Recovery of global symmetries in a 't Hooftian universe

Alexandre Furtado Neto*

ORCID 0000-0001-9435-6566

UNESP Alumnus

Abstract

All fundamental Planck scale symmetries are restored on a global level when a new charge is postulated in a finite, closed, Euclidean discrete space. Gravity emerges as a residual effect of the electromagnetic force in this scenario, resulting in a deterministic toy universe driven by a single input parameter. The model is developed using a constructive approach. Randomness is identified using a Chaitin argument. Aleph_0 definite value is tied to the size of the universe. This is not an interpretation of Quantum Mechanics, but a deeper attempt to describe nature.

PACS numbers: 12.10.Kt

Keywords: cellular automaton, nonlocality, emerging gravity, unification, cardinality

1 Introduction

Wheeler [1] coined the aphorism 'it from bit'. With this, he meant that anything physical, any *it*, derives its existence from discrete binary choices, or *bits*. This gives support to the notion that information has an ontological nature. The concept implies that physics, particularly quantum physics, isn't really about reality, but just our best description of what we observe.

In this regard, cellular automata (CAs) are mathematical idealizations of physical systems in which space and time, an evolution parameter, are discrete. Their attractiveness comes from the notion that simple rules can lead to very complex behavior, tending to long and interesting evolutions.

An alternative representation of the universe is developed in this work using the cellular automaton paradigm. The theme has been explored for a long time (see [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] for example). However, these studies generally remain in the abstract realm or present very limited models. Here, a full $4+1$ core specification is posited. Although it is a qualitative analysis for the time being, the model is amenable to immediate computational investigation.

*alexandre.com@yahoo.com

Just classical logic and plain integer math, along with a hint of topology, are used in the dynamics, making up a constructive approach.

This initiative was driven by the difficulty of unifying the two pillars of physics, quantum mechanics and general relativity, in a supposed quantum gravity. Such difficulty, in the author's view, is due to the introduction of imaginary numbers both in Schrodinger's equation and in the spacetime metric.

It will be clear that foundationally this is not an interpretation of Quantum Mechanics, but a deeper attempt to describe nature.

Finally, the reader will be delighted when he sees that purely intuitive arguments are used, leaving out obscure or abstract concepts.

2 Space and time

2.1 The register

Each cell contains the same amount of information characterized by a bit string formatted as shown in the table below.

Name	Type	Symbol	Obs.
active	B	<i>on</i>	Cell role (stable/draft)
Physical properties			
charge	B6	<i>ch</i>	Bits d, c_2, c_1, c_0, w, q
momentum	V	<i>p</i>	
spin	V	<i>s</i>	
frequency	U	<i>f</i>	
affinity	U	<i>a</i>	Used to group bubbles in packets
Wavefront			
lifetime	U	<i>t</i>	
direction	U3	<i>dir</i>	
origin	V	<i>o</i>	
synchronism	U	<i>sync</i>	
sine gen.	S	<i>u, v</i>	<i>v</i> carries the sinusoidal phase
Superluminal variables			
flash	U	<i>flash</i>	
pole	V	<i>pole</i>	
Footprint			
propensity	V	<i>vp0</i>	
Auxiliary data			
code	U4	<i>code</i>	Interaction code
noise	U	<i>n</i>	Interaction test
floor	U	<i>floor</i>	Sublattice index

where U means unsigned integer, S signed integer, V three-dimensional integer vector and B a boolean. This structure is called a *register*.

2.2 The lattice

A collection of $2L$ coexisting $SIDE^3$ -sized Euclidean lattices each, closed on themselves as three-dimensional tori, represents an absolute inertial reference frame. Each cell contains a register and are synchronized by a common time $t \in \mathbb{N}$, that is, they pulsate in unison, exchanging information with its nine neighbors only (six in dimensions xyz , two in dimension u and one in dimension v). This exchange is done alternately, so time homogeneity is recovered at each two clock ticks, configuring a *cellular automaton*.

The single input parameter $SIDE$ was estimated from the size of the observable universe at a fraction $(1/2^{59})^1$ of the Planck scale as

¹The exact granularity of the universal fabric will be obtained when it is verified that the model

$$\begin{aligned} SIDE &= 2^{order}, \\ &\approx 4.78 \times 10^{80}, \end{aligned}$$

where $order = 268$. The arithmetic uses $SIDE$ as module.

The torus is the simplest topological structure capable of holding a 3D universe and charge quantization. The finitude of the universe is a necessary condition to reach this quantization (Smith [13]). Cosmological observations favor a multi-connected rather than a simply connected universe (see Lachieze-Rey and Luminet [14]).

The space is endowed with memory, which records the visit of the particles. This contributes to the appearance of self-interference, as in the case of the double slit experiment. This scheme was originally conceived by Sciarretta in [11]. The main idea is to leave a copy of the visitor's momentum on the lattice, which decays following the rule

$$p[t+1] = p[t] \left(1 - \left(\frac{p_0}{t} \right)^2 \right),$$

being eventually passed to the evolving bubble, which in turn decays as

$$p[t+1] = p[t] \left(1 - \frac{1}{2t} \right),$$

where p_0 is the momentum since the last transfer to the particle and t is the lifetime of the particle.

Taking all contributions into account in a typical tabletop experiment, after many repetitions, this value converges to the known QM probabilities involving a square root and a space-dependent sine function. Note that these expressions must be carefully adapted for integer arithmetic.

3 Bubbles

Most properties in the registers spread to the expanding cells (see table above).

A *bubble* is an expanding spherical wavefront of information organized on the registers. Whenever vector $\mathbf{p}$ aligns with vector $\mathbf{o}$, such a perturbation is cast. The von Neumann neighborhood D_{irs} is explored in a Case, Rajan and Shende algorithm from [15]. There they show that the accommodation of the wavefront takes $2D$ clock ticks (one light step, *LIGHT*), where D is the cube diagonal, so bubbles expand with the same speed c , increasing $|\mathbf{o}|$ in the occupied registers. In some cases, concurrent access to a common cell is attempted, being *per se* an undecidable problem. A tree algorithm is then Just classical logic and plain integer math, along with a hint of topology, are used in the dynamics, composing a constructive approach used to overcome this limitation and select a path. Incidentally, this direction index behaves like a modulo 6 random number, being the basis for decision making in interactions (see Chaitin [17] arguing the intrinsic randomness of mathematics). Only time *LIGHT* has physical meaning, the time tick is only for low-level sync.

A bubble is eventually reissued from a point calculated on its surface by resetting the origin vector ($\mathbf{o} = \mathbf{0}$) and/or aligning vectors $\mathbf{o}$ and $\mathbf{p}$ when interacting with other bubbles.

Each bubble lives in a separate 3d-space L_i , $i = \{1...2L\}$ to avoid propagation issues. F bubbles with a common affinity $a_j = A$, $j = \{1...F\}$, have frequency $f_j = F$, their

generates the same number of particles present in the universe. This provisional value was estimated using four times the Eddington number, thus taking into account electrons as well as antiparticles.

de Broglie frequency. If two such bubbles with some or all opposite charges and having $\mathbf{o}_1 = \mathbf{o}_2$ (that is, superposing) is called a *pair*.

The bubbles also have a ϕ sine phase coupled to f , achieved by an integer-only recursive oscillator (see Turner [18]). Moreover, they leave a footprint of their passage every time they are re-emitted, consisting of the vector $\mathbf{o}$, vector $\mathbf{p}$ and affinity a . They allow self-interference.

During wavefront accommodation the value of *noise* is updated using the selected direction index ($Dirs[index]$) and latter compared with $|\phi|$, which indicates whether a bubble-bubble interaction is legal — this produces the observed harmonic behavior at the ensemble level.

Charges are represented by bits. The new *dualitat* charge d was introduced to preserve symmetries on a global level. The *electric* charge q is associated as ever with attraction/repulsion. The three *color* charges c_2, c_1, c_0 enforce the strong force structure. Let signature be $sig = c_2 + c_1 + c_0$. A bubble is matter M if $sig < 2$, or antimatter $\overline{M}$, otherwise. Also, it is neutral N , if $sig = 0$ or anti-neutral $\overline{N}$, if $sig = 3$. The definition $color = 2^2c_2 + 2^1c_1 + 2^0c_0$ will be used latter. On the other hand, the *weak* charge w , or *chirality*, is associated with congruence. Since the universe is closed, the addition of the dualitat charge is not an *ad hoc* solution, but actually guarantees this sense of congruence.

Vector $\mathbf{p}$ is related to motion direction, while vector $\mathbf{s}$ to spatial rotation. I will refer to them as *momentum* and *spin* for convenience. The cell on the surface of the bubble pointed by vector $\mathbf{p}$ from its own center is defined as its *pole*.

Finally, a loose association between the terms *bubble* and (unit of) *energy* is assumed, so energy is a conserved property, since bubbles are never created or destroyed. Vector $\mathbf{p}$ of each bubble never changes, so momentum is a globally conserved quantity also.

4 Interactions

4.1 Overview

Interactions are evaluated at the last tick of the wavefront accommodation. Prior to interaction, the frequency and noise variables of the bubbles are updated. For instance, pairs are classified according to their charge content.

When a bubble is reissued after interacting with another bubble and, in addition, their poles coincide, all bubbles with the same affinity are also reissued, characterizing an ontological *collapse*. The possibility of the collapsing parts having a space-like separation is supported by the fact that the phase of accommodation of the wavefront lasts $2D$ ticks, sufficient for the information to propagate to all regions of the lattice.

There are four possible results for the two interacting bubbles: *i*) they cross each other just changing their phases; *ii*) both are re-emitted from the contact point, *cp* (annihilation); *iii*) they are re-emitted from their respective poles (cohesion); *iv*) one is re-emitted from the *cp*, while the other from the parallel transported pole (inertia). Figure 1 depicts these cases.

4.2 Basic cycle

The detailed algorithmic description of all interaction rules is contained in the Appendix. Below I will make a brief description of those rules.

Half of the cells are active, while the other half are passive, alternating roles in time. The basic loop consists of the following steps

Figure 1: General interaction types.

Four types of interactions can occur when the surfaces of the expanding bubbles cross each other: *a)* phase only; *b)* reissue of the two bubbles; *c)* collapse; *d)* parallel transport (inertia). The red ball is the momentum cell, the arrow is the momentum vector. The cyan ball is the parallel transported point. See Section 4.1 for details.

- **Pairing**

Data is copied from draft cells to the stable cells.

- **Update**

Draft cells are shifted in the v dimension and compared with their stable counterparts until a complete loop of $SIDE^2$ steps. Properties such as the frequency and pair classification are updated.

- **Interact**

After having data being copied back to the stable cells, again, the draft cells are shifted in the v dimension and compared for possible interactions. Besides using the sinusoidal phase, all possibilities of charge content and overlapping are considered to complete the interaction detection, being finally filtered by the selection rules of empirical evidences, being enforced or suppressed.

- **Expand**

All bubbles expand one light step in their respective xyz sub-lattice. When momentum $\mathbf{p}$ completes a light step, it releases a new perturbation wavefront from its old position in the form of a *messenger*, the carrier of the static forces. The sinusoidal signal is also evaluated during this phase. Simultaneously, a superluminal signal, the *flash*, spreads at the maximum speed, helping to enforce the collapse mechanism.

4.3 Initial state

Let S be the number of states resultant of the combination of all register values, all positions in the lattice and all bubbles L . From any state k , the system eventually enters an endless period loop T_k with definite net Shannon entropy. The real world corresponds to one of these loops with $T_R \ll S$.

Among the countless possibilities, I choose the following platonic solution for the initial state problem. A total of $L \equiv SIDE^2$ bubbles are uniformly distributed on the $z = 0$ plane of the lattice at start up, the *hologram*. Since $SIDE \bmod 3 = 1$, one single row of the hologram has all $s_x = s_y = 0$, $s_z = \pm SIDE/2$ and $d = s_z \geq 0$, inducing one topologically

trapped magnetic monopole, per sector, in line with Smith [13]. The momentum vectors, in turn, receive the value $\mathbf{p} = \mathbf{k} \times \mathbf{s}$, where $\mathbf{k}$ is the z direction.

We are done with the axiomatic part by now. In the remainder of the paper, the arguments give support to the choices made and compose a minimalist interpretation, carefully avoiding unnecessary speculation.

4.4 Evolution

The system evolves through this *unphysical era* until it reaches a stationary sequence of states, the very very long *physical cycle* of period T_R , or Poincaré cycle. During the unphysical era, most bubbles are annihilated, creating a vacuum, both in *Orbis*, the sector with $d = 0$, or in the *Dark Sector*, with $d = 1$. The remaining bubbles within each sector, segregate on islands of same size with positive or negative charges (charge quanta) due to the aforementioned monopole. Antimatter tends to migrate to the Dark Sector due to inter sector interactions. Observe that masses from Orbis are not seen from the Dark Sector and vice versa. Clearly, the two sectors coexist spatially.

4.5 Packets

Each region of space contains a homogeneous distribution of electrically neutral superposing bubbles (zero-point energy, or a vacuum), basically photons and Z particles, but also the charged W bosons, whether completely or partially formed. Some of these bubbles are recruited via the messenger interaction above, being dynamically added to a local *packet* of same affinity bubbles. If the packet contains colored or neutral non-superposing bubbles, different pairs will be captured from the vacuum.

Meanwhile, other bubbles wander away from the packet (dissipation) such that, when combined with the inherent non-linearity of the underlying system, an equilibrium situation is reached including many *propellers* (see definition below), configuring in this way the *rest mass* of the packet. In other words, we have self-organization and entanglement². The non-equilibrium caused by nearby charges or masses can change the motion of the packet, which in turn changes the effective mass.

Packets possessing a quantized set of non-superposing bubbles will have a collective “fermionic” character, while the pairs overlaid with anti-aligned spins give a packet a “bosonic” character, distinguished by their charge balance. The exception are the neutrino fragments, formed as NN (ν) or $\bar{N}\bar{N}$ ($\bar{\nu}$) with their aligned spins, which cannot form pairs as above, and therefore also present a “fermionic” character. In other words, if the packet includes a (quantized) population of non-overlapping bubbles with identical charge q then we have a *fermion*. Rather, if the population is formed by equal net weak charge, we have a neutral massive *boson*, and if the pairs also have non trivial electric balance then we have a charged weak boson. If, diversely, color is involved and no overlapping, then we have a quark fragment. Finally, if these bubbles are combined in pairs with the same non-neutral color, then we have a gluon fragment. The exception to the rule is the photon, which is an expading shell ‘packet’.

Ephemeral resonance states, including radial vibration modes³ (see Itô [19]) induced by the presence of neutrinos (which also help to conserve angular momentum), are also possible, adding to the formation of a mass spectrum.

The *Hofer effect* is the expected tendency for all spins of a packet to align radially either inward or outward (spin up/down), as predicted in Hofer [20]. In that work, there

²A recent study supports this view (see Zhi-Bo Yang et al. [16]).

³Empirically in three generations of leptons and quarks.

is an explanation of how magnetic effects emerge from symmetry breaking of this spherical pattern, thereby supporting the Stern-Gerlach experiment.

4.6 Long range energy exchange

A pure *photon* γ is a multipair ($f = 2n$, $n > 0$) where each pair has all its charges in opposition. A photon fragment having the same affinity of a packet is dubbed a *propeller*.

The long range behavior of photons is due to the absence of other bubbles with the same a value to allow an inertial interaction (not a propeller), interacting instead either as a light-matter scattering or a refraction process above.

It may happen that two bubbles in a packet, when interacting as above, are re-emitted from the same point, that is, their poles coincide, so a pair is formed. These coincidences may repeat so that overlapping of many bubbles having a common a value becomes possible. If they move away stimulated by another bubble-bubble process, a photon is released — spontaneous emission being a special case for faint fields. In other words, photons are normally released in an atomic electronic decay, shaped by the spherical harmonics originating from charge quantization. This is the main explanation for the ubiquitous presence of all sort of quanta. The primary quantized quantity, is truly the electric charge.

Photons also come similarly from *bremssstrahlung*. In this scenario, the spectrum is continuous since it has nothing to do with spherical harmonics.

The messenger *versus* bubble interaction also implies that photons may lose components while traveling (photon aging) thereby supporting General Relativity observations and the CMB.

4.7 Weak decay

No randomness is involved in the calculation of the interactions, so the model is in fact deterministic, but the ubiquitous presence of weak charges imply in an apparently random decay of particles. The W^- and Z bosons, even if not completely formed (virtual particles?), are responsible for it.

4.8 Higher order particle structures

When reorganizing after a collapse, bosons and fermions can be formed in a variety of ways, spins can be flipped etc. With all these ingredients, bound states of the strong and electromagnetic forces are a natural consequence.

5 Discussion

G. 'tHooft [12] shows that systems of this type can be associated to a permutation operator and, therefore, can be mapped to a large Hilbert space. Thus, at least in principle, all the machinery of operator mechanics, in particular the Schrödinger equation, can be used to analyze them — the point here is that the system certainly presents a quantum behavior. In this context, Born's rule arises naturally without the need to be postulated, bringing with it quantum probabilities. As a consequence, violation of Bell-type inequalities are expected, induced by the intrinsic non-locality. Perhaps new analysis tools should be devised to explore the model in search of physics beyond the Standard Model.

The model is non-local under the light-time basis, though being strictly local, but no signaling is possible at any classical limit. An intuitive argument allows us to associate

SIDE to the much sought for cardinality of natural numbers $\aleph_0$ (see Asperó and Schindler [21]) — the maximum number with physical meaning in the universe is just *SIDE*.

Last but not least, a gravity-like dynamics emerges as a residual effect of the electromagnetic force (see also Assis [22]) — the halos formed in the sea of messengers around the masses caused by the absorption of messengers (perceived as not shielding because of the relatively low number of messengers absorbed) causes an imbalance in the distribution of momentum, resulting in an always attractive phenomenon, or gravity.

We are, therefore, facing an ontological⁴ economic framework on a subplankian scale. It promises to be robust at high momentum transmitted to the package / particle as in nature. The huge number of bubbles forming the particles — indeed a mini universe each — gives material support to superpositions and qubits. These rules can be refined, given sufficient computational power and programming support⁵, and evaluated if they in fact allow building a predictive, *bona fide*, theory.

Funding and competing interests

“The author declares that no funds, grants, or other support were received during the preparation of this manuscript.”

“The author has no relevant financial or non-financial interests to disclose.”

References

- [1] Wheeler, J.A. *Information, physics, quantum: the search for links*, In Complexity, entropy and the physics of information (ed. W. Zurek). Reading, MA: Addison-Wesley (1990).
- [2] Zuse, K. *Rechnender raum*, Elektronische Datenverarbeitung, **8**, 336-344 (1967).
- [3] Feynman, R. *The character of the physical law*, MIT, ISBN 0 262 56003 8 (1967).
- [4] Gardner, M. *The fantastic combinations of John Conway's new solitaire game "life"*, Sci. Am. **223**, 120-123 (1970).
- [5] Minsky M. *Cellular vet al.acuum*, Int. J. Theor. Phys. **21**: 537-551 (1982).
- [6] Margolus N. *Universal cellular automata based on the collisions of soft spheres*, Adamatzky A. (eds) Collision-Based Computing. Springer, London (2002).
- [7] Wolfram, S. *A new kind of science*, Wolfram Media, {23-60}, 112, and 865-866 (2002).
- [8] Fredkin E. *An introduction to digital philosophy*. Int. J. Theor. Phys. **42** (2): 189–247 (2003).
- [9] Elze, H.T. *Action principle for cellular automata and the linearity of quantum mechanics*, Phys. Rev. A **89** 012111 (2014).
- [10] H.-T Elze *Qubit exchange interactions from permutations of classical bits*, Int. J. Quant. Info., **17** (08), 1941003 (2019).

⁴Ontology is actually an always receding rule marking the frontier of the unfathomable.

⁵An implementation under development is accessible in Reference [23]

- [11] Sciarretta, A. *A local-realistic model of quantum mechanics based on a discrete space-time*, Found. Phys. **48** (1), 60–91, (2018).
- [12] 't Hooft, G. *The cellular automaton interpretation of quantum mechanics*, In: van Beijeren, H., et al. (eds.) *Fundamental Theories of Physics*. Springer, Berlin (2016).
- [13] Smith, W.D. *Charge quantization, the topology and 3-dimensionality of the universe, and abolishing monopoles*, <https://www.semanticscholar.org> (2000).
- [14] Lachieze-Rey M., and J.P. Luminet *Cosmic topology*, Phys. Rept. **254** (1995).
- [15] Case, J., D. Rajan, and A. Shende *Spherical wave front generation in lattice computers*, Int. J. Comput. Inf. (1994).
- [16] Yang, Zhi-Bo et al. *Entanglement emerges from dissipation-structured quantum self-organization*, arXiv:2109.12315v1 (2021).
- [17] G. J. Chaitin, *Algorithmic Information Theory*, in IBM Journal of Research and Development, **21**, no. 4, pp. 350-359 (1977).
- [18] C. Turner, *Recursive discrete time sinusoidal oscillators*, IEEE Signal Processing Mag, p103-111 (2003).
- [19] Itô, D. *Cohesive force of electron and Nambu's mass formula*, Prog. Theor. Phys. **47**, no. 3 (1972).
- [20] Hofer, W.A. *Elements of physics for the 21st century*, J. Phys.: Conf. Ser. **504**, 012014 (2014).
- [21] Asperó, D., R. Schindler *Martin's Maximum implies Woodin's axiom (*)*. Annals of Mathematics, **193** (3), 793-835 (2021).
- [22] Assis, A.K.T. *Deriving gravitation from electromagnetism*, Can. J. Phys. **70**, 330-340 (1992).
- [23] Furtado Neto, A. *It from bit - a concrete attempt*, GitHub repository, <https://github.com/automaton3d/automaton>, (2021).

Appendix: source code

Here are gathered all the rules of information exchange between cells of the cellular automaton. They are the essence of this work, intended to possess an **axiomatic** character.

I have used a C-like notation instead of traditional algorithm convention for clearness. No subroutines were used, nor were any optimizations attempted to keep the code as general as possible.. The parameter *ORDER* comes from Section 2.

```

1  /*
2  =====
3   Toy universe source code
4
5   Author: Alexandre Furtado Neto
6
7   Version : V1.2
8  =====
9   */
10
11  // CA symbols
12
13  #define ORDER      268
14  #define SIDE       (1<<ORDER)
15  #define MASK       (SIDE-1)
16  #define DIAG       (2*MASK)
17  #define LIGHT      (2*DIAG)
18  #define LIGHT2     (LIGHT*LIGHT)
19  #define SIDE2      (SIDE*SIDE)
20  #define SIDE3      (SIDE*SIDE2)
21  #define SHIFT      (ORDER/2)
22  #define S          (SIDE/2)
23
24  // Physical symbols (used with variable code)
25
26  #define PHOTON      0x0010
27  #define GLUON       0x0020
28  #define NEUTRINO    0x0040
29  #define Z           0x0080
30  #define W           0x0100
31  #define ELECTRON    6
32  #define QUARK       7
33  #define FERMION     (ELECTRON | QUARK | NEUTRINO)
34  #define BOSON       (PHOTON | GLUON | Z | W)
35  #define COLLAPSE    0x02
36
37  // Color symbols
38
39  #define C_MASK      0x07
40  #define Q_MASK      0x08
41  #define W_MASK      0x10
42  #define D_MASK      0x20
43  #define NEUTRAL     0x00

```

```

44
45 // Automaton cell structure (register)
46
47 typedef struct {
48     uint t;        // lifetime
49     uint3 dir;      // uint3: 3 bit unsigned int
50     bool active;    // sublattice
51     uint f;        // frequency
52     uint a;        // affinity
53     uint5 chrg;     // charge (d,c2,c1,c0,w,q)
54     uint o[3], p[3], s[3]; // vectors
55     uint vp0[3];    // mpmentum propensity
56     int phi;        // int: signed int with ORDER bits
57     uint noise;     // uint: unsigned int with ORDER bits
58     uint4 code;     // interaction result
59     uint sync;      // wavefront sync
60     uint u, v;      // harmonic phase
61
62     // Superluminal variables
63
64     uint flash;
65     uint pole[3];
66
67 } Cell;
68
69 // Macros
70
71 #define ISNULL(v) (v[0]==0 && v[1]==0 && v[2]==0)
72 #define ISEQUAL(v,u) (v[0]==u[0] && v[1]==u[1] && v[2]==u[2])
73 #define RESET(v) {v[0]=0;v[1]=0;v[2]=0;}
74 #define COPY(u,v) {u[0]=v[0];u[1]=v[1];u[2]=v[2];}
75 #define MOD2(v) (v[0]*v[0]+v[1]*v[1]+v[2]*v[2])
76 #define ALIGNED(u,v) (u[1]*v[2]-u[2]*v[1]+u[2]*v[0]-u[0]*v[2]+u
    [0]*v[1]-u[1]*v[0]==0)
77
78 Cell *stb, *draft;
79
80 /*
81  * Same code for all cells.
82  */
83 void main() {
84
85     //////////// STEP1: PAIRING ////////////
86
87     Copy all variables from draft to stb
88
89     //////////// STEP2: COMPARE ////////////
90
91     // Compare columns
92

```

```

93  if (draft->t % LIGHT == 0) {
94
95      Shift 'vertically' all draft cells
96
97      // Compare 'columns'
98
99      if (stb->b == draft->b && ISEQUAL(stb->o, draft->o)) {
100         // Same sector?
101         //
102         if (((draft->chrg ^ stb->chrg) & D_MASK) == 0) {
103             // Do they have same affinity?
104             //
105             if (draft->a == stb->a) {
106                 if (draft->code == COLLAPSE) {
107                     stb->code = 0;
108                     stb->f = 1;
109                     stb->a = stb->floor;
110                 }
111                 //
112                 // Are bubbles superposing
113                 //
114                 else if (ISEQUAL(draft->o, stb->o)) {
115                     // Virgin?
116                     //
117                     if (stb->code == 0) {
118                         // Pair formation
119                         //
120                         unsigned char cc =
121                             (stb->chrg & C_MASK) ^ (draft->chrg & C_MASK);
122                         unsigned char ww =
123                             (stb->chrg & W_MASK) ^ (draft->chrg & W_MASK);
124                         unsigned char qq =
125                             (stb->chrg & Q_MASK) ^ (draft->chrg & Q_MASK);
126                         //
127                         if (cc == 0 && ww == 0 && qq == Q_MASK) {
128                             stb->a = draft->a;
129                             stb->code = NEUTRINO;
130                             stb->f++;
131                         } else if (cc == 0 && ww == W_MASK && qq == Q_MASK)
132                             {
133                                 stb->a = draft->a;
134                                 stb->code = GLUON;
135                                 stb->f++;
136                             } else if (cc == C_MASK && ww == 0 && qq == 0) {
137                                 stb->a = draft->a;
138                                 stb->code = W;
139                                 stb->f++;
140                             } else if (cc == C_MASK && ww == 0 && qq == Q_MASK)
141                                 {
142                                     stb->a = draft->a;

```

```

141         stb->code = Z;
142         stb->f++;
143     } else if (cc == C_MASK && ww == W_MASK && qq ==
144               Q_MASK) {
145         stb->a = draft->a;
146         stb->code = PHOTON;
147         stb->f++;
148     }
149 }
150 }
151 }
152 }
153 }
154
155 ////////// STEP3: REPLICATE //////////
156
157 // Once each light step
158 //
159 if (draft->t % LIGHT == 0) {
160     //
161     // Copy only variables that changed in compare()
162     //
163     stb->b = draft->b;
164     stb->f = draft->f;
165     stb->code = draft->code;
166 }
167
168 ////////// STEP4: INTERACT //////////
169
170 if (draft->t % LIGHT == 0)
171 {
172     // Play pseudo dices against the sine phase
173     //
174     if (stb1->noise > abs(stb1->v) &&
175         stb2->noise > abs(stb1->v) &&
176         (!ISNULL(stb1->p) || !ISNULL(stb2->p))) {
177         // Preserve momentum for parallel transport
178         //
179         COPY(draft1->pole, stb2->p);
180         COPY(draft2->pole, stb1->p);
181         //
182         if (stb1->code == BOSON && stb2->code == BOSON) {
183             // Boson x boson
184             //
185             // Same sector?
186             //
187             if (((stb1->chrg ^ stb2->chrg) & D_MASK) == 0) {
188                 // Gluon x gluon?
189                 //

```

```

190     if ((stb1->chrg & C_MASK) != 0 && (stb1->chrg & C_MASK)
191         != C_MASK &&
192         (stb2->chrg & C_MASK) != 0 && (stb2->chrg & C_MASK)
193         != C_MASK) {
194         // Exchange colors, ignoring other chrgs
195         //
196         draft1->chrg &= ~C_MASK;
197         draft2->chrg &= ~C_MASK;
198         draft1->chrg |= stb2->chrg & C_MASK;
199         draft2->chrg |= stb1->chrg & C_MASK;
200         //
201         // Pole pole
202         //
203         COPY(draft1->pole, draft1->p);
204         COPY(draft2->pole, draft2->p);
205         //
206         // Start flash flooding
207         //
208         draft1->flash = SIDE;
209         draft2->flash = SIDE;
210     }
211     //
212     // Neutral 1?
213     //
214     else if ((stb1->chrg & C_MASK) == 0) {
215         if ((stb1->code & Q_MASK) == 1) {
216             // Photon 1 or Z 1
217             //
218             if ((stb1->code & W_MASK) == 1) {
219                 // Photon 1
220                 //
221                 if ((stb2->code & W_MASK) == 0 && (stb2->code &
222                     Q_MASK) == 0 &&
223                     (((stb2->chrg >> 1) ^ stb2->chrg) & 1) == 0)
224                 {
225                     // Photon 1 x W 2
226                     //
227                     // Pole pole
228                     //
229                     // Copy momentum information for flash
230                     //
231                     COPY(draft1->pole, draft1->p);
232                     COPY(draft2->pole, draft2->p);
233                     //
234                     // Start flash flooding
235                     //
236                     draft1->flash = SIDE;
237                     draft2->flash = SIDE;

```



```

237     }
238   } else {
239     // Z 1
240     //
241     if ((stb2->code & W_MASK) == 0 && (stb2->code &
242       Q_MASK) == 0 &&
243       (((stb2->chrg >> 1) ^ stb2->chrg) & 1) == 0) {
244       // Z 1 x W 2
245       //
246       // Pole pole
247       //
248       // Copy momentum information for flash
249       COPY(draft1->pole, draft1->p);
250       COPY(draft2->pole, draft2->p);
251       //
252       // Start flash flooding
253       //
254       draft1->flash = SIDE;
255       draft2->flash = SIDE;
256     }
257   }
258 }
259 }
260 else if ((stb2->chrg & C_MASK) == 0) {
261   if ((stb2->code & Q_MASK) == 1) {
262     // Photon 2 or Z 2
263     //
264     if ((stb2->code & W_MASK) == 1) {
265       // Photon 2
266       //
267       if ((stb1->code & W_MASK) == 0 && (stb1->code &
268         Q_MASK) == 0 &&
269         (((stb1->chrg >> 1) ^ stb1->chrg) & 1) == 0) {
270         // Photon 2 x W 1
271         //
272         // Pole pole
273         //
274         // Copy momentum information for flash
275         COPY(draft1->pole, draft1->p);
276         COPY(draft2->pole, draft2->p);
277         //
278         // Start flash flooding
279         //
280         draft1->flash = SIDE;
281         draft2->flash = SIDE;
282       }
283     } else {
284       // Z 2

```

```

285          //
286          if ((stb1->code & W_MASK) == 0 && (stb1->code &
          Q_MASK) == 0 &&
287          (((stb1->chrg >> 1) ^ stb1->chrg) & 1) == 0) {
288          // Z 2 x W 1
289          //
290          // Pole pole
291          //
292          // Copy momentum information for flash
293          //
294          COPY(draft1->pole, draft1->p);
295          COPY(draft2->pole, draft2->p);
296          //
297          // Start flash flooding
298          //
299          draft1->flash = SIDE;
300          draft2->flash = SIDE;
301      }
302  }
303  }
304  }
305  }
306  }
307  else if (stb1->code == FERMION && stb2->code == BOSON) {
308      // Fermion x boson
309      //
310      // Isolate chrg bits
311      //
312      unsigned c1 = (stb1->chrg & C_MASK) & C_MASK;
313      unsigned q1 = ((stb1->chrg & Q_MASK) >> 3) & 1;
314      unsigned w1 = ((stb1->chrg & W_MASK) >> 4) & 1;
315      unsigned d1 = (stb1->chrg & D_MASK) >> 5;
316      unsigned c2 = (stb2->chrg & C_MASK) & C_MASK;
317      unsigned q2 = ((stb2->chrg & Q_MASK) >> 3) & 1;
318      unsigned w2 = ((stb2->chrg & W_MASK) >> 4) & 1;
319      unsigned d2 = (stb2->chrg & D_MASK) >> 5;
320      //
321      if (d1 != d2) {
322          // SUPRESSED
323          // (Only same sector are allowed to interact in this
          way)
324          //
325          return;
326      }
327      //
328      // Quark x gluon?
329      //
330      if (c1 != NEUTRAL && c1 != ~NEUTRAL && c2 != NEUTRAL &&
          c2 != ~NEUTRAL) {
331          if (c1 == ~c2) // TODO why?? {

```

```

332         // Blindly exchange colors, ignoring all other chrgs
333         //
334         draft1->chrg &= ~C_MASK;
335         draft2->chrg &= ~C_MASK;
336         draft1->chrg |= c2;
337         draft2->chrg |= c1;
338         //
339         // Pole pole
340         //
341         // Copy momentum information for flash
342         //
343         COPY(draft1->pole, draft1->p);
344         COPY(draft2->pole, draft2->p);
345         //
346         // Start flash flooding
347         //
348         draft1->flash = SIDE;
349         draft2->flash = SIDE;
350     }
351 }
352 //
353 // Quark x [photon, Z, W]?
354 //
355 else if (c1 != NEUTRAL && c1 != ~NEUTRAL) {
356     // Is it a propeller?
357     //
358     if (stb1->a == stb2->a) {
359         // Inertia
360         //
361         // Copy momentum information for flash
362         //
363         if (ALIGNED(draft1->o, draft1->p)) {
364             COPY(draft1->pole, draft2->p);
365             COPY(draft2->pole, draft2->p);
366         } else {
367             COPY(draft1->pole, draft1->p);
368             COPY(draft2->pole, draft1->p);
369         }
370         //
371         // Start flash flooding
372         //
373         draft1->flash = SIDE;
374         draft2->flash = SIDE;
375     } else if (q1 == q2) {
376         // Pole pole
377         //
378         // Copy momentum information for flash
379         //
380         COPY(draft1->pole, draft1->p);
381         COPY(draft2->pole, draft2->p);

```

```

382          //
383          // Start flash flooding
384          //
385          draft1->flash = SIDE;
386          draft2->flash = SIDE;
387      } else {
388          // Pole pole
389          //
390          // Copy momentum information for flash
391          //
392          COPY(draft1->pole, draft1->p);
393          COPY(draft2->pole, draft2->p);
394          //
395          // Start flash flooding
396          //
397          draft1->flash = SIDE;
398          draft2->flash = SIDE;
399      }
400  }
401  //
402  // Electron x [photon, Z, W]?
403  //
404  else {
405      // Is it a propeller?
406      //
407      if (stb1->a == stb2->a) {
408          // Inertia
409          //
410          // Copy momentum information for flash
411          //
412          if (ALIGNED(draft1->o, draft1->p)) {
413              COPY(draft1->pole, draft2->p);
414              COPY(draft2->pole, draft2->p);
415          } else {
416              COPY(draft1->pole, draft1->p);
417              COPY(draft2->pole, draft1->p);
418          }
419          //
420          // Start flash flooding
421          //
422          draft1->flash = SIDE;
423          draft2->flash = SIDE;
424      } else if (q1 == q2) {
425          if (w1 == w2) {
426              if (c1 == c2 == 0 && w1 == 1) {
427                  // Pole pole
428                  // Copy momentum information for flash
429                  //
430                  COPY(draft1->pole, draft1->p);
431                  COPY(draft2->pole, draft2->p);

```

```

432      //
433      // Start flash flooding
434      //
435      draft1->flash = SIDE;
436      draft2->flash = SIDE;
437  } else if (c1 == c2 == C_MASK && w1 == 0) {
438      // Pole pole
439      //
440      // Copy momentum information for flash
441      //
442      COPY(draft1->pole, draft1->p);
443      COPY(draft2->pole, draft2->p);
444      //
445      // Start flash flooding
446      //
447      draft1->flash = SIDE;
448      draft2->flash = SIDE;
449  }
450  }
451  } else {
452      if (w1 == w2) {
453          if (c1 == c2 == 0 && w1 == 1) {
454              // Pole pole
455              // Copy momentum information for flash
456              //
457              COPY(draft1->pole, draft1->p);
458              COPY(draft2->pole, draft2->p);
459              //
460              // Start flash flooding
461              //
462              draft1->flash = SIDE;
463              draft2->flash = SIDE;
464          } else if (c1 == c2 == C_MASK && w1 == 0) {
465              // Pole pole
466              //
467              // Copy momentum information for flash
468              //
469              COPY(draft1->pole, draft1->p);
470              COPY(draft2->pole, draft2->p);
471              //
472              // Start flash flooding
473              //
474              draft1->flash = SIDE;
475              draft2->flash = SIDE;
476          }
477      }
478  }
479  }
480  }
481  //

```

```

482 // Boson x fermion
483 //
484 else if (stb1->code == BOSON && stb2->code == FERMION) {
485 // Isolate chrg bits
486 //
487 unsigned c1 = (stb1->chrg & C_MASK) & C_MASK;
488 unsigned q1 = ((stb1->chrg & Q_MASK) >> 3) & 1;
489 unsigned w1 = ((stb1->chrg & W_MASK) >> 4) & 1;
490 unsigned d1 = (stb1->chrg & D_MASK) >> 5;
491 unsigned c2 = (stb2->chrg & C_MASK) & C_MASK;
492 unsigned q2 = ((stb2->chrg & Q_MASK) >> 3) & 1;
493 unsigned w2 = ((stb2->chrg & W_MASK) >> 4) & 1;
494 unsigned d2 = (stb2->chrg & D_MASK) >> 5;
495 //
496 if (d1 != d2) {
497 // SUPRESSED
498 // (Only same sector are allowed to interact in this
499 // way)
500 //
501 return;
502 }
503 // Quark x gluon?
504 //
505 if (c2 != NEUTRAL && c2 != ~NEUTRAL && c1 != NEUTRAL &&
506     c1 != ~NEUTRAL) {
507     if (c2 == ~c1) // TODO why?? {
508         // Blindly exchange colors, ignoring all other chrgs
509         //
510         draft2->chrg &= ~C_MASK;
511         draft1->chrg &= ~C_MASK;
512         draft2->chrg |= c2;
513         draft1->chrg |= c1;
514         //
515         // Pole pole
516         //
517         // Copy momentum information for flash
518         //
519         COPY(draft1->pole, draft1->p);
520         COPY(draft2->pole, draft2->p);
521         //
522         // Start flash flooding
523         //
524         draft1->flash = SIDE;
525         draft2->flash = SIDE;
526     }
527 }
528 // Quark x [photon, Z, W]?
529 //

```

```

530     else if (c1 != NEUTRAL && c1 != ~NEUTRAL) {
531         // Is it a propeller?
532         //
533         if (stb1->a == stb2->a) {
534             // Inertia
535             //
536             // Copy momentum information for flash
537             //
538             if (ALIGNED(draft1->o, draft1->p)) {
539                 COPY(draft1->pole, draft2->p);
540                 COPY(draft2->pole, draft2->p);
541             } else {
542                 COPY(draft1->pole, draft1->p);
543                 COPY(draft2->pole, draft1->p);
544             }
545             //
546             // Start flash flooding
547             //
548             draft1->flash = SIDE;
549             draft2->flash = SIDE;
550         } else if (q1 == q2) {
551             // Pole pole
552             //
553             // Copy momentum information for flash
554             //
555             COPY(draft1->pole, draft1->p);
556             COPY(draft2->pole, draft2->p);
557             //
558             // Start flash flooding
559             //
560             draft1->flash = SIDE;
561             draft2->flash = SIDE;
562         } else {
563             // Pole pole
564             //
565             // Copy momentum information for flash
566             //
567             COPY(draft1->pole, draft1->p);
568             COPY(draft2->pole, draft2->p);
569             //
570             // Start flash flooding
571             //
572             draft1->flash = SIDE;
573             draft2->flash = SIDE;
574         }
575     }
576     //
577     // Electron x [photon, Z, W]?
578     //
579     else {

```



```

580 // Is it a propeller?
581 //
582 if (stb1->a == stb2->a) {
583 // Inertia
584 //
585 // Copy momentum information for flash
586 //
587 if (ALIGNED(draft1->o, draft1->p)) {
588 COPY(draft1->pole, draft2->p);
589 COPY(draft2->pole, draft2->p);
590 } else {
591 COPY(draft1->pole, draft1->p);
592 COPY(draft2->pole, draft1->p);
593 }
594 //
595 // Start flash flooding
596 //
597 draft1->flash = SIDE;
598 draft2->flash = SIDE;
599 } else if (q1 == q2) {
600 if (w1 == w2) {
601 if (c1 == c2 == 0 && w1 == 1) {
602 // Pole
603 // Copy momentum information for flash
604 //
605 COPY(draft1->pole, draft1->p);
606 COPY(draft2->pole, draft2->p);
607 //
608 // Start flash flooding
609 //
610 draft1->flash = SIDE;
611 draft2->flash = SIDE;
612 } else if (c1 == c2 == C_MASK && w1 == 0) {
613 // Pole pole
614 //
615 // Copy momentum information for flash
616 //
617 COPY(draft1->pole, draft1->p);
618 COPY(draft2->pole, draft2->p);
619 //
620 // Start flash flooding
621 //
622 draft1->flash = SIDE;
623 draft2->flash = SIDE;
624 }
625 }
626 } else {
627 if (w1 == w2) {
628 if (c1 == c2 == 0 && w2 == 1) {
629 // Pole pole

```

```

630          //
631          // Copy momentum information for flash
632          //
633          COPY(draft1->pole , draft1->p);
634          COPY(draft2->pole , draft2->p);
635          //
636          // Start flash flooding
637          //
638          draft1->flash = SIDE;
639          draft2->flash = SIDE;
640      } else if (c1 == c2 == C_MASK && w2 == 0) {
641          // Pole pole
642          //
643          // Copy momentum information for flash
644          //
645          COPY(draft1->pole , draft1->p);
646          COPY(draft2->pole , draft2->p);
647          //
648          // Start flash flooding
649          //
650          draft1->flash = SIDE;
651          draft2->flash = SIDE;
652      }
653  }
654 }
655 }
656
657 }
658 //
659 // fermion x fermion
660 //
661 else {
662     // Isolate chrg bits
663     //
664     unsigned c1 = (stb1->chrg & C_MASK) & 7;
665     unsigned q1 = ((stb1->chrg & Q_MASK) >> 3) & 1;
666     unsigned w1 = ((stb1->chrg & W_MASK) >> 4) & 1;
667     unsigned d1 = (stb1->chrg & D_MASK) >> 5;
668     unsigned c2 = (stb2->chrg & C_MASK) & 7;
669     unsigned q2 = ((stb2->chrg & Q_MASK) >> 3) & 1;
670     unsigned w2 = ((stb2->chrg & W_MASK) >> 4) & 1;
671     unsigned d2 = (stb2->chrg & D_MASK) >> 5;
672     //
673     //
674     // Matter/antimatter flags
675     //
676     bool matter1 = c1 == 0 || c1 == 1 || c1 == 2 || c1 == 4;
677     bool matter2 = c2 == 0 || c2 == 1 || c2 == 2 || c2 == 4;
678     //
679     // Same sector?

```

```

680      //
681      if (d1 == d2) {
682          // quark x quark?
683          //
684          if (c1 != NEUTRAL && c1 != ~NEUTRAL &&
685              c2 != NEUTRAL && c2 != ~NEUTRAL) {
686              // Same color and electric chrg?
687              //
688              if (c1 == c2 && q1 == q2) // TODO: include affinity??
689                  weak chrg?? {
690                  // Quark cohesion
691                  //
692                  // Copy momentum information for flash
693                  COPY(draft1->pole, draft1->p);
694                  COPY(draft2->pole, draft2->p);
695                  //
696                  // Start flash flooding
697                  //
698                  draft1->flash = SIDE;
699                  draft2->flash = SIDE;
700              }
701              //
702              // Complementary charges?
703              //
704              else if (c1 == ~c2 && w1 == ~w2 && q1 == ~q2) {
705                  // Quark annihilation?
706                  //
707                  // cp x cp
708                  //
709                  // Copy momentum information for flash
710                  //
711                  if (ALIGNED(draft1->o, draft1->p)) {
712                      COPY(draft1->pole, draft1->p);
713                      COPY(draft2->pole, draft1->p);
714                  } else {
715                      COPY(draft1->pole, draft2->p);
716                      COPY(draft2->pole, draft2->p);
717                  }
718                  //
719                  // Start flash flooding
720                  //
721                  draft1->flash = SIDE;
722                  draft2->flash = SIDE;
723                  //
724                  if (collapse) {
725                      // Disintegrate the packet
726                      //
727                      draft1->code = COLLAPSE;
728                      draft2->code = COLLAPSE;

```

```

729         }
730     }
731 }
732 //
733 // quark x electron?
734 //
735 else if (c1 != NEUTRAL && c1 != ~NEUTRAL) {
736     if (q1 == q2 && matter1 == matter2) {
737         // Implement repulsion
738     } else if (q1 != q2 && matter1 == matter2) {
739         // Implement attraction
740     }
741 }
742 //
743 // Electron x electron
744 //
745 else if ((c1 == NEUTRAL || c1 == ~NEUTRAL) && c1 == c2)
746 {
747     if (q1 == q2 && matter1 == matter2) {
748         // Implement cohesion
749         //
750         // Copy momentum information for flash
751         //
752         COPY(draft1->pole, draft1->p);
753         COPY(draft2->pole, draft2->p);
754         //
755         // Start flash flooding
756         //
757         draft1->flash = SIDE;
758         draft2->flash = SIDE;
759     } else if (q1 != q2 && w1 != w2 && matter1 != matter2
760 ) {
761         // Electron annihilation?
762         //
763         // cp cp
764         // Copy momentum information for flash
765         //
766         if (ALIGNED(draft1->o, draft1->p)) {
767             COPY(draft1->pole, draft1->p);
768             COPY(draft2->pole, draft1->p);
769         } else {
770             COPY(draft1->pole, draft2->p);
771             COPY(draft2->pole, draft2->p);
772         }
773         //
774         // Start flash flooding
775         //
776         draft1->flash = SIDE;
777         draft2->flash = SIDE;
778         //

```

```

777         if (collapse) {
778             // Disintegrate the packet
779             //
780             draft1->code = COLLAPSE;
781             draft2->code = COLLAPSE;
782         }
783     }
784 }
785 }
786 //
787 // Different sectors
788 //
789 else {
790     bool s1 = (c1 == c2 == 0 && q1 == q2 == 1 && d1 == d2
791               == 0);
792     bool s2 = (c1 == c2 == 7 && q1 == q2 == 0 && d1 == d2
793               == 1);
794     bool s3 = (c1 == c2 != 0 != 7 && q1 == q2);
795     //
796     bool c1 = (q1 == q2 && w1 == w2 && s1 == s2);
797     bool c2 = (q1 != q2 && w1 != w2 && s1 != s2);
798     bool c3 = (d1 == 0 && w1 == 0 && w1 != w2);
799     bool c4 = (d1 == 1 && w1 == 1 && w1 != w2);
800     //
801     if ((d1 == 0 && s1 == s2) || (d1 == 1 && s1 == s3 && s1
802         != s2)) {
803         // Swap colors
804         //
805         int c1 = stb1->chrg & C_MASK;
806         int c2 = stb2->chrg & C_MASK;
807         draft1->chrg &= ~C_MASK;
808         draft2->chrg &= ~C_MASK;
809         draft1->chrg |= c2;
810         draft2->chrg |= c1;
811         //
812         // cp cp
813         //
814         // Copy momentum information for flash
815         //
816         if (ALIGNED(draft1->o, draft1->p)) {
817             COPY(draft1->pole, draft1->p);
818             COPY(draft2->pole, draft1->p);
819         } else {
820             COPY(draft1->pole, draft2->p);
821             COPY(draft2->pole, draft2->p);
822         }
823         //
824         // Start flash flooding
825         //
826         draft1->flash = SIDE;

```

```

824         draft2->flash = SIDE;
825         //
826         if (collapse) {
827             // Disintegrate the packet
828             //
829             draft1->code = COLLAPSE;
830             draft2->code = COLLAPSE;
831         }
832     }
833     //
834     // Chiral?
835     //
836     else if (c1 || c2 || c3 || c4) {
837         // Change hands
838         //
839         int w1 = stb1->chrg & W_MASK;
840         int w2 = stb2->chrg & W_MASK;
841         draft1->chrg &= ~W_MASK;
842         draft2->chrg &= ~W_MASK;
843         draft1->chrg |= w2;
844         draft2->chrg |= w1;
845         //
846         // Copy momentum information for flash
847         //
848         COPY(draft1->pole, draft1->p);
849         COPY(draft2->pole, draft2->p);
850         //
851         // Start flash flooding
852         //
853         draft1->flash = SIDE;
854         draft2->flash = SIDE;
855     }
856 }
857 }
858 }
859 }
860
861 /////////////// STEP5: EXPAND ///////////////////
862
863     draft->t++;
864     //
865     // Decay momentum propensity
866     //
867     if (stb->t > 0 && !ISNULL(stb->p)) {
868         // Non-trivial affinity?
869         //
870         if (stb->a) {
871             // Decay as space
872             //
873             draft->p[0] *= ((draft->t * draft->t) -

```

```

874         draft->vp0[0] * draft->vp0[0]) / (draft->t *
            draft->t);
875     draft->p[1] *= ((draft->t * draft->t) -
876         draft->vp0[1] * draft->vp0[1]) / (draft->t *
            draft->t);
877     draft->p[2] *= ((draft->t * draft->t) -
878         draft->vp0[2] * draft->vp0[2]) / (draft->t *
            draft->t);
879 } else {
880     // Decay as particle
881     //
882     draft->p[0] *= (2 * draft->t - 1) / (2 * draft->t);
883     draft->p[1] *= (2 * draft->t - 1) / (2 * draft->t);
884     draft->p[2] *= (2 * draft->t - 1) / (2 * draft->t);
885 }
886 }
887 //
888 // Spread cell contents if not empty
889 //
890 if (stb->f > 0 || draft->flash > 0) {
891     // Last tick?
892     //
893     if (stb->t % LIGHT == 0) {
894         // Sine recursive algorithm
895         //
896         draft->v = (K2D * stb->v + K2N * ((K1D * stb->u -
897             K1N * stb->v) / K1D)) / K2D;
898         draft->u = (K1D * ((K1D * stb->u - K1N * stb->v) /
899             K1D) -
900             K1N * draft->v) / K1D;
901     }
902     //
903     // Re-emitted?
904     //
905     if (stb->flash && ALIGNED(stb->o, stb->pole)) {
906         // Bubble is in fact reissued
907         //
908         draft->t = 0;
909         draft->sync = LIGHT2;
910         draft->u = SIDE2 / 2;
911         draft->v = 0;
912         RESET(draft->o);
913         draft->flash = 0;
914         return;
915     }
916     //
917     // Explore von Neumann directions
918     //
919     Cell* neighbor;
920     bool mature = stb->f > 0 && stb->t * stb->t > stb->sync;

```



```

920     for (int dir = 0; dir < 6; dir++) {
921         char vdir[3] = { 0, 0, 0 };
922         neighbor = getPointer(dir, draft, (char*)vdir);
923         //
924         // Is flash active?
925         //
926         if (stb->flash > 0 && neighbor->flash == 0) {
927             neighbor->flash = stb->flash;
928             COPY(neighbor->pole, stb->pole);
929         }
930         //
931         // Test if branch is legal
932         //
933         if(mature && isAllowed(dir, vdir, stb->o, stb->dir))
934             {
935                 // Copy necessary values
936                 //
937                 neighbor->t = draft->t;
938                 neighbor->dir = dir;
939                 neighbor->f = stb->f;
940                 neighbor->a = stb->a;
941                 neighbor->chrg = stb->chrg;
942                 neighbor->u = draft->u;
943                 neighbor->v = draft->v;
944                 //
945                 neighbor->o[0] = (stb->o[0] + vdir[0]);
946                 neighbor->o[1] = (stb->o[1] + vdir[1]);
947                 neighbor->o[2] = (stb->o[2] + vdir[2]);
948                 //
949                 COPY(neighbor->s, stb->s);
950                 COPY(neighbor->p, stb->p);
951                 //
952                 // Schedule for spherical evolution
953                 //
954                 neighbor->sync = LIGHT2 * MOD2(neighbor->o);
955             }
956         //
957         // Decrement flash
958         //
959         if (draft->flash > 0)
960             draft->flash--;
961         //
962         // Bubble propagated?
963         //
964         if(mature) {
965             // Clean data
966             //
967             draft->code = 0;
968             //

```

```

969          // Bubble meets itself?
970          //
971          if (MOD2(stb->o) == 3 * SIDE * SIDE / 4) {
972              // Reissue by wrapping
973              //
974              draft->sync = 0;
975              draft->t = 0;
976          } else {
977              // Erase origin cell
978              //
979              draft->f = 0;
980          }
981      }
982  }
983 }

```