The Pāṇini Machine

Gérard Huet

Emeritus, Inria Paris Center

Distinguished Lecture, University of Hyderabad

November 7th 2019

Paninian myths and fake news

- Pāṇini's grammar is perfect
- It is written in Sanskrit
- Natural language understanding is a big AI challenge
- Thus Sanskrit is the ultimate programming language for AI applications
- NASA is working secretely on this paradigm (conspiration theory)
- This was explained in Forbes Magazine in 1987 (fake news)

Origin of the myth

- Rick Briggs' article, AI Magazine 1985
- Suggests the use of Nyāya for knowledge representation
- Interesting reading, śāstrically correct, innovative, relevant
- But not substantiated by concrete development so far
- Rick Briggs vanished
- Thus it is hard to stop the rumors

Pāṇini deserves better credit

- Sanskrit is not really a natural language
- Neither vernacular nor mother-tongue
- It is a high-register learned language for rational argumentation, refined from Prakrits current in North India at the time
- Classical Sanskrit is co-extensive with Pāṇini's grammar
- Descriptive versus prescriptive
- Sanskrit evolved only within the *de facto* prescriptive grammar
- and thus the grammar precision paradoxically improved over time!
- Pāṇini's grammar is not written in Sanskrit
- It is not a computer program per se
- It is a formal document prescribing mechanical operations
- Similar to the operations manual of an abstract machine

What kind of computer are you talking about ?

Computers are number-crunching machines. Grammars concern speech production, syntax, meaning, etc. How could Aṣṭādhyāyī be compared to a computer ?
Computers are universal computing machines, they can operate on any symbolic material, that may represent any natural phenomenon. Machines are more specialized automata.
Jacquard machines automate textile design production.
Automatic pianos automate music production. Here Pāṇini machines automate Sanskrit speech production.

Actual concrete Pāṇini machines

People may object to my terminology, and ask: "What machine are you talking about? Show me a concrete Pāṇini machine." My answer to this objection is easy. Any competent paṇḍit is a biological Pāṇini machine. By learning by rote the grammar, paṇḍits are able to physically realize the Paninian operations, at least in the following sense: they teach their students how to operate the grammar, and demand of their students to be able to give the precise sequence of *sūtras* justifying their linguistic productions. Furthermore, if the student produces a *prakriyā* sequence that either invokes illegally a sūtra or results in a non-intended meaning of the enunciation, they are able to point out the precise point where the student has erred. In this sense, they are living Pāṇini machines!

Note that this does NOT suggest that paṇḍits have developed a neuronal structure in their brain that emulates a Pāṇini machine, but just that they have internalized the grammar enough to explain proper linguistic production accordingly.

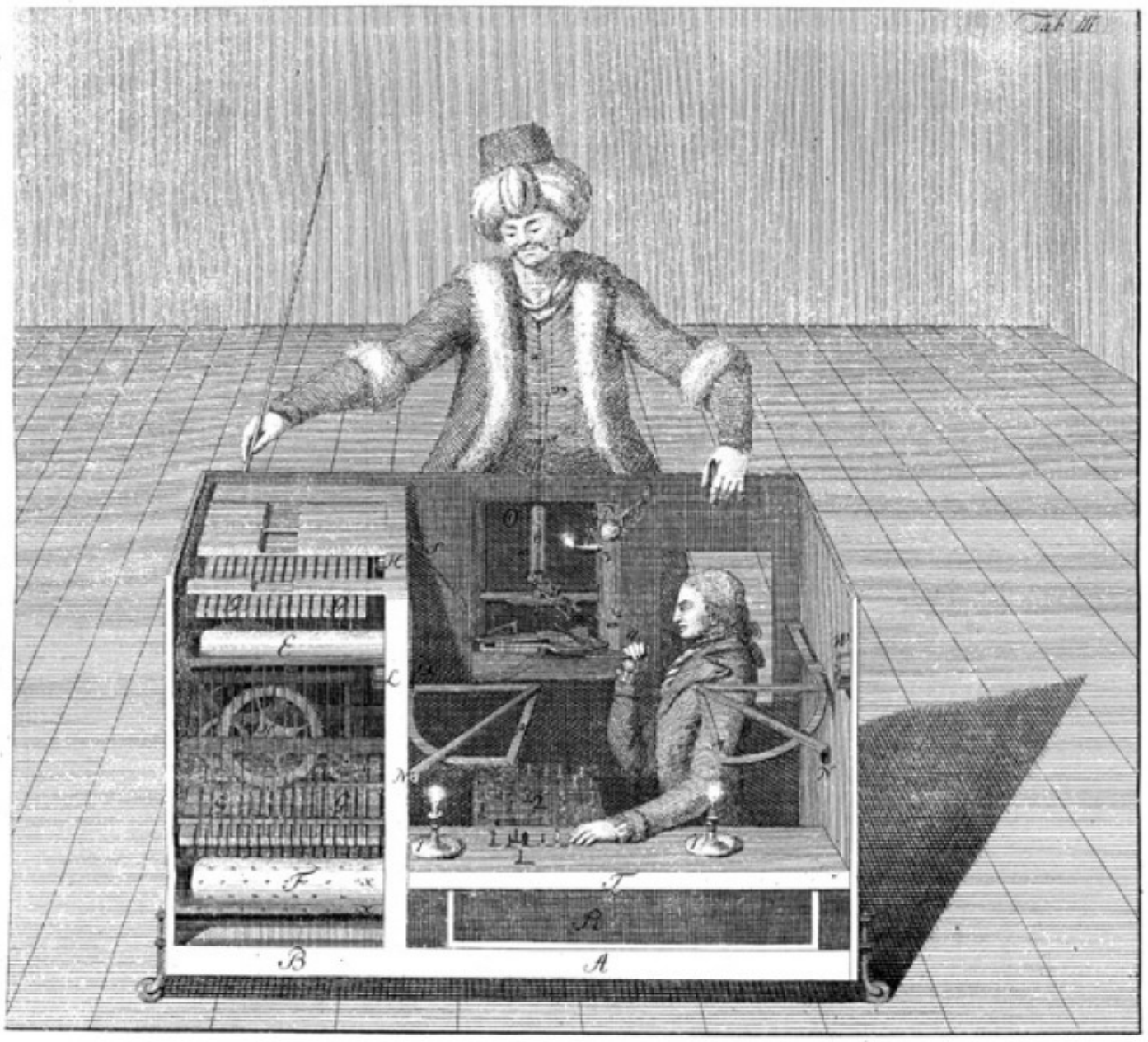
More objections

A more subtle objection is that pandits may be using all kinds of additional knowledge about the language, and somehow use hidden meta rules (paribhāṣā) not explicitly stated in Aṣṭādhyāyī. This suggests that a software implementation executing actual electronic hardware which emulates the machine should definitely settle the matter.

We shall come back to electronic Panini machines later. For the moment we shall use the automatic piano analogy, and imagine the pandit as an organ player. The organ has many complex keyboards, one for the sūtras, one for the dhātupāṭha, one for the uṇādisūtras, etc. The organ is not speaking Sanskrit in real time, though, only at the end of a sentence do we have speech production. Then we'll replace the pandit with a tape input.

We have to keep in mind the 18th century mechanical Turk automaton that claimed to play chess automatically: we have to make sure that no pandit is hidden inside the machine.

Debunking the Turk Automaton fraud



Overview of Pāṇini's grammar

- Is it a generative grammar ?
- Is it a dependency grammar ?
- In a nutshell, it is both: generative morphology, dependency sentence structure
- Morphology generation uses string rewriting in the manner of Post systems
- Sentential consistency deals with dependency constraint analysis: *kāraka* = semantic role, *ākāṅkṣā* = dependency
- Semantics is Situation theory
- Language is symbolic: it is more theater than describing reality
- Loop *pada* formation, then *kāraka* assignment, the final phonetic smoothing in *tripādī section*

Generative in what sense ?

- Grammarians do not agree on what is generated from what
- arthapakṣī: generate the form from its meaning
- śabdapakṣī: generate its meaning from the form
- Both points of view are wrong, but each is half-true, there is mutual recursion between form and meaning in the grammar
- What is generated is a pair ⟨śabda,artha⟩
- Thus the relevant notion is a Sanskrit **sign** in the sense of de Saussure

But then, what is the input ?

- Language is a coin with two sides: speaking and understanding
- Pāṇini's grammar gives rules on how to speak meaningfully
- It defines *śabdasṛṣṭiḥ* rather than explaining *śabdabodhaḥ*
- Thus the input is the locutor's communicative intention: vivakṣā
- The locutor intends to communicate meaning, but its form also matters
- denotation (*abhidhā*) versus connotation (*vyañjanā*)
- choice of a synonym (e.g. to fit meter) also benefits from connotations of its homonyms for *dhvani*

Constructive sign elaboration

Using the grammar has the general form of elaborating a sign recursively from the signs of its components. The primitive signs are the verbal roots given with their (atomic) meaning, both components being extracted from root tables (*dhātupāṭhaḥ*). Morphology elaborates words from their component bases (*prakṛtiḥ*) and suffixes (*pratyayaḥ*). Thus, on the nominal side, we get *kṛdanta* stems (*prātipadika*) from *dhātu* roots and *kṛt* suffixes, *taddhitānta* stems from previous stems and *taddhita* suffixes, inflected nominal forms (*subantas*) from stems and *sup* suffixes. Similarly, we get conjugated verbal forms (*tiṅantas*) from roots, *lakāras* and *tiṅ* suffixes. In all cases, these *pratyaya* suffixes bear all the information to compute mechanically both the resulting compound form (*śabda*) and its meaning (*artha*) expressed as a canonical prose paraphrase (*vigraha*).

Mechanical computation

All these elaboration procedures are really exact computing processes. This justify considering Aṣṭādhyāyī not just as a grammar, but actually as the operating manual of an abstract computer, which we name the Pāṇini Machine. We are going to look at one example of computation of this machine, and on the way understand the basic data and control structures designed by Pāṇini.
As an initial step we must better understand both the data component and the meta-descriptive formalism of the machine.

Basics of Paninian calculus

The object language of the machine is Sanskrit represented at the phonemic level, which is the discretization of articulated speech. This level is of a finer granularity than the syllabic level, explicit e.g. in writing in syllabic alphabets such as devanāgarī. Thus the relevant notion is varṇa (roughly corresponding to the modern notion of phoneme). The varṇamālā is the Sanskrit alphabet, as a list of 50 varṇas, carefully arranged in an algebraic manner, first with vowels, then with consonants, each represented as 5 layers corresponding to articulation points, each layer giving the cross product of two phonetic boolean features (surd/voiced, unaspirated/aspirated), plus one nasal.

Putting to shame the English alphabet

a	ā	i	ī	u	ū	ṛ	ṝ	ḷ	e	ai	o	au
						ṃ	ḥ					
				k	kh	g	gh	ṅ				
				c	ch	j	jh	ñ				
				ṭ	ṭh	ḍ	ḍh	ṇ				
				t	th	d	dh	n				
				p	ph	b	bh	m				
			y	r	l	v	ś	ṣ	s	h		

Figure: Sanskrit varṇamālā

Metalanguage

This well-thought-out algebraic structure of the discretized speech domain probably predates Pāṇini, since prior grammarians had studied phonetics (*zikṣā*) in the framework of the *prātiśākhya* treatises. But a brilliant innovation of the grammar is to duplicate the standard alphabet to serve as a set of meta-linguistic markers, usable to denote the micro-code operations of the machine and various meta-linguistic parameters such as the *it* markers in root description. These are called anubandhas.

Example: Śivasūtras

The grammar starts by giving another view of the varṇamala:

a i u ṇ |
ṛ ḷ k |
e o ṅ |
ai au c |
ha ya va ra ṭ |
la ṇ |
ña ma ṅa ṇa na m |
jha bha ñ |
gha ḍha dha ṣ |
ja ba ga ḍa da ś |
kha pha cha ṭha tha ca ṭa ta v |
ka pa y |
śa ṣa sa r |
ha l ||

anubandhas are marked in red.

Condensed definitions

The Śivasūtras are used to define abbreviations for the families of phonemes sharing common treatment in the grammar, called *pratyāhāras*. Each *pratyāhāra* is of the form 'XaY', where X is a *varṇa* and Y is an *anubandha*, and it denotes the set of phonemes between X and Y (markers excluded). For instance, nasals are denoted by ña**m**, vowels are denoted by a**c**, consonants are denoted by ha**l**.

This is a very compact representation of all subsets of *varṇas* that are needed as characteristic properties for the machine operations.

Please note that there is only one redundancy in the above sūtras: phoneme *h* appears twice. Actually, Wiebke Petersen proved that this redundancy is unavoidable, and that this listing is thus optimally compact.

A worked-out example

Let us show quickly how to derive the stem *kāraka* in the sense of actor, i.e. 'agent of acting'. This stem is a primary derivative (*kṛdanta*) obtained by root *kṛ* (to act), with morpheme *aka* affixed to morpheme *kār*, obtained by raising root *kṛ* to its second grade by the *vṛddhi* operation. Here is the (simplified) Paninian derivation.

First, we retrieve the sign for root *kṛ*, by looking up the roots table (*dhātupāṭhaḥ*). At entry *kṛ*, we get: ḍukṛñkaraṇe. We first peel off the morphological parameters ḍu and ñ of the root, record them, and extract the sign components: *kṛ* (the *śabda* phonetic component) and acting its *artha* meaning component (since the locative *karaṇe* means "in the sense of acting"). Thus we start with sign ⟨*kṛ*,acting⟩.

A worked-out example (continued)

Next, since we intend to express the notion of agent, we go to the section of the grammar concerning agent nouns, starting with sūtra (3.1.133): *ṇvultṛcau*, i.e. "both (kṛtpratyayas) *ṇvul* and *tṛc* (are applicable to any root)". By selecting the first component *ṇvul* we are now licensed to affix *pratyaya ṇvul* to the current *prakṛti* '*kṛ*', yielding string *kṛṇvul*. Now the string rewriting proceeds. The first operation is denoted by anubandha *ṇ*, which is microcode for the *vṛddhi* operation, rewriting *'kṛ'* into *'kār'*. Next the anubandha string *vu* invokes an abbreviation mechanism, which expands into śabda *'aka'*, which is thus appended to '*kār*' to yield string *'kāraka'*. The last anubandha *l* indicates that the accent precedes the suffix, yielding accented śabda *'kā́raka'*. And since the sūtra is in the section of agent nouns, the new computed sign is

$$\langle \textit{kā́raka}, \text{agent of acting} \rangle.$$

We may then use sup-pratyayas etc to get an inflected pada

A worked-out example (end)

If you think this trivial example is complex, please consider that it has been considerably simplified, since a lot of book-keeping administration has been omitted, such as checking that the invocation of the sūtra is not barred by possible application of rules having higher priority.
In fact, it is not complex, it is just very-low level programming of the machine at its micro-code level. This is similar to programming in machine-language in the early days of computer science. In contrast, the conflict-resolution rules that manage the relative priority and mutual blocking/feeding of the rules (*anuvṛtti*) are rather hairy, to say the least. We shall not discuss these rules, nor the meta-rules that direct the general control flow of the machine, and restrict our attention to the actual data-processing prescriptive rules (*vidhi sūtras*).

Some terminology

Let us call **script** the sequence of vidhi rules necessary to derive a Sanskrit sign. We say the script is **Paninian** if it is correct with respect to the conflict resolution rules. The analogy with computer programming is that the scripts are the programs of the machine, and that checking that they are Paninian is analogous to compile-time type-checking and other sanity checks of the compiler.

In search of a Paninian programming language

Our scripts are actually close to the *prakriyā* explanations of the grammatical tradition - condensed invocations of the sūtras such as *ḍukṛñ-ṇvul-su*. Organizing them systematically in hierarchical manner would lead to some kind of abstract syntax of a somewhat esoteric programming language. It would be an interesting research program to exactly define this formal language, and use it as the conceptual basis for a software implementation of Pāṇini's machine, with the following ingredients.

Paninian programming framework

- Design of core microcode interpreter 'Pāṇinīyam'
- Design of a formal 'Prakriyā' machine language
- Compilation of Prakriyā scripts into Pāṇinīyam
- Decision procedure for recognizing a script as Paninian
- Design of a high-level language 'Vivakṣā' for expressing clear statements of the locutor's communication intention
- Writing of a compiler for Vivakṣā, generating Prakriyā scripts that are Paninian by construction

This tentative **modular** research program sketches a complete chain of production, from Vivakṣā statements to executable Paninian Prakriyā scripts

Paninian variations

The sketch above concerns pada construction. The sentence-level realization could be realized in more of a constraint programming methodology.
Also, this bottom-up production of the script could be reversed, partially or totally, by top-down synthesis. Seen from a proof theory angle, this would be analogous to synthesis of proofs by backward reasoning in sequent calculus rather than forward reasoning from axioms in natural deduction. This would be more natural from the angle of expressing meaning. Operating the machinery would be analogous to logic programming à la Prolog rather than functional programming.
The design could also be modularized by using dictionaries, where stems could be stored with their pre-computed signs.
Let us now turn at previous attempts at making Pāṇini machines, starting with hardware.

Wrong Pāṇini machines

Harkare's hardware Pāṇini machine

Harkare's Pāṇini machine demonstrated

The machine was presented at the International Sanskrit Computational Linguistics Symposium in Hyderabad (2009):

Sri Gunde Rao Harkare

Quoting from "Languages and Literary Cultures in Hyderabad", ed. Kousar.J. Azam, Routlege 2018 (chapter 10): Sri Gunde Rao Harkare (1887-1979) was an eminent scholar, a multilinguist, eminent critic and a multifaceted genius. He was fluent in Persian, Arabic, Telugu, Marathi and English. He acted as a translator in the Special Criminal Court of Hyderabad and toured extensively in the dominion of the Nizam, then became p.a. to the Chief Judge at the High Court, then District Judge, Deputy Collector, and Sessions Judge at Gadwal. He developed a passion for learning Sanskrit, obtained the title of Vacaspati at the Academy of Navadvip, and Vidya Bhushan at Ayodhya and Belgaum, Vidya Bhaskara, Pandit by Tirupati Devasthanam, Certificate of Honour by Rashtrapati. In his later life he worked on a teaching aid scheme called "Sanskrit Grammar Made easy" 'in a technological form' and composed a guide 'How to Handle the Machine'.

Software attempts at implementing Aṣṭādhyāyī

- Shivamurthy Swamiji Gaṇakāṣṭādhyāyī
- Peter Scharf & Malcolm Hyman LIES
- Pawan Goyal, Lakshmidhar Behera & Amba Kulkarni Computer Simulation of Aṣṭādhyāyī 2008
- Anand Mishra PhD thesis Heidelberg 2009
- Wiebke Petersen & Oliver Hellwig 2009
- Sridar Subbanna & Srinivasa Varkhedi Conflict resolution techniques 2009
- Amba Kulkarni, Pawankumar & Rāmakṛṣṇamācāryulu WSC 2015
- G. Huet. Sanskrit signs and Pāṇinian scripts WSC 2015
- Dhaval Patel & Shivakumari Katuri Prakriyāpradarśinī Subanta generator WSC 2015
- Sarada Susarla, T. M. Rao & Sai Susarla PAIAS WSC 2018
- Samir Sohoni & Malhar Kulkarni Computational Aṣṭādhyāyī WSC 2018

Software attempts (following)

The above contributions have discussed various problems in software representation of Paninian concepts, such as conflict resolution techniques. Some have succeeded in emulating *vidhi* portions of the grammar. But a comprehensive solution still seems far away.

Actually, some scholars have expressed doubts about the whole endeavour. Peter Scharf explains difficult points in conflict resolution, which will need specific additional research. In his recent monograph "Modeling the Pāṇinian system of Sanskrit Grammar" (Heidelberg University Press, 2019), Anand Mishra expresses doubts at direct simulation of Aṣṭādhyāyī devices. Thus a full emulator of Pāṇini's machine is still an open problem, but we see no theoretical impossibility at implementing a software implementation reasonably consistent with Aṣṭādhyāyī.

Fundamental contributions of Pāṇini to informatics

No matter how long it will take to write a software simulation of Aṣṭādhyāyī, it remains that Pāṇini made important contributions to informatics and information theory, 25 centuries before the first investigations of recursive function theory in mathematical logic, and the advent of electronic computers, as already remarked by Saroja Bhate & Subash Kak in "Pāṇini grammar and Computer Science", Annals of the Bhandarkar Oriental Research Institute, vol. 72 (1993).
Thus it is not an overstatement that Pāṇini should be considered as an Informatics pioneer.

Pāṇini as Ādiganakaḥ

- meta-linguistic markers
- record notation encoding
- formal string rewriting
- hierarchical scope
- selection by pattern-matching
- conflict resolution for non-determinism
- object-oriented descriptions (taddhita suffixes)
- keen information theory awareness, compaction by sharing

Thank you for your attention