J Indian Philos (2007) 35:487–520
DOI 10.1007/s10781-007-9025-5

Positional Value and Linguistic Recursion

John Kadvany

Computation and Natural Language

The confluence of linguistic and mathematical thought in ancient India provides a unique view of how modern mathematics and computation rely on linguistic and cognitive skills. The linchpin of the analysis is the use of positional notation as a counting method for ancient and modern arithmetical procedures. Positional notation is a primary contribution from India to the development of modern mathematics, and in ancient India bridges mathematics to Indian linguistics. Pāṇini's grammar, while not thought of as mathematical, uses techniques essential to modern logic and the theory of computation, and is the most thorough-going historical example of algorithmic and formal methods until the nineteenth century. Taken together, modern logic and ancient algorithmics show how computation of all kinds is constructed from language pattern and use.

To set the stage we start with the contemporary idea that all kinds of mathematics can be thought of as sets of formulas or sentences expressed in some formal language. Such sets are called *theories*, and are often thought of as being algorithmically generated by some precise rules of proof, such as the rules of predicate logic applied to domain-specific, or ''non-logical,'' axioms with specially defined terms. So there are theories of arithmetic based on axioms for addition and multiplication; set theories based on axioms for set membership and formation; theories of the real numbers; various kinds of geometry, algebra, and so on. Today such proof systems can also be thought of as computations, which mainly means spelling out the details by which an

J. Kadvany (✉)
Principal, Policy & Decision Science,
1070 College Avenue,
Menlo Park, CA 94025, USA
e-mail: john@johnkadvany.com

axiomatic theory is constructed using a formal language and its grammar, much as is done for modern computer programming languages.

Such a computational perspective is used here to examine Pāṇini's creation of a formal system and the algorithmic power afforded by positional notation. Because Pāṇini's grammar models spoken Sanskrit, and the grammar is designed for oral expression, a unique nexus is created between computation and natural language. It is rarely noticed that Pāṇini's formalism, while intended only to describe the Sanskrit of his time, uses techniques which can be directly extended to represent any proof or computational system. Indian positional notation, effectively the same as we use today, in turn had an important linguistic representation through positional Sanskrit *number words*. Hence the algorithmically powerful tool of positional notation is rigorously expressed in a natural language described by a formal grammar expressed in almost the same spoken language.

The analysis will show how mathematical recursion, like that found in any computing language, but also just the basic arithmetic used throughout mathematics, can be constructed from weaker recursive patterns typically found in all natural languages. The formation of the Sanskrit positional number words as compounds can be seen as an increase in computing power *and* a type of language change: from a version of Sanskrit with only non-positional number words to one with positional number words. The language change[1] is modeled as a *grammaticalization*, in this case a construction for positional notation using Sanskrit number word compounds. By extending Pāṇini's grammar to a computational system, the construction can be assessed in terms of the algorithmic strength associated with positional and non-positional computations. The computational increase is due in part to cognitive skills through which the new language pattern is created from the old. This intentional capability is generally obscured by the role for writing in modern formalisms and models of computation. Pāṇini's grammar makes it possible to use a formalized *spoken* language as a computational model, leading to conjectures about the cognitive basis for modern computation and mathematics.

The next two sections review features of Indian mathematics[2] and linguistics needed to develop this computational and linguistic framework.

Algorithms, Positional Value, Sūtras

Algorithms are a ubiquitous product of Indian mathematics from ancient times onward. They are often the primary goal and not an incidental by-product. There are algorithms for geometrical constructions, and arithmetical operations including multiplication, division, and finding roots and

[1] On grammaticalization and language change see Aitchison (2001), Barber (2000), Brinton & Tragugott (2005), and Deutscher (2005).

[2] On Indian mathematics see Datta & Singh (1935), Emch et al. (2005), and Keller (2006); see Bag (1975), Nayar (1975), Ifrah (2000), Woepcke (1863) on positional number words.

powers. There are solution procedures for many basic algebra problems, and alternative algorithms for arithmetical operations, such as multiplication or finding roots, have computational advantages depending on how number signs are laid out and manipulated on a dust board. Difficult algebraic problems are solved by rapidly converging algorithms, and in some cases problems are solved with optimal or nearly optimal efficiency. By the medieval era there are procedures for computing trigonometric and other functions as recursive arithmetic approximations, much like power series today, e.g. $\sin(x) = x - x^2/(3 \times 2) + x^5/(5 \times 4 \times 3 \times 2) - \ldots$. These procedures provide important examples of what Alan Turing (1936) called *computable* real numbers, showing the close connection between arithmetic, the discrete infinite, and higher mathematics.

In all this there are unambiguous and conscious conceptions that algorithmic method involves processes of symbolic manipulation. Numeric and other symbols must be initialized in proper relative positions and then new symbols introduced, replaced or erased with reference to an informal layout of lines, boxes, or other perceived formats. Multiplication was described using metaphors of ''killing'' or ''destroying,'' such as *hanana, vadha, kṣaya.* Those terms referred to steps in which numerals were successively rubbed out, or ''destroyed,'' and then replaced. Such descriptions apparently were introduced along with positional numerals and are amply justified by a variety of algorithmic methods. Arithmetic was *pāṭīgaṇita,* compounding *pāṭī/board* and *gaṇita/science of calculation.* Calculations of all kinds were carried out on dustboards of limited size, making erasure and replacement an essential tool of memory management. These procedures are only meaningful with respect to coherent views and practices of symbolic behavior. Indian algebra was differentiated from arithmetic by its use of symbols whose values were indeterminate and unknown, as opposed to determinate and known values like 21 or 303. Bhāskara II around 1150 described algebra as analysis (*bīja*) assisted by letters or symbols (*varṇa*), whose properties are discovered through considerable intellectual effort.

Symbol usage is a special human skill to which we will return in the formation of positional number words. It is fundamental to the modern idea of algorithmic thinking as *generic* procedural exactness. Not only written or spoken numbers, but words, discrete sounds, bricks, weavers, beer jars or any other discretely defined entities can be treated symbolically, with their manipulation or processing by counting or other algorithmic methods characterized as a symbolic procedure. The modern idea that algorithms can be defined through different events, behaviors and media appears in ancient India outside of mathematics in procedures for ritual design and execution, including means for the recursive combinations of chants, marches or offerings in a larger composite ritual. Correct execution, especially in a mostly non-literate culture, required systematic means for efficiently describing a hierarchy of activities and options for their sequential expression. The ritual rules did that, implying what we can call today the formation of potentially infinite ritual patterns using finite means.

For our purposes, what matters is familiarity in India with algorithmic methods, especially recursive ones, across a range of media and behavioral

goals. That familiarity implies mastery of symbolic processing in alternative forms. A fire altar is made out of bricks as the only practical material. But an ''altar'' is defined by its role, as is a chess board, whose pieces can be made two feet high for games played in a public park. Neither the altar nor the chess board mean anything, but they have symbolic roles when their functions are defined through assigned relations and rules which dictate their use in the context of some relevant task. The ease with which we today treat mathematics and computation as symbolic manipulation arose much later in Europe than in India, and is often associated with the ideas of François Viète, Simon Stevin and René Descartes. These mathematicians began the interpretation of mathematics as a symbolic language without special reference to an idealized domain of number objects. Lacking such realist and platonist philosophies to overcome, mathematical expertise with symbolic heuristics was for centuries a dominant feature of Indian mathematics. Algorithmic expertise was also widely expressed outside of mathematics, narrowly conceived, just as occurs today.

Positional Value

Premodern Indian mathematics is generally bereft of proofs. The upside of that is a thoroughly constructive mathematics, meaning that mathematical facts get demonstrated through replicable procedures for carrying out concrete calculations. For that, a ubiquitous tool used throughout Indian mathematics is linear positional notation. In 2025, the sign 2 means 20 and 2000, and in 202 it means 2 and 200. Linear means that a positional numeral $a_1 \ldots a_n$ can represent 10^n values with just n symbols $\{a_i\}$ being used, each one a sign for $0 \rightarrow 9$.

The Romans and Greeks also had notations in which value was determined by a multiplicative rule. At times Roman $\overline{\text{M}}$ meant M *multiplied* by the ''bar'' factor of 1,000, two bars meant multiply by 1,000,000 and so on; the bar could also be applied to a composite symbol like CL. Such notation is ''quadratic,'' in that up to n^2 bars and base numerals (I, V, X, etc.) may be required to code 10^n values, depending on how units and bars are chosen. The reduction of an exponentially growing list of values to a quadratic number of symbols is a significant improvement over additive notations, like Roman numerals lacking a multiplicative bar or similar device. But multiplicative Roman numerals can require clumsy ''stacks'' of bars to code the same information carried by linear position. That difference leads to considerable streamlining for the simplest arithmetical computations and especially the recursive construction of new algorithms. For example, Indian algorithms for finding roots use the equality $a_1 \ldots a_n = a_1 \times 10^{n-1} + \ldots + a_{n-1} \times 10^1 + a_n$ to sequentially process the numerals $a_1, \ldots, a_n$ making up $a_1 \ldots a_n$ and whose (square or cube) root is to be calculated.

More generally, Indian algorithms use tabular and multi-line layouts to describe arithmetic calculations over finite lists of inputs or repeated transformations of equations. Input data and intermediate products in positional form can literally be fit into the bounded and semi-standardized computing

area created by a dust board. Two-dimensional tables alone are a great boon to arithmetic procedures, as shown by the impressive layouts of Babylonian scribes, who also discovered a linear positional notation. But inserted or replaced elements have to be compact enough for storage. It can be easy to insert a short list $a_1 \ldots a_n$ into a table ''cell,'' and columns and rows can recursively list such lists. But it is hard to use sizable or disorderly tables as table elements, and even harder to recognize in them patterns suggestive of higher-order rules. With positional notation, algorithmic steps involving symbol manipulation, erasure, and replacement become practical, especially for algorithms creating large outputs from small inputs. Erasure and replacement make up the ubiquitous ''assignment'' operation of modern computation. A programming step may assign $N \leftarrow N + 1$, meaning the *value* associated with variable N is to be replaced by the value now *stored* for N and incremented by 1; that value then is placed in the *storage location* for N. Replacement and erasure are not trivial features of dust board use; they reflect the necessity to manage memory and computing resources directly in algorithmic design, and many Indian algorithms do just that. A compact base notation for which storage requirements are minimal, and physical replacement as simple as possible, is therefore of the highest value. We will see below how replacement rules play a central role in Pāṇini's grammar, and describe much stronger links to modern computation.

Arithmetic and Computation

A relevant question is to ask what computational power has been made accessible through efficient arithmetic for + and ×. For example, $\sqrt{2} = 1.41421\ldots$ or $\pi = 3.14159\ldots$, and most other numbers needed for applied mathematics, are computable real numbers calculated by approximating series. So given just + and ×, *and* means for combining them into new algorithms, one can consider whether *all* computable numbers can be so defined, or whether additional basic operations are needed. A surprising fact from modern logic is that + and × are sufficient to define algorithms for *all* computable real numbers in decimal form. So + and × are not ''merely'' arithmetic. Properly conceived, they can be used to construct some quite advanced mathematics and as found in the proto-calculus and infinite series of premodern India.

Indeed, formal systems for just addition and multiplication are sufficient to define *universal computation*, meaning that *all* algorithms for some domain, say the natural numbers, can be listed and computed by a *single* algorithm expressed in a fixed formalism. So, famously, Alan Turing defined the universal Turing machine $\boldsymbol{U}$, really just a list of lists, so that $\boldsymbol{U}(m, n)$ equals the value, if one exists, computed by the mth Turing machine with input n. Similarly Kurt Gödel implicitly showed in 1931 that axiomatic theories strong enough to represent basic properties of addition and multiplication could represent all computable functions through a single master formula involving just +, × and logical symbols. All models of computation or provability have similar means for constructing a master formula or analogous representation.

Universal computation represents the theoretical computing power of a typical programming language, meaning the range of functions the language can be used to represent and calculate.

The arithmetic of addition and multiplication therefore makes large progress toward universal computation. The algorithms and their properties have to be discovered, and no notation does that. But by facilitating efficient arithmetic for + and ×, positional notation makes intelligible algorithms which are otherwise almost impossible to formulate and then compute. Much of this paper is about this rapid unfolding of algorithmic power through such simple arithmetic operations.

Universal computation is an important modern idea not present in Indian algorithmic method, whether in mathematics or linguistics. Indian mathematics lacks a formalism for algorithm expression, while Indian linguistics used formalisms without a role for anything like universal computation. Toward the goal of using Pāṇini grammar to formalize algorithms or axiomatized mathematics a final observation is needed about the expression of Indian mathematical knowledge in spoken Sanskrit.

Sūtra expression

Indian mathematical algorithms are often expressed as spoken verse, similar to rule-like knowledge for ritual execution, moral and personal guidance, prosodic analysis, and linguistic description. Hence it is tempting to identify the sūtra style as a generic means for procedural expression. But the sūtra form is properly a communicative style for procedural thought. Many mathematical and linguistic sūtras express exact procedures in summary form, but sūtras themselves are not especially rule-like. Even in Indian mathematics, sūtras are condensed mnemonics, at times nearly cryptic. Their brevity and poetic formulation aids memorization and recitation, especially when learning many by heart. Writing materials were always scare in India, so recitation guided by the sūtra form plays a material role in preserving and communicating knowledge. Typically sūtra codification requires explanation by a teacher, fellow-student, or other adept; that process therefore sustains considerable traditions of exegesis, commentary and authorization of interpretations. The knowledge required to execute an algorithm, even just for simple arithmetic operations, is carried by individual memory, with procedural exactness being triggered by, for example, actual dust board usage. That includes the ability to perceive physical symbols, gesture to them, identify symbolic patterns, isolate working areas, indicate erasure and replacement procedures, and comment on inscription technique. The sūtra is an entryway to that internalized process, but is much less than the whole.

With sūtra content so curtailed, it can be natural to minimize its importance for the expression of mathematical algorithms. But what matters for us is the identification of spoken Sanskrit as a standard of rigor. Orally transmitted knowledge faces considerable challenges to accurately preserve meaning, and one solution is to standardize the underlying language using explicit rules.

The relevant formalism is Pāṇini's grammar, which has no direct bearing on the *content* of mathematical algorithms. But Pāṇini's grammatical rules were codified using modern techniques for formal language definition, so the sūtra formulation creates a shared space for computation, formal description, and Sanskrit as a natural language.

The passage between mathematics and language is aided by Sanskrit positional number *words*. These words are needed for compact sūtra expression of large numbers, just as ordinary positional numerals are needed for algorithm execution on bounded dust boards. Such words must respect formal standards for word formation, even as they are defined by a multiplicative rule unlike most others in typical natural languages. But that is to get ahead of our story. The next goal is to identify just how to express mathematical algorithms of any kind using Pāṅini grammar as the underlying formal approach.

Pāṇini Grammar

Pāṇini's grammar,[3] composed perhaps sometime around the fifth century BC, and likely the product of generations of Sanskrit scholars, is in some ways the most powerful and thorough-going expression of algorithmic method in the ancient world. Our interest lay in gross features of Pāṇini's methods making this claim true, rather than his approaches to specific linguistic phenomena. With the benefit of hindsight, surprisingly little needs to be added to Pāṇini's techniques to make them generally useful for representing any algorithm or axiomatic system, analogous to the use today of first-order logic to axiomatize most any mathematical theory, or a Turing machine to model a specific algorithm or a class of algorithms.

Our view is that Pāṇini grammar is a *formal* system in *oral* form, intended to produce grammatically correct Sanskrit words and sentences along with their correct pronunciation. It is the first formal system, well before Gottlob Frege's late nineteenth century *Begriffsschrift* for first-order predicate logic. Similar steps are used to construct ancient and modern formalisms, in spite of differences of media and purpose: an oral formalism for spoken Sanskrit versus a written or graphical formalism for a logical or other formal calculus. Pāṇini's grammar has a finite symbolic basis defined by the Sanskrit phonemes, verb roots, noun stems, and word affixes; these are the initial inputs for the many types of rules ultimately generating Sanskrit expressions. The finitely specified grammar of nearly 4,000 sūtras is also recursive, with derived verbs or nominals re-input to formation rules, especially through word compounding. In modern terms that implies the potentially infinite use of finite resources, a primary motivation for generative grammars. The entire apparatus is primarily an analytical and descriptive device for which no psychological reality is claimed. *Saṃskṛt* itself, prefixing the root *kṛ* with *saṃs*, just means ''well-formed'' or ''well-constructed.''

[3] On Pāṇini grammar see Brough (1951), Sharma (1987), Staal (1972, 1988).

Because Sanskrit is strongly inflected, has a rich case structure, and has powerful means for constructing word compounds, Pāṇini's focus is on word, rather than sentence, derivation. A derivation is created by selecting roots and stems needed for a word or sentence, and then selecting and applying rules which lead to the expression of some action or event in terms of syntactic features such as tense, aspect, number, and several Sanskrit cases. The *sandhi* rules use the phoneme set to adjust pronunciation as affixes are adjoined to roots and stems, and in compound formation. No master rule enumerates or guides word derivations. The grammar's user decides how to start a derivation based on some intended meaning. *Kāraka* rules provide means to make that start, but their use is not automatic. These rules indicate how to mark words, or pre-words, to represent types of action or thematic roles, as in *Rāma is piercing the deer in the forest for Sītā*. Once that preliminary framework is created, a more automatic process dictates the formation of utterable expressions. Stem and root annotation, using specially designated Sanskrit sounds as markers, ensures that correct relations of syntactic agreement, reference and other conditions are carried along in the derivation of arbitrarily complex word and sentence forms.

As in modern logic, considerable effort is expended distinguishing the ''mention'' of actual Sanskrit and the ''use'' of a formal grammar to describe and generate it. Pāṇini constructed a formal metalanguage for linguistic description by extending ordinary Sanskrit; this formalism then expresses the algorithms used to derive spoken expressions. For modern logicians the analogous task is organized through definitions of *symbol sets, formulas, axioms, proofs*, and *theorems*; or computational categories such as *commands*, *subroutines*, *modules* and *programs*. That task is considerably simplified through writing and informal mathematical language. Because Pāṇini's system was orally expressed in his extended Sanskrit, the same tasks had to be accomplished directly through language structure. Yet the steps are much the same. Defined terms called *saṃjñās* can designate categories of words or expressions, including those created by particular rule applications. Once named, these categorical definitions are available to construct new rules. Rule application and derivations are guided by metarules, *paribhāṣās,* distinguished from ''operational'' rules generating Sanskrit expressions proper. For example, metarules govern the introduction of new technical terms, rule ordering in Pāṇini's listing, and the resolution of rule conflicts. Modern logic takes for granted our ability to introduce symbols, organize them in lists, and combine the symbols into varieties of defined categories, all before much of interest is done with domain-oriented rules. Pāṇini and his predecessor grammarians had to complete such tasks orally in a methodologically coherent way.

Pāṇini's formal rigor should be distinguished from the intricate process of rule application. His many rules are context-dependent and subject to multiple, prioritized interactions. As noted, no procedure exists, short of checking all rule options, to find the individual rule leading to a target expression, assuming such a rule exists. That situation can be typical for generative grammars. But it is undesirable for mathematics or computation, which

require a more orderly and ultimately rule-governed approach for rule application itself. Nonetheless, Pāṇini's approach to rule formation demonstrates the theoretical generality of his techniques.

To start, just one kind of ''event'' occurs in rule application and derivations: to *replace* one expression E by another E', reflecting additions, changes, deletions, placements, or sound changes. E could be a starting set of roots and stems, or an intermediate product, like a compound, created by prior rule applications. Indian mathematical algorithms use erasure and modification to replace one dust board computation with another; here, parallel methods are similarly universal, with replacement meaning that E' follows E in recitation, or if expressed in written Sanskrit, produced on a succeeding line. A derivation is a sequence of such replacements, starting from roots, stems and guiding kāraka roles. From Pāṇini's perspective, the challenge is to control derivations so that they lead only to grammatical Sanskrit. That means, for example, maintaining case agreement between distinct words, or insuring that a word remains plural while other changes occur, or triggering a sound change when an affix is adjoined to some previously derived word form.

A key device for derivation control is Pāṇini's use of Sanskrit sounds as artificial metalinguistic affixes. The markers identify syntactic features or linguistic constraints, and guide derivations forward when referenced by relevant rules. For example, to derive *Rāma is piercing the deer with the bow* (*Rāmo dhanuṣā mṛgaṁ vidhyat*), *Rāma* can be marked as the agent; *pierce* / *vyadh* is the verb of action; *bow* / *dhanuṣ* is an instrument; *deer* / *mṛga* is the object of piercing. That information is indicated by affixes *sU, Ṇi, Ṭā*, *am* and *LAṬ*, giving new expressions *rāma* + *sU, vyadh* + *LAṬ, mṛga* + *am, dhanuṣ* + *Ṭā*. Additional rules ultimately transform these into derived words including sound changes occurring in affixation.

The capitals represent ''indicatory'' or auxiliary sounds, called *IT*, which identify the relevant syntactic information. These controlling sounds get deleted while any remaining sounds might be combined with the stem or root. Another example is provided by the listing of Sanskrit phonemes in the *Śivasūtras*. Largely organized in terms of a sound's vocalization using the speech organs, *IT* markers are interpolated so that several sublists can be referenced by abbreviations in subsequent rule statements, something like $x_1,\ldots,x_n$ for a list starting with x_1 and ending with x_n. *IT* markers can also indicate defined categories, such as derived compounds, which then can be recursively referenced for further compounding or other uses. Pāṇini sometimes also uses ordinary Sanskrit case endings metalinguistically. A genitive case ending can indicate a term X to be replaced, a nominative ending marks the replacing expression Y, and a locative case marks an adjacent term or context Z; in modern terms these are the elements defining ''context-sensitive'' rules written $X \rightarrow Y/Z$.

Pāṇini has no mathematical or computational content at all, but all the tasks needed for constructing a modern axiomatic system or simple computing scheme appear in multiple mastered forms using ordinary resources of spoken Sanskrit extended by special markers and sounds. Hence to express formal

arithmetic or numerical algorithms using Pāṇini's methods, one approach would be to spell out the tasks to be performed, and then mimic Pāṇini's practices to create lists of symbols, categorical definitions, and rules and constraints, say those adequate to codify a standard formulation of first-order logic. ''Mathematics'' could be segregated from other language using new kāraka definitions to indicate a sentence involves ''computations'', ''numbers'', or ''proofs''. The process would also identify new formal symbols with Sanskrit sounds, affixes or words, so that the system, its computations, and proofs can be expressed orally.

Such an approach to formalization is sufficient, but there is a more powerful way of seeing that Pāṇini's natural language methods are sufficient to define computational processes generally. For Pāṅini has mastered a *single* algorithmic technique which is sufficient for such generic representation. The size and complexity of Pāṇini's system makes that accomplishment hard to recognize, but it provides a direct link to contemporary computation and formalization. In modern terms, Pāṇini's replacement rules are instances of formalisms studied by Emil Post in the 1930s and 1940s and which are used today for programming language design. A Post production system $\boldsymbol{P}$ is defined by a finite *alphabet* $A = \{a_1,\ldots,a_n\}$ of discrete symbols; *rewrite rules* $\{r_i\}$ defined on symbol lists, such as r: $aX \rightarrow aaX$, meaning that any expression starting with an a can be replaced by itself with another a prefixed to it; and *axioms* Ax used to begin rule application. A possibly infinite ''language'' $L_{\boldsymbol{P}}$ is generated by applying the rules recursively starting with the axioms.

In general a rule can be of the form r: $g_0X_1g_1X_2g_2\ldots X_ng_n \rightarrow h_0Y_1h_1Y_2\ldots Y_mh_m$. The g's and h's are *fixed* strings of individual symbols from the finite alphabet A, including null strings. Each X_i is an arbitrary *variable* string over A. The Y's can be any of the X's, including repetitions, so that $\{Y_i\} \subseteq \{X_i\}$. Post has no linguistic rules, just algorithmic ones, and Panini's grammar as a whole is poorly thought of as a Post system *simpliciter*. But the modern notation $E \rightarrow E'$ is not anachronistic for Pāṇini's rule application, nor is the notion of a succession of rule applications by which a produced expression is derived. As needed for Post's approach, Pāṇini can identify lists of constant expressions, variable expressions, and their mixture. Like a Post system, Pāṇini's goal is descriptive: to ''write out,'' to speech, grammatically well-formed Sanskrit expressions. Pāṇini's grammar is essentially a set of replacement rules based on a finite ''alphabet''—ironic as that term may be for an oral culture—and conclusions can be drawn from Post's work not following from the many other modern approaches to computation. The reason is Post's rediscovery of Pāṇini's method of auxiliary markers.

Given a production system $\boldsymbol{P}$ generating language $L_{\boldsymbol{P}}$, Post showed that $L_{\boldsymbol{P}}$ could be replicated using rules in a standard format by using auxiliary symbols. Post proved that any $L_{\boldsymbol{P}}$ could be produced by a system $\boldsymbol{P}^*$ with an alphabet A^* including A, and whose rules all have the form $gX \rightarrow Xh$, with g and h fixed strings. Since A^* extends A, $\boldsymbol{P}^*$ produces more than just $L_{\boldsymbol{P}}$, which only uses symbols from A. Post showed how to insure that the expressions produced by $\boldsymbol{P}^*$ which use *only* symbols from A will be *exactly* the expressions

L_P The situation is that faced by Pāṇini: a series of potentially infinite patterns is to be reproduced by a finite set of replacement rules. Post showed it was possible to do that in complete generality using new auxiliary symbols in $A^*{\sim}A$ (i.e. in A^* but not A) to control the production of exactly L_P over A. The challenge in Post's proof is eliminating auxiliary symbols once they are used to create expressions from L_{P^*}, so that only the target expressions L_P remain. The required replacements of the form $gX \rightarrow Xh$ are even suggestive of affixing, through the patterns of fixed strings appended to the start or end of any expression X.

It follows that if a grammarian can define rules using the method of auxiliary symbols, and apply them in rules of the form $gX \rightarrow Xh$, he or she can in principle reproduce the expressions defined by *any* Post production system in Post's canonical format. Post's methods also show that many variant systems can be simulated, particularly axiom systems with multiple antecedents in their production rules, such as $\{A,\ if\ A\ then\ B\} \rightarrow B$. It is similarly possible to construct markers and productions which mimic the state-transition logic of Turing machines. Post production systems therefore have the same algorithmic power as Turing machines or any other standard computational model. The same is therefore true for Pāṇini grammar taken as a paradigm for system construction. Post's results show that Pāṇini's method of auxiliary markers is a generic method for the simulation of any production system at all, but expressed orally using Sanskrit sounds as formal symbols. Pāṇini did not just create *a* formalism. He discovered a technique adequate to represent *any* typical formal system with a finite basis and recursively defined rules. His remarkable accomplishment was to see that the devices needed to accomplish that are already present in natural language. That shows, as a corollary, how formal systems of virtually arbitrary computational or axiomatic power can be directly constructed from mastered speech. Such artificial languages are the continuation of natural language grammar by its own means; their usefulness and ''generosity'' should be understood as extensions of otherwise typical natural language skills.

Having accomplished the goal of showing that arbitrary computational or axiomatic systems can be codified using Pāṇini grammar, we now return to Indian mathematics and the expression of numbers in spoken Sanskrit using positional number words. Pāṇini's apparatus will be used to extend the number word model to a formalized[4] ''oral arithmetic'' below.

Positional Number Words

Sanskrit positional number words are one of several means used in Indian mathematics for oral number representation. Other examples include Piṇgala's use, around the third century BC, of n short and long syllables in

[4] On formalized arithmetic see Enderton (1972); Fischer & Rabin (1974) and Young (1985) on bounded multiplication in additive arithmetic; Boolos (1979) on provability predicates. Davis et al. (1994) and Minsky (1967) discuss Post production systems. Davis (1965) includes Post's papers.

poetic metre to enumerate values up to 2^n, with the syllables a proxy for counting in base 2. In the sixth century, the astronomer Āryabhaṭa used Sanskrit phonemes, non-positionally, to represent values up to 10^{18}. Like these techniques, positional number words are used for codifying numbers in speech rather than writing, and all are mostly unusable for direct computation. However for centuries positional number words were a standard means for expressing numeric precision in verse form by mathematicians and astronomers.

For Sanskrit number words, choices can be made from ten *sets* of words for the basic numbers, $0 \rightarrow 9$. As an example, one of the oldest available Sanskrit calendar dates is expressed as *śrutīndriyarasa.* This word is built from *śruti.indriya.rasa* with ''.'' used here as a visual separator. Translated as *Veda, properties, senses* these stand for 4, 5, 6:

Veda.properties.senses
4 5 6

The number follows an early convention to give the least significant figure first, so that the date is actually 654 (= 732 CE). *Śruti* is 4 because it refers to the four *recited* Vedas. Here is an example of a transliterated number verse from the astronomical text *Sūrya Siddhānta* (''Sun Theory'' ca. 4$^{\text{th}}$ century CE):

candra ucchasya agniśūnyāśvivasusarpārnava yuge
vāmaṃ pātasya vasvagniyamāśviśikhidasrakāḥ

Here *agniśūnyāśvivasusarpārṇava* names 488,203 and *vasvagniyamāśviśikhidasra* names 232,238. The verse means ''The number of revolutions of the apogee of the moon in a yuga (a 4,320,000 year cycle) is 488,203, and of its waning node, 232,238.'' The Sanskrit positional word system is entirely rigorous and facilitates the easy formation of number words for recitation, even changing words for same digit inside a numeral, just as writers change vocabulary just to avoid repetition: instead of expressing 333 as *tri.tri.tri.* you could combine *agni.mūrti.loka*. Varied examples occur across centuries of Indian mathematics and astronomy. One approach was to also use names for *pairs* of digits, thus reducing the number of names needed. Mādhava, sometime around the Fourteenth century, for example, described 2,827,433,388,233, which approximates the product of π with 9×10^{11}, using individual words for the pairs 33 and 27, and single-digit number words for the rest.

Sanskrit number names are not unique, but they are unambiguous and efficient when used as intended. Hundreds of different number words can have the same value, and the larger the value, the greater the number of possible names. Such choices make it easier to find a euphonious and compact representation for any numeric value, which again is a primary motivation for the system. Some Sanskrit number words for $0 \rightarrow 9$ were shared across dialects, so some names became standardized and helped ensure accurate communication across a huge land. The number words are also redundant

compared to simpler symbols for $0 \rightarrow 9$, because small errors do not prevent the word name from being correctly understood. Contextual cues provided by word neighbors limit word endings and provide additional redundancy in the oral communication channel. Sanskrit number words are also appropriate in ritual contexts in which accounting numerals and even written language could be polluting. Ritual practices were described in verse using recursive patterns, so positional number words fit well in a tradition in which spoken Sanskrit efficiently expressed arbitrary precision for all kinds of activities.

For a system of positional number words to be useful, the word choices and background language need to facilitate word transitions implied by combining the basic number words used as the basis for the positional recursion. If the compound words are to be used poetically, morphological and phonological rules must facilitate euphonious and memorable combinations. English may be poor parent language to construct a positional number word system, but ancient Sanskrit is exactly suited to it. Sanskrit grammar facilitates complex word compounds whose pronunciation is governed by the sandhi rules smoothly joining one word with the next. All languages have such rules, but may be used more or less in casual versus formal speech, and may be more or less central for how the language is used as a whole. For Sanskrit, the many compounding and sandhi rules cover much of how the language functions. Like ancient Greek or Latin, Sanskrit is highly inflected, and also uses a rich case structure to mark grammatical relations. English has plural and possessive cases, as in *dog* $\rightarrow$ *dogs* or *he* $\rightarrow$ *his*, and we inflect regular verbs with *-ed* to form the past tense, as in *I buttered my bread*. But English is mostly ''isolating'' and non-inflectional, making syntax greatly dependent on word order and providing limited opportunities for complex composite word forms. These are just empirical facts about language differences and are matters of degree rather than kind. Modern English evolved from Old English which was highly inflected and allowed for considerable free word ordering, similar to Sanskrit and Latin. Over several centuries the elaborate case endings of Old English were lost, and today many individual prepositions and verbal auxiliaries have taken their place. Hence language modification, as occurred in the use of case rules and affixes for Pāṇini's grammar, and the use of compounding for number words, can be straightforward and is historically common. Indeed languages are always in flux, and in the next section positional number word formation will be characterized as a kind of language change.

Comparisons to English illustrate why the linguistic structure needed to construct multiplicative positional number words are not specific to Sanskrit. In English, 102 could be written or spoken as *holy.void.love*, or just *holy-voidlove*. But also, say, as *bodynullmarriage*. Because many word beginnings and endings are predictable (e.g. *ts* is a common ending but not a beginning) position starts and stops can be recognized without additional markers for parsing, just as additional markers are not used for usual positional notation. In English these compounds are not particularly mellifluous, easy to pronounce, or memorable. But that's only because English is no longer optimized to join word endings and beginnings so freely. English has many

phonological rules to change pronunciation when combining morphemes, like *dog* + *s* → */dogz/* but *cat* + *s* → */cats/* and not */catz/*. It just happens that in English, these rules are not obviously useful or easy to implement with number words like *holyvoidlove*. But given some labor, such number word rules could be made to work smoothly. Language design also makes it possible for number words to be treated as ''formal symbols.'' Phonemic recognition and production makes possible the ''duality of patterning'' characteristic of language systems: while meaningless in themselves, phonemes create symbolic contrasts through which larger morphemes are built and differentiated, as among */bad/, /dad/, /sad/, /mad/, /lad/*. Hence treating number words as ''merely'' symbols, as needed for formal description generally, is not especially logical or mathematical. As far as numeral valuation goes, number word semantics is based on a positional rule using the place of a word inside of the compound. Position is often used in natural language to determine syntactic roles and meaning—*Man bites dog* is news but not *Dog bites man*, with word order determining the subject and direct object—so number word meaning relies on a linguistic capability probably utilized in all languages. Notably, positional value is calculated using a count of where a sign appears in a larger symbol; such a ''parameterized'' rule contrasts with simpler recursive constructions, such as noun phrases built from other noun phrases, verb phrases built from verb phrases, or subordinate clauses built from subordinate clauses. Position is often used to determine meaning and syntactic roles, but grammatical rules typically do not rely on an increasing sequence of natural numbers as parameters. Yet that occurs here, showing how an advanced number concept can be constructed from generic natural language skills.

The function of Sanskrit positional number words, like the sūtra form, is communicative and not computational. Perhaps that is why the earliest expressions of the positional concept (without a zero) in India are as Sanskrit number words. The positional idea may have first appeared in number words around 200 BC to express very large numbers in compact form, but with no arithmetical algorithms. Number words are linguistically straightforward, and Pāṇini had described compounding rules of much greater complexity several hundred years earlier. Strictly speaking, a recursive positional rule would have to be added to Pāṇini's grammar to define number word semantics, and to mark an expression as being about a number rather than the moon, sun, or marriage. But given the prestigious role of language and linguistics, the positional number words must have been recognized as an elegant use of Sanskrit recursive compounding. For computation, the positional technique would have to be converted to orthodox written numerals along with the invention of calculation techniques for addition, subtraction, multiplication and division. Since the number words are still perfect for sūtras they were retained in that role.

The Indian origin of positional notation therefore *may* have been in spoken Sanskrit, but since number words have limited computational usage, there would have been an essential role for inscription to develop even the simplest arithmetic. Inscription is vital for mathematics because it overcomes major

limitations of short-term memory applied to rapidly fading speech. Inscription presents number symbols using visually perspicuous patterns in stable and effectively permanent media, making possible powerful illusions of autonomous computation. This undoubted benefit is due to writing's role as a natural language prosthesis whose extreme efficiency obscures the symbolic skills on which it relies. The Sanskrit positional number words, combined with Pāṇini's oral grammar, provide an opportunity for avoiding that filter with respect to mathematical computation. As noted, positional number words are made possible through general language skills which may or may not be universal: duality of patterning, the use of discrete units, a role for position as a symbolic marker, and recursive patterning in which counting is applied to linguistic products themselves. These features, even if present in writing, are just better understood when expressed in speech. ''Written forms,'' wrote Ferdinand de Saussure, ''obscure our view of language. They are not so much a garment as a disguise.'' Because positional value is computationally powerful, and because Pāṇini's grammar can be used in principle to formalize any algorithm, the positional number words provide a starting point for computations conceived as oral instead of written expressions. In that way, the linguistic expression of multiplicative recursion provided by positional number words can be extended to computation generally, thereby unfolding universal computation from natural language skills. Through such a thought-experiment, linguistic features of positional number words can also be compared to their associated computing strength.

To prepare the way, and as suggested by example of English language change above, the relevant linguistic feature of positional number words is their *formation* from arithmetically and linguistically simpler non-positional predecessors. To that language change we now turn.

Grammaticalization of Position

Word compounding is the structural feature facilitating the formation of positional number words and is the heuristic center of the Sanskrit language. The function of the linguistic discrete infinite in any language is to efficiently form arbitrarily complex expressions, so the use of compounding to represent numeric precision in ultracompact positional number words is not adventitious. Various compounding styles were highly mastered in Sanskrit and their grammatical roles would have to be a focus of any systematic treatment of the language. This does not mean that compounding, in contrast to other techniques used across the world's languages to produce new word and sentence forms, is particularly special as a recursive device. Only that it happened to be highly mastered in terms of poetic and scientific expressiveness, compactness, and mnemonic optimization. That mastery was also described in detail using formal grammatical theories expressed in extensions of oral Sanskrit. All that expertise creates a cognitive foundation for the representational redescription of *non*-positional compounds through a positional rule.

Sanskrit includes several simple compound types, analogous to, but more inclusive than, English forms like *seashore, horseback, wastepaper*. English also has *dvandva* compounds, in which nouns of some type are aggregated, like *Metro-Goldwyn-Mayer* or *murder-suicide*. A Sanskrit dvandva can concatenate any number of items, like English constructions using … *and* … *and* …, and that is the typical number word construction. Varied compound types can be further recursively combined, making possible single-word formations of arbitrary precision much like an English clause, with sandhi rules fusing phonemes at word boundaries for correct pronunciation. Thus there is an orderly procedure for expressing syntactically varied and arbitrarily precise ideas in a single compound, and in Sanskrit poetry very long or complex compounds are used to express rich, often polysemic meaning, with extreme concision. These powerful methods make grammatical analysis of compounds complex, but luckily, as an extension of Sanskrit compounding, positional notation is linguistically elementary. Number words are built by combining simple nominals, so the additional complexity generally associated with compounds does not occur. Unlike Sanskrit as a whole, which has a mostly free word order, word position *is* relevant for complex compounds, so the use of number word position to determine place value is a straightforward extension of standard usage.

What is remarkable about the positional words is the use of ordinal place in the number word as parametric data for a linguistic construction. The counted places in the word symbol itself—the ''1s'', ''10s'' position, and so on, as features of the compound form—are used to define number word meaning. For comparison, English syntax marks time or its passage through, among other devices, the past tense, verbal auxiliaries, and participles, as in *She proved the theorem, He will wash the car,* or *The argument is falling apart*. These constructions are useful because we can associate them with pragmatic conceptions of time and event occurrence. Different languages use grammar to mark different features, so while English happens to grammaticalize time using tensed verbs, Chinese does not, and so uses other means to express temporal distinctions. In some languages, aspects of a perceived scene, say whether a described object is directly visible or not, is grammatically marked; other languages use syntax to indicate the quality of knowledge expressed, such as whether one experienced an event directly or learned of it second-hand. Gender marking in aboriginal languages can codify cosmological classifications reflecting a conceptual model of culturally linked categories. Even the apparently simple use of prepositions, like a ring being *on* a finger, or an apple being *in* a bowl, rely on schemes for spatial relations: Korean prepositions, for example, can express how ''tight'' or ''loosely'' objects fit together, a relation not directly marked at all in English. Through the perception of linguistic form, positional number words rely on cognitive models as in other types of syntactic marking.

For number words, what gets perceived is language itself, described using an additive arithmetic which is more elementary than the multiplicative rule being defined. Additive number word constructions, as in *two million* (+) *three hundred thousand* (+) *four hundred* (+) *twenty five* are common in many

languages, perhaps because natural language syntax of many types can easily incorporate additive constructions. For large numbers, additive constructions also quickly become inefficient beyond the highest named unit, like *trillions,* and difficult for even modest calculations. The problem is that notation size eventually grows in proportion to the magnitude of the number named, making it necessary to count about as high as the number just to label it. Number systems can always be improved to overcome such difficulties, including the addition of higher number units. The genius of positional number words is that, like positional notation generally, higher number units are automatically generated by the positional rule. The exponential increase in number values so represented is perhaps the acme of Sanskrit compactification. Multiplicative rules are not typical of natural languages and the mere existence of positional number words in Sanskrit, regardless of computational inconvenience, signals an unusual transition between linguistic and mathematical recursion. The driver of multiplicative recursion is perceived data about the symbol to be interpreted, and that is the revolutionary breakthrough. Used perhaps in all languages to mark varied syntactic or semantic roles, position is here used to make a numeral's place a parameter for computing a number word's value. Instead of some folk model of spatial organization, the passage of time, the speaker's knowledge, or an ancient cosmology, positional notation mobilizes a cognitive model directed at the perceived layout of the symbol, guided by additive arithmetic already available in the language. The utilization of simpler arithmetic to count position also means that the conception is well-founded, i.e. not viciously circular.

For us there are two basic questions. One is how such an idea could arise as a linguistic and cognitive phenomenon. The second is what happens computationally when additive non-positional notations are improved to multiplicative positional notations. The remainder of this section addresses the first question, and the next two sections address the second.

The historian Franz Woepcke proposed in 1863 that positional number words originated as a linguistic improvement over non-positional number words. Woepcke's idea was that positional number words originated by recognizing redundant patterns specific to Sanskrit non-positional number words. Following are the building blocks for such a grammatical construction. In just what order the developments might occur is less important than their eventual confluence.

A first step is the use of Sanskrit names for powers of ten as number units. In English and many other languages, the creation of number units often proceeds in groups like *millions, billions, trillions*. For these, succeeding units increase by more than one power of the base. So a (US) billion is not 10 million, but 1000 million, and a trillion is 1000 billion. To create number words involving intermediate powers we say *one hundred and six million* or *sixty-five billion*. Lacking terms for intermediate powers, like 10^7 or 10^8, a ''hybrid'' name is used to construct the desired value. The approach is common and leads to additive systems mixing bounded multiplications within additions, dictated by the units skipping intermediate powers; these bounded

computations will return in the analysis below. The benefit of increasing by an increment such as 10^3 is that successive number words, such as *quadrillion* and *quintillion,* advance that much farther along the number line. A pattern like *-illions* can become psychologically salient with use, and it is easy to associate names with powers 10^{3n}. But that advantage can wrongly focus attention on a linguistic paradigm which does not easily generalize into *variable* powers. There is nothing wrong with skipping powers of ten, or any other rule. But if the underlying task is to automate the lexical formation of number units, such as *millions, billions, trillions*, then that iterative process of language change has itself to be suggestive of a new procedure.

Something different than the *million–billion–trillion* paradigm appeared in India before the positional breakthrough. First, individual powers of 10, like 10^{11}, 10^{12}, 10^{13}, sometimes with powers in the 100s, were given *arbitrary* names, rather than a following a linguistic *illions*-like pattern, or any linguistic pattern at all. In addition to the use of arbitrary names, there were names for *all* powers to around 10^{17}, with no powers skipped, as happens from *millions* to *billions*. Examples include *padma* (10^9), *kharva* (10^{10}), *nikharva* (10^{11}), *mahāpadma* (10^{12}), *shanka* (10^{13}), *samurda* (10^{14}), *madhya* (10^{15}), *antya* (10^{16}), *parādha* (10^{17}). Beyond 10^{17} there were names for various high powers (10^{140} = *asaṅkhyeya* is ''innumerable''), perhaps developed for special problems like counting the number of ''atoms'' in the universe, and similar to the problem of Archimedes' *Sand Reckoner*. His answer $[(10^8)^{10^8}]^{10^8}$ was a *myriad–myriad units of the myriad–myriad-th order of the myriad–myriad-th period.* In the third century *Lalitavistara*, the young Buddha is reported to have demonstrated his prowess by enumerating names for powers up to what we would calculate as 10^{421}, apparently without positional units. Because names of powers are arbitrary and numerous, they would initially be harder to learn than names following an *-illions*-type pattern. But the effort is compensated by hypercompact compounds suitable for versification and formed using ingrained compounding techniques. Assuming that many numbers actually needed fell below 10^{18}, a large magnitude for many purposes, the use of names for all individual powers below 10^{17} meant that up to this limit, numbers could be expressed uniformly using an additive rule and no ''hybrid'' multiplications. In English a bounded multiplication *five hundred million* is used because no name exists for 10^8. The Sanskrit compound could use the exact power needed, such as *five nyarbuda,* so to speak, for 5×10^8. With names for all the powers, say to 10^{18}, ''coefficients'' needed to express a number could be limited to $1 \rightarrow 9$, or $0 \rightarrow 9$ if zero was known. Using a combination of new and old to emphasize the way in which the compounds would be perceived, 3,682,439 could be additively expressed as

(∗) **9** and **3** ***daśa*** and **4** ***śata*** and **2** ***sahasra*** and **8** ***ayuta*** and **6** ***lakṣa*** and **3** ***prayuta***
9 3 4 2 8 6 3 (= 3,682,439)

The *and* (Sanskrit *ca*) would generally not appear because compounding eliminates it, and then sandhi rules join phonemes at word boundaries. The

and/ca is a reminder that the value uses multiplications by $0 \rightarrow 9$ and then additions; powers were also often listed in descending order. Woepcke's conjecture comes down to the perceptual saliency of number words like (∗), in which names for powers are expressed without complicating coefficient calculations. Because of the many substitutions possible, *arbitrary* names for high powers implies that names for number units take on the role of formal placeholders or grammatical ''slots.''

If a zero is needed, say for 205, the 10^1 place could be left unsaid, the verbal equivalent of the positional blank space used by Babylonians before their zero. Powers were arranged in ascending or descending order, and not listed freely like colors or animals, even though named powers make permutations possible. Hence the ordering of named powers would make it apparent whether a particular power was present or not. Uses of numeric zero occur in Piṅgala as early as 200 BC, but only centuries later does a zero appear explicitly in positional numbers. Hence it is unknown whether a zero-type coefficient occurred in the oldest number words, and probably did not. Since number words numbers were likely unused for calculation, ''skipping'' a coefficient had no computing implications. However, in reciting, say, 3670832, with a single skipped power, it is helpful to mark the zero power rather than saying nothing, just because the listener might guess a power was missed due to fatigue, inattention, or no expectation of a ''blank.'' That wouldn't be needed for 1000002, just as we say *one million and two*; but for 3670832 or 10101010 marking skipped powers is more useful. In this way, a spoken zero *may* have been devised to complete the positional mechanism. That only means using a word to mean none of a power, and the positional zero was eventually named by words like *śūnya*/void, *abhra*/sky and *ambara*/space, just as names for $1 \rightarrow 9$ were evocative of concepts, deities or other memorable entities. With multiple choices possible, names for zero too would also be perceived as formal coefficients just like names for $1 \rightarrow 9$.

The zero then is an additional innovation, but not more essential than the other positional building blocks: arbitrary named powers, their ordering, and avoidance of hybrid coefficients. With those mastered techniques, the multiplicative schema implicit in the non-positional number word system is ready for *grammaticalization* by a positional rule. That is the modern formulation of Woepcke's conjecture: patterned non-positional usage, as in (∗), can be reanalyzed into a rule which generalizes the information provided by the named powers and their coefficients. Since the names of powers of 10 are arbitrary, their role is just to fill a slot marking the power of ten. But then the named powers are redundant, since the power is also identified by its place in the compound: if *any* name is acceptable, *none* is essential. With no ''hybrid'' coefficients, names for $0 \rightarrow 9$ alone are effectively positional when interpreted as coefficients for powers of ten. The names for powers are candidates for linguistic grammaticalization because they merely fill variable slots dependent on an overpracticed compounding form. This is the cognitive means by which a perceived feature of the compound symbol could automate the potentially infinite formation of higher number units by finite means.

For comparison, another grammaticalization example is the formation of English auxiliaries, such as *will* as a future tense auxiliary, a bleached-out version of its earlier, Old English meaning as an ordinary verb of promise or intention. We can now say *the car will work after I fix it* or *my dog will fetch the bone*, without meaning that the car or dog wills or intends anything at all. The abstraction was a natural one based on general usage and a need to streamline future-oriented speech, especially given the loss of Old English inflections through which tenses were expressed. Similar changes created modal auxiliaries like *must, should, can*. Today these have special syntactic roles, like not taking direct objects, while they formerly behaved like ordinary verbs. The changes from content to function words occurred over many years, were not the product of conscious invention, and were influenced by other shifts in English usage, especially pronunciation. Such complexity and indeterminacy can be typical of grammaticalization, so that new functional patterns or words, like the many English prepositions, are the combined product of changes in sound, word, and sentence patterns. Similar conditions for linguistic change apply to number word formation. Sanskrit sound patterns are relevant because to eliminate named powers, it has to be possible to create euphonious number words using only names for $0 \rightarrow 9$. That is no obstacle because of the free choice of multiple number words for $0 \rightarrow 9$, and Sanskrit was already optimized for large, compact compounds. Similarly, word position needs to determine numeral valuation, but inside compounds position already was used to determine meanings of even moderately elaborate verse forms. It would be perceptually salient that the numerical information conveyed by number power words is also represented by word position in the sound pattern.

Therefore the change from (*) to a positional rule would rely on mental models like those accompanying the formation of *will, can* or *must* as function words, or other grammaticalizations. When language patterns get used to mark possession, tense, location, speaker attitude, or much else, the pattern is often motivated by some useful cognitive generalization. For tense that may be a useful conception of past, present or future, while for possession some notion of ownership or control. What typically influences grammatical change is some first-order experience like perceptions of time or place, features of the topic or object of interest, or the speaker's conversational role. With positional notation, language and compounded number words themselves are the perceived objects, principally through our linguistic behavior in structuring ever-higher number units. That behavioral process is abstracted in grammar by taking a number position to mark a higher number unit. One can literally observe, and describe in a rule, how a new number unit, as a power of 10, is constructed by appending additional individual number words to an existing number, and how the value of the new word is recursively determined from its subwords.

Such cooperation of pattern and perception is also characteristic of historical language change, and is here mobilized to create a new mathematical technique. An association is created between an idealized physical configuration of the numeral symbol and new counting procedures. There a transfer of function from the perception of symbols—described in terms of first position, last

position, symbol to the left, symbol to the right, symbol length, and so on—to arithmetical properties of numbers. In positional value the transfer occurs by making a variable feature of the number symbol a parameter of its interpretation. The Sanskrit number words just make it easier to see how the perception of symbols is transformed into an arithmetical abstraction which then facilitates more arithmetic, computation and mathematics. As with other cases of grammaticalization, the change is motivated by the same efficiency and increased precision mandated by the sūtra tradition. The representational improvement was spectacular, as decimal positional notation uniformly represents 10^n numbers using just n places. Pāṇini's grammar was likewise described as pouring an ocean into a cow's hoof.

We therefore have the desired linguistic and cognitive characterization of positional number words: they are a grammaticalization of an additive non-positional notation using arbitrary names for powers of ten. The next step is to assess the difference in computational power using additive and multiplicative rules.

Pāṇini Arithmetic

We now give, as a thought-experiment, algorithms, proofs and computations expressed in two natural language fragments which are also formal arithmetic theories. In one, the only arithmetical operation is addition, simulating the formation of non-positional number word patterns which ''look like'' positional numbers, but without the full multiplication needed to define a uniform positional rule. That weaker theory is called *additive Pāṇini arithmetic,* or $\mathbf{ADD}_{PA}$. It represents all additive notations and the computations possible with addition as the basic operation. The second theory is called *multiplicative Pāṇini arithmetic,* or $\mathbf{MULT}_{PA}$, which extends $\mathbf{ADD}_{PA}$ by adding multiplication axioms. That makes it possible to define number units, positions, powers, and other positional features not definable using only additive rules but which are needed to make significant use of positional numerals.

The procedure for creating these oral arithmetics is straightforward. We saw above that Pāṇini grammar can be used to represent any formal system using either auxiliary symbols or Pāṇini's methods for symbol lists, category formation and recursive definition. Here two modern formal theories are identified with oral representations in which the Sanskrit number word paradigm gets extended to an entire formal system. There are ''formula-words,'' ''axiom-words,'' ''proof-words,'' ''computation-words,'' and everything-else-words as needed for that purpose. Proofs and computations in $\mathbf{ADD}_{PA}$ and $\mathbf{MULT}_{PA}$ are derivations composed from these expressions, just as in modern logic or Pāṇini grammar, but relying on spoken rather than written language for symbol definition, formation and manipulation. Because these theories are thought of as directly expressed in natural language, we assume that computational comparisons between $\mathbf{ADD}_{PA}$ and $\mathbf{MULT}_{PA}$ reflect cognitive models at work in the grammaticalization of position and multiplicative rule

formation. The definitions of $\mathbf{ADD}_{PA}$ and $\mathbf{MULT}_{PA}$ make it straightforward to compare what can be computed or proved using only addition versus addition combined with multiplication.

Steps $(a) \rightarrow (e)$ below sketches the codification of the two theories as spoken Sanskrit. One can imagine a programming language in which basic symbols are identified with phonemes, morphemes, or words, and with other expressions generated by linguistic rules which happen to be computations. Such correspondences can be established by fiat, but as noted, we assume that the *formation* of a multiplicative rule requires cognitive and linguistic skills as described earlier.

(a) Add to Pāṇini grammar *symbols* needed for a first-order theory for arithmetic, such as $\oplus, \otimes, =, \neq, x, y, z, \mathbf{0}, \mathbf{1}, (,), ', \exists, \forall, \vee, \&, \rightarrow, \sim$. Similar to positional number words, these symbols are represented by selected or artificial Sanskrit words or other markers.

(b) Add rules defining categories such as *variables, constants, connectives, terms*, *formulas, equations,* and *sentences*. The categories are defined separately for $\mathbf{MULT}_{PA}$ and $\mathbf{ADD}_{PA}$ reflecting whether $\oplus$ and $\otimes$, or only $\oplus$, are used.

(c) Add rules identifying *non-logical axioms*. $\mathbf{MULT}_{PA}$ uses rules for $\oplus$ and $\otimes$, while $\mathbf{ADD}_{PA}$ uses only rules for $\oplus$. Rules for $\mathbf{ADD}_{PA}$ define addition recursively in terms of ''add 1,'' and multiplication is recursively in terms of addition in $\mathbf{MULT}_{PA}$. These following axioms for $\mathbf{ADD}_{PA}$ would be translated into Sanskrit word forms: $\mathbf{0} \oplus \mathbf{1} = \mathbf{1}$; $\forall x(x \oplus \mathbf{1} \neq \mathbf{0})$; $\forall x(x \neq \mathbf{0} \rightarrow \exists y (y \oplus \mathbf{1} = x))$; $\forall x \forall y \ (x \oplus \mathbf{1} = y \oplus \mathbf{1} \rightarrow x = y)$; $\forall x(x \oplus \mathbf{0} = x)$; $\forall x \forall y \ (x \oplus (y \oplus 1) = (x \oplus y) \oplus 1)$; $\forall x \forall y \ (x = y \vee \exists z(z \oplus x = y \vee z \oplus y = x))$; Induction schema for formulas $\varphi(x)$ using $\oplus$ and with free variable x: $\varphi(\mathbf{0}) \ \& \ \forall x (\varphi(x) \rightarrow \varphi(x \oplus \mathbf{1})) \rightarrow \forall x \varphi(x)$. The Induction schema ensures that $\mathbf{ADD}_{PA}$ is the strongest possible first-order additive theory and that any additive notation is definable in it. $\mathbf{MULT}_{PA}$ is defined by the $\mathbf{ADD}_{PA}$ axioms plus axioms recursively defining multiplication from addition: $\forall x(x \otimes \mathbf{0} \ \mathbf{0})$; $\forall x \forall y \ (x \otimes (y \oplus \mathbf{1}) = (x \otimes y) \oplus x)$.

(d) Add rules for *positional* and *non-positional number words* to fill roles usually played by symbols $\mathbf{0}, \ldots, \mathbf{9}$. Individual number powers and non-positional number words can be defined in $\mathbf{ADD}_{PA}$ because that theory allows for the arbitrarily high but bounded multiplications needed for (*). Number word synonyms can be created by additional axioms like ***śūnya*** $=$ ***abhra***, forming two names for 0.

(e) Add rules of inference for first-order logic using the above axioms for $\mathbf{ADD}_{PA}$ and $\mathbf{MULT}_{PA}$. The two theories are identified with the infinite sets of theorems provable from their axioms.

Rules $(a) \rightarrow (e)$ generate proofs and theorems from the given definitions, just as Pāṇini grammar generates grammatical Sanskrit as its target. Techniques for introducing symbols as in (a) are demonstrated in Pāṇini by his lists of phonemes and other primitives. Definitions such as (b, c) are illustrated by Pāṇini's defined categories which he also uses recursively. The positional

numbers of (*d*) were devised after Pāṇini, but can be defined as compounds marked as numbers. Positional number words only involve bounded multiplications with $0 \rightarrow 9$, and so can be included in $\mathbf{ADD}_{PA}$. But using them to multiply, calculate with exponents, or carry out multiplicative algorithms can only occur in a stronger system like $\mathbf{MULT}_{PA}$. Individual positional number words can be defined in the weaker system, but no rules which allow their non-additive features to be exploited for algorithm execution. These relations between $\mathbf{ADD}_{PA}$ and $\mathbf{MULT}_{PA}$ make the two theories in combination a good model for positional grammaticalization. The logical rules of (*e*) are modern but still algorithmic, and can be coded using auxiliary symbols. Grammarians could use such modern rules following the advice of ritual theorists that the original meanings of mantras need not be known for correct ritual execution.

The memory requirements to recite proofs or computations in oral arithmetic are considerable. But many theoretical models of proofs and computation also assume arbitrarily large memory registers, tapes, instruction tables, or symbol sets. $\mathbf{ADD}_{PA}$ and $\mathbf{MULT}_{PA}$ are just a different type of idealization. To make the theories closer to mathematical practice, they could be extended to describe dust board calculations. Rules could define a dust board in terms of cells, rows, and columns, and then dictate how data is entered, modified, or deleted from various relative or absolute locations as directed by oral ''commands.'' A procedure for converting phonemic information to dust board representations, such as standard numerals **0**,…,**9**, could be defined, with computations or proofs still spoken out as they are written. An abstract dust board is potentially infinite, but so are the compounds, sentences and other constructions for the oral language, or the semi-infinite tape and expressions of a modern Turing machine. Linguists as early as Pantañjali in the third century BC recognized that the point of a grammar was to codify linguistic generality using limited rules, so this idealizing assumption is not anachronistic. As noted above, even before the Indian linguists recursive ritual descriptions allowed potentially infinite behaviors composed of altar constructions, marches, chants, and oblations.

Where we really need a modern perspective is in the enumeration of all proofs or computations, as through (*e*) above; Pāṇini, recall, has no universal procedure. With that notion we can compare the two oral arithmetical theories in terms of their computing power.

Language Change in Pāṇini Arithmetic

$\mathbf{ADD}_{PA}$ and $\mathbf{MULT}_{PA}$ are good proxies for the computational power afforded by non-positional and positional notations respectively. The use of positional notation presumes multiplication and addition, so some minimal definitional properties of both operations have to be spelled out as in $\mathbf{MULT}_{PA}$.

$\mathbf{ADD}_{PA}$ can represent all *non*-positional notations, as these use only additive operations with individually defined number units, such as Roman V, X, or L, or Sanskrit number words for powers 10^n. New units can be used to

define further additive notations, and such continued unit formation is the process automated by a positional rule. The extent of additive combinations in $\mathbf{ADD}_{PA}$ is set by *bounded multiplication.* Formulas $Mult_n$ can be explicitly defined in $\mathbf{ADD}_{PA}$, meaning no additional axioms are required, to represent multiplication when one multiplicand is always less than a fixed bound: e.g. $Mult_n(x, y, z)$ correctly represents $a \times b = c$ as long as a is less than $2^{2\hat{}n}$ (with $2\hat{}n = 2^n$), with b arbitrary. Non-positional notations like $(*)$ can be defined by multiplications using bounded coefficients, say $0 \rightarrow 9$, multiplied against individually defined, arbitrarily large units. Since non-positional notations are built from bounded multiplications and additions, $\mathbf{ADD}_{PA}$ includes all of them. What cannot be arranged in $\mathbf{ADD}_{PA}$ is for powers 10^n to be defined for *variable n* instead of individually *fixed* numerals; similarly the formulas $Mult_n(x, y, z)$ are defined one-by-one in $\mathbf{ADD}_{PA}$ and not by a single master formula. In $\mathbf{ADD}_{PA}$, no formula can *uniformly* define positional numerals in terms of successively increasing powers, nor other basic properties making use of arbitrary positions. This limitation corresponds to the functional ''slots'' associated with number word grammaticalization.

Such uniform definition can be accomplished in $\mathbf{MULT}_{PA}$, one of the simplest systems in which positional *properties* can be defined. Positional numerals do not define multiplication, but they rely on it, even for such a basic notion as a ''number unit'' defined in terms of powers. Hence variable powers, and number units as arbitrarily high powers, represent an increase in computing power measured by sentences provable in $\mathbf{MULT}_{PA}$ but not its weaker subtheory $\mathbf{ADD}_{PA}$. Such provability is formally equivalent to computation. $\mathbf{ADD}_{PA}$ and $\mathbf{MULT}_{PA}$ are also natural language fragments, so this computing increment is introduced with the grammaticalization forming positional number words. Pāṇini's metalinguistic and derivational techniques rely only on affixing and other grammatical devices already used in Sanskrit, so everything in sight is a grammatical construction from natural language. Hence the computational improvements occurring in $\mathbf{MULT}_{PA}$ compared to $\mathbf{ADD}_{PA}$ should be attributed in part to cognitive capabilities making multiplicative rule formation possible.

The large computational increase from $\mathbf{ADD}_{PA}$ to $\mathbf{MULT}_{PA}$ is due to universal computation being representable in $\mathbf{MULT}_{PA}$ but not $\mathbf{ADD}_{PA}$. Axioms for $\mathbf{MULT}_{PA}$, including general rules for logical inference, imply a single formula $\boldsymbol{T}(x, y, z)$, using $\oplus$ and $\otimes$, enumerates all algorithmic computations. If $A(i) = j$ is an algorithm on the natural numbers (with *no j* for some *i* possible), there is a numeral $\boldsymbol{e}_A$ such that $\boldsymbol{T}(\boldsymbol{e}_A, \boldsymbol{i}, \boldsymbol{j})$ is provable in $\mathbf{MULT}_{PA}$ if and only if $A(i) = j$, where $\boldsymbol{i}$ and $\boldsymbol{j}$ are names for i and j. $\boldsymbol{T}(x, y, z)$ may be constructed from a simpler formula representing ''after w steps of computation x on input y, the computation gives result z or is not yet defined.'' This formula is not assumed, but can be constructed from the $\mathbf{MULT}_{PA}$ axioms using just first-order logic, while such representation is not possible in $\mathbf{ADD}_{PA}$. Thus surprisingly little is required in addition to Pāṇini grammar to represent all algorithms and their computations, showing how close the combined methods of Indian linguistics and mathematics are to modern

computation: Pāṇini grammar provides the formalism and positional notation provides the algorithmic power. Positional notation then is more than a ''convenient'' method of numeric expression. It is defined by two arithmetic operations, + and ×, which in some of their simplest formulations logically imply universal computation.

Further differences between $\mathbf{MULT}_{PA}$ and $\mathbf{ADD}_{PA}$ support the proposal that the increased representational power is attributable to intentional skills through which a multiplicative pattern is recognized and expressed as a rule. First, once × is included, as in $\mathbf{MULT}_{PA}$, functions or categories defined recursively from + and × can be explicitly defined without further recursive axioms. In $\mathbf{MULT}_{PA}$, but not $\mathbf{ADD}_{PA}$, the ''elimination of recursion'' makes possible explicit definitions for *formulas, derivations, proofs, computations* and related categories. Next, $\mathbf{ADD}_{PA}$ is *decidable*, meaning an algorithm $A_{ADD}(S)$ computes whether any additive S is derivable in $\mathbf{ADD}_{PA}$, say $A_{ADD}\ (S) = 1$ if S is derivable in $\mathbf{ADD}_{PA,}$ and $A_{ADD}(S) = 0$ if not. Formulas S can be coded in $\mathbf{MULT}_{PA}$ as natural numbers, and so A_{ADD} can be represented in $\mathbf{MULT}_{PA}$, but not in $\mathbf{ADD}_{PA}$. Hence a single formula in the stronger theory describes what can be proven in the weaker additive arithmetic. With A_{ADD}, $\mathbf{MULT}_{PA}$ can also prove the consistency of its subtheory $\mathbf{ADD}_{PA}$. Finally, $\mathbf{ADD}_{PA}$ is *complete*, meaning that for any sentence S limited to the additive language of $\mathbf{ADD}_{PA}$, either S or ~S is derivable in $\mathbf{ADD}_{PA}$. $\mathbf{ADD}_{PA}$ is usually called Presburger arithmetic for Mojzesz Presburger who proved its completeness and decidability in 1928. In contrast, in 1931 Gödel showed that many theories $\boldsymbol{T}$ which, like $\mathbf{MULT}_{PA}$, could represent provability and related notions, were always *in*complete: if $\boldsymbol{T}$ is consistent, a (self-referential) sentence G could be constructed such that neither G nor $\sim G$ was provable in $\boldsymbol{T}$. Finally, Gödel's theorem on the unprovability of consistency, applied to $\mathbf{MULT}_{PA,}$ depends on how provability is represented, making the result ''intensional''; such linguistic phenomena are typically symptomatic of intentional roles.

The power of $\mathbf{MULT}_{PA}$ comes at a price. In 1974 Michael Fischer and Michael Rabin proved that any algorithm A_{ADD} as described has a minimum complexity level: there will always be sentences S, with say n symbols, such that the calculation $A_{ADD}(S)$ takes at least 2^{2^n} steps. The formulas $Mult_n(x, y, z)$ above for bounded multiplication were discovered by Fischer and Rabin through this work. Thus $\mathbf{ADD}_{PA}$ is both algorithmically simple and quite complex, while $\mathbf{MULT}_{PA}$ is strong enough to characterize $\mathbf{ADD}_{PA}$ through a single formula for A_{ADD}.

Positional notations do not create universal computation, but the weakest multiplicative systems for which they are meaningful do. Conversely, the additive operations often possible in natural languages do not use a notion like variable number units, and their computational power falls correspondingly short. In this way, positional notations require cognitive models through which additive computations are schematized and made the reflective object of new computations. After all, to state relations among positional symbols requires abilities to identify and manipulate the symbolic parts in a symbolic whole. Even if expressed by a single stereotypical example, such as ''the 3 in 351 means

3 hundred, but the 3 in 3510 means three thousand,'' such skill is still required. Those reflective skills are ultimately also needed to construct an algorithm like A_{ADD} which completely characterize what is provable or not in $\mathbf{ADD}_{PA}$.

Here is another perspective on the cognitive prerequisites for multiplicative rule formation. Computation can use all manner of symbolic forms, including written or spoken numerals, logical formulas, computer programs, chess pieces, mantras, knotted ropes, body parts, or notched sticks. The fact that universal computation has such a simple basis has been discovered repeatedly through varied approaches to computation and provability. Options may be more or less efficient, or use different computing resources, but all lead to mathematically equivalent formulations. Since there is no mathematical bound to computational idioms, such equivalences represent an informal assumption about computation generally. That is summarized by the *Church-Turing thesis*: *all* effectively computable functions can be formalized by a Turing machine or many other formal computational models; no effectively computable function gets left out of the formalisms, all of which have equivalent scope, and have related properties of complexity, undecidability and incompleteness. In this vast computing universe, positional notation is a cognitive and computational aid for taming exponential growth in algorithm output and design.

Here is an *extended* Church-Turing thesis, intended to summarize the claim that all computation relies on cognition[5] and intentionality via symbolic representations. The extended Church-Turing thesis is that universal computing power always relies on cognitive skills at least as strong as those needed to construct positional number words as a grammaticalization of non-positional number words. That means that no media, particularly writing, completely removes the cognitive models used in language change and formation. Instead, the underlying intentional skills are redescribed through the mathematical idioms characteristic of universal computation. Stated positively, the capability to reflect on and transform language structure facilitates the unfolding of computational power from addition to multiplication and universal computation. For, starting with spoken Sanskrit, Pāṇini extended the language using its own resources into a formal descriptive metalanguage. The positional number words, with their multiplicative principle, also can be constructed in the spoken language via grammaticalization. Combining Pāṇini's method with this mathematical content and some additional rules gets us more or less continuously from natural language to universal computation. The artifice of Pāṇini arithmetic shows in small steps how to construct a formal calculus using spoken Sanskrit as the starting point. The intentional skills needed to construct the grammar, positional notation, and additional arithmetic also play a role in orthodox formalisms relying on writing or its

[5] On cognition, language and symbolic skills see Baron-Cohen (1997). Clark (1996), Gentner & Goldin-Meadow (2003), Tomasello (1999, 2003); on representational redescription see Karmiloff-Smith (1992). Turing's discovery is discussed by Gandy (1995).

derivatives, and where intentional capabilities are taken for granted, disguised, or otherwise occluded.

Positional notation in number word form is constructed by positing a single rule, much like an axiom, which generates a potential infinity of higher number units and positionally interpreted symbols. The concept is expressed through the recursive rule

$$(**)\ a_n \ldots a_0 = a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0,$$

its equivalent, or stereotyped positional examples. The grammaticalization model suggests that this symbolic pattern relies on the same intentional skills needed for language use—variously known as species-specific capabilities of ''mindreading'' or ''theory of mind'', upon which symbolic skills of many types depend and whose absence or impairment is even considered empirically diagnostic of autism. The formation of positional number words relies on these skills through grammaticalization just as in other instances of language formation. Since these skills facilitate symbolic manipulation generally, a natural conjecture is that they enable the increases in computing power associated with positional notation and related multiplicative procedures. Historically, expressions remotely like (**) do not occur until the modern era. But even informal expressions of positional rules, whether in India, or by Viète, Stevin, and other early moderns, are intended as multiplicative rules applying to potentially infinite symbolic expressions. Stevin, for example, advocated positional value to express decimal fractions, $.a_1a_2a_3\ldots$, which is just another uniform rule for interpreting a potential infinity of *decreasing* number units. What is necessary for the formulation of such concepts, no matter how expressed historically, are ordinary capabilities of linguistic recursion *plus* pattern formation relying on our powerful intentional skills. Mathematical recursion, in the form of multiplicative and higher arithmetic functions, is constructed from this linguistic and cognitive basis.

We now turn to the modern theory of computation with this combined cognitive and historical perspective in mind.

Computation and the Tunnel

Positional notation is the thread connecting ancient Indian mathematics and linguistics to contemporary computation. Not just because of the notation's immense usefulness, but because of another sibling role between mathematics and linguistics. Regard for generative models in linguistics has arced high and low since their introduction by Noam Chomsky in the 1950s. But regardless of one's views on language and how to model it, Chomsky's[6] generative methods revived techniques used two millennia earlier by Pāṇini, who is even noted on

[6] On Chomsky's mathematical background see Chomsky (1963), Chomsky & Miller (1963), and Pullum & Scholz (2006); on his cognitive theories and their influence see Chomsky (1980) and Gardner (1987).

the opening page of *Aspects of the Theory of Syntax*. Chomsky wanted to create quasi-axiomatic models for our remarkable linguistic creativity in terms of the potentially infinite, recursive use of finite resources. Today, many want more than a descriptive grammar like Pāṇini's, with claims often made for a grammar's cognitive or biological reality. But it is still striking that basic recursive formalisms should be rediscovered thousands of years after their first appearance. Using Frits Staal's apt term, a metaphorical ''tunnel''[7] reaches from ancient Indian linguistics to the modern theory of formal languages. The algorithmic methods seem almost to disappear and then emerge a half-century ago. This unusual recurrence in the history of ideas came about because of the role played by positional notation in the history of modern mathematics and mathematical logic.

How Chomsky discovered his entrance to the tunnel is clear. He was expert in the new methods of mathematical logic, including Post production systems which he acknowledged as a source for the rewrite rules paradigm. But Chomsky's ''Cartesian'' intention was always to use generative grammars as models of cognition. Over the years he has proposed various ''universal'' grammars meant to parameterize or otherwise codify innate options for human language syntax. The project is thoroughly anti-intentional in that the use of language for communication is taken as mostly irrelevant to language structure, acquisition or competence. Hence Chomsky's mentalist goal was to model cognition, just minus intentionality, using ''Galilean'' computational models of language competence and structure. Grammars described mental representations whose processing did not fundamentally rely on intentional skills. The approach was buttressed by arguments that natural language patterns could not be explained probabilistically nor in strict behaviorist terms. The anti-behaviorist alternative was that mental representations could be rule-driven generative processes acting on discrete symbols, certainly a radical perspective at the time for psychology and linguistics.

A modern basis for this cognitive paradigm came not from Emil Post's rewrite rules but Alan Turing and the computational model now called Turing machines and mentioned repeatedly above. In 1936 Turing published his approach to mathematically precise effective procedures, ''On Computable Numbers With An Application to the Entscheidungsproblem [decision problem].'' For decades reaching back to the nineteenth century, logicians had devised many kinds of formal or neoformal systems: ones for elementary arithmetic and geometry, like those of Giuseppe Peano and David Hilbert; axiomatizations for all of mathematics such as Zermelo-Fraenkel set theory or Russell and Whitehead's *Principia Mathematica*; Gottlob Frege's first-order predicate logic; simple propositional logics; and several others. These systems often demonstrated that some body of mathematical knowledge could be codified as a certain type of axiomatic or rule-based system, and many relied on similar procedures involving a recursive definition of a formal language using generative rules. Hence another motivation, aside from the search for

[7] The ''tunnel'' is from Staal (2006) and ''grammarians of reason'' below from Goody (1987).

mathematical foundations or novel proof techniques, was to characterize what a rule-governed ''effective procedure'' should be at all. In spite of intense attention to rigor and many alternative approaches, a troubling issue by around 1930 was the absence of an accepted mathematical characterization for the elementary procedures underlying the construction of formal systems themselves, regardless of their specific content or philosophical interpretations. The consistency of outcomes using different formalisms, and a growing toolkit of robust methods, suggested some core idea of effective procedure which remained mathematically elusive. Post's production systems and Alonzo Church's lambda-calculus were part of this quest, and both are essential today to the theory of programming languages. Turing's proposal, formulated to solve a problem in mathematical logic set earlier by David Hilbert (and solved independently by Church just before Turing), was ultimately recognized as clarifying and grounding the inchoate concept of rule-based effective procedure.

Turing's approach uses a thought experiment about language use. He imagines a computer, as human calculators were known, following a set of precise instructions for calculating some arithmetic function using pencil and paper: a statistical table, predicted tides, equation solutions, logarithmic or trigonometric values, and so on. Turing wanted to describe the computer's actions generically using elementary operations of symbolic manipulation, such as the erasure and replacement of a single sign. The computer could work with any finite set of symbols written to a potentially infinite paper tape. Hence Turing assumed no practical limitation on the computer's memory or resources to carry out a calculation, as long as the procedure itself was finitely specified and fixed. Like Post's analysis, Turing's applies to the generic construction of potentially infinite computations, proofs, grammatical expressions, or other symbolic products, using finite resources. Turing's rules are organized as a table of quintuples $\langle S, s, T, t, right/left/same \rangle$ using, in addition to the finite symbol set, a finite set of auxiliary ''states'' which control the task being carried out: if you the computer is in state S and reading symbol s on the tape, then (i) erase s and replace it with symbol t (possibly s again or a blank); (ii) set the new state to T (possibly $= S$); (iii) read the next symbol one discrete tape ''square'' to the *right, left,* or in the *same* position. Then (iv) find the quintuple applying to the new state-symbol combination for T and the next symbol being read; and finally (v) recursively proceed with the instructions just described. Turing's quintuples and ''states'' control the products of his systems much like Post's and Pāṇini's auxiliary symbols.

Turing's approach was considered definitive, principally as advocated by Gödel, because his procedures were perceived as quasi-mechanical processes for manipulating symbols of any type without regard to intended meaning. No genuine sets, platonic ideas, abstract numbers, or other entities needed to be assumed as real interpretants of the symbols defined by their functional role in computations, proofs, productions, or whatever else is generated. Most saw such actions as mechanistic, and intentional skills clearly are vastly reduced and regimented. As far as the scribe must identify instances of

discrete symbols as identical or different, and apply rules organized as table entries, some basic cognitive skills are still at work. Turing regarded auxiliary states as proxies for the computer's ''state of mind,'' and Saussure's advice to mind the ''disguise'' of writing and its apparent autonomy is here well-taken. Notwithstanding such tacit knowledge, it was judged that symbolic writing played no special role in Turing's thought-experiment. It did not matter *mathematically* whether symbols were represented as written or in other forms, how the instruction table was laid out, or instructions executed. Ultimately this judged independence of symbolic media and behavior became the distinction between software and hardware. That interpretation was taken up by cognitivists following Chomsky's lead, for Turing's model suggested that rules could be realized by some unspecified biological mechanism. Prima facie, Chomsky had a profound alternative to the behaviorist rejection of mental phenomena involving symbolic processing.

Whether that approach to linguistic skills is ultimately right or wrong, the variability of computing media and symbol formation is a deeper aspect of the tunnel going beyond the formal correspondences of Pāṇini's, Post's and Chomsky's rewrite rules. Pāṇini's expression of his grammar as oral sūtras is a marvelous illustration that computational procedures can be *of* what you like *in* what media you like. His grammar is the greatest example of a rigorous computational system in non-written media before the electronic age; as shown earlier, it can even be conceived as generating an arithmetical and proof system via additional production rules and auxiliary symbols. As far as content goes, Turing's scribe does not have to know what her symbols mean, just as Vedic ritualists[8] claimed that mantra interpretation was unnecessary, only procedural correctness.

Turing's great insight was that his description for the operations of a generic computer *T* was itself an effective procedure. The description of computation as recursively processing quintuples ⟨*S, s, T, t, right/left/same*⟩ could itself be codified using a Turing machine description, demonstrating that the new computing idiom could be used as its own metalanguage. Neither Pāṇini, Gödel nor others achieved this metalinguistic task with such transparency. Turing's *universal machine* ***U*** instructs a single scribe like other Turing machines, taking as tape input the instruction table of any Turing machine *T,* plus any input *I* to be calculated by *T*. Turing machines read linear lists, not tables, so the data is laid out with auxiliary markers, much like Pāṇini's *Śivasūtras*. ***U*** simulates *T* by following the directives of *T*'s table applied to *I*, using marked-out work areas to track *T's* ''current symbol,'' ''current state,'' and intermediate calculations. Turing could quickly and negatively solve Hilbert's decision problem using the universal machine. Today, modern computing languages like Fortran, C, Basic or Java are used in an environment of compilers, interpreters, and operating systems which function together as a universal machine. Programming rules are configured as data, and are interpreted and executed using a single master procedure, just as Turing envisioned.

[8] On Vedic ritual and Sanskrit grammar see Staal (1983, 1990) and Renou (1941).

Universal computation returns us to the tunnel and Chomsky's use of generative formalisms to represent natural language syntax. Natural language complexity is not a reflection of computational strength, but rather a mosaic of semantical and syntactic dependencies. So for language description, ancient or modern generative grammars are much *weaker* than needed for universal computation. Full-blown ''transformational'' grammars can represent universal computation, and so were seen as too powerful for representing natural language structure. That observation is consistent with our perspective that positional notations and multiplicative procedures are just beyond the recursive forms typical of natural languages. But Chomsky's systems *followed* the formalization of algorithmic processes by Turing, Gödel, Church, Post and others. So the generative program had to *limit* the computational power inherent in the mathematics motivating various approaches to formalization, whether in pure logic, set theory, arithmetic, or the generic systems studied by Turing and Post. Chomsky himself constructed a hierarchy of grammars below universal computation which is still fundamental to programming language theory and design.

In contrast, none of the pioneers in mathematical logic would have questioned the role of multiplicative and more complex algorithmic procedures in mathematics of any kind. By the late nineteenth century, great controversies surrounded the infinite, methods of proof involving the real numbers, and the scope of logic. But arithmetical algorithms were taken as an ubiquitous and essential feature of algebra, geometry, analysis, number theory, and other specializations. Such mathematics became possible through several innovations. One was the generalization of algebraic techniques to all kinds of mathematical objects defined at varying levels of abstraction. Also needed were axiomatic methods and refined proof methodology. And another ubiquitous technique is positional notation. A modern theorem selected at random will have in it or some nearby lemma a role for multiplicative arithmetic applied to integer dimensions, numerical indices, the n-fold composition of functions, recursive approximations to infinite sums, or much else. For good reason, Pierre-Simon de Laplace wrote in 1795 that

> It is India that gave us the ingenuous method of expressing all numbers by the means of ten symbols, each symbol receiving a value of position, as well as an absolute value; a profound and important idea which appears so simple to us now that we ignore its true merit, but its very simplicity, the great ease which it has lent to all computations, puts our arithmetic in the first rank of useful inventions, and we shall appreciate the grandeur of this achievement when we remember that it escaped the genius of Archimedes and Apollonius, two of the greatest minds produced by antiquity.

Here then is the route through Turing's tunnel. The mastery of positional algorithms, along with other innovations, made possible the computational mathematics which logicians took as their objects of interest. While for Indian mathematics, the order of discovery from grammar to computation was just reversed from modern times. The Sanskrit grammarians first discovered

methods for representing exact recursive procedures by modeling empirical languages, building on the techniques of ritual theorists. Indian mathematicians did not need the formalisms of Indian grammar, while Indian grammarians, as today, did not need algorithmically strong procedures to model natural language. But they did need formalisms. In contrast to modern logicians, they devised exact methods for procedural analysis *without* experience with multiplicative arithmetic and its algorithms. Starting with their data of spoken Sanskrit, and motivated by techniques of ritual description, the ancient grammarians directly created methods to describe generative patterns in phonology, morphology and syntax.

The formal-empirical grammars were then legitimately seen as standards of procedural exactness, with influences ranging from the *Laws of Manu* to the *Kāmasūtra*. Following the oldest grammatical theories came the discovery of positional notation and its computing power, even as that power was not fully understood. There was also no explicit conception, as there is today, of the connections between grammatical formalisms and mathematical algorithms, particularly proof procedures. To make this connection took centuries, involving new roles for axiomatics, foundational ''crises,'' and much else. But one prerequisite was mastery of a sufficient variety of symbolic and computational methods to motivate formalist methods. A turning point came with Viète's algebra and symbolic philosophy, reinforced by Stevin and others like John Wallis, for whom positional notation made algebra the ''universal art.'' Among the innumerable ways to count, positional notation, because of its efficiency and flexibility, became a primary means for representing recursive procedures. The adoption of positional notation by Arab mathematicians and its importation to Europe eventually made universal computation available in modern mathematics and then modern logic. The arithmetic facilitated by positional value was correctly seen as essential to modern mathematics, and is implicit in many of the earliest formal systems, even those of intuitionism. The generative behavior of human computers and symbolic logicians became the basis for Turing's theory of computation.

Thus Pāṇini is to spoken Sanskrit as Turing and other early logicians are to modern mathematics and early formalisms. Turing did not use a natural language like Sanskrit for his generative analysis. But he had many examples of formal systems, most of which assumed positional notation or its equivalent. These systems were the object of Turing's analysis, and their algorithmic power implied universal computation through elementary arithmetic. Once Turing's analysis clarified the status and generality of effective procedures, it was possible to focus on *weaker* systems not intended for arithmetic at all. Seeing much of this plus Turing's cognitive implications, Chomsky reversed the tunnel relative to distant Sanskrit ancestors. But the tunnel was completed by Turing and other pioneering logicians who in this way are behavioral linguists. Their techniques rely on models of symbol usage or natural language, notably Turing's computing scribe, but also Hilbert's finitary symbols and Post's rewrite rules, to model the creations of modern mathematics. For this they are all grammarians of reason, arithmetic and mathematics.

Acknowledgements I am grateful to the International Institute for Asian Studies for sponsoring my participation in the 2006 workshop "The Generosity of Artificial Languages in an Asian Perspective," and especially to Frits Staal for his continued interest and advice. Jens Høyrup's extensive critique of an early draft provided invaluable direction and stimulation. Thanks for comments or discussion go to Judith Aissen, Martin Davis, Guy Deutscher, Ivor Grattan-Guinness, Mike Graves, Don Knuth, Peter Pesic; and to Alex Jaker for Sanskrit help.

References

Aitchison, Jean (2001). *Language change: Progress or decay*? (3rd ed.). New York: Cambridge University Press.

Bag, A. K. (1975). al-Biruni on Indian Arithmetic, *Indian Journal for the History of Science, 10*, 174–184.

Barber, Charles (2000). *The English language* revised edition. New York: Cambridge University Press.

Baron-Cohen, Simon (1997). *Mindblindness: An essay on autism and theory of mind*. New York: Cambridge University Press.

Boolos, George (1979). *The unprovability of consistency*. New York: Cambridge University Press.

Brinton, Laurel, & Traugott, Elizabeth (2005). *Lexicalization and language change*. New York: Cambridge University Press.

Brough, John (1951). Theories of general linguistics in the Sanskrit grammarians. In Staal 1972: 402–413.

Chomsky, Noam (1963). Formal properties of grammars. In Luce et al. 323–418.

Chomsky, Noam (1980). *Rules and representations*. New York: Columbia University Press.

Chomsky, Noam, & Miller, George (1963). Introduction to the formal analysis of natural languages. In Luce et al. 269–322.

Clark, Herbert (1996). *Using language*. New York: Cambridge University Press.

Datta, Bibhutibhushan, & Singh, Avadesh Narayan (1935). *History of Hindu mathematics* I, II. Delhi: Bharatiya Kala Prakashan. (Reprinted (2001)).

Davis, Martin Davis (Ed.) (1965). *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions.* Hewlett, New York: Raven Press.

Davis, Martin Davis, Sigal, Ron, & Weyuker, Elaine (1994). *Computability, complexity, and langauges*: *Fundamentals of theoretical computer science* (2nd ed.), New York: Academic Press.

Deutscher, Guy (2005). *The unfolding of language*. New York: Metropolitan Books.

Emch, Gerard, Sridharan, R., & Srinivas, M. D. (Eds.) (2005). *Contributions to the history of Indian mathematics*. New Delhi: Hindustan Book Agency.

Enderton, Herbert (1972). *A mathematical introduction to logic*. New York: Academic Press.

Fischer, Michael, & Rabin, Michael (1974). Super-exponential complexity of Presburger arithmetic. SIAM-AMS Proceedings, No. 7, American Mathematics Society, Providence, R.I., 27–41.

Gandy, Robin (1995). The confluence of ideas in 1936. In Rolf Herken (Ed.), *The universal turing machine: A half-century survey* (2nd ed.). New York: Springer-Verlag.

Gardner, Howard (1987). *The mind's new science: A history of the cognitive revolution*. New York: Basic Books.

Gentner, Dedre, & Goldin-Meadow, Susan (Eds.) (2003). *Language in mind: Advances in the study of langauge and thought*. Cambridge: MIT Press.

Goody, Jack (1987). *The interface between the written and the oral*. New York: Cambridge University Press.

Ifrah, Georges (2000). *The universal history of numbers: From prehistory to the invention of the computer*. (David Bellos et al. trans.). New York: John Wiley & Sons.

Karmiloff-Smith, Annette (1992). *Beyond modularity: A developmental approach to cognitive science*. Cambridge, MIT Press.

Keller, Agathe (2006). *Expounding the mathematical seed: A translation of Bhāskara I on the mathematical chapter of the Āryabhatīya* I, II. Boston: Birkhäuser.

Klein, Jacob (1936). *Greek mathematical thought and the origin of algebra*. (Eva Brann trans. 1968). New York: Dover (Reprint (1992)).

Luce, R. Duncan, Bush, R., & Galanter, E. (Eds.) (1963). *Handbook of mathematical psychology* (Vol. II). New York: John Wiley & Sons.
Minsky, Marvin (1967). *Computation: Finite and infinite machines*. New York: Prentice-Hall.
Nayar, B. K. (1975) al-Biruni and science communication in Sanskrit. *Indian Journal for the History of Science, 10*, 249–252.
Pullum, Geoffrey, & Scholz, Barbara (2005). Contrasting applications of logic in natural language syntactic description. In Petr Hájek, et al. (Eds.), *Logic, methodology and philosophy of science: Proceedings of the twelfth international congress* (pp. 481–503). London: King's College Publications.
Renou, Louis (1941). Les connexions entre le rituel et la grammaire en Sanskrit. In Staal 1972 : 435–469.
de Saussure, Ferdinand (1915/1959). *Course in general linguistics*. (Wade Baskin trans. C. Bally et al. (Eds.)). New York: McGraw-Hill.
Sharma, Rama Nath (1987). *The* Aṣṭādhyāyī *of Pāṇini* I: *Introduction to the* Aṣṭādhyāyī *as a grammatical device*. New Delhi: Munshiram Manoharlal.
Staal, Frits (Ed.) (1972). *Reader on the Sanskrit grammarians*. Cambridge: MIT Press.
Staal, Frits (1983). *Agni: The Vedic fire ritual* I, II. Delhi: Motilal Banarsidass (Reprint (2001)).
Staal, Frits (1988). *Universals: Studies in Indian logic and linguistics*. Chicago: University of Chicago Press.
Staal, Frits (1990). *Ritual and mantras: Rules without meaning*. Delhi: Motilal Banarsidass.
Staal, Frits (2006). Artificial languages across sciences and civilizations. *Journal of Indian Philosophy, 34*, 87–139.
Tomasello, Michael (1999). *The social origins of human cognition*. Cambridge: Harvard University Press.
Tomasello, Michael (2003). *Constructing a language: A usage-based theory of language acquistion*. Cambridge: Harvard University Press.
Turing, Alan (1936). On computable numbers with an application to the Entscheidungsproblem. In Davis 115–153.
Woepcke, Franz (1863). Mémoire sur la Propagation des Chiffres Indiens. *Journal Asiatique*, Sixth Series, *1*, 442–529. In Fuat Sezgin (Ed.), *Franz Woepcke*, *Études sur les Mathématiques Arabo-Islamiques* (Vol. 2). Frankfurt: Institute for the History of Arabic-Islamic Science and the Goethe University, 1986.
Young, Paul (1985). Gödel theorems, exponential difficulty and undecidability of arithmetic theories: An exposition. In Anil Nerode, & Richard Shore (Eds.), *Recursion theory*: *Proceedings of symposia in pure mathematics* (Vol. 42). Providence: American Mathematical Society.