The Self-Referral Dynamics of Computation: An Introduction to a Course on Algorithms

▪

Paul Corazza, Ph.D.

ABOUT THE AUTHOR

Paul Corazza, Ph.D., received his Bachelor of Arts degree in Western Philosophy from Maharishi International University in 1978 and his M.S. and Ph.D. degrees in Mathematics from Auburn University in 1981 and 1988, respectively. He was awarded a Van Vleck Assistant Professorship at University of Wisconsin for the years 1987–1990. He worked in the Mathematics Department at Maharishi International University in the years 1990–95. Following a career as a software engineer, he rejoined faculty at Maharishi University of Management in 2004 and currently serves a joint appointment in the Departments of Mathematics and Computer Science. Dr. Corazza has published more than a dozen papers in Set Theory, focused primarily on the quest for providing an axiomatic foundation for large cardinals based on a paradigm derived from Maharishi Vedic Science.

A B S T R A C T

The immense range of processes and behaviors that make up our universe is somehow managed by nature's computational dynamics, but how exactly is all of it done? Physics has discovered mathematical models for virtually every observable phenomenon, but does nature use anything like "mathematical formulas" to accomplish its ends? Maharishi Vedic Science suggests that, at its deepest level, Nature's performance is a self-referral performance; the mistake-free creation and governance that characterize Nature's expression are due to a certain subtle level of activity that is ordinarily hidden from view—the self-interacting dynamics of silence iteslf. Moreover, quantum field theory has provided a modern-day validation of this ancient wisdom in its discovery that all observables in the form of forces acting on objects are the expression of the self-interacting computational dynamics of a single unified field, a field whose all-encompassing, unified status is also ordinarily hidden from view. In this paper we show that, in fact, the very mathematical notion of computation is based on self-referral dynamics of an expanded (in fact, unmanifest) domain of operators. Having described the way in which every computable function arises as the fixed point of such an operator, we discuss how the dynamics of creation from within this self-referral field, as described by Patanjali's Yoga Sutras, are reflected in this mathematical space of operators as they give rise to all possible computations.

Introduction

An algorithm is a procedure or sequence of steps for computing values from given inputs. Whenever we write a program in Java or C, for example, we are providing a sequence of commands that tell the computer how to proceed in order to output the required values on a given input.

A course on algorithms considers problems that can be solved by finding a suitable algorithm and analytical methods which help to determine if a particular algorithm is more or less efficient than another one that happens to solve the same problem.

In this article, we look more deeply into what underlies algorithms—what is a computation, really? This is interesting since much of what we do depends on efficient computation, because we rely on technology (whose functioning is based on algorithms of various kinds), and also because we do our own form of "computation" in the very way we

conduct our lives—from working out a plan for the day to managing shopping lists to building a career.

In the history of scientific endeavor, man has attempted to understand nature by providing mathematical models. Modern physics has done an excellent job in providing models and computational techniques that allow us to understand and predict nature's functioning in various ways. For example, we have a very accurate model of the motion of planets, so that we can compute with an extremely high degree of accuracy where any of the planets of our solar system will be at any time, within an enormous time span.

Procedures for modeling natural processes are, of course, algorithms, and it follows that, in modern times, understanding and predicting nature's behavior is facilitated by the implementation of sophisticated algorithms on high-powered computers.

But how does nature itself compute the actions for which we have models? What are nature's computations that underlie the motion of the planets and stars? Does nature somehow make use of equations of motion that have been obtained in physics, or does it use some other mechanism for computing?

Again, this is inherently interesting because nature is so powerful. So much is accomplished by the intelligence of nature—it is natural to aspire after the ability to harness the power of nature in some way.

In looking for how nature itself carries out its behavior, one must be willing to accept that many of the secrets underlying the world we see may be hidden beneath the surface. Consider this analogy: Suppose you go to see a magician perform. Perhaps he does the well-known trick of sawing a woman in half. The woman gets into an oblong box, the box is covered with its lid, and the magician pulls out a saw and starts cutting the box in half. It appears that the woman is being cut into two, and this impression is augmented by the fact that the magician moves the two pieces of the box apart, with both halves of the woman apparently also being separated. At the end of the trick, the pieces of the box are returned to their original position and the woman steps out of the box unharmed.

We must answer the question "What just happened? What was the procedure behind the performance just witnessed?" If you try the trick

at home, you quickly discover that there was more to it than meets the eye.

The general principle is this: There is more to the intelligence underlying the performance than is revealed in the performance itself.

If you attempt to perform the trick by simply copying the steps that you were able to observe, you discover that your algorithm, though partially correct, falls short of the full effect.

This example is deeply related to all that we do in our lives and all that science does as it attempts to understand nature. Suppose you want to build a successful career. You may watch the behavior of other people who appear successful and so you may decide that your "algorithm" for success is to copy the steps that you have observed from these successful individuals. The problem with this algorithm is the same as the problem in reproducing the magic trick: You may not be seeing the true computational dynamics that underlie the success of these individuals. Attempting to copy their steps may therefore not be fruitful at all.

Modern physics has made a similar discovery: to understand nature's true computational dynamics requires an investigation that probes deep beneath the surface. Physics began with an understanding that the world consists only of particles (or objects) and forces (like gravity and electromagnetism) that act on them. But a deeper understanding that emerged was that both material particles and forces are expressions of a deeper reality, called *quantum fields*—all particles, like electrons and protons, and all forces, like gravity and electromagnetism, are excitations of an abstract unbounded field. For instance, electrons are excitations of the electron field; gravity is an excitation of the gravitational field. The next step was the recognition that many of these fields that seem to be quite different from each other are, at a very fine level of nature's functioning—that is, at small time and distance scales—in fact identical. Electrons and protons, at a small scale, are seen to be in fact excitations of the same field. And at the finest level of all, known as the *Planck scale*, all force and matter fields are seen to arise from a single unified field. (See Hagelin, 1987.)

What physics has discovered is that everything that we see—all objects, processes, and dynamics—are ultimately the consequence of certain computational dynamics of a single field, at a very fine level of nature. It is by virtue of the unseen dynamics of this unified field

that the "show" we observe in day-to-day life is made possible. These dynamics can be described in a simple way as the interaction of the unified field with itself—*self-referral dynamics*.

This is a remarkable discovery. It tells us that life is not truly as it appears. Things happen according to dynamics that are not obvious on the surface. We see a world of objects and forces, but both of these are effects of a much subtler cause—an abstract field interacting with itself somehow appears as this world of objects and forces.

Now we return to our initial question: What are the true computational dynamics of nature? Physics has discovered that the answer lies in this abstract unified field. And, although the mathematical description of these dynamics is quite complicated, one point about these dynamics is certain and easy to express: they are inherently *self-referral*. Indeed, it is by virtue of the unified field's self-interaction that all that is observed and experienced in the physical world takes place. This discovery alerts us to an important fact about the universe in which we live: behind the scenes, self-referral dynamics are at work in the structuring of all that we experience.

The Truth about Computation Itself

We wish to examine the very idea of computation itself to discover what is fundamental. Ultimately, we would like to harness the essence of computation in some way to enhance our "computing power" both in a scientific sense and also in the subjective sense mentioned earlier. Since it seems that self-referral dynamics are fundamental for computation in nature, we are interested to see if self-referral dynamics are in fact fundamental to the very notion of computation as it is understood in mathematics.

In mathematics, a computation is performed as an implementation of an algorithm—a sequence of procedural steps is applied to a concrete set of input values in order to produce an output value. Though these steps can often be carried out "by hand," we can describe the action of algorithms in a more general way by saying that algorithms are implemented by computer programs. For example, let us consider a Java program that implements the factorial function. Recall that the factorial of an integer n is the product of all natural numbers less than or equal to n.

```
int factorial(int n) {
   int accum = 1;
   for(int i = 2; i <= n; ++i){
      accum *= i;
   }
   return accum;
}
```

This particular implementation of the factorial function is called an *iterative program* because computations are performed iteratively in a loop. The same function can be computed in a different way using the technique of recursion:

```
int factorial(int n) {
   if(n < 0) return 0;
   if(n == 0 || n == 1) return 1;
   return n * factorial(n-1);
}
```

Here, recursive calls are made to `factorial` in order to perform the computation. Recursion has a noticeable flavor of self-referral in it since, to compute `factorial` in this case we call `factorial` itself: we are in effect declaring `factorial(n)` to be equal to `n * factorial(n - 1)`. This fact tells us that the mathematical notion of computation involves, at least in some cases, certain self-referral dynamics. However, even this example of recursion is not a pure self-referral expression since `factorial(n)` is being computed in terms of `factorial` applied to a smaller argument n - 1; we do not see `factorial` being defined entirely in terms of itself. At this point in our investigation, we see very little evidence that mathematical computation is really grounded in self-referral dynamics.

As we look more deeply into the question, a first step is to notice the remarkable fact that *every* program can be rewritten as a recursive program, not just the `factorial` program. This fact can be demonstrated in the following way:

Let f: N → N be a computable function; that is, f is a function for which we can define a Java method (or a method in any procedural language) `int java_f(int n)` with the property that, for any natural

number k, if $r = f(k)$, then the output of `java_f` on input k is r. Let $t_0 = f(0)$.

We can, in a computable way, obtain a function en that outputs, for each n, the value $f(n + 1)$ from the pair of inputs $n, f(n)$, namely,

$$en(n, y) = \begin{cases} f(n+1) & \text{if } y = f(n) \\ 0 & \text{otherwise} \end{cases}$$

Now we can define a function g: N → N by recursion whose input/output behavior is identical to that of f:

$$g(0) = t_0$$
$$g(n + 1) = en(n, g(n)).$$

To prove $f = g$, one reasons by induction: Certainly $f(0) = t_0 = g(0)$. Assuming $f(n) = g(n)$, then $f(n + 1) = en(n, f(n)) = en(n, g(n)) = g(n + 1)$.

We have shown that all computation is reducible to recursive computation. And we can see that definition by recursion has some flavor of self-referral dynamics. A question that was asked by logicians many decades ago is this: We see that using recursion, a function like the factorial function fact can be defined by

$$\text{fact}(n) = F(n, \text{fact}(n - 1))$$

where $F(x, y) = x * y$. But is there a more elegant kind of self-referral dynamics whereby fact could be defined in the following, seemingly circular way?

$$\text{fact} = F(\text{fact})$$

Such a "definition" would be an unqualified expression of self-referral dynamics in the mathematics of computation, since, if it were possible to make sense of such a definition, the function fact would be seen to arise entirely in terms of itself, through some unseen self-interacting dynamics.

In fact, it is indeed possible to define the factorial function in this way. We examine the details of how this can be done. (See Aczel, 1977, and Weinless, 1987, for a fuller discussion.) Let S denote the collection of all functions g: $A \rightarrow$ N, where $A \subseteq$ N. For example, functions like

$g(n) = 2n$ and $g(n) = n^2$ are in S, but also functions having restricted domain are in S, such as $g(n) = \log_2 n$ whenever n is a power of 2 (here, the domain A of g is just the set of powers of 2: $\{1, 2, 2^2, 2^3, \ldots\}$).

We define a function F: $S \to S$ as follows: For any g: $A \to \mathrm{N}$ in S, let A' denote the set consisting of 0 together with all successors of elements of A. For instance, if $A = \{0, 2, 5\}$, then $A' = \{0, 1, 3, 6\}$. Given g, we let $F(g)$ be the function $h : A' \to \mathrm{N}$ defined so that for all $n \in A'$:

$$h(n) = \begin{cases} 1 & \text{if } n = 0, 1 \\ n * g(n-1) & \text{otherwise} \end{cases}$$

Notice that, for $h(n)$ to be defined, $n - 1$ must be in the domain of g—and this is guaranteed by our definition of A'. Notice in particular that, if $\perp$ denotes the "empty function"—the function that has no inputs or outputs, and so has domain the empty set $\emptyset$—then $F(\perp)$ is a function h defined on the set $\{0\}$ so that $h(0) = 1$.

Let's show that fact = F(fact) by induction: Let $h = F$(fact). Then $h(0) = h(1) = 1$, and so likewise fact(0) = fact(1) = 1. Assuming $h(n) = \mathrm{fact}(n)$, we have

$$h(n + 1) = (n + 1) * \mathrm{fact}(n) = \mathrm{fact}(n + 1).$$

By induction, it follows that $h = \mathrm{fact}$. We therefore have a completely "circular" or self-referral definition of the factorial function:

$$\mathrm{fact} = F(\mathrm{fact}).$$

Notice that here the input argument fact of F is the same as the output—fact is called a *fixed point* of F. This observation is an instance of an important mathematical result:

Recursive Operator Theorem.[1] For every recursively defined number-theoretic function f, there is an operator F: $S \to S$ such that f is the least fixed point of F; that is, f can be defined by

$$f = F(f).$$

1 The name we have given to this theorem is for the convenience of this paper; in the literature, this result has not been formally assigned a name. The idea behind the proof is that the rules (in the sense of Aczel, 1977) that define a given recursion can be canonically transformed into a monotone continuous operator F: $S \to S$ whose least fixed point is the solution to the recursion.

We have shown that all computation is fundamentally self-referral computation. However, this truth about computation is not obvious on the surface—in fact, this insight is not available in the ordinary framework in which computation is done in applications. In order to discover this truth about computation, it was necessary to consider an expanded context: our discovery was made in the world of operators $S \to S$, which is a much larger infinite domain than that of computable functions.

This fact, that an expanded context is necessary to discover the self-referral nature of computation, parallels the discovery of self-referral dynamics at the basis of the physical universe: the self-referral dynamics at the basis of the universe are discovered in physics at the Planck scale—that is, at extremely small time and distance scales. Therefore, these dynamics are ordinarily not apparent in the ordinary world of forces acting on objects.

Self-Referral Dynamics in Life

The questions "How does Nature compute?" and "What is the ultimate truth about the nature of computation?" have also been addressed by the ancient texts and brought into the contemporary stream of knowledge by Maharishi Mahesh Yogi. According to Maharishi Vedic Science, the life that we see, with one step of the story of our lives coming after another, is in fact a manifest, obvious unfoldment of unseen self-referral dynamics, the self-referral dynamics of pure intelligence (Maharishi, 1995, pp. 165–171). Nature's computation is in fact a self-referral performance. Each expression in the manifest world is the result of pure consciousness curving back on itself, collapsing from infinitely expanded value to a point (Maharishi, 1996, p. 539). The Veda declares: *Richo akshare parame vyoman*. . . . This means that the fundamental structuring impulses of the creation arise in the collapse of the fully expanded value of intelligence, to its point, and subsequent expansion from point to unboundedness (Maharishi, 1996, p. 484).

The sequential flow that is observed in natural processes and the sequential nature of computations as we do them in mathematics and computer science are, from the Vedic perspective, expressions of an eternal self-referral process, a "hum" within the unboundedness of pure intelligence (see Lester, 1987, pp. 306–309).

These mechanics are embodied in the first word of Rik Veda: AGNIM. 'A' represents the fully expanded value of pure intelligence, and 'K' (note: A + K = AG) represents the fully contracted value, the point value. 'N' is negation, which in this case means "negation of the contraction," in other words, expansion back to unboundedness. 'I' is continuation, so we find a continuation of collapse and expansion. And 'M' is the simultaneous occurrence of all frequencies, representing the infinitely frequent collapse and expansion embodied in the previous letters (for a more detailed discussion with references, see Lester, 1987, pp. 306–309).

Let us now ask: What is the practical value of this knowledge? For an answer, let us return for a moment to the magic show mentioned earlier. Suppose you decide you want to be able to do the trick the magician did. As we have observed already, without gaining access to the unseen dynamics of the magician's performance, it will be impossible to do so. The key to success in this case is the ability to gain access to, and make use of, these hidden dynamics.

Analogously, to achieve the greatest success in pursuing the many goals we have for our own lives, the key is to gain access to, and make use of, the hidden dynamics of life itself; in this case, we seek to harness the unseen dynamics by which Nature governs the universe. Such an immense intelligence and power would of course be of great value for our own individual lives. In building a career, a family, wealth, personal fulfillment—there really would be no limit to the benefits from tapping Nature's structuring dynamics. Just as in the case of the magician, these dynamics are not obvious on the surface; it is necessary to dig deeper to access this level of life.

Achieving this goal is the purpose of the Transcendental Meditation technique that we teach as part of the curriculum at Maharishi University of Management. It provides a key to tap this vast field of intelligence and provides the practical value—the "technology"—of the Vedic wisdom we have been discussing. One of the ancient Vedic texts, called the Bhagavad-Gita, summarizes in three verses the practical application of the wisdom of the ancient texts (Maharishi, 1969):

- *Nistraigunyo bhavarjuna*
  Go beyond the field of change; transcend; contact the field of Being. (2.45)

- *Yogastah kuru karmani*
  Established in Being, perform action. (2.48)

- *Durena hy avaram karma buddhi-yogad dhananjaya*
  Far away indeed from the balanced intellect is action devoid of greatness. (2.49)

Regular dives into the unbounded aspect of our own intelligence automatically accomplish these aims. Contact with the transcendental field is contact with the field from which Nature operates. The Veda describes the transcendent as *yasmin deva adhi vishve nisheduh*—the home of all the Laws of Nature (Maharishi, 1996, pp. 113, 515). Waking up to this field automatically brings Nature's self-referral dynamics into the flow of our individual lives, leaving "action devoid of greatness" far behind.

Directly Activating Nature's Self-Referral Dynamics

In the Transcendental Meditation program, we stir the transcendental field and automatically bring benefits into our daily life. But it is possible to more directly engage Nature's self-referral dynamics through an advanced program, the TM-Sidhi program.

In the TM-Sidhi program, acting from the settled state of awareness arising in meditation, one introduces a faint intention for a particular result or "Sidhi." The intention is a kind of point value, and our settled state of awareness, which entertains this intention, is at the same time awake to the unbounded value of intelligence. So there is an interaction between point and unboundedness, and in the togetherness of these, a "result" comes—the result that was originally intended.

Maharishi Patanjali, part of the Vedic tradition of knowledge, described this technique of engaging Nature's intelligence in the following way in the Vedic text called the *Yoga Sutra* (see Aranya, 1977). He explains that it is by virtue of the togetherness of *Dharana* (intention), *Samadhi* (transcendence), and *Dhyana* (flow of awareness) (*Yoga*

Sutra 3.1, 3.2, and 3.3) that the result of the intention, a particular Sidhi, is produced. Patanjali calls (Yoga Sutra 3.4) the togetherness of these three *Samyama*. The TM-Sidhi program makes use of Patanjali's principles in a simple and automatic way. The result is that a Sidha is able to more directly engage Nature's self-referral dynamics (see Wallace, 1986).

If we take a closer look at the deeper self-referral dynamics that underlie computation itself, we can see that the way Nature computes in its self-referral way, particularly in the way in which Sidhis arise from *Samyama,* is strikingly analogous to the way computation itself works at the deeper mathematical levels. To see the analogy, let's return to our computation of the factorial function fact using the operator $F : S \to S$. Recall we were able to show that

$$\text{fact} = F(\text{fact}).$$

Let us now see how the function fact actually emerges by way of self-referral dynamics within the expanded domain of operators $S \to S$ in much the same way as a Sidhi arises from *Samyama.*

We begin with F; F is the operator that has been specifically encoded with information that is intended to produce the factorial function. It represents the intent behind our self-referral computation and corresponds to *Dharana*.

Next, we allow F to interact with the empty function $\perp$ by application: $F(\perp)$. The empty function is the function that accepts no input and has no output, corresponding in programming terms to a function such as the following:

```
void empty() {
   while (true) {  ;  }
}
```

The empty function can be viewed as a field of all possibilities—no commitment has been made to particular input/output behavior; it is an analogue to the unbounded nature of intelligence, and corresponds in our analogy to *Samadhi*. We now apply F to $\perp$ not just once but repeatedly.

$$(*) \qquad \perp \subseteq F(\perp) \subseteq F(F(\perp)) \subseteq \ldots$$

This sequential unfoldment is like the flow of awareness described by Patanjali—*Dhyana*.

In this togetherness of intent (F), transcendence ($\perp$), and flow ($*$), what emerges is a result, a computation. The result in this case is obtained by forming the union

$$h = \perp \cup F(\perp) \cup F(F(\perp)) \cup \ldots$$

One can then show that h is in fact the (unique) fixed point of F:

$$\begin{aligned} F(h) &= F(\perp) \cup F(\perp) \cup F(F(\perp)) \cup \ldots \\ &= F(\perp) \cup F(F(\perp)) \cup \ldots \\ &= h. \end{aligned}$$

Moreover, as we showed before by induction, such a fixed point for F must actually be the factorial function itself. That is,

$$h = \text{fact}.$$

The Language of Self-Referral in the Field of Computation

We have seen how, by expanding the context of study from computable functions $\mathrm{N} \to \mathrm{N}$ to the domain of operators $S \to S$, we can locate the self-referral dynamics underlying all computation. In the 1930s, a mathematical language, called the λ-calculus, was developed by Alonzo Church (1932/3) to provide a mathematical foundation for computation. This language expresses these self-referral dynamics in a more foundational way.

Church was seeking to build all of mathematics from the concept of transformation, the concept of a function. Though the result of his research fell short of being a foundation for *all* of mathematics, it turned out to be a foundation for the notion of *computation*. Church envisioned a universe built up entirely of functions. An important implication of such a universe is that, not only does a function f act on its inputs (as is usually the case with functions), but the domain (and range) of f includes all possible functions as well, *including f itself*. In particular, this means that some meaning must be attached to a term of the form $f(f)$. Any such universe would have to be highly self-referral.

Church's formal λ-calculus has the following elements: An infinite set of variables $v_1, v_2, \ldots$ together with λ-terms defined recursively by:

(1) Any variable is a term.
(2) (*Application*) If M and N are terms, (MN) is a term.
(3) (*Abstraction*) If M is a term and x is a variable, $(\lambda x.M)$ is a term.

The λ-notation is a way of specifying a function. Stepping outside the formal λ-calculus for a moment, one can use the λ-notation to specify ordinary functions in a convenient way. For instance, the function that takes n to $2n$ can be expressed by writing,

$$\lambda n.2n.$$

"$\lambda n.2n$ is the function that takes argument n to the value $2n$."

In our definition of λ-terms, clause (3) says, intuitively, that if M is a term, the "function" that takes x to M is also a term. Multiple applications of clause (3) are typically expressed in abbreviated form. For instance $\lambda x.\lambda y.M$ is written $\lambda xy.M$. The complexity of nested parentheses that can occur in practice, in writing down terms according to rules (2) and (3), tends to become unmanageable, so there is an "association to the left" convention: It is understood that the expression MNP is shorthand for $((MN)P)$.

λ-terms can often be "reduced" to simpler terms. Many of these reductions follow our intuitive expectation. For instance, the term $(\lambda x.y)M$ reduces to simply y. The intuitive reason is that $\lambda x.y$ is the function that takes any x to constant value y. If this function is applied to the input M, the output is simply y. We write $(\lambda x.y)M \rightarrow y$. As an exercise, the reader may wish to verify (intuitively, as we have done here) why $(\lambda x.x(xy))M \rightarrow M(My)$.

More formally, the λ-calculus specifies rules for transforming λ-terms to other λ-terms (see Barendregt, 1984). The key axioms of the λ-calculus are the *β-rule*, for performing a reduction (as described in the previous paragraph), and the *ξ-rule*, which says that equal terms remain equal after abstraction. Here are the formal statements:

β-rule: $(\lambda x.M)N = M[x/N]$.

ξ-rule: if terms M, N are equal, then terms $\lambda x.M$, $\lambda x.N$ are also equal.

Here the notation $M[x/N]$ means that all occurrences of x are replaced by inserting the term N (there is a small technical point about how N is to be inserted, but we skip over this issue in this brief introduction).

Notice that the λ-calculus doesn't have much in it—just variables, two rules for combining the variables into terms, and two rules to regulate interaction of terms. This simplicity is reminiscent of pure intelligence itself—there's nothing much there! The formula for the dynamics of this simple field of pure intelligence is given by the first word of Rik Veda: AGNIM. Pure intelligence collapses to a point and expands to infinity—this is the character of its self-interacting dynamics. Analogously, in the λ-calculus, the self-interacting dynamics of λ-terms are governed by two rules: Application (the β-rule), which describes how the infinitely expanded value of an abstraction term is collapsed to a point, and Abstraction (the ξ-rule), which regulates the process by which points may be expanded to the infinitely expanded value of an abstraction term or "function."

As we have seen, the λ-calculus gives rise to λ-terms. Among the terms of the λ-calculus are the objects that are familiar to us for use in computation: We can locate a copy of the natural numbers and representations of the concept of an ordered pair, for example. Even more significantly, we can represent all computable functions in the language of the λ-calculus, and when this is done, the λ-calculus allows us to prove that every computable function arises as a fixed point of a λ-term, and, moreover, the very process of recursion itself is seen to be the result of locating a fixed point for a suitable λ-term. (Details are given in the Appendix.) Therefore, the insight discussed earlier—that every computation arises through self-referral dynamics of an expanded operator domain, in the sense that each arises as the (least) fixed point of one such operator—is given precise expression in the language of the λ-calculus. In other words, the language of the λ-calculus reveals the hidden self-referral dynamics of computation.

As a final observation showing the natural link between the language of the λ-calculus and the self-referral dynamics of pure intelligence, we recall that the transformations of the field of intelligence

naturally divide into three: knower (Rishi), object of knowing (Chhandas), and process of knowing (Devata). The λ-calculus itself has this natural threefold division: Being a purely functional language, its essence is Devata, or transformation. What emerges within this field of pure dynamism are "precipitations": numbers, ordered pairs, computable functions, etc. These are the objects of knowledge, the Chhandas value. And the driving intelligence behind all of it is the simple set of axioms for the λ-calculus which regulate the dynamism of this domain; these two axioms (the β-rule and the ξ-rule) happen to correspond to the two fundamental directions of flow of pure consciousness itself: collapse to a point (application) and expansion to infinite (abstraction).

Conclusion

In this introduction to Algorithms, we began by asking about the nature of the computing process, and in particular, how nature performs the computations that produce the flow of existence that we observe in the world. An analogy from everyday experience suggested to us that the truth concerning nature's way of computing, and concerning computation in general, is very likely to be hidden from view, not obvious from casual observation. Moreover, a survey of recent discoveries from quantum physics suggests that nature's functioning, at its basis, is inherently self-interacting and entirely beyond the range of observable phenomena. We asked whether this self-interacting characteristic, inherent in nature's fundamental computational dynamics, is actually a characteristic of the mathematics of computation itself. Our investigation showed that this is indeed the case: Every computation, represented in the form of an algorithm or computer program, can be formulated as a recursive computation, and every recursive computation can be seen to arise as a fixed point of an operator on the class of partial functions from N to N; that is, from purely self-referral dynamics arising in an expanded function space. Moreover, we showed how this phenomenon, that recursive computations arise as fixed points of operators, finds natural expression in the language of the λ-calculus, in which one can demonstrate that every computation arises as a fixed point of a λ-term. These mathematical self-referral dynamics, at the heart of mathematical computation on N, are, like the dynamics of the

unified field, "hidden from view" in the sense that they can be discovered only in a domain (the domain of operators from *S* to *S*) that is far vaster than the ordinary domain of computable functions.

Finally, we looked into the practical application of this insight about computation and how nature computes. A natural desire is to harness for oneself the enormous intelligence and organizing power of Nature's self-referral functioning; the practical application is to dramatically uplift every area of life by bringing to bear Nature's cosmic level of functioning to the realm of individual life. The key to accomplish this goal, as we discussed, is to open individual awareness to the level at which Nature carries out its self-referral performance—the level of the Unified Field of all the Laws of Nature. Simply by contacting this field, through the Transcendental Meditation program, one immediately begins to experience the influence of this field in daily life. Moreover, we saw that one could further engage Nature's intelligence through the Maharishi TM-Sidhi program[SM], a program that provides a modern, easily practiced formulation of the *Sutra* practice described in the classic Vedic text, Patanjali's Yoga Sutras.

The vision proclaimed by the ancient texts—that Nature's organizing power has its basis in the self-referral dynamics within the field of pure consciousness, and that simply allowing awareness to dive into this field of pure consciousness is enough to draw Nature's powerful organizing dynamics to support individual life—finds validation in the very mathematics of computation: modern mathematics itself proclaims that all computation is, at its basis, a self-referral performance of a deep, infinitely vast field.

References

Aczel, P. (1977). An introduction to inductive definitions. In J. Barwise (Ed.), *Handbook of mathematical logic* (pp. 739–782), New York: North Holland.

Aranya, H. (1977). *Yoga philosophy of Patanjali*. Calcutta University Press.

Barendregt, H.P. (1984). *The lambda calculus: Its syntax and semantics*. Amsterdam: North-Holland.

Church, A. (1932/3). A set of postulates for the foundation of logic. *Annals of Mathematics, 33*, 346–366, and *34*, 839–864.

Cutland, N. (1980). *Computability, an introduction to recursive function theory.* Cambridge University Press.

Hagelin, J. (1987). Is consciousness the Unified Field? A field theorist's perspective. *Modern Science and Vedic Science, 1*, 29-87.

Lester, B. (1987). Unified field based computer science: Towards a universal science of computation. *Modern Science and Vedic Science, 1*, 267-321.

Maharishi Mahesh Yogi (1969). *On the Bhagavad-Gita: A new translation and commentary, chapters 1–6 with Sanskrit text.* Baltimore: Penguin Press.

Maharishi Mahesh Yogi (1995). *Maharishi's Absolute Theory of Government.* India: Age of Enlightenment Publications.

Maharishi Mahesh Yogi (1996). *Maharishi's Absolute Theory of Defence.* Holland: Maharishi Vedic University Press.

Rogers Jr., H. (1988). *Theory of recursive functions and effective computability.* Cambridge Massachusetts: The MIT Press. (First published 1967, Academic Press.)

Wallace, R.K. (1986). *The Maharishi Technology of the Unified Field: The neurophysiology of enlightenment.* Fairfield, Iowa: MIU Neuroscience Press.

Weinless, M. (1987). Self-referral in the foundations of mathematics. In *Consciousness-Based education: A foundation for teaching and learning in the academic disciplines, vol. 5, Consciousness-Based education and mathematics* (P. Corazza, A. Dow, C. Pearson, D. Llewellyn, eds.), Maharishi University of Management Press, Fairfield, IA, 2009.

Appendix

We give here some of the details of the syntax of the λ-calculus and illustrate the point made earlier that computable functions can be represented in the λ-calculus and arise in every case as fixed points of λ-terms (for a more detailed treatment, see Barendregt, 1984). To illustrate these points, we once again use the factorial function 'fact' as a point of reference.

We list some λ-terms that are used frequently and mention a few results that can be proven:

1. $\mathbf{I} = \lambda x.x$.
2. $\omega = \lambda x.xx$.
3. $\Omega = \omega\omega$.
4. (Truth values) $\mathbf{T} = \lambda xy.x$, $\mathbf{F} = \lambda xy.y$.
5. (Conditionals) Suppose B, M, N are λ-terms. Represent "if B then M else N" by the term
$$BMN$$
6. Note that
$$\mathbf{T}MN \to M$$
$$\mathbf{F}MN \to N$$

7. (Ordered pairs) Given λ-terms M, N, we represent the ordered pair (M, N) by
$$[M, N] = \lambda z.zMN$$

   We also represent the operations that extract the first and second elements of a pair by:
$$(X)_0 = X\mathbf{T}$$
$$(X)_1 = X\mathbf{F}$$

   It therefore follows that $([M, N])_0 \to M$ and $([M, N])_1 \to N$.

8. (*Natural numbers*) We represent the natural numbers 0, 1, 2, . . . as λ-terms using the notation $\overline{0}$, $\overline{1}$, $\overline{2}$, . . . , defined as follows:
$$\overline{0} = \mathbf{I}$$
$$\overline{1} = [\mathbf{F}, \overline{0}] \to [\mathbf{F}, \mathbf{I}]$$
$$\overline{2} = [\mathbf{F}, \overline{1}] \to [\mathbf{F}, [\mathbf{F}, \overline{0}]]$$
$$\overline{3} = [\mathbf{F}, \overline{2}] \to [\mathbf{F}, [\mathbf{F}, \overline{1}]] \to [\mathbf{F}, [\mathbf{F}, [\mathbf{F}, \overline{0}]]]$$
$$\overline{n+1} = [\mathbf{F}, \overline{n}].$$

9. (*Predecessor and Successor*) Represent the predecessor and successor functions of arithmetic by **P** and **S**, respectively:

$\mathbf{P}\ x = \lambda x.x\mathbf{F}$ (it follows that $\mathbf{P}\ \overline{n+1} = \overline{n}$)

$\mathbf{S}\ x = \lambda x.[\mathbf{F}, x]$ (it follows that $\mathbf{S}\ \overline{n} = \overline{n+1}$).

10. (*Test for zero*) The term **Zero** allows us to test whether a term is the term $\overline{0}$.

$$\mathbf{Zero}\ x = \lambda x.x\mathbf{T}.$$

This definition allows us to prove that

$$\mathbf{Zero}\ \overline{0} = \mathbf{T}$$

$$\mathbf{Zero}\ \overline{n+1} = \mathbf{F}.$$

11. (*λ-definable functions*) We can represent all computable functions $\mathrm{N} \to \mathrm{N}$ in the λ-calculus. We focus on functions defined on all natural numbers (*total* functions), but there is an analogous treatment for partially defined functions as well. We say that f: $\mathrm{N} \to \mathrm{N}$ is *λ-definable* if there is a λ-term F such that, for all k, n, we have

$$f(k) = n \quad \text{if and only if} \quad F\overline{k} = \overline{n}.$$

In other words, for all k,

$$F\overline{k} = f(k).$$

12. (*Multiplication is λ-definable*) It can be shown that ordinary addition and multiplication are λ-definable. We assume this here and give the name **Mult** to the λ-term for multiplication. Therefore, for any natural numbers m, n, we have by λ-definability:

$$\mathbf{Mult}\ \overline{m}\ \overline{n} = \overline{m \cdot n}.$$

13. (*Fixed-Point Theorem*) We think of λ-terms as functions, defined on this highly dynamic domain. The Fixed-Point Theorem tells us that every λ-term has a fixed point.

Fixed-Point Theorem. For every F there is X such that $FX = X$.

Proof. Let $W = \lambda x.F(xx)$. Let $X = WW$. Then

$$X = (\lambda x.F(xx))W \rightarrow F(WW) = FX.$$

Generalized Fixed-Point Theorem. Suppose $C = C(f, x, y, z)$ is a λ term involving only the free variables f, x, y, z (and similarly for bigger sets of variables). Then there is a λ-term F such that for all λ-terms X, Y, Z,

$$FXY\,Z \rightarrow C(F, X, Y, Z)$$

(where $C(F, X, Y, Z)$ is obtained from $C(f, x, y, z)$ by substitution). (Notice that in the theorem statement, F is defined in terms of itself.)

The Generalized Fixed-Point Theorem is proved by using the Fixed-Point Theorem to obtain a fixed point for

$$\lambda fxyz.C(f, x, y, z).$$

Example of Generalized Fixed-Point Theorem. We show that there is a λ-term F such that

$$Fxy = FyxF.$$

We use the Generalized Fixed Point Theorem to prove this. Let $C(f, x, y)$ be the term $fyxf$. Let F be the λ-term given by the theorem. Then F satisfies $Fxy \rightarrow FyxF$.

14. (*Recursive factorial defined in λ-calculus*) This example shows how factorial is represented in the λ-calculus as a fixed point of a λ-term. Equally important, it shows how recursion itself arises via the existence of fixed points for λ-terms, illustrating explicitly the self-referral dynamics underlying recursion and computation generally. We show that factorial function fact is λ-defined by

the λ-term F specified by the following self-referral expression (guaranteed to exist by the Generalized Fixed-Point Theorem):

$$Fx = (\text{if } \mathbf{Zero}\ x \text{ then } \overline{1} \text{ else } \mathbf{Mult}\ (F(\mathbf{P}\ x))x).$$

We prove that F λ-defines fact by induction on n. If $n = 0$, then

$$F\overline{0} = \overline{1} = \overline{fact(0)}.$$

Assuming $F\overline{n} = \overline{fact(n)}$,

$$\begin{aligned} F\overline{n+1} &= \mathbf{Mult}\ (F(\mathbf{P}\ \overline{n+1}))\ \overline{n+1} \\ &= \mathbf{Mult}\ (F\ \overline{n})\ \overline{n+1} \\ &= \mathbf{Mult}\ \overline{fact(n)}\ \overline{n+1} \\ &= \overline{fact(n)\cdot(n+1)} \\ &= \overline{fact(n+1)}. \end{aligned}$$