MATHEMATICS AND ARTIFICIAL INTELLIGENCE

The Possibility for a Machine-Native Mathematics

How mathematical reasoning may change when its representations and primitive operations can change with it.

Komorebi P.

Revised reading edition

29 October 2026

Author's note

I publish under the name Komorebi P. because I want the work to be judged first through its definitions, arguments, formal statements, code, and reproducible results. A name, biography, or institutional affiliation cannot do that work for a reader. The pseudonym gives those questions some distance from the claims. It is a working arrangement, rather than a commitment to permanent anonymity.

The Komorebi project has developed with substantial AI assistance. That assistance includes drafting, mathematical exploration, programming, calculations, and review. I am responsible for what I choose to publish and for correcting it. Neither AI assistance nor the existence of a formal development establishes that every claim in the collection is correct. The public project and authorship statement explains these responsibilities and the project's publication circumstances.

About this edition

This essay considers how mathematical reasoning might change when a system can revise the representations and operations available to it. It explains the conceptual architecture of Komorebi. The individual papers, formal developments, and software packages remain the sources for their own technical claims.

This reading edition brings that account together with the current papers and companion studies. It revises the prose and presentation, makes the assumptions in the mathematical examples explicit, and supplies an editable manuscript. The references lead to the reading editions and their supporting evidence.

Contents

Author's note i
About this edition . . . i
1 Introduction 1
2 Representation, abstraction, and residuals 2
2.1 Changing a representation . . . 2
2.2 Abstraction as controlled identification . . . 2
2.3 The canonical task abstraction . . . 3
2.4 Which future operations are allowed? . . . 3
2.5 Comparing abstractions . . . 4
2.6 Measuring a failed transport . . . 5
2.7 Refinement and the depth of a distinction . . . 6
2.8 Laws acting on abstractions . . . 6
3 Correspondence and the Langlands test bed 8
3.1 Why correspondence matters . . . 8
3.2 Papers X, XI, and XII . . . 8
3.3 A possible machine-facing record . . . 9
4 Mathematical artifacts for machine reasoning 10
4.1 What machine-native means here . . . 10
4.2 More than one form of a result . . . 11
4.3 Understanding and reusable structure . . . 11
4.4 Human explanation and machine detail . . . 12
4.5 The corpus as a research object . . . 12
4.6 Related developments in the collection . . . 13
5 Learning, language growth, and self-improvement 14
5.1 Keeping what a reasoning episode discovers . . . 14
5.2 A state description for recursive improvement . . . 14
5.3 Capability under finite resources . . . 15
5.4 The productivity of improvement . . . 16
5.5 Stable abstractions and useful forgetting . . . 17
5.6 The cost of finding a representation . . . 17
5.7 Observation, intervention, and decisions . . . 18
6 What would count as progress? 19
6.1 A calculus that produces something new . . . 19
6.2 Five properties to look for . . . 19
6.3 A controlled empirical comparison . . . 20
6.4 A first completion criterion . . . 20
6.5 Questions for the next stage . . . 21
6.6 Closing . . . 21
References 22

CHAPTER 1

Introduction

Mathematics developed within communities of human readers. Its notation reflects what people can recognize and remember. Its proofs reflect what people find convincing and explanatory. Its subjects also reflect the history of the communities that taught and developed them. Even when the objects were far beyond ordinary perception, their descriptions had to pass between human minds.

Those choices have been extraordinarily productive. They should still be distinguished from the mathematical truths expressed through them. A useful notation is a way of reaching a result. It is not the only possible way to represent the objects involved.

Mathematics now has other readers. Proof assistants check formal derivations. Computer algebra systems work with representations chosen for calculation. AI systems can write programs, construct intermediate models, and search over executable objects. These developments raise a question that extends beyond the speed of theorem proving. What happens when a reasoner can help construct the mathematical language in which it works?

Komorebi approaches that question through representation change and abstraction. It asks how a mathematical object can be described, what a change of description preserves, and which distinctions can be discarded for a specified purpose. It also asks how a useful representation can become a reusable operation in later reasoning.

The practical cycle is straightforward to describe. Represent an object, choose an abstraction, transport the relevant structure, record any defect, refine what is missing, and reconstruct what the task requires. If the same pattern recurs, ask whether it deserves a mathematical name and a place in the system's working language.

An essay can explain why that cycle is worth studying. Showing that it produces useful mathematics requires more. The rest of this essay separates the elementary mathematical constructions from the proposed research architecture and the evidence that would be needed to judge it.

CHAPTER 2

Representation, abstraction, and residuals

2.1 Changing a representation

A mathematical object may have several representations,

$$R_1(X), \quad R_2(X), \quad R_3(X), \quad \dots.$$

A problem that is difficult in one description may be simple in another. Familiar mathematical methods already depend on this fact. Coordinates, transforms, presentations, and equivalences often succeed by exposing structure that was difficult to see in the original description.

Suppose that the object X has representations $R_\alpha(X)$. In addition to studying an object within one representation, one can study maps

$$T_{\alpha\beta} : R_\alpha(X) \longrightarrow R_\beta(X).$$

The map becomes part of the mathematical problem. We can ask which invariants survive, which operators commute with it, what information it loses, and whether the original description can be recovered. We can also ask how two such changes compose.

For a machine reasoner, choosing a representation could therefore be part of a formal search. The system would need to compare descriptions and their costs, rather than accept the initial formalization as a fixed starting point.

2.2 Abstraction as controlled identification

Changing a representation and discarding information are different operations. In the simplest setting, an abstraction of a set or state space X is a surjection

$$\alpha : X \twoheadrightarrow A.$$

It identifies states according to the equivalence relation

$$x \sim_\alpha y \quad \Longleftrightarrow \quad \alpha(x) = \alpha(y).$$

Unless α is invertible, some information is lost. Whether the loss is acceptable depends on the questions the abstraction must answer.

Let

$$\mathcal{F} = \{f_i : X \longrightarrow Y_i\}_{i \in I}$$

be a family of tasks or observations. The abstraction is sufficient for this family if every task factors through it. For each i, there must be a map $\bar{f}_i$ such that

$$f_i = \bar{f}_i \circ \alpha.$$

Here the kernel of a map means its equivalence relation of equal outputs. Write

$$E_\alpha = \ker(\alpha), \qquad E_{\mathcal{F}} = \bigcap_{i \in I} \ker(f_i).$$

Then

$$\alpha \text{ is sufficient for } \mathcal{F} \quad \Longleftrightarrow \quad E_\alpha \subseteq E_{\mathcal{F}}.$$

Every pair of states identified by α must already be indistinguishable to all the tasks under consideration. This is a precise version of the instruction to keep the information that matters.

2.3 The canonical task abstraction

The task family determines a quotient

$$Q_{\mathcal{F}} = X/E_{\mathcal{F}}, \qquad q_{\mathcal{F}} : X \twoheadrightarrow Q_{\mathcal{F}}.$$

This quotient answers every task in $\mathcal{F}$. If another abstraction $\alpha : X \twoheadrightarrow A$ is sufficient, there is a unique map $h : A \longrightarrow Q_{\mathcal{F}}$ with

$$q_{\mathcal{F}} = h \circ \alpha.$$

Thus $Q_{\mathcal{F}}$ is the coarsest exact abstraction sufficient for the chosen questions. Its universal property concerns that task family. Changing the questions can change the quotient.

2.4 Which future operations are allowed?

Two states that look the same now may look different after an operation. An abstraction designed only for the present observation can therefore discard a distinction that a later calculation needs.

Let a monoid M act on X through

$$\rho : M \longrightarrow \mathrm{End}(X),$$

and let $F : X \longrightarrow Y$ be an observation. The behavior map records every observation reachable through the allowed action:

$$\mathcal{B}_{\rho,F} : X \longrightarrow Y^M, \qquad \mathcal{B}_{\rho,F}(x)(m) = F(\rho_m x).$$

Its kernel is

$$x \sim_{\rho,F} y \quad \Longleftrightarrow \quad F(\rho_m x) = F(\rho_m y) \quad \text{for every } m \in M,$$

or, equivalently,

$$E_{\rho,F} = \bigcap_{m \in M} \ker(F \circ \rho_m).$$

The quotient $Q_{\rho,F} = X/E_{\rho,F}$ retains the distinctions that can become observable under the permitted operations. What can safely be forgotten depends on both the observation and the operation language.

Call an equivalence relation R on X M-stable if

$$xRy \quad \Longrightarrow \quad \rho_m x \, R \, \rho_m y \quad \text{for every } m \in M.$$

The behavioral relation is the greatest M-stable equivalence relation contained in $\ker(F)$:

$$E_{\rho,F} = \max\{R : R \subseteq \ker(F),\ R \text{ is an } M\text{-stable equivalence relation}\}.$$

This gives a specific abstraction problem. Discard as much state as possible while preserving every distinction that could affect a future admissible observation. Different subjects may supply different states, actions, and observations while sharing this form of question.

2.5 Comparing abstractions

Abstractions can themselves be ordered. Given surjections $\alpha : X \twoheadrightarrow A$ and $\beta : X \twoheadrightarrow B$, write $\alpha \preceq \beta$ when $\beta = h \circ \alpha$ for some map h. Equivalently,

$$E_\alpha \subseteq E_\beta.$$

The finer abstraction preserves more distinctions. The coarser one identifies more states. A joint abstraction is given by

$$(\alpha \otimes \beta)(x) = (\alpha(x), \beta(x)), \qquad E_{\alpha\otimes\beta} = E_\alpha \cap E_\beta.$$

Its codomain is the image of the pair map, so it remains a surjective abstraction even when that image is smaller than $A \times B$.

An exact change of representation $r : X \xrightarrow{\sim} X'$ preserves information. An abstraction $\alpha : X \twoheadrightarrow A$ determines which distinctions remain visible. Keeping these roles separate makes the resulting calculus easier to interpret.

For a map $r : X \longrightarrow Y$, equivalence relations have an inverse image r^*. Their pushforward r_* is the equivalence relation generated by the direct image, rather than the direct image alone. With these conventions,

$$r_*R \subseteq S \quad \Longleftrightarrow \quad R \subseteq r^*S, \qquad r_* \dashv r^*.$$

For an exact representation equivalence, these operations give inverse isomorphisms of the corresponding abstraction spaces. In this sense, the abstraction calculus does not depend on a particular choice of coordinates.

2.6 Measuring a failed transport

An attempted transport may fail to preserve an operator. In a linear setting, the failure can be measured directly. Let X and Y be vector spaces over the same field, let T_X and T_Y be linear endomorphisms, and let $f : X \longrightarrow Y$ be linear. Define

$$\partial_T(f) = T_Y f - f T_X.$$

The residual vanishes exactly when f intertwines the two operators. For composable linear maps $X \xrightarrow{f} Y \xrightarrow{g} Z$, it satisfies

$$\partial_T(gf) = \partial_T(g)f + g\partial_T(f).$$

The linear hypotheses matter. They make subtraction and the distributive rules for composition available. The same identity also makes sense in a suitable additive setting with bilinear composition.

For a chain $X_0 \xrightarrow{f_1} X_1 \xrightarrow{f_2} \cdots \xrightarrow{f_n} X_n$, the residual expands as

$$\partial_T(f_n \cdots f_1) = \sum_{k=1}^{n} f_n \cdots f_{k+1}\, \partial_T(f_k)\, f_{k-1} \cdots f_1.$$

The empty products at the ends of the chain are identity maps. This formula records how a local defect contributes to the final one. It can help identify where exactness was lost and which part of a construction needs repair.

2.7 Refinement and the depth of a distinction

An insufficient abstraction can be refined. Suppose the allowed operations have a specified nonnegative integer depth $\ell(m)$. Define

$$E_n = \bigcap_{\ell(m)\le n} \ker(F \circ \rho_m).$$

As the horizon grows, more observations are available and the relations become finer:

$$E_0 \supseteq E_1 \supseteq E_2 \supseteq \cdots, \qquad Q_0 \longleftarrow Q_1 \longleftarrow Q_2 \longleftarrow \cdots.$$

This refinement also gives a way to measure distinguishability. Let $\tau(x, y)$ be the least allowed operation depth at which an observation distinguishes x and y. Set $\tau(x, y) = \infty$ when no allowed operation distinguishes them. Then

$$d(x, y) = \begin{cases} 0, & \tau(x, y) = \infty, \\ 2^{-\tau(x,y)}, & \tau(x, y) < \infty. \end{cases}$$

This is a pseudoultrametric:

$$d(x, z) \le \max\{d(x, y), d(y, z)\}.$$

After quotienting by behavioral equivalence, it becomes an ultrametric. A small distance means that a distinction takes a deeper operation to reveal. The construction therefore gives a geometry to the observation horizon, as well as an order on abstractions.

2.8 Laws acting on abstractions

Once abstractions form a mathematical space, operations on that space can be studied in their own right.

Here abstractions are identified through their kernel equivalence relations, or equivalently up to isomorphism of their quotient targets. This convention gives a set of abstractions on the fixed set X.

An operation of arity n has the form

$$\Lambda : \mathrm{Abs}(X)^n \longrightarrow \mathrm{Abs}(X).$$

Refinement, coarsening, joint abstraction, and closure under an action are examples of the kinds of operation one might study. This suggests several levels of description: objects, representations of objects, abstractions, laws acting on abstractions, algebras of those laws, and policies that select or construct them.

A law intended to describe structure independent of coordinates should respect exact changes of representation. Schematically, for $r : X \xrightarrow{\sim} X'$, one asks for

$$r_* \Lambda_X(\alpha_1, \dots, \alpha_n) \simeq \Lambda_{X'}(r_*\alpha_1, \dots, r_*\alpha_n).$$

The equivalence here compares abstractions through their corresponding quotient data. Such a condition helps separate a construction that depends on convenient coordinates from a law that survives a change of description.

There is a related question between domains. Suppose a translation $\Phi : \mathcal{C} \longrightarrow \mathcal{D}$ induces a map Φ_* on abstraction spaces. Given a law $\Lambda_{\mathcal{C}}$, does a corresponding law $\Lambda_{\mathcal{D}}$ satisfy

$$\Phi_* \Lambda_{\mathcal{C}} = \Lambda_{\mathcal{D}} \Phi_*'^{n}?$$

Transporting a law can provide a method for constructing a family of consequences. It requires more than transporting one theorem, because the operation itself must remain meaningful in the target setting.

Recurring procedures often become named mathematical objects. A calculation becomes an operator, a symmetry becomes a group action, or a decomposition becomes a transform. A reasoning system might make that promotion explicit. When a sequence of compatible operations $\alpha_1, \dots, \alpha_k$ recurs, its composite

$$\gamma = \alpha_k \circ \cdots \circ \alpha_1$$

may be worth keeping as a new primitive. Later searches can then use the composite directly. The question is whether this change improves later reasoning enough to justify its cost.

CHAPTER 3

Correspondence and the Langlands test bed

3.1 Why correspondence matters

The Langlands program is a useful precedent for thinking about mathematical translation. Arithmetic and automorphic objects can remain different while correspondences relate selected aspects of their structure. The value of such a correspondence depends on the compatibility data it preserves.

At a broad schematic level, one encounters global arithmetic data, local Galois or Weil data, local automorphic data, and global automorphic data. Passing between them requires careful control of Frobenius characteristic polynomials, inertia, monodromy, conductors, local factors, determinants, epsilon data, and global compatibility. The relevant correspondences belong to the classical theory. They are not new theorems supplied by Komorebi.

This provides a concrete meaning for the kind of universality considered here. Several domains might share methods for describing a correspondence, recording its hypotheses, composing its maps, and locating failures of compatibility. They need not share one normal form or become the same mathematical subject.

3.2 Papers X, XI, and XII

Komorebi Papers X, XI, and XII develop three connected parts of this arithmetic example: residual seeds and transport, local packet geometry, and patching with Hecke data. Each paper states which constructions it supplies and which arithmetic inputs it requires.

Paper X, *Universal Residual Seeds*, organizes deformation forests, transport categories, and conditional system completion. Its finite arithmetic separator uses a supplied Frobenius occurrence input. Its carrier construction closes supplied compatible data under a selected finite constituent packet, with the required local comparison laws stated explicitly.

The conditional completion theorem has two separate outputs: a transported target family and a recovered summand inside a supplied carrier. Identifying them requires a further comparison. The actual source members, carrier, and arithmetic attachment data remain inputs. This distinction matters whenever a formal transport construction is used to discuss an arithmetic object. See 1.

Paper XI, *Tame and Wild Packet Geometry for Local Langlands*, develops finite Frobenius packets, tensor refinement, Gauss phases, finite Heisenberg groups, projective commutator envelopes, and gluing on finite transport graphs. Its restricted arithmetic constructions include odd and dyadic families under their stated classical inputs. The general arithmetic realization problem remains Conjecture 66.

The open work is now more specific than a request for abstract local data. It includes constructing the arithmetic candidate, establishing the full characterizing local-factor identities for that same candidate, and supplying the required character and phase data in the stronger cases. A conditional comparison of supplied candidates and a restricted tame attachment are established at their own scopes. They do not supply the general existence statement. See 2.

Paper XII, *Patching Support and Hecke Packets*, develops local quotient conditions, support and patching interfaces, compatible-system constructions, and conditional automorphic completion. Its finite and completed constructions organize how compatible data can be assembled and how global points factor through the required local conditions.

An arithmetic application still supplies its deformation problem, applicable patching package, full patched support, and a compatible Hecke point over the intended coefficient field. Classicality and the attached representation must concern the selected eigensystem. Propagation compares already existing compatible families under common hypotheses. It does not create missing companions, and it does not assume a solution of Paper XI's general realization problem. See 3.

Together, the papers show how a common calculus can keep transport, arithmetic realization, and global assembly distinct while expressing the conditions that connect them. The usefulness of the interface depends on preserving those conditions through each step.

3.3 A possible machine-facing record

A local component might be recorded schematically as

$$\mathcal{L}_v = (K_v, W_{K_v}, I_v, \rho_v, N_v, \mathrm{Frob}_v, P_v, a_v, \varepsilon_v, \Delta_v, \mathrm{Cert}_v).$$

The tuple is an interface proposal. It records the local field, Weil data, inertia, representation, monodromy, Frobenius, local polynomial, conductor, epsilon information, compatibility defects, and certificates that need to agree. A global record would also need compatible-family data, patching constraints, and the bridge that produces the intended automorphic or Galois attachment.

Langlands seeks deep correspondences between particular arithmetic and automorphic worlds. Komorebi's proposed role is to help express, compose, and inspect correspondence structures. Whether it can help discover new ones is a further research question. A common interface would be useful only if it preserves the distinctions on which the underlying mathematics depends.

CHAPTER 4

Mathematical artifacts for machine reasoning

4.1 What machine-native means here

I use *machine-native mathematics* for mathematical representations designed with machine reasoning as a central constraint. The phrase concerns how objects and operations are organized. Generating a proof with a machine, or writing a proof that a machine can check, is not sufficient by itself.

A machine-facing representation may need explicit transformation contracts, typed invariants, verifiable certificates, residuals, abstraction operators, reconstruction maps, and structural signatures that survive a change of coordinates. Human exposition can describe those objects without carrying all their operational detail in the same form.

There are already suggestive examples of systems constructing useful intermediate descriptions. ARC Prize reported that GPT-6 Astra developed compact symbolic notes while working on unfamiliar ARC-AGI-3 environments. These notes described objects, rules, state, and plans. In a separate tool-enabled harness, the model also built game-specific parsers, planners, search procedures, and software libraries. ARC Prize distinguished those evaluation settings, which matters when interpreting the results. See 4.

OpenAI's ARC-AGI-3 report concerns another part of this problem. Preserving reasoning between actions and replacing rolling truncation with compaction improved the reported results for GPT-5.6 Sol while reducing output-token use. This supports a bounded observation about that harness: retaining useful state can change the reasoning that remains possible over a long interaction. It does not establish that the system created a new mathematical calculus. See 5.

AlphaEvolve offers a different example. Google DeepMind describes a system that searches over executable algorithmic candidates, evaluates them, and uses successful constructions in further searches across mathematics and computing. Executable objects become material for an ongoing search, rather than only the final output of one. See 6.

These examples motivate the question in this essay. They are not implementations or validations of Komorebi. The proposed next step is to describe representation construction as explicit mathematics and test whether doing so helps a reasoning system solve problems it could not otherwise solve as effectively.

4.2 More than one form of a result

A mathematical construction can be represented by a definition, a program, a proof object, a test harness, and rules for further transformations. These forms have different responsibilities. A test can check selected behavior. A proof object can certify a formal statement. An implementation can instantiate the construction. Their agreement needs to be established, rather than inferred from the fact that they share a name.

Execution can nevertheless be part of the mathematical artifact itself. An object with a defined interface can be instantiated, transformed, and used by another program or reasoner. A paper may explain that interface, while source files and formal objects specify details that would be cumbersome to carry in prose.

A compiler's intermediate representation provides a useful analogy. Several source languages may translate into a common operational form, undergo transformations there, and then be reconstructed in a target language. A mathematical intermediate representation would have to retain more than syntax. A schematic record might be

$$\mathsf{K} = (X, R, \mathcal{F}, \mathcal{L}, \alpha, \mathcal{I}, \Delta, \mathrm{Ref}, \mathrm{Rec}).$$

Here X is the mathematical system, R its present representation, $\mathcal{F}$ the task family, and $\mathcal{L}$ the admissible transformation language. The remaining fields record an abstraction, preserved invariants, structural defects, refinement data, and reconstruction data. A certified transformation between records would state what it preserves and what conditions its output requires.

That proposal creates an engineering and mathematical obligation. The fields must have precise types, the transformations must respect them, and the certificates must refer to the same objects as the implementations. Naming the fields does not discharge those obligations.

4.3 Understanding and reusable structure

A proof establishes a consequence under stated assumptions. An explanation often goes further by showing why several consequences follow from one mechanism. Finding that mechanism can reduce the amount of separate reasoning needed later.

A machine-facing calculus could make this form of compression explicit. It could search for representations in which a collection of results becomes instances of one abstraction or transport law. The result would be useful if the law can be applied again, with its assumptions and costs understood.

This broadens the search problem. A conventional automated prover is often given a formal language and asked to find a derivation. A system with representation search would also choose descriptions, construct abstractions, compare translations, and decide which auxiliary objects to introduce. A successful procedure might then be compiled into a primitive available to later searches.

The distinction is practical. A shorter proof in a fixed language and a better language for a family of problems are different achievements. A useful evaluation should be able to recognize either one and measure what it costs.

4.4 Human explanation and machine detail

Machine-native representations need not be inaccessible to people. A useful architecture should support movement between human and machine descriptions. A machine may benefit from an extensive record of types and dependencies. A human reader may need a conceptual account that suppresses routine execution detail.

The reconstruction back to a human-facing account need not reproduce every internal step in familiar notation. It should expose the mathematical objects, assumptions, invariants, and consequences needed to understand what was established and how it relates to existing work.

Formal certainty and intelligibility should therefore be treated separately. A derivation accepted by a proof assistant may be difficult to explain. A persuasive argument may omit details needed by a formal checker. Neither difficulty is resolved by calling one format authoritative for every purpose.

Provenance adds another responsibility. A proof certificate answers a question about validity within a formal system. A provenance record explains where the certified object came from, which sources it used, and what transformations were applied. A record may need to retain discarded information, unresolved defects, and assumptions that remain external to the certificate.

4.5 The corpus as a research object

A research project can organize these materials as a structured corpus. Formal definitions, theorem statements, proof objects, implementations, computational witnesses, tests, counterexamples, dependency graphs, and representation maps can be related explicitly.

Different readers could then request different views. A person might follow the conceptual argument. A proof assistant might read the dependency graph. A computational agent might use an executable construction. Another reasoning system might request its abstraction and transport interfaces.

Calling these views part of one corpus does not make them agree automatically. Their correspondence is an obligation of the project. If that obligation is met, the paper can become one carefully prepared view of a larger mathematical object.

4.6 Related developments in the collection

Several companion studies give these questions a more concrete form. *Machine-Native Reasoning* studies evidence, correction, and the memory needed to retain distinctions that later evidence can reveal. Its reviewed core includes an exact fixed-width memory bound for correction under partition evidence. This connects the choice of an abstraction to what a reasoner must remember, rather than only to the size of its present description. See 7.

Proof Path Theory studies discovery fields, certificates, compression, and mathematical memory. Its finite trace checker requires both valid derivation steps and a final saturation check to certify the least seeded closure. A typed exchange format lets a proposed certificate be checked against a separately supplied problem. The proved Lean checker and codecs have a defined boundary, while the Python implementation and text interface retain their own testing obligations. See 8.

Realization Boundaries asks what observations fail to determine. It gives families with the same unmarked finite-quotient data and profinite completion but different isomorphism types and computable presentation requirements. A common observation system can therefore leave important realization information unresolved. This is a concrete reason to record what a representation forgets. See 9.

Stratified Relational Workspace Theory extends the question to people, models, artifacts, and computational processes that each hold only part of a shared structure. It studies persistent invariants under transfer, joint reconstruction, and tests of whether the relevant capacity belongs to their interaction. The proposed empirical certification still requires review and testing. These studies connect representation change to memory, proof exchange, reconstruction limits, and collaboration. See 10.

CHAPTER 5

Learning, language growth, and self-improvement

5.1 Keeping what a reasoning episode discovers

Suppose Π_t is a policy for choosing representations and abstractions. A reasoning episode produces evidence D_t, after which the policy may change:

$$\Pi_{t+1} = U(\Pi_t, D_t).$$

If the episode discovers and certifies a law Λ_t, the law library may also grow:

$$\mathscr{L}_{t+1} = \mathrm{Cl}(\mathscr{L}_t \cup \{\Lambda_t\}).$$

The closure operation must be specified by the system. The intended process is to solve a problem, identify reusable structure in the solution, certify it, and make it available to future reasoning. Whether the new material helps depends on how it is represented and searched.

5.2 A state description for recursive improvement

Recursive improvement need not mean a system changing its neural weights. It can also occur in the mathematical machinery that the system uses to produce later improvements.

Write a reasoning state as

$$\mathfrak{S}_t = (L_t, \mathcal{T}_t, \mathscr{L}_t, \Pi_t, \mathcal{V}_t).$$

The fields are the operative language, the certified theorem and construction library, the library of representation and abstraction laws, the strategy policy, and the verifier or certificate discipline. On problem data P_t, a reasoning episode produces a trace

$$D_t = \mathrm{Solve}(\mathfrak{S}_t, P_t).$$

A modification generator proposes

$$\mu_t = G(\mathfrak{S}_t, D_t).$$

The proposal might add a theorem, introduce an abstraction law, turn a recurring procedure into a primitive, or change the policy that proposes future modifications. If it meets the current certification obligations,

$$\mathcal{V}_t(\mu_t) = 1,$$

the state can be updated by

$$\mathfrak{S}_{t+1} = U(\mathfrak{S}_t, \mu_t).$$

There are at least three distinct kinds of change to keep apart. Knowledge accumulation enlarges $\mathcal{T}_t$ while leaving the language and improvement policy essentially fixed. Calculus growth enlarges the reusable language or law library. Recursive improvement changes machinery that participates in producing later improvements, such as Π_t or the law-generating part of G.

Schematically, these changes can be written as

$$\mathcal{T}_t \subsetneq \mathcal{T}_{t+1},$$

$$L_t \subsetneq L_{t+1} \quad \text{or} \quad \mathscr{L}_t \subsetneq \mathscr{L}_{t+1},$$

and

$$(\Pi_t, G_t) \longrightarrow (\Pi_{t+1}, G_{t+1}).$$

Adding a theorem alone is knowledge accumulation. A primitive that changes how later searches are carried out can be part of a recursive mechanism. A policy that improves the process of finding such primitives is a stronger case. Those distinctions need to be tied to an observable gain, rather than inferred from a change in notation.

5.3 Capability under finite resources

Fix a problem distribution P, a resource budget B, and a scoring rule $s(p, a)$. Let $\Sigma_B(\mathfrak{S})$ be the strategies expressible by state $\mathfrak{S}$ within that budget. A bounded capability functional is

$$C_B(\mathfrak{S}; \mathsf{P}) = \mathbb{E}_{p\sim\mathsf{P}}\left[\sup_{\sigma\in\Sigma_B(\mathfrak{S})} s\big(p, \sigma(p)\big)\right].$$

The expression requires an agreed cost model, a defined convention for the feasible strategy set, and a score for which the supremum and expectation are meaningful. The scoring rule is external to the system's own account of its progress.

Relative to these choices, an update gives a certified capability improvement if it preserves its verification obligations and satisfies

$$C_B(\mathfrak{S}_{t+1}; \mathsf{P}) > C_B(\mathfrak{S}_t; \mathsf{P}).$$

It is recursive in the sense used here if the modification was generated using the earlier state and changes a component involved in generating later modifications.

There is a simple monotonicity observation. Suppose a conservative extension preserves every previously feasible strategy at the same or lower cost, so that

$$\Sigma_B(\mathfrak{S}_t) \subseteq \Sigma_B(\mathfrak{S}_{t+1}).$$

If the external scoring rule is unchanged and the quantities are defined, then

$$C_B(\mathfrak{S}_{t+1}; \mathsf{P}) \geq C_B(\mathfrak{S}_t; \mathsf{P}).$$

This follows from taking the supremum over a larger feasible set for each problem and then the expectation. It does not guarantee a strict improvement. It also does not guarantee that a particular implemented search policy will find the best available strategy. A system can acquire more expressive resources and still use them poorly.

5.4 The productivity of improvement

A stronger question concerns how efficiently a system discovers useful updates. One possible definition is

$$\Gamma_B(\mathfrak{S}) = \mathbb{E}\left[\frac{\big(C_B(U(\mathfrak{S}, \mu); \mathsf{P}) - C_B(\mathfrak{S}; \mathsf{P})\big)_+}{\mathrm{Cost}_{\mathrm{imp}}(\mu)}\right], \qquad (x)_+ = \max(x, 0).$$

Here the expectation is over proposals made by the current improvement policy. Improvement cost must be positive on those proposals, and the expectation must be defined. Both the proposal distribution and the cost model are part of the specification.

On this definition, recursive acceleration would require

$$\Gamma_B(\mathfrak{S}_{t+1}) > \Gamma_B(\mathfrak{S}_t).$$

The abstraction calculus does not imply that inequality. An improvement process can plateau, cycle, regress, or find changes whose costs outweigh their value. The expression identifies a claim to test.

A relative stopping condition can also be stated. Fix a generator G, verifier $\mathcal{V}$, distribution P, and budget B. A state $\mathfrak{S}^*$ is stable for that improvement mechanism if

$$\nexists \mu \in G(\mathfrak{S}^*) \quad \text{such that} \quad \mathcal{V}(\mu) = 1 \\ \text{and} \quad C_B(U(\mathfrak{S}^*, \mu); \mathsf{P}) > C_B(\mathfrak{S}^*; \mathsf{P}).$$

This says that the specified mechanism has no available certified update with a strict capability gain. It does not say that no better reasoning system exists.

Komorebi proposes a possible route through this loop. A trace can reveal a recurring representation change or abstraction pattern. A certified law extracted from that pattern can enlarge the law library, change the search policy, and alter how the next problem is approached:

$$\mathfrak{S}_t \longrightarrow D_t \longrightarrow \Lambda_t \longrightarrow \mathscr{L}_{t+1} \longrightarrow \Pi_{t+1} \longrightarrow \mathfrak{S}_{t+1}.$$

The empirical burden is to make this loop productive over repeated problems. Writing its steps symbolically is a specification, not evidence that a system can perform them with a measurable gain.

5.5 Stable abstractions and useful forgetting

A small abstraction can be brittle. A slight change in the task or permitted operations may force a system to recover distinctions it has discarded. The coarsest exact quotient for today's question may therefore be a poor working representation for tomorrow's questions.

If the task family changes from $\mathcal{F}$ to $\mathcal{F}'$, its canonical quotient can change from $Q_{\mathcal{F}}$ to $Q_{\mathcal{F}'}$. It is natural to ask how large that change is in an agreed comparison structure. An abstraction condition number might be sought in the schematic form

$$\kappa_{\mathrm{abs}} \sim \frac{\text{change in the selected abstraction}}{\text{change in task or operator specification}}.$$

This is a research proposal. Distances or other comparison measures on the numerator and denominator would have to be defined before it became a numerical quantity. The underlying design question is clear: how much extra structure is worth retaining to make an abstraction useful across nearby tasks?

A growing law library raises a complementary problem. A primitive can become redundant, dominated by another procedure, or too costly to keep in active search. A useful system may need to promote, reuse, measure, compress, and sometimes forget its primitives. Growth in the number of available names is not by itself a measure of progress.

5.6 The cost of finding a representation

Let $C(P \mid R)$ denote the resources needed to solve a problem P in representation R. The right representation can reduce this cost considerably. Finding and maintaining it still cost something.

A possible objective is

$$C^*(P) = \inf_R \left[C_{\text{discover}}(R) + C(P \mid R) + \lambda C_{\text{maintain}}(R) \right], \qquad \lambda \geq 0.$$

The weight and all three cost terms must be specified for a concrete use. The formula expresses a practical point: a useful change of coordinates can move part of the work into representation discovery. An evaluation that starts after the formalization has been supplied may miss that work.

Human and machine criteria of elegance may overlap without coinciding. A person may value a short visual argument. A reasoning system may benefit from a description that lowers branching, certificate size, transport cost, or the cost of future reuse. A representation that looks cumbersome to a human could still reveal an economical algebra of transformations. Its value should be judged through those properties and through the explanations that can be reconstructed from it.

5.7 Observation, intervention, and decisions

Predicting observations, choosing interventions, and selecting decisions are different tasks. A representation sufficient for one need not be sufficient for another.

If admissible operations include interventions $a \in \mathcal{A}$, one can ask that

$$F(\rho_{m,a} x) = F(\rho_{m,a} y)$$

hold for every permitted operation and intervention. The resulting equivalence depends on that enlarged family of questions.

For a specified decision problem, let $\pi^* : X \longrightarrow A$ be a chosen optimal decision rule. An abstraction $\alpha : X \longrightarrow Q$ is sufficient for that rule when

$$\pi^* = \bar{\pi} \circ \alpha$$

for some $\bar{\pi}$. One can separately ask for value sufficiency,

$$V^*(x) = \bar{V}(\alpha(x)).$$

These are separate factorization requirements. They do not form an unconditional hierarchy. A chain of increasingly demanding sufficiency conditions can be obtained only after the task families and action languages are specified with the required inclusions. In particular, value sufficiency should not be called stronger than decision sufficiency without further assumptions.

The distinction matters for mathematical agents. Choosing a lemma, changing a representation, or introducing an auxiliary object changes the state of a reasoning problem. A useful state abstraction must preserve what those choices need, rather than only compress the symbols currently on the page.

CHAPTER 6

What would count as progress?

6.1 A calculus that produces something new

Komorebi does not need every mathematical subject to share a privileged representation. Its proposed common ground is a collection of laws concerning what can be preserved, discarded, transported, recovered, or reorganized between descriptions. Whether those laws are useful is a mathematical question.

Rewriting existing work in a common notation is a weak test. A sufficiently expressive language can redescribe many things without revealing anything useful about them. A more demanding test starts with two domains represented as $K(A)$ and $K(B)$. Suppose a structural law is found in $K(A)$. Can it be transported to $K(B)$ and yield a theorem, algorithm, representation, or computational advantage that was not put into the translation by construction?

That is the kind of result that would give a common calculus substance. It would show that the interface carries information useful beyond the example that motivated it.

6.2 Five properties to look for

A machine-checkable proof, a computer algebra expression, a numerical program, and generated mathematical prose each show a different use of a machine. None alone demonstrates that the representational architecture has changed in the sense proposed here.

I would look for at least five properties in a machine-native calculus:

1. **Explicit operations.** Decomposition, transport, abstraction, refinement, and reconstruction have defined mathematical types and contracts.
2. **Structural certification.** A transformation records its domain, preserved invariants, discarded distinctions, residuals, and reconstruction conditions.
3. **Productive transport.** A structural law moves between representations and produces consequences that were not built into the translation.
4. **Composition.** Operations combine into larger procedures while retaining their correctness obligations and provenance.
5. **Language growth.** Useful recurring constructions can become primitives available to later reasoning.

The fifth property should not weaken the first four. A system that can enlarge its lan-

guage also needs a reliable way to check what the additions mean and when they may be used.

6.3 A controlled empirical comparison

Let M be a reasoning system with a fixed representation language L, and let M_K be an otherwise comparable system equipped with a calculus that can construct new intermediate representations and abstractions. Give both systems previously unseen problems from several domains under matched resource budgets.

Final accuracy is one outcome, but it is not the whole comparison. Record proof success, computational cost, search depth, transfer to other domains, reuse of earlier discoveries, and the cost of constructing and checking new primitives. Measure later performance on held-out problems, so that reuse is distinguished from fitting the problems that produced the primitive.

If M_K discovers a certified law and incorporates it into its operative library, an improvement on later problems could support the claim that changing the language helped. Appropriate controls would still be needed to separate the contribution of the law from extra computation, a stronger search policy, or access to information the fixed-language system did not receive.

There is a corresponding failure condition. If the system does not accumulate useful abstractions, or if they do not improve later reasoning relative to the matched comparison, the strongest interpretation proposed in this essay has not been demonstrated. A larger corpus or a more elaborate vocabulary would not repair that absence of evidence.

6.4 A first completion criterion

A broad research program needs a point at which its central claim can be judged. For the version of Komorebi described here, a reasonable first criterion would require the following pieces to work together:

1. A formally verified core for task-relative abstraction, canonical quotients, refinement, and action-relative behavioral equivalence.
2. A verified bridge from quotient abstractions to the project's projector and residual machinery.
3. Several mathematically independent realizations of the same abstraction theorem.
4. A nontrivial transport of a law between domains that produces a result not inserted into the translation.
5. A machine-readable intermediate representation for the relevant objects, invariants, residuals, reconstruction maps, and provenance.
6. A representation-search system that uses that intermediate representation on previously unseen problems.

7. A controlled comparison in which adaptive language growth improves on an otherwise comparable fixed-language system under matched resources.
8. A self-generated primitive that is formally certified, admitted to the operative language, and reused successfully in later held-out reasoning.

Together these would describe an implemented research object with a tested contribution. Until then, the broader proposal remains a research program. The individual mathematical results and software artifacts should continue to be assessed at their own recorded scopes.

6.5 Questions for the next stage

Several questions concern the calculus itself. Which abstraction laws survive an exact change of representation? Which residuals have useful composition laws? What geometries arise on abstraction spaces? When can a law be transported between domains without losing the hypotheses that give it meaning?

Other questions concern a working reasoner. Can it discover useful primitives, and how should it decide which ones to retain? How stable are its abstractions when tasks drift? How can exact and approximate abstraction coexist? Can the system inspect and improve some of its own abstraction procedures while keeping their formal levels and certification obligations clear?

The cost of changing a language belongs in those questions. A representation that simplifies one calculation may be too expensive to discover or maintain. A successful calculus would need to explain when that trade is worthwhile and when familiar mathematics already provides the better description.

6.6 Closing

A person may need an explanation, a proof assistant a formal derivation, and a computational agent an executable construction. Relating these forms is already part of mathematical practice.

A machine reasoner may also help choose the representations, distinctions, and primitive operations used in later reasoning. Some useful descriptions may be unfamiliar to people. Their value would have to be shown through invariants, composition laws, consequences, and recoverable connections to existing work.

The question is how a reasoner can help construct a useful mathematical language while remaining answerable for what it preserves or loses. Komorebi studies this through distinguishability under transformation: which differences matter, which can be forgotten, and which appear after an operation or a change of description.

That is a concrete direction for research. Its value will depend on the mathematics and the working systems it produces.

References

[1] Komorebi P. *Universal Residual Seeds: Deformation forests, transport categories, and conditional system completion.* Paper X. Reading edition, source files, and evidence.

[2] Komorebi P. *Tame and Wild Packet Geometry for Local Langlands.* Paper XI. Reading edition, source files, and evidence.

[3] Komorebi P. *Patching Support and Hecke Packets: Conditional automorphic completion.* Paper XII. Reading edition, source files, and evidence.

[4] Greg Kamradt. OpenAI's GPT-6 Astra on ARC-AGI-3. ARC Prize, 3 September 2026. Consulted 29 October 2026.

[5] Ilan Bigio and Ted Sanders. How enabling two settings tripled our scores on the ARC-AGI-3 benchmark. OpenAI, 29 July 2026. Consulted 29 October 2026.

[6] AlphaEvolve team. AlphaEvolve: How our Gemini-powered coding agent is scaling impact across fields. Google DeepMind, 7 May 2026. Consulted 29 October 2026.

[7] Komorebi P. *Machine-Native Reasoning: Reviewed Mathematical Core.* Reading edition, source files, and evidence.

[8] Komorebi P. *Proof Path Theory: Certificates, compression, discovery fields, and mathematical memory.* Reading edition, source files, and evidence.

[9] Komorebi P. *Realization Boundaries: Finite Grammar, Infinite Groups, and the Limits of Reconstruction.* Reading edition, source files, and evidence.

[10] Komorebi P. *Stratified Relational Workspace Theory: A Core-Transfer Framework for Distributed Cognition, Collaboration, and Epistemic Systems.* Reading edition, source files, and evidence.