

Elementary Cellular Automata as Multiplicative

- Automata
- Daniel W. McKinley ¹
- 1 Independent Researcher, USA

Summary

-8

9

10

11

12

13

Elementary cellular automata (ECA) are a set of simple binary programs in the form of truth tables called Wolfram codes that produce complex output when done repeatedly in parallel, and quaternions are frequently used to represent 3D space and its rotations in computer graphics. Both are well-studied subjects, this Java library puts them together in a new way. This project changes classical additive cellular automata into multiplicative automata (Wolfram, 2002, p. 886) via permutations, hypercomplex numbers, and pointer arrays. Valid solutions extend the binary ECA to complex numbers, produce a vector field, make an algebraic polynomial, and generate some very interesting fractals.

Statement of Need

Very loosely analogous to De Morgan's law in Boolean algebra, the main algorithm produces 15 several multiplicative versions of any given standard additive binary Wolfram code up to 32 16 bits and is written to support user supplied complex 1-D input at row 0 with choice of type 17 of multiplication tables and partial product tables among other parameters. It produces an 18 algebraic polynomial and complex vector field output for any given Wolfram code, and the 19 hypercomplex 5-factor identity solution allows for the complex extension of any binary cellular 20 automata. The Cayley-Dickson and Fano construction libraries may be of value to the open 21 source community as well. 22

There are other cellular automata implementations, Mathematica (Inc., n.d.), CellPyLib 23 (Antunes, 2021), a JOSS Python project from three years ago, and others. This is not designed 24 to replace those awesome general purpose utilities, it's focused on the set of Wolfram code 25 operations. The GUI is designed to show enough to conclude that the math works and give 26 a rough idea of aggregate behavior over parameters and the algorithm code is designed to 27 be able to split off and be plugged in somewhere else. There are useful things you can build 28 on it directly or indirectly, like making Bloch spheres out layers of complex number output, 20 Fourier analysis, 2D automata, a complex version of the prime number automata (Wolfram, 30 2002, p. 640), Gray code and full group theory morphisms of multiplication tables and paths, 31 and making N-D ellipses out of the paths through the multiplication tables, that are clear 32 directions to go in but subject to a different set of decisions like application-specific tech debt 33

and potential translation to C++ or Python and out of scope of this paper. 34

McKinley. (2024). Elementary Cellular Automata as Multiplicative Automata. Journal of Open Source Software, 0(0), 6968. https://doi.org/10. 1 xxxxxx/draft.

DOI: 10.xxxxx/draft

Software

- Review C²
- Repository ☑
- Archive 🗗

Editor: ☑

Submitted:	08 July 2024
Published:	unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.€ International License (CC BY 4.0).



Functions

³⁶ Hypercomplex unit vector implementation

	Negative sign bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Complex	2's place	1	i	-1	-i												о
Quatemions	4's place	1	i	j	k	-1	-i	-j	-k								
Octonions	8's place 💌	1	e1	e2	e3	e4	e5	e6	e7	-1	-e1	-e2	-e3	-e4	-e5	-e6	-e7

The Cayley-Dickson (CD) and Fano support classes are discussed in greater detail in the 38 readme and the documentation, they along with the Galois class provide sets of multiplication 39 tables to be compared with cellular automata. The CD multiplication implementation 40 permutes the steps of splitting and recombining hypercomplex numbers to increase the scope 41 of the CD equation, (a,b)x(c,d) = (ac - d * b, da + bc*), where * is the conjugate. It 42 verifies itself by producing the vector product of two symmetric groups of its degree operating 43 on the up/down recursion factoradics when interacting with other CD multiplications and is 44 the permutations of the unit vector bit layers excluding the negative sign. The Fano library 45 octonions produce a triplet that is a linear match to the CD octonions as triplets{0} when the 46 up and down recursion factoradics are equal, and produce the triplet set of John Baez's Fano 47 plane as triplets{10}. (Baez, 2001). 48

49

37

- ⁵⁰ The main algorithm uses a set of permutations operating on cellular automata input, each
- ⁵¹ permutation permuting the neighborhood, becoming a factor, with four kinds of multiplications.
- $_{52}$ The multiplication tables are input as 2D but used as N-D, where N=numFactors.



	Multiplications A	Multiplications B	Multiplications C	Multiplications D
Туре	Hypercomplex or finite, brute-force of all the permutations of that number of factors	Cartesian product summed by a hypercomplex or finite partial product table	Complex product	Permutation composition
Size	Wolfram code length = L	Size of neighborhood, log2(L)	Size of neighborhood, log2(L)	Size of neighborhood, log2(L)
Function	Validates permutation group, reproducing the Wolfram code as a pointer array	Applies a valid solution to a user given complex neighborhood	Like B, but does the normalization before the multiplication	Orders the cell's neighborhood vector from (B), post multiplication, pre normalization
Scope	Entire Wolfram code, every possible binary neighborhood	Single given input neighborhood	Single given input neighborhood	Single given input neighborhood
Produces	Set of permutations that changes the additive automata to multiplicative, with the given multiplication table	Polynomial	Output visually similar to B	Vector
Data type	Binary	Complex	Complex	Discrete permutation
Base 2 sum of neighborhood	Construction of factors, pre multiplication	Normalization, post multiplication	Construction of factors, pre multiplication	n/a
N-th root in normalization	n/a	N = size of neighborhood	N = number of factors	n/a

- 55 Multiplications A, additive to multiplicative
- ⁵⁶ r = specific Wolfram code

- ⁵⁷ n = binary neighborhood = 1*columnZero* + 2*columnOne*+ 4*columnTwo...2*^{(col-}
- ⁵⁸ *umn*)columnCol, points to its value in r
- $_{\scriptscriptstyle 59}$ $\,$ h = hypercomplex unit vector from binary
- $_{\scriptscriptstyle 60}$ $\,$ H = inverse of h, binary value from hypercomplex unit vector
- $_{61}$ p = a permutation of the neighborhood
- ⁶² using hypercomplex multiplication, a valid permutation set produces:
- $\label{eq:constraint} {}_{\rm 63} \quad {\rm WolframCode}(r,\,n) = {\rm WolframCode}(r,\,H(h(p(n))\,*\,h(p(n))\,*\,h(p(n))\,\,...\,\,numFactors),\,though$
- ⁶⁴ n may or may not equal H(...)
- ⁶⁵ WolframCode(r, $H(h(p(n)) * h(p(n)) * h(p(n)) \dots$ numFactors)) is a pointer array that always
- ⁶⁶ points to an equal value (0,1) within WolframCode(r, _)
- $_{67}$ each h(p(n)) in a valid solution is a factor template in the multiplication table for all values of



its axis 68

73

The first set of multiplications, column A, brute forces all possible sets of permutations on all 69

possible binary neighborhoods of the Wolfram code. A permutation in the set rearranges the 70

- columns of the input neighborhood, these become a set of factors. A valid set of permutations 71
- is one that, for all possible input neighborhoods, the set of constructed factors using the 72
 - permuted neighborhoods always multiplies out to a value that points to an equal value within
- the Wolfram code. The set of multiplication results is a pointer array that reproduces the 74
- original Wolfram code for every possible binary neighborhood. 75
- Identity solutions of 5 factors using all zero permutations exist for Wolfram codes up to 32 bits 76
- in this library using hypercomplex numbers, XOR and Galois addition and more bits require 77
- more factors in increments of four. Galois multiplication takes a mix of numbers of factors 78
- to get the identity multiplication result array according to their, there is a function in the 79
- GaloisField class that provides it. The factors constructed are a loose diagonal through the 80
- multidimensional multiplication table, starting at the origin and ending at the opposite corner 81 82
- while zig-zagging. The path lengths of each factor and the result are included in ValidSolution
- results. 83

05

Permutations of 3 bit neighborhoods 84

Permutation: 0, [0, 1, 2, 3, 4, 5, 6, 7]
Permutation: 1, [0, 1, 4, 5, 2, 3, 6, 7]
Permutation: 2, [0, 2, 1, 3, 4, 6, 5, 7]
Permutation: 3, [0, 4, 1, 5, 2, 6, 3, 7]
Permutation: 4, [0, 2, 4, 6, 1, 3, 5, 7]
Permutation: 5, [0, 4, 2, 6, 1, 5, 3, 7]

Flattened path through a six dimensional multiplication table 86





- Multiplications B and C apply a valid solution from the first set of multiplications to any given 90
- individual neighborhood with binary, non-negative real, and complex values. Multiplication B is 91 the Cartesian product of the permuted neighborhoods, using a closed partial product table to 92
- generate a polynomial. Multiplication C does the binary sum of complex neighborhood, then
- 93 multiplies as complex. Both B and C take the n-th root of the result, with n = numColumns94

McKinley. (2024). Elementary Cellular Automata as Multiplicative Automata. Journal of Open Source Software, 0(0), 6968. https://doi.org/10. 4 xxxxxx/draft.



- and n = numFactors, respectively. Multiplications B and C both include a binary weighted 95
- sum of the neighborhood, same as the construction of the factors from A, though B and C 96
- use complex. B, as part of the normalization and C as the construction. Multiplication C is 97
- the permutation composition product. B, just before the normalization is a neighborhood of 98
- multiplication results, with each column of it being a unit vector coefficient. This multiplication 99
- result neighborhood is permuted by the inverse of the permutation composition product to 100
- properly order the output vector. There are a couple of normalization parameters and a hybrid 101
- multiplicative-additive output option that are discussed more in the documentation. 102

ECA rule 54 Multiplication Table to use Permuted Cay Specific solution to use 0 Degree: 2 = quaternions, 3 = octonions, etc., if applicable 2 Number of factors to use 5 Number of factors in the ECA, 1 row = 3 bit neighborhood, 2 rows = 5 bit neighborhood 1 Partial product table, size = places x places Galois addition Keeps functions from running longer than the user want, in seconds 30 the calculate button produces all solutions for the chosen parameters 1 this button re-randomizes and displays the ECA rule with the particular solution number chosen 1 Deep search using above parameters 1 Width of random input 200	vley-Dickson
Multiplication Table to use Permuted Ca Specific solution to use 0 Degree: 2 = quaternions, 3 = octonions, etc., if applicable 2 Number of factors to use 5 Number of factors to use 1 Partial product table, size = places x places Galois addition Keeps functions from running longer than the user want, in seconds 30 The calculate button produces all solutions for the chosen parameters 1 Uhis button re-randomizes and displays the ECA rule with the particular solution number chosen 1 Deep search using above parameters 1 Width of random input 200 5 Number of factors in logic gate search 5 Logic gate, AND = 8, OR = 14, XOR = 6, etc 6: XOR Logic gate solution: 2 Which multiplication table to use XOR Partial product table Galois addition	/ley-Dickson
Specific solution to use 0 Degree: 2 = quaternions, 3 = octonions, etc., if applicable 2 Number of factors to use 5 Number of rows in the ECA, 1 row = 3 bit neighborhood, 2 rows = 5 bit neighborhood 1 Partial product table, size = places x places Galois addition Keeps functions from running longer than the user want, in seconds 30 the calculate button produces all solutions for the chosen parameters	
Degree: 2 - quaternions, 3 - octonions, etc., if applicable 2 Number of factors to use 5 Number of rows in the ECA, 1 row - 3 bit neighborhood, 2 rows = 5 bit neighborhood 1 Partial product table, size = places x places Galois additio Keeps functions from running longer than the user want, in seconds 30 the calculate button produces all solutions for the chosen parameters 5 this button re-randomizes and displays the ECA rule with the particular solution number chosen 0 Deep search using above parameters 5 Width of random input 200 5 Number of factors in logic gate search 5 Logic gate, AND = 6, OR = 14, XOR = 6, etc 6: XOR Logic gate solution: 5 Which multiplication table to use XOR Partial product table Galois additio	
Number of factors to use 5 Number of rows in the ECA, 1 row = 3 bit neighborhood, 2 rows = 5 bit neighborhood 1 Partial product table, size = places x places Galois additio Keeps functions from running longer than the user want, in seconds 30 The calculate button produces all solutions for the chosen parameters 30 This button re-randomizes and displays the ECA rule with the particular solution number chosen 0 Deep search using above parameters 1 Width of random input 200 5 Number of factors in logic gate search 5 Logic gate, AND = 8, OR = 14, XOR = 6, etc 6: XOR Uoigic gate solution: XOR Which multiplication table to use XOR	
Number of rows in the ECA, 1 row = 3 bit neighborhood, 2 rows = 5 bit neighborhood 1 Partial product table, size = places x places Galois additio Keeps functions from running longer than the user want, in seconds 30 the calculate button produces all solutions for the chosen parameters 30 this button re-randomizes and displays the ECA rule with the particular solution number chosen 0 Deep search using above parameters 30 Width of random input 200 5 Logic gate, AND = 8, OR = 14, XOR = 6, etc 6: XOR Logic gate solution: XOR Which multiplication table to use XOR Partial product table Galois additio	
Partial product table, size = places x places Galois additio Keeps functions from running longer than the user want, in seconds 30 the calculate button produces all solutions for the chosen parameters 30 this button re-randomizes and displays the ECA rule with the particular solution number chosen 0 Deep search using above parameters 30 Width of random input 200 5 Logic gate, AND = 8, OR = 14, XOR = 6, etc 6: XOR Logic gate solution: XOR Which multiplication table to use XOR Partial product table Galois additio	
Keeps functions from running longer than the user want, in seconds 30 the calculate button produces all solutions for the chosen parameters 30 this button re-randomizes and displays the ECA rule with the particular solution number chosen 30 Deep search using above parameters 30 Width of random input 200 5 Logic gate, AND = 8, OR = 14, XOR = 6, etc 6: XOR Logic gate solution: XOR Which multiplication table to use XOR Partial product table Galois addition	n, XOR, 3x3
the calculate button produces all solutions for the chosen parameters this button re-randomizes and displays the ECA rule with the particular solution number chosen Deep search using above parameters Width of random input 200 Number of factors in logic gate search Logic gate, AND = 8, OR = 14, XOR = 6, etc Logic gate solution: Which multiplication table to use Which multiplication table to use Partial product table Galois additio	
this button re-randomizes and displays the ECA rule with the particular solution number chosen Deep search using above parameters Width of random input 200 Number of factors in logic gate search Logic gate, AND = 8, OR = 14, XOR = 6, etc Logic gate solution: Which multiplication table to use XOR Partial product table Galois additio	Refresh
Deep search using above parameters Width of random input 200 Number of factors in logic gate search Logic gate, AND = 0, OR = 14, XOR = 6, etc Logic gate solution: Which multiplication table to use XOR Partial product table Galois addition	Display specific solution
Width of random input 200 5 Number of factors in logic gate search 5 Logic gate, AND = 8, OR = 14, XOR = 6, etc 6: XOR Logic gate solution: 6: XOR Which multiplication table to use XOR Partial product table Galois additio	Start deep search
With of random input 200 5 Number of factors in logic gate search 5 Logic gate, AND = 8, OR = 14, XOR = 6, etc 6: XOR Logic gate solution: 0 Which multiplication table to use XOR Partial product table Galois addition	
Number of factors in logic gate search 5 Logic gate, AND = 8, OR = 14, XOR = 6, etc 6: XOR Logic gate solution: 2 Which multiplication table to use XOR Partial product table Galois addition	· · · · · · · · · · · · · · · · · · ·
Logic gate, AND = 8, OR = 14, XOR = 6, etc 6: XOR Logic gate solution: Which multiplication table to use XOR Partial product table Galois additio	
Logic gate solution: XOR Which multiplication table to use XOR Partial enduct table Galois additio	
Which multiplication table to use XOR Partial product table Galois additio	
Partial product table Galois additio	
	n, XOR, 2x2
Refresh logic gate solutions	Refresh
Display specific logic gate solution	Display specific solution
Search all logic gates for solutions and crossreference gates that have solutions in common	Deep logic gate search
Table Display Degree, 2 = quaternions, 3 = octonions, 4 = sedonions	
Cayley-Dickson permutation number, (cdz,), down in recursion	
Cayley-Dickson permutation= number, (, cdo), up in recursion	
Fano plane octonions 0	
Galois Field, Prime 2	
Galois Field, Power 1	
Length of permutations 4	
Refresh permuted Cayley-Dickson solutions	Refresh
Display tables with above parameters	Display specific tables
Compare Fano-generated octonions with permuted Cayley-Dickson octonions	Fano/CD Compare
Compare permuted CD with permuted CD	
Picks a random Wolfram code with 5 factors, identity solution	CD v CD

ECA 54, binary and non negative real







ValidSolution	-
Wolfram code: [0, 1, 1, 0, 1, 1, 0, 0]	
Permutation: 0 Permuted Axis: [0, 1, 2, 3, 4, 5, 6, 7]	
times	
Permutation: U Permuted Axis: [U, 1, 2, 3, 4, 5, 6, 7]	
Immes	
fermulation: 0 Fermulae Axis, [0, 1, 2, 3, 4, 5, 6, 7]	
Bermutation 0. Permutad Avia: 10.1.2.3.4.5.6.71	
firmation of remated was, [0, 1, 2, 3, 4, 5, 6, 1]	
Permutation: 0 Permuted Axis: 10.1.2.3.4.5.6.71	
times	
Equals: [0, 1, 2, 3, 4, 5, 6, 7]	
Apply Wolfram code to multiplication result	
Equals: [0, 1, 1, 0, 1, 1, 0, 0]	
Original Wolfram code: [0, 1, 1, 0, 1, 1, 0, 0]	
Permutation composition product: 0, inverse: 0	
Multiplication table two: 0	
Multiplication table type, o	
[0, 1, 2, 3, 4, 3, 5, 1] [1, 4, 7, 5, 6, 3, 6]	
12 3 4 5 6 7 0 1	
[3 6 1 4 7 2 5 0]	
4.5.6.7.0.1.2.3	
6. 0. 3. 6. 1. 4. 7. 2	
[6, 7, 0, 1, 2, 3, 4, 5]	
[7, 2, 5, 0, 3, 6, 1, 4]	
numFactors: 5 numBits: 3	
$\frac{1}{4}(2^{A}-5)^{A}(2^{A}-5)$	
$5^{((a+q))^{(b+q)}}(x^{(a+q)}) + 5^{((a+q))^{(b+q)}}(x^{(a+q)})$	
$5^{+}((a^{4})^{+}(b^{4})^{+}(c^{0}0)) + 10^{+}((a^{3})^{+}(b^{0})^{+}(c^{2})) + 30^{+}((a^{2})^{+}(b^{2})^{+}(c^{1})) + 5^{+}((a^{4})^{+}(b^{4})^{+}(c^{0}0)) + 20^{+}((a^{4})^{+}(b^{4})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(b^{4})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(b^{4})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(b^{4})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(b^{4})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(b^{4})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(c^{3})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(c^{3})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(c^{3})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(c^{3})^{+}(c^{3})) + 20^{+}(a^{4})^{+}(c^{3})^{+}(c^{3})) + 20^{+}(c^{3})^{+}(c^{3})^{+}(c^{3})) + 20^{+}(c^{3})^{+}(c^{3})) + 20^{+}(c^{3})^{+}(c^{3})) + 20^{+}(c^{3})^{+}(c^{3})) + 20^{+}(c^{3})^{+}(c^{3})) + 20^{+}(c^{3})^{+}(c^{3})) + 20^{+}(c^{3})^{+}(c^{3})) + 20^{+}(c^{3})) + 20^{+}(c^{3})^{+}(c^{3})) + 20^{+}(c^{3})) + 20^{+}(c^{3}))$	
10*((a^0)*(b^3)*(c^2)) + 1*((a^0)*(b^0)*(c^5))	
$5^{((a^{4})^{(b^{0})^{(c^{1})} + 10^{((a^{3})^{(b^{2})^{(c^{0})} + 30^{((a^{2})^{(b^{1})^{(c^{2})} + 20^{((a^{1})^{(b^{3})^{(c^{1})} + 5^{((a^{1})^{(b^{0})^{(c^{1})} + 5^{((a^{1})^{(b^{0})^{(c^{1})} + 5^{((a^{1})^{(b^{1})^{(c^{1})} + 5^{((a^{1})^{(c^{1})} + 5^{((a^{1})} + 5^{((a^{1})^{(c^{1})} + 5^{((a^{1})} + 5^{(a^{1})} + 5^{((a^{1})^{(c^{1})} + 5^{((a^{1})^{(c^{1})} + 5^{((a^{1})^{(c^{1})} + 5^{((a^{1})^{(c^{1})} + 5^{((a^{1})^{(c^{1})} + 5^{((a^{1})^{(a^{1})} + 5^{(a^{1})} $	
1*((a^0)*(b^5)*(c^0)) + 10*((a^0)*(b^2)*(c^3))	
ECA 54, solution output, complex	





References

- Antunes, L. M. (2021). CellPyLib: A python library for working with cellular automata. *Journal* of Open Source Software, 6(67), 3608. https://doi.org/10.21105/joss.03608
- Baez, J. (2001). *The octonions* (10.1090/S0273-0979-01-00934-X). Bulletin of the American Mathematical Society.
- 120 Inc., W. R. (n.d.). Mathematica, version 14.0. https://www.wolfram.com/mathematica
- 121 Wolfram, S. (2002). A new kind of science. Wolfram Media. ISBN: 1579550088