PERSPEX MACHINE VII: THE UNIVERSAL PERSPEX MACHINE

COPYRIGHT

Perspex Machine VII: The Universal Perspex Machine

James A.D.W. Anderson*
Computer Science, The University of Reading, England

Abstract

The perspex machine arose from the unification of projective geometry with the Turing machine. It uses a total arithmetic, called *transreal arithmetic,* that contains real arithmetic and allows division by zero. Transreal arithmetic is redefined here. The new arithmetic has both a positive and a negative *infinity* which lie at the extremes of the number line, and a number *nullity* that lies off the number line. We prove that nullity, 0/0, is a number. Hence a number may have one of four signs: negative, zero, positive, or nullity. It is, therefore, impossible to encode the sign of a number in one bit, as floating-point arithmetic attempts to do, resulting in the difficulty of having both positive and negative zeros and NaNs. Transrational arithmetic is consistent with Cantor arithmetic. In an extension to real arithmetic, the product of zero, an infinity, or nullity with its reciprocal is nullity, not unity. This avoids the usual contradictions that follow from allowing division by zero. Transreal arithmetic has a fixed algebraic structure and does not admit options as IEEE, floating-point arithmetic does. Most significantly, nullity has a simple semantics that is related to zero. Zero means "no value" and nullity means "no information." We argue that nullity is as useful to a manufactured computer as zero is to a human computer.

The perspex machine is intended to offer one solution to the mind-body problem by showing how the computable aspects of mind and, perhaps, the whole of mind relates to the geometrical aspects of body and, perhaps, the whole of body. We review some of Turing's writings and show that he held the view that his machine has spatial properties. In particular, that it has the property of being a 7D lattice of compact spaces. Thus, we read Turing as believing that his machine relates computation to geometrical bodies.

We simplify the perspex machine by substituting an augmented Euclidean geometry for projective geometry. This leads to a general-linear perspex-machine which is very much easier to program than the original perspex-machine. We then show how to map the whole of perspex space into a unit cube. This allows us to construct a fractal of perspex machines with the cardinality of a real-numbered line or space. This fractal is the universal perspex machine. It can solve, in unit time, the halting problem for itself and for all perspex machines instantiated in real-numbered space, including all Turing machines. We cite an experiment that has been proposed to test the physical reality of the perspex machine's model of time, but we make no claim that the physical universe works this way or that it has the cardinality of the perspex machine. We leave it that the perspex machine provides an upper bound on the computational properties of physical things, including manufactured computers and biological organisms, that have a cardinality no greater than the real-number line.

Keywords: transreal arithmetic, universal perspex machine, universal Turing machine, mind-body problem.

1 Introduction

The *perspex machine*[4-7,13,15] arose from the unification[4,5] of projective geometry with the Turing machine. The unification was carried out, in effect, by identifying the parts of a Turing machine with geometrical objects, and the operations of the Turing machine with geometrical transformations. Only integer co-ordinates were used to model the Turing machine, but the use of homogeneous co-ordinates in the projective geometry raises the question of how to deal with division by zero in an arbitrary transformation. Division by zero is allowed in *transrational arithmetic*[4] and in its generalisation to *transreal*[6] arithmetic. These arithmetics are redefined here in a way that both extends and simplifies the them. Transrational arithmetic now has three *strictly transreal* fractions: *positive infinity,* $\infty \equiv k/0 \equiv 1/0$, *negative infinity,* $-\infty \equiv -k/0 \equiv -1/0$, with k a non-zero integer; and *nullity,* $\Phi \equiv 0/0$. These fractions are numbers for the historically acceptable reason that they arise as solutions to an equation.[4] For example, in a right triangle with unit hypotenuse we have $\tan(\pi/2) \equiv 1/0 \equiv \infty$. Dilatating the triangle by a positive factor k we have $\tan(\pi/2) \equiv k/0 \equiv 1/0$. Similarly,

* J.A.D.W.Anderson@reading.ac.uk, http://www.reading.ac.uk/~sssander, http://www.bookofparagon.com
Computer Science, The University of Reading, Reading, Berkshire, England, RG6 6AY.

$\tan(3\pi/2) \equiv -k/0 \equiv -1/0 \equiv -\infty$. Dilatating by a zero factor k we have $\tan(\pi/2) \equiv k/0 \equiv 0/0 \equiv \Phi$. It is astonishing that nullity has formerly been regarded as an undefined object, rather than being a number, when it arises as the solution to such a well known equation describing the ratio of two sides of a triangle. In this respect, the IEEE standard[12] which calls $0/0$ NaN (Not a Number) is badly motivated, because NaN is evidently a number. That standard also allows a range of options for dealing with infinities and NaN which is not appropriate to the development of a fixed algebraic structure like the *transarithmetics*. In Section 2 we give a derivation of transrational arithmetic which both extends and simplifies the earlier arithmetics.[4,6] In particular, the new arithmetic is consistent with Cantor arithmetic[16] and leads to a very simple semantics for nullity that is related to zero. Zero means "no value" and nullity means "no information."

In Section 3 we review some of Turing's writings[17,19,20] and show that he held the view that his machine has spatial properties. In particular, that it is a 7D lattice of compact spaces. In other words, we argue that Turing believed his machine relates computation to geometrical bodies and is not purely an abstract machine as some commentators claim.

This view is put more strongly in our *perspex thesis* which claims that the relationship between mind and body can be understood in terms of a perspex machine:

> *The perspex machine can simulate all physical things, including mind, to arbitrary precision and, conversely, all physical things, including mind, instantiate a perspex machine.*

The perspex machine can be understood as the structure and operation of a 4D *program-space* of 4×4 matrices, or *perspexes*.[5] Mathematically, the perspex is simultaneously: a shape (simplex), collections of which can tessellate all geometrical bodies that occur in its space; a point-wise, general-linear motion, collections of which can compose all motions in its space; a computer instruction, collections of which can execute all Turing computable programs and all Turing incomputable programs with cardinality no greater than the real number line; and the perspex is an artificial neuron which can do all of the preceding things. By hypothesis, all of the operations of a concrete perspex-machine are physical operations on physical things which may refer simultaneously to abstract mathematical objects and to physical objects. On the materialistic assumption, everything that exists, including mind, is physical so, by hypothesis, the perspex machine has the power to simulate a mind arbitrarily closely. Thus, the perspex thesis offers a solution to the mind-body problem by being a physical thing that implements an abstract, mathematical machine that simulates the operations of any mind arbitrarily closely. Furthermore, if a perspex machine has a mind of its own, it is necessarily true that it physically carries out the operations of its own mind exactly.

The perspex thesis is not simply a linking hypothesis that justifies discussion of mind as a physical process.[14] It provides a simple model of the relationship that allows mathematical models of mental phenomena to be constructed[3,6] and, we suppose, to be put to empirical, instrumental tests. It also offers a solution to the other-minds problem. Perspex spaces can be transformed smoothly one into another[7] so, in principal, any mind and body can be transformed smoothly into another. For example, John Smith at time zero, S_0, and Jane Doe at time zero, D_0, may be blended into one person, P_t, at time t, as $P_t = M_t + (1-t)S_0 + tD_0$ for real t increasing from 0 to 1. We suppose that the additional memory, M_t, allows the blended person, P_t, to remember some, if not all, of the transition. Recall of this memory may be delayed until $t > 1$ so that the blended person is not changed by the additional memory during the transition. If the two end points of the transition are remembered then P_t knows what it is like to be the actual bodies and minds S_0 and D_0 and, therefore, has the experience to settle the other-minds problem for these two individuals. More weakly, if any two instants of the transition are remembered then P_t knows what it is like to be some other minds and bodies and, therefore, has the experience that other minds and bodies exist. It remains an open question whether P_t has the cognitive ability to understand the other-minds problem and declare a solution to it. If John Smith is a cat and Jane Doe is a goldfish it is unlikely that any P_t can voice an opinion on the other minds problem, though the intermediate animal might be tested to see if it has cat-like and/or goldfish like abilities. Only in the case that there is no recall of the transition does the blending fail to yield any information about other minds. Whilst it is not feasible to blend biological organisms, it is practical to blend robots. If we have a pool of robots that have adapted to the world in some way then we may produce blends of them and put their fitness to the test. This provides an alternative to genetic algorithms.

The perspex machine can also be understood as a 20D space composed of a 4D program-space and $4 \times 4 = 16$ dimensions corresponding to a perspex. However, this space is not static. The model of time within the perspex machine[5] allows control to visit different parts of the 4D spacetime before it collapses into a solution.[6] In this respect the perspex machine has some commonality with modern theories of physics, along with their conundrums for causality.

Whilst we may adopt a pragmatic solution to the problems of causality in any particular perspex machine, it is important that the abstract perspex-machine has sufficient flexibility to adopt whatever causality our physical universe has. Indeed, one would want to modify the abstract perspex-machine so that it operates in a way that is identical to the physical universe. This may involve introducing a variety of computational objects. The current adoption of the perspex, as a monad, is a philosophical and theoretical simplification, not a claim to having specified a universal, physical entity.

In Section 4 we simplify the perspex machine by substituting an augmented Euclidean geometry for projective geometry. This leads to a general-linear perspex-machine which is very much easier to program than the original perspex-machine. We then show how to map the whole of perspex space into a unit cube. This allows us to construct a fractal of perspex machines with the cardinality of a real-numbered line and space. This fractal is the universal perspex machine. It can solve, in unit time, the halting problem for itself and for all perspex machines instantiated in real-numbered space, including all Turing machines. It can implement Turing's choice[17] and oracle machines.[19] We also cite a proposed experiment[6] to test the physical reality of the perspex machine's model of time, but we make no claim that the physical universe has the properties of the perspex machine. We leave it that the perspex machine provides an upper bound on the computational properties of any physical things, including manufactured computers and biological organisms, that have a cardinality no greater than the real-number line. This view is consistent with some other views of the physical (un)realisability of super-Turing computers.[21]

2 Transrational Arithmetic

In order to derive transrational arithmetic we make three changes to a standard, constructive derivation[8] of rational arithmetic, though we also accommodate the consequences of these changes. Firstly, we define the canonical form of transrational numbers so that the denominator is non-negative, rather than positive. Secondly, we add a clause to the greater-than and less-than operators so that they can compare a positive and a negative infinity. Thirdly, we add a clause to the addition operator to handle the addition of infinities in a way that is consistent with Cantor arithmetic. All of the other definitions and operations adopted here occur as standard formulae in some derivation of the rational numbers, but this is not to say that all treatments of the rational numbers carry over to transrational numbers. Allowing division by zero introduce some freedom into the development of arithmetics that contain rational arithmetic as a sub-arithmetic. Hence, different selections of definitions that are equivalent in the rational case may lead to different super-arithmetics. The arithmetic developed here is designed to be particularly suited to Euclidean geometries.

Firstly, we adopt the standard *sign* function for use in the development of transrational arithmetic; but, for completeness, we extend it to return a numerical value denoting the sign of nullity. Note that *sign* takes on four values which can be encoded exactly in two bits, not one bit as in floating-point arithmetic.[12] Compressing the sign into one bit raises the difficulty of having objects $+0$, -0, $+\mathrm{NaN}$, and $-\mathrm{NaN}$ where only 0 and NaN can be justified algebraically.

$$\mathrm{sgn}(a) = \begin{cases} 1 & : a > 0; \\ 0 & : a = 0; \\ -1 & : a < 0; \\ \Phi & : a = \Phi. \end{cases} \qquad \text{(Eqn. 1)}$$

We allow all fractions of integers, n_i/d_i, but reduce them to a canonical form which is the usual form for rational numbers, extended to deal with fractions that have a zero denominator.

We compute the highest, common, signed denominator, k, of non-zero numerator n_1 and non-zero denominator d_1 as follows. We compute c as the greatest, common, positive divisor of n_1, d_1. We then compute $k = \mathrm{sgn}(d_1)c$ and find the unique n_2, d_2 such that $n_1 = kn_2$ and $d_1 = kd_2$. Then:

$$\text{the canonical form of a fraction of arbitrary integers } n_1/d_1 \text{ is } \begin{cases} 0/1 & : n_1 = 0, d_1 \neq 0; \\ \mathrm{sgn}(n_1)/0 & : d_1 = 0; \\ n_2/d_2 & : n_1, d_1 \neq 0. \end{cases} \qquad \text{(Eqn. 2)}$$

For all fractions of integers in canonical form we define equality, ordering, and the arithmetical operations as follows. Note that the result of the arithmetical operations, (Eqn. 6) – (Eqn. 9), must be reduced to canonical form.

$$n_1/d_1 = n_2/d_2 \text{ iff } n_1 = n_2 \;\&\; d_1 = d_2 . \tag{Eqn. 3}$$

$$n_1/d_1 > n_2/d_2 \text{ iff } \begin{cases} n_1/d_1 = \infty \;\&\; n_2/d_2 = -\infty \text{ or} \\ n_1 d_2 > n_2 d_1 . \end{cases} \tag{Eqn. 4}$$

$$n_1/d_1 < n_2/d_2 \text{ iff } \begin{cases} n_1/d_1 = -\infty \;\&\; n_2/d_2 = \infty \text{ or} \\ n_1 d_2 < n_2 d_1 . \end{cases} \tag{Eqn. 5}$$

$$n_1/d_1 + n_2/d_2 = \begin{cases} (n_1 + n_2)/0 \;:\; d_1 = d_2 = 0; \\ (n_1 d_2 + n_2 d_1)/(d_1 d_2) \;:\; \text{otherwise.} \end{cases} \tag{Eqn. 6}$$

$$-(n_1/d_1) = (-n_1)/d_1 . \tag{Eqn. 7}$$

$$(n_1/d_1) \times (n_2/d_2) = n_1 n_2/d_1 d_2 . \tag{Eqn. 8}$$

$$(n_1/d_1)^{-1} = (d_1/n_1) . \tag{Eqn. 9}$$

From (Eqn. 3) we have that all of the transrational numbers, $Q^* = Q \cup \{-\infty, \infty, \Phi\}$, are distinct over equality. In particular, $\Phi = \Phi$, that is, $0/0 = 0/0$, whereas the IEEE standard[12] has $\text{NaN} \neq \text{NaN}$, that is, $0/0 \neq 0/0$. This difference arises because the IEEE standard assumes that $0/0$ is not a number and so has an undefined semantics, whereas the tangent argument given in our introduction proves that $0/0$ is a number. This number has a well-defined semantics, as we are now demonstrating.

In the following tables: $q_i = n_i/d_i$ with $n_i, d_i > 0$, $-\infty = -1/0$, $0 = 0/1$, $\infty = 1/0$, and $\Phi = 0/0$. In Table 1: the element *T* means unconditionally true, *F* means unconditionally false, and *C* means conditionally true or false, identically as it is true or false in rational arithmetic. In Table 2: and Table 3: the number q_3 has its magnitude calculated exactly as in rational arithmetic, and the alternate sign in $\pm q_3$ denotes that the sign is fixed exactly as in rational arithmetic.

Note, from Table 1:, that all of the rational (and real) numbers, augmented with $\pm\infty$ are well-ordered on the number line and that Φ is not ordered with respect to any number on the line and, therefore, lies off the number line as shown in Figure 1. This is consistent with complex, quaternion, and octonion arithmetics where numbers off the line are not ordered, but do have a modulus. We now derive the modulus of the strictly transrational numbers. Thus:

$$|\Phi| = \sqrt{\Phi^2} = \sqrt{\Phi} = \Phi ; \tag{Eqn. 10}$$

$$|\infty| = \sqrt{\infty^2} = \sqrt{\infty} = \infty = \sqrt{\infty} = \sqrt{-\infty^2} = |-\infty| . \tag{Eqn. 11}$$

By contrast, the IEEE treatment[12] of ordering and equality is incoherent. In particular, the IEEE definition that $0/0 \neq 0/0$ is not founded on any numerical property, but is motivated by the mistaken view that $0/0$ is not a number.

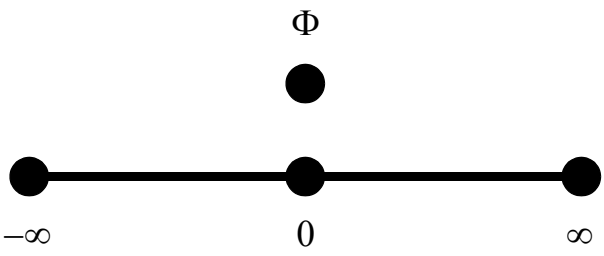

Figure 1: Nullity lies off the real number line augmented with positive and negative infinity.

$n_1/d_1 > n_2/d_2$		$-q_2$	0	q_2	$-\infty$	∞	Φ
		$(-n_2)/d_2$	$0/1$	n_2/d_2	$(-1)/0$	$1/0$	$0/0$
$-q_1$	$(-n_1)/d_1$	C	F	F	T	F	F
0	$0/1$	T	F	F	T	F	F
q_1	n_1/d_1	T	T	C	T	F	F
$-\infty$	$(-1)/0$	F	F	F	F	F	F
∞	$1/0$	T	T	T	T	F	F
Φ	$0/0$	F	F	F	F	F	F

Table 1: Greater Than

$n_1/d_1 + n_2/d_2$		$-q_2$	0	q_2	$-\infty$	∞	Φ
		$(-n_2)/d_2$	$0/1$	n_2/d_2	$(-1)/0$	$1/0$	$0/0$
$-q_1$	$(-n_1)/d_1$	$-q_3$	$-q_1$	$\pm q_3$	$-\infty$	∞	Φ
0	$0/1$	$-q_2$	0	q_2	$-\infty$	∞	Φ
q_1	n_1/d_1	$\pm q_3$	q_1	q_3	$-\infty$	∞	Φ
$-\infty$	$(-1)/0$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	Φ	Φ
∞	$1/0$	∞	∞	∞	Φ	∞	Φ
Φ	$0/0$	Φ	Φ	Φ	Φ	Φ	Φ

Table 2: Addition

$n_1/d_1 \times n_2/d_2$		$-q_2$	0	q_2	$-\infty$	∞	Φ
		$(-n_2)/d_2$	$0/1$	n_2/d_2	$(-1)/0$	$1/0$	$0/0$
$-q_1$	$(-n_1)/d_1$	q_3	0	$-q_3$	∞	$-\infty$	Φ
0	$0/1$	0	0	0	Φ	Φ	Φ
q_1	n_1/d_1	$-q_3$	0	q_3	$-\infty$	∞	Φ
$-\infty$	$(-1)/0$	∞	Φ	$-\infty$	∞	$-\infty$	Φ
∞	$1/0$	$-\infty$	Φ	∞	$-\infty$	∞	Φ
Φ	$0/0$	Φ	Φ	Φ	Φ	Φ	Φ

Table 3: Multiplication

Note, from Table 2:, that $\infty + \infty = \infty$ and, from Table 3:, $q\infty = \infty$, both of which are consistent with Cantor arithmetic (the arithmetic of cardinal numbers).[16] The whole of Table 2: and Table 3: is consistent with IEEE, floating-point arithmetic.[12] In particular, $(-\infty) + (-\infty) = -\infty$ and $\infty - \infty = \Phi$. Thus, transrational arithmetic defines the same arithmetical operations on the strictly transrational numbers as IEEE, floating-point arithmetic does, but has the advantage that ordering is well defined.

Allowing division by zero in augmented versions of the standard arithmetics usually introduces a contradiction arising from the axioms $0 \times n = 0$ and $n \times n^{-1} = 1$ for all numbers n, whence $0 \times 0^{-1} = 0, 1$. However, transrational arithmetic does not introduce this contradiction, because:

$$n \times n^{-1} = \begin{cases} \Phi & : \ n \in \{-\infty, 0, \infty, \Phi\}, \\ 1 & : \ \text{otherwise.} \end{cases} \qquad \text{(Eqn. 12)}$$

Transrational arithmetic is generalised to transreal arithmetic, as usual,[6] by writing an irrational number i as $i/1$. The extension to complex, quaternion, and octonion arithmetics follows immediately, as does the extension to integer and natural arithmetics.

In conclusion, transrational and transreal arithmetics allow division by zero without introducing the usual contradictions attendant on division by zero. The real number line, augmented with negative and positive infinities, is well ordered and nullity, which lies off the number line, is not ordered. This is consistent with all of standard number systems, but is not consistent with IEEE, floating-point arithmetic which is now seen to be incoherent. In particular, floating-point arithmetic uses one sign bit, but real arithmetic has three signs – negative, zero, positive – and transreal arithmetic has four signs – negative, zero, positive, and nullity. If floating-point arithmetic is to account correctly for the number $0/0$ then it must encode four signs, which it could do in exactly two sign bits. It could then be a total, *trans-floating-point* arithmetic with no error states.[15] This would remove the need for exception handling, thereby simplifying floating-point units, and would mean that all floating-point pipelines operate without stalling on exceptions. This might make *transfloat* processors smaller, more reliable, and cheaper than floating-point processors.

3 The Turing Machine

3.1 Spatiotemporal Properties of the Turing Machine

Turing defines the spatiotemporal properties of his machine in a number of places. In,[20] page 439, we read:

> *Strictly speaking there are no such machines [as discrete state machines]. Everything really moves continuously. But there are many kinds of machine which can profitably be thought of as being discrete state machines. For instance in considering the switches of a lighting system it is a convenient fiction that each switch must be definitely on or definitely off. There must be intermediate positions, but for most purposes we can forget about them.*

Thus, Turing accepts a model of physics in which all physical motions are continuous (as may be the case in a quantum physics with a temporal continuum of hidden variables) but he requires that his continuum can be digitised. This implies that the digits are closed and bounded, i.e. compact, spaces. Turing makes this spatial claim explicit in,[17] page 249:

> *If we regard a symbol as literally printed on a square we may suppose that the square is* $0 \le x \le 1$, $0 \le y \le 1$. *The symbol is defined as the set of points in this square, viz. the set occupied by the printer's ink. If these sets are restricted to be measurable, we can define the "distance" between two symbols as the cost of transforming one symbol into the other if the cost of moving unit area of printer's ink unit distance is unity, and there is an infinite supply of ink at* $x = 2$, $y = 0$. *With this topology the symbols form a conditionally compact space.*

Regardless of whether symbols are printed on paper, encoded in electrical voltages, or otherwise instantiated, the restriction that the sets are measurable is required for practical digitisation, but this leads to the conditional conclusion that digitised symbols are compact spaces. Alternatively, if the sets are not measurable then digitisation cannot be defined in

any physically realisable way, but digitised symbols may then have a non-compact topology. Thus, on Turing's reading of physics, which is conditional on the measurability of quantities, symbols are compact spaces. It follows that the whole of the Turing Machine has spatial properties.

Firstly, in,[17] section 1, page 231, we read:

> *The possible behaviour of the machine at any moment is determined by the m-configuration* q_n *and the scanned symbol* $S(r)$. *This pair* q_n, $S(r)$ *will be called the "configuration": thus the configuration determines the possible behaviour of the machine.*

Turing goes on to define the *m*-configuration as a tuple of symbols. We may, therefore, read the above as defining that the possible behaviour of the machine, at one moment, is entirely defined by symbols, and that these symbols are compact spaces. In other words, the possible behaviour of the machine, at an instant, is defined by a tuple, or lattice, of compact spaces. In,[17] page 240, Turing defines a *standard description* in terms of five-tuples. Therefore the machine Turing describes is defined, at an instant, by a 5D lattice of compact spaces. The behaviour of the machine over time relies additionally on a tape. Turing defines that the tape is one dimensional,[17] page 249. As he has earlier defined that the tape head may move left and right relative to the tape it implies that the tape is an orientable, 1D space.

> *I assume then that the computation is carried out on one-dimensional paper, i.e. on a tape divided into squares. I shall also suppose that the number of [different] symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent.*

In,[17] page 232, Turing defines that the continuous motion of his machine may be understood as if it were a discrete motion taking place in units of a *move.*

> *At any stage of the motion of the machine, the complete sequence of all symbols on the tape, and the m-configuration will be said to describe the complete configuration at that stage. The changes of the machine and tape between successive complete configurations will be called moves of the machine.*

Turing illustrates this arrangement with an example,[17] on page 235. As moves are *successive* we may hold that they take place in an orientable time-line, requiring that a temporal dimension be added to the spatial dimensions of the machine.

In summary, the Turing machine is defined by a 5D lattice of compact spaces that is the *m-configuration* of the machine, and a 1D lattice of compact spaces that is the machine's *tape.* Together these form a 6D lattice called the *configuration.* This 6D lattice changes discretely over a 1D, time axis. Thus, a 7D lattice of compact spaces describes all of the *moves* of a Turing Machine. In other words, the Turing Machine is a 7D lattice of compact spaces.

Turing defines that the machine's vocabulary of symbols is finite. He makes no use of his observation that symbols can be transformed one into the other. By contrast, the perspex machine[5] uses an infinite continuum of perspexes as super-symbols and can exploit the transformation of one super-symbol – a Turing computable or incomputable number – into another, and hence the transformation of one operation into another.[7]

It should be noted that some people regard the Turing Machine as being a purely symbolic machine with no spatial properties whatsoever. This view finds no support in Turing's writings[17-20] and is contrary to accepted views of physics. We invite anyone who believes that symbols are devoid of spatiotemporal properties to say how it is that symbols can be recorded and processed without occupying any physical space, and without this processing occurring at any physical time. An appeal to Platonic Ideal Forms fails immediately, because the Forms have spatial properties, such as being a perfect circle, and temporal properties, such as existing forever without decay. In any case, the argument that the existence of a symbolic description extinguishes spatial properties is absurd. The existence of analytical geometry does not deprive Euclidean space of spatial properties, far less does it eliminate space from the physical universe we live in.

3.2 Non-Universal Nature of Turing Computation

Turing[17] defines a machine, U, beginning at page 241, that can emulate all of his *automatic-machines*. However, U cannot emulate his *choice-machines* and, in particular, U cannot emulate his *oracle-machines.* It is an abuse of language

to call U a "universal Turing machine." This common abuse conceals the possibility of constructing an abstract machine, such as the perspex machine, that can emulate all of Turing's machines. Before seeing how this is done, it is useful to consider the behaviour of Turing's machines. See,[17] page 232. In the following extract, Turing refers to his section one. The relevant extract appears as the third quotation in our section immediately above.

> *If at each stage the motion of a machine (in the sense of §1) is completely determined by the configuration, we shall call the machine an "automatic-machine" (or a-machine).*
>
> *For some purposes we might use machines (choice machines or c-machines) whose motion is only partially determined by the configuration (hence the use of "possible" in §1). When such a machine reaches one of these ambiguous configurations, it cannot go on until some arbitrary choice has been made by an external operator. This would be the case if we were dealing with axiomatic systems. In this papers I deal only with automatic machines, and will therefore often omit the prefix a-.*

Thus, choice-machines are non-deterministic, they stall when faced with ambiguity. This kind of Turing machine is restarted by an external agency that lies outside the domain of Turing's theory of computation. By contrast, abstract perspex-machines are deterministic and, therefore, require no external agency. The abstract, universal perspex-machine has cells, each of which is a perspex machine, that can examine and interact with its subordinates. Thus, a cell can be the external agency that restarts another, internal, perspex machine. In this way the restart is part of the theory of perspex computation. Furthermore, the perspex thesis links the abstract perspex-machine to the physical universe so that there is, by hypothesis, nothing in the physical universe that is external to a perspex machine. In other words, there are no agencies external to the perspex machine. In this physical sense, no abuse of language is involved in the phrase, "universal perspex-machine," because the perspex machine can, by hypothesis, carry out all physically possible computations. However, cardinality limits on the universal perspex-machine mean that while it can emulate infinitely many,[16] $\aleph_1$, perspex machines, it cannot emulate them all, $\aleph_2$.

The universal perspex-machine, or something like it, is needed to deal non-arbitrarily with axiomatic systems where there is a choice, at any one step of a proof, of which axiom to invoke next or which new sentence to accept as an axiom.

Turing,[19] page 172-173, defines a special kind of choice-machine, called the oracle-machine, or *o*-machine, that evaluates functions, equivalent to the halting function, that cannot be computed by an automatic-machine and which cannot, therefore, be computed by U.

> *Let us suppose that we are supplied with some unspecified means of solving number-theoretic problems; a kind of oracle as it were. We shall not go any further into the nature of this oracle apart from saying that it cannot be a machine. With the help of the oracle we could form a new kind of machine (call them o-machines), having as one of its fundamental processes that of solving a given number-theoretic problem. More definitely these machines are to behave in this way. The moves of the machine are determined as usual by a [state] table except in the case of moves from a certain internal configuration o. If the machine is in the internal configuration o and if the sequence of symbols marked with l [on the tape] is then the well-formed† formula A, then the machine goes into the internal configuration p or t according as it is true or not that A is dual [halts]. The decision as to which is the case is referred to the oracle.*

Turing is perfectly clear that oracle-machines are not automatic-machines. He continues as follows.

> *These [o-]machines may be described by [state] tables of the same kind as those used for the description of a-machines, there being no entries, however, for the internal configuration o.*

In other words, an automatic-machine, and hence a U machine, has no specification in its state tables of how an oracle works. It simply stalls in the state *o* and waits for the oracle to re-start it. In other words, it is possible to conceive of computations, such as those performed by oracle-machines, that cannot be performed by the, so called, Universal Turing Machine, U.

In a footnote to the second quotation above, Turing writes:

> *† Without real loss of generality we may suppose that A is always well formed.*

It is important to recognise that Turing makes this statement in the context of a paper[19] dealing with logic. Standard logics have no use for badly-formed formulae so Turing feels no loss at their exclusion. However, the loss is grievous. Human computers can process badly formed-formulae, for example, human mathematicians can, and all to often do, correct flaws in mathematical presentations. Yet Turing declares that this form of human computation is beyond the power, even, of an oracle-machine that can solve the halting problem for an automatic-machine. By contrast all symbolic (and super-symbolic) formulae translate into geometrical shapes in the perspex machine, and are processed by it. Thus, the perspex machine is total, or universal, over all formulae, both well-formed and badly-formed. In this sense the perspex machine captures a very important aspect of human computation that the Turing machine misses – the ability to make mistakes, to work out their consequences, and, potentially, to correct them.

3.3 Super-Turing Computation and Turing's Halting Problem

Akl[1] reviews the known kinds of super-Turing computation, though some of his conclusions are reversed by Welch.[21] Akl reviews hypercomputations where the computational steps in a computer speed up. For example, if the speed of computation doubles at each step, the machine can complete infinitely many steps in two units of time.[9] The proof of this is similar to Zeno's arrow paradox. Hence, a hypercomputer can solve Turing's halting problem in a finite time, which may be scaled to unit time. Turing's halting problem requires that $\aleph_0$ computations are performed, where $\aleph_0$ is the cardinality of the integers.[16] However, the perspex machine can lay out computations on a segment of the real number line. This segment has the cardinality of the real-number line[16] itself, $\aleph_1 = 2^{\aleph_0} \gg \aleph_0$. The perspex machine can map a line of computations onto a result in zero time, but the result has not collapsed into a fixed state until it is read from a point at a different instant. So it may take an asymptotically small, but finite, time to access the completed computation. This time may be scaled to unit time. But the perspex machine can do more than solve the halting problem for one Turing machine. The powerset[16] of a set of cardinality $\aleph_0$ has cardinality $\aleph_1$ so the perspex machine can solve, in unit time, the halting problem for the set of all Turing machines. In other words, the perspex machine can partition the set of all Turing machines into halting and non-halting subsets in unit time. This result holds for all of Turing's automatic, choice, and oracle-machines.

Akl also describes a class of computations where a Turing machine capable of performing n steps of a computation in one unit of time fails in computations that require $n+1$ steps. He identifies three classes of these problems, though the first two classes are similar. Firstly, problems where $n+1$ time-varying data must be read from multiple Turing tapes in one unit of time. Secondly, problems where $n+1$ physical variables depend on each other and must be measured, and hence read from multiple Turing tapes, in one unit of time. And, finally, problems where a mathematical constraint holds at all times between $n+1$ data that are present within the Turing Machine. Whilst Turing machines with more and more tapes may be constructed, there is always a problem that is too large for the machine. Thus, Akl concludes, the Turing machine is not a candidate for the title of "universal computer." By contrast, the perspex machine has access to $\aleph_0$ states so it can solve all of the problems reviewed by Akl.[1] We do, in fact, claim that the perspex machine is a universal computer over the domain of physically possible computations up to cardinality $\aleph_1$, and that it does physical things that are not computations.

3.4 Summary

The Turing machine has spatial properties that derive from several sources. Firstly, Turing defines that symbols are compact spaces. Symbols can be transformed one into another, but Turing makes no use of this. Secondly, Turing defines that the machine's tape is a 1D, orientable object. Thirdly, Turing defines that the entire machine, at an instant, is a 5-tuple of symbols or, equivalently, a 5D lattice of compact spaces. Finally, the machine changes state over a 1D, orientable time-line. Therefore the Turing machine is a 7D lattice of compact spaces. This lattice lies within the perspex machine, as demonstrated by the existence of the proof given later of the super-Turing nature of the perspex machine.

Turing defines three kinds of machine in the papers[17-20] cited here: automatic-machines, choice-machines, and oracle-machines. Turing's machine U which is, today, known as the Universal Turing Machine, is universal only over the domain of automatic-machines. It cannot emulate choice-machines or oracle-machines. Both choice-machines and oracle-

machines are non-deterministic, they stall in the presence of ambiguity and must be re-started by an external agency. By contrast, the perspex machine is deterministic and can emulate automatic-machines. It can, however, halt deterministically by executing the perspex halting instruction. Hence, it can emulate a non-deterministic stall by finding the non-determinism and halting. The abstract, universal perspex-machine can also emulate a restart, because it contains many communicating perspex machines which can restart their subordinates. Therefore it can emulate choice-machines. The abstract, universal perspex-machine has sufficient computational power to solve the Turing halting problem for the entire set of Turing machines. Therefore it can emulate the entire set of oracle-machines. Furthermore, the abstract, universal perspex-machine can execute all known forms of hypercomputation – up to cardinality $\aleph_1$. As we shall see next, it can even solve its own halting problem, and the halting problem for any one perspex machine, and the halting problem for all $\aleph_1$ physically realisable perspex-machines, but it cannot solve the halting problem for the entire set of perspex machines with cardinality $\aleph_2$, because it has access to at most $\aleph_1$ states.

The perspex machine can process all formulae, be they well- or badly-formed. In fact, it can process continuous things that are not formulae of any kind. It is so general that it provides a universal model of physics with cardinality no greater than $\aleph_1$.

4 The Universal Perspex-Machine

The universal perspex-machine differs from previous versions of the perspex machine in three important respects. Firstly, the machine uses a general-linear instruction, not a linear one. This makes it easier to operate on individual elements of a perspex by isolating a single element by pre- and post-multiplication by *j*-matrices,[2,15] and by composing a perspex from individual elements via matrix addition. Secondly, the universal perspex-machine has a cellular structure that allows fractal machines to be constructed with cardinality $\aleph_1$. Thirdly, control can jump simultaneously in all spatial axes, not in only two axes as formerly. Hence, jumper[15] instructions do not have to be introduced artificially. We now describe the universal perspex-machine, repeating parts of earlier definitions, so that a complete specification is given here. The reader may still wish to refer to earlier papers for a fuller account of the operation of the machine.[4-7,13,15] We also discuss the perspex halting problem.

4.1 General-Linear Instruction

Firstly, we define the general, and some special, perspexes.

$$\begin{bmatrix} x_1 & y_1 & z_1 & t_1 \\ x_2 & y_2 & z_2 & t_2 \\ x_3 & y_3 & z_3 & t_3 \\ x_4 & y_4 & z_4 & t_4 \end{bmatrix}. \qquad \text{(Eqn. 13)}$$

$$Z = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \qquad \text{(Eqn. 14)}$$

$$H = \begin{bmatrix} \Phi & \Phi & \Phi & \Phi \\ \Phi & \Phi & \Phi & \Phi \\ \Phi & \Phi & \Phi & \Phi \\ \Phi & \Phi & \Phi & \Phi \end{bmatrix}. \qquad \text{(Eqn. 15)}$$

Now we define the operation of the perspex machine.

$$\text{continuum}(a) = \begin{cases} a, a \neq H; \\ Z, a = H. \end{cases} \qquad \text{(Eqn. 16)}$$

$$\begin{aligned} &\text{jump}((\overrightarrow{z+p})_{11}, t) \text{ transfers control form } p\text{:} \\ &\quad \text{to } \left(\left[t_1\ 0\ 0\ t_4\right]^T + p\right) \text{ if } (\overrightarrow{z_{11}+p} < 0); \\ &\quad \text{to } \left(\left[0\ t_2\ 0\ t_4\right]^T + p\right) \text{ if } (\overrightarrow{z_{11}+p} = 0); \\ &\quad \text{to } \left(\left[0\ 0\ t_3\ t_4\right]^T + p\right) \text{ if } (\overrightarrow{z_{11}+p} > 0); \\ &\quad \text{to } \left(\left[t_1\ t_2\ t_3\ t_4\right]^T + p\right) \text{ otherwise.} \end{aligned} \qquad \text{(Eqn. 17)}$$

$$\begin{aligned} &\text{continuum}(\overrightarrow{z+p}) + \sum_i \left(\overrightarrow{x^{(i)}+p^{(i)}}\right)\left(\overrightarrow{y^{(i)}+p^{(i)}}\right) \to (\overrightarrow{z+p}); \\ &\qquad \text{jump}((\overrightarrow{z+p})_{11}, t). \end{aligned} \qquad \text{(Eqn. 18)}$$

The *perspex machine* is a 4D space where every point has transreal co-ordinates. Every point in *perspex space* contains a perspex. A perspex, (Eqn. 13), is a 4×4 matrix of transreal co-ordinates laid out as column vectors x, y, z, t. The perspex machine is started at a point p or at several points, $p^{(i)}$, in space, which may include all of the points in a volume of space. At each point the machine performs the operation given by (Eqn. 18) on the perspex stored at p. The notation $\vec{p}$ denotes the contents of the point p, i.e. the perspex stored at the point p. Thus, $(\overrightarrow{x+p})(\overrightarrow{y+p})$ denotes a multiplication of perspexes stored at locations $x+p$ and $y+p$. The addition of the *continuum* term forces the machine to increment the contents of the point $z+p$. The summation of products, $\sum_i \left(\overrightarrow{x^{(i)}+p^{(i)}}\right)\left(\overrightarrow{y^{(i)}+p^{(i)}}\right) \to (\overrightarrow{z+p})$, denotes that i perspexes may form products and write them, by incrementation, into the common location $z+p$. The function *continuum* maps the halting perspex, H, (Eqn. 15), onto the zero matrix, (Eqn. 14). This manoeuvre prevents the increment being forced to H in every case.[4,7] The *jump* function transfers control from the point p to a, not necessarily distinct, point p' in space. The otherwise clause of (Eqn. 17) is satisfied when $(\overrightarrow{z+p})_{11} = \Phi$. If p' contains the halting perspex, H, then the control thread halts, otherwise it continues at p'. Every point in space contains H by default. Non-null programs are created by instantiating some points with non-halting perspexes and starting the machine at one or more of these points.

Originally the perspex machine employed absolute addressing of data to reflect the fact that the registers in an Unlimited Register Machine[11] or, equivalently, the read-write head of a Turing machine,[17] is at a fixed, absolute position. However, the perspex machine also employed relative addressing of control to reflect the fact that in the Unlimited Register Machine control moves forward, by default, to the adjacent instruction in a program, and in the Turing machine the tape moves relatively left or right by one cell. Whilst this lead to a simple constructive proof that the perspex machine can do everything that the Turing machine can do, the mixture of absolute and relative addressing is arbitrary in terms of the perspex machine itself. The type of addressing can be changed by striking out p in parts of (Eqn. 18). With p in place the formula describes relative addressing, with p struck out it describes absolute addressing. (Eqn. 18) explicitly describes a 4D machine, but fixing one or more rows or columns of a perspex reduces the dimensionality of the machine. As written, the formula involves a matrix multiplication, but other multiplications could be used. For example, previous versions of the perspex machine used a normalised matrix-multiplication to instantiate projective geometry.[5] Previous version also lacked the addition term, making them linear, but not general-linear, perspex machines. Replacing (Eqn. 18) with other

functions or allowing higher dimensional perspexes would radically alter the machine. Thus, an entire family of perspex machines is created by restricting or modifying the formulae given above.

A perspex machine with all absolute addressing is tied to its input and output devices. In order to duplicate a processing thread so that, say, adjacent pixels in an image are processed, it is necessary to transform a program by transforming both the location of its perspexes and their contents. With relative addressing it is necessary to change only the locations of the program. Mixed addressing machines require a correspondingly more complex transformation to relocate them. The point at issue is that replicating and relocating programs is an inherent part of "genetic" algorithms, but relocating a relative program can easily dissociate it from its input and output devices. In order to reduce this sensitivity to position it is possible to interpolate perspexes[7] so that a small movement away from an input or output device, in other words a synaptic gap, has only a small effect on the data transferred. But if this approach is taken, the form of interpolation is a critical part of the machine, in much the same way that chemical concentration gradients and electrical-field decay play a critical role in animal brains. Thus, when trying to admit "genetic" replication one is very soon faced with "physical" design considerations, even when implementing a virtual perspex-machine. This is because the perspex machine is inherently spatial.

4.2 Proof of Super-Turing Computation

The perspex machine is inherently capable of at least one super-Turing operation because the jump part of its instruction, (Eqn. 17), can compare an arbitrary, real number to zero. We showed earlier that the linear perspex-machine can do everything that the Turing Machine can do.[5] We now modify that proof so that it proves that the general-linear perspex-machine can do everything that the Turing Machine can do. In developing this proof we take advantage of *j*-matrices[2,15] to isolate an element of a matrix. We use the addition term in the perspex instruction to compose a matrix from corresponding elements. Thus, the proof stands as an example of the use of the general-linear instruction.

In order to generalise the proof,[5] all we need to show is that the general-linear perspex-instruction can compute the four operations *A, C, G, T* of the Unlimited Register Machine (URM).[5,11] As all of the 4×4 matrices used in the proof are identity in the middle two rows and columns, it is convenient to adopt the short hand of omitting these two rows and columns. We begin by giving the *j*-matrices.

$$J_0 = Z = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}; \qquad J_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}; \qquad J_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}; \qquad J_3 = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{Eqn. 19}$$

4.2.1 Perspex G

Writing Z into the location z zeros the element $\vec{z}_{11}$ as required by the URM instruction $G(z)$. In order to achieve this we first write H into location z so that the addition term has no effect, we then form Z as a product and write it into the location z. Hence, with control passed to successive instructions, it is sufficient to compute:

$$\begin{aligned} &\text{continuum}(\vec{z}) + IH \rightarrow \vec{z}; \qquad \text{jump}(\vec{z}_{11}, t); \\ &\text{continuum}(\vec{z}) + IZ \rightarrow \vec{z}; \qquad \text{jump}(\vec{z}_{11}, t). \end{aligned} \tag{Eqn. 20}$$

4.2.2 Perspex C

The URM instruction $C(z)$ increments the contents of register z. Adding J_1 to the location z has the same effect. Hence, it is sufficient to compute:

$$\text{continuum}(\vec{z}) + IJ_1 \rightarrow \vec{z}; \qquad \text{jump}(\vec{z}_{11}, t). \tag{Eqn. 21}$$

4.2.3 Perspex T

The URM instruction $T(m, n)$ writes the contents of register m into register n. Hence, with control passed to successive instructions, it is sufficient to compute:

$$\begin{aligned} &\text{continuum}(\vec{\vec{n}}) + IH \to \vec{\vec{n}}; \qquad &&\text{jump}(\vec{\vec{n}}_{11}, t); \\ &\text{continuum}(\vec{\vec{n}}) + I\vec{m} \to \vec{\vec{n}}; \qquad &&\text{jump}(\vec{\vec{n}}_{11}, t). \end{aligned} \tag{Eqn. 22}$$

4.2.4 Perspex A

The URM instruction $A(m, n, p)$ compares registers m and n then jumps to instruction p if $m = n$, but otherwise *advances* to the next instruction q in the program. It is sufficient for a perspex to jump depending on the value of $m_{11} - n_{11}$. This may be done using an infix subtraction matrix, S, to compute an intermediate perspex, R, with $r_{11} = m_{11} - n_{11}$.

$$S = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

$$R = \begin{bmatrix} m_{11} & 1 \\ 1 & 0 \end{bmatrix} S \begin{bmatrix} n_{11} & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} m_{11} - n_{11} & -1 \\ 1 & 0 \end{bmatrix}.$$

Hence, a sufficient perspex program computes R and then performs the jump:

$$\text{jump}\left(\vec{\vec{z}}_{11}, \begin{bmatrix} q' & p' & q' & 0 \end{bmatrix}^T\right). \tag{Eqn. 23}$$

Where p' and q' are, respectively, the relative jumps from the current location z to p and to q. This completes the proof.

4.3 Perspex Fractals

Fractals[10] are geometrical point-sets. Rendering a fractal in perspex space arranges that a perspex is stored at every point in the fractal. Thus, the fractal becomes a *perspex fractal*. However, we want to form fractals of perspex machines so that one perspex machine may emulate another and so that a perspex machine may explore the space of computations. In this way, we might hope, a fractal will explore and prune computations in a similar way to a symbolic, tree-based search in a Turing machine, but always retaining the cardinality and spatial properties of the perspex machine.

There are many ways to construct fractals of perspex machines, but, perhaps, the simplest is to map the whole of perspex space into a unit hypercube and then to form a fractal from the hypercube. Figure 2 shows a cross-section of one such arrangement. The real part of perspex space is mapped onto a unit hypercube using the *arctanq* function[4,6] modified so that it uses a signed, real, square root instead of a signed, integer, square root. Hence $(\text{arctanq}(a) + 1)/4$ maps the whole of the real-number line, with ordinate a, onto the line segment $(0, 1/2]$. We construct the hypercube a as shown in the Figure. We then form the mappings $\infty \to 3/4$ and $\Phi \to 1$. This completes the construction.

We call a space that contains a perspex machine a *cell*. By constructing the hypercube cells with *arctanq* we arrange that the cell is open below and closed above. Hence, we may stack cells without any space between them, yet they remain disjoint. Operating within a cell by companding with *arctanq* prevents a cell from communicating with adjacent cells, but, if one cell contains another, the superordinate cell may communicate with a subordinate cell. Hence, a superordinate cell may be an oracle to all of its subordinates, and the first cell may be an oracle to all perspexes, including itself. The ability to operate by companding requires that a perspex machine can run an emulator that accesses the local cell as if it were the whole of perspex space. An emulator can also be used to solve the perspex halting problem.

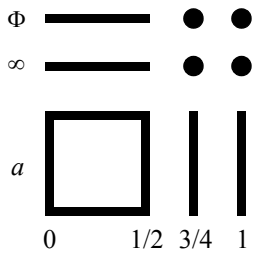

Figure 2: Cross-section of a perspex machine mapped onto a unit hypercube.

It is useful to lay out $\aleph_1$ cells along a line segment. This may be done by using an *x*-*y* plane of perspexes to index a unique length along Peano's plane-filling curve.[10] This curve is defined in the range $[0, 1]$, but the lower bound is not used in the *arctanq* map so the Peano curve may be scaled so that it is accessed in the range $(0, 1/2]$. This range of indexes forms the *y*-axis of a cubical cell, with the *z*-axis a conventional spatial axis, and the *t*-axis a conventional temporal axis. The perspex machine in this 3D cell uses companding to operate on it as if it were a 4D perspex machine. $\aleph_1$ of these cells are laid out in the range $(0, 1]$ along the *x*-axis. Thus, a fractal of $\aleph_1$ cells is mapped onto the unit hypercube.

4.4 The Perspex Halting Problem

Turing's halting problem is an abstract problem. A specification of an automatic-machine is placed on a Turing machine's tape and the machine is to decide whether or not the specified machine will halt if it is executed.[17] Turing shows that no automatic-machine can solve this problem and that whilst an oracle-machine can solve the halting problem for an automatic-machine it cannot solve it for itself and, hence, an oracle-machine cannot solve the halting problem for an arbitrary oracle machine.[17] Both Turing's and Gödel's proofs rely on enumeration so they apply to machines with a cardinality of at most $\aleph_0$, but the perspex machine has cardinality $\aleph_1$ so it is immune from the negative results in these proofs. In fact, many hypercomputers can solve Turing's halting problem for automatic-machines.[9]

The perspex machine is not simply a computer so a solution to an abstract halting-problem does not necessarily deal with all of the halting cases of a perspex machine. We might allow that the perspex machine takes input from the universe at various times during its operation and that it is re-started at various times by some oracles and that some oracles induce error in the machine. Even in these cases we might hope for a solution to the halting problem by exploiting the time travelling abilities of the perspex machine by having it look into the future to see if it has halted; but time travel itself raises difficulties. It can be the case that a perspex machine reads and writes perspexes from and to a time distant from the time at which the perspex is executed. The machine has no record of execution so to detect writes it must copy the whole of spacetime and look for changes. In other words it must emulate perspex spacetime to solve the halting problem. But emulation provides a simpler, direct solution. The perspex machine halts only when it executes the halting instruction, H, so an emulator can flag the halt before executing H. An emulation can detect its own halting over all of the time lines it explores and can arrange to store the halting decisions at fixed points in spacetime. Thus, the machine may have access to its halting decisions at any time by time travel, or at some fixed, future time by accelerating its operations, subject to interference from an oracle. A universal perspex-machine with emulation may solve the halting problem for any one perspex machine, including itself, and for a collection of $\aleph_1$ perspex machines. By hypothesis this is all of the machines that can be instantiated in the universe. However, the universal perspex machine cannot solve the halting problem for the set of all possible perspex machines because this set, being the set of all subsets of perspex space, has cardinality $\aleph_2$. Naturally, one is most interested in halting results that relate to physically possible machines; but it is useful to consider emulation in its own right, because it allows for executive control, such as imposed by a computer's operating system and, we suppose, by a human's consciousness.

Figure 3 gives pseudo code for a perspex emulator. The conditional structures and assignments are explained in.[15] The arithmetic and jump parts are explained in the current paper. The emulator reserves one location, b, as a global flag to indicate whether the emulator is still running or else has halted. The emulator is composed of a 2D plane of perspexes $a^{(i,j)}$. The i^{th} line in a plane emulates one control thread. The perspex $a^{(i,0)}$ is a flag that indicates whether the current thread is still running or has halted. The program is entered at the label *enter:* where the emulator is initialised with the

$\aleph_1$ perspexes to be emulated. The label *loop:* denotes the instructions that clear the global flag, b, and start one cycle of the emulator. The *forall* structure indicates that all of the lines of the emulator are to by executed in synchrony. The emulator checks to see if the emulated perspex is the halting perspex H, if so it does not increment the global flag b, but it does clear the local flag $a^{(i,0)}$ to indicate that the current thread has halted. It then performs the exit actions and halts. Otherwise it emulates the current perspex. It increments the global flag and sets the local one. It constructs two copies of the emulated perspex. One copy has its jump part munged so that it performs the arithmetic specified by the emulated perspex, the other has its arithmetic part munged so that it fetches the next perspex to be emulated as if it had performed the specified jump. The emulator then checks the global flag. If it has not been incremented then all of the emulated threads have signalled a halt. The emulator performs any *exist actions* for each time line and then halts. Otherwise it jumps back to the label *loop:* and executes another cycle of the emulator. When examined from a time different from both $a^{(i,0)}$ and b the oscillating flags have collapsed into a fixed state.[6] Thus, the emulator solves the halting problem for the conditions stated, though one might want different conditions and a different causality.

```
enter:    write the perspexes to be emulated into a^(i,1).
loop:     write Z to b as a global flag indicating that all emulated threads have halted.
          forall a^(i,1) do
              if     a^(i,1) = H
              then   write Z to a^(i,0) to flag the halt of the current emulated control thread.
                     jump to label 3 to execute the exit actions and the halt.
              else   write I to a^(i,0) to flag continuing execution of the current emulated control thread.
                     increment b to flag continuing execution of an emulated control thread.
                     write a^(i,1) to a^(i,2) to preserve a copy of the perspex to be emulated.
                     overwrite a^(i,1)_(j,4) so that it specifies a jump to label 1 in the emulator.
                     jump to a^(i,1) to execute the arithemetic part of the emulated instruction.
label 1:             write a^(i,3) so that it reads a^(i,2)_(j,4), performs identity, writes result to a^(i,1), and jumps to label 2.
                     jump to a^(i,3) to emulate the jump part of the emulated instruction.
label 2:             if     b = Z
label 3:             then   perform exit actions.
                            execute H.
                     else   jump to loop.
                     endif.
              endif.
          endforall.
```

Figure 3: Pseudo code for a perspex emulator.

5 Discussion

We have proved that $0/0$ is a number. It lies off the number line and is not ordered with respect to any number on the real-number line augmented with positive and negative infinity. When nullity arises as the result of an exact calculation it means that there is no unique solution on the number line – though there may be infinitely many non-unique solutions on the line.[2] It can also be that nullity itself, which lies off the number line, is a unique solution[2] or else one of many.[2] Thus, nullity itself gives no information about whether a unique solution exists or where any non-unique solutions lie on or off the number line. We summarise this by saying that nullity means "no information," even though additional calculations might allow us to classify the nullity solutions. This leads naturally to the use of nullity as a "don't know" or "don't care" value in databases and avoids the widespread bug of using a supposedly out-of-range number to mean "don't know" – only for later software updates to bring the value into range with disastrous consequences. We suggest that having a

unique number, nullity, that means that there is no information and which does not interfere with any calculation on the number line is as useful to a manufactured computer as zero is to a human computer.

The "no information" interpretation of nullity is also consistent with the perspex machine. The perspex with all elements nullity is the halting perspex, it gives no information about how the computation should be continued. This is the default perspex in the whole of space so that uninstantiated parts of space automatically instruct a halt. This interpretation makes software more secure. The universal perspex machine allocates spaces for calculations in a fractal. No attempt to read, write, or execute perspexes outside the allocated space can be instructed, and any attempt to execute an uninstantiated perspex inside the allocated space halts. Thus, calculations can be kept within secure fractal bounds, no matter how intricate or close the calculation of results becomes.

In theory the perspex machine is super-Turing, but we have no physical means of implementing a perfect perspex-computer so we cannot access these super-Turing properties in practice. However, a digital computer can access the properties of fractal security, totality of arithmetic, blending of perspex programs, and, no doubt, many more properties. This makes the perspex machine useful in everyday computations. In particular, transrational arithmetic is so simple that it could be taught to primary school children in place of teaching fractions. If this were done, it would mean that the general population would have access to a total arithmetic, rather than, as now, using a partial arithmetic which fails on division by zero, and which encourages programmers to specify faulty computer arithmetic and introduce bugs by using real numbers to denote "don't know" and "don't care" values. The economic and social value of more secure computation might be considerable.

6 Conclusion

We have proved that nullity, $0/0$, is a number for the historically acceptable reason that it arises as the solution to an equation, in this case, an elementary trigonometric equation. Nullity, and positive and negative infinity, can be added to all of the standard arithmetics. This makes them total, which is to say that no exceptions (processing errors) can occur within the arithmetics. Hence, floating-point units and, more generally, arithmetical processing units have no use for exception handling circuitry. This means that they can be fabricated with fewer elements on a chip, making them cheaper and more reliable, as well as being inherently more useful because they operate correctly in corner cases that are currently problematical. In particular, the transarithmetics deal coherently with $0/0$, whereas the IEEE, floating-point treatment[12] of $0/0$ is incoherent.

We have also introduced a general-linear form of the perspex instruction and have shown that it is super-Turing. We have shown that the abstract perspex-machine can solve its own halting problem and that it can partition the entire set of Turing programs into the Turing computable and Turing incomputable subsets. We also argue that Turing believed that his machine has spatial properties, making it a special case of the perspex machine in its nature, as well as in its computational properties. Regardless of its theoretical advantages, the general-linear, perspex instruction is of practical significance because it greatly simplifies programming the perspex machine and implementing perspex compilers.[15] It also makes it easier to construct filters via convolution.[7] The perspex machine can process badly-formed formulae and things that are not formulae of any kind. However, the perspex machine does have theoretical limitations. It cannot compute any set with cardinality greater than the real-number line.

The ability to process badly-formed formulae is significant. It means that a perspex computer is inherently able to deal with partial and erroneous data and, to the extent that approximately similar perspex programs instruct approximately similar computations,[7] it means that badly formed programs will not, in general, be catastrophically wrong. We might hope that this will make perspex computers resistant to damage, will allow machine learning to improve on faulty learning, and will support new forms of genetic algorithm. In these respects, the perspex instruction makes manufactured computers more like human beings. This may be of interest to Artificial Intelligence and might aid the development of more natural human-computer interfaces. One, *extreme,* theoretical possibility is that a manufactured computer might blend with a human being either as an interface, or as a practical demonstration of the existence of other minds.

We have presented a very simple semantics for nullity. Nullity means "no information" whereas zero means "no value." We invite the reader to consider whether the development of nullity is as significant as the development of zero.

7 References

1 Akl, S.G., *The Myth of Universal Computation* Technical Report 2005-429, School of Computing, Queen's University, Kingston, Ontario, Canada, K7L 3N6.

2 J.A.D.W. Anderson "Representing Geometrical Knowledge" *Phil. Trans. Roy. Soc. Lond.* series B, vol. 352, no. 1358, pp. 1129-1139, Aug. 1997.

3 J.A.D.W. Anderson, "Robot Free Will" in van Harmelen, F. (ed) *ECAI 2002. Proceedings of the 15th European Conference on Artificial Intelligence* pp. 559-563, IOS Press, Amsterdam, 2002.

4 J.A.D.W. Anderson, "Exact Numerical Computation of the Rational General Linear Transformations" in *Vision Geometry XI* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 4794, 22-28 (2002).

5 J.A.D.W. Anderson, "Perspex Machine" in *Vision Geometry XI* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 4794, 10-21 (2002).

6 J.A.D.W. Anderson, "Perspex Machine II: Visualisation" in *Vision Geometry XIII* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 5675.

7 J.A.D.W. Anderson, "Perspex Machine III: Continuity Over the Turing Operations" in *Vision Geometry XIII* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 5675.

8 F. Ayres, *Modern Abstract Algebra* Schaum's Outline Series, McGraw-Hill, (1965).

9 G.S. Boolos & R.C. Jeffrey, *Computability and Logic,* 3rd edition, Cambridge University Press (1989).

10 R.M. Crownover, *Introduction to Fractals and Chaos,* Jones and Bartlett Publishers (1995).

11 N.J. Cutland, *Computability,* Cambridge University Press (1980).

12 IEEE, 754-1985 IEEE Standard for Binary Floating-Point Arithmetic.

13 C. Kershaw & J.A.D.W. Anderson, "Perspex Machine VI: A Graphical User Interface to the Perspex Machine" in *Vision Geometry XIV* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 6066 (2006).

14 Searl, J. R. *Intentionality - An Essay on the Philosophy of Mind* Cambridge University Press (1983).

15 M. Spanner & J.A.D.W. Anderson, "Perspex Machine V: Compilation of C Programs" in *Vision Geometry XIV* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 6066 (2006).

16 S. Swierczkowski, *Sets and Numbers,* Routledge & Kegan Paul, (1972).

17 A.M. Turing, "On Computable Numbers, with an Application to the Entscheidungs Problem" *Proc. Lond. Math. Soc.* series 2, vol. 42, pp. 230-265 (1937).

18 A.M. Turing, "On Computable Numbers, with an Application to the Entscheidungs Problem. A correction." *Proc. Lond. Math. Soc.* series 2, vol. 42, pp. 544-546 (1937).

19 A.M. Turing, "Systems of Logic Based on Ordinals" *Proc. Lond. Math. Soc.* series 2, vol. 45, pp. 161-228, (1939).

20 A.M. Turing, "Computing Machinery and Intelligence" in *Mind,* vol. LIX, no. 236, pp. 33-60, (1950).

21 P.D. Welch, "On the Possibility, or Otherwise, of Hypercomputation" *Brit. J. Phil. Sci.* vol. 55, pp. 739-746, (2004).