The Universal Rewrite System as a Variable Regular Expression

Preprint · May 2024

CITATIONS
0

READS
137

2 authors:

Sydney Rowlands
7 PUBLICATIONS **19** CITATIONS
SEE PROFILE

Peter Rowlands
University of Liverpool
98 PUBLICATIONS **571** CITATIONS
SEE PROFILE

The Universal Rewrite System as a Variable Regular Expression

Sydney Rowlands
Peter Rowlands

September 30 2024

jrist29@gmail.com
p.rowlands@liverpool.ac.uk
Physics Department University of Liverpool Oliver Lodge Laboratory Oxford St, Liverpool. L69 7ZE, UK

Keywords: Universal Rewrite System, Variable Regular Expression, Variable Finite Automata, Dirac operator matrix, binary encoding, quantum computing

Abstract

We have talked about the URS being the language of physics where each word or string has a semantic meaning of a zero totality [1],[2],[3]. We have also brought up the fact that the URS as a language has an infinite number of symbols or letters in its alphabet[4],[5],[6]. How can the universe compute a language with an infinitely sized alphabet with an infinite number of words that represent increasing larger cardinalities of zero totality? Can this be given a finite representation, or will the language of nature be a closed book to us forever because we have finite memories that struggle to remember large finite-sized collections? Surprisingly, it turns out that it is possible to accomplish a major simplification of an infinitely-sized problem like this and turn it into a two-state machine that can remember an infinite number of symbols. The counterintuitive fact is that this unbounded problem does not require an unbounded solution. Despite the fact that the Universal Rewrite System is a language with an infinite number of symbols, the solution to finitely representing this system does not require a matching number of infinite states. The resolution is to use a Variable Finite Automaton. We divide all the infinite symbols of the Universal Rewrite System into three variables of two types: two bound variables and one free variable. This simple assignment converts an infinitely sized problem into a finite sized solution, and it works because all the infinite symbols get binned into the two categories of variables.

1 The Universal Rewrite System

Nature is in a state of perpetual change. Normally, these changes are described by separately defined processes, but is there an underlying universal process which is common to all the changes? If there is, we would expect it to be simple in principle but complex in its effects. Change can only be defined against something which stays the same. Creation is accompanied by conservation. So, what is it that changes and what stays the same? And can we find a simple process at the most fundamental level that will always apply to every situation in nature? We believe that one candidate stands out. This is the Universal Rewrite System (URS), devised by Rowlands and Diaz in 2002 [7]. This is based on the idea that Nature has no definable characteristics and has a totality of zero after every process. Intimations of this are present in Newton's third law of motion, where every action by an object is accompanied by an equal and opposite reaction from the rest of the universe. Inside the object or system, the equal and opposite forces cancel to zero. In the URS, any symbol, such as R, must have a complementary R', which effectively cancels it.

The universal rewrite system, first proposed in 2002, has been discussed and elaborated on in several subsequent publications [4],[6],[8],[9]. It is founded on an answer to the age-old question 'Why is there something rather than nothing?', which asserts that the question is misdirected because there really is nothing rather than something, in the sense that the 'universe' or 'reality' always has a total value of zero. In other words, Nature has no defining characteristic because any such characteristic could not be explained in a

fundamental way. The only state which can never be subjected to more fundamental explanation is that of zero.

In the rewrite system, the alphabet, or defining symbol set, must be constructed in such a way that it creates a totality zero. We then imagine a succession of zero totality alphabets of increasing cardinality or size each of which incorporates the previous one defined. The method of incorporation is strict and minimalistic. Just one new symbol is created for each new alphabet. If we consider ourselves as observers from within the entire system of reality, we may start with a zero totality of the form (R, R'), where R and R' complement each other, ensuring that the total remains zero. The next alphabet must include (R, R') as a sub-alphabet. The minimum zero-totality extension would then be (R, R', A_0, A_0'), where A_0 = the complex imaginary i_0. To find out how the terms relate to each other, we could concatenate the entire alphabet with individual terms or with a sub-alphabet. Each concatenation of any term with the alphabet will produce the entire alphabet as a unique automorphism, ensuring that each term is unique.

So, the concatenation, which means placing terms together, without algebraic significance, will produce nothing new if it is between any alphabet and any symbol or sub-alphabet of that alphabet, but it will ensure that each symbol is unique. This is a process we can describe as *conservation*. If, however, we concatenate any alphabet with itself, then this must produce a new extended alphabet incorporating the old one, because no alphabet can be the final one in the zero-totality sequence. This process we can describe as *creation*. We can symbolise the respective processes using $\rightarrow$ and $\Rightarrow$. Creation can only occur if the new alphabet is constructed to obey the conserve rule. So, creation and conservation are not separate processes but aspects of one.

Applying conservation to (R, R'), we find

$$(R)(R, R') \rightarrow (R, R')$$
$$(R')(R, R') \rightarrow (R', R)$$

so that

$$(R)(R) \rightarrow (R)$$
$$(R')(R) \rightarrow (R')$$
$$(R)(R') \rightarrow (R')$$
$$(R')(R') \rightarrow (R)$$

Although we could, in principle, switch R and R', this would yield no new information, simply resulting in an exchange of symbols.

Now, when we concatenate the whole alphabet with itself, we have to ensure that (R, R') does not remain unique, and a new creation ensures that. (R, R') (R, R') $\Rightarrow$ (R, R', A_0, A_0') has the same effect as taking the concatenation of (R, R') with (R, A_0), where the single new symbol A is created. In addition, the creation can only be true if the conservation process holds as well. That is, each component concatenates with the alphabet to form a unique automorphism of it.

$$(R)(R, R', A_0, A_0') \rightarrow (R, R', A_0, A_0')$$
$$(R')(R, R', A_0, A_0') \rightarrow (R', R, A_0', A_0)$$
$$(A_0)(R, R', A_0, A_0') \rightarrow (A_0, A_0', R', R)$$
$$(A_0')(R, R', A_0, A_0') \rightarrow (A_0', A_0, R, R')$$

We note that, to ensure A is distinct from R, we need

$$(A_0)(A_0) \rightarrow (R')$$
$$(A_0)(A_0') \rightarrow (R)$$
$$(A_0')(A_0) \rightarrow (R)$$
$$(A_0')(A_0') \rightarrow (R')$$

and this will be true of all new symbols introduced in further creations. At the next stage, where we introduce the new symbol B, the process of ensuring that the new alphabet only produces itself when concatenated with its sub-alphabets requires that we now introduce concatenated terms, such as AB, AB' into the alphabet, which now becomes equivalent to concatenation of the previous alphabet with (R, B).

$$(\mathrm{R},\ \mathrm{R'},\ \mathrm{A}_0,\ \mathrm{A}_0')(\mathrm{R},\ \mathrm{R'},\ \mathrm{A}_0,\ \mathrm{A}_0') \Rightarrow (\mathrm{R},\ \mathrm{R'},\ \mathrm{A},\ \mathrm{A},\ \mathrm{B},\ \mathrm{B'},\ \mathrm{AB},\ \mathrm{AB'})$$
$$\equiv$$
$$(\mathrm{R},\ \mathrm{B})(\mathrm{R},\ \mathrm{R'},\ \mathrm{A}_0,\ \mathrm{A}_0') \Rightarrow (\mathrm{R},\ \mathrm{R'},\ \mathrm{A},\ \mathrm{A'},\ \mathrm{B},\ \mathrm{B'},\ \mathrm{AB},\ \mathrm{AB'})$$

As in all previous cases, if we successively perform the $\rightarrow$ operation with (R'), (A), (A'), (B), (B'), (AB), (AB'), or any combination of these, subject to the necessary condition that their combination is always less than the full alphabet, then the result will always be a unique automorphism of the alphabet.

$$(\mathrm{R})(\mathrm{R},\ \mathrm{R'},\ \mathrm{A},\ \mathrm{A'},\ \mathrm{B},\ \mathrm{B'},\ \mathrm{AB},\ \mathrm{AB'}) \rightarrow (\mathrm{R},\ \mathrm{R'},\ \mathrm{A},\ \mathrm{A'},\ \mathrm{B},\ \mathrm{B'},\ \mathrm{AB},\ \mathrm{AB'})$$
$$(\mathrm{R'})(\mathrm{R},\ \mathrm{R'},\ \mathrm{A},\ \mathrm{A'},\ \mathrm{B},\ \mathrm{B'},\ \mathrm{AB},\ \mathrm{AB'}) \rightarrow (\mathrm{R'},\ \mathrm{R},\ \mathrm{A'},\ \mathrm{A},\ \mathrm{B'},\ \mathrm{B},\ \mathrm{AB'},\ \mathrm{AB})$$
$$\ldots$$

Since none of the terms are changed, all of these automorphisms are equivalent to the original alphabet.

However, the results of concatenations of AB and AB' with themselves still have to be established. The only options available for (AB) (AB) and (AB)' (AB)', if the concatenations of AB and AB' with the alphabet must produce automorphisms of the alphabet, are either R or R'. If either becomes R (the commutative option), then AB and AB' become effectively indistinguishable from R, and nothing new has been produced. So, the only option for both is R' (the anti-commutative option). Then the concatenations (AB) (AB)' and (AB)' (AB) will become R'.

This is very significant, for the anti-commutative option will not be repeatable when the alphabet is extended to incorporate new terms, such as (C), (D), etc., whereas the commutative option could be repeated indefinitely. The anti-commutative option effectively produces a closed cycle with components (A, B, AB) and their conjugates, which excludes any further C, D -type term of anti-commuting with them. All other such terms will commute. Further terms will then generate identical but independent cycles, such as (C, D, CD) and (E, F, EF), to infinity. As we will see, the URS anti-commutativity effectively causes both 3-dimensionality and discreteness.

There exists an alternative symbolism to using A, B, C, D, E, F, ... for the infinite square roots of -1, which in this work are interpreted as an infinite succession of quaternion sets. Since, by definition of the quaternion group, the square roots of -1 exist as two types, i and j. This other notation system would capture faithfully the infinite square roots of -1, but at the same time, avoid any need for numerical subscripting and eliminate the problem of exhausting any finite alphabetic symbol set (e. g. A $-$ Z) representing an infinite sequence of square roots of -1. This notation divides the infinite sequence of square roots of -1 into to two infinite sequences, one for i and one for j without any need for numerical subscripting. It is as follows:

$$\mathrm{A} \leftarrow i_1$$
$$\mathrm{B} \leftarrow j_1$$
$$\mathrm{Aa} \leftarrow i_2$$
$$\mathrm{Bb} \leftarrow j_2$$
$$\mathrm{Aaa} \leftarrow i_3$$
$$\mathrm{Bbb} \leftarrow j_3$$
$$\ldots \leftarrow \ldots$$

Previously, we created a Mathematica program [6] to generate the Universal Rewrite pattern using a single line of active code. An infinite tensor product was able to output the universe in a single step (lines 3 to 4 below).

$$\mathrm{n} = \mathrm{Input[n]}$$
$$\mathrm{m} = \mathrm{n} - 1$$
$$\mathrm{If\ [n == 0,\ T = 0,\ T = \{}X_0, -X_0\mathrm{\}]}$$
$$\mathrm{Do\ [T = TensorProduct[T,\ \{}X_0, X_i\mathrm{\}],\ \{i,\ 1,\ m,\ 1\}]}$$
$$\mathrm{Evaluate[T]}$$

We also ran the program for n = 1 to 6 and were able to generate the first 6 orders, where any power p of X_0 is equal to the identity: X_0^p = identity (1) for any integer p, X_1 = A$_0$, $-X_1$ = A$_0$', X_2 = A, $-X_2$ = A', X_3 = B, etc.

2 Application of the Universal Rewrite System (URS) to Mathematics and Physics

One of the things that the rewrite structure achieves is a perfectly repetitive sequence of independent systems. This is exactly the condition needed to define integers or natural numbers. If the rewrite system is comprehensive and complete, then the repetition of quaternion-like systems, such as (R, A, B, AB), will be the first time that integers appear in the system. The 'quaternions' also introduce the concepts of dimensionality and discreteness. So, these mathematical concepts are not independent of this structure but a natural outcome of it. If we then apply the concept of 'number' to the system, we will then find that other types of number, e.g. real, imaginary, etc, will also emerge from it, as well as various mathematical operations between numbers. Perhaps for the first time, mathematics will be an integral part of the emerging fundamental system, and not something merely 'applied' to it[1]. It is an important aspect of a fundamental theory or methodology to ensure that it is not dependent on systems which are external to it and imported merely for convenience.

One of the first applications is to Clifford algebra. Here, we see that the sequence of terms R, R', A_0, A_0', A, A', B, B', AB, AB', ... is entirely parallel in structure to the sequence 1, -1, i_0, $-i_0$, i_1, $-i_1$, j_1, $-j_1$, $i_1 j_1 = k_1$, $-i_1 j_1 = -k_1$,... where i_0, $-i_0$ is the complex imaginary unit and i_1, j_1, k_1 are quaternion imaginary units. At this point, we reach the anti-commutation limit (i.e. 3 dimensions). Further symbols would have to lead to a completely new and independent quaternion system. An incomplete quaternion system would act as a complexification.

So, the first three alphabets would be

(1, -1)	real numbers
(1, -1, i_0, $-i_0$)	complex numbers
(1,-1, i_1, $-i_1$, j_1, $-j_1$, $i_1 j_1$, $-i_1 j_1$)	quaternions

The next alphabet would be complex quaternions (or multivariate vectors). Then the fifth would be double quaternions, and the sixth complex double quaternions, leading to an infinite series of independent quaternion systems, with incomplete ones being represented by complex numbers. It is remarkable that the rewrite system should reproduce, essentially from the idea of totality zero, a system parallel to one of the most significant mathematical structures used in physics. These systems are based on numbers which are not assumed in the rewrite structure. However, as we have seen, the rewrite system can also be shown to produce number systems from first principles, and this can be done using the same process.

It is significant that discreteness occurs in the rewrite structure at order 8 (which we can repackage as order 2, using quaternions as a 3-D discrete structure), and here we find the origin of integers. The fourth column in the table below replaces (R, A, B, AB) with 1 and ($-$R,$-$A, $-$B, $-$AB) with -1 to show that the Universal Rewrite System can be rewritten to introduce numbers, effectively recasting itself as a discrete sequence.

discrete 1 $\leftarrow$ (R, A, B, AB)	(order 8)
discrete -1 $\leftarrow$ ($-$R,$-$A, $-$B, $-$AB)	(order 8)
$\pm\, i_0 \leftarrow \pm$ A_0	
$\pm\, i_1 \leftarrow \pm$ C	
$\pm\, j_1 \leftarrow \pm$ D	

order 2	±R			Continuous
order 4	± R, ± A_0			Continuous
order 8	± R, ± A, ± B, ± AB	order 2	± 1	Discrete 1 × 3-D
order 16	± R, ± A, ± B, ± AB, ± A_0, ± A_0A, ± A_0B, ± A_0AB	order 4	± 1, ± i_0	Discrete 1 $\frac{1}{2}$ × 3-D
order 32	± R, ± A, ± B, ± AB, ± C, ± CA, ± CB, ± CAB, ± D, ± DA, ± DB, ± DAB, ± CD, ± CDA, ± CDB, ± CDAB	order 8	± 1, ± i_1, ± j_1, ± i_1j_1	Discrete 2 × 3-D
order 64	±R, ±A, ±B, ±AB, ±C, ±CA, ±CB, ±CAB, ±D, ±DA, ±DB, ±DAB, ±CD, ±CDA, ±CDB, ±CDAB, ±A_0, ±A_0A, ±A_0B, ±A_0AB, ±A_0C, ±A_0CA, ±A_0CB, ±A_0CAB, ±A_0D, ±A_0DA, ±A_0DB, ±A_0DAB, ±A_0CD, ±A_0CDA, ±A_0CDB, ±A_0CDAB	order 16	±1, ±i_1, ±j_1, ±i_1j_1, ±i_0, ±i_0i_1, ±i_0j_1, ±$i_0i_1j_1$	Discrete 2 $\frac{1}{2}$× 3-D
...	...	...	...	...

With number now established as a possible outcome, we can consider how physical ideas which depend on number evolve. The first four alphabets can be considered as expressing successive number systems which coincide with the successive properties of the physical quantities mass, time, charge and space, which can be summarised in the table [1],[2],[3]:

mass	real	conserved	continuous
time	imaginary	nonconserved	continuous
charge	imaginary	conserved	discrete
space	real	nonconserved	discrete

The significance of the symmetry shown here, which can be seen to be that of a D_2 or Klein-4 group, is that a particular closure has been achieved at this level, since all the properties cancel overall and produce a zero totality. In addition, all the physical properties are direct consequences of the algebraic ones. Some of the distinctions between property and anti-property (or its opposite) are subtle rather than obvious, but all present exact oppositions when fully examined. Nonconservation appears to originate in an extra complex i_0 in the quantity's description. It is a very definite property with major physical consequence, and not just the absence of conservation. The nonconserved aspects of space and time lead to their description by differentials and are the origin of such things as gauge invariance and quantum uncertainty. The continuous / discrete distinction has several forms and could also be described as commutative / anticommutative or nondimensional / dimensional.

There are several very significant consequences of the parameter group. The tensor product of the first 4 algebras,

$$\text{real numbers} \times \text{complex numbers} \times \text{quaternions} \times \text{complex quaternions}$$

is equivalent to the algebra of the sixth alphabet: complex double quaternions. This is the algebra of the Dirac equation, the equation referring to the most fundamental and elementary state in physics: the fermion. This has been explored in many earlier papers and it is believed to be the source of all physical information. The 64 units of the Dirac algebra include 12 sets of 5 units, each of which acts as a generator of the entire algebra (either immediately or when multiplied by complex i), and each of which, when multiplied by appropriate *scalar* coefficients, equivalent to E, p_x, p_y, p_z and m, is a complete description of any given Dirac state [1], [2], [3] ($\hbar = 1$ and c = 1). In this form, the total squares to 0, leading to another level of zero totality. This *nilpotent* structure, when squared, provides the most general description of the conservation of energy, which is the basis of the definition of all physical systems of any type and size.

The 64-part algebra is also one on which all known physical particles can be structured (a representation which can be done in two separate ways) and forms the basis of the 64-component genetic code [10],[11] with

a one-to-one correspondence. The Klein-4 structure for mass, time, charge and space is additionally the basis for the C, P and T symmetries [1], [2], [3], and for the parallel systems to be found in those cellular automata [12] which have long-term stability.

The *nilpotent* structure, with its characteristically broken 5-fold symmetry, relates to geometrical structures with 5-fold symmetries (including Platonic solids, Penrose tiling and quasicrystals), and parallels to these have been outlined in earlier work [1],[13], [14]. The connections with the Laws of Form and Category Theory will be discussed in later sections of this paper. Numerous parallels with larger-scale structures have been outlined in earlier publications [1],[15],[16],[17].

3 Transcribing the Universal Rewrite System into a Regular Expression

The Specification for the Universal Clifford Algebra of order 2^n

- Function(0) = 2^n (This function offers a possible reason why there is no beginning and no end to the universe.)
- Function$^{-1}(2^n) = 0$

That is, if we input 0 we obtain 2^n; if we input 2^n we obtain 0. The implementation is of the form:

- Function(0) = $(x_1 + x_2)(x_1 + \mathrm{y})^*$
- Function(0) = $(x_1 + \mathrm{y})^*(x_1 + x_2)$
- Function$^{-1}((x_1 + x_2)(x_1 + \mathrm{y})^*) = 0$
- Function$^{-1}((x_1 + \mathrm{y})^*(x_1 + x_2)) = 0$

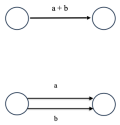
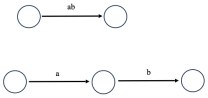
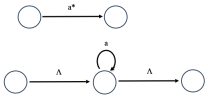
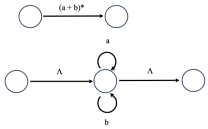
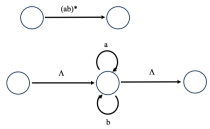
Here, we understand x terms to be fixed and the single y term to vary. We can encode the internal representation of the URS using data structures provided by Clifford algebra product rules. To translate the URS and provide the formal language, we will use new symbol assignments, where R $\rightarrow x_1$, R' $\rightarrow x_2$, and A_0, A, B, C, ... $\rightarrow y$. Here, we have bound variables X = $\{x_1, x_2: x_1 = 1, x_2 = -1\}$ and the single free variable $\{y\} = \{y: y = \pm\sqrt{x_2}_{m(n)}$ where $m(n) = \frac{1}{4}(2n - 1 + (-1)^{(n-1)}) - \frac{1}{2}$, $m(n) \geq 1\}$, the index labelling number for i and j. To introduce new symbols into the alphabet without changing the net zero sum it might seem that taking the square root of either x_1 or x_2 would be a valid choice but simple mathematical reasoning will show that only one square root gives the novelty needed to extend the alphabet and in fact the source of the new symbols must be square roots of negative one, not square roots of positive one: $y = \pm\sqrt{x_2}_{m(n)}$ extends the alphabet with a new symbol, not $y = \pm\sqrt{x_1}_{m(n)}$ since $y^2 = x_1$ implies $y = x_1$ and $y = x_2$, meaning that the two square roots of positive one are real numbers whereas the two square roots of negative one are not real numbers, they are new types of numbers. We are forced to conclude that to create a new symbol beyond x_1 and x_2, y must be two square roots of negative one or $y = \pm\sqrt{x_2}_{m(n)}$.

2^n	$i_{m(n)}$ and $j_{m(n)}$	$i_0^{((n+1) \mod 2)}$	
n	$m(n) = \frac{1}{4}(2n - 1 + (-1)^{(n-1)}) - \frac{1}{2}$	$((n+1) \mod 2)$	
1	–	0	$\{1, -1\}$
2	–	1	$\{1, -1\} \otimes \{1, i_0\}$
3	1	0	$\{1, -1\} \otimes \{1, i_1\} \otimes \{1, j_1\}$
4	1	1	$\{1, -1\} \otimes \{1, i_1\} \otimes \{1, j_1\} \otimes \{1, i_0\}$
5	2	0	$\{1, -1\} \otimes \{1, i_1\} \otimes \{1, j_1\} \otimes \{1, i_2\} \otimes \{1, j_2\}$
6	2	1	$\{1, -1\} \otimes \{1, i_1\} \otimes \{1, j_1\} \otimes \{1, i_2\} \otimes \{1, j_2\} \otimes \{1, i_0\}$

4 Regular Languages

Assuming the properties of regular languages, there exist two forms of regular expression that every other type of regular expression seems reducible to. Physical evidence exists to support this claim [4]. If there exists a foundational origin for the physical world then it seems that there must exist an original expression to start

all other expressions. Regular expressions in formal language theory can be expressed with the operations listed in the following table[18]. The A and B symbols in the table are not necessarily those of the URS. A + B is A OR B. A.B is A AND B. A* is an infinite AND (product) of A's.

empty set	$\emptyset$	{ }	
parallel sum	a + b	a + b	
sequential product = concatenation	ab	ab	
star closure	a*	a^n, n $\geq$ 0	
star sum	(a + b)*	$(a + b)^n$, n $\geq$ 0	
star product	(ab)*	$(ab)^n$, n $\geq$ 0	
All words over $\Sigma = \{x_1, x_2, y\}$	$(x_1 + x_2 + y)^*$	$(x_1 + x_2 + y)^n$, n $\geq$ 0	

5 URS AS A VARIABLE REGULAR LANGUAGE

If the universal rewrite system's representation in formal language theory is a regular language over an infinite alphabet, how can a finite state machine recognise it? Theoretically a regular language with an infinite alphabet would require an infinite state set to remember every symbol read from the infinite alphabet. This problem is solved by a machine that processes variables instead of symbols. A Variable Finite Automaton uses variable assignments that can be fixed and varying simultaneously.

The Variable Finite Automaton is the optimal solution to the finite state representation problem of regular languages over infinite alphabets. A Variable Finite Automaton (VFA) is defined as an ordered pair (Σ, A) where Σ is an infinite alphabet and A is a nondeterministic finite automaton[20]. Let V = {v_1, ..., v_n: n ϵ N} be the finite alphabet of the nondeterministic finite automaton A and let Σ = {w_p: p $\geq$ 1, p ϵ N} be the infinite alphabet of the Variable Finite Automaton. Let Γ_A = X $\cup$ {y}, where X = {x_1, x_2}, is the finite variable alphabet processed by the finite variable automaton (Σ , A). The variables in X cannot change what symbols they are assigned to from V but the variable y can change what symbol it is assigned to from V. In total there are three different alphabets to consider: the finite alphabet V of the nondeterministic finite automaton A, the infinite alphabet of the variable finite automaton (Σ, A) and the finite variable alphabet Γ_A of the variable finite automaton (Σ, A). The bound variable assignments in X allow the same symbols to occur repeatedly and the changing assignment of {y} allows new symbols to occur. The variables provide for an infinite alphabet representation of regular languages without changing the number of states needed to process the language.

The universal rewrite system (URS) written as a variable regular language is expressed by two non-equivalent distinct types of variable regular expressions over the finite variable alphabet X = {x_1, x_2} $\cup$ {y}:

$(x_1 + x_2)(x_1 + \mathrm{y})^*$ and $(x_1 + \mathrm{y})^*(x_1 + x_2)$. Therefore, the finite variable alphabet for the URS is $\Gamma_A = \mathrm{X} \cup \{\mathrm{y}\}$ where $\mathrm{X} = \{x_1, x_2: x_1 = 1$ and $x_2 = -1\}$ is the set of bound variables and $\{\mathrm{y}\} = \{\mathrm{y}: \mathrm{y} = \mathrm{i}_{m(n)}$ or $\mathrm{y} = \mathrm{j}_{m(n)}$, $\mathrm{m(n)} \geq 1$, $\mathrm{m(n)} \epsilon\ \mathrm{N}\}$ is the set of the single free variable y.

The semantics of the universal rewrite system can be expressed by two different variable regular expressions. The first expression is $\Gamma^*_{A_1} = (x_1 + x_2)(x_1 + \mathrm{y})^*$, where $x_1 = 1$ and $x_2 = -1$, $x_1 \neq x_2$ and $\mathrm{y} = \mathrm{i}_{m(n)}$ or $\mathrm{y} = \mathrm{j}_{m(n)}$, $\mathrm{m(n)} \geq 1$, $\mathrm{m(n)} \in \mathrm{N}$ is the same as $\Gamma^*_{A_1} = \{\mathrm{v}_1 ... \mathrm{v}_k ... \mathrm{v}_n$: $\mathrm{v}_1 = x_1$ or $\mathrm{v}_1 = x_2$, $\mathrm{v}_k = \mathrm{y} = \mathrm{i}_{m(n)}$ or $\mathrm{v}_k = \mathrm{y} = \mathrm{i}_{m(n)}$, $\mathrm{m(n)} \geq 1$, $\mathrm{k} \neq 1$, where $2 \leq \mathrm{k} \leq \mathrm{n}$ and $\mathrm{m(n)}$, $\mathrm{n} \in \mathrm{N}\}$. The second expression is $\Gamma^*_{A_2} = (x_1 + \mathrm{y})^*(x_1 + x_2)$, where $x_1 = 1$ and $x_2 = -1$, and $\mathrm{y} = \mathrm{i}_{m(n)}$ or $\mathrm{y} = \mathrm{j}_{m(n)}$, $\mathrm{m(n)} \geq 1$, $\mathrm{m(n)} \in \mathrm{N}$ is the same as $\Gamma^*_{A_2} = \{\mathrm{v}_n ... \mathrm{v}_k ... \mathrm{v}_1$: $\mathrm{v}_1 = x_1$ or $\mathrm{v}_1 = x_2$, $\mathrm{v}_k = \mathrm{y} = \mathrm{i}_{m(n)}$ or $\mathrm{v}_k = \mathrm{y} = \mathrm{i}_{m(n)}$, $\mathrm{m(n)} \geq 1$, $\mathrm{k} \neq 1$, where $2 \leq \mathrm{k} \leq \mathrm{n}$ and $\mathrm{m(n)}$, $\mathrm{n} \in \mathrm{N}\}$. These expressions describe the same language only when the variables of $\Gamma^*_{A_1}$ and $\Gamma^*_{A_2}$ are assigned the symbols of the universal rewrite system. Syntactically these two expressions describe different languages because concatenation is not commutative. However, they are semantically the same language when they are interpreted as the universal rewrite system: syntactically $(x_1 + x_2)(x_1 + \mathrm{y})^* \neq (x_1 + \mathrm{y})^*(x_1 + x_2)$ but semantically over the URS $(x_1 + x_2)(x_1 + \mathrm{y})^* = (x_1 + \mathrm{y})^*(x_1 + x_2)$.

This determines the language of the infinite alphabet Σ to be $\Sigma^* = \{\mathrm{w}_1 .. \mathrm{w}_k ... \mathrm{w}_n : \mathrm{w}_1 = \mathrm{v}_1 = x_1$ or $\mathrm{w}_1 = \mathrm{v}_1 = x_2$ and $\mathrm{w}_k = \mathrm{v}_k = \mathrm{y} = \mathrm{i}_{m(n)}$ or $\mathrm{w}_k = \mathrm{v}_k = \mathrm{y} = \mathrm{j}_{m(n)}$, $\mathrm{m(n)} \geq 1$, $\mathrm{k} \neq 1$, $2 \leq \mathrm{k} \leq \mathrm{n}$, $\mathrm{m(n)}$, $\mathrm{n} \in$ Natural numbers, $\mathrm{N}\}$ or $\Sigma^* = \{\mathrm{w}_n .. \mathrm{w}_k ... \mathrm{w}_1 : \mathrm{w}_1 = \mathrm{v}_1 = x_1$ or $\mathrm{w}_1 = \mathrm{v}_1 = x_2$ and $\mathrm{w}_k = \mathrm{v}_k = \mathrm{y} = \mathrm{i}_{m(n)}$ or $\mathrm{w}_k = \mathrm{v}_k = \mathrm{y} = \mathrm{j}_{m(n)}$, $\mathrm{m(n)} \geq 1$, $\mathrm{k} \neq 1$, $2 \leq \mathrm{k} \leq \mathrm{n}$, $\mathrm{m(n)}$ and $\mathrm{n} \in \mathrm{N}\}$.

The semantics of $x_1 = 1$, $x_2 = -1$, $\mathrm{y} = \mathrm{i}_{m(n)}$ or $\mathrm{y} = \mathrm{j}_{m(n)}$ put in tabular form looks like:

URS	1	-1	$\mathrm{i}_{m(n)}$	$\mathrm{j}_{m(n)}$
Γ_A	x_1	x_2	y	y
V	v_1	v_1	v_k	v_k
Σ	w_1	w_1	w_k	w_k

5.1 Variable Regular Grammar

For $(x_1 + x_2)(x_1 + \mathrm{y})^*$ the grammar production rules become

$$\mathrm{q}_0 \rightarrow (x_1 + x_2)\mathrm{q}_1$$
$$\mathrm{q}_1 \rightarrow (x_1 + \mathrm{y})\mathrm{q}_1 + \epsilon$$

where ϵ = the empty string.

For $(x_1 + \mathrm{y})^*(x_1 + x_2)$ the grammar production rules become

$$\mathrm{q}_0 \rightarrow (x_1 + \mathrm{y})\mathrm{q}_0 + (x_1 + x_2)\mathrm{q}_1$$
$$\mathrm{q}_1 \rightarrow \epsilon$$

where ϵ = the empty string.

To reproduce the URS, the free variable y takes assigned values from the Clifford numbers i_0, i_1, j_1, i_2, j_2, ... The most fundamental regular expression to generate the universe uses only the three basic regular language operations. There are a few classical Kleene algebra axioms that are relevant to the interpretation of this special regular expression. First some notation:

Let $\emptyset$ = the empty set, λ = the empty string whose length $|\lambda| = 0$, and a, b, c are arbitrary symbols from an arbitrary alphabet Σ. Based on the work of Stephen Kleene [18], the class or set of sets of regular expressions over an alphabet is defined recursively as follows:

1. λ and $\emptyset$ are regular expressions over Σ.
2. Every letter σ in the alphabet Σ is a regular expression over Σ.

3. If R_1 and R_2 are regular expressions over Σ, then $(R_1 + R_2)$, (R_1R_2), and R_1^* are regular expressions over Σ.

Why do we care about regular expressions and what does this have to do with the Universal Rewrite System? The claim we are making is that the URS is the pattern that everything reduces to. If the URS is a specific pattern, what is the simplest way to express and automate this pattern − as a regular expression and subsequently automated as a finite automaton? First, however, let us explain what a regular expression is. In simple terms, a regular expression is a word pattern, in computer science an alphabetic, numeric, or alphanumeric string. These are the properties of the operations of parallel sum, product or concatenation, and star closure for the letters a, b, c, etc.. The last rule, no. 14, is a logical extension, not usually stated, but necessary for the regular expression table which follows.

1. $a + b = b + a$
2. $ab \neq ba$
3. $(a + b) + c = a + (b + c)$
4. $a(b + c) = ab + ac$
5. $(b + c)a = ba + ca$
6. $a(b + c) \neq (b + c)a$
7. $(a + b)(c + d) = ac + ad + bc + bd$
8. $(c + d)(a + b) = ca + cb + da + db$
9. $(a + b)(c + d) \neq (c + d)(a + b)$
8. $a^* = \lambda + a + a^2 + ... + a^k + a^{k+1} + a^*$ $(k \geq 0)$

with the special case: $a^* = \lambda + aa^*$

10. $a^* = a^*a^* = (a^*)^* = (\lambda + a)^*$
11. $\emptyset^* = \lambda^* = \lambda$
12. $(a + b)^* = (a^* + b^*)^* = (a^*b^*)^* = (a^*b)^*a^* = a^*(ba^*)^*$
13. Generally $(a + b)^* \neq a^* + b^*$
14. $a^0 = \lambda$ (empty string)

The URS semantic reinterpretation of the symbols in the regular expression will mean that axiom 2 is changed in the reinterpretation to

$$ab = ba$$

thereby equating axiom 4 and 5

$$a(b + c) = (b + c)a$$

which allows the equation of the two sides of

$$(x_1 + y)^*(x_1 + x_2) = (x_1 + x_2)(x_1 + y)^*$$

5.2 Regular Expressions

In the table shown below, we expand out the Create for the URS for both forms to show that they produce the same language semantically:

$$\text{Create: } (x_1 + y)^*(x_1 + x_2) = (x_1 + y)^n(x_1 + x_2),\ n \geq 0$$
$$\text{Create: } (x_1 + x_2)(x_1 + y)^* = (x_1 + x_2)(x_1 + y)^n,\ n \geq 0$$

The following table demonstrates the semantic equality of the two syntactically unequal Create expressions when the symbols of the two Create expressions are assigned the algebraic symbols of the Universal Rewrite System.

Create	$(x_1 + \mathrm{y})^0(x_1 + x_2)$ $= \lambda(x_1 + x_2)$ $= x_1 + x_2$	$(x_1 + x_2)(x_1 + \mathrm{y})^0$ $= (x_1 + x_2)\lambda$ $= x_1 + x_2$
Conserve	$x_1(x_1 + x_2)$ $= (x_1x_1 + x_1x_2)$ $= x_1 + x_2$ $x_2(x_1 + x_2)$ $= (x_2x_1 + x_2x_2)$ $= x_1 + x_2$	$x_1(x_1 + x_2)$ $= (x_1x_1 + x_1x_2)$ $= x_1 + x_2$ $x_2(x_1 + x_2)$ $= (x_2x_1 + x_2x_2)$ $= x_1 + x_2$
Create	$(x_1 + \mathrm{y})^1(x_1 + x_2)$ $= (x_1x_1 + x_1x_2 + \mathrm{y}x_1 + \mathrm{y}x_2)$ $= x_1 + x_2 + \mathrm{y} + \mathrm{y}x_2$	$(x_1 + x_2)(x_1 + \mathrm{y})^1$ $= (x_1x_1 + x_1\mathrm{y} + x_2x_1 + x_2\mathrm{y})$ $= x_1 + \mathrm{y} + x_2 + x_2\mathrm{y}$
Conserve	$x_1(x_1 + x_2 + \mathrm{y} + \mathrm{y}x_2)$ $= (x_1x_1 + x_1x_2 + \mathrm{y}x_1 + x_1\mathrm{y}x_2)$ $= x_1 + x_2 + \mathrm{y} + \mathrm{y}x_2$ $x_2(x_1 + x_2 + \mathrm{y} + \mathrm{y}x_2)$ $= (x_2x_1 + x_2x_2 + x_2\mathrm{y} + x_2\mathrm{y}x_2)$ $= x_2 + x_1 + x_2\mathrm{y} + \mathrm{y}$ $\mathrm{y}(x_1 + x_2 + \mathrm{y} + \mathrm{y}x_2)$ $= (\mathrm{y}x_1 + \mathrm{y}x_2 + \mathrm{yy} + \mathrm{yy}x_2)$ $= (\mathrm{y} + \mathrm{y}x_2 + x_2 + x_2x_2)$ $= \mathrm{y} + \mathrm{y}x_2 + x_2 + x_1$ $\mathrm{y}x_2(x_1 + x_2 + \mathrm{y} + \mathrm{y}x_2)$ $= (\mathrm{y}x_2x_1 + \mathrm{y}x_2x_2 + \mathrm{y}x_2\mathrm{y} + \mathrm{y}x_2\mathrm{y}x_2)$ $= (\mathrm{y}x_2 + \mathrm{y}x_1 + x_2x_2 + x_2x_2x_2)$ $= (\mathrm{y}x_2 + \mathrm{y} + x_1 + x_1x_2)$ $= \mathrm{y}x_2 + \mathrm{y} + x_1 + x_2$	$x_1(x_1 + \mathrm{y} + x_2 + x_2\mathrm{y})$ $= (x_1x_1 + x_1\mathrm{y} + x_1x_2 + x_1x_2\mathrm{y})$ $= x_1 + \mathrm{y} + x_2 + x_2\mathrm{y}$ $\mathrm{y}(x_1 + \mathrm{y} + x_2 + x_2\mathrm{y})$ $= (\mathrm{y}x_1 + \mathrm{yy} + \mathrm{y}x_2 + \mathrm{y}x_2\mathrm{y})$ $= (\mathrm{y} + x_2 + \mathrm{y}x_2 + x_1)$ $x_2(x_1 + \mathrm{y} + x_2 + x_2\mathrm{y})$ $= (x_2x_1 + x_2\mathrm{y} + x_2x_2 + x_2x_2\mathrm{y})$ $= x_2 + x_2\mathrm{y} + x_1 + x_1\mathrm{y}$ $= x_2 + x_2\mathrm{y} + x_1 + \mathrm{y}$ $x_2\mathrm{y}(x_1 + \mathrm{y} + x_2 + x_2\mathrm{y})$ $= (x_2\mathrm{y}x_1 + x_2\mathrm{yy} + x_2\mathrm{y}x_2 + x_2\mathrm{y}x_2\mathrm{y})$ $= (x_2\mathrm{y} + x_2x_2 + \mathrm{y}x_1 + x_1x_2)$ $= x_2\mathrm{y} + x_1 + \mathrm{y} + x_2$
Create	$(x_1 + \mathrm{y})^2(x_1 + x_2)$ $= (x_1 + \mathrm{y})(x_1 + \mathrm{y})(x_1 + x_2)$ $= (x_1x_1 + x_1\mathrm{y} + \mathrm{y}x_1 + \mathrm{yy})(x_1 + x_2)$ $= (x_1x_1x_1 + x_1\mathrm{y}x_1 + \mathrm{y}x_1x_1 + \mathrm{yy}x_1$ $+ x_1x_1x_2 + x_1\mathrm{y}x_2 + \mathrm{y}x_1x_2 + \mathrm{yy}x_2)$ $= x_1 + \mathrm{y} + \mathrm{y} + \mathrm{yy} +$ $x_2 + \mathrm{y}x_2 + \mathrm{y}x_2 + \mathrm{yy}x_2$	$(x_1 + x_2)(x_1 + \mathrm{y})^2$ $= (x_1 + x_2)(x_1 + \mathrm{y})(x_1 + \mathrm{y})$ $= (x_1x_1 + x_1\mathrm{y} + x_2x_1 + x_2\mathrm{y})$ $= (x_1x_1x_1 + x_1\mathrm{y}x_1 + x_2x_1x_1 + x_2\mathrm{y}x_1$ $+ x_1x_1\mathrm{y} + x_1\mathrm{yy} + x_2x_1\mathrm{y} + x_2\mathrm{yy})$ $= x_1 + \mathrm{y} + x_2 + x_2\mathrm{y} +$ $\mathrm{y} + \mathrm{yy} + x_2\mathrm{y} + x_2\mathrm{yy}$
...	...	...

To simplify, we can use the following properties of x_1, x_2 and y:

Derived symbol relations from Conserve:

$$x_1x_1 = x_1$$
$$x_2x_2 = x_1$$
$$x_1x_2 = x_2x_1 = x_2$$
$$x_1\mathrm{y} = \mathrm{y}x_1 = \mathrm{y}$$
$$x_2\mathrm{y} = \mathrm{y}x_2 = \mathrm{y}$$

Derived symbol identities from Conserve:

$$x_1 = 1$$
$$x_2 = -1$$
$$\mathrm{y} = i_0,\ i_1,\ j_1,\ i_2,\ j_2,\ ...$$

5.3 Variable Finite Automata

The Universal Rewrite System is generated by a specific regular expression and recognised by an equally specific finite automaton. Grumberg et al [20] created the concept of Variable Finite Automata to allow finite state machines to process an infinite alphabet using the same number of states as the finite alphabet. We are going further by stating that: For every finite state machine based on a finite alphabet set there exists a variable finite state machine based on an infinite alphabet set. That is, each finite alphabet has an infinite alphabet twin, like local and nonlocal descriptions in physics. How can a state machine recognise strings from an infinite alphabet without needing to have an infinite set of states to remember each symbol? The solution given by Grumberg et al is to divide the symbols of the infinite alphabet into two types of variable sets: a bound variable set X and a single free variable set {y}.

5.3.1 The Non-Deterministic Variable Finite Automaton

The left side of the universal rewrite system semantic equation

$$(x_1 + y)^*(x_1 + x_2) = (x_1 + x_2)(x_1 + y)^*$$

is

$$(x_1 + \mathrm{y})^*(x_1 + x_2)$$

and the Variable Finite Automaton that recognises it is non-deterministic.

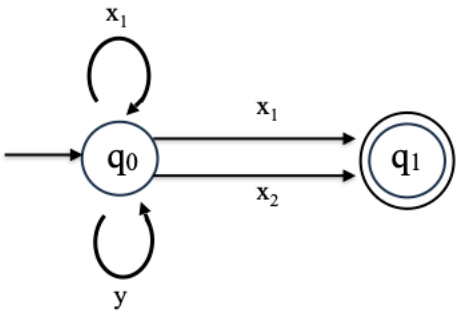


Figure 1: Non-Deterministic Variable Finite Automaton

	x_1	x_2	y
q_0	$\{q_0, q_1\}$	q_1	q_0
q_1	$\emptyset$	$\emptyset$	$\emptyset$

The number of states of a non-deterministic machine are 2 raised to the number of states of the deterministic version of the machine, Q.

states Q = {q_0, q_1}
final state F = {q_1}
subsets of states 2^Q = {$\emptyset$, {q_0}, {q_1}, {q_0, q_1}}

Written as a deterministic variable finite automaton the non-deterministic variable finite automaton is

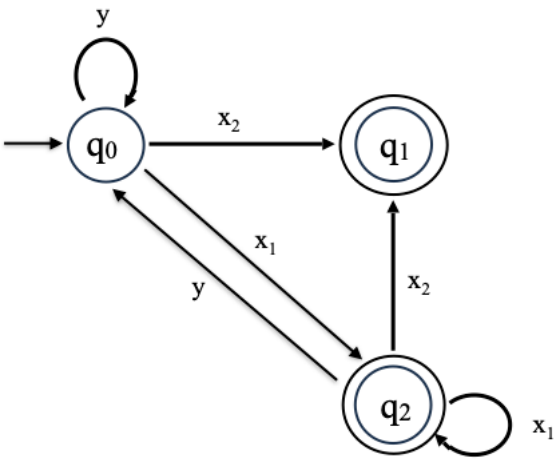


Figure 2: NonDeterministic Finite Automaton to Deterministic Finite Automaton

	x_1	x_2	y
q_0	q_2	q_1	q_0
q_1	$\emptyset$	$\emptyset$	$\emptyset$
q_2	q_2	q_1	q_0

states Q = {q_0, q_1, q_2}
final states F = {q_1, q_2}

5.3.2 The Deterministic Variable Finite Automaton

The right side of the universal rewrite system semantic equation

$$(x_1 + y)^*(x_1 + x_2) = (x_1 + x_2)(x_1 + y)^*$$

is

$$(x_1 + x_2)(x_1 + y)^*$$

and its Variable Finite Automaton recogniser is

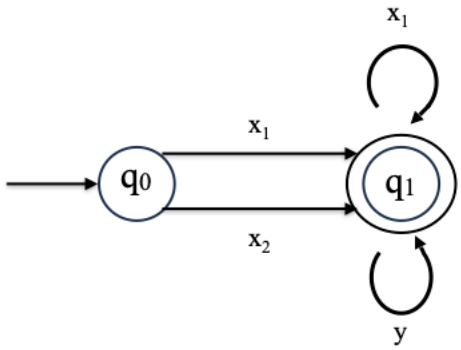


Figure 3: Deterministic Variable Finite Automaton

with transition function

	x_1	x_2	y
q_0	q_1	q_1	$\emptyset$
q_1	q_1	$\emptyset$	q_1

The number of states of a deterministic machine are just the number of states, Q.

states Q = $\{q_0, q_1\}$
final state F = $\{q_1\}$

The simultaneous necessity of deterministic and non-deterministic finite automata can be compared to the simultaneous necessity of locality and non-locality in the subject of nilpotent quantum mechanics, where they are defined as inside and outside the bracket defining the nilpotent operator[1],[2],[3]. However, the two machines (Regexes) describe the same language only when assigned to the URS symbols, e.g. in a Clifford algebra. It is remarkable that these two machines, which we propose describe all processes in Nature, require only two states at the minimal level of complexity.

It is surely very significant that the machine that is designed to generate all the structures in nature is a two-state machine and that it comes in two forms, which can be considered as complementary, performing their tasks in opposite orders. This is very much the structure of the URS, and it relates closely to anti-commutativity. Duality and complementarity are everywhere in nature to ensure totality zero. We can see this in the double space or double Clifford algebra of the URS, and in the factors of 2 and $\frac{1}{2}$ that occur everywhere in physics [1],[2],[3]. At a more subtle level, the duality and significance of the factor 2 are seen in the prevalence of quadratic and second-order differential equations, of Pythagoras theorem, of inverse-square laws, of the squaring of the Dirac operator, of the holographic principle, and probably even of Fermat's last theorem. Doubling and squaring and their inverse processes are fundamental in the structure of nature in maintaining totality zero.

The use of variables in the machines enables us to see the pattern of the machines that govern nature using only two states. The actual transitions would require an infinite number of terms and an infinite memory but the pattern can be established if there is a degree of repetition, which can be represented by making y a variable. This is connected to the insertion by Clifford of repeating quaternion systems into the infinite Grassmann algebra to create the Clifford algebra, which appears to be the algebra of nature.

Zero totality runs throughout the Chomsky hierarchy of formal languages. Zero totality can be expressed as a Type 3 language (as shown above), processed by two types of two state Variable Finite Automata. It can also be expressed as a Type 0 language L = $\{0^{2^n}, n \geq 0\}$ accepted by the five-state Turing Machine[21]. In addition, we can compose the machines where the zero totality represented by 0 in the expression 0^{2^n}, can be reinterpreted as $(x_1 + y)^*(x_1 + x_2)$ or $(x_1 + x_2)(x_1 + y)^*$ and composed as input into the type 0 language L = $\{0^{2^n}, n \geq 0\}$ that is accepted by the five state Turing machine shown in the diagram below.

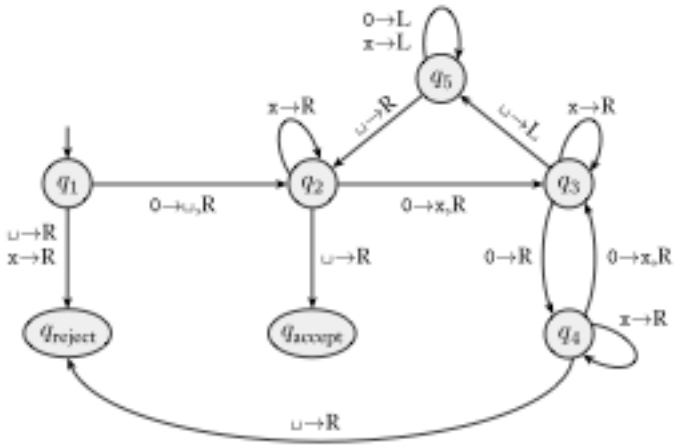


Figure 4: Diagram courtesy of Sipser

5.4 Type 3 Languages

Type 3 languages are the simplest and the most restricted languages in the Chomsky hierarchy of formal language theory. They use Boolean type operators to combine and manipulate strings and sets of strings.

Operation	Operation	Expression
Parallel	$+$	$x + y$
Sequential	$\times$	xy
Sequential Repetition	$*$	x^n, $n \geq 0$

The distributive law of + over × applies even in formal language theory. All the possible words of length two are described by the regular expression

$$(a + b)(c + d)$$

are given by the 'product' of

$$(a + b)(c + d) = ac + ad + bc + bd.$$

6 The URS Syntax

The Universal Rewrite System is generated by a specific regular expression and recognised by an equally specific finite automaton. For every finite state machine based on a finite alphabet set there exists a variable finite state machine based on an infinite set of alphabet symbols. How can a state machine recognise strings from an infinite alphabet without needing to have an infinite set of states to remember each symbol? As we have seen it is by converting the symbols of the infinite alphabet into two types of variables: bound X and free $\{y\}$.

6.1 The URS Grammar

For $(x_1 + x_2)(x_1 + y)^*$ the grammar becomes

$$q_0 \rightarrow (x_1 + x_2)q_1$$
$$q_1 \rightarrow (x_1 + y)q_1 + \epsilon$$

where ϵ = the empty string

For $(x_1 + y)^*(x_1 + x_2)$ the grammar becomes

$$q_0 \rightarrow (x_1 + y)q_0 + (x_1 + x_2)q_1$$
$$q_1 \rightarrow \epsilon$$

where ϵ = the empty string

To reproduce the URS, the free variable y must take assigned values from the following Clifford numbers i_0, i_1, j_1, i_2, j_2, ...

6.2 The URS Semantics

$$\boxed{\text{Create:}(x_1 + y)^*(x_1 + x_2) = (x_1 + x_2)(x_1 + y)^*}$$
$$=$$
$$\boxed{\text{Create:}(x_1 + y)^n(x_1 + x_2) = (x_1 + x_2)(x_1 + y)^n, \text{ n} \geq 0}$$

The identity of y can be determined from the order of the URS where the order of the URS = the word length minus 1. The word length = n + 1 where n = $\log_2 2^n$ and where the order of the URS = 2^n, $n \geq 1$.

The definition of a Variable Finite Automaton (VFA) is a pair Σ_A, A, where Σ is an infinite alphabet and A is a nondeterministic finite pattern automaton that recognises type 3 or regular languages[19]. For the URS regular expression, let the alphabet or symbol set of the finite pattern automaton A be $\Gamma_A = X \cup \{y\}$ where $X = \{x_1, x_2 | x_1 = 1 \text{ and } x_2 = -1\}$ is the set of bound variables and $\{y\} = \{y | y = i_{m(n)}$ or $y = j_{m(n)}$, $m(n) \in N\}$ is the set of the single free variable y. More specifically, let the symbols (aka

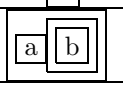
letters) of the finite pattern automaton A be $v_1...v_k...v_n$ where $2 \leq k \leq n$ and $n \in N$ and let the elements of the infinite alphabet Σ be $w_1...w_k...w_n$ where $2 \leq k \leq n$ and $n \in N$. Then the finite pattern automaton A for the URS regular expression, Γ_A, becomes $\Gamma_A^* = \{v_1...v_k...v_n | v_1 = x_1$ or $v_1 = x_2$, $k \neq 1$, $2 \leq k \leq n$ where $n \in N\}$ where the URS regular expression is split between the fixed variable alphabet $X = \{v_1 | v_1 = x_1$ or $v_1 = x_2\}$ and the set of the singular free variable $\{y\} = \{y = i_{m(n)}$ or $y = j_{m(n)}$ where $m(n) \in N\} = \{v_k | v_k = i_{m(n)}$ or $v_k = j_{m(n)}, k \neq 1, 2 \leq k \leq n$ where $m(n), n \in N\}$. The finalised form of the URS regular expression as expressed in the bound and free variables of the finite pattern automaton A is $\Gamma_A^* = \{(x_1 + x_2)(x_1 + y)^* = (x_1 + y)^*(x_1 + x_2)$, where $x_1 = 1$ and $x_2 = -1$, $x_1 \neq x_2$ and y $= i_{m(n)}$ or y $= j_{m(n)}$, $m(n) \in N\}$. (Note that the first letter v_1 is always only assigned to $x_1 = 1$ or $x_2 = -1$.) This determines the language of the infinite alphabet Σ to be $\Sigma^* = \{w_1..w_k...w_n | w_1 = v_1 = x_1$ or $w_1 = v_1 = x_2$ and $w_k = v_k = y = i_{m(n)}$ or $w_k = v_k = y = j_{m(n)}$, $k \neq 1$, $2 \leq k \leq n$, $m(n), n \in N\}$.

The semantics symbol assignment put in tabular form. $x_1 = 1$, $x_2 = -1$, y $= i_{m(n)}$ or y $= j_{m(n)}$

1	-1	$i_{m(n)}$	$j_{m(n)}$
x_1	x_2	y	y
v_1	v_1	v_k	v_k
w_1	w_1	w_k	w_k

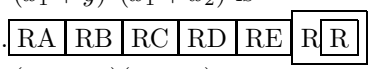
7 Boundary Logic

Let a be a simple statement. Then the expressions of the Universal Rewrite System can be found in Boundary Logic and Boundary Mathematics[22],[23].To represent regular expressions in Boundary logic, we use the following operations:

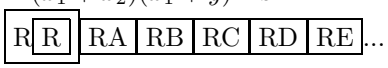
not a	
a or b	ab
a and b	
a implies b	
a or not b	
a and not b	

The Universal Rewrite System Variable Regular Expression translated into Boundary Logic expression now becomes:

$(x_1 + y)^*(x_1 + x_2)$ is

void = ... 

$(x_1 + x_2)(x_1 + y)^*$ is

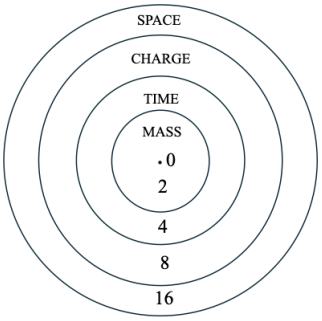
void =  ...

In Boundary Mathematics, we have

positive one	$( ) = \#^0 = 1$
zero = placeholder	$< > = 0$
negative one	$< ( ) > = < \#^0 > = 0 - 1 = -1$
negative infinity	$[ ] = log_{\#} 0 = -\infty$
james number J	$log_{\#}(-1) = [<( )>]$
i_1	$(\frac{J_1}{2}) = \#^{\frac{J_1}{2}}$
j_1	$(\frac{J_2}{2}) = \#^{\frac{J_2}{2}}$
$i_1 j_1$	$(\frac{J_1}{2})(\frac{J_2}{2}) = (\#^{\frac{J_1}{2}})(\#^{\frac{J_2}{2}}) = \#^{\frac{J_1+J_2}{2}}$

Zero in boundary systems is serving as a notational placeholder, as void would be symbolised as an empty space, which according to the Universal Rewrite System is not void or nothing. Boundary systems represent void as an 'empty' space but, as seen from the Universal Rewrite System, void $\neq$ 'empty' space. In the Universal Rewrite System void is restored to absolutely nothing as the starting point for the Universal Rewrite System (= everything). Where Boundary systems need an empty space to mark, these systems assume that space is 'empty' or that void = space (order 16) minus mass (order 2) minus time (order 4) minus charge (order 8). However, the Universal Rewrite System indicates that 'empty' space $\neq$ void and that space is not empty: space is the product of mass (order 2) $\times$ time (order 4) $\times$ charge (order 8)): zero = $<>\neq$ void.

In a diagram representing the Universal Rewrite System, the successive orders representing the physical parameters can be shown as concentric Venn diagrams with corresponding orders of terms 2, 4, 8, 16. It can be seen that space does not correspond to void which is represented here by a point in the centre, with zero order.



The Variable Regular Expression for URS translated into Boundary Mathematics expression becomes:

$$(x_1 + y)^*(x_1 + x_2) \text{ is void} = ...[()(\tfrac{J_1}{2})][()(\tfrac{J_2}{2})][()(\tfrac{J_3}{2})][()(\tfrac{J_4}{2})][()(\tfrac{J_5}{2})][()<()>]$$
$$(x_1 + x_2)(x_1 + y)^* \text{ is void} = [()<()>][()(\tfrac{J_1}{2})][()(\tfrac{J_2}{2})][()(\tfrac{J_3}{2})][()(\tfrac{J_4}{2})][()(\tfrac{J_5}{2})]...$$

One of the main ways to try to generate ideas from fundamentals is to use Spencer-Brown's laws of form (LOF), forming the basis of Boundary Logic and Mathematics, as we have used them in this section. Here, we use a mark or boundary, in this case represented by a box, to mark a distinction between a and NOT a. The law of crossing, with a box inside a box, cancels the operation. The law of calling says repeating the mark has no effect on the operation; here, it would be two boxes next to each other would be equivalent to a single box. In the URS the equivalent of the laws of calling and crossing do not occur at the initial stages where we have create and conserve, but we can generate them at a higher stage as here. The URS create and conserve processes cannot be identified as the LOF calling and crossing because these laws need a double space of 64 elements to be able to have an environment (one space) to draw the mark, with the nilpotent fermion as the first space. Create and conserve can start from nothing. It is essential to recognise some of the hidden assumptions of Boundary Logic and Boundary Mathematics, although they can also be made useful to us. According to the URS, nothing does not equate to empty space which occurs at order 16 according to the birth-ordering of the URS, not at the first stage of the process. Both Boundary Mathematics and Boundary Logic assume that space is empty or void, but space in the URS is the fourth concept in the sequence (order 16), after the true void (absolutely nothing) and then mass (order 2), time (order 4), charge (order 8), and physical space cannot exist without these parameters as well, and without them being necessary sub-algebras. The URS shows that empty space and void are different concepts.

8 A Three Valued Logic Rewrite

Assuming the results in [26], let the three valued logic of NOT, XOR and AND be defined using modular 3 negation, modular 3 addition and modular 3 multiplication. Let $\oplus_3$ symbolise modular 3 addition and let $\otimes_3$ symbolise modular 3 multiplication as represented in the tables below.

	$-_3$
0	0
1	2
2	1

$\oplus_3$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\otimes_3$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Suppose the values 0, 1, and 2 are rewritten as the sign combinations $1 \to +$, and $2 \to -$, $0 \to (\pm)$ or $-0 \to (\mp)$. (Consider a pair [a, b] and define $-$[a, b] = [$-$b,$-$a]. Then $-[1,-1] = [-(-1), -1] = [1,-1] = 0$ and $-[-1,1] = [-1,-(-1)] = [-1, 1] = -0$. This can be used to make a four valued logic or a three valued logic if we collapse [1,$-$1] and [$-$1,1] by equating [1,$-$1] and [$-$1,1] which in standard notation becomes 0 = -0.[27],[28].) Then a three valued logical calculus can be represented using these sign combinations and the logical calculations can be shown to be consistent with modular 3 addition and modular 3 multiplication using the conventional values of 0, 1, and 2. Using both 0 as ($\pm$) and -0 as ($\mp$) gives the same values as the modular 3 addition and modular 3 multiplication tables shown at the beginning of this section.

	$-_3$
$(\pm)$	$(\mp)$
$+$	$-$
$-$	$+$

Using 0 as ($\pm$)

$\oplus_3$	$(\pm)$	$+$	$-$
$(\pm)$	$(\mp)$	$+$	$-$
$+$	$+$	$-$	$(\pm)$
$-$	$-$	$(\mp)$	$+$

Using -0 as ($\mp$)

$\oplus_3$	$(\mp)$	$+$	$-$
$(\mp)$	$(\pm)$	$+$	$-$
$+$	$+$	$-$	$(\pm)$
$-$	$-$	$(\mp)$	$+$

Using 0 as ($\pm$)

$\otimes_3$	$(\pm)$	$+$	$-$
$(\pm)$	$(\mp)$	$(\pm)$	$(\mp)$
$+$	$(\pm)$	$+$	$-$
$-$	$(\mp)$	$-$	$+$

Using -0 as ($\mp$)

$\otimes_3$	$(\mp)$	$+$	$-$
$(\mp)$	$(\mp)$	$(\mp)$	$(\pm)$
$+$	$(\mp)$	$+$	$-$
$-$	$(\pm)$	$-$	$+$

Using both 0 as $(\pm)$ and -0 as $(\mp)$

$\otimes_3$	$(\mp)$	$+$	$-$
$(\pm)$	$(\pm)$	$(\pm)$	$(\mp)$
$+$	$(\mp)$	$+$	$-$
$-$	$(\pm)$	$-$	$+$

Using both 0 as $(\pm)$ and -0 as $(\mp)$

$\otimes_3$	$(\pm)$	$+$	$-$
$(\mp)$	$(\pm)$	$(\mp)$	$(\pm)$
$+$	$(\pm)$	$+$	$-$
$-$	$(\mp)$	$-$	$+$

$$(\pm) \oplus_3 (\pm) = \begin{pmatrix}(+ \oplus_3 +)\\(- \oplus_3 -)\end{pmatrix} = (\mp)$$
$$(\mp) \oplus_3 (\mp) = \begin{pmatrix}(- \oplus_3 -)\\(+ \oplus_3 +)\end{pmatrix} = (\pm)$$

$$(\pm) \oplus_3 + = ((+ \oplus_3 +) \oplus_3 -) = (- \oplus_3 -) = +$$
$$(\pm) \oplus_3 - = (+ \oplus_3 (- \oplus_3 -)) = (+ \oplus_3 +) = -$$

$$(\mp) \oplus_3 + = (- \oplus_3 (+ \oplus_3 +)) = (- \oplus_3 -) = +$$
$$(\mp) \oplus_3 - = ((- \oplus_3 -) \oplus_3 +)) = (+ \oplus_3 +) = -$$

$$(\pm) \otimes_3 (\pm) = \begin{pmatrix}(+ \otimes_3 +) \oplus_3 (- \otimes_3 -)\\(+ \otimes_3 -) \oplus_3 (- \otimes_3 +)\end{pmatrix} = \begin{pmatrix}(+ \oplus_3 +)\\(- \oplus_3 -)\end{pmatrix} = (\mp)$$
$$(\mp) \otimes_3 (\mp) = \begin{pmatrix}(- \otimes_3 -) \oplus_3 (+ \otimes_3 +)\\(- \otimes_3 +) \oplus_3 (+ \otimes_3 -)\end{pmatrix} = \begin{pmatrix}(+ \oplus_3 +)\\(- \oplus_3 -)\end{pmatrix} = (\mp)$$
$$(\pm) \otimes_3 (\mp) = \begin{pmatrix}(+ \otimes_3 -) \oplus_3 (- \otimes_3 +)\\(+ \otimes_3 +) \oplus_3 (- \otimes_3 -)\end{pmatrix} = \begin{pmatrix}(- \oplus_3 -)\\(+ \oplus_3 +)\end{pmatrix} = (\pm)$$
$$(\mp) \otimes_3 (\pm) = \begin{pmatrix}(- \otimes_3 +) \oplus_3 (+ \otimes_3 -)\\(- \otimes_3 -) \oplus_3 (+ \otimes_3 +)\end{pmatrix} = \begin{pmatrix}(- \oplus_3 -)\\(+ \oplus_3 +)\end{pmatrix} = (\pm)$$

$$(\pm) \otimes_3 + = \begin{pmatrix}(+ \otimes_3 +)\\(- \otimes_3 +)\end{pmatrix} = (\pm)$$
$$(\pm) \otimes_3 - = \begin{pmatrix}(+ \otimes_3 -)\\(- \otimes_3 -)\end{pmatrix} = (\mp)$$

$$(\mp) \otimes_3 + = \begin{pmatrix}(- \otimes_3 +)\\(+ \otimes_3 +)\end{pmatrix} = (\mp)$$
$$(\mp) \otimes_3 - = \begin{pmatrix}(- \otimes_3 -)\\(+ \otimes_3 -)\end{pmatrix} = (\pm)$$

9 Cellular Automata

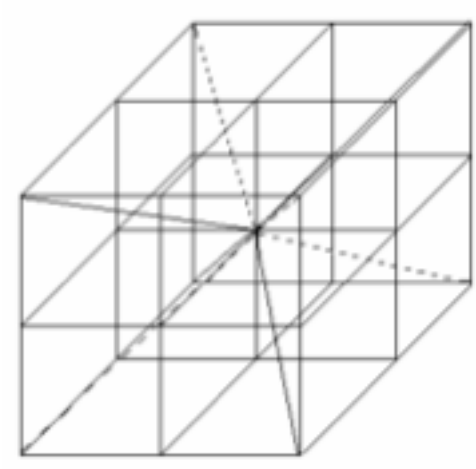


Figure 5: Diagram courtesy of Rowlands[12]

Rowlands has shown how the properties of certain binary cellular automata (as investigated by Mainzer and Chua) [24] can be derived from the physical interpretation of the URS [12],[24]. We can extend the treatment of cellular automata to ternary or three - valued logic. Let us take a cube as a lattice of 27 points: A cellular automaton cube of 27 points can be labelled with quaternion coordinates (i, j, k) using a balanced ternary system 1, 0, -1 = 1, 0, 2 as modular three encoding where the assignments are: 1 => 1, 0 => 0, -1 => 2; (i, j, k) = (0, 0, 0) is the centre of cube. The CPT transformation matrices oscillate between all the 27 points of the balanced ternary Boolean cube with intersecting planes [4],[5],[6]. The two variable regexes $(x_1 + y)^*(x_1 + \sqrt{x_2})$ and $(x_1 + \sqrt{x_2})(x_1 + y)^*$, where y = $\sqrt{x_2}$, are an infinite number of these ternary cubes juxtaposed next to each other.

$$\mathrm{F(p,\ q,\ r)} = \mathrm{(p,\ q',\ r)}$$
$$\mathrm{F(i,\ j,\ k)} = \mathrm{(i,\ j',\ k)}$$
$$\mathrm{F}((\sqrt{x_2})_1,\ (\sqrt{x_2})_2,\ (\sqrt{x_2})_3) = \mathrm{F}((\sqrt{x_2})_1,\ (\sqrt{x_2})_2',\ (\sqrt{x_2})_3)$$

For binary cellular automata, Mainzer and Chua have four 4 rules (110, 124, 137, 193) showing long-term stability (with transitions equivalent to identity, C, P, T) in the basic 256 that arise from purely binary logic. We can extend the Boolean binary cube with 8 vertices to a ternary cube with 27. As the order n of the URS increases, the index numbers of the quaternions label the points of the ternary cube, juxtaposed as an increasing sequence of quaternions.

p	q	r	Mod 3 Rule 110	Mod 3 Rule 124	Mod 3 Rule 137	Mod 3 Rule 193
0	0	0	0	0	2	2
0	0	1	1	0	0	1
0	0	2	2	0	1	0
0	1	0	1	1	0	0
0	1	1	1	1	1	1
0	1	2	1	1	2	2
0	2	0	2	2	1	1
0	2	1	1	2	2	1
0	2	2	0	2	0	1
1	0	0	0	1	1	0
1	0	1	1	1	1	1
1	0	2	2	1	1	2
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	2	2	2	2	2
1	2	0	2	1	1	2
1	2	1	2	2	2	2
1	2	2	2	0	0	2
2	0	0	0	2	0	1
2	0	1	1	2	2	1
2	0	2	2	2	1	1
2	1	0	1	1	2	2
2	1	1	2	2	2	2
2	1	2	0	0	2	2
2	2	0	2	0	1	0
2	2	1	0	2	2	0
2	2	2	1	1	0	0

10 Nilpotent Dirac Operator as a 32 x 32 Matrix

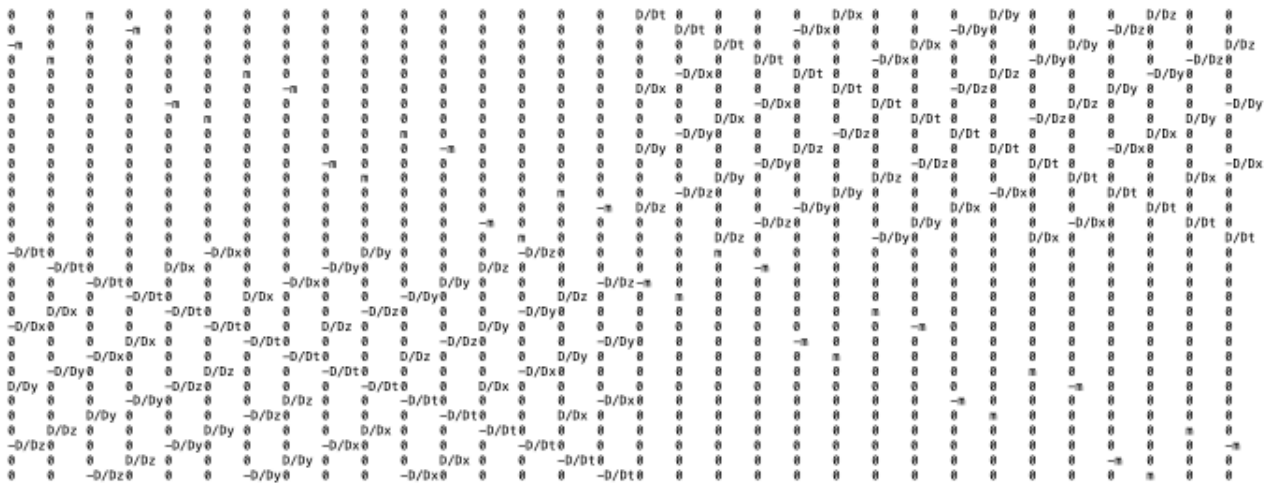
After descrambling the Dirac equation using the 64-part Clifford algebra [1],[2],[3] we have started work on converting all the 64 algebraic operators back into matrix form, this time into 32×32 real number matrices. Why? The idea is to code the process of solving the nilpotent Dirac equation in a high-level language such as C++ and transfer the code into a special purpose field programmable gate array (FPGA) designed solely to find solutions to the nilpotent Dirac equation. Using existing object-oriented C++ code that is written to handle matrices as instantiated objects we will use this code to program the FPGA. We have started encoding the 64-part Clifford algebra into native binary with the purpose of enabling and rendering faster, more complicated computational solutions to the nilpotent Dirac equation, bypassing the compilation stage of a high-level language which would cost computing time and space.

Here we have produced a matrix, structured on one specific pentad:

$$i_3 i_1 j_1 + i_3 i_2 i_1 + i_3 j_2 i_1 + i_3 j_2 i_2 i_1 + j_1$$

This is the rewrite basis for the fermion nilpotent operator

$$i\boldsymbol{k}E + \boldsymbol{i}\mathbf{i}p_x + \boldsymbol{i}\mathbf{j}p_y + \boldsymbol{i}\mathbf{k}p_z + \boldsymbol{j}m$$



It is possible to use any of the twelve pentads to do the structuring and we now also have a Java program to compute the matrix. The matrix, additionally indicates why the 64-part algebra produces two different representations for the fundamental fermions, one based on the fermions as pentads (12×5 of the 64) and the other using the individual elements of the algebra to represent 48 fermions and 16 non-fermionic algebraic space elements [2]. While the individual units head the rows and columns, the contents are the pentads.

It seems probable that the URS can be encoded in binary because both the URS and binary expand at each order as powers of 2 or 2^n. In this way, the URS can find a natural expression in binary, whether the 2's are in the form 1/0 or +/-. Here, we would need to show that all the numbers of the URS have a unique sign pattern representation that obeys the Clifford Algebra Product in native binary, both in 1/0 and in +/- forms, or any two complementary pairs. In section 12 of this paper, we present calculations using the +/- form, to reveal the sign patterns of each element in the URS using the Clifford product, and employing this product to encode the sign patterns into binary digits. This can generate the internal representations for data structures in a potential programming language built for the URS as well as leading to the construction of a URS calculator built at the circuit level.

11 The Universal Rewrite System in Category Theory

Let W, U, T, and Z be any objects (in this discussion, objects are taken as sets) in any category with products. Let N be the set of natural numbers. Then the following diagram shows the Universal Rewrite System as a category.

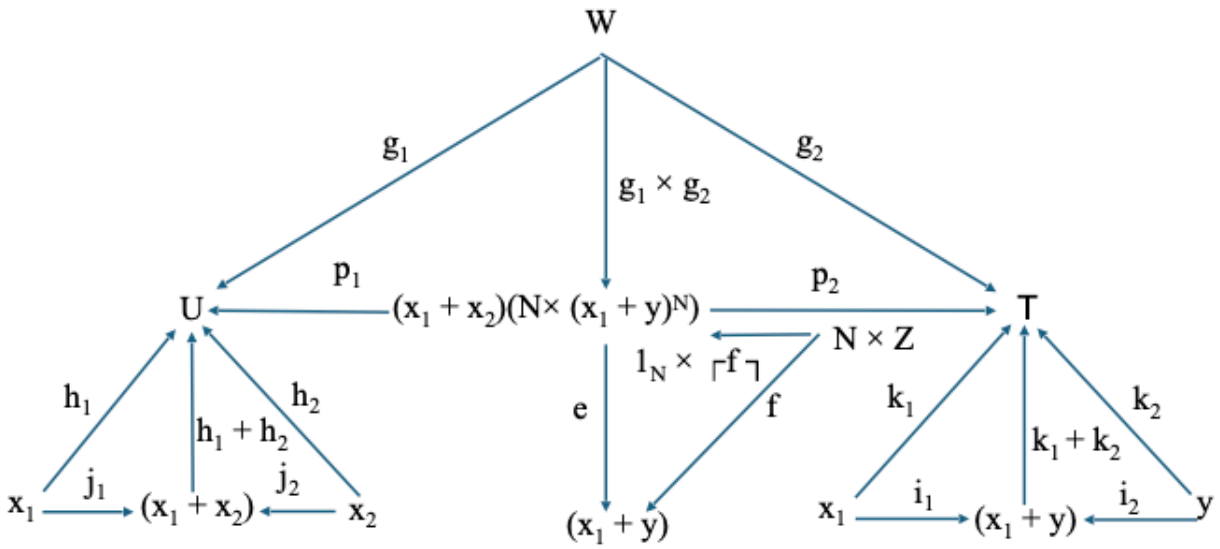


Figure 6: The Universal Rewrite System as a Category

11.1 The Universal Rewrite System as a Binary Map Object

Let 2 represent the set {1, -1} and let n represent the set of generators of order 2^n. Then the Universal Rewrite System can be expressed as a map object.

n	2^n: n → 2
{1}	{1} → {1, -1}
{1, i_1}	{1, i_1} → {1, -1}
{1, i_1, j_1}	{1, i_1, j_1} → {1, -1}
{1, i_1, j_1, i_2}	{1, i_1, j_1, i_2} → {1, -1}
{1, i_1, j_1, i_2, j_2}	{1, i_1, j_1, i_2, j_2} → {1, -1}
{1, i_1, j_1, i_2, j_2, i_3}	{1, i_1, j_1, i_2, j_2, i_3} → {1, -1}

12 Encoding the Universal Rewrite System into Binary

This section details how to represent the Universal Rewrite System pattern in binary. This binary representation can be converted into an encoding scheme ready for implementation into computing circuitry. This encoding is computed using one or the other of the two logical functions XOR or XOR's negation, namely, XNOR. If multiple XOR or XNOR gates are connected in parallel they will be able to process a binary digit pattern representing the unique sign patterns of all the algebraic elements of the URS, according to the particular XOR or XNOR gate that is being computed in parallel. All the gates have to be of the same type, this encoding scheme will not work if parallel XOR gates are mixed with parallel XNOR gates.

Let n, m, $\in \mathrm{Z}^+$ such that n, $m \geq 1$. There are two types of quaternion products. The first type of product takes the defining form of $i_n j_n$ which as Hamilton discovered anti-commutes (the order in which the terms of the products are multiplied creates an opposite difference in the result). What this means for the binary encodings is that the minus sign must be factored into the anti-commutative product whenever the order of the binary encodings of i_n and j_n change. The second type of product is of the form $i_n i_m$ or $i_n j_m$ $(n \neq m)$ and this form commutes without a minus being factored into the commutative product (the order in which the terms of the products are multiplied does not create a difference in the result). For this commutative product, when the order of the binary encodings are changed, no minus sign occurs in the product. The two signature sign patterns for all the i's and j's and the length of the sign patterns determine the identity of that particular i and j. The identity of a particular i and j is indicated by the numerical subscript next to the i or j. These two features are all that are needed to faithfully encode the URS into binary notation, without the need for data structures in higher levels of code (i.e. class definitions in object oriented languages).

a_0b_0			b_0	b_0
			1	-1
			$+$	$-$
a_0	1	$+$	$+$	$-$
a_0	-1	$-$	$-$	$+$

$\neg$XOR		$+$	$-$
		1	0
$+$	1	1	0
$-$	0	0	1

XOR		$-$	$+$
		1	0
$-$	1	0	1
$+$	0	1	0

If XOR is chosen as the gate for computation, then all the gates must be XOR, no mixing of XOR and XNOR is allowed because choosing the XOR forces the coding for 1 to be $-$ and for 0 to be $+$. Alternatively, if the XNOR gate is chosen, then the encoding for 1 is $+$ and for 0 is $-$. Sign consistency is mandatory for the entire binary encoding scheme to work. The rules shown in the table below are used for the entire rewrite system and are only used for identical quaternions, that is, i quaternions in the same quaternion set indicated in the Clifford algebra by the same subscripted number and j quaternions in the same quaternion set indicated in the Clifford algebra by the same subscripted number, up to a sign difference, in the XOR/XNOR gates. Positive one, 1, as a sign pattern and at any order, 2^p, other than order 2 reduces from $(+)^n$ to $(+)$. Likewise, negative one, -1, as a sign pattern and at any order, 2^p, other than order 2 reduces from $(-)^n$ to $(-)$:

order 2^p	$\pm 1 \times \pm 1$	$\frac{\pm a_n b_n}{a_n b_n}$
order 4	$\pm i_0 \times \pm i_0$	$\frac{\pm a_1 b_0 a_0 b_1}{a_n b_n}$
order 8	$\pm i_1 \times \pm i_1$	$\frac{\pm a_1 b_0 a_0 b_1 a_3 b_2 a_2 b_3}{a_n b_n}$
order 8	$\pm j_1 \times \pm j_1$	$\frac{\pm a_2 b_0 a_3 b_1 a_0 b_2 a_1 b_3}{a_n b_n}$
order 32	$\pm i_2 \times \pm i_2$	$\frac{\pm a_1 b_0 a_0 b_1 a_3 b_2 a_2 b_3 a_5 b_4 a_4 b_5 a_7 b_6 a_6 b_7}{a_n b_n}$
order 32	$\pm j_2 \times \pm j_2$	$\frac{\pm a_2 b_0 a_3 b_1 a_0 b_2 a_1 b_3 a_6 b_4 a_7 b_5 a_4 b_6 a_5 b_7}{a_n b_n}$

Order 2^n	URS Algebraic Elements	URS Binary Encoding
2	1	$(+)$
	-1	$(-)$
4	1	$(+)$
	-1	$(-)$
	i_0	$(+\ -)$
	$-i_0$	$(-\ +)$
8	1	$(+)$
	-1	$(-)$
	i_1	$(+\ -\ +\ -)$
	$-i_1$	$(-\ +\ -\ +)$
	j_1	$(+\ +\ -\ -)$
	$-j_1$	$(-\ -\ +\ +)$
	$i_1 j_1$	$(+\ -\ +\ -)(+\ +\ -\ -)$
	$-i_1 j_1$	$(-\ +\ -\ +)(+\ +\ -\ -)$

16	1	$(+)$
	-1	$(-)$
	i_1	$(+\,-\,+\,-)$
	$-i_1$	$(-\,+\,-\,+)$
	j_1	$(+\,+\,-\,-)$
	$-j_1$	$(-\,-\,+\,+)$
	i_1j_1	$(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_1j_1$	$(-\,+\,-\,+)(+\,+\,-\,-)$
	i_0	$(+\,-)$
	$-i_0$	$(-\,+)$
	i_0i_1	$(+\,-)(+\,-\,+\,-)$
	$-i_0i_1$	$(-\,+)(+\,-\,+\,-)$
	i_0j_1	$(+\,-)(+\,+\,-\,-)$
	$-i_0j_1$	$(-\,+)(+\,+\,-\,-)$
	$i_0i_1j_1$	$(+\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_0i_1j_1$	$(-\,+)(+\,-\,+\,-)(+\,+\,-\,-)$
32	1	$(+)$
	-1	$(-)$
	i_1	$(+\,-\,+\,-)$
	$-i_1$	$(-\,+\,-\,+)$
	j_1	$(+\,+\,-\,-)$
	$-j_1$	$(-\,-\,+\,+)$
	i_1j_1	$(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_1j_1$	$(-\,+\,-\,+)(+\,+\,-\,-)$
	i_2	$(+\,-\,+\,-\,+\,-\,+\,-)$
	$-i_2$	$(-\,+\,-\,+\,-\,+\,-\,+)$
	i_2i_1	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,-\,+\,-)$
	$-i_2i_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,-\,+\,-)$
	i_2j_1	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_2j_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-)$
	$i_2i_1j_1$	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_2i_1j_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,-\,+\,-)(+\,+\,-\,-)$
	j_2	$(+\,+\,-\,-\,+\,+\,-\,-)$
	$-j_2$	$(-\,-\,+\,+\,-\,-\,+\,+)$
	j_2i_1	$(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$-j_2i_1$	$(-\,-\,+\,+\,-\,-\,+\,+)(+\,-\,+\,-)$
	j_2j_1	$(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$-j_2j_1$	$(-\,-\,+\,+\,-\,-\,+\,+)(+\,+\,-\,-)$
	$j_2i_1j_1$	$(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-j_2i_1j_1$	$(-\,-\,+\,+\,-\,-\,+\,+)(+\,-\,+\,-)(+\,+\,-\,-)$
	i_2j_2	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)$
	$-i_2j_2$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-\,+\,+\,-\,-)$
	$i_2j_2i_1$	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$-i_2j_2i_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$i_2j_2j_1$	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$-i_2j_2j_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$i_2j_2i_1j_1$	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_2j_2i_1j_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$

64	1	$(+)$
	-1	$(-)$
	i_1	$(+\,-\,+\,-)$
	$-i_1$	$(-\,+\,-\,+)$
	j_1	$(+\,+\,-\,-)$
	$-j_1$	$(-\,-\,+\,+)$
	i_1j_1	$(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_1j_1$	$(-\,+\,-\,+)(+\,+\,-\,-)$
	i_2	$(+\,-\,+\,-\,+\,-\,+\,-)$
	$-i_2$	$(-\,+\,-\,+\,-\,+\,-\,+)$
	i_2i_1	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,-\,+\,-)$
	$-i_2i_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,-\,+\,-)$
	i_2j_1	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_2j_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-)$
	$i_2i_1j_1$	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_2i_1j_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,-\,+\,-)(+\,+\,-\,-)$
	j_2	$(+\,+\,-\,-\,+\,+\,-\,-)$
	$-j_2$	$(-\,-\,+\,+\,-\,-\,+\,+)$
	j_2i_1	$(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$-j_2i_1$	$(-\,-\,+\,+\,-\,-\,+\,+)(+\,-\,+\,-)$
	j_2j_1	$(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$-j_2j_1$	$(-\,-\,+\,+\,-\,-\,+\,+)(+\,+\,-\,-)$
	$j_2i_1j_1$	$(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-j_2i_1j_1$	$(-\,-\,+\,+\,-\,-\,+\,+)(+\,-\,+\,-)(+\,+\,-\,-)$
	i_2j_2	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)$
	$-i_2j_2$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-\,+\,+\,-\,-)$
	$i_2j_2i_1$	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$-i_2j_2i_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$i_2j_2j_1$	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$-i_2j_2j_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$i_2j_2i_1j_1$	$(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_2j_2i_1j_1$	$(-\,+\,-\,+\,-\,+\,-\,+)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	i_0	$(+\,-)$
	$-i_0$	$(-\,+)$
	i_0i_1	$(+\,-)(+\,-\,+\,-)$
	$-i_0i_1$	$(-\,+)(+\,-\,+\,-)$
	i_0j_1	$(+\,-)(+\,+\,-\,-)$
	$-i_0j_1$	$(-\,+)(+\,+\,-\,-)$
	$i_0i_1j_1$	$(+\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_0i_1j_1$	$(-\,+)(+\,-\,+\,-)(+\,+\,-\,-)$
	i_0i_2	$(+\,-)(+\,-\,+\,-\,+\,-\,+\,-)$
	$-i_0i_2$	$(-\,+)(+\,-\,+\,-\,+\,-\,+\,-)$
	$i_0i_2i_1$	$(+\,-)(+\,-\,+\,-\,+\,-\,+\,-)(+\,-\,+\,-)$
	$-i_0i_2i_1$	$(-\,+)(+\,-\,+\,-\,+\,-\,+\,-)(+\,-\,+\,-)$
	$i_0i_2j_1$	$(+\,-)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_0i_2j_1$	$(-\,+)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-)$
	$i_0i_2i_1j_1$	$(+\,-)(+\,-\,+\,-\,+\,-\,+\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_0i_2i_1j_1$	$(-\,+)(+\,-\,+\,-\,+\,-\,+\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	i_0j_2	$(+\,-)(+\,+\,-\,-\,+\,+\,-\,-)$
	$-i_0j_2$	$(-\,+)(+\,+\,-\,-\,+\,+\,-\,-)$
	$i_0j_2i_1$	$(+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$-i_0j_2i_1$	$(-\,+)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$i_0j_2j_1$	$(+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$-i_0j_2j_1$	$(-\,+)(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$i_0j_2i_1j_1$	$(+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_0j_2i_1j_1$	$(-\,+)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$i_0i_2j_2$	$(+\,-)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)$
	$-i_0i_2j_2$	$(-\,+)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)$
	$i_0i_2j_2i_1$	$(+\,-)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$-i_0i_2j_2i_1$	$(-\,+)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)$
	$i_0i_2j_2j_1$	$(+\,-)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$-i_0i_2j_2j_1$	$(-\,+)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,+\,-\,-)$
	$i_0i_2j_2i_1j_1$	$(+\,-)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$
	$-i_0i_2j_2i_1j_1$	$(-\,+)(+\,-\,+\,-\,+\,-\,+\,-)(+\,+\,-\,-\,+\,+\,-\,-)(+\,-\,+\,-)(+\,+\,-\,-)$

p	2^p	n = $\frac{(p-1)}{2}$	n = $\frac{(p-2)}{2}$	2^{2p}:$(x_1 + x_2)(x_1 + y_0)(x_1 + (y_1)^{2^{n-1}})^n(x_1 + (y_2)^{2^{n-1}})^n$, $y_0 = i_0 = (+-)$, $y_1 = i_1 = (+--+-)$, $y_2 = j_1 = (++--)$	2^{2p-1}:$(x_1 + x_2)(x_1 + (y_1)^{2^{n-1}})^n(x_1 + (y_2)^{2^{n-1}})^n$, $y_1 = i_1 = (+--+-)$, $y_2 = j_1 = (++--)$
1	2	0	–	–	$(x_1 + x_2)$
2	4	–	0	$(x_1 + x_2)(x_1 + y_0)$	–
3	8	1	–	–	$(x_1 + x_2)(x_1 + y_1)(x_1 + y_2)$
4	16	–	1	$(x_1 + x_2)(x_1 + y_0)(x_1 + y_1)(x_1 + y_2)$	–
5	32	2	–	–	$(x_1 + x_2)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)$
6	64	–	2	$(x_1 + x_2)(x_1 + y_0)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)$	–
7	128	3	–	–	$(x_1 + x_2)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)(x_1 + (y_1)^4)(x_1 + (y_2)^4)$
8	256	–	3	$(x_1 + x_2)(x_1 + y_0)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)(x_1 + (y_1)^4)(x_1 + (y_2)^4)$	–
9	512	4	–	–	$(x_1 + x_2)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)(x_1 + (y_1)^4)(x_1 + (y_2)^4)(x_1 + (y_1)^8)(x_1 + (y_2)^8)$
10	1024	–	4	$(x_1 + x_2)(x_1 + y_0)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)(x_1 + (y_1)^4)(x_1 + (y_2)^4)(x_1 + (y_1)^8)(x_1 + (y_2)^8)$	–
...	...	...	...	...	...

Ultimately, the URS, as a binary system, is more fundamental than Clifford algebra, although this algebra is one of the most important systems it generates. In binary encoding, all i's and j's of any order have the same internal representation in the sign pattern. This is because it only uses symbols generated by the binary process, (1, −1), (1, i_1), (1, j_1), (1, i_2), (1, j_2), (1, i_3) ... and their products, instead of creating extra symbols, such as k. The sign sequence associated with i_n and j_n quaternions are constructed of repeated units, respectively of $(+ - + -)$ and $(+ + - -)$, whose length depends on the order in the URS. If we consider i_0 as a complex number, its sign sequence is $(+ -)$ but, in orders of even powers of 2 such as order 64 where complex numbers occur in, the subscript 0 can be renamed to be the same value as the subscript of the quaternion i in the next higher odd power of 2 of the URS so as to prevent the appearance of asymmetry caused by a break in the naming convention. So for instance at Order 64, 2 to the power 6, where a complex number i_0 appears, the subscript of the complex i_0 can be renamed as i_3 which will at the current order of 64 have a sign pattern of length 2 but at the next higher order of 128, the length of the sign pattern of i_3 will be 16, twice the length of i_2 and j_2. This continuation in the value of the subscript naming convention prevents a name break because the complex numbers can always be rewritten as incomplete quaternions. The lack of any process for multiplying sequences of different length or of the same length but different type means that non-identical quaternions concatenate while identical quaternions (up to a sign difference) *multiply* using XOR or XNOR. In Clifford algebra, on the other hand, all i's and j's have unique name identifiers indicated by their subscripts. Non-identical quaternions with different subscripts and quaternions of the same type (either i or j) with different subscripts still concatenate or juxtapose; however, quaternions of different types (either i or j) with the same subscript can multiply to produce another *product* quaternion of a third type (k) with the same subscript, so $i_m j_m = k_m$, $k_m i_m = j_m$ and $j_m k_m = i_m$. In the binary encoding there is no equivalent simple representation for k_m and no new symbol is generated by the URS.

From the perspective of the binary encoding, how can the correct orders of the URS and therefore correct product types (anti-commutative or commutative) be determined if the code looks like an infinite string of plus and minus signs (or 0s and 1s)? Assuming current string demarcation in current binary computing exists to define where one string ends and another string begins, how does the machine know what order of the URS it is computing, what type of product it is computing, and when to include or not include the minus sign in anti-commutative or commutative multiplication? From the sign pattern and the length of the sign pattern the identity of a particular quaternion, a complete quaternion i (with subscript n for i_n) or equivalently an incomplete quaternion i (with subscript 0 for i_0), is known which is always given by the subscript number attached to that particular quaternion i or j. The quaternion subscript number is a numerical label that closes off the membership of the set for a particular set of quaternions i_n, j_n, and $i_n j_n$. Each set of quaternions with their particular subscript number, or identification number, has a unique characteristic sign pattern length. The parity of the power of 2, p, of an order of the URS 2^p determines whether that order has an integral number of quaternion sets (p odd) or has an integral of number of sets plus one-half a quaternion set (i.e. a complex number) (p even).

power of 2, p = $\log_2 2^p$, p ∈ Z^+, p > 0	order of URS, 2^p	length of sign pattern, $\frac{1}{2}(2^p)$	quaternion label for odd power, $\frac{(p-1)}{2}$	quaternion label for even power, $\frac{p}{2}$	i_n	j_n	length of i at order 2^p as a function of p
1	2	1	0 (no quaternions at this order)				
2	4	2		1	1		2
3	8	4	1		1	1	4
4	16	8		2	2		2
5	32	16	2		2	2	8
6	64	32		3	3		2
7	128	64	3		3	3	16
...	...	...	...	...	...	...	..

12.1 Example Calculations

$(-i_0i_2j_2i_1j_1)(-i_0i_2j_2i_1j_1)$	$((-+)(+-+-+-+-+)(++--++--)(+--+-+)(+++--))((-+)(+-+-+-+-+)(+++--++--)(+--+-+)(+++--))$
$(i_0i_2j_2i_0i_2j_2i_1j_1i_1j_1)$	$((-+)(+-+-+-+-+)(++--++--)(+--+-+)(+++--)(-+)(+-+-+-+-+)(+++--++--)(+--+-+)(+++--))$
$(i_0i_0i_2j_2i_2j_2i_1j_1i_1j_1)$	$((-+)(+-+-+-+-+)(++--++--)(-+)(+-+-+-+-+)(+++--++--)(+--+-+)(+++--)(+--+-+)(+++--))$
$-(i_0i_0i_2j_2i_2j_2i_1j_1i_1j_1)$	$-((-+)(-+)(+-+-+-+-+)(+--+-+++--)(+--+-+)(+++--++--)(+--+-+)(+++--)(+--+-+)(+++--))$
$-(-(i_0i_0i_2i_2j_2j_2i_1i_1j_1j_1))$	$-(-((-+)(-+)(+-+-+-+-+)(+--+-+---++)(+--+-+)(+++--++--)(+--+-+)(+++--)(+--+-+)(+++--)))$
$(i_0i_0i_2i_2j_2j_2i_1i_1j_1j_1)$	$((-+)(-+)(+-+-+-+-+)(+--+-+---++)(+--+-+)(+++--++--)(+--+-+)(+++--)(+--+-+)(+++--))$
$((-1)(-1)(-1)(-1)(-1))$	$(--)(-------)(-------)(---)(----)$
-1	$(-----------------------)$
	$(-)$
$(i_2i_1j_1)(j_1)$	$(+--+--+-+)(+--+-)(+++--)(+++--)$
$(i_2i_1j_1j_1)$	$(+-+--+-+-+)(+--+-)(----)$
$(i_2i_1(-1))$	$(---)(+-+-+-+-+)(+-+-)$
$(-1)i_2i_1$	$(-)(+-+-+-+-)(+--+-)$
$-i_2i_1$	$(-+-+-+-+)(+--+-)$
$(i_2i_1j_1)(i_2i_1)$	$((+-+-+-+-)(+--+-)(+++--))((+-+-+-+-)(+-+-))$
$(i_2i_1j_1i_2i_1)$	$(+-+-+-+-+)(+--+-)(+++--)(+-+-+-+-+)(+-+-)$
$-(i_2i_1i_2i_1j_1)$	$-(+-+-+-+-)(+--+-)(+-+-+-+-)(+-+-)(+++--)$
$-(i_2i_2i_1i_1j_1)$	$-(+-+-+-+-+)(+-+-+-+-+)(+-+-)(+-+-)(+++--)$
$(-i_2i_2i_1i_1j_1)$	$(-+-+--+-+)(+--+-+-+-)(+-+-)(+-+-)(++--)$
$(1)(-1)j_1$	$(++++++++)(+--+-)(+-+-)(+++--)$
$(-1)j_1$	$(++++++++)(---)(+--)$
$-j_1$	$(+)(-)(+--)$
	$(-)(+--)$
	$(--++)$

Now that the quaternions are labelled and known from the length of the sign pattern, how would the computer "know" when to take a commutative product or an anti-commutative product since from inside the code all the sign patterns look the same between all i's and j's at a certain order? The answer comes from the fact that only identical quaternions (same type of quaternion, same i or same j and same index number n) really combine as *multiplication*: namely to produce -1. All the other non-identical quaternions combine as concatenation and so never really reduce down to one quaternion. In particular, all i's of the URS have the same sign pattern but different lengths of the same sign pattern are indicated by their different subscript numbers. All j's of the URS have the same sign pattern but different lengths of the same sign pattern are indicated by their different subscript numbers. All i's and all j's that have the same subscript number have the same sign pattern and the same length of the sign pattern. Sequences with different lengths of sign patterns cannot be multiplied. In effect, the only multiplications as such are for (i_n^2) and for (j_n^2). There is no cross-multiplication between i_n and i_m or between j_n and j_m, or even between i_n and j_n. In the coding, there is only concatenation between non-identical quaternions. The coding is thus not the same as the usual i, j, k form of Clifford algebra and quaternions, but rather an effective representation of its results. This is why we cannot reduce the concatenation of i_nj_n to an equivalent k_n. The multiplication for this binary encoding is more faithful to the more directly binary combinations in the rewrite system (in Section 10.1) than the algebraic i, j, k because all non-identical quaternions, that is cross terms, merely concatenate (even i_n's and j_n's of the same quaternion set concatenate and are only connected algebraically by their mutual anti-commutativity) and only identical quaternions truly *multiply* (not concatenate) according to the two separate multiplication rules for i and j. No quaternion is ever copied throughout the entire rewrite system as every quaternion occurs at most only once in the sequence of the orders of the URS.

Because of the novelty and relative difficulty of the method, it has been considered helpful to include a few worked examples. The difficulty arises from the fact that binary encoding does not allow the simple third symbol $k = ij$ of the quaternion system, but only the two symbols i and j along with any products, which, in this case, means just ij. So, we can only accommodate quaternions because this product can be included alongside i and j, using anti-commutativity for ji = -ij. This suggests, incidentally, why an intrinsically binary URS cannot be structured on octonions, which would require additional product terms which binary notation could not accommodate.

Though an individual calculation may appear involved, it is part of a systematic procedure which could be automated, and the + /- binary can be converted for implementation to the more usual 1 / 0 using either the XOR or XNOR gates as shown in the two logic gate tables earlier in this section. Throughout this discussion whenever a multiplication sign, $\times$, appears it indicates that the XOR or XNOR gate is performing one of the identical quaternion (same type of quaternion, i or j and same the subscript: this applies irrespectively to both positive and negative identical quaternions) multiplication rules between identical quaternions within the same order of the URS: $i \times i$, $j \times j$. From a practical consideration, this encoding scheme could form the basis for the construction of a URS calculator and/or URS programming language.

Another aspect that might concern implementation can be seen by taking decimal values for the binary structures of i_n, j_n. We can use the following formulae to calculate these:

$i_n = \Pi_{m=1}^{n}(2^{2^m} + 1)$

$j_n = 3(\Pi_{m=2}^{n}(2^{2^m} + 1))$

This means that they follow the series:

$i_1 = (2^2 + 1)$
$i_2 = (2^2 + 1)(2^4 + 1)$
$i_3 = (2^2 + 1)(2^4 + 1)(2^8 + 1)$
$i_4 = (2^2 + 1)(2^4 + 1)(2^8 + 1)(2^{16} + 1)$
$i_5 = (2^2 + 1)(2^4 + 1)(2^8 + 1)(2^{16} + 1)(2^{32} + 1)$
$j_1 = (2^1 + 1)$
$j_2 = (2^1 + 1)(2^4 + 1)$
$j_3 = (2^1 + 1)(2^4 + 1)(2^8 + 1)$
$j_4 = (2^1 + 1)(2^4 + 1)(2^8 + 1)(2^{16} + 1)$
$j_5 = (2^1 + 1)(2^4 + 1)(2^8 + 1)(2^{16} + 1)(2^{32} + 1)$

Terms of the form $(2^{2^n} + 1)$ have a significance in mathematics as Fermat numbers [29], and it is interesting to see them

appearing at such a fundamental level in Nature's most basic system. Of course, these will quickly grow to very large numbers and make more complex implementations increasingly difficult, possibly suggesting that, even if the essentially simple and P (polynomial) rewrite system is at the basis of the whole of Nature, situations where P is significant may quickly appear to be NP (non-polynomial). One possible procedure is to effectively zip the files concerning the rewrite structure by redefining the self-composed exponential function 2^{2^n} as the exponent 2^n, with possible iterations at higher levels. This exponent, used to 'zip' the rewrite structure into a compressed binary structure, is extracted by taking the logarithm to the base 2 of the self-composed exponential function 2^{2^n}: $2^n = \log_2(2^{2^n})$. The original file structure can be restored, or 'unzipped', by exponentiating the exponent, 2^n to the power of 2: 2^{2^n}. It is quite possible that Nature already performs this kind of process in the Renormalisation Group.

13 Booth's Representation, Modulus 3 and the Universal Rewrite System

There exists another way to introduce the elements of the Universal Rewrite System, i_0, i_1, and j_1 into binary. Let the elements of {0, 1, |1)} where |1) = -1 (Booth's representation), close over the operations $+_3$ and $\times_3$.

$+_3$	0	1	|1)
0	0	1	|1)
1	1	|1)	0
|1)	|1)	0	1

$\times_3$	0	1	|1)
0	0	0	0
1	0	1	|1)
|1)	0	|1)	1

Then the URS elements, i_0, i_1, and j_1, can be rewritten into this modulus 3 ternary notation. The major advantage of this notation is that the symbols 0 and 1 return to their original values. The novelty is the introduction of a third value, -1 = |1). So rewriting + as 1, – as -1 = |1), 0 can now go back to signifying nothing instead of taking the value – as in the strict binary notation (which introduced the possibility of the Fermat number representation for the URS elements). Let i_0 = 1|1), i_1 = 1|1)1|1), j_1 = 11|1)|1), -i_0 = |1)1|1), -i_1 = |1)1|1)1|1), -j_1 = |1)11|1)|1). It needs to be said that the concatenation rules for like and unlike URS elements are independent of any specific notation. Therefore, the concatenation of unlike elements and the combination rules for like elements apply to this modulus 3 notation as well. See next table.

order 2^p	$\pm 1 \times \pm 1$	$\frac{\pm a_n b_n}{a_n b_n}$
order 4	$\pm i_0 \times \pm i_0$	$\frac{\pm a_1 b_0 a_0 b_1}{a_n b_n}$
order 8	$\pm i_1 \times \pm i_1$	$\frac{\pm a_1 b_0 a_0 b_1 a_3 b_2 a_2 b_3}{a_n b_n}$
order 8	$\pm j_1 \times \pm j_1$	$\frac{\pm a_2 b_0 a_3 b_1 a_0 b_2 a_1 b_3}{a_n b_n}$
order 32	$\pm i_2 \times \pm i_2$	$\frac{\pm a_1 b_0 a_0 b_1 a_3 b_2 a_2 b_3 a_5 b_4 a_4 b_5 a_7 b_6 a_6 b_7}{a_n b_n}$
order 32	$\pm j_2 \times \pm j_2$	$\frac{\pm a_2 b_0 a_3 b_1 a_0 b_2 a_1 b_3 a_6 b_4 a_7 b_5 a_4 b_6 a_5 b_7}{a_n b_n}$

p	2^p	n = $\frac{(p-1)}{2}$	n = $\frac{(p-2)}{2}$	2^{2p}:$(x_1 + x_2)(x_1 + y_0)(x_1 + (y_1)^{2^{n-1}})^n(x_1 + (y_2)^{2^{n-1}})^n$, y_0 = i_0 = (+ –), y_1 = i_1 = (+ – + –), y_2 = j_1 = (+ + – –)	2^{2p-1}:$(x_1 + x_2)(x_1 + (y_1)^{2^{n-1}})^n(x_1 + (y_2)^{2^{n-1}})^n$, y_1 = i_1 = (+ – + –), y_2 = j_1 = (+ + – –)
1	2	–	–	–	$(x_1 + x_2)$
2	4	–	0	$(x_1 + x_2)(x_1 + y_0)$	–
3	8	1	–	–	$(x_1 + x_2)(x_1 + y_1)(x_1 + y_2)$
4	16	–	1	$(x_1 + x_2)(x_1 + y_0)(x_1 + y_1)(x_1 + y_2)$	–
5	32	2	–	–	$(x_1 + x_2)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)$
6	64	–	2	$(x_1 + x_2)(x_1 + y_0)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)$	–
7	128	3	–	–	$(x_1 + x_2)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)(x_1 + (y_1)^4)(x_1 + (y_2)^4)$
8	256	–	3	$(x_1 + x_2)(x_1 + y_0)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)(x_1 + (y_1)^4)(x_1 + (y_2)^4)$	–
9	512	4	–	–	$(x_1 + x_2)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)(x_1 + (y_1)^4)(x_1 + (y_2)^4)(x_1 + (y_1)^8)(x_1 + (y_2)^8)$
10	1024	–	4	$(x_1 + x_2)(x_1 + y_0)(x_1 + y_1)(x_1 + y_2)(x_1 + (y_1)^2)(x_1 + (y_2)^2)(x_1 + (y_1)^4)(x_1 + (y_2)^4)(x_1 + (y_1)^8)(x_1 + (y_2)^8)$	–
...	...	...	...	...	...

We have here rewritten the Universal Rewrite System's algebraic elements using the Booth's notation for -1 [3], as a modulus 3 logic system (Section 8) using the sign representations for the imaginary elements i_0, i_1, and j_1 respecting the concatenation of unlike terms and the XOR/XNOR operation when like terms concatenate (Section 12) in the variable regular expression. This variable expression (now not regular) is split into even, $(x_1 + x_2)(x_1 + y_0)(x_1 + (y_1)^{2^{n-1}})^n(x_1 + (y_2)^{2^{n-1}})^n$, and odd, $(x_1 + x_2)(x_1 + (y_1)^{2^{n-1}})^n(x_1 + (y_2)^{2^{n-1}})^n$ powers of 2. If the modulus 3 notation values for i_0, i_1, and j_1 are substituted into the variables y_0, y_1, and y_2, located in the even variable expression and the odd variable expression respectively, then the entire Universal Rewrite system, when expanded to the desired order, can be compiled or interpreted.

14 The Universal Rewrite System and Quantum Computing

The Universal Rewrite System is quintessentially natural quantum computing, and so can serve as a model for quantum computing technological systems. The URS closes at order 64, making the Dirac algebra at order 64 the basis of Hilbert space, and as such the wave-function solution of the Dirac equation becomes the physical hardware for quantum computing systems. The following table provides the connection between quantum computing and the binary encoding of the URS. The binary encoding of the URS, with Booth's representation of negative one as a sequence of 1's, can be directly transferred into qubits.

p	2^p	$n = \frac{[p-1]}{4}$	$n = \frac{p}{2}$	2^{2p}:$(x_1 + x_2)(x_1 + y_0)(x_1 + (y_1)^{2^{n-1}})^n (x_1 + (y_2)^{2^{n-1}})^n$, $y_0 = \{i_1 = (+ - + -), i_2 = (i_1)^2, i_3 = (i_1)^4, i_4 = (i_1)^8, i_5 = (i_1)^{16}, ...\}$, $y_1 = i_1 = (+ - + -)$, $y_2 = j_1 = (+ + - -)$	2^{2p-1}:$(x_1 + x_2)(x_1 + (y_1)^{2^{n-1}})^n (x_1 + (y_2)^{2^{n-1}})^n$, $y_1 = i_1 = (+ - + -)$, $y_2 = j_1 = (+ + - -)$
1	2	–	–	–	$(x_1 + x_2)$
2	4	–	1	$(x_1 + x_2)(x_1 + i_1)$	–
3	8	1	–	–	$(x_1 + x_2)(x_1 + i_1)(x_1 + j_1)$
4	16	–	2	$(x_1 + x_2)(x_1 + i_2)(x_1 + i_1)(x_1 + j_1)$	–
5	32	2	–	–	$(x_1 + x_2)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$
6	64	–	3	$(x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	–
7	128	3	–	–	$(x_1 + x_2)(x_1 + i_3)(x_1 + j_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$
8	256	–	4	$(x_1 + x_2)(x_1 + i_4)(x_1 + i_3)(x_1 + j_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	–
9	512	4	–	–	$(x_1 + x_2)(x_1 + i_4)(x_1 + j_4)(x_1 + i_3)(x_1 + j_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$
10	1024	–	5	$(x_1 + x_2)(x_1 + i_5)(x_1 + i_4)(x_1 + j_4)(x_1 + i_3)(x_1 + j_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	–
...	...	...	...	...	...

The most general expression for the URS in Hilbert Space in relation to the Bloch Sphere, is: Here $(1,\ i_n)$ are orthogonal, therefore each $(1,\ i_n)$ forms a plane, and since there are an infinite number of square roots of –1 there exist also an infinite number of planes. The angle of rotation of any plane is determined by the momentum component of the Ψ_{k-3}. Conventionally, Ψ_{k-3} does not have a unique rotation, but this method has uniqueness built in.

$$\Pi_{n=4}^{\infty}(1 + i_n)\Psi_n\Sigma_{k=4}^{k=p}(\Pi_{n=4}^{n=k}i_n)\Psi_{k-3}$$

$$(1 + i_4)\Psi_4(i_4\Psi_1)$$
$$(1 + i_5)(1 + i_4)\Psi_5(i_5\Psi_2 + i_4\Psi_1)$$
$$(1 + i_6)(1 + i_5)(1 + i_4)\Psi_6(i_6\Psi_3 + i_5\Psi_2 + i_4\Psi_1)$$
$$(1 + i_7)(1 + i_6)(1 + i_5)(1 + i_4)\Psi_7(i_7\Psi_4 + i_6\Psi_3 + i_5\Psi_2 + i_4\Psi_1)$$
$$(1 + i_8)(1 + i_7)(1 + i_6)(1 + i_5)(1 + i_4)\Psi_8(i_8\Psi_5 + i_7\Psi_4 + i_6\Psi_3 + i_5\Psi_2 + i_4\Psi_1)$$
$$...$$

as shown in the following table:

p	2^p	i_n	Ψ_n	Ψ_{k-3}	$\Pi_{n=4}^{\infty}(1, i_n)\Psi_n$	$\Sigma_{k=4}^{k=p}(\Pi_{n=4}^{n=k}i_n)\Psi_{k-3}$
8	256	i_4	$\Psi_4 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$\Psi_1 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$(1 + i_4)\Psi_4$	$i_4\Psi_1$
10	1024	i_5	$\Psi_5 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$\Psi_2 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$(1 + i_5)(1 + i_4)\Psi_5$	$i_5\Psi_2 + i_4\Psi_1$
12	1024	i_6	$\Psi_6 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$\Psi_3 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$(1 + i_6)(1 + i_5)(1 + i_4)\Psi_6$	$i_6\Psi_3 + i_5\Psi_2 + i_4\Psi_1$
14	1024	i_7	$\Psi_7 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$\Psi_4 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$(1 + i_7)(1 + i_6)(1 + i_5)(1 + i_4)\Psi_7$	$i_7\Psi_4 + i_6\Psi_3 + i_5\Psi_2 + i_4\Psi_1$
16	1024	i_8	$\Psi_8 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$\Psi_5 = (x_1 + x_2)(x_1 + i_3)(x_1 + i_2)(x_1 + j_2)(x_1 + i_1)(x_1 + j_1)$	$(1 + i_8)(1 + i_7)(1 + i_6)(1 + i_5)(1 + i_4)\Psi_8$	$i_8\Psi_5 + i_7\Psi_4 + i_6\Psi_3 + i_5\Psi_2 + i_4\Psi_1$
...	...	...	...	...	...	...

where p is an even power of 2, of the URS, such that $p \geq 8$.

The angle or direction between Ψ_n and Ψ_{k-3} must always be nonzero in order for Pauli exclusion to be nonzero. In other words since Ψ_n and Ψ_{k-3} share the same internal algebraic 'hardware' shown in the table, the coefficients for E, $\mathbf{p}$, m must be unique (with a unique index number) that underlies the no cloning theorem of quantum computing. So no cloning is inbuilt into the URS and can be stated and shown to be reflected in the unique labels of the incomplete quaternions i_n: $\forall\Psi_{k-3}\exists!i_n$. For all distinct wave functions Ψ_{k-3}, there exists a unique i_n coefficient ensuring nonzero Pauli exclusion between any distinct wave-functions, Ψ_n and Ψ_{k-3}, and in the case of the binary encoding in the URS, the length of the sign pattern is unique for every i_n, unlike in the conventional formalism for quantum computing.

15 Summary

The Universal Rewrite System seems to have been a major success in starting from the concept of totality zero and leading to a series of zero alphabets which underlie the fundamental concepts of mathematics and physics. In mathematics it produces a version of integers and discreteness, and generates an infinitely expanding Clifford algebra[1],[2],[3]. In physics, the first four alphabets can be seen to represent, both physically and mathematically, the four fundamental parameters mass, time, charge and space, which have been taken in earlier papers to represent the entire basis of physics. This, in addition, is a closed system because the four parameters, from their complementary physical properties and from their Klein-4 group representation, produce another zero totality. At the same time the tensor product of the algebraic form of the four alphabets is a 64-part algebra, and group, recognisable as that used in the Dirac equation, the one related to the fundamental, fermionic, state.

Using the five generators of this algebra in any of the twelve simultaneously available forms within the 64-part Dirac group gives us a form of the fermionic state which has the special property of being nilpotent or a square root of zero. This object when created from a universal totality of zero leaves a complementary vacuum state, which is its negative and represents the rest of the universe for that state. Since the fermionic state is nilpotent, both the combination (or product) state between created fermion and resulting vacuum or rest of the universe, and the superposition (or addition) state between these two are always zero.

Since nature appears to be systematic in generating the ultimate physical state and its component parameters by a deterministic process, it seems probable that a computerised process could simulate the entire development, and the Universal Rewrite System was created with this object in view. The principle having been established, the next stage required making this process a practicable and useful one. In this paper, we have established that this is possible using the two Variable Regular Expressions. Here, we have a method of using a finite variable expression to represent a potentially infinite number of symbols, as occurs in natural processes. Our first step is to transcribe the Universal Rewrite System into a Regular Expression. Significantly, we don't need to start the universe − the function gives us everything from nothing all the time. It suggests that the universe is eternal, without beginning and without end, and nonfinite in both space and time.

The repetitiousness in the Variable Regular Expression ultimately explains why the universe repeats the same kind of process at every level. In physics we have the Renormalisation Group, but, in more general terms we have similar structures for the atom, cell, planetary system and galaxy, and the same 64-part algebra for both the fermion and for DNA.

A significant finding in this paper is that the whole of natural processes can be simulated using only a two-state machine to automate the language. This can be done in two ways, either as a deterministic finite automaton or as a nondeterministic one, in a way that is compatible with the theory of computational intelligence of finite state machines. The two processes are different but have the same outcome when applied to something like the Clifford algebra of the Universal Rewrite System. The Regular Expression, as shown in this paper, can also be expressed, as we have seen, in Boundary Logic, Boundary Mathematics, Cellular Automata and Category Theory[25]. In physical theory, the five-fold operator of the Dirac equation can be restructured as a 32 × 32 matrix, offering the possibilities of immediate computation leading to the spectrum of fundamental particles. Encoding the Universal Rewrite System naturally into binary will be an important development for many practical applications. This is accomplished in sections 12 and 13 with a complete binary specification for all the URS elements up to order 64 and beyond. It is seen that the URS is a binary process more fundamental than the algebra it creates. Practical applications should be immediately possible.

References

[1] P. Rowlands *Zero to Infinity: The Foundations of Physics*, World Scientific, Singapore, 2007

[2] P. Rowlands *The Foundations of Physical Law*, World Scientific, Singapore, 2014

[3] B. Diaz and P. Rowlands The infinite square roots of −1 *International Journal of Computing Anticipatory Systems* 2006 19 229-235

[4] S. Rowlands and P. Rowlands A universal rewrite system adapted for formal language theory, classical and quantum computing, *Journal of Physics: Conference Series*, 2197 (1), 012024, 2022

[5] S. Rowlands and P. Rowlands The Universal Rewrite System, in B J Gabrys and D Sahoo (eds), Universum, 2023, 401-426

[6] S. Rowlands and P. Rowlands The universal rewrite system coded, *Journal of Physics: Conference Series*, 2197(1):012023, March 2022

[7] P. Rowlands and B. Diaz. A universal alphabet and rewrite system, arXiv: cs.OH/0209026

[8] P. Rowlands and B. Diaz. A computational path to the nilpotent Dirac equation, *International Journal of Computing Anticipatory Systems*, 16, 203-18, 2005

[9] P. Rowlands and B. Diaz. The infinite square roots of 1, *International Journal of Computing Anticipatory Systems*, 19, 229-235, 2006

[10] V. Hill and P. Rowlands Natures code I, *AIP Conference Proceedings*, 1051, 117-126, 2008

[11] P. Rowlands and V. Hill A mathematical representation of the genetic code, in R. Amoroso, L. Kauffman and P. Rowlands (eds.), *Unified Field Mechanics Natural Science Beyond the Veil of Spacetime Proceedings of the IX Symposium Honoring Noted French Mathematical Physicist Jean-Pierre Vigier*, World Scientific, 2015, 553-559

[12] P. Rowlands Cellular Automata and the Foundations of Physics, SIPS, 2022, 16, 57-68

[13] V. Hill and P. Rowlands Natures Fundamental Symmetry Breaking, *International Journal of Computing Anticipatory Systems*, 25, 144-159, 2010

[14] V. Hill and P. Rowlands The Numbers of Natures Code, *International Journal of Computing Anticipatory Systems*, 25, 160-175, 2010

[15] P. Marcer and P. Rowlands The Logic of Self-Organising Systems, *AAAI Technical Reports* 2010-08-020

[16] P. Marcer and P. Rowlands A Computational Unification of Scientific Law: Spelling out a Universal Semantics for Physical Reality, in R. Amoroso, L. Kauffman and P. Rowlands (eds.), *The Physics of Reality Space, Time, Matter, Cosmos*, World Scientific, 2013, 50-59

[17] P. Marcer, P. Rowlands and W. Schempp Self-organised computational rewrite language L predicating an optimal thermodynamic cosmic birth-order automorphic evolution of intelligent life, and consciousness, as natures IT, *Journal of Physics: Conference Series*, 1251 (1), 012032, 2019

[18] S. Kleene *Introduction to MetaMathematics*, Ishi Press 2009

[19] Z. Manna *Mathematical Theory of Computation* (McGraw-Hill) 1974

[20] O. Grumberg, O. Kupferman, and S. Sheinvald Variable Automata over Infinite Alphabets 2010 in Dediu A-H, Fernau H and Martan-Vide C (eds) LATA 2010 LNCS 6031 561-572

[21] M. Sipser *Introduction to the Theory of Computation*, Learning 2022

[22] G. Spencer Brown *Laws of Form* The Julian Press 1972

[23] W. Bricken *Iconic Arithmetic* Unary Press 2019

[24] K. Mainzer and L. Chua *The Universe as Automaton: From Simplicity and Symmetry to Complexity: 1* Springer 2012

[25] S. Rowlands and P. Rowlands Category theory applied to the Klein-4 parameter group and the universal rewrite system, ANPA44, 2023 (to be published)

[26] Shawn Davis, Representing Propositional Logic Connectives with Modular Polynomials article

[27] L. H. Kauffman; personal correspondence

[28] L. H. Kauffman, Laws of Form - A Survey of Ideas, in *Laws of Form - a fiftieth anniversary*. Papers related to the conference (LoF50) held at the University of Liverpool, Liverpool, August 9?10, 2019. Edited by Louis H. Kauffman, Fred Cummins, Randolph Dible, Leon Conrad, Graham Ellsbury, Andrew Crompton and Florian Grote, Ser. Knots Everything, 72 World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, [2023], ©2023. pp. 1-88

[29] M. A. Houlden; personal communication

Acknowledgement. We are grateful to Sarai Sheinvald and Orna Kupferman for clarification of various points raised in their paper.