Suganita

Part I: Token Specification

A. Chawla
REAL Institute

December 3, 2025

Abstract

This document presents the first part of a multi-stage blueprint for सुगणिता (``well-computed''), a programming language written entirely in Devanāgarī script and designed for microcontroller platforms, specifically the Arduino Uno (ATmega328P). Unlike superficial localization efforts that simply translate Western programming constructs, सुगणिता draws conceptually from Indian intellectual traditions: Vedic mathematics, Nyāya logic (classical Indian reasoning), and Pāninian grammatical theory.

Part I focuses on the lexical layer---the fundamental vocabulary of the language. It establishes design goals, outlines a phased development plan, and provides a complete token specification for versions v0--v2. The token tables cover both high-level language keywords (function definitions, control flow) and low-level virtual machine operations (stack manipulation, arithmetic, hardware I/O). Each token is documented with its Devanāgarī form, ASCII transliteration, and English gloss linking it to its philosophical background.

Key design choices include representing ``no operation'' as शून्यः (śūnya, constructive pause rather than meaningless gap) and modeling conditional jumps after the structure of classical Indian argumentation. The hardware constraints of the Arduino Uno (2 KB RAM, 32 KB flash, 8-bit architecture) enforce discipline on the token design, ensuring the language remains implementable on resource-constrained platforms while preserving its conceptual foundations.

This work is dedicated to Kriya Yoga master Paramahamsa Hariharananda Giri (1907-2002).

Contents

1 Design goals for सुगणिता **3**

1.1 Motivation and identity 3

1.2 Hardware and engineering constraints 3

1.3 Philosophical and mathematical background 4

1.4 Scope of Part I 4

2 Development plan overview **6**

2.1 Phase 0: Concept and constraints 6

2.2 Phase 1: Arduino–hosted VM prototype 6

2.3 Phase 2: External compiler and language syntax 7

2.4 Phase 3: Assembly implementation of the VM 7

2.5 Phase 4: Lightweight ML augmentation 7

2.6 Phase 5: Self–hosting and expansion 8

3 Token specification for versions v0–v2 **9**

3.1 Core language keywords 9

3.2 Control flow and logical structure 10

3.3 Operators and punctuation 11

3.4 Virtual machine opcodes (v0–v2) 12

3.5 Built–in hardware functions (source–level) 14

3.6 ML–related tokens (planned for v2) 15

4 Summary and outlook **17**

1 Design goals for सुगणिता

1.1 Motivation and identity

The language सुगणिता is intended to differ from a purely transliterated or ``localised'' Western language in three crucial ways:

1. Script and surface form. All identifiers, keywords, and structural tokens of the language are expressed in Devanāgarī. Source files are stored as UTF--8 and are visually legible to anyone familiar with Sanskrit or modern Indian languages using this script.
2. Conceptual grounding. The interpretation of tokens, especially at the control, arithmetic and logical levels, is guided by Indian mathematical and philosophical categories. For example, the instruction traditionally called ``NOP'' (no operation) is replaced by शु (śūnya), representing a constructive pause rather than a meaningless gap.
3. Hardware orientation. The first several years of the project explicitly target the Arduino Uno. This constraint enforces clarity, compactness, and discipline on the language design: the token set must be implementable in a few kilobytes of RAM and a few tens of kilobytes of flash, on an 8–bit architecture.

1.2 Hardware and engineering constraints

The initial target architecture is the ATmega328P microcontroller, as configured on the Arduino Uno board:

- 8–bit AVR core at 16 MHz
- 2 KB SRAM
- 32 KB flash program memory (approximately 30 KB usable after bootloader)
- 1 KB EEPROM
- No hardware floating point unit
- Simple peripherals: timers, ADC, PWM, digital IO, UART

These constraints inform the token design:

- The virtual machine will be stack–based, using 16–bit logical words implemented over 8–bit registers.
- The bytecode instruction space is limited; opcodes must be compact and semantically dense.
- Strings and Devanāgarī text will be emitted as UTF–8 sequences but interpreted as opaque bytes by the VM.
- Arithmetic in v0–v2 will focus on integers; fixed–point or scaled integer arithmetic may later represent real values.

1.3 Philosophical and mathematical background

At a conceptual level, सुगणिता aligns core language affordances with six strands of the Indian knowledge tradition: Vedic mathematics informs arithmetic semantics; Nyāya logic shapes control flow; Pāninian grammar guides lexical discipline; Sānkhya and Yoga concepts govern execution state; Vākya tradition ensures meaningful surface forms; and embedded ML praxis enables lightweight inference. The goal is not mere ornamentation, but principled mapping from philosophical categories to concrete implementation.

Concept diagram (overview). The diagram below sketches how traditions inform layers of the language. It is illustrative rather than prescriptive.

Pragmatic constraints. All mappings must respect the ATmega328P envelope: 2 KB SRAM, ∼30 KB usable flash, 8-bit arithmetic, and no FPU. Accordingly, v0--v2 prioritize: integer-first arithmetic, compact branching encodings (16-bit jump targets), UTF-8 treated opaquely in the VM, and ML as minimal fixed-point primitives callable without complicating the core interpreter loop

1.4 Scope of Part I

This Part I addresses the lexical layer only:

- Keyword and operator names

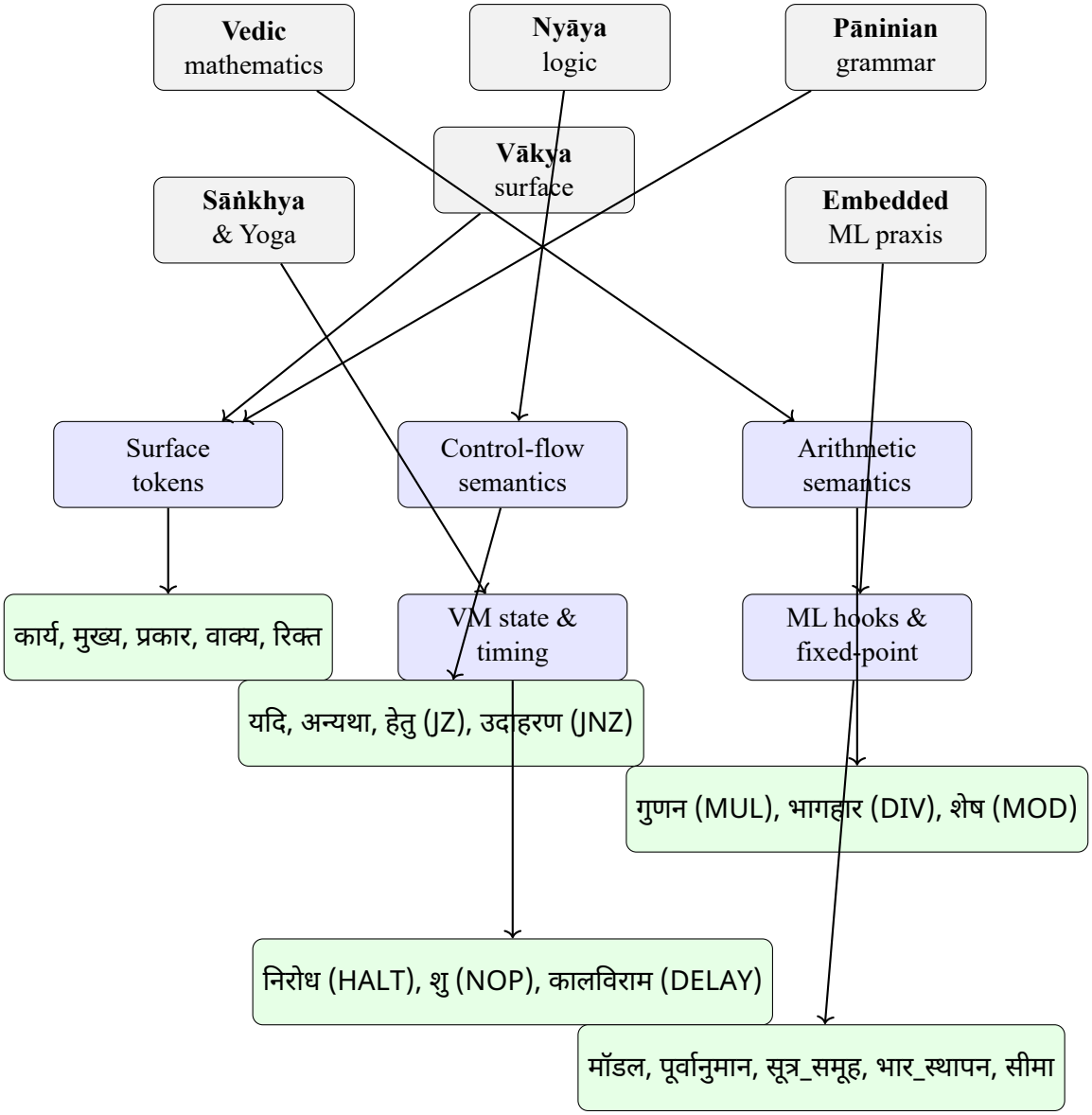

Figure 1: Conceptual mapping from Indian traditions to language layers and token examples in सुगणिता

- Punctuation tokens and structural delimiters
- Virtual machine opcode names and their semantic labels
- Built–in function names for hardware interaction

Grammar, semantics, and virtual machine implementation details will appear in later parts.

2 Development plan overview

The overall development path is structured into phases. Only the tokens relevant to v0--v2 are specified in this document; however, the plan contextualizes how these will be used.

2.1 Phase 0: Concept and constraints

- Fix the hardware target to Arduino Uno / ATmega328P.
- Decide on UTF--8 as the sole source encoding for `.su` files.
- Choose a stack–based virtual machine model and a 16–bit logical word size.
- Reserve ranges of bytecode opcodes for arithmetic, control, IO, and future ML extensions.
- Decide that early implementation will be in Arduino C++ (for ease), followed by a rewrite in AVR assembly.

2.2 Phase 1: Arduino–hosted VM prototype

- Implement a minimal interpreter loop in C++ on the Arduino Uno.
- Support:
  - stack operations,
  - integer arithmetic,
  - unconditional and conditional jumps,
  - serial output,
  - a HALT instruction.
- Hard--code small bytecode pages in flash to test the VM.
- Use simple Devanāgarī strings for serial output to verify the complete UTF--8 path.

2.3 Phase 2: External compiler and language syntax

- Implement a compiler on a host computer (language still to be chosen) which:
  - reads सुगणिता source code,
  - tokenizes using the token specification in this document,
  - parses according to the basic grammar (to be specified later),
  - emits bytecode (`.subc`) using the opcode set defined here.
- Introduce language--level keywords (function definitions, conditionals, loops) mapped to the low–level VM tokens.

2.4 Phase 3: Assembly implementation of the VM

- Rewrite the VM core in AVR assembly, using the same opcode semantics.
- Allocate fixed memory regions for:
  - VM state (instruction pointer, stack pointer, frame pointer),
  - data stack,
  - global variables, and
  - hardware state caches if needed.
- Implement the hardware IO opcodes: digital IO, analog input, delay, and serial.

2.5 Phase 4: Lightweight ML augmentation

- Introduce ML tokens for simple models suitable for an 8–bit microcontroller:
  - linear models or very small multi–layer perceptrons with a handful of parameters,
  - hand–coded fixed–point inference.
- Ensure the ML tokens are integrated without altering the core VM structure.

2.6 Phase 5: Self–hosting and expansion

As the toolchain stabilizes:

- Gradually rewrite the host compiler in सुगणिता itself.
- Port the VM and language to more powerful microcontrollers.
- Expand the token set conservatively when new capabilities are needed.

3 Token specification for versions v0–v2

The token system for सुगणिता can be grouped into several categories:

1. Core language keywords
2. Types and literals
3. Operators and punctuation
4. Control–flow and structure keywords
5. Virtual machine opcodes
6. Built–in hardware functions
7. ML–related high–level constructs

In the tables below, each row lists:

- the Western or conventional concept,
- the intended Devanāgarī token,
- an ASCII transliteration,
- an English gloss.

Only tokens planned for implementation in v0, v1 or v2 are included.

3.1 Core language keywords

These are high–level language keywords, visible to the programmer in सुगणिता source files.

Western concept	Devanāgarī token	Transliteration	English gloss / role
Function definition	कार्य	kārya	Introduces a function definition

Entry point (main)	मुख्य	mukhya	Designated main entry function
Return from function	वापस	vāpas	Return a value and exit function
Type keyword	प्रकार	prakāra	Used in type annotations (v2)
Integer type	पूर्णांक	pūrnānka	16--bit signed integer
Fixed--point / real type	दशमलव	daśamala	Real values represented as scaled integers (v2)
Boolean type	सत्यता	satyatā	Boolean truth value
String type	वाक्य	vākya	UTF--8 string
Void / unit type	रिक्त	rikta	No return value
Model definition	मॉडल	mōdal	Declares a statistical or ML model (v2)
Training block	प्रशिक्षण	prasiksana	Marks a training configuration block (for host side)
Prediction / inference	पूर्वानुमान	pūrvānumāna	Request a model prediction
Dataset reference	डाटा	dātā	Points to a dataset configuration (host side)
Infinite loop, forever	अनवरत	anavarata	Used as a convenience keyword for endless loops (v2)

3.2 Control flow and logical structure

Western concept	Devanāgarī token	Transliteration	English gloss / role
If	यदि	yadi	Conditional branching (if)

Else	अन्यथा	anyathā	Alternative branch (else)
While loop	जबतक	jabtaka	While a condition holds
For loop	क्रम	krama	Iterative sequence (for)
Break	विच्छेद	vicched	Terminate current loop early
Continue	पुनः	punah	Skip to next iteration
Logical true	सत्य	satya	Boolean true literal
Logical false	असत्य	asatya	Boolean false literal

3.3 Operators and punctuation

Because सुगणिता aims for an entirely Devanāgarī surface, even the roles of assignment and statement termination are symbolized by meaningful characters. For practicality, however, the early implementations may accept some ASCII operators alongside the preferred Vedic forms; the table below documents the intended canonical forms.

Role	Token (Devanāgarī or symbol)	Transliteration / alias	Explanation
Assignment	ः	visarga	The visarga marks association (name ः value)
Statement terminator	।	danda	End of sentence / statement, mirroring Sanskrit orthography
Block start	⨯	---	Introduces a block (like {)
Block end	৺	---	Ends a block (like })
Call / parameter open	[	---	Function or macro call argument list, open
Call / parameter close	]	---	Function or macro call argument list, close

Addition	TBD in part II	+	Encoded in the numerals as in Aryabhatiya (v0: internally mapped to ADD)
Subtraction	TBD in part II	–	Encoded in the numerals as in Aryabhatiya Subtraction
Multiplication	TBD in part II	*	Encoded in the numerals as in Aryabhatiya Multiplication
Division	TBD in part II	/	Encoded in the numerals as in Aryabhatiya Division
Equality comparison	तुल्य	tulya	Keyword operator for equality (v2 sugar)
Inequality comparison	अतुल्य	atulya	Keyword operator for inequality
Logical and	अन्वय	anvaya	Conjunctive logical and
Logical or	व्यतिरेक	vyatireka	Disjunctive logical or
Logical not	निषेध	nisedha	Negation operator

3.4 Virtual machine opcodes (v0–v2)

The following table aligns conceptual VM operations with their Devanāgarī names. The actual bytecode values (0x00, 0x01, etc.) are assigned in a separate VM specification; here we focus on the semantic token names.

Opcode (abstract)	Western label	Devanāgarī token	Transliteration	Role / meaning
OP_SHU	NOP	शु	śu / śūnya	Neutral operation, time–filler, alignment
OP_PUSHI8	Push 8--bit	प्रवेश	praveśa	Push 8--bit immediate on stack

OP_PUSHI16	Push 16--bit	दीर्घप्रवेश	dīrgha–praveśa	Push 16--bit immediate
OP_POP	Pop	विसर्जन	visarjana	Discard top of stack
OP_DUP	Duplicate	पुनरावृत्ति	punarāv□tti	Duplicate top stack value
OP_SWAP	Swap	परिवर्त	parivarta	Swap top two stack values
OP_ADD	Add	परिवर्तनम्	parivartanam	Add two values, return sum
OP_SUB	Subtract	व्यवकलन	vyavakalana	Subtract, LHS minus RHS
OP_MUL	Multiply	गुणन	gunana	Integer multiplication
OP_DIV	Divide	भागहार	bhāgahāra	Integer division
OP_MOD	Remainder	शेष	sesa	Remainder after division
OP_CMP_EQ	Compare equal	तुल्य	tulya	Push 1 if equal, else 0
OP_CMP_NE	Compare not equal	अतुल्य	atulya	Push 1 if not equal
OP_CMP_LT	Compare less	हीन	hīna	LHS less than RHS (signed)
OP_CMP_GT	Compare greater	श्रेष्ठ	srestha	LHS greater than RHS
OP_AND	Logical and	अन्वय	anvaya	Logical conjunction
OP_OR	Logical or	व्यतिरेक	vyatireka	Logical disjunction
OP_NOT	Logical not	निषेध	nisedha	Logical negation
OP_JMP	Unconditional jump	संचरण	samcarana	Move instruction pointer

OP_JZ	Jump if zero	हेतु	hetu	Conditional branch on false
OP_JNZ	Jump if nonzero	उदाहरण	udāharana	Conditional branch on true
OP_CALL	Call subroutine	आह्वान	āhvāna	Invoke function, push return address
OP_RET	Return	प्रत्यावर्तन	pratyāvartana	Pop return address, jump back
OP_HALT	Halt	निरोध	nirodha	Stop program execution
OP_PRINT_INT	Print integer	अङ्कउक्ति	aṅka–ukti	Print top of stack as integer
OP_PRINT_CHAR	Print char	अक्षरउक्ति	aksara–ukti	Print top of stack as byte
OP_PRINT_STR	Print literal string	वाक्यउक्ति	vākya–ukti	Emit string embedded in code
OP_PIN_MODE	Pin mode	संधान	sandhāna	Configure pin direction
OP_DIGITAL_WRITE	Digital write	स्पर्शलेख	sparśa–lekha	Write digital value to pin
OP_DIGITAL_READ	Digital read	स्पर्शपाठ	sparśa–pātha	Read pin as 0/1
OP_ANALOG_READ	Analog read	रसपाठ	rasa–pātha	Read analog sensor (0--1023)
OP_DELAY_MS	Delay	कालविराम	kāla–virāma	Pause execution for milliseconds

3.5 Built–in hardware functions (source–level)

At the language level, the following built–ins will correspond to one or more VM opcodes, but appear to the programmer as function calls:

Purpose	Devanāgarī name	Transliteration	Role
Print integer	अङ्क_लिखो	aṅka_likho	Print integer to serial
Print string	लिखो	likho	Print a string
Read digital pin	स्पर्श_पढ़ो	sparśa_padho	Return 0/1 for a pin
Write digital pin	स्पर्श_लिखो	sparśa_likho	Set digital pin high or low
Read analog pin	रस_पढ़ो	rasa_padho	Return sensor value
Wait for time	रुको	ruko	Delay in ms

3.6 ML–related tokens (planned for v2)

For early ML capabilities, most heavy training will occur off–device. On the Arduino side, we primarily need tokens for invoking small models and handling their parameters.

Concept	Devanāgarī token	Transliteration	Role
Model construct	मॉडल	mōdal	Declare a model symbol
Model run / predict	पूर्वानुमान	pūrvānumāna	Invoke model with inputs
Small tensor	सूत्र_समूह	sūtra_samūha	Compact vector/matrix (fixed shape)
Load weights	भार_स्थापन	bhāra_sthāpana	Initialize weight values in flash
Linear mapping	रेखा_परिवर्तन	rekhā_parivartana	One affine transform step

Threshold	सीमा	sīmā	Apply threshold (e.g. step function)

These tokens are sufficient to encode simple control policies and decision rules learned elsewhere and embedded on the Arduino as constant parameters.

4 Summary and outlook

This Part I specification has assembled:

- a concise statement of overall design goals for सुगणिता,
- a phased development plan focused initially on Arduino Uno,
- and a concrete, implementable token set for language keywords, operators, opcodes, hardware primitives, and basic ML hooks for versions v0--v2.

Three design choices illustrate this conceptual alignment. First, शून्य replaces a neutral "NOP" with a meaningful sunya operation. Second, control-flow operations like conditional jump are associated with हेतु (cause) and उदाहरण (example), echoing Nyāya structure. Third, निरोध (cessation) gives semantic depth to the idea of halting computation.

In Part II, ``Grammar and structural forms'', the next steps will be:

1. to define the precise lexical rules for identifiers, including permissible Devanāgarī codepoints and combining marks;
2. to describe the context--free grammar for expressions, statements, and module structure;
3. to specify how control flow constructs map systematically to the underlying opcodes;
4. and to demonstrate small complete programs in सुगणिता and their compiled bytecode.

Subsequent parts will address the virtual machine layout, the Arduino C++ implementation, and the eventual AVR assembly version. Over time, as the language stabilizes, the token set may grow; however, the aim will be to preserve the conceptual clarity and Vedic orientation recorded in this first blueprint.

Acknowledgments

This work was produced with the assistance of large language models.

Tradition	Core idea	Language feature layer	Example tokens	Implementation notes
Vedic mathematics	Sūtra-driven computation (e.g., ऊर्ध्वतिर्यग्भ्याम्: vertically and crosswise)	Arithmetic semantics and future specialized ops	गुणन (OP_MUL), भागहार (OP_DIV), शेष (OP_MOD)	Start with integer ops (v0–v2), reserve opcode space for patterned arithmetic; validate cost on 8-bit AVR before inclusion.
Nyāya logic	Five-part inference (प्रतिज्ञा, हेतु, उदाहरण, उपनय, निगमन)	Control flow semantics and branching vocabulary	यदि (if), अन्यथा (else); हेतु (OP_JZ), उदाहरण (OP_JNZ)	Model conditional jumps as cause/example; later sugar for multi-claim branching; keep jump targets 16-bit for compact code.
Pāṇinian grammar	Rule-governed transformations (सूत्र, संधि)	Lexical discipline and parsing transforms	दण्ड (I as statement end), विसर्ग (: association), आह्वान (OP_CALL), प्रत्यावर्तन (OP_RET)	Deterministic tokenizer over Devanāgarī; UTF-8 treated as opaque bytes in VM; grammar-level transforms constrained to linear-time passes.
Sāṅkhya and Yoga	Purusha/Prakṛti distinction; निरोध (cessation)	Execution state and halting semantics	निरोध (OP_HALT), शु (OP_SHU; सून्य pause)	HALT maps to a quiescent VM state; NOP as constructive pause for timing/alignment; expose timing semantics via कालविराम (OP_DELAY_MS).
Vākya tradition (śāstric clarity)	Meaningful surface aligned with roles	Surface tokens and keyword design	कार्य (function), मुख्य (entry), प्रकार (type), वाक्य (string), रिक्त (void)	Prefer semantically grounded lexemes over transliteration; maintain compact encodings; provide ASCII aliases only for early tooling.
Early ML praxis (embedded)	Small, fixed models; affine maps and thresholds	High-level ML hooks and compact tensors	मॉडल, पूर्वानुमान, सूत्र_समूह, भार_स्थापन, रेखा_परिवर्तन, सीमा	Host-side training; device-side fixed-point inference; weights in flash; preserve VM simplicity by treating ML as callable primitives.

Table 1: Mapping philosophical traditions to language and VM features (v0–v2 focus, with reserved room for later expansion).