

## COGNITION: DISCRETE OR CONTINUOUS COMPUTATION?

Paul Smolensky<sup>a,\*</sup>

<sup>a</sup>*Cognitive Science Department, Johns Hopkins University, Baltimore, MD 21218, USA*

---

‘Cognition is computation.’ This radical idea we owe in part to philosophers such as Aristotle and Hobbes, but most of all, to Turing, who made computation a tool with power sufficient to spur the founding of a new field: cognitive science. Rooted in a long history, two main currents shape cognitive science: both computational, but only one following a course apparently foreseen by Turing himself. In this essay we will explore these two streams, focusing on the key question: what is the relation between meaning and mechanism? More specifically: what is the mapping that links the meaningful elements of cognition—abstract concepts in the mind, and the knowledge that links them—to the mechanistic elements of computation—basic units of data, and the operations that act on them?<sup>1</sup>

For our purposes, we can take computation to be the *reduction of unboundedly complex processes to combinations of simple ones*. A *computational architecture* is thus defined by (i) a set of simple, *primitive* processes that are assumed given; (ii) a general type of data that these processes operate upon; and (iii) a finite set of operations that join primitive processes to create more complex processes which they then in turn combine to create still more complex processes.

Between them, the two streams of cognitive science deploy two classes of computational architecture. The first class comprises the *discrete architectures*, of which a key example is the Turing machine; other examples include Babbage’s Analytical Engine, Church’s function systems, Kleene’s algebras, and Post’s string-rewriting systems. This last example is closest to the discrete architectures central to cognitive science. The type of data employed is *symbol strings* (i.e., sequences), e.g.,  $(x+(2*y))=z$ . The primitive processes involve taking a string of symbols, say  $w=z$ , and substituting for one symbol, say  $w$ , a string of symbols, say  $(x+u)$ , producing  $(x+u)=z$ ; this simple process is a *rewrite rule*, denoted  $w \rightarrow (x+u)$ . Combining many rewrite rules, it is possible to compute complex sets of strings, such as the set of all well-formed equations of algebra, or the set of all well-formed programs in a computer language such as Java, or even a set of strings constituting an approximation to the sentences of English (where each symbol denotes an English word). In the latter case, the rules constitute a *rewrite-rule grammar* with rules like  $\text{Sentence} \rightarrow \text{NounPhrase}_{\text{subject}} \text{VerbPhrase}_{\text{predicate}}$ .

**1.** What is the purpose of computational reduction? Before automatic computers, clerks were employed to carry out extensive calculations. These human *computors* needed to be instructed in terms of simple operations they could reliably perform by hand or with basic mechanical aids, without relying on intuitions as a mathematician might. Reducing complex calculations to a sequence of simple operations allowed

---

\*Corresponding Author: Department of Cognitive Science, Johns Hopkins University, 3400 N. Charles St., Baltimore, MD 21218-2685, USA., Tel: (410) 516-5250, Fax: (410) 516-8020, Email: [smolensky@jhu.edu](mailto:smolensky@jhu.edu)

<sup>1</sup>Turing’s ideas concerning the formal details of computation and his ideas for relating machines to minds are both well-attested in his writings, but the latter are discussed at a level of generality that makes it difficult to interpolate how they were intended to be implemented in the former. The virtual absence of discussion of this mapping by Turing leaves it unclear whether it was simply taken for granted that this mapping is simple and transparent, as assumed by the symbolic paradigm in cognitive science introduced below. Thus the emphasis here will be on characterizing the conceptual relations underlying approaches to relating mind to machine in contemporary cognitive science, without attempting historical analysis. It is clear that Turing’s work laid the foundations for and is consistent with the symbolic paradigm, and it is clear that a fundamental principle of the alternative, subsymbolic paradigm—distributed conceptual representation—was either not considered, not considered important, or not considered correct by Turing.

these complex computations to be performed by human computers, who were using a mental faculty we can call the *conscious rule interpreter*. For human computers, the rules can be stated in English, using a circumscribed vocabulary referring to simple operations. It is this type of computational reduction that is formalized in discrete computation. And it is this type of reduction that Turing apparently envisaged for simulating the human mind.

Crucially, this type of reduction *conflates meaning with mechanism*. The complex process being carried out has the purpose of taking meaningful inputs (say, a target location) and producing meaningful outputs (rocket launch parameters). The mechanisms that perform the process operate on these meaningful elements: these are the operations described by the instructions that human computers follow.

This type of computational reduction defines the first stream of cognitive science: the *symbolic paradigm*. It is illustrated by what Haugeland (1989) terms ‘Good Old Fashioned Artificial Intelligence’, where programs consist of rules manipulating meaningful symbols that refer to the elements of the conceptual world in which the inputs and outputs reside. The same is true in the Newell & Simon (1972)/citeanderson1996architecture school of cognitive science, which models novice geometry students as literally internalizing the rules of their textbooks; with experience, these rules are modified to more efficiently achieve their effects, but remain, even in the expert, procedures (now unconscious) that manipulate symbols meaningful in the problem domain (symbols that refer to points, lines, triangles). When Turing (1950) discusses the *Argument from the informality of behavior*, he addresses the kinds of ‘laws of behavior’ that later developers of ‘expert systems’ would seek: propositions stated in terms of the concepts experts use to cognize their domain, concepts which are formalized as symbols that are meaningful in this sense and, simultaneously, are the tokens manipulated by the mechanisms of computation, which operate on the propositions encoding the identified laws of expert behavior.

In the symbolic paradigm of cognitive science, then, the type of computational reduction afforded by discrete computation is deployed for not only the conscious rule interpreter internal to the mind of the human computer or novice geometry student—it is also invoked for the *intuitive processor* in the mind of the expert geometer, a processor that delivers inferences independently of consciously accessible justifications. The intuitive processor of the symbolic paradigm is performing logical inference via operations manipulating meaningful symbols, just as the conscious rule interpreter does: the differences concern only accuracy, efficiency and conscious access.

2. In the other approach to cognitive science—the *subsymbolic paradigm*—conscious rule interpretation is analyzable in the same terms as in the symbolic paradigm, but intuition is not (Smolensky, 1988). This is crucial because intuition includes the large majority of mental processes studied in cognitive science. Expertise resides in the intuitive processor, and we are all experts in perception, action, common sense, and language. If Turing’s (1950) discussion of a machine playing the imitation game, or of training an intelligent machine through verbal instruction, is applied to the human machine, the necessary command of language is a capability of the human *intuitive* processor, and if the subsymbolic paradigm is correct, such a machine cannot operate within the confines of a discrete computational architecture operating on meaningful symbols.

For in the subsymbolic paradigm, the intuitive processor is formalized with a type of computation falling outside the class of discrete architectures. In a *continuous architecture*, data is numerical, and the primitive processes are arithmetic operations; data changes over time according to a differential equation, which combines multiple operations. The class of continuous computational architectures includes Thomson (Lord Kelvin)’s (1876) mechanical integrator, Bush’s (1931) Differential Analyzer, Pour-El’s (1974) theory of analog computation, Blum, Shub & Smale’s (1989) theory of computation over the real numbers, as well as the continuous neural network models of Amari (1977), Hinton & Anderson (1981), Grossberg (1982),

Hopfield (1984), Kohonen (1984), Rumelhart & McClelland (1986), and many others.<sup>2</sup> In these latter networks, the data consists in  $n$  real numbers  $a_1, a_2, \dots, a_n$ ;  $a_k$  is called ‘the activation value of the  $k$ th unit (or neuron)’; together, they specify a point in  $\mathbb{R}^n$ , the ‘activation vector’  $\mathbf{a}$ —a ‘pattern of activity’. A typical differential equation is  $da_k/dt = -a_k + f(\sum_j W_{kj}a_j)$ , where  $f(x) \equiv [1 + e^{-x}]^{-1}$  and each parameter  $W_{kj}$  is called ‘the connection (or synaptic) weight from unit  $j$  to unit  $k$ ’.

Because of the universality of discrete computation, any continuous computation can be approximately simulated with discrete computation<sup>3</sup>—but the crucial point is that then *the symbols manipulated must refer to activation values and weights*, not to meaningful concepts and to rules that relate them. In the subsymbolic paradigm, *mechanism operates below the level of meaning* (Hofstadter, 1986), whether the mechanism be given in its most natural formalization as continuous computation, or in its discrete-computational approximation. Here is why.

According to the subsymbolic paradigm, in the intuitive processor, a concept is not represented as a symbol governed by the rules of a symbolic program: *a concept is represented by an entire activation vector*, which is governed by a differential equation that operates on individual activity values, each of which is but a small part of a vector that is meaningful. This is *distributed conceptual representation*. In the symbolic paradigm, the concept COFFEE is encoded as a symbol which appears in propositions encoding knowledge of the relation of this concept to others. But in the subsymbolic paradigm, COFFEE is encoded as a vector. Changing the numbers in this vector can yield the vector encoding the concept SCONE. Individual numbers in these vectors do not correspond to concepts about the world of gustatory delicacies to which the computation refers: only the entire pattern of numbers constituting the activation vector  $\mathbf{a}$  has such a conceptual interpretation. A discrete program simulating the continuous computation that defines the processing occurring within this subsymbolic model will have symbols that refer to ‘the fifth activation value’, not to CREAM.

Indeed, in the McCulloch & Pitts (1943) discrete calculus inspired by Turing (1936), the basic elements are propositions that refer to the binary active/inactive status of a node in a network. If such a predicate is equated with an “idea”—a proposition over *concepts*—we are in the symbolic paradigm.<sup>4</sup> A network is but a notational variant of a complex proposition or equation; networks provide a notation that is often more convenient for describing parallel computation than is the string-based notations that are so well suited to sequential computation. The distinction between network- and string-based notations is largely orthogonal to the distinction between symbolic and subsymbolic cognitive models. It is true that conscious rule interpretation has a sequential nature (computers execute only one operation at a time), so lends itself well to string-based notation, while intuitive processing does not have an overtly sequential character. Intuition is modeled by unconscious sequential computation in the symbolic paradigm, but by (unconscious) parallel computation in the subsymbolic paradigm; this is reflected in the propensity of theorists to write the former in string- and the latter in network-based notations. What is crucial is that, whatever the notation, the primitive operations of the computational architecture operate on conceptually meaningful elements (symbols) in the symbolic paradigm, but on sub-conceptual elements (activations) in the subsymbolic paradigm.

---

<sup>2</sup>Continuous computation includes some, but not all, of what is called ‘parallel distributed processing’, ‘connectionism’, or ‘neural networks’; Turing’s own work on network machines (Turing, 1948) falls in the discrete computational class, like the work of McCulloch and Pitts discussed below.

<sup>3</sup>The questions under consideration here concern how best to model the internal structure of mental processing; issues concerning digital vs. analog computability are not particularly central and can be put aside.

<sup>4</sup>As mentioned in note 1, it is not clear what mapping between node activations and concepts was imagined for the network machines introduced by Turing or by McCulloch and Pitts (their “ideas immanent in nervous activity”). If they envisioned something other than a transparent mapping under which each node encodes a conceptual proposition, then they seem not to have discussed the formal structure, or any of the implications of, a non-transparent mapping. (Nor the combinatorial explosion of nodes entailed by a ‘node = proposition’ mapping. A population of  $n$  nodes each with  $v$  discriminable values of course provides not  $n$ , but  $v^n$  patterns for potential use as representations.)

3. Why have a significant proportion of cognitive scientists adopted a computational architecture deploying continuous mechanisms that operate beneath the level of meaning?

The subsymbolic paradigm is motivated by both the human mind and the human machine. Regarding the mind, during the 1970s certain psychologically-oriented cognitive scientists became frustrated with the rigidity of the symbolic paradigm for capturing human mental processes. Their theories called for ‘partially active’ words during sentence processing, ‘spreading activation’ to yield flexible associations between concepts, and, ultimately, learning procedures accumulating quantitative degrees of associations between sub-conceptual properties of experience from which emerge over time the coordinated aggregates which function as concepts, like Hebb’s (1949) cell assemblies.

Regarding the human machine, it has long been an important part of the mind-body problem to connect the mental to the physical brain, and computational reduction offers the first prospect for carrying this out rigorously. This requires, however, that the computational architecture deployed has primitive operations, data, and combinators that a brain could provide. And this is what the continuous architectures employed in the subsymbolic paradigm achieve, according to our current best understanding of the appropriate level of neural organization: a mental concept is encoded in the activity of many neurons, not a single one, and a given population of neurons can host any of multiple patterns encoding multiple mental concepts; the activity of a neuron is a continuously varying quantity at the relevant level of analysis; combination of information from multiple neurons has an approximately linear character, while the consequences of this combination for neural activation has a non-linear character that imposes minimum and maximum levels. The primitives provided by the continuous architectures of the subsymbolic paradigm are within the capabilities of the brain—that the brain has yet *more* complexity than assumed in these architectures does not compromise the subsymbolic paradigm’s computational reduction of mental to neural processing.

While brain theory is far from achieving a settled state, this conception of continuous neural processing has generally displaced earlier notions according to which the relevant level of analysis was taken to be one where neural activations are binary (firing/not-firing), as assumed by the early discrete-computational network descriptions of the brain developed by Turing (1948) as well as McCulloch & Pitts (1943). Similarly, early on, the search for the meaning of neural activation targeted individual cells, but recent years have seen an explosion of research in which neural meaning is sought by recording ‘population codes’ over hundreds of neurons, or patterns of aggregated activity over millions of neurons (functional Magnetic Resonance Imaging, fMRI).

4. The implications of distributed representation—of taking vectors in  $\mathbb{R}^n$  as the basic data type of the computational architecture—are many (Smolensky & Legendre, 2006). The basic operations of continuous mathematics provided by vector space theory now do the fundamental computational work. Consider, for example, language processing, a key domain for cognitive science. Instead of stringing together two conceptual-level symbols to form Frodo<sub>subject</sub> lives<sub>predicate</sub>, we add together two vectors, the first representing ‘Frodo as subject’ and the second ‘lives as predicate’. The vector encoding ‘Frodo as subject’ results from taking a vector representing Frodo and a vector representing ‘subject’ and combining them with a vector operation called the tensor product. The basic mapping operation of vector space theory, linear transformation, provides the core of mental processing. Vector similarity, as conventionally defined using the vector inner product, serves to model conceptual similarity, a central notion of psychological theory. And a notion from dynamical systems theory proves to have major implications as well: the differential equations governing many subsymbolic models have the property that as time progresses, a quantity called the *Harmony* steadily increases. Harmony can be interpreted as a numerical measure of the *well-formedness* of the network’s activation vector with respect to the weights. The objective of network computation is to produce the representation (vector) that is maximally well-formed—*optimal*.<sup>5</sup>

---

<sup>5</sup>Definitions: addition,  $[\mathbf{a} + \mathbf{b}]_k \equiv a_k + b_k$ ; tensor product,  $[\mathbf{a} \otimes \mathbf{b}]_{kj} \equiv a_k b_j$ ; linear transformation,  $[\mathbf{F}\mathbf{a}]_k \equiv \sum_j F_{kj} a_j$ ; inner

On a practical level, aspects of this continuous computational theory enable analysis of the conceptual meaning of measured neural activity patterns; this is now becoming pervasive in cognitive neuroscience (e.g., fMRI). And on a theoretical level, a nascent theory of vectorial computation is starting to tackle the kinds of computability questions initiated for discrete computation by Turing and his contemporaries.

5. The symbolic and subsymbolic paradigms conflict as regards the modeling of intuitive *processes*: the former, but not the latter, posit mechanisms manipulating conceptually meaningful elements. Nonetheless, with respect to mental *representations*—even within the intuitive processor—it is possible to achieve a degree of integration between the two paradigms by embedding discrete representations within a continuous vector space. Imagine the continuous space of all activation vectors as a Euclidean plane ( $\mathbb{R}^n$ , with  $n = 2$ ). Imagine that stuck into the plane are a set of flags, each bearing a symbol string. So at a particular point  $\mathbf{x}$  there is a flag labeled  $b$ , at another point  $\mathbf{y}$  a flag labeled  $ab$ , and at  $\mathbf{z}$  a flag bearing  $aab$ . The symbol strings  $b$ ,  $ab$ , and  $aab$  have been *embedded in*  $\mathbb{R}^n$  at the vectors  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{z}$ .

A function  $f$  over symbol strings (e.g.,  $f : b \mapsto ab, ab \mapsto aab, \dots$ ) can, for a significant class of functions, be achieved through a basic subsymbolic function, a linear transformation  $\mathbf{F}$  over  $\mathbb{R}^n$  (i.e.,  $\mathbf{F} : \mathbf{x} \mapsto \mathbf{y}, \mathbf{y} \mapsto \mathbf{z}, \dots$ ).<sup>6</sup> Such strings might represent conceptual structures, in which case the *function computed* by such a subsymbolic system has a perfectly precise description within the symbolic paradigm—employing symbols which individually have conceptual meaning—but the *process* by which this function is computed does not. Returning to language processing, this yields a theory in which the syntax of a native speaker can be specified by a symbolic grammar, but the mental processes that manipulate syntactic mental representations can be specified only subsymbolically (in terms of connection weights derived from the symbolic grammar).

Such a theory of mental grammars has led to a new subsymbolically-grounded theory in a cognitive domain long central to the symbolic paradigm: universal grammar. Recall that in many subsymbolic models, processing is optimization: it produces representations that have maximal Harmony (well-formedness). The Harmony of the representation  $\mathbf{z}$ —the location of the flag bearing  $aab$ —can be taken to be the *grammatical well-formedness* of the symbolic representation  $aab$ . According to such a *Harmonic Grammar*, the grammatical strings are those with maximal Harmony (Legendre, Miyata & Smolensky, 1990). And the Harmony of  $aab$ , it turns out, can be computed in terms of *constraints* like ‘a cannot follow b’: a string (like  $baa$ ) that violates this constraint incurs a Harmony penalty  $-s$ , where  $s$  is the strength of that constraint in the grammar. In actual natural language grammars, an example constraint is ‘no plural subject for a singular verb’ (violated by dogs barks).

Previous approaches to human grammar (Chomsky, 1965) are primarily based in discrete computation theory: a grammar is a set of rewrite rules which provide a step-by-step set of instructions—suitable for a human computer—for constructing grammatical structures. That is, grammars are specified as *processes*. Harmonic Grammars are specified instead in terms of *products*: a product of language processing is grammatical iff it optimally satisfies the grammar’s constraints; a process for generating such optimal products is not specified by the grammar, but is left to a separate theory analyzing how to carry out optimization over such constraints. The new contribution to the theory of universal grammar derives from the following strong hypothesis: the constraints are the same in all human grammars—only their strengths vary from language to language. The empirical success of this hypothesis can be summarized: what is universal across languages is found in the products, not the processes, of language generation.

Constraint-based approaches to grammar can also be pursued with discrete computational architectures. Indeed it is a striking discovery of recent years that the Harmonic Grammars of human languages have a strong tendency to display a special property: the strength of any given constraint is greater than that

---

product,  $[\mathbf{a} \cdot \mathbf{b}] \equiv \sum_k a_k b_k$ ; Harmony,  $H(\mathbf{a}) \equiv \frac{1}{2} \sum_{k,j} a_k W_{kj} a_j + \sum_k h(a_k)$ .

<sup>6</sup>For a fully recursive function over strings of unbounded length, we need  $n = \infty$ , but nonetheless  $\mathbf{F}$  can be finitely specified.

of all weaker constraints combined.<sup>7</sup> This entails that all that is needed to determine grammaticality—optimality—is the *ranking* of constraints from strongest to weakest within a particular grammar. This is *Optimality Theory* (Prince & Smolensky, 1997, 2004), which is the theory that actually introduced the strong universality hypothesis in the form: the constraints in all grammars are the same; only their relative ranking varies across languages. In Optimality Theory, grammars *specify functions* in discrete computational terms; but considered as part of the subsymbolic paradigm, the human mental *processes* that actually compute these functions must be specified using continuous computation—optimization—over Harmonic Grammars realized in vectorial distributed conceptual representations.

6. If cognition is computation, we must ask, what are the primitive computational elements, and how do they map onto cognitive entities? For the cognitive faculty of conscious rule interpretation, the computational primitives are the symbol-manipulating operations of discrete computation, with individual symbols mapping onto individual concepts. Mechanism operates on meaningful elements. For intuitive cognition, the same holds—according to the symbolic paradigm of cognitive science. In the subsymbolic paradigm, however, the computational primitives are the numerical operations of continuous computation, and a concept corresponds to an entire vector. Mechanism operates on individual numbers, activation values, beneath the level of meaning. Subsymbolic computation reduces complex mental processes to simple brain processes. Vector space theory provides tools now widely used for conceptual interpretation of recorded activation patterns in the brain. Dynamical systems theory provides tools for interpreting subsymbolic computation as optimization. Applied to language, this leads to a theory of grammar in which what is universal is the optimality-defining criteria for evaluating the products of language processing—as opposed to the process of producing these representations, previously the subject matter of mainstream grammatical theory.

The universe of computation opened up to us by Turing includes not just the discrete class of architectures, but also the continuous class; not just symbolic, but also vectorial representation of concepts; the means to formalize grammatical knowledge not just as procedures for computation, but also as criteria for evaluating products of computation. Undoubtedly, the universe of computation holds other unexplored architectures for creating machine intelligence and for understanding human cognition.

---

<sup>7</sup>More precisely, the Harmony penalty resulting from a single violation of a given constraint is greater than the maximal Harmony penalty that can result from all weaker constraints combined.

- Amari, S. (1977). Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, 27, 77–87.
- Blum, L., Shub, M., & Smale, S. (1989). On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21, 1–46.
- Bush, V. (1931). The differential analyser, a new machine for solving differential equations. *Journal of the Franklin Institute*, 212, 447–488.
- Chomsky, N. (1965). *Aspects of the theory of syntax*. MIT Press.
- Grossberg, S. (1982). *Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control*. Reidel.
- Haugeland, J. (1989). *Artificial intelligence: The very idea*. MIT Press.
- Hebb, D. (1949). *The organization of behavior: A neuropsychological theory*. Wiley.
- Hinton, G., & Anderson, J. (1981). *Parallel models of associative memory*. Lawrence Erlbaum Associates.
- Hofstadter, D. (1986). Waking up from the Boolean dream, or, subcognition as computation. In D. Hofstadter (Ed.), *Metamagical themas: Questing for the essence of mind and pattern* (pp. 631–665). Bantam Books.
- Hopfield, J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America*, 81, 3088–3092.
- Kohonen, T. (1984). *Self-organization and associative memory*. Springer.
- Legendre, G., Miyata, Y., & Smolensky, P. (1990). Harmonic grammar—a formal multi-level connectionist theory of linguistic well-formedness: Theoretical foundations. *Proceedings of the Cognitive Science Society*, 12, 388–395.
- McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5, 115–133.
- Newell, A., & Simon, H. (1972). *Human problem solving*. Prentice-Hall.
- Pour-El, M. (1974). Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Transactions of the American Mathematical Society*, 199, 1–28.
- Prince, A., & Smolensky, P. (1997). Optimality: From neural networks to universal grammar. *Science*, 275, 1604–1610.
- Prince, A., & Smolensky, P. (2004). *Optimality Theory: Constraint interaction in generative grammar*. Blackwell.
- Rumelhart, D., McClelland, J., & the PDP Research Group (1986). *Parallel distributed processing*. MIT Press.
- Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11, 1–23.

- Smolensky, P., & Legendre, G. (2006). *The harmonic mind: From neural computation to Optimality-Theoretic grammar*. MIT Press.
- Thomson, W. (1876). On an instrument for calculating the integral of the product of two given functions. *Proceedings of the Royal Society London*, 24, 266–275.
- Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42, 230–265.
- Turing, A. M. (1948). *Intelligent machinery: A report by A. M. Turing*. National Physical Laboratory.
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59, 433–460.