 doi:10.20944/preprints202107.0513.v2

Recovery of global symmetries in a 't Hooftian universe

Alexandre Furtado Neto*

UNESP Alumnus

Abstract

A new charge is postulated in a finite, closed, Euclidean discrete space to restore all fundamental symmetries on a global level. 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. Randomness is identified using a Chaintin argument. Aleph0 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.

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

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

*alexandre.com@yahoo.com

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

2 Space and time

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 an equal amount of information (a *register*) and are synchronized by a common time $t \in \mathbb{N}$, 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 $(1/8)$[1] of the Planck scale as

$$\begin{aligned} SIDE &= 2^{order}, \\ &\approx 6.582 \times 10^{63}, \end{aligned}$$

where $order = 212$. 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]).

3 Bubbles

Each register contains six bits of charge d, c_2, c_1, c_0, w, q, a few integer variables, e.g., $b, f, \phi, \varphi, noise$, and four integer vectors $\boldsymbol{p}, \boldsymbol{s}, \boldsymbol{o}, \boldsymbol{pole}$. Most properties spread to the expanding cells, with the exception of the vector $\boldsymbol{p}$, which is passed to a single cell in its path.

A *bubble* is an expanding spherical wavefront of information organized on the registers. Whenever vector $\boldsymbol{p}$ moves, it casts such a perturbation. The von Neumann neighborhood $Dirs$ is used 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), where D is the cube diagonal, so bubbles expand with the same speed c, increasing $|\boldsymbol{o}|$ 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 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 [16]).

A bubble is eventually reissued from a point calculated on its surface by resetting the origin vector ($\boldsymbol{o} = \boldsymbol{0}$) when interacting with other bubbles. One extra variable named *code* is used to qualify bubble superpositions.

Each bubble lives in a separate 3d-space L_i, $i = \{1...2L\}$ to avoid propagation issues. F superposing bubbles with a common bond $b_j = B$, $j = \{1...F\}$, have frequency $f_j = F$. Two such bubbles with some or all opposite charges is called a *pair*.

Bubbles also have a sinusoidal phase ϕ coupled to f, achieved by a recurrent algorithm. Together with the related variable φ, they implement interference and self-interference (see Sciarretta [11]). Each occupied register has $\varphi = \phi$, which decays as

$$\varphi^{k+1} = \varphi^k \left(1 - \frac{1}{2k}\right)$$

[1]The exact granularity will be obtained when it is verified that the model generates the same number of particles present in the universe.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

at each *light step* k. During wavefront accommodation the value of *noise* is updated using the selected direction index ($Dirs[index]$) and latter compared with $|\phi + \varphi|$, which indicates whether a bubble-bubble interaction is legal — this produces the observed harmonic behavior at the ensemble level. Momentun $\boldsymbol{p}$ is moved to the cell that minimizes the ***pole*** vector.

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 $color = 2^2c_2 + 2^1c_1 + 2^0c0$ will be used latter. On the other hand, the *weak* charge w, or *chirality*, is associated with congruence.

Vector $\boldsymbol{p}$ is related to motion direction, while vector $\boldsymbol{s}$ 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 $\boldsymbol{p}$ from its own center is defined as its *pole*.

Finally, a loose association between the terms *bubble* and *(unit of) energy* is assumed.

4 Interactions

4.1 Basic rules

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

When a bubble is reissued after interacting with another bubble and, in addition, their poles coincide, all bonded bubbles 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.

4.2 Interactions

The detailed algorithmic description of all interaction rules is contained in the Appendix. Half of the cells are active, while the other half are passive, alternating roles in time. The basic loop consists of the following steps

- **Commute**
  The active/passive roles are exchanged and data from the active cells are copied to the passive cells.

- **Update**
  Passive cells are shifted in the v dimension and compared until a complete loop of $SIDE^2$ steps. Properties such as the frequency are updated.

- **Interact**
  After having data being copied back to the active cells, again, the passive cells are shifted in the v dimension and compared for possible interactions, which are encoded in the *code* variable.

- **Expand**
  All bubbles expand one light step in their respective xyz sub-lattice. When momentum $\boldsymbol{p}$ moves, it releases a new perturbation wavefront from its old position.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

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 $SIDE \, \boldsymbol{mod} \, 3 = 1$, 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 $\boldsymbol{p} = \boldsymbol{k} \times \boldsymbol{s}$, where $\boldsymbol{k}$ 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.

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 bonded bubbles. If the packet contains colored or neutral bubbles with $f = 1$, 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 *mass* of the packet. The non-equilibrium caused by nearby charges or masses can change the motion of the packet.

Packets possessing a quantized set of bubbles with $f = 1$ 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, even possessing $f = 2$, also present a "fermionic" character. In other words, if the packet includes a (quantized) population of bubbles with $f = 1$ 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 envolved and $f = 1$, 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'.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

Ephemeral resonance states, including radial vibration modes[2] (see Itô [17]) 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 [18]. In that work, there is an explanation of how magnetic effects emerge from symmetry breaking of this spherical pattern.

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 bonded to a packet is dubbed a *propeller*.

The long range behavior of photons is due to the absence of other bubbles with the same b 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-emmited 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 b 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 *bremsstrahlung*. 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 ubiquituous 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 Schrodinger equation, can be used to analyze them — the point here is that the system certainly presents a quantum behavior.

[2]Empirically in three generations of leptons and quarks.

 doi:10.20944/preprints202107.0513.v2

In this context, Born's rule arises naturally without the need to be postulated, bringing with it quantum probabilities. Perhaps new analysis tools should be devised to explore the model in search of physics beyond the Standard Model.

Last but not least, a gravity-like dynamics emerges as a residual effect of the electromagnetic force (see also Assis [19]) — 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.

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 [20]) — the maximum number with physical meaning in the universe is just $SIDE$.

We are, therefore, facing an ontological[3] 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 support[4], and evaluated if they in fact allow building a predictive, *bona fide,* theory.

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 vacuum*, 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).

[11] Sciarretta, A. *A local-realistic model of quantum mechanics based on a discrete spacetime*, Found. Phys. **48** (1), 60–91, (2018).

[3] Ontology is actually an always receding rule marking the frontier of the unfathomable.

[4] An implementation under development is accessible in Reference [21]

[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] G. J. Chaitin, *Algorithmic Information Theory*, in IBM Journal of Research and Development, **21**, no. 4, pp. 350-359 (1977).

[17] Itô D. *Cohesive force of electron and Nambu's mass formula*, Prog. Theor. Phys. **47**, no. 3 (1972).

[18] Hofer, W.A. *Elements of physics for the 21st century*, J. Phys.: Conf. Ser. **504**, 012014 (2014).

[19] Assis, A.K.T. *Deriving gravitation from electromagnetism*, Can. J. Phys. **70**, 330-340 (1992).

[20] Asperó, D., R. Schindler *Martin's Maximum implies Woodin's axiom (*)*. Annals of Mathematics, **193** (3), 793-835 (2021).

[21] Furtado Neto, A. *It from bit - a concrete attempt*, GitHub repository, https://github.com/automaton3d/automaton, (2021).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

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. The parameter *ORDER* comes from Section 2.

```
/*
 ================================================================
 Name    : CUIE algorithm
           (commute, update, interact, expand)
 Version  : V1.1
 ================================================================
 */

// CA symbols

#define ORDER    212
#define SIDE     (1<<ORDER)
#define MASK     (SIDE-1)
#define DIAG     (2*MASK)
#define LIGHT    (2*DIAG)
#define LIGHT2     (LIGHT*LIGHT)
#define SIDE2    (SIDE*SIDE)
#define SIDE3    (SIDE*SIDE2)
#define SHIFT    (ORDER/2)
#define PHOTON     1
#define GLUON    2
#define NEUTRINO   3
#define Z        4
#define W        5

// Charge masks

#define C_MASK   0x07
#define Q_MASK   0x08
#define W_MASK   0x10
#define D_MASK   0x20

// Automaton cell structure

typedef struct {
  uint3 dir;     // uint3: 3 bit unsigned int
  uint3 type;
  boolean active;
  uint f;
  uint t
  uint b;
  uint5 charge;
  uint o[3], p[3], s[3], pole[3];
  int phi;     // int: signed int with ORDER bits
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
45    uint noise;    // uint: unsigned int with ORDER bits
46    uint4 code;
47    uint synch;
48    int sine, cosine;
49    uint ctrl;
50
51  } Cell;
52
53  // Macros
54
55  #define ISNULL(v)     (v[0]==0 && v[1]==0 && v[2]==0)
56  #define ISEQUAL(v,u)   (v[0]==u[0] && v[1]==u[1] && v[2]==u[2])
57  #define RESET(v)      {v[0]=0;v[1]=0;v[2]=0;}
58  #define COPY(u,v)     {u[0]=v[0];u[1]=v[1];u[2]=v[2];}
59  #define MOD2(v)       (v[0]*v[0]+v[1]*v[1]+v[2]*v[2])
60  #define nextV(c)      {c->type&0x02?c-(SIDE3*(SIDE2-1)):c+SIDE3}
61  #define CELL          sizeof(Cell)
62  #define S             (SIDE/2)
63
64  Cell *stable, *draft;
65
66  /*
67   * Same code for all cells.
68   */
69  void main() {
70
71    ////////// STEP1: COMMUTE //////////
72
73    // Copy all variables
74    //
75    stable->t = draft->t;
76    stable->dir = draft->dir;
77    stable->charge = draft->charge;
78    stable->code = draft->code;
79    stable->noise = draft->noise;
80    stable->b = draft->b;
81    stable->synch = draft->synch;
82    stable->f = draft->f;
83    COPY(stable->p, draft->p);
84    COPY(stable->s, draft->s);
85    COPY(stable->o, draft->o);
86    COPY(stable->pole, draft->pole);
87    //
88    // Commute roles
89    //
90    stable->active = false;
91    draft->active = true;
92    Cell *temp = draft;
93    draft = stable;
94    stable = temp;
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
95
96      ////////// STEP2: COMPARE //////////
97
98      // Compare columns
99
100     if (draft->t % LIGHT == 0) {
101       //
102       // Shift 'vertically'
103       //
104       Cell temp = *cell;
105       for (int j = 0; j < SIDE2; j++) {
106         Cell* next = nextV(cell);
107         if (j == SIDE2 - 1)
108           next = &temp;
109         cell->f = next->f;
110         cell->b = next->b;
111         cell->charge = next->charge;
112         COPY(cell->o, next->o);
113         COPY(cell->p, next->p);
114         COPY(cell->s, next->s);
115         cell->phi = next->phi;
116         cell->code = next->code;
117         /////
118         //
119         // Compare 'columns'
120         //
121         if (stable->b == draft->b && ISEQUAL(stable->o, draft->o))
                {
122           if (draft->code == 0) {
123             if (((draft->charge & C_MASK) ^ (stable->charge &
                    C_MASK)) == C_MASK &&
124                 ((draft->charge & W_MASK) ^ (stable->charge &
                      W_MASK)) == W_MASK &&
125                 ((draft->charge & Q_MASK) ^ (stable->charge &
                      Q_MASK)) == Q_MASK)
126               draft->code = PHOTON;
127             else if (((draft->charge & C_MASK) ^ (stable->charge &
                    C_MASK)) == 0 &&
128                     ((draft->charge & W_MASK) ^ (stable->charge &
                        W_MASK)) == W_MASK &&
129                     ((draft->charge & Q_MASK) ^ (stable->charge &
                        Q_MASK)) == Q_MASK)
130               draft->code = PHOTON;
131             else if (((draft->charge & C_MASK) ^ (stable->charge &
                    C_MASK)) == 0 &&
132                     ((draft->charge & W_MASK) ^ (stable->charge &
                        W_MASK)) == 0 &&
133                     ((draft->charge & Q_MASK) ^ (stable->charge &
                        Q_MASK)) == Q_MASK)
134               draft->code = PHOTON;
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
135             else if (((draft->charge & C_MASK) ^ (stable->charge &
                    C_MASK)) == 0 &&
136                      ((draft->charge & W_MASK) ^ (stable->charge &
                           W_MASK)) == 0 &&
137                      ((draft->charge & Q_MASK) ^ (stable->charge &
                           Q_MASK)) == Q_MASK)
138               draft->code = PHOTON;
139             else if (((draft->charge & C_MASK) ^ (stable->charge &
                    C_MASK)) == C_MASK &&
140                      ((draft->charge & W_MASK) ^ (stable->charge &
                           W_MASK)) == 0 &&
141                      ((draft->charge & Q_MASK) ^ (stable->charge &
                           Q_MASK)) == Q_MASK)
142               draft->code = PHOTON;
143             else if (((draft->charge & C_MASK) ^ (stable->charge &
                    C_MASK)) == C_MASK &&
144                      ((draft->charge & W_MASK) ^ (stable->charge &
                           W_MASK)) == 0 &&
145                      ((draft->charge & Q_MASK) ^ (stable->charge &
                           Q_MASK)) == 0)
146               draft->code = PHOTON;
147             //
148             if (draft->code != 0)
149               draft->f++;
150           }
151           else if (draft->code == stable->code) {
152             draft->f++;
153           }
154         }
155         cell = nextV(cell);
156       }
157     }
158
159     ////////// STEP3: REPLICATE //////////
160
161     // Once each light step
162     //
163     if (draft->t % LIGHT == 0) {
164       //
165       // Copy only variables that changed in compare()
166       //
167       stable->f = draft->f;
168       stable->code = draft->code;
169     }
170
171     ////////// STEP4: INTERACT //////////
172
173     // Once each light step
174     //
175     if (draft->t % LIGHT == 0) {
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
176     int sig1 = ((stable->charge ^ draft->charge) & C_MASK) == 0
            &&
177       (draft->charge & C_MASK) == 0 &&
178       ((stable->charge ^ draft->charge) & Q_MASK) == 0 &&
179        (draft->charge & Q_MASK) == Q_MASK &&
180       ((stable->charge ^ draft->charge) & D_MASK) == 0 &&
181        (draft->charge & D_MASK) == 0;
182     int sig2 = ((stable->charge ^ draft->charge) & C_MASK) == 0
            &&
183       (draft->charge & C_MASK) == C_MASK &&
184       ((stable->charge ^ draft->charge) & Q_MASK) == 0 &&
185       (draft->charge & Q_MASK) == 0 &&
186       ((stable->charge ^ draft->charge) & D_MASK) == 0 &&
187       (draft->charge & D_MASK) == D_MASK;
188     int sig3 = ((stable->charge ^ draft->charge) & C_MASK) == 0
            &&
189       (draft->charge & C_MASK) != 0 &&
190       (draft->charge & C_MASK) != C_MASK &&
191       ((stable->charge ^ draft->charge) & Q_MASK) == 0;
192     //
193     int c1 = ((stable->charge ^ draft->charge) & Q_MASK) == 0 &&
194        ((stable->charge ^ draft->charge) & W_MASK) == 0 && sig1
               == sig2;
195     int c2 = ((stable->charge ^ draft->charge) & Q_MASK) != 0 &&
196       ((stable->charge ^ draft->charge) & W_MASK) != 0 && sig1 !=
              sig2;
197     int c3 = (stable->charge & D_MASK) == 0 && (stable->charge &
            W_MASK) == 0 &&
198       ((stable->charge ^ draft->charge) & W_MASK) != 0;
199     int c4 = (stable->charge & D_MASK) == D_MASK &&
200       (stable->charge & W_MASK) == W_MASK &&
201       ((stable->charge ^ draft->charge) & W_MASK) != 0;
202     //
203     if(ISNULL(stable->pole))
204       COPY(draft->pole, stable->p);
205     //
206     // Play pseudo dices
207     //
208     if (stable->noise > abs(stable->phi) &&
209       draft->noise > abs(draft->phi) &&
210       (!ISNULL(stable->p) || !ISNULL(draft->p))) {
211       if (((stable->charge ^ draft->charge) & D_MASK) == 0) {
212         // Same sector?
213         //
214         if (stable->f == 1 && draft->f == 1) {
215           // F x F
216           //
217           if (((stable->charge ^ draft->charge) & Q_MASK) != 0) {
218             // Annihilation?
219             //
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
220             stable->b = (stable->b * draft->b) % SIDE2; // ???
                    erro ???
221             draft->b = stable->b;
222             //
223             // Reissue R1 and R2 from this
224             //
225             RESET(stable->pole);// ???
226             COPY(draft->pole, stable->p);
227           }
228           else if (sig1 || sig2 || sig3) {
229             //
230             // Similar?
231             //
232             // Cohesion
233             //
234             if (stable->b != draft->b) {
235               stable->b = (stable->b * draft->b) % SIDE2;
236               draft->b = stable->b;
237             }
238             //
239             // s1 <-> s2
240             //
241             int temp;
242             temp = stable->s[0];
243             draft->s[0] = stable->s[0];
244             stable->s[0] = temp;
245             temp = stable->s[1];
246             draft->s[1] = stable->s[1];
247             stable->s[1] = temp;
248             temp = stable->s[2];
249             draft->s[2] = stable->s[2];
250             stable->s[2] = temp;
251             //
252             // Reissue R1 from pole(R1) and R2 from pole(R2)
253             //
254             RESET(stable->o);
255             RESET(draft->o);  //???
256             COPY(draft->pole, stable->p); // ???
257           }
258         }
259         else if (stable->f > 1 && draft->f > 1) {
260           //
261           // B x B
262           //
263           if (((stable->charge ^ ~draft->charge) & C_MASK) == 0
                  &&
264             stable->code == draft->code && draft->code == GLUON)
                    {
265             //
266             // gluon-gluon?
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
267             //
268             // Swap colors
269             //
270             int temp = stable->charge & C_MASK;
271             stable->charge &= ~C_MASK;
272             stable->charge |= (draft->charge & C_MASK);
273             draft->charge &= ~C_MASK;
274             draft->charge |= temp;
275             //
276             // Reissue R1 from pole(R1)
277             //
278             RESET(stable->o);
279             draft->draft->dir = 0;  // replicar !!!
280             draft->t = 0; // replicar !!!
281           }
282           else if (!ISNULL(stable->p) && !ISNULL(draft->p)) {
283             if (c1 || c2 || c3 || c4)
284             {
285               // chiral?
286               //
287               // Reissue R1 and R2 from cp1
288               //
289               draft->b = stable->b;
290               RESET(stable->o);
291               //
292               // Reissue R2 from cp1
293               //
294               RESET(draft->o);
295             } else {
296               // TODO
297             }
298           }
299           else if (sig1 != 0 && sig1 != 3 && sig2 != 0 && sig2 !=
                3) {
300             int temp = stable->charge & C_MASK;
301             stable->charge &= ~C_MASK;
302             stable->charge |= (draft->charge & C_MASK);
303             draft->charge &= ~C_MASK;
304             draft->charge |= temp;
305             draft->b = stable->b;
306             //
307             // Reissue R1 and R2 from cp1
308             //
309             draft->b = stable->b;
310             RESET(stable->o);
311             RESET(draft->o);
312           }
313         }
314         if (stable->f == 1 && draft->f > 1) {
315           //
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
316             // F x B
317             //
318             if ((stable->charge & C_MASK) != 0 &&
319               (stable->charge & C_MASK) != C_MASK &&
320               (draft->charge & C_MASK) != 0 &&
321               (draft->charge & C_MASK) != C_MASK) {
322               int temp = stable->charge & C_MASK;
323               stable->charge &= ~C_MASK;
324               stable->charge |= (draft->charge & C_MASK);
325               draft->charge &= ~C_MASK;
326               draft->charge |= temp;
327               draft->b = stable->b;
328               //
329               // Reissue R1 from pole(R1) and R2 from pole(R2)
330               //
331               RESET(stable->o);
332               RESET(draft->o);
333             } else {
334               draft->b = stable->b;
335               //
336               // Reissue R1 and R2 from this
337               //
338               RESET(stable->pole);
339               RESET(stable->o);
340               RESET(draft->pole);
341               RESET(draft->o);
342             }
343           }
344           else if (stable->f > 1 && draft->f == 1) {
345             //
346             // B x F
347             //
348             if ((stable->charge & C_MASK) != 0 &&
349               (stable->charge & C_MASK) != C_MASK &&
350               (draft->charge & C_MASK) != 0 &&
351               (draft->charge & C_MASK) != C_MASK) {
352               int temp = stable->charge & C_MASK;
353               stable->charge &= ~C_MASK;
354               stable->charge |= (draft->charge & C_MASK);
355               draft->charge &= ~C_MASK;
356               draft->charge |= temp;
357               //
358               // Reissue R1 from pole(R1) and R2 from pole(R2)
359               //
360               RESET(stable->o);
361               RESET(draft->o);
362             } else {
363               draft->b = stable->b;
364               //
365               // Reissue R1 and R2 from this
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
366             //
367             RESET(stable->pole);
368             RESET(draft->pole);
369           }
370         }
371         else if (stable->b == draft->b) {
372           //
373           // Messenger interactions
374           //
375           if (!ISNULL(stable->p)) {
376             //
377             // REISSUE(stable, POLE(stable))
378             //
379             RESET(stable->pole);
380             //
381             // REISSUE(draft, TRANSPORT(draft, stable));
382             //
383             draft->pole[0] = stable->o[0] - draft->o[0];
384             draft->pole[1] = stable->o[1] - draft->o[1];
385             draft->pole[2] = stable->o[2] - draft->o[2];
386           } else {
387             //
388             // REISSUE(draft, POLE(draft));
389             //
390             RESET(draft->pole);
391             //
392             // REISSUE(stable, TRANSPORT(stable, draft));
393             //
394             stable->pole[0] = draft->o[0] - stable->o[0];
395             stable->pole[1] = draft->o[1] - stable->o[1];
396             stable->pole[2] = draft->o[2] - stable->o[2];
397           }
398         }
399       } else {
400         //
401         // Inter-sector
402         //
403         if ((((stable->charge & D_MASK) == 0 && sig1 == 2) || ((
               stable->charge & D_MASK) == 1 &&
404           sig1 == 3))) {
405           int temp = stable->charge & C_MASK;
406           stable->charge &= ~C_MASK;
407           stable->charge |= (draft->charge & C_MASK);
408           draft->charge &= ~C_MASK;
409           draft->charge |= temp;
410           //
411           // Reissue R1 and R2 from this
412           //
413           RESET(stable->pole);
414           RESET(draft->pole);
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
415           } else if (c1 || c2 || c3 || c4) {
416             // Chiral?
417             //
418             int temp = stable->charge & W_MASK;
419             stable->charge &= ~W_MASK;
420             stable->charge |= (draft->charge & W_MASK);
421             draft->charge &= ~W_MASK;
422             draft->charge |= temp;
423             //
424             // Reissue R1 and R2 from this
425             //
426             RESET(stable->pole);
427             RESET(draft->pole);
428           }
429         }
430       }
431     }
432
433     ////////// STEP5: EXPAND //////////
434
435       // Update tracking info
436       //
437       if (draft->ctrl > 0) {
438           // Track decay
439           //
440           draft->phi *= (1 - 1 / (2 * draft->t));
441           //
442           // Minsky circle algorithm
443           //
444           int xNew = draft->cosine - (draft->sine >> SHIFT);
445           int yNew = draft->sine + (draft->cosine >> SHIFT);
446           draft->cosine = xNew;
447           draft->sine = yNew;
448           //
449           draft->ctrl--;
450       }
451       //
452       // Spread cell contents if not empty
453       //
454       if (draft->f > 0) {
455           draft->t++;
456           Cell* neighbor;
457           //
458           // Momentum evolution
459           //
460           if (!ISNULL(draft->p) && draft->t * draft->t > draft->
                synch) {
461             //
462               // Select the only path for momentum
463               //
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
464             for (int dir = 0; dir < 6; dir++) {
465                 char vdir[3] = { 0, 0, 0 };
466                 neighbor = getPointer(dir, draft, (char *)vdir);
467                 //
468                 // Predict next pole value
469                 //
470                 char pole[3];
471                 COPY(pole, draft->pole);
472                 pole[0] -= vdir[0];
473                 pole[1] -= vdir[1];
474                 pole[2] -= vdir[2];
475                 //
476                 // Test if pole shrunk
477                 //
478                 if (MOD2(pole) < MOD2(draft->pole)) {
479                     neighbor->pole[0] = pole[0];
480                     neighbor->pole[1] = pole[1];
481                     neighbor->pole[2] = pole[2];
482                     //
483                     neighbor->t = draft->t;
484                     neighbor->f = draft->f;
485                     neighbor->b = draft->b;
486                     neighbor->charge = draft->charge;
487                     //
488                     neighbor->o[0] = draft->o[0] + vdir[0];
489                     neighbor->o[1] = draft->o[1] + vdir[1];
490                     neighbor->o[2] = draft->o[2] + vdir[2];
491                     //
492                     COPY(neighbor->s, draft->s);
493                     COPY(neighbor->p, draft->p);
494                     //
495                     // Synchronize to keep inside spherical
                            wavefront
496                     //
497                     neighbor->synch = LIGHT2 * MOD2(neighbor->o);
498                     int t = neighbor->t;
499                     if (ISNULL(neighbor->pole)) {
500                         neighbor->t = 0;
501                         neighbor->synch = LIGHT;
502                         neighbor->f = 1;
503                         neighbor->b = 0;
504                         neighbor->code = 0;
505                         RESET(neighbor->o);
506                         //
507                         // This characterizes free propagation
508                         //
509                         COPY(neighbor->pole, neighbor->p);
510                     }
511                     //
512                     // Keep only a single copy of momentum
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
513                         //
514                         RESET(draft->p);
515                         //
516                         // When momentum moves, it casts a spherical
                                perturbation
517                         //
518                         draft->t = 0;
519                         draft->synch = LIGHT;
520                         draft->f = 1;
521                         draft->b = 0;
522                         draft->code = 0;
523                         RESET(draft->o);
524                         break;
525                     }
526                 }
527             }
528             //
529             // Phase cells spread synchronized
530             //
531             if (draft->t * draft->t > draft->synch) {
532               //
533                 // Explore von Neumann directions
534                 //
535                 Cell* neighbor;
536                 for (int dir = 0; dir < 6; dir++) {
537                     char vdir[3] = { 0, 0, 0 };
538                     neighbor = getPointer(dir, draft, (char*)vdir);
539                     //
540                     if (!ISNULL(neighbor->p))
541                         continue;
542                     //
543                     // Test if branch is legal
544                     //
545             //
546             // Test if branch is legal
547             //
548
549             boolean allowed = false;
550             //
551             // Calculate new origin vector
552             //
553             int x = o[0] + vdir[0];
554             int y = o[1] + vdir[1];
555             int z = o[2] + vdir[2];
556             //
557             // Test for expansion
558             //
559             int d1 = MOD2(o);
560             int d2 = x * x + y * y + z * z;
561             if (d2 <= d1)
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
562             goto TEST;
563           //
564           // Wrapping test
565           //
566           if (x == S + 1 || x == -S || y == S + 1 || y == -S || z
                ==S + 1 || z == -S)
567             goto TEST;
568           //
569           // Root allows all six directions
570           //
571           int level = abs(x) + abs(y) + abs(z);
572           if (level == 1) {
573             allowed = true;
574             goto TEST;
575           }
576           //
577           // x axis
578           //
579           if (x > 0 && y == 0 && z == 0 && dir == 0) {
580             allowed = true;
581             goto TEST;
582           } else if (x < 0 && y == 0 && z == 0 && dir == 1) {
583             allowed = true;
584             goto TEST;
585           }
586           //
587           // y axis
588           //
589           else if (x == 0 && y > 0 && z == 0 && dir == 2) {
590             allowed = true;
591             goto TEST;
592           } else if (x == 0 && y < 0 && z == 0 && dir == 3) {
593             allowed = true;
594             goto TEST;
595           }
596           //
597           // z axis
598           //
599           else if (x == 0 && y == 0 && z > 0 && dir == 4) {
600             allowed = true;
601             goto TEST;
602           }
603           else if (x == 0 && y == 0 && z < 0 && dir == 5) {
604             allowed = true;
605             goto TEST;
606           }
607           //
608           // xy plane
609           //
610           else if (x > 0 && y > 0 && z == 0) {
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
611             if (level % 2 == 1)
612               allowed = (dir == 0 && draft->dir == 2);
613             else
614               allowed = (dir == 2 && draft->dir == 0);
615           }
616           else if (x < 0 && y > 0 && z == 0) {
617             if (level % 2 == 1)
618               allowed = (dir == 1 && draft->dir == 2);
619             else
620               allowed = (dir == 2 && draft->dir == 1);
621           }
622           else if (x > 0 && y < 0 && z == 0) {
623             if (level % 2 == 1)
624               allowed = (dir == 0 && draft->dir == 3);
625             else
626               allowed = (dir == 3 && draft->dir == 0);
627           }
628           else if (x < 0 && y < 0 && z == 0) {
629             if (level % 2 == 1)
630               allowed = (dir == 1 && draft->dir == 3);
631             else
632               allowed = (dir == 3 && draft->dir == 1);
633           }
634           //
635           // yz plane
636           //
637           else if (x == 0 && y > 0 && z > 0) {
638             if (level % 2 == 0)
639               allowed = (dir == 4 && draft->dir == 2);
640             else
641               allowed = (dir == 2 && draft->dir == 4);
642           }
643           else if (x == 0 && y < 0 && z > 0) {
644             if (level % 2 == 0)
645               allowed = (dir == 4 && draft->dir == 3);
646             else
647               allowed = (dir == 3 && draft->dir == 4);
648           }
649           else if (x == 0 && y > 0 && z < 0) {
650             if (level % 2 == 0)
651               allowed = (dir == 5 && draft->dir == 2);
652             else
653               allowed = (dir == 2 && draft->dir == 5);
654           }
655           else if (x == 0 && y < 0 && z < 0) {
656             if (level % 2 == 0)
657               allowed = (dir == 5 && draft->dir == 3);
658             else
659               allowed = (dir == 3 && draft->dir == 5);
660           }
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
661             //
662             // zx plane
663             //
664             else if (x > 0 && y == 0 && z > 0) {
665               if (level % 2 == 1)
666                 allowed = (dir == 4 && draft->dir == 0);
667               else
668                 allowed = (dir == 0 && draft->dir == 4);
669             }
670             else if (x < 0 && y == 0 && z > 0) {
671               if (level % 2 == 1)
672                 allowed = (dir == 4 && draft->dir == 1);
673               else
674                 allowed = (dir == 1 && draft->dir == 4);
675             }
676             else if (x > 0 && y == 0 && z < 0) {
677               if (level % 2 == 1)
678                 allowed = (dir == 5 && draft->dir == 0);
679               else
680                 allowed = (dir == 0 && draft->dir == 5);
681             }
682             else if (x < 0 && y == 0 && z < 0) {
683               if (level % 2 == 1)
684                 allowed = (dir == 5 && draft->dir == 1);
685               else
686                 allowed = (dir == 1 && draft->dir == 5);
687             }
688             else {
689               // Spirals
690               //
691               int x0 = x + S;
692               int y0 = y + S;
693               int z0 = z + S;
694               //
695               switch (level % 3) {
696                 case 0:
697                   if (x0 != S && y0 != S)
698                     allowed = (z0 > S && dir == 4) || (z0 < S && dir
                          == 5);
699                   break;
700                 case 1:
701                   if (y0 != S && z0 != S)
702                     allowed = (x0 > S && dir == 0) || (x0 < S && dir
                          == 1);
703                   break;
704                 case 2:
705                   if (x0 != S && z0 != S)
706                     allowed = (y0 > S && dir == 2) || (y0 < S && dir
                          == 3);
707                   break;
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 21 September 2021 doi:10.20944/preprints202107.0513.v2

```
708                 }
709               }
710             }
711                     TEST:
712             if(allowed)
713             {
714                         neighbor->t = draft->t;
715                         neighbor->dir = dir;
716                         neighbor->f = stable->f;
717                         neighbor->b = stable->b;
718                         neighbor->charge = stable->charge;
719                         //
720                         neighbor->o[0] = stable->o[0] + vdir[0];
721                         neighbor->o[1] = stable->o[1] + vdir[1];
722                         neighbor->o[2] = stable->o[2] + vdir[2];
723                         //
724                         COPY(neighbor->s, stable->s);
725                         //
726                         // Schedule for spherical evolution
727                         //
728                         neighbor->synch = LIGHT2 * MOD2(neighbor->o);
729             }
730                     //
731                     draft->f = 0;
732               }
733           //
734           draft->f = 0;
735           RESET(draft->p);
736       }
737   }
```