCA (and later ECCA) support both glider phases (mod2), but just 2 glider phases (mod8) so it is perfectly suited to build p2 patterns, but p8 resp p256 patterns are challenging. This was another modification we needed to do to the program transforming patterns to lane salvos. Now the allowed glider phases of each color are the part of the program input. The salvo database is mostly used for p2 patterns, p8 patterns are exceptions. The salvos for p2 patterns are internally transformed to the version corresponding to (mod8) phase restrictions. The p8 exceptions are solved by adding turners to modify the incoming glider phase. A p256 pattern is created as a p8 pattern with a prepared starting seed, where the seed expects the restricted glider phase to be started with (that was the case of one color restricted p256 DBCA as well).

8 Extreme Compression Construction Arm

The responsibility of the other construction arm is bigger than the responsibility of the DBCA, and especially its task depends on the final pattern, which could be arbitrary huge. Construction costs are reduced roughly by a factor of 8 thanks to using DBCA arm rather than the binary one. This is why we are much more careful to make the number of bits per emitted glider as small as possible. This is why the arm is named Extreme Compression Construction Arm (ECCA in short). The biggest improvement was achieved in the arm block positioning. The clear winner uses the instruction set $m4+r, m2, m1, c+p, p, (f)s+m1$, where encoding from DBCA is extended. Move in the direction corresponding to the inner state is encoded by m , starting new instruction set is encoded by r , and direction inner state change is encoded by s . When there are several micro-instructions included, the order of execution is denoted by left to right order. Of course the $+p$ near c is not important as following p could compensate it, but this variant is easier to implement.

Note the f instruction is not conditioned. Whenever we inform the arm block is in proximity of final position (no more moves by 4 is required), we can move the block by at most 3fd, choose color and phase and emit the glider (what does $d1$). We can choose to change direction of the arm block movement, but in that case we do 1 move after the direction change.

When we were thinking about the DBCA we have not considered m and s options as they seemed to be rather expensive to implement, and as the number of gliders produced by DBCA in the entire RCT15 project is 9353, improving efficiency by one bit per glider would discount the DBCA work by 9353 bits, while prolonging salvo constructing DBCA by 112 gliders would require roughly the same number of bits. The direction switching mechanism cannot be made that effectively (and it would save less than a bit per glider), so the decision not to use m , and s instructions in DBCA was appropriate.

8.1 ECCA design

The heart of ECCA is based on the heart used for the DBCA. The main difference is in the movements set. We use the fact the $d1$ micro-instruction uses the same two gliders as $p1$, o , and e micro-instructions. Each of $p1$, o , e micro-instruction differ from $d1$ by adding one well timed glider to the lane dependent on the micro-instruction. This is why move lane signal is split to $d1$ signal and $g0$ pair of gliders going through semisnark and yellow filtering to the $p1$ lane. Semisnark depending on the internal state lets through either the glider passing yellow lane or the one annihilating there. The number of moves is implemented by a glider collision destroying 4, 2 or 1 glider from the $p256$ stream of gliders whose purpose is to annihilate another $p256$ stream whose purpose is to go to move lane if allowed. The $d1$ signal actually uses this double annihilation logic to send the pair of corresponding gliders by annihilating glider which would otherwise annihilate the glider pair.

The phase switching differs from the one used in DBCA by inserting glider just before the phase semisnark, but filtering such glider after semisnark by the yellow lane remains. Direction switching uses the same logic as phase switching. Actually as the first glider from $G0$ corresponding to d instruction annihilates at the yellow line, it is sufficient to destroy the first glider before the semisnark to force the direction switch and the corresponding $m1$ instruction would still work correctly. There was local optimization in the $m4+r$ instruction using glider collision with the move lane to create an additional glider

used to perform r . Similarly the $s + m1$ instruction destroys no move glider signal by a reaction allowing emitting glider to the micro instruction s lane.

One more feature should be emphasized. The ECCA arm is emitting SW gliders, its block is pushed in NW direction. For the DBCA destroyal we require gliders in NE direction, what could be achieved by synchronised emission of a NW glider whose collision with the SW glider redirects the glider NE (kickback). The path to emit the synchronized glider have to be built and later destroyed to allow full SW functionality. We have decided to use a bumper (p8 reflector) on the kickback lane. This restricts the DBCA destroyal program to only one phase (mod256) and one color option). Local seed of destruction was added such that glider of the other parity (the only (mod8) corresponding phase) on the kickback lane clearly destroys the bumper and the eater waiting after the bumper eats all following gliders there.

8.2 Selfmodification

We decided to build one color ECCA and use selfmodification to build the other color using ECCA similarly as we did it with DBCA. In the case of DBCA the selfmodification saved a lot of bits. In the case of ECCA we have to recheck the decision improves the total bitcount as the DBCA efficiency is much closer to the ECCA efficiency. It used 1145 gliders (7538 bits) to create the pattern including the seed of destroyal using selfmodification. Seems salvo unrestricted to proximity of the Arm and not color restricted would have around 918 gliders, what would cost about 9850 bits (considering 10.73 bits per glider of the RiskDBCA), so the Selfmodification was the correct choice.

8.3 Switching

During the RCT15 execution we start with the Binary arm, continued with DBCA, then with ECCA, and finishing with an empty space. Switching from Binary arm to DBCA requires building reflectors on both signal lanes. This is why a sleeping pattern of salvo converting two separate S stack items is created. The sleeping pattern is launched by a cordership sending (synchronized) gliders to it. So the sleeping pattern of the cordership should be built to be started by a single glider. The Binary arm stack leaves one more S behind the position for the 2 reflectors, but otherwise it is repositioned before them. After the reflectors are activated the rest of the stack is destroyed. The first reflected gliders hit turners to start the p256 DBCA and let the input lanes clean. Only two obstacles are left on the $p4$ lane, so the first $p4$ instruction does just $p2$.

Switching from DBCA to ECCA is more challenging. We should redirect all remaining incoming signals to ECCA using long enough trajectory to gain required time. This is why reflectors in the far south east should be build by DBCA. DBCA has to create corderabsorbers in the far south east as well, before it redirects incoming gliders to ECCA. The solution is to build reflectors to ECCA on both signal lanes behind the reflectors to DBCA (from the remaining S) and destroy the DBCA reflectors at an appropriate time. The appropriate time is after all the DBCA program to far south east is sent (much earlier then it arrives). We have to move the arm block to far south east before. We convert some starting trash from the start of south east GPSE (it exceeds the GPSE trash lane, so it is targetable) to create glider turner (seed) there.⁶ Than we create switching seed

⁶There is actually a natural p2 restricted turner on far south east. It leaves a lot of trash after invocation. It could become cheaper to use it and destroy the trash rather to slow² creating the turner,



Figure 9: RiskECCA already started by 2 gliders coming from NE, input bits were coming from SE, first 0 bit was redirected to destroy FSE reflectors, the kickback reaction redirecting NE was already finished by other phase glider destroying bumper on extra output of Green boat hitting insertor, White portion was already built by the Green one, as all other required work of ECCA have been finished, glider coming from NE starts its self destruction.

which after invocation destroys DBCA arm block near the origin, sends synchronized glider pair, such that one is redirected by the created turner and both collide creating block exactly on the DBCA lane restoring the arm block far away. At the same time a glider is sent to destroy an eater in one color insertor. During the DBCA program sent to far south east each glider of the corresponding color would leak from the insertor. We create cheap N -gliders absorber on the leaking lane and we make sure the DBCA program on far south east would use $N + 1$ gliders of the color and the last glider is of the color. The last leaking glider corresponding to it is the only glider leaking through the N -gliders absorber. It is used to destroy the DBCA reflectors. Actually there are 2 obstacles left on the '0' signal lane, so next two '2' signals are converted to '1' signals, but this is absolutely OK as the ECCA arm block should be repositioned to destroy DBCA and a lot of '1' signals is required to do so.

Thanks to agnosticization of the lane programs we have a lot of options to select color of the glider in the salvo. This is why we can choose N from interval of about 100 options all giving the optimal bit length of the program. One block and one eater are sufficient for the N -gliders absorber, shortening their distance by 19fd decreases N by 26.

Switching from ECCA to empty requires destroying the ECCA and its reflectors. The reflectors on the signal lane could be destroyed by ECCA together with DBCA as all ECCA bits are already reflected. To destroy reflectors on far south east the (extra) first '0' signal coming to ECCA is turned back to far south east where it hits seed of destruction of the reflectors (and the DBCA arm block). To destroy ECCA the seed of destruction in far south east sends back a glider which hits seed of destruction of the ECCA. Seeds of destruction are created at the same time as the pattern to destroy.

8.4 Cleaning

RCT15 should leave only the final pattern seed and glider which will hit the seed. Everything else should be destroyed. DBCA after the other color is started destroys irregular remnants of the initial collision of gliders emitted by GPSE's. Then it prepares central site for GPSE's collision to finish in a known clear state. It fixes the last two bits of the bit program.

DBCA have to destroy all irregularities from the start of GPSE on the far south east so only an almost periodic pattern with period 32fd remains on southeast GPSE track. There are irregularities at the signal creating positions, depending on the bit created, echo glider and '2' neighbour glider. Cordership traveling south east can convert an 8fd periodic pattern near its trajectory to another 8fd periodic pattern. Beam search positioning such corderships found a salvo of 39 corderships able to convert this almost periodic pattern to (periodic) empty space. Corderabsorbers destroying these corderships are required. What DBCA should prepare on far south east (ECCA cannot cheaply reach this direction).

ECCA destroys the DBCA and all the reflectors in the signal lanes. The ECCA was designed for incoming signals with the same phase (mod256), but rather far from each other. As the time distance of the incoming bits halves each bit, the spacing is sufficient at start, but the final bits are problematic. Fortunately $m4 + r$ instruction is fast enough to process the last bits. This fixes the last three signals to be '1' (after the last glider is issued). This leaves ECCA in a known state so the seed of destroyal could count on it (we can fix direction after the last glider issued).

but this part uses only 159 bits, so if there would be an improvement, it would be negligible.

The arm block of the ECCA shifts a bit by final $m4 + r$ instructions, this is why decision was made to destroy it from south east direction where the position is known. This is an additional job of the seed of ECCA destroyal to send a glider there.

ECCA has to destroy irregularities caused by collision of GPSE's. After cleaning the central region it has to sent the already mentioned corderships to destroy the south east GPSE track.

ECCA has to destroy irregularities in the far south west from the GPSE's start there. This leaves south west track in a periodic pattern with period 32fd. Salvo of 18 corderships is sufficient to convert it to (periodic) empty space. Corderabsorbers destroying these corderships are required. ECCA will create absorbers before sending the corderfeet there. The program in far south west is a bit inefficient (slow²) as we have to create turner whenever we want to sent glider in north west or south east direction there (what is required as the GPSE trash prevents us from hitting the site directly).

ECCA has also to destroy irregularities in far north west from the start of GPSE there. This leaves north west track in an almost periodic pattern with period 32fd. There are irregularities caused by 2 gliders coming from the initial glider collision, and from gliders from '1' signal going through the epicenter. The irregularities have two forms depending on the distance corresponding to the next signal. Salvo of 23 corderships can convert this pattern to (periodic) empty space. Corderabsorbers destroying these corderships are required. The last corderabsorber is modified to issue glider back to the epicenter (in a known lane). This would be the last living part left from the initial 15 gliders upto the pattern seed.

The pattern seed has to be built by ECCA near the epicenter such that this glider will start the pattern.

To work on far north west we should reposition the arm there. We create an arm block there during the cleanup of the GPSE's collision. We are lucky the accessible part of the starting remnants forms a one time turner so we have to sent a synchronized pair of gliders to north west such that the turned one collides with the other making block on the lane where the ECCA arm block should be.

After the pattern seed is created. The near arm block is destroyed and the far north west program (creating the absorbers) sent. To destroy the near arm block, we should have an accessible target such that we can make an appropriate turner there (in an arbitrary distance from the ECCA). Trash agnosticise for the lane pattern program could be used there if such a target is not provided.

8.5 Additional shortening

We have discovered, the cleaning of FSW neednot be done by a lot of slow² gliders and the free lines in the periodic pattern could be used for the purposes. Another beam search was used which found salvo of 159 gliders destroying the nonperiodicity. It used just 3 slow² gliders. Even the corderabsorbers on FSW neednot be build by slow² gliders, and the free lines in the periodic pattern could be used. There could be probably small improvement in this part as the database of salvos used in pslmake is not optimized for use of lane restricted salvos, but the salvo uses just 685 gliders, so the cleanup and corderabsorbers on FSW require just 5752 bits.

We have created new program looking for seeds of destroyal. The original methology was adding still lives to keep reaction going choosing among still lives such stabilising the reaction. Only p2 patterns were considered stable for these purposes. New program

accepts longer periods and allows escaping gliders and we consider them to be mostly advantageous. We consider the pattern to be split to regions which have to be solved almost independently, it seems that the choice of the incoming glider is very important and we optimize it. We try to convert each region to generalized one time splitters allowing few escaping gliders (even zero). We stop the seed of destroyal calculation, when it seems more advantageous to hit it with a new fresh glider than prolonging the reaction. The program is not ready to connect the regions by additional generalised splitters to finish the construction. That has to be done manually. Collection of cheap splitters/turners organised according to distinct criterias is helpfull for such purposes. (Dealing with p8 patterns requires correct color and p8 phase as well, otherwise we have to chose cheap among p2 or p1 versions. It's hard to organize turners considering reaction envelopes, program running simulations would help.) On several places we have modified the original ECCA proposal by shifting its components while preserving functionality. Sometimes it allowed us to connect output glider directly to required input lane without need of turners. This would be difficult to code. We have used excess gliders from the non-reset lane as a cheap source of required gliders reducing number of required extra splitters, we have rather used cheap turners.

During the RCT15 project even the Cordership seed was simplified. There is still opportunity to reduce the corderfleets cost by using new cordership construction. Originally the recipe required 94.8 gliders pre cordership (SE average depending on the relative positions of the previous ship). Current Cordership seed requires 84.8 gliders pre cordership (SE average). There are 80 corderships launched by ECCA. This leads to estimated improvement about 5600 bits.

8.6 Summary

We have shown the 15 gliders are sufficient, but their initial distance should be astromonic. The original proof used pattern dependent part multiplied by constant of size roughly $2^{1560000}$. We have reduced the pattern dependent part slightly by improving ECCA efficiency and we have reduced the multiplication constant size to roughly 2^{573013} . We do not expect the exponent can be significantly more reduced.

References

- [Buc08] W. R. Buckley. Computational ontogeny. *Biological Theory*, 2008.
- [Cod68] E. F. Codd. *Cellular automata*. New York: Academic Press, 1968.
- [Hut10] T. J. Hutton. Codd's self-replicating computer. *Artificial Life*, 2010.
- [JG22] Nathaniel Johnston and Dave Greene. *Conway's Game of Life: Mathematics and Construction*. Self-published, 2022.
- [Moo62] E. F. Moore. Machine models of self-reproduction. 1962.
- [Myh63] J. Myhill. The converse of Moore's Garden-of-Eden theorem. 1963.
- [ST22] Ville Salo and Ilkka Törmä. What can oracles teach us about the ultimate fate of life? *International Colloquium on Automata, Languages and Programming*, 2022.

- [vNB66] John von Neumann and A. W. Burks. *Theory of self-reproducing automata*. Urbana, University of Illinois Press, 1966.