1. Introduction and Context

Biological systems have many characteristics that allow us to consider them alive. One important definition of a living system was offered by Schrödinger: “When it goes on ‘doing something’, moving, exchanging material with its environment, and so forth, and that for a much longer period than we would expect an inanimate piece of matter to ‘keep going’ under similar circumstances.” (Schrödinger, 74). All of these actions may be summed up in the term metabolism, or controlled chemical reactions. While there is the concept of the second law of thermodynamics, that is, the tendency for entropy to increase in a closed system, biological organisms, being open systems, succeed in persisting by metabolizing in a purposive manner: “the essential thing in metabolism is that the organism succeeds in freeing itself from all the entropy it cannot help producing while alive.” (Schrödinger, 76).

Therefore, the hallmark of a living organism can be defined as is its ability to maintain its structure (in a localized decrease in entropy) through the cyclical metabolic process. When these processes end, the creature dies, and we are left with a piece of matter that no longer is able to effect this localized reversal of entropy.
This is all well and good when it comes to identifying an organism as alive. But what of the origins of life? The origins of life are studied through a number of means, including the use of micropaleontology to try to examine what actually happened billions of years ago on Earth, re-creation of certain situations using \textit{in vitro} experiments, and computational simulations. This last area uses many different techniques to arrive at a picture of how life might have originated. \textit{Theoretical computational biology} (this area is often related to artificial life) attempts to find analogies between theoretical computational constructs and biological ones, in order to create informed simulations and models.

In looking at prebiotic systems, we are left with the task of arriving at ordered structures, namely organisms, from very simple and rather disordered parts. In addition, as discussed above, the hallmark of biological structures is that they function using a metabolism to create and maintain order through the uptake of free energy and the release of waste into the environment. Therefore, we can simply seek a sustained decrease in entropy for an open system. A system with such an energetics would be a reasonable one to examine further, and perhaps even be called alive. Therefore, what should be sought is a suitable computational substrate within which to allow a metabolizing system to persist with low entropy.

As simple biotic systems arose from their chemical precursors, a reasonable system to model is that of chemistry. Since the relevant calculations in chemistry are often prohibitively difficult, what can often be done is to abstract the relevant properties of a chemistry and create a bare minimum chemical system known as an \textit{artificial chemistry}. An artificial chemistry is one where the principles of chemistry, such as interaction and recombination, are abstracted and very simple computational primitives are used. These primitives could be anything from strings to matrices, and, in fact, there have been a burgeoning number of these recently (Dittrich et al.).
However, one aspect that has been absent from these substrates is a unified conception of energy. Some models have microscopic reversibility (a necessity based on chemistry and physics) built into them, yet lack a notion of energy and its conservation. Others might have both these concepts but do not have the complexity required to create new and novel structures. In addition, the concept of entropy has not been encompassed within the artificial chemistries currently in existence (see related work in a section below).

This paper describes an artificial chemistry that has a useful, reasonable and broad analogue to energetic notions. This artificial chemistry, called erg, is what might be termed a computational energetics. In addition, the long-term goal is a search for a ruleset that converges on the pair of conditions discussed above: a system that, when closed, increases in entropy (presumably only up to a certain point of equilibrium), but when open, the portion of the system being examined decreases in entropy as a marker for biotic-like behavior. This goal, broadly viewed, is in fact one of the Open Problems in Artificial Life, as enumerated by Bedau et al.

However, before this search can be done, this system, like any novel substrate, must be investigated, and its properties must be examined. What has been done is a preliminary examination of the erg system and the characteristics of a number of rulesets in order to illuminate its properties, such as the effects of the nature of the ruleset on behavior. These results are presented, along with conclusions and further directions for this research program.

**erg: a Computational Energetics**

In creating erg to have a notion of computational energetics, we must look at what is meant by energetics. Energy, as well as its associated concepts, must be defined and abstracted, in order that a computational equivalent of these ideas can be created. First of all, energy is
defined as the capacity to do work. Here we can define it specifically as the capacity to change the surrounding system. Energy can come in two forms: potential energy and kinetic energy.

Kinetic energy is a measure (a microscopic one rather than a macroscopic measure like temperature) of the amount of energy of motion. Potential energy is the remaining energy in the closed system that is “bound up.” Potential energy often takes the form of energy of position or of structure. Within erg, kinetic energy is based on the speed and the size of the particles/molecules, while potential energy is found in the structure (size and arrangement) of the molecules (see next section for details). Due to erg containing a large simplification of chemistry and no non-reactive collisions, kinetic and potential energy are only exchanged upon interaction. Kinetic energy is distributed based on an inverse relationship (the more KE in a molecule, the less energy that is bound up, or PE, within the molecule, just as in physics).

Furthermore, just as in physical systems there are the notions of conservation of energy and conservation of matter, as well as interactions that are reversible on a microscopic scale, these are all found in erg (Pauling). While the amount of kinetic energy and potential energy is constantly changing within the closed erg system, the overall energy is constant. In addition, if the erg system is closed, there will always be a constant number of atoms within the system, although the number of molecules may change. Similarly, the reactions that combine and split particles are all entirely reversible, as specified by the ruleset of the system.

With these energetic concepts in place, the entropy of an erg simulation may be determined. Generally, entropy is a measure of the dispersal of energy within a system and may be viewed as the metric for the probability distribution of this energy. In both information theory and thermodynamics, the more uncertain the localization of the energy (i.e. the more missing information and the more even the probability distribution), the higher the entropy (Baierlein).
In statistical mechanics, the method of measuring entropy is one of measuring the number of microstates of the system (and then taking the logarithm multiplied by a constant). A microstate is one possible energy arrangement of the system. The higher the number of possible arrangements, the higher the entropy (Baeirlein; Pauling).

In erg, the entropy is calculated based on the distribution of the kinetic energy and potential energy of the particles in the system. The system can be closed, or this simply can be a measure within a certain boundary. In this way, a reasonable measure for entropy may be determined at any timestep.

While this lays the foundation for an understanding of a computational energetics, it is not clear that simply any ruleset for erg will satisfy the conditions necessary for biological systems. Therefore, as mentioned above, the rulesets were examined using a stochastic search, as a first analysis, to attempt an analysis of the ruleset space, and the possible discovery of a ruleset necessary for the conditions of closed system entropy increase and open system entropy decrease.

It must be noted that this ruleset space traversal via search has physical plausibility and is not simply an arbitrary mechanism to get the results desired. Within Nature, the number of possible species that can be produced is very large due to the rich collection of physico-chemical laws. The complexity of Nature provides for the innumerable permutations of chemistry, which is the fodder for the origins of life. With such a simplified abstraction of chemistry found in erg, the ruleset selection process is simply acting as the equivalent of the traversal of the rich rule-space of Nature. In this way, the relevant subset of chemical rules that gives complexity and simple prebiotic characteristics can be searched for within the erg system.
2. Materials and Methods

Details of the erg System

Overview and Energetics

The erg system website can be found at http://purl.oclc.org/NET/erg When erg is viewed, it appears as a two-dimensional reactor of interacting rings or circles. The internal structure can be viewed for each molecule, or to make the system run faster, it can also be simply viewed as circles:

![Two views of the erg system. Left: the internal structure of the molecules is visible. Right: the structures are viewed as colored circles to allow the simulation to run more quickly.](image)

Each particle or molecule in the erg system (these two terms, as well as the term “ring,” will be used interchangeably) consists of a ring of atomic components, or atoms. There are seven types of atoms, represented by a number and a color, the colors being purely for aesthetic effect. The number of atoms within each particle is represented by $m$ (size is considered to be roughly
analogous to mass). Throughout the simulation, while the number of particles may change, the number and types of the atoms in the simulation (if it is a closed system) remains constant. This is one of the criteria (albeit for open-ended evolution) that is discussed in a paper by Suzuki et al.

Figure 2. Atoms are numbered 0-6 (here the numbers are shown along with the colors). The active site is the rightmost atom but is not indicated visually differently in any way. The size $m$ is equal to the number of atoms, which in this case is 26.

At any given timestep, a ring has one active atom, known as the active site, which is where reactions may take place. Each molecule has a speed ($v$) that is initially determined by a random function at the outset of the simulation, or is based on conservation laws, as explained in the Reaction section below.

Each molecule also has a direction (angle of motion) and a direction of rotation (either clockwise or counterclockwise), which is determined by a simple function of the atoms within the molecule, as seen below (this function is calculated at the time of the particle’s creation). The rotation allows the active site to change in a predictable manner:

angle of motion (in radians) = sum of atom-directional values mod 2*PI

rotation = sum of atoms (if even, clockwise / if odd, counterclockwise)

\[ d_0 = -2, \ d_1 = -2, \ d_2 = -1, \ d_3 = -0, \ d_4 = 1, \ d_5 = 2, \ d_6 = 2 \]

Atom-directional values for each type of atom
The kinetic energy (KE) of each particle is proportional to a value based on the speed and size of the particle. Potential energy (PE) is a function of the structure of the molecule and its size (using information theory calculations). Due to conservation of energy, for a closed system

\[ PE + KE = E_T = \text{constant value} \]

Since reaction reversibility is based on structure, the kinetic energy and speed of the products is dictated by the PE calculations (this is a curious and thought-provoking side-effect). The kinetic energy and potential energy of each particle is determined as follows \((i)\) indicates it is for the individual particle and not for the system):

\[ KE_i = \frac{1}{2} m_i v_i^2 \] (a similar formula is used in conventional dynamics)

\[ PE_i = H_i m_i \]

\(H\) is the measure of informational entropy for the particle (the Shannon entropy), and is calculated for both first order and second order approximations and then summed (going any higher is a prohibitively computationally intensive calculation) (Baierlein; Togneri and deSilva):

\[ H = - \sum p_j \log p_j \]

\(j\) is the number of each atomic group (either first-order or second-order)

As mentioned above, entropy is calculated as the distribution of the potential and kinetic energy of the system’s particles. The entropy of the two types of energy are calculated separately and later summed (as will be explained below); entropy \(\text{KE}\) will be used as an example.
The calculation begins with knowing the total KE in the system (which is simply the sum of each molecule’s KE). Intuitively, if all the KE were found in only a single particle, this would be the microstate with the lowest entropy, since there is an entirely unequal distribution of energy.

In statistical thermodynamics, often, macroscopic measures such as temperature are used to calculate the entropy of the system. These measures, along with the Boltzmann distribution, help scientists to arrive at calculations for the amount of entropy for the system. However, since in erg there is complete knowledge of the system, a different calculation may be used, which is based on calculating the number of microstates. The entropy formula is as follows (Pauling):

\[ S = k \ln W \] (where S is entropy, k is a constant and W is the number of microstates)

W may be calculated based on the following equation:

\[ W = \prod_{i=1}^{N} \frac{N_i}{N!} \]

N = number of molecules
N_i = number of molecules in the same energy partition
n = possible energy partitions (ranging from 0-KE_T)

\[ S = \log KE \left( \prod_{j=1}^{N} n^{N_j} \right) + \log PE \left( \prod_{j=1}^{N} n^{N_j} \right) \]

Energy partition bins, which are of arbitrary but equal size and are simply a way of grouping the energy states since they are not distinct quanta, are created. Next, after determining how many particles fit into each bin, with the commensurate probabilistic mathematics (as per the above equation), we arrive at the number of microstates. The logarithm is then taken (an approximation of this is simply arrived at by returning the number of digits in the number, which
approximates the base-10 logarithm), and we receive the kinetic entropy. A similar calculation is made for the potential entropy and these two measures are summed. In this way, erg has a measurable entropy value at any given timestep, for both a closed system and any spatial boundary of a system.

Reactions

There are two types of reactions in the erg system: combinations (or fusions) and fissions (or splits). Combinations occur when two particles are overlapping positionally and have compatible active sites (based on the reaction ruleset). Fissions occur when a single particle is found to be unstable (according to a simple symmetry function) and has the appropriate active sites available for reaction.

A ruleset is made up of rules of the following form, where each letter stands for an atom:

1. A B + C D
2. E F + G H

Each rule is made up of two rulesites (the two contiguous atoms that must be present on the particles in order for a fusion to occur). In order for a fusion reaction to occur, one particle must have an active site that is part of one rulesite of a rule, and the second particle must have an active site that is part of the other rulesite of the rule. In addition, the two particles can only interact if they are overlapping (this means that sometimes particles can initially not be able to interact even though they are overlapping, since their active sites are constantly rotating). The two particles combine such that the rulesites are split in half and recombined:
**Before:** Particle 1: 0 2 3 5 6 2  
Particle 2: 4 5 6 1 2 4 5

**After:** Particle 1': 0 4 5 4 2 1 6 5 2 3 5 6 2

The particles are shown in linear form, with their active sites at the far left. The underlined and bolded portions are the rulesites for the rule \([0 2 + 4 5]\).

While there are not actually discrete steps, they can be presented as such in order that the nature of the interaction can be seen.

**Step 1:** The two particles before interaction.

**Step 2:** The two particles have the proper activated sites for interaction (particle 2 has been inverted for ease of seeing this abstract interaction)

**Step 3:** The original particles are destroyed and the new bonds are the only ones remaining.

**Step 4:** The direction of the new particle is as displayed above.

**Step 5:** The newly created particle which is the reaction product.

Figure 3. A schematic of the fusion interaction.

Similarly, when a particle is found to be unstable, the particle will simply split back into its constituent particles, provided that the active site is the first component of one of the rulesites.

Stability is calculated by looking at the pseudo-symmetrical nature of the particle:
A function looks at the first half of the atoms and checks to see if the atom that is approximately across from it within the ring has the same parity (is odd or even). If they all have the same parity, then the molecule is stable. Otherwise, it is unstable (further examples of rulesets and their properties are provided in the Results section).

The system can be either bounded (closed) or unbounded (open). If it is closed, then the particles simply loop around to the other side, giving the system a torus-like topology. They loop around when their centers are beyond the boundaries of the system; as a byproduct of this, since position is dictated by center position, a large molecule near the edge would still be unable to interact with the other side. If the system is open, upon the center of a particle leaving the bounds, a new randomly generated particle is created at the edge of the system. In this way, the number of particles is kept relatively stable, i.e. the number of particles is only increased through interactions occurring within the system. This preserves the locality of density changes in an open system (if the local system increases or decreases the density, the surrounding system does not necessarily wipe these changes out).

A shortcoming of the system is that the calculation of the kinetic and potential energy suffers from various counter-intuitive notions as well as certain features, which might even be construed as last-minute hacks. One that was mentioned above is that the kinetic energy is calculated based on the potential energy, and not the other way around. While this has interesting theoretical ramifications, there were also certain practical consequences, such as forbidding certain interactions if the energy constraints would yield a negative kinetic energy. Even with these constraints included, it was noticed that there were occasional incidences of negative kinetic energies; this was remedied with a quick fix (the particle was given purely potential energy and its kinetic energy was set to zero – without regard to its velocity). Clearly, this fix is
neither desirable nor physically plausible, and even ruins the reversibility, but it seems that the condition for this fix was transient enough that it did not affect the system in any large fashion. In addition, there are some divide-by-zero issues, which are mapped to zero. This seems to be a common solution to this issue, however, and is not a large concern. However, any new version of erg simply cannot have features that disrupt the reversibility and conservation laws, and will be addressed in the concluding section.

**Details of implementation**

The simulation was created in the Processing system by Ben Fry and Casey Reas (found at http://www.processing.org/). This system provides a simple mechanism for creating graphical applets, complete with multithreading, quickly and efficiently. It is a wrapper for Java and the languages are nearly identical.

The random generation of rulesets and entropy analysis were done entirely within Java since no graphics were needed. In addition, this allowed for quick simulation and running the evolutionary system on a separate machine. In addition, any chosen rulesets can then simply be placed within the code of the Processing system and be visually simulated.

Erg source code for the closed system in the Processing environment is found in the Appendix. Other code is downloadable from the erg website at http://purl.oclc.org/NET/erg
Erg as Compared to Previous and Related Work

There have been two intriguing artificial chemistries that examine the edge of abiotic and biotic behavior, in this case, self-replication. The first, Squirm3, while very interesting and powerful, lacks a structural theory of the particles in the system, limiting the possibilities of interaction (erg is more open-ended and similar to physical chemistry in this respect) (Hutton). In addition, Squirm3 is on a rectilinear grid, which prevents the full range of movement in some cases. Squirm3 also has no notion of energy whatsoever. JohnnyVon, the second artificial chemistry system, does not suffer from the deficiency of a grid (it uses a more reasonable approximation of physics), has an approach to energy, including some notion of potential energy, but each particle is a cellular automaton, which is unlike the unchanging atoms that are found within erg (Smith et al). In addition, both models do not have a measure of entropy of the system, something that is integral to erg.

Benkö’s toy model of chemistry includes a reasonable notion of entropy and even mentions the possibility of an open-ended chemical system, but deems it too computationally-intensive and leaves it unimplemented. (Benkö et al.) Adami has measured entropy of self-replicating agents; unfortunately, this is already past the boundary that separates biotic and abiotic behavior (erg aims to measure entropy in an artificial chemical system independent of biotic behavior) (Adami et al). Toffoli discusses invertible cellular automata, which contain microscopic reversibility, a similar requirement in the erg system (Toffoli and Margolus). Toffoli even mentions the potential for entropy investigation within this CA substrate. It does not appear however that an entropy analysis is ever done. In addition, erg approaches entropy and energetic simulation within a different substrate, that of an artificial chemistry.
3. Results

With the erg system in place, it was deduced through initial test runs that a simple genetic algorithm would be insufficient, not to mention extremely slow, yielding little useful results. Therefore, what was undertaken, as mentioned in the Introduction, was a preliminary analysis of the system using a stochastic traversal of the rulespace. The various analyses of the erg system, and their results will be elaborated below.

Initial Enumeration of the Entropy-Space

Initially, 100 random rulesets, with an average of approximately 20 rules each, were created. They were each run on the same initial system for 100 timesteps (this small number was used so as to get a quick initial measure of the space) as a closed system, followed by 100 timesteps as an open system. The entropy change during these two periods was measured and then graphed as a scatterplot, seen below [note: all of these values are entropy changes, not entropy measures]:

![Entropy for Closed System vs. Open System](image-url)
Figure 4. Entropy changes in a closed system versus the changes in an open system. There are 100 rulesets, each with an average number of rules of approximately 20.

Next, this scatterplot was converted into a three-dimensional plot, with the height of each point representing the size of the ruleset for each run:

![3D Scatterplot](image)

Figure 5. Entropy changes in a closed system versus the changes in an open system versus ruleset size. There are 100 rulesets, each with an average number of rules of approximately 20.

Plots of the size of the rulesets, as relative to the entropy changes for both the closed and open systems, can be seen below:
Figure 6. Ruleset size versus entropy changes in a closed system. There are 100 rulesets, each with an average number of rules of approximately 20.

Figure 7. Ruleset size versus entropy changes in an open system. There are 100 rulesets, each with an average number of rules of approximately 20.

Clearly, there are some interesting trends. However, to see if these trends continue for larger rulesets, a run similar to the one above, but with 81 rulesets of an average of approximately 100 rules each, was run. The resulting graphs generated from these data are seen below:
Figure 8. Entropy changes in a closed system versus the changes in an open system. There are 81 rulesets, each with an average number of rules of approximately 100.

Figure 9. Ruleset size versus entropy changes in a closed system. There are 81 rulesets, each with an average number of rules of approximately 100.
Figure 10. Ruleset size versus entropy changes in an open system. There are 81 rulesets, each with an average number of rules of approximately 100.

Figure 11. Entropy changes in a closed system versus the changes in an open system versus ruleset size. There are 81 rulesets, each with an average number of rules of approximately 100.
It would appear from these graphs that, for randomly generated rulesets, the trend of a ruleset is that the larger the size, the more predictable the change in entropy. This appears to reach a limit as ruleset size becomes large, with the changes in entropy for both a closed and open system approaching a constant value. One rationale for this is that the larger the ruleset, the quicker the interactions. With the number of timesteps for each run being the same and the only variable being the rulesets, the larger the number of rules allows for an increased number of reactions, until the final entropy is reached more rapidly. Unfortunately, it does not appear that any of the rulesets achieved our criteria of an increase in entropy in a closed system, with a decrease in entropy in an open system. Perhaps the location for satisfaction of these criteria is to be found in rulesets of a small size (although even this does not seem promising).

**Particle Size Analysis**

Another type of analysis was conducted after these initial analyses were completed. Random rulesets were generated to see how the size of particles changed over time (a closed system only was used in these runs). A certain type of graphical representation was created for viewing these data, wherein the x-axis represents the timestep, the y-axis represents the size of the particles, and the color represents the number of particles of that size (black means fewer and white means more). These representations can be seen below:
Figure 12. Run 1 for small ruleset. The x-axis is timestep, y-axis is particle size, color is number of particles of that size (black is fewer, white is more).

Figure 13. Run 2 for small ruleset. The x-axis is timestep, y-axis is particle size, color is number of particles of that size (black is fewer, white is more).
Figure 14. Run 1 for large ruleset. The x-axis is timestep, y-axis is particle size, color is number of particles of that size (black is fewer, white is more).

Figure 15. Run 2 for large ruleset. The x-axis is timestep, y-axis is particle size, color is number of particles of that size (black is fewer, white is more).
Figure 16. The rules that comprise the small and large rulesets.

The final sizes are graphed below:

Figure 17. Final particle size distribution for the two runs for both the small and large rulesets.
What can be seen from the view of the erg system using these visualizations is that there is a trend towards large numbers of small particles (thereby increasing entropy). If the ruleset is larger, this trend is simply more pronounced. This is in line with the previous result that a larger result simply allows for quicker arrival to a high level of entropy. This characteristic, that the systems increase in entropy easily towards an apparent maximum, will be discussed in the next section.
4. Conclusion and Further Directions

As can be seen from the Results section, erg is a system with a great deal of behavior that is similar to the physical world. Much as systems increase in entropy, including a trend towards small particles, so too does the erg system. In addition, erg demonstrates that, at least through random search of the space of rulesets, the discovery of an open system whose entropy decreases is a very difficult task. This is an extremely reasonable conclusion based on our usual notion of life – it did not arise over a short period of time, but required an extremely long period of time during which “random natural experiments” were conducted, prior to the appearance of the first self-replicating biological agent.

One clear further direction of this research is the search for a ruleset which does satisfy the above requirement for “life-like” behavior. This would be an exceedingly exciting result indeed, but would most likely require an enormous amount of computing time to be accomplished. Some sort of genetic algorithm might be used, such as ones with multiple objectives, which help to eliminate the possibility of getting stuck in what are termed “mediocre stable states.”

If such a system is found, then the resulting components can be looked at for their informational content. This is because once a system achieves biotic behavior, the informational component is considered to be dominant in relation to the thermodynamic importance of the system (Rasmussen et al.). In a system with locally decreasing entropy, the internal structure of the molecules might be expected to have more complicated structural elements (the informational entropy of the particles in this system might be higher). This is an important question that should be investigated.
As can be seen, while erg is a good first foray into the arena of creating a computational energetics for an artificial chemistry, it suffers from a number of shortcomings. Broadly, one is its complexity. While erg is certainly an abstract chemistry, it is a rather complicated and computationally-intensive one. Due to its complexity, the rulespace is extremely high-dimensional and complex, which prevents simple analysis of its behavior through a quick exploration of the rulespace. However, it still demonstrated various characteristics that are physically reasonable.

In addition, the mathematically rigorous analysis of this system is prohibitive due to its somewhat ad-hoc nature. While the rules are simple and elegant, because rotating rings are not a known mathematical entity, creating a new analytical framework would have been very difficult. Similarly, the various methods of simulating an open system must be examined, since there are more than simply the one method that was chosen for erg, and possibly more physically reasonable ones as well. In addition, as mentioned earlier, there are a number of issues surrounding the calculation of the kinetic and potential energy in the system that cause unacceptable deviations from theoretical considerations and must be addressed in any future version of erg.

What would a next version of erg look like, then? What is desired is an erg-like system, one with a holistic and rich notion of energy and entropy, but one that is elegant, simple and mathematically and computationally tractable. Some possible plans for future erg-systems include various techniques to speed up the simulations, such as those taken from computational chemistry. These include the use of Metropolis-like methods if exact simulation might be intractable, as well as the possible use of a Gillespie model for coupled reactions (Beichl and Sullivan; Gillespie). Also, the toy-model of chemistry, while computationally intractable to use
for something similar to erg yet still encapsulate all of chemistry, might be useful for a more limited and abstract artificial chemistry (Benkő et al.).

Of course, these shortcomings do not detract from the research program that has been undertaken here. There are certain design principles behind erg which have been stressed. These include having a built-in notion of energy and entropy, the possibility for open-ended molecular construction and interaction, and the flexibility of many different types of rulesets for varied functionality.

Biologists and chemists recognize the primary importance of examining the structured flow of energy and the nature of entropy within simple biotic systems and the origins of life. Erg, then, is a first foray into bringing computational abstractions, namely artificial chemistries, to bear on this intriguing conceptual issue. More importantly, it has some physical realism, with behavior similar to chemical systems found in nature. Hopefully, more analysis and further models will provide additional insight into the part that energetic notions, such as entropy, play in the origins of life.
5. Works Cited


6. Appendix

Processing source code for erg (all source code, including Java source code, may be downloaded at http://purl.oclc.org/NET/erg)

// Sam Arbesman

// Rotating Ring Artificial Chemistry

color[] colors = new color[10];
int[] directions = new int[10];

// variable loading
int numRings; int maxInit; int gridJump; int minSizeRings;
int myWidth, myHeight; int maxNumRings; BFont metaBold; BFont titleFont;
int[] hist;
int[] histPE;
int[] histKE;
int[][] spatialSize; int[] spatialNumber; float[][] spatialDensity;
int subdivision;
boolean seeInnards;
energetics energy;
Vector theCombineRules;
Interact interaction;
int numstepssofar;

Ring[] rings;
Stack freeRings;

// initialization and setup
void setup()
{
    size(600, 500);
    metaBold = loadFont("Univers55.vlw.gz");
    titleFont = loadFont("Futura-Book.vlw.gz");
    myWidth = width - 200;
    myHeight = height - 100;
    subdivision = 40;
    numstepssofar = 0;

    colorMode(RGB, 255);
    background(0);

    smooth();
    ellipseMode(CENTER_DIAMETER);
    noStroke();

    colors[0] = color(255, 128, 0); // a0
    colors[1] = color(255, 196, 0); // b1
    colors[2] = color(179, 179, 179); // c2
    colors[3] = color(0, 76, 5); // d3
    colors[4] = color(205, 0, 25); // e4
    colors[5] = color(51, 0, 205); // f5
    colors[6] = color(80, 26, 128); // g6

    directions[0] = -2;
    directions[1] = -2;
    directions[2] = -1;
    directions[3] = 0;
directions[4] = 1;
directions[5] = 2;
directions[6] = 2;

theCombineRules = new Vector();
energy = new energetics();

// the rules of combination
theCombineRules.add(new combineRule("0 0","0 0");
theCombineRules.add(new combineRule("1 4","6 4");
theCombineRules.add(new combineRule("0 2","4 2");

interaction = new Interact(theCombineRules,1);

// initialize some rings

numRings = 200;
maxInit = 30;
gridJump = 5; // initial temperature or so
minSizeRings = 2;
maxNumRings = 2000;
seeInnards = false;

rings = new Ring[maxNumRings];
freeRings = new Stack();

int theLength = 0,xpos = 0,ypos = 0;
String temp = new String();
float dx,dy,rr,diff;

for(int i=0;i<numRings;i++)
{
    theLength = int(random(minSizeRings,maxInit));
    temp = "";
    for(int j=0;j<theLength;j++)
    {
        temp = temp + String.valueOf(int(random(7))) + " ";
    }
    temp = temp + String.valueOf(int(random(7)));
    xpos = int(random(myWidth));
    ypos = int(random(myHeight));
    rings[i] = new Ring(temp,xpos,ypos);
}

for(int w=numRings;w<maxNumRings;w++)
    freeRings.push(new Integer(w));

void keyReleased()
{
    if (key == 'i' || key == 'I')
    {
        if(seeInnards)
            seeInnards = false;
        else

seeInnards = true;
}
}

// main loop
void loop()
{
  background(0);

  float dx,dy,rr,diff,rA;
  String newSymbols = new String();
  int theSize= numRings;
  Vector newRings = new Vector();
  int numDots = 0;
  int numberSymbols = 0;
  for(int i = 0; i<maxNumRings; i++)
  {
    if(rings[i] != null)
    {
      rings[i].drawRing();
      rings[i].updateRing();

      //////////////
      /// Split interaction
      //////////////
      if(rings[i].isStable() != true && rings[i].stale != true)
      {
        Vector newSmaller = interaction.split(rings[i]);
        Ring unstable = rings[i];
        int originalLength = unstable.getNumSym();
        float xposition = unstable.getXpos();
        float yposition = unstable.getYpos();
        float originalPE = unstable.PE;
        float originalKE = unstable.KE;
        float remainingE = originalPE + originalKE;

        if(newSmaller != null)
        {
          String A = (String)newSmaller.elementAt(0);
          String B = (String)newSmaller.elementAt(1);

          float thePE = energy.calculatePE(unstable.getSymArray());

          if(thePE < remainingE)
          {

            unstable.setSymbols(B);
            float newPos1 = xposition + originalLength - unstable.getNumSym(); // change both ring speeds (speed) here based on entropy/energy numbers
            float newPos2 = xposition - originalLength + (A.length()/2 + i); // change both ring speeds (speed) here based on entropy/energy numbers
            unstable.setX(newPos1);
            unstable.stale = true;
            unstable.PE = thePE;

            Ring stableOne = new Ring(A,newPos2,yposition);
            stableOne.stale = true;
            remainingE = remainingE - unstable.PE - stableOne.PE;

          }
        }
      }
    }
  }
if((unstable.PE + stableOne.PE) != 0)
{
    unstable.KE = remainingE * (stableOne.PE/(unstable.PE +
    stableOne.PE));
    stableOne.KE = remainingE * (unstable.PE/(unstable.PE +
    stableOne.PE));
} else
{
    unstable.KE = 0;
    stableOne.KE = 0;
}

unstable.speed = (float)Math.sqrt(unstable.KE*2/unstable.getNumSym());
stableOne.speed = (float)Math.sqrt(stableOne.KE*2/stableOne.getNumSym());

int emptyPos = 0;
if(!freeRings.empty())
{
    emptyPos = ((Integer)freeRings.pop()).intValue();
    rings[emptyPos] = stableOne;
}
}

// end fission stuff
////////////////////////

////////////////////
// Begin Combination Rules
////////////////////
if(rings[i].stale != true)
{
    //code from Discourse section with much modification
    for(int id=i+1; id<maxNumRings; id++)
    {
        if(rings[id] != null && rings[i].stale != true)
        {
            dx = rings[id].getXpos() - rings[i].getXpos();
            dy = rings[id].getYpos() - rings[i].getYpos();
            rr = rings[id].getNumSym()/2 + rings[i].getNumSym()/2;
            diff = dx*dx + dy*dy;

            // If overlap ( from Pythagorean theorem )
            if(diff < (rr*rr))
            {
                // Where the Combination Rules Act
                String result = interaction.combine(rings[i],rings[id]);
                if(result != null)
                {
                    // put all the rule stuff here
                    newSymbols = result;
                    float xposition = rings[i].getXpos() + rings[id].getXpos();
                    float yposition = rings[i].getYpos() + rings[id].getYpos();
                    xposition = xposition/2;
            }
yposition = yposition/2;
int numSymbols = newSymbols.length()/2 + 1;

float remainingE = rings[i].PE + rings[i].KE + rings[id].PE + rings[id].KE;
float thePE = energy.calculatePE(rings[i].getSymArray());

if (thePE < remainingE)
{
    rings[id] = null;
    freeRings.push(new Integer(id));
    rings[i].setSymbols(newSymbols);
    rings[i].setX(xposition);
    rings[i].setY(yposition);
    rings[i].PE = thePE;
    remainingE = remainingE - rings[i].PE;
    rings[i].KE = remainingE;
    rings[i].speed = (float)Math.sqrt(rings[i].KE*2/rings[i].getNumSym());
    rings[i].stale = true;

    // change the ring speed (speed) here based on entropy/energy numbers

    //
    //println(newSymbols);
    //while(!keyPressed){};
    ///
    newSymbols = "";
}

hist = new int[1000]; // divided by size 1 - 100
//PE = new int[100];
spatialSize = new int[myWidth/subdivision][myHeight/subdivision];
spatialNumber = new int[myWidth/subdivision][myHeight/subdivision];
spatialDensity = new float[myWidth/subdivision][myHeight/subdivision];

float totalPE = 0;
float totalKE = 0;
int actualRingNumber = 0;
int[] realRingSizes = new int[maxNumRings];
int maxvalue = 0;
for (int q=0; q<maxNumRings; q++)
{
    if (rings[q] != null)
    {
        rings[q].stale = false; // important thing
        numberSymbols = rings[q].getNumSym();
        if (numberSymbols > maxvalue)
        {
            maxvalue = numberSymbols;
        }
    }
}
realRingSizes[actualRingNumber] = numberSymbols;
umDots = numDots + numberSymbols;
totalPE += rings[q].PE;
if(rings[q].KE < 0)
{
    rings[q].PE += rings[q].KE;
    rings[q].KE = 0;
}
totalKE += rings[q].KE;
hist[numberSymbols]++;
actualRingNumber++;
}

int entropyPE =
(((energy.calcPEntropy(totalPE,actualRingNumber,rings)).toBigInteger()).toString()).length();
int entropyKE =
(((energy.calcKEnergy(totalKE,actualRingNumber,rings)).toBigInteger()).toString()).length();
fill(153);
rect(myWidth,0,width-myWidth,height);
rect(0,myHeight,width,height-myHeight);
fill(30);
textFont(titleFont,50);
text("erg",myWidth+65,40);
textFont(metaBold, 14);
text("Press 'i' to toggle Internal view",30,myHeight+20);
//text(totalPE, myWidth+30, 30); // total PE
//text(totalKE, myWidth+30, 50); // print total KE
text("Total Energy:",myWidth+30,70);
text(totalKE+totalPE, myWidth+30, 90); // print total E

text("Entropy of the System:", myWidth+30, 110);
text(entropyPE+entropyKE, myWidth+30, 130); // print total Entropy

///////////////////////////////////////
// Draw a Histogram
////////////////////////////////////////
//fill(100);
//stroke(100);
/*for (int i=0; i<hist.length; i++) {
    hist[i] = int(hist[i]/maxvalue * actualRingNumber);
}* /
*/
for (int i=0; i<hist.length; i++) {
    rect(i*5,490 - hist[i]*4,4,hist[i]*4);
    if(i % 5 == 0)
        rect(i*5,492,1,3);
}
*/
int scaler = actualRingNumber/maxvalue;
int otherscale = maxvalue/actualRingNumber;
for (int i=0; i<maxvalue; i++){
    fill(hist[i]*10);
    //text(realRingSizes[i],5 + numstepssofar*5,490 - i*4);
    //rectMode(CENTER_DIAMETER);
    //rect(5 + numstepssofar,490 - i*scaler,1,scaler);
```java
//rectMode(CORNER);
}
numstepsssofar++;
}

////////////////////////
// The Ring Class
////////////////////////
class Ring{
    private String symbols;
    private int activeSite;
    private float xpos;
    private float ypos;
    private int numSymbols;
    private int posRot; // shows whether clockwise or counter-clockwise
    private int[] symArray;
    private float rotation = 0.0;
    private float scaling = 1;
    private float angle;
    public boolean stale = false;
    public float speed = 2+random(6);
    public float PE,KE;

    Ring(String s, float xtemp, float ytemp){
        symbols = s; xpos = xtemp; ypos = ytemp;
        numSymbols = symbols.length()/2 + 1;
        symArray = splitInts(symbols);
        int count = 0; int dirCount = 0;
        for(int sym = 0; sym < numSymbols; sym++){
            count = count + getSymbol(sym);
            dirCount = dirCount + directions[getSymbol(sym)];
        }
        activeSite = getSymbol(0);
        if(count % 2 == 0){
            posRot = 1;
        } else {
            posRot = -1;
        }
        angle = dirCount % (2*PI);
        PE = energy.calculatePE(symArray);
        KE = energy.calculateKE(symArray,speed);
    }

    void setSymbols(String s){
        symbols = s;
        numSymbols = symbols.length()/2 + 1;
        symArray = splitInts(symbols);
        int count = 0; int dirCount = 0;
        for(int sym = 0; sym < numSymbols; sym++){
            count = count + getSymbol(sym);
            dirCount = dirCount + directions[getSymbol(sym)];
        }
    }
}
```
if(count % 2 == 0){
    posRot = 1;
} else {
    posRot = -1;
}
angle = dirCount % (2*PI);

int getSymbol(int q){return (splitInts(symbols))[q];}
String getSymbols(){return symbols;}
void setX(float x){xpos = x;}
void setY(float y){ypos = y;}
float getXpos(){return xpos;}
float getYpos(){return ypos;}
int getNumSym(){return numSymbols;}
int getActiveSite(){return activeSite;}
int[] getSymArray(){return symArray;}

void updateRing()=>
    if(stale)
    return;
    String move = new String();
    if(posRot == 1){
        move = String.valueOf(symbols.charAt(0));
        symbols = symbols.substring(1) + move;
        symbols = symbols.trim();
    } else {
        move = String.valueOf(symbols.charAt(symbols.length()-1));
        symbols = move + symbols.substring(0,symbols.length()-1);
        symbols = symbols.trim();
    }
symArray = splitInts(symbols);
xpos = xpos + speed*cos(angle);
ypos = ypos + speed*sin(angle);
if(xpos > myWidth)
    xpos = xpos - myWidth;
if(xpos < 0)
    xpos = xpos + myWidth;
if(ypos > myHeight)
    ypos = ypos - myHeight;
if(ypos < 0)
    ypos = ypos + myHeight;
}

void drawRing()=>
    if(stale)
    return;
    float y = 0;
    float divisor = 2*PI/numSymbols;
    if(seeInnards)
    {
        for(int sym = 0; sym < numSymbols; sym++)
        {
            scale(scaling);
fill(colors[getSymbol(sym)]); // makes appropriate color
ellipse(cos(y+rotation)*numSymbols/2+xpos,sin(y+rotation)*numSymbols/2+ypos,2,2);
//fill(0);
//textFont(metaBold, 12);

//text(getSymbol(sym),cos(y+rotation)*numSymbols/2*3+xpos,sin(y+rotation)*numSymbols/2*3+ypos);
y = y + divisor;
}
}  
else{
fill(colors[getActiveSite()]); // makes appropriate color
ellipse(xpos,ypos,numSymbols,numSymbols);
}

// for even-numbered rings, checks to see if the opposite symbols match
// for odd-numbered rings, looks pseudo-across but can allow one symbol not to match anything
boolean isStable(){
    for(int i = 0;i < numSymbols/2;i++){
        if((getSymbol(i) + getSymbol(i + numSymbols/2) % 2) != 0){
            return false;
        }
    }
    return true;
}
}

//////////////////////////////// // Rule Stuff
////////////////////////////////

class Interact{

    private Vector combineRules;
    public int entropyFitness = 0;
    public int complexityFitness = 0;
    public int[][] results;

    Interact(Vector rules1,int numtrials)
    {
        combineRules = rules1;
        results = new int[numtrials][numtrials];
    }

    // needed for genetic algorithm
    void mutate()
    {
    }

    ////////////////////////////////////// // Controls combination reaction
    //////////////////////////////////////

    String combine(Ring ring1,Ring ring2)
    {
        boolean reverse = true;
        boolean foundRule = false;

        for(int x=0;x<combineRules.size() && foundRule == false;x++)
{  
   if(ring1.getSymbols().startsWith(((combineRule)combineRules.elementAt(x)).getI1()) &&  
      ring2.getSymbols().startsWith(((combineRule)combineRules.elementAt(x)).getI2()))  
   {  
      reverse = false;  
      foundRule = true;  
   }  
   else  
   if(ring1.getSymbols().startsWith(((combineRule)combineRules.elementAt(x)).getI2()) &&  
      ring2.getSymbols().startsWith(((combineRule)combineRules.elementAt(x)).getI1()))  
   {  
      reverse = true;  
      foundRule = true;  
   }  
}

// a rule is chosen

if(!foundRule)
   return null;

// make the arrays for manipulation
String sym1,sym2;
if(reverse)
{  
   sym1 = ring2.getSymbols();  
   sym2 = ring1.getSymbols();  
}
else
{
   sym1 = ring1.getSymbols();  
   sym2 = ring2.getSymbols();  
}

StringBuffer ring2buff = new StringBuffer(sym2);

String B = ring2buff.reverse().toString();
B = B.substring(0,B.length() - 2);

String totalS = sym1.substring(0,1) + " " + sym2.substring(0,1) + " " + B + " " + sym1.substring(2);
   //println(totalS);
return totalS;
}

///////////////////////////////////////////////////////////////////////
// Controls split/fission reaction
///////////////////////////////////////////////////////////////////////

Vector split(Ring ring)
{
   int[] sym = splitInts(ring.getSymbols());
   int act = sym[0];

   Vector toFind = new Vector();
   Vector goodRules = new Vector();

   // checks to see if there are possible rules that can work
   for(int x=0;x<combineRules.size();x++)
   {

int[] input1 = splitInts(((combineRule)combineRules.elementAt(x)).getI1());
int[] input2 = splitInts(((combineRule)combineRules.elementAt(x)).getI2());

// upon interaction, the elements are paired in the resulting ring like so (number of
// ring, position in rule):
// [(1,1)(2,1)]
// [(2,2)(1,2)]

int twoOne = input2[0];
int twoTwo = input2[1];
int oneTwo = input1[1];

input1[1] = twoOne;
input2[0] = twoTwo;
input2[1] = oneTwo;

String sinput1 = join(input1, " ");
String sinput2 = join(input2, " ");

if(ring.getSymbols().startsWith(sinput1))
    toFind.add(sinput2);
else if(ring.getSymbols().startsWith(sinput2))
    toFind.add(sinput1);

if(toFind.size() == 0)
    return null;

int[] breakPoints = new int[20];
int[] itty = new int[2];
int sizePoints = 0;

for(int loc=2; loc<sym.length-1; loc++)
{
    itty[0] = sym[loc];
    itty[1] = sym[loc+1];

    for(int x=0; x<toFind.size(); x++)
    {
        if(((String)toFind.elementAt(x)).equals(join(itty, " ")))
        {
            breakPoints[sizePoints] = loc;
            sizePoints++;
        }
    }
}

if(sizePoints == 0)
    return null;

// have all the positions for the splitting so now one is chosen
int choice = breakPoints[int(random(sizePoints))];
String A = new String();
String B = new String();

A = (ring.getSymbols()).substring(4, choice*2 + 1) + " " + ring.getSymbols().substring(2, 3);
// now reverse it

StringBuffer ring2buff = new StringBuffer(A);
A = ring2buff.reverse().toString();

B = (ring.getSymbols()).substring(0, 1) + " " + ring.getSymbols().substring(choice*2 + 2);

Vector totalS = new Vector();
totalS.add(A);
totalS.add(B);
return totalS;
}
}

////////////////////////////////////////////////
// Simple Rule Classes
////////////////////////////////////////////////

// The groundwork for the rules
class combineRule{
    private String input1, input2;

    combineRule(String i1, String i2){
        input1 = i1; input2 = i2;
    }

    String getI1(){return input1;}
    String getI2(){return input2;}

    boolean compareTo(combineRule c){
        if(input1 == c.input1 && input2 == c.input2)
            return true;
        else
            return false;
    }
}

////////////////////////////////////////////////
// Energetic calculation class
////////////////////////////////////////////////
class energetics{
    int numParticles = 7;

    energetics()
    {
    }

    public float calculatePE(int[] symbols){
        String s = join(symbols, "" unhappiness);  // remove second "" from "" unhappiness"
        float totalPE = 0;

        for(int x = 1; x < 3; x++)
            totalPE += calcOrder(s, x);
return totalPE*symbols.length;
}

public float calculateKE(int[] symbols,float speed){
    return 0.5 * symbols.length * (float)Math.pow(speed,2);
}

float calcOrder(String s,int order){
    int[] occurs = new int[(int)(Math.pow(numParticles,order))];
    float entropy = 0;
    float bins = s.length()-(order - 1);
    for(int x=0;x<(int)bins;x++)
        occurs[((new java.math.BigInteger(s.substring(x,x+order),numParticles)).intValue()))++;
    for(int i=0;i=numRings;)
    {
        if(rings[q] != null)
        {
            occurs[(int)(Math.floor(rings[q].PE))++;
        }
    }
    java.math.BigDecimal expo;
    java.math.BigDecimal facto;
    for(int i=0;i= 0)
    {
        expo = new java.math.BigDecimal(base.pow(occurs[i]));
        facto = new java.math.BigDecimal(Factorial.fac(occurs[i]));
        entropy = entropy.multiply(expo.divide(facto,4,java.math.BigDecimal.ROUND_HALF_UP));
    }
    return entropy; // must still take logarithm of this (is done by counting number of characters)
}

public java.math.BigDecimal calcKEntropy(float totalE,int numRings,Ring[] rings) {
    int[] occurs = new int[(int)totalE];
}
java.math.BigDecimal entropy = new java.math.BigDecimal(java.math.BigInteger.ONE);

java.math.BigInteger base = java.math.BigInteger.valueOf((long)occurs.length);
float numParticles = numRings;

for(int q=0; q<rings.length; q++)
{
    if(rings[q] != null)
    {
        occurs[(int)Math.floor(rings[q].KE)]++;
    }
}

java.math.BigDecimal expo;
java.math.BigDecimal facto;
for(int i=0; i<occurs.length; i++)
{
    if(occurs[i] != 0)
    {
        expo = new java.math.BigDecimal(base.pow(occurs[i]));
        facto = new java.math.BigDecimal(Factorial.fac(occurs[i]));
        entropy = entropy.multiply(expo.divide(facto, 4, java.math.BigDecimal.ROUND_HALF_UP));
    }
}
return entropy; // must still take logarithm of this (is done by counting number of characters)

} // based on http://leepoint.net/notes-java/25data/10numbers/60factorial.html

static class Factorial
{
    static java.math.BigInteger fac(int n)
    {
        java.math.BigInteger result = java.math.BigInteger.ONE;
        for (int i=2; i<=n; i++)
        {
            result = result.multiply(java.math.BigInteger.valueOf((long)i));
        }
        return result;
    }
}