RAMTERA TECHNICAL WHITE PAPER SERIES

WP-09

Twelve Thousand Ways to Say It

What SVRF rule deck coverage actually means across eight process nodes

Coverage is quoted as a percentage. The percentage is meaningless unless you know what it is a percentage of.

~12,000	8	~5,000	200+
DISTINCT CONSTRUCTS	PROCESS NODES	ON ONE NODE ALONE	GEOMETRIC PRIMITIVES

Document	WP-09 — Twelve Thousand Ways to Say It
Author	Sameer Pawanekar, Founder & Chief Executive Officer, Ramtera Inc.
Revision	v1.0
Date	August 2026
Classification	Public

ABSTRACT

Physical verification tools are routinely described as supporting a rule language, and their coverage of that language is quoted as a percentage. Both statements conceal more than they reveal. This paper sets out what a foundry SVRF rule deck actually contains, how many distinct constructs a tool must implement to execute one, how that number grows across process nodes, and why the denominator of any coverage percentage matters more than the numerator. It reports the construct counts observed across eight process nodes spanning four foundry generations.

01 What is in a rule deck

A foundry rule deck is an executable specification of manufacturability. It is not a list of minimum widths and spacings; it is a program.

A deck defines derived layers by boolean and geometric operations on drawn layers, and then applies constraints to those derived layers. It describes connectivity, so that spacing rules can be conditioned on whether two shapes are electrically the same net. It recognises devices from layer combinations. It expresses density and fill requirements over sliding windows. It encodes multi-patterning colouring constraints, antenna ratios, and design-for-manufacturability checks that are advisory rather than blocking.

A deck for an advanced node runs to tens of thousands of statements, organised across many files, with preprocessor-style conditional inclusion selecting which portions apply for a given metal stack, device set, or design style.

> **CONDITIONAL INCLUSION IS PART OF THE LANGUAGE**
> The conditional structure is not decoration. A single deck can describe several materially different rule sets depending on the options selected at invocation. A tool that resolves those conditions incorrectly runs a coherent, self-consistent, wrong rule set — and reports clean.

02 Why the construct count is so large

The number of *rules* in a deck and the number of *constructs* a tool must implement to execute it are different quantities, and the second is much larger than the first.

Operand ordering

The same logical check can frequently be written with its operands in more than one order, all legal. A tool must accept every ordering, and each ordering is a distinct statement form to be recognised.

Optional modifiers

Constraints carry optional qualifiers — edge orientation, projection requirements, corner handling, region restrictions, connectivity conditions. Each combination that appears in a real deck is a form the tool must handle. The combinatorics are what drive the count.

Node-specific extensions

Foundries extend the language. A construct that exists at one node may not exist at another, and a construct that exists at both may carry different qualifiers. Supporting multiple nodes is therefore not a matter of parameterising one rule set; it is a union of overlapping but genuinely different vocabularies.

03 Observed construct counts

The figures below are the counts of distinct SVRF statement forms our parser must recognise to execute the decks concerned. They are counts of statement forms, not of rules.

Scope	Distinct constructs
A single advanced node	~5,000
Union across eight process nodes	**~12,000**
Geometric primitives implementing them	200+

Table 1 — Distinct SVRF statement forms.

The relationship between those two figures is the interesting part. Eight nodes do not require eight times the constructs of one, because the overlap is substantial; nor do they require only slightly more, because the divergence is real. The observed ratio is roughly two and a half.

The generational spread

Node class	Count	Character
Advanced FinFET	2	12 nm class; multi-patterning, complex device recognition
Advanced FD-SOI	1	22 nm class; body-bias and well structures
Mature CMOS	2	180 nm class, including high-voltage variants
Automotive CMOS	1	350 nm class; reliability and isolation rules
Further nodes	2	Supported; not characterised in this document

Table 2 — Generational spread of the supported node set. Individual nodes and foundries are not named in this document.

WHY THE RANGE MATTERS MORE THAN THE COUNT

The spread from 350 nm to 12 nm is what makes the union large. A mature automotive node and an advanced FinFET node share the basic geometric vocabulary and diverge almost everywhere else — multi-patterning, device recognition, and density requirements in particular have no counterpart at the mature end.

04 Executing a full deck set

Construct counts describe what a parser must recognise. Whether the tool can actually run a foundry's deck is a separate and stronger claim, and the honest unit for it is deck files executed, not percentages.

Measure	Status
Rule deck files parsed and executed, one advanced node	**All 99**
Rule language	SVRF, executed natively
Intermediate translation format	**None** — no conversion step
Deck supplied at	Run time, by the user

Table 3 — Deck execution status for one advanced node.

Why native execution matters here

An alternative architecture translates the foundry deck into an internal rule format, and then executes that. This introduces a translation stage whose fidelity is itself a source of error, and which must be re-validated every time the foundry issues a deck revision.

Executing the deck directly removes that stage. The consequence for coverage is significant: there is no category of rule that translates lossily, because nothing translates. A construct is either implemented or it is not, and the

distinction is visible rather than buried in a conversion report.

05 How to read a coverage claim

Coverage percentages are quoted constantly and mean very little without their denominator. The questions that make one interpretable:

Question	Why it matters
Percentage of what — rules, statements, or constructs?	These differ by an order of magnitude. A deck with 3,000 rules may contain 5,000 constructs.
On which node's deck?	Coverage on a mature node says little about an advanced one
Which deck revision?	Foundries revise decks; coverage is measured against a specific version
Parsed, or parsed and executed?	Accepting a statement and producing correct results from it are different achievements
Executed, or executed and correlated?	Producing results is not the same as producing the right results
Which conditional configuration?	A deck's conditional branches select different rule sets; coverage differs between them

Table 4 — Questions that make a coverage figure interpretable.

> **CONDITIONAL COVERAGE IS COVERAGE TOO**
> The last row is the one most often unstated and most often material. A tool can report high coverage on the configuration it was developed against and substantially lower coverage on a different metal stack selection from the same deck — because the conditional branches taken are different, and the untaken branches were never exercised.

The claim worth making

The most informative statement a tool can make is not a percentage at all. It is: *these deck files, at this revision, in this configuration, parse and execute; these specific constructs are unimplemented; here is the correlation of the results against a reference.* That is longer than a percentage and considerably more useful.

06 Conclusion

The scale of a modern rule deck is routinely underestimated. Roughly five thousand distinct statement forms for a single advanced node, and roughly twelve thousand across eight nodes spanning 350 nm to 12 nm, is the actual size of the implementation problem behind a physical verification tool that claims multi-node support.

Because that scale is invisible from the outside, coverage claims are difficult to evaluate and consequently easy to inflate. The remedy is not more precise percentages but a different unit: deck files executed, constructs unimplemented named explicitly, and correlation reported separately from coverage.

About Ramtera

Ramtera Inc. develops **LVR**, a physical verification engine that executes foundry SVRF rule decks natively, covering design rule checking, layout versus schematic, electrical rule checking, layout XOR, density and fill, and parasitic extraction. LVR is written in modern C++ with a Qt6 interface, and has been validated against publicly available process design kits including SKY130 and IHP SG13G2.

Ramtera also develops **PRE**, a place-and-route engine sharing LVR's geometry kernel, and maintains a portfolio of granted patents, registered copyrights and trademarks across its verification and implementation technologies.

Ramtera Inc. is incorporated in the United States, with research and development conducted by Ramtera Design Automation India Private Limited in Bangalore, India.

Contact	
Web	https://ramtera.com
General enquiries	info@ramtera.com
Technical support	support@ramtera.com
Series	Ramtera Technical White Paper Series
This document	WP-09, revision v1.0, August 2026