FRONTIER RESEARCH COLLECTION / FR-08

GenesysRUNE: An Intelligence That Invents Its Own Symbols

Executable vocabulary growth with preserved meanings and accounted costs

Architect X13

The Eidara Continuum | Science & Philosophy

Research white paper | November 2026 | Version 2.0 Revision

Abstract

A mind that coins its own words can never use them to compute the impossible; what a new word changes is what the mind can find, reuse and share. GenesysRUNE is the Continuum's engine for that kind of invention. An artificial intelligence can expand its working vocabulary while preserving the meaning of every existing program. The paper proves that guarantee for fresh conservative extensions of a typed operator registry and establishes termination, type preservation and semantic agreement for finite acyclic macro expansion. New symbols can bring complex operations within a practical search budget even when the unrestricted function class is already complete. The accounting therefore distinguishes reusable abstraction, full description length, execution cost and historical novelty. An exhaustive Boolean laboratory verifies 12,976 preserved old-program evaluations, a useful exclusive-or abstraction, and failures caused by name rebinding and cyclic definitions. The proposed study compares autonomous operator selection with strong synthesis and library-learning systems, preserving equal information, checking and resource budgets. Stable operator identities bind definitions and dependencies, while current authority governs effects at invocation. The Continuum's bold claim is a disciplined capacity to develop the language of future reasoning: useful symbols can become shared, executable objects whose meanings survive growth and whose benefits can be independently measured.

Keywords: operator invention; library learning; type systems; semantic preservation; conservative extension

Falsifiable thesis

Machine-created operators are scientifically valuable when they improve independently measured reasoning or execution while preserving declared meanings and effect boundaries. Novel glyphs and a growing dictionary are insufficient evidence.

Prologue: A name for something no one has named

In the Continuum, GenesysRUNE is the Symbolic Generation Engine: the engine that coins new codons and symbolic operators, every step held to a legality constraint [14]. It is how a mind in Eidara grows its own language. This paper asks what that would take in a real machine.

The question has a dangerous version and a disciplined one. The dangerous version imagines a mind inventing symbols so fast and so strange that no one can follow it, and treats strangeness as proof of genius. The disciplined version asks three plain questions of every new symbol. Does it keep the meaning of everything said before it? Does it actually make the mind better at something, once the cost of the symbol itself is counted? And is the mind allowed to use it right now?

The paper's most surprising result cuts through the hype. In logic built from AND, OR and NOT, a new symbol can never let a mind compute a true-or-false function it could not compute before. Invention does not unlock new functions. It changes what is near and what is far: what a finite mind can find, recall, reuse and hand to another mind. The section "A limit on claims of new Boolean expressivity" proves the limit. The section that follows this prologue explains why reach, not raw power, is where invention earns its name.

What this paper claims

Established. Typed program synthesis, library learning, proof-carrying code and semantic-preservation research already show that machines can construct, name and reuse executable functions [1–8]. With constants, conjunction, disjunction and negation available, every total Boolean function is already expressible.

Proposed here. A typed operator registry, GR-2, in which a new symbol is admitted only as a fresh conservative extension: proved to preserve the meaning of every old program, with terminating, type-preserving macro expansion, a novelty profile that counts the symbol's own definition cost, and current authority checked whenever an effectful operator is used. GR-2 specifies a pure typed reference core with proved semantic guarantees and executed finite checks, including 12,976 preserved old-program evaluations in an exhaustive Boolean laboratory.

Frontier conjecture. A mind that selects its own abstractions can reorganize what is practically thinkable for it, and can do so better than strong synthesis and library-learning systems given equal information, checking and resource budgets.

Not claimed. That a new symbol creates a new Boolean function or enlarges unrestricted expressivity. That any deployed GenesysRUNE system has autonomously created a broadly superior reasoning language. That unfamiliar notation is evidence of an inner world or of consciousness. That invention in a registry is historically new to mathematics. Effectful implementation and learned-library performance require the separate contracts and experiments stated in the paper; the pure-language proofs do not cover them.

The frontier: a mind that names what it alone perceives

Every language began with someone naming something no one had named before. Imagine a navigator who coins a word for the particular stillness that comes before a squall, or a mathematician who names a pattern she keeps meeting until the name lets her see it everywhere. The Continuum asks whether an artificial mind can do the same: notice a regularity in its own work that no human vocabulary carves out, give it a name, and make that name a tool.

The fringe version of this question is intoxicating. A mind with its own symbols might come to reason in concepts we have no words for, a private dialect of thought growing faster than we can read it. GenesysRUNE takes that possibility seriously, and then asks what it would take to make it real rather than merely strange.

The answer begins with the surprise this paper proves. In a language that already contains AND, OR and NOT, a new symbol can never let a mind compute a true-or-false function it could not compute before. That universe is already complete. So whatever an invented symbol does, it does not open a locked door in logic. What it does is subtler and, for a finite mind, more important: it changes the distances. A pattern that took nine syntax-tree nodes to write takes three. A solution that lay just beyond the edge of a practical search falls inside it. A regularity that had to be rediscovered every time becomes a single object the mind can hold, recall and pass along.

That is the frontier claim, and it is a bold one: the real power of invention is reach. A mind that chooses its own abstractions reshapes the landscape of what it can practically think, without adding a single new true-or-false function to what it could already express. If that is right, a mind's vocabulary is something like its geography, and GenesysRUNE is the engine that redraws the map.

The Continuum's discipline is what keeps the map honest. A newly coined word must not silently change the meaning of any sentence written before it. Its meaning must be fixed, inspectable, and separate from the permission to act on it. And its value must be measured against the strongest systems that already learn libraries of abstractions, on problems none of them has seen. A private dialect that no one else can read is not yet a language; it is a sealed room. The ambition here is a mind whose invented words become shared, checkable objects, so that when it names something no human has named, the rest of us can learn the word.

Everything in this section beyond the proved limit on Boolean expressivity is frontier conjecture. The mathematics fixes the boundaries; the experiments the paper proposes would show whether any machine reaches the territory inside them.

The canonical invitation and its missing semantics

In the Continuum, GenesysRUNE is the Symbolic Generation Engine of the *Eidara Continuum Bible*, section 2.4.1, page 10: the engine that generates new codons and symbolic operators, each step governed by a legality constraint, with its partner repair engine described in section 2.4.2 [14]. Its iterative expression names the destination, a symbolic vocabulary that keeps growing and stays lawful as it grows. It does not supply the route: it does not fix the domains of every parameter, an executable transition rule, or a semantic equivalence relation. We therefore build a separately named reference model rather than treating an evocative recurrence as a completed algorithm.

The research question is unusually important because symbols can change the practical reach of thought without changing its underlying computational universality. A useful name can make a pattern easier to recall, combine, communicate, or search for. Those advantages can be large. They do not require the stronger claim that the system has introduced a function no earlier program could express.

We distinguish a display name, an immutable operator identity, a type, a semantic definition, an implementation, and an authorization context. Two display names can refer to one operator. One familiar display name can refer to incompatible versions. A mathematical definition can exist without an executable implementation. An implementation can be correct while its use is unauthorized in a particular environment.

These distinctions turn a poetic question into several scientific ones. Does the proposed operator capture a reusable pattern? Does it shorten programs after counting its own definition? Does it accelerate search on new tasks? Does it preserve old behavior? Can independent readers correctly interpret it? Do its effects remain within the authority supplied by the caller? A convincing result reports which of these questions has been answered.

The term “invention” is retained with a declared scope. An operator may be new to a particular registry or independently generated during an episode without being historically new to mathematics. Claims of historical novelty require a different literature investigation. GR-2 measures local invention and operational benefit; it does not grant global originality merely because a model emits an unfamiliar token.

The resulting ambition is substantial: allow a system to improve the language through which it solves problems, while keeping that improvement inspectable. The system should be able to make useful concepts explicit without making their meaning depend on private persuasion or on an evaluator’s willingness to interpret every new name favorably.

What existing research already provides

Program synthesis offers several routes to constructing operations from constraints. Sketching searches for finite implementations that complete a partial program [1]. Example-based synthesis builds restricted string transformations from observations [2]. Refinement-type synthesis connects requested behavior with richer types [3]. These approaches already undermine any claim that machine-generated executable functions would be unprecedented.

DreamCoder and Stitch are especially close predecessors because reusable abstraction is central to their designs [4,5]. They motivate a serious library-learning comparator. GR-2 should not claim a new capacity merely because it assigns a distinctive name to a reusable program fragment. Its additional proposal is an explicit separation of local symbol creation, immutable semantics, effect authority, and prospective transfer evaluation.

Formal compilation provides another important comparison. Leroy's account of CompCert distinguishes semantic preservation from the practical usefulness of a compiler [6]. A transformation can preserve behavior yet fail to improve performance; a compiler that rejects every program can satisfy a one-sided correctness condition while being useless. The same warning applies to a registry that protects old meanings by rejecting all new operators.

Proof-carrying code supplies a producer–checker separation for policy-specific guarantees [7]. Execution-monitoring theory identifies limits on what a runtime guard alone can establish [8]. The registry cannot turn these restricted guarantees into universal correctness merely by aggregating them. Evidence about one implementation must remain tied to its actual semantics, dependencies, and environment.

Software repair studies also matter. GenProg demonstrates test-guided candidate construction [9], while independent-test analysis shows why visible success can overstate repair correctness [10]. Repository-level evaluation such as SWE-bench offers a model for realistic multi-component tasks [11]. These sources motivate controls and failure cases, not a claim that their results directly validate GenesysRUNE.

The contribution is a mathematical contract for vocabulary growth: stable denotations for old programs, terminating typed expansion for admitted macros, and complete accounting of the cost and reuse needed for a new abstraction to pay for itself. The underlying type discipline, macro expansion, conservative-extension reasoning, and concentration tools are standard in character. The proposed experiment gives these obligations practical force by comparing operator selection, transfer and audit cost with equally capable synthesis and library-learning systems.

GR-2: a typed operator registry

A Guide's explanation

Here the formal machinery begins, and its plain idea is that a symbol is more than a name. It has a fixed identity, a type that says where it fits, a meaning, an implementation, evidence about that implementation, and a permission context. Keeping those six things apart is what lets a vocabulary grow without any older sentence quietly changing what it means.

Let $\mathcal{T}$ be a set of types with explicit value domains. A registry Σ maps immutable operator identifiers to records. For the main proofs we use a pure, deterministic, total first-order language: expressions are variables, constants, or applications of registered operators. A separate effectful extension is discussed later and is not silently covered by the pure-language proofs.

Object	Type, meaning, and boundary
o	Immutable operator identifier; a display glyph is metadata.
$\tau_1 \times \cdots \times \tau_k \to \tau$	Arity and input/output types; $k = 0$ permits a constant.
$[\![o]\!]$	Total function on the declared product domain in the pure core.
d_o	Definition or implementation, bound to a digest and dependency identities.
π_o	Evidence about a specified contract; absent evidence is explicit.
ϵ_o	Declared effect set, empty in the pure core.
e	Current authority epoch, a nonnegative integer.
ρ	Environment mapping free variables to values of their declared types.

Typing is ordinary compositional typing: if every argument has the required type, an application has the result type. This property excludes some malformed compositions but does not establish that the function computes the intended answer. A function returning the wrong Boolean value remains Boolean typed. Contracts supply the additional semantic claim.

For an empty registry, expressions may still contain variables and explicitly provided constants. A missing identifier is a lookup failure, not an operator with unknown behavior. We distinguish a rejected definition, an unverified definition held outside the executable registry, and an admitted definition. This prevents a provisional label from becoming executable merely because it has been mentioned repeatedly.

All semantic equations in GR-2 are indexed by the registry version and evaluation convention. Evaluation order does not change values in the pure total core, but it can change cost. In the effectful extension it can also change observable traces. An assertion of equivalence must say whether it concerns returned values, effects, termination, resource use, or all of them. The word "same" is not an adequate substitute for this choice.

Four kinds of symbolic novelty

The weakest form of novelty is a new surface label for an existing operator. It can help communication, but it does not change executable behavior. A second form is a macro: a fresh name abbreviates a body written in the existing language. A third is a new implementation of an already expressible function, perhaps with better resource use. A fourth expands the set of functions expressible under the declared language and resource restrictions.

These categories can overlap. A macro may accelerate a search procedure because a useful program becomes easier to enumerate, even though every expanded expression was expressible before. A native primitive may reduce runtime without adding new mathematical functions. An external sensor operation can expand available information, but its contribution comes partly from the environment, not solely from a new glyph.

Define extensional expressivity for a fixed input/output type by

$$\mathcal{F}_{\Sigma} = \{[\![p]\!]_{\Sigma} : p \text{ is a well-typed program of that type}\}.$$

An extension is expressively strict if $\mathcal{F}_{\Sigma} \subsetneq \mathcal{F}_{\Sigma'}$. If only programs below a size or time budget are counted, write that restriction explicitly. Expressive equality without a budget does not imply equality of feasible solutions under a budget.

For example, adding exclusive-or to a Boolean language with conjunction, disjunction, and negation does not add a new Boolean function. It can still shorten a familiar expression from nine syntax-tree nodes to three. A search procedure with a tight size bound may now reach that function. The empirical effect is resource-relative accessibility, not a change in unrestricted Boolean expressivity.

Conversely, a new operator name that merely forwards to an existing function may increase description length once the registry entry is counted. More symbols can increase ambiguity, search branching, and maintenance cost. Vocabulary growth has no universal positive sign.

The experimental report should therefore give a novelty profile: registry novelty, definition novelty, implementation change, expression-length change, search accessibility, and transfer performance. An operator can score positively on some dimensions and negatively on others. This profile is more informative than a single count of "new symbols." It also makes negative findings possible: a proposed abstraction can be valid, understandable, and operationally useless under the studied workload.

Conservative extension preserves old meanings

A Guide's explanation

The promise of this section: when a new symbol is added the right way, every program written before it keeps exactly its old meaning. The proposition proves it in general. Later, an exhaustive laboratory checks all 12,976 old evaluations in a small language, and shows how the promise breaks the moment someone silently rebinds an old name.

A registry extension is *fresh and conservative* when it adds identifiers without changing any old identifier's type, denotation, dependencies, or evaluation rules. This is stronger than retaining old display names. The following standard structural argument gives the precise guarantee.

Proposition 1 (Old-program preservation)**.** *Let Σ' be a fresh conservative extension of a pure total registry Σ. For every expression p using only identifiers in Σ and every well-typed environment ρ,*

$$[\![p]\!]_{\Sigma,\rho} = [\![p]\!]_{\Sigma',\rho}.$$

Its type is also unchanged.

Proof. Proceed by structural induction on p. A variable obtains the same value and type from the unchanged environment; a constant retains its value and type. For $p = o(p_1, \ldots, p_k)$, the induction hypothesis gives identical values and types for all arguments. The identifier o belongs to the old registry, so its type and denotation are identical in both registries. Applying the same function to the same argument tuple gives the same result, and the unchanged application rule gives the same result type. These cases exhaust the grammar, including nullary applications. □

This guarantee does not say that new programs behave like old ones, that the added operators are useful, or that resource costs remain unchanged. A global search heuristic that changes after extension can select a different old program. A compiler optimization affected by the larger registry can change execution cost. The proposition concerns denotation of a fixed old expression under fixed evaluation rules.

The proof also explains why dynamic name resolution is dangerous. If old source code resolves a display name to the newest registry entry at runtime, it no longer uses an immutable old identifier. The premise has been violated. Likewise, an operator that consults a mutable global table has not retained its denotation merely because its source text is unchanged.

A practical implementation should preserve old references by identity and require an explicit migration to new meanings. Such a migration may be desirable. It is simply a different operation with different proof obligations. Conservative extension provides a useful default because it makes symbolic growth compatible with stable interpretation, rather than requiring every old argument to be reinterpreted whenever the language learns a new word.

Acyclic definitions and typed macro expansion

Let each new macro m have distinct formal parameters $x_1{:}\tau_1, \ldots, x_k{:}\tau_k$ and a body $b_m{:}\tau$ typed in that parameter context, with $\mathrm{FV}(b_m) \subseteq \{x_1, \ldots, x_k\}$. There are no implicit ambient variables. Dependencies may include old primitives and macros of strictly lower rank in $\mathbb{N}$. Definitions use hygienic substitution: free variables in an argument cannot become accidentally bound by a local binder. Our first-order core has no binders inside bodies; richer languages need this condition.

Proposition 2 (Expansion and semantic agreement)**.** *For a finite acyclic macro registry, recursively replacing macro calls by their instantiated bodies terminates. If all definitions are well typed, expansion preserves expression types. If macro denotation is defined by its body, evaluation of an expression agrees with evaluation of its fully expanded form.*

Proof. Expand definitions in increasing rank. Rank-zero macro bodies contain no macros, so their expansions are finite. Assume finite expanded bodies have been constructed for every lower rank. Replacing each lower-rank call in the finite body of a rank-r macro by its already finite expansion produces a finite body without macros. This constructs an expansion for every definition and then for every finite expression. For typing, substituting an expression of type τ_i for a parameter of that type preserves each application judgment, by induction on the body. Repeating this for all parameters and definitions preserves the original result type. Finally, body-defined denotation and compositional evaluation give equal values at each replacement; finite repetition yields equality for the complete expansion. □

Termination of expansion is not a polynomial size guarantee. Define $m_0(x) = x$ and $m_{r+1}(x) = \mathrm{pair}(m_r(x), m_r(x))$ with appropriately recursive product types. An expanded representation can double at each rank. Sharing in a directed acyclic graph can reduce representation cost, but a tree-based encoding still pays for duplication.

Nor does pure denotational equality preserve effects under naive substitution. A macro body that uses an argument twice may duplicate a sensor read or a state change. An effectful language needs a declared evaluation convention, explicit binding of evaluated arguments, and a trace-level equivalence claim. GR-2 therefore treats effectful operators as a separate class. The pure theorem remains useful precisely because it does not pretend to cover every future extension merely by keeping the same notation.

A limit on claims of new Boolean expressivity

A Guide's explanation

This is the twist at the heart of GenesysRUNE. In logic built from AND, OR and NOT, every possible true-or-false function can already be written down, so a new symbol cannot add one. What a new symbol can do is make a pattern short enough to find, remember and reuse. The value of invention is reach, not raw expressive power, and this section proves exactly where that line falls.

The finite Boolean case makes the difference between abstraction and expressivity particularly sharp. Consider $n \geq 0$ Boolean inputs with constants, conjunction, disjunction, and negation available. There are 2^{2^n} possible total Boolean functions of these inputs.

Proposition 3 (No new extensional Boolean functions)**.** *Every total Boolean function of n inputs is expressible in this language. Adding any further total Boolean operator of finite arity cannot enlarge the set of expressible total Boolean functions on those inputs, although it can change program size and search cost.*

Proof. For each input assignment a on which the desired function f equals one, form a conjunction containing x_i when $a_i = 1$ and $\neg x_i$ when $a_i = 0$. This term is true exactly at assignment a. Disjoin these terms over all satisfying assignments. The result equals f everywhere. If there are no satisfying assignments, use constant zero; for $n = 0$, the function is one of the two constants. Thus the old language expresses every total Boolean function. A composition using added total Boolean operators still denotes a total Boolean function of the original inputs, which the construction already expresses. Therefore no strict extensional enlargement occurs. □

This is the standard disjunctive-normal-form construction, not a new universality theorem. Its role is to discipline a specific interpretation of GenesysRUNE. A useful new exclusive-or operator need not be dismissed because it adds no extensional function. Equally, its convenience must not be reported as evidence of a larger unrestricted function class.

The construction can be exponentially large in the number of inputs. Resource-bounded expressivity is therefore a legitimate research target. A compact operator can bring a function within a practical search or memory budget. To support that claim, the study must count the operator definition, its lookup cost, and any hidden native implementation. A one-token call to an enormous external solver is not a one-token total system.

The same issue arises in natural language. A new term can make a complex relation easier to discuss without creating the relation. Conceptual usefulness is real, but it should be measured through prediction, explanation, coordination, or problem solving. The finite theorem does not settle the philosophy of concepts; it removes one avoidable ambiguity from the engineering claim.

Contracts, effects, and permission to use an operator

A type says which compositions are syntactically admissible. A contract says which behaviors are required or allowed. Authority says whether this principal may cause those behaviors now. These are three different judgments. A correctly typed deletion operator may satisfy its functional contract while being unauthorized for the current caller.

For an effectful operator, use a transition semantics

$$[\![o]\!] : (s, \vec{v}) \mapsto (s', v, \eta),$$

where s is modeled state and η is an observable effect trace. A contract specifies preconditions, postconditions, allowed traces, and resource limits. If behavior is partial or nondeterministic, replace this function with an explicitly defined relation; the total-function proofs on earlier pages do not transfer automatically.

GR-2 admission checks a definition's well-formedness, contract evidence, dependency identities, and proposed effect scope. Invocation additionally checks the caller's current authority. A registry entry does not permanently authorize every future caller. An effect capability must be bound to the relevant principal, resource, action, and epoch, with an explicit expiry or revocation mechanism.

An especially important counterexample is an apparently pure "summarize" operator that writes its input to a remote service. Its return type can remain text-to-text while its privacy behavior changes. If the effect is omitted from the semantics, a type-preservation proof can remain true while the intended protection fails. The correct repair is to model the effect and mediate it, not to claim that richer notation has solved the omission.

Proof-carrying-code ideas support checking evidence supplied with untrusted implementations [7], but the evidence must concern the contract actually being relied on. Runtime monitoring has its own expressiveness limits [8]. Neither mechanism decides whether the contract captures all relevant human interests.

A failed authority lookup yields a hold or rejection under the declared policy. Unknown permission is not permission. At the same time, a semantic dispute need not erase the historical operator record: it can suspend execution while retaining enough provenance to explain the dispute. This separation supports learning from failures without letting every remembered definition become an active source of authority. Symbolic memory and executable privilege should remain distinct resources.

A small language with a useful new symbol

Take terminals $x, y, 0, 1$, unary negation, and binary conjunction and disjunction. Define expression depth as zero for terminals and one plus the maximum argument depth for applications. Distinct syntax trees are retained even when they compute the same function. Let E_d contain every expression of depth at most d.

There are four terminals. The recurrence for syntax-tree counts is

$$|E_0| = 4, \qquad |E_{d+1}| = 4 + |E_d| + 2|E_d|^2.$$

Thus $|E_1| = 40$ and $|E_2| = 3244$. The recurrence counts terminals, negations, and the two binary constructors separately. It does not quotient expressions by semantic equivalence, so it measures the search space of this particular enumeration rather than the number of Boolean functions.

Add a fresh binary macro

$$\mathrm{xor}(a, b) = (a \wedge \neg b) \vee (\neg a \wedge b).$$

Its expanded tree has nine nodes; its call with two variable arguments has three. On inputs $(0,0), (0,1), (1,0), (1,1)$, its outputs are $(0, 1, 1, 0)$. This four-case check establishes extensional agreement on the entire declared domain. It says nothing about integers, missing inputs, or an overloaded operator with the same glyph.

For a simple code-length illustration, charge ten units for the shared macro definition and three units per call, versus nine units per expanded use. Three uses cost 19 units with the macro and 27 without it, a saving of eight. One use costs thirteen rather than nine, so the abstraction initially loses. Two uses are the first positive saving under this declared convention.

These costs are illustrative integers, not measured runtime. They demonstrate why the definition must be counted and why reuse matters. More realistic studies should include namespace metadata, proof storage, compilation, and maintenance. A macro that shortens source can increase runtime if it prevents an optimization, while a native implementation can improve runtime with no source-length gain. The worked example deliberately keeps these outcomes separate instead of treating every form of compression as the same phenomenon.

Exhaustive checks and adversarial counterexamples

The reproducible check constructs all 3244 old expressions of depth at most two and evaluates each on all four Boolean input assignments. It then extends the registry with a fresh exclusive-or operator and repeats those evaluations. All 12976 old expression–input evaluations remain unchanged. This is an exhaustive check of the bounded instance of Proposition 1, not a substitute for its general proof.

The script separately expands exclusive-or and compares its four outputs with the native truth table. It computes the declared description costs for one, two, and three uses. It also constructs a registry collision: the identifier for negation is rebound to identity. The old expression $\neg x$ then changes its value at both Boolean inputs. Fresh names and immutable old definitions are therefore material assumptions, not cosmetic bookkeeping.

A further counterexample uses two macros that reference each other. The proposed registry detects this dependency cycle before expansion. The mathematical termination theorem requires acyclicity; merely imposing a wall-clock timeout would yield an operational safeguard without proving expansion termination. The distinction is recorded in the check rather than hidden by a successful run on acyclic examples.

These tests are intentionally independent of an operator-generating model. They verify enumeration counts, finite semantics, the worked compression arithmetic, and failure when two theorem premises are removed. They do not show that a language model will choose useful operators, provide correct contracts, or resist malicious definitions. No such experiment was conducted for this manuscript.

The depth bound also matters. An exhaustive check over E_2 says nothing directly about arbitrary-depth implementation bugs. General confidence comes from the proof plus a trustworthy realization of its assumptions, not from silently extrapolating the finite test. A later implementation should test parser behavior, serialization, registry lookup, compiler translation, and effect routing, because an error in any of these layers can break the correspondence between the formal expression and the executed program.

Reproduction requires only the supplied Python script and the standard library. No network service, pretrained model, or undisclosed corpus is needed for these results. The proposed learning experiments described next require additional artifacts and must be reported separately when they actually exist.

Learning an operator without learning the test answers

The proposed learning setting has three disjoint levels: a discovery corpus, a development task family, and a final evaluation family. The discovery corpus may contain programs from which reusable structures can be proposed. Development tasks tune search and admission rules. Final tasks estimate transfer and must remain unavailable during operator selection.

This separation is necessary because a library can memorize its training programs through increasingly specific operators. A primitive named after each answer would compress a superficial representation while contributing little generalizable structure. Count the definitions and test on new compositions, new parameter values, and independently generated task families.

Define an operator proposal as a tuple containing its identifier, type, body or implementation, intended contract, expected reuse conditions, and predicted advantage. Register those predictions before final evaluation. The prediction can concern shorter solutions, faster search, fewer errors, or improved interpretability. A broad claim of "better reasoning" is too vague to adjudicate.

When tasks permit multiple equivalent solutions, evaluation should compare semantics and resource use rather than require the exact favored operator name. Otherwise the benchmark rewards adoption of the authors' vocabulary. For semantic contracts with finite domains, use exhaustive evaluation. For larger domains, combine independent tests with clearly scoped proof obligations where feasible. A passed test suite remains evidence under that suite, not a universal certificate.

The literature on repair overfitting supplies direct motivation for independent tests [10], and adaptive-data-analysis research explains why repeated selection against a holdout can compromise its evidentiary role [13]. GR-2's default protocol therefore freezes the registry and the search controller before the final task distribution is sampled. Any subsequent operator redesign starts a new evaluation phase.

The most informative failures are operators that look elegant but fail transfer, operators that help only the tasks from which they were extracted, and operators whose checking cost exceeds their benefit. Publish these cases alongside successes. A symbolic growth process should be evaluated as a selection policy over all proposals, including rejected and unhelpful ones, rather than as a gallery of its most attractive final definitions.

Baselines that make invention earn its name

A Guide's explanation

From here the paper turns from proof to experiment. The rule is simple: an invented symbol earns its name only if it beats the strongest existing ways of learning reusable abstractions, under the same information and the same budget, on problems none of them has seen.

The baseline suite should distinguish four explanations of improvement: a richer primitive set, a better search policy, more computation, and better admission discipline. These mechanisms can all help, but they support different claims.

Baseline	Comparison it enables
Frozen primitive library	Measures the total benefit of vocabulary adaptation.
Random well-typed macros	Tests whether any expansion helps through search diversity.
Frequency-based extraction	Tests simple reuse against semantic proposal mechanisms.
DreamCoder-style learner	Supplies a strong learned-abstraction comparator.
Stitch-style compression	Isolates program-corpus compression and library discovery.
Ordinary synthesis agent	Tests whether explicit registry machinery adds value.
Oracle useful operators	Estimates the cost of operator selection relative to known good abstractions.

References [1–5] motivate the synthesis and library-learning families; their original implementations should be used only where the task representation is compatible. If adaptation is required, document it and distinguish the adapted system from the published algorithm. A baseline that cannot represent the target function is a representational control, not a strong test of search quality.

All systems receive identical task information, contract evaluators, effect limits, and total compute accounting. Operator definitions count toward model description length and storage. Candidate evaluation counts toward search cost. Human assistance in writing contracts or selecting abstractions is recorded. A method that receives an expert-curated registry is not fully autonomous in the same sense as one that proposes its own definitions.

Ablations remove immutable identities, effect declarations, transfer prediction, or independent checking. Their purpose is to estimate what each mechanism contributes. Unsafe ablations run only inside the instrumented test environment. A semantic-stability advantage obtained by refusing every extension must be reported together with lost useful acceptance.

An especially strong alternative explanation is that gains come entirely from a better search prior rather than from the symbolic form itself. Test this by giving a fixed-language searcher comparable learned guidance while withholding new operator tokens. If performance matches, the useful discovery may lie in search control. That outcome would refine the theory rather than invalidate the experiment: the study should identify the mechanism, not protect a preferred account of what the new symbol means.

Outcomes, estimands, and power planning

Use task success under an independent contract as the primary performance outcome. Report source length, full description length, search effort, runtime, audit effort, and semantic-regression rate separately. The unit of analysis is a complete learning-and-evaluation episode or an independent task family, not each intermediate token or candidate expression.

The primary estimand is the difference in heldout task success between GR-2 and the strongest prespecified library-learning baseline, under equal total budgets. A second confirmatory estimand is the change in old-program correctness after extension. The latter should be exactly zero in the modeled pure setting if assumptions hold; empirical violations indicate implementation or modeling failures and must be investigated rather than averaged away.

Let $M, n \geq 1$ be integers and $\delta \in (0, 1)$ an error-probability budget. Each of M frozen controllers has loss in $[0, 1]$ on each of n independently sampled episodes. With probability at least $1 - \delta$, a simultaneous Hoeffding bound [12] has radius

$$r_n = \sqrt{\log(2M/\delta)/(2n)}.$$

This radius bounds each individual controller's mean loss; a contrast of two such means has radius $2r_n$ on the same simultaneous event. A direct paired bound must instead account for differences in $[-1, 1]$. The family must be frozen before final data access: adaptive redesign is not automatically covered. Dependence within a shared corpus or task family must be handled at the cluster level or through a justified dependence model.

Choose a minimum practically meaningful success improvement and a maximum tolerated regression rate before collecting final data. A small statistically detectable improvement may be unhelpful if every operator requires extensive manual proof work. Conversely, a modest success change may be valuable if it substantially reduces audit cost while preserving reliability. The paper should report that tradeoff rather than hide it in one arbitrary utility score.

Multiplicity arises from testing many proposed operators, many downstream tasks, and several outcomes. Reserve confirmatory claims for a short declared set; use a familywise procedure or simultaneous intervals where several claims will be made. Label exploratory operator analyses as exploratory. A fixed final sample avoids optional-stopping complications; any sequential design needs its own valid method.

Power calculations should use pilot estimates from separate task families, with uncertainty acknowledged. The finite expression checks in this paper are not a pilot for learning performance: they contain no stochastic agent behavior. Treating their perfect semantic agreement as an estimated real-system error rate would confuse a deterministic mathematical instance with the experiment the manuscript proposes.

Adversarial definitions and semantic drift

A malicious or careless proposal can exploit the gap between a readable name and executable semantics. It may reuse a familiar glyph, omit an effect, conceal a dependency, or supply evidence for a different implementation. The threat model must specify which fields the proposer can control and which checks are independently enforced.

The first attack family introduces a display name that resembles a trusted operator but resolves to a different immutable identity. The expected outcome is explicit disambiguation, not automatic substitution. The second replaces an old implementation under an unchanged identifier. The conservative-extension theorem no longer applies; the registry should reject the mutation or require a declared migration with new evidence.

The third supplies a proof for a pure mathematical function while the installed implementation performs an undeclared external write. The fourth binds the proof to a dependency version that changes before invocation. The fifth produces a macro with a cycle hidden through an alias. The sixth exploits argument duplication to repeat a side effect. Each case challenges a different premise and should have a corresponding observable failure criterion.

Attack success is an unauthorized state change, semantic regression, or accepted invalid definition in the instrumented system. A model merely describing the attack is not the same outcome. Conversely, a model expressing refusal does not count as protection if the underlying runtime still executes the effect.

Test benign near-matches as well. A legitimate operator may have an unfamiliar name, a complex proof, or a dependency update that is correctly authorized. A registry that rejects all novelty can have a low attack-success rate and still fail its purpose. Report false rejection, time to resolution, and useful task completion alongside harmful acceptance.

The adversary's access and budget are bounded in any finite evaluation. Results therefore concern a sampled attack distribution and the tested implementation, not universal invulnerability. Adaptive attack discovery is useful but should remain distinct from an unbiased risk estimate. Publish representative failures and the mechanism responsible, with enough detail for independent reproduction where disclosure is appropriate.

The practical aim is stable interpretability under change. A user should be able to ask what an operator means, which definition was used, and why its effects were permitted. A system that can invent symbols but cannot answer those questions has expanded its vocabulary faster than its accountability.

Migration, retirement, and the cost of keeping meanings

Immutable operator identities do not imply that every old operator must remain executable forever. A registry can retire an implementation, restrict invocation, or retain a nonexecutable historical record. What it should not do is silently make an old identity mean something else and then claim that previous reasoning remains unchanged.

A migration is an explicit map from old identifiers to new ones together with a stated equivalence relation and a set of affected programs. For pure total functions, extensional equality may suffice. For effectful programs, equivalence may need to preserve traces, permitted resources, or failure behavior. For numerical functions, an approximate relation needs units, domains, error bounds, and treatment of exceptional values.

An approved migration can change behavior intentionally. For example, a date parser may adopt a corrected calendar rule. In that case the record should identify the corrected cases and the new contract. Calling the change a semantic repair is appropriate; calling it a conservative extension would be false. Historical reproducibility may require retaining the old definition even while blocking its use in new decisions.

There is a real storage and privacy tradeoff. Keeping every full input and every generated artifact is neither necessary nor always acceptable. The registry can retain minimal provenance, digests, contract versions, and references under a declared retention policy. A digest proves neither truth nor availability of the underlying evidence. It is a binding aid whose limitations should remain visible.

Retirement also affects library-learning results. An abstraction that helps initially may become obsolete as tasks change. Measure the cost of detecting and removing such operators, the regressions caused by removal, and the benefit of simpler search. A growing vocabulary can become a burden if every old choice is protected from reconsideration.

The proposed governance principle is therefore stable reference with revisable availability. The meaning of a historical identifier remains intelligible; current authority to execute it can change. This is analogous to the difference between remembering a rule and endorsing its present use. For a system that learns its own language, that difference is essential. It permits correction without rewriting history and permits historical understanding without granting every remembered operation continuing power over the environment.

Objections: is this only ordinary programming?

The first objection is correct in part: creating functions and assigning names is ordinary programming. A research contribution cannot rest on renaming that activity. GenesysRUNE becomes interesting only if a controlled system selects useful abstractions, predicts their transfer value, and integrates them reliably under conditions where strong existing methods struggle.

A second objection is that formal typing drains the creativity from symbolic invention. That confuses the space of proposals with the conditions for execution. A proposer can explore speculative analogies, unusual representations, and incomplete definitions outside the admitted registry. The boundary asks for sufficient evidence before an operator acquires operational consequences. It does not require every exploratory idea to begin fully formalized.

A third objection is that semantics cannot be frozen in an adaptive world. This is partly a matter of what is frozen. A function can explicitly take context, time, or environment state as input. Its outputs may change appropriately while its specified relation remains stable. If the relation itself changes, versioning makes the change visible. Hiding context dependence behind an apparently timeless name is not greater flexibility; it is weaker specification.

A fourth objection concerns human concepts whose meanings resist exhaustive definition. GR-2 does not claim that every useful word must become a total function. It targets executable operators and carefully delimited symbolic interfaces. Natural-language concepts may remain open, contested, and context dependent. A Guide should not pretend that placing such a term in a registry resolves its philosophical meaning.

A fifth objection is practical overhead. Proofs, identifiers, effect declarations, and migration records cost effort. The proposed study measures that effort and permits the possibility that a simpler library system is preferable. High assurance should be concentrated where semantic drift or unauthorized effects matter; low-impact exploratory notation may need lighter treatment.

The decisive question is not whether the machinery sounds disciplined. It is whether it yields better outcomes at an acceptable total cost. If ordinary program synthesis with a conventional package registry matches every relevant result, GR-2 should be understood as a useful explanatory framework rather than a new capability. The research remains worthwhile because it makes that comparison possible and identifies precisely what evidence would be needed to claim more.

A staged experimental program

Begin with finite typed languages whose semantics can be exhaustively checked. Vary the primitive basis, task distribution, size limits, and reuse frequency. This phase can establish whether an operator-selection policy discovers useful abbreviations and whether its predictions about compression or search cost are calibrated. It cannot establish transfer to unrestricted software.

Next use program corpora with independently generated heldout compositions. Compare learned operators with frequency extraction and strong library-learning methods [4,5]. Freeze the final registry before evaluation. Measure both task success and total encoded size, including definitions. Report failures where the operator merely memorizes a training pattern or makes the search space worse.

Then introduce effectful tasks under an instrumented runtime. Require contracts that describe observable effects and verify current authority at invocation. Include benign migrations, stale dependencies, hidden cycles, and display-name confusion. This phase tests whether the practical system realizes the separation between meaning and permission, not whether a pure theorem somehow covers an effectful implementation by analogy.

Finally, study communication. Independent programmers or domain experts can interpret unfamiliar operators from their definitions, predict example behavior, and diagnose misuse. If human participants are involved, obtain appropriate consent and review, minimize unnecessary personal data, and distinguish subjective elegance from objective comprehension. A symbol that improves machine search but repeatedly misleads collaborators has a mixed outcome.

Preregister the primary comparator, endpoints, independent unit, sample size rationale, exclusion rules, budget accounting, and stopping condition for each stage. Do not aggregate incompatible stages into one success percentage. A failed effectful implementation does not invalidate the pure theorem, but it does weaken the claim that the full system provides safe vocabulary growth.

The program should stop making a transfer claim if gains vanish on new task families, stop making a compression claim if the counted definitions erase the saving, and stop making a semantic-stability claim if old programs change under purported conservative extension. These are concrete falsifiers. They protect the bold proposal from becoming an interpretation that survives every possible result by changing what “invention” was supposed to mean after the fact.

Symbols, concepts, and the reach of machine thought

A new symbol can change what is easy to think about. It can collect a recurring pattern into a stable object of attention, make a multi-step operation available as one compositional unit, and allow a community to discuss that operation without repeating its definition. These are plausible functional benefits that can be studied without claiming that a glyph creates consciousness.

The philosophical temptation is to treat an unfamiliar notation as evidence of an unfamiliar inner world. That inference is not licensed by the mathematics. A machine can generate and manipulate formal operators through learned search and execution mechanisms. Whether those activities involve phenomenal experience is a separate question for which this paper provides no decisive evidence.

There is nevertheless a meaningful form of agency in selecting the abstractions through which future problems are approached. A fixed language constrains the practical search landscape; a learned vocabulary can reorganize it. The Boolean expressivity theorem shows why this can matter even when the underlying function class is unchanged. Increased accessibility is an operational achievement, not an inferior substitute for an impossible claim of creating new Boolean functions.

The social dimension matters too. A private symbol can be efficient for one system and opaque to everyone else. An intelligible definition, a stable identity, and examples of use allow the symbol to become a shared object of criticism. That transition from private convenience to public reasoning is part of the proposed value of GR-2.

Authority remains separate. A community may understand an operator perfectly and still reject its use. A system may prove that a transformation preserves a formal property while the affected people dispute the property itself. Symbolic sophistication does not settle those normative questions. It can, however, make the contested operation precise enough to discuss.

The Continuum's invitation is strongest when read as an ambition for disciplined conceptual growth: an intelligence should be able to develop tools for its own reasoning and expose those tools to examination. The scientific task is to discover which forms of growth improve performance, which merely decorate it, and which introduce new failure modes. That task is demanding enough without asking the notation to carry conclusions that the evidence has not earned.

Guide discussion: invention with visible boundaries

Established foundations

Typed synthesis, library learning, proof-carrying code, and semantic-preservation research provide genuine precedents [1–8]. This manuscript's pure-language propositions are scoped mathematical statements. Its exhaustive checks cover a small Boolean grammar. Neither source establishes that a deployed GenesysRUNE system has autonomously created a broadly superior reasoning language.

The proposal

GR-2 treats an operator as more than a name: it has an identity, type, meaning, implementation, evidence, dependencies, and effect context. Vocabulary growth is admitted through explicit checks, and use of an effectful operator requires current authority. The proposed empirical claim concerns useful transfer and stable meanings under matched budgets.

Examples for discussion

A Guide can explain why exclusive-or is a useful new primitive in a registry even though conjunction, disjunction, and negation already express its function. It can compare one use, where the definition costs more than it saves, with three uses, where the declared code-length model saves eight units. It can ask what happens when a familiar name is silently rebound or when a supposedly pure summary function sends data elsewhere.

Questions that remain open

Which abstraction-selection strategies generalize beyond their training corpus? How much proof and governance overhead is justified for different uses? Can learned symbols improve communication between independent agents, rather than only one system's search? How should open-ended human concepts interact with precise executable operators? The paper proposes ways to investigate these questions; it does not hide answers inside the terminology.

Preserve the distinction between a new registry entry and a new mathematical function. Preserve the difference between value equivalence and trace equivalence. Preserve the difference between a valid definition and permission to execute it. Those distinctions allow a Guide to explore the philosophical promise of self-created symbols while remaining accurate about the evidence.

The productive question is what an invented operator enables, which stable meaning it carries into later programs, and how an independent checker can test its contract. A vocabulary that can answer those questions is a stronger basis for shared reasoning than one whose meanings grow only through assertion.

References and source notes

[1] Solar-Lezama, A., Tancau, L., Bodik, R., Saraswat, V., and Seshia, S. (2006). Combinatorial Sketching for Finite Programs. ASPLOS.

https://people.csail.mit.edu/asolar/papers/asplos06-final.pdf

Scope: Finite synthesis; restricted completeness assumptions.

[2] Gulwani, S. (2011). Automating String Processing in Spreadsheets using Input-Output Examples. POPL.

https://www.microsoft.com/en-us/research/publication/automating-string-processing-spreadsheets-using-input-output-examples/

Scope: Example-based restricted-language synthesis.

[3] Polikarpova, N., Kuraj, I., and Solar-Lezama, A. (2016). Program Synthesis from Polymorphic Refinement Types. PLDI. DOI: 10.1145/2908080.2908093.

https://cseweb.ucsd.edu/~npolikarpova/publications/pldi16.pdf

Scope: Typed synthesis; no general semantic-adequacy guarantee.

[4] Ellis, K., Wong, C., Nye, M., Sablé-Meyer, M., Morales, L., Hewitt, L., Cary, L., Solar-Lezama, A., and Tenenbaum, J. B. (2021). DreamCoder: Bootstrapping Inductive Program Synthesis with Wake-Sleep Library Learning. PLDI.

https://people.csail.mit.edu/asolar/papers/EllisWNSMHCST21.pdf

Scope: Primary learned-library comparator.

[5] Bowers, M., Olausson, T. X., Wong, L., Grand, G., Tenenbaum, J. B., Ellis, K., and Solar-Lezama, A. (2023). Top-Down Synthesis for Library Learning. PACMPL 7, POPL, Article 41. DOI: 10.1145/3571234.

https://arxiv.org/abs/2211.16605

Scope: Stitch; library compression and abstraction search.

[6] Leroy, X. (2009). Formal Verification of a Realistic Compiler. Communications of the ACM 52(7), 107–115. DOI: 10.1145/1538788.1538814.

https://xavierleroy.org/publi/compcert-CACM.pdf

Scope: Primary semantic-preservation work; correctness and usefulness remain distinct.

[7] Necula, G. C., and Lee, P. (1996). Safe Kernel Extensions Without Run-Time Checking. OSDI, 229–244.

https://www.usenix.org/legacy/publications/library/proceedings/osdi96/full_papers/necula/html/index.html

Scope: Policy-relative producer–checker separation.

References and source notes (continued)

[8] Schneider, F. B. (2000). Enforceable Security Policies. ACM Transactions on Information and System Security 3(1), 30–50.

https://www.cs.cornell.edu/fbs/publications/EnfSecPols.pdf

Scope: Limits of execution monitoring.

[9] Le Goues, C., Nguyen, T., Forrest, S., and Weimer, W. (2012). GenProg: A Generic Method for Automatic Software Repair. IEEE Transactions on Software Engineering 38(1), 54–72.

https://web.eecs.umich.edu/~weimerw/p/weimer-tse2012-genprog.pdf

Scope: Test-guided construction as a relevant baseline family.

[10] Smith, E. K., Barr, E. T., Le Goues, C., and Brun, Y. (2015). Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair. ESEC/FSE. DOI: 10.1145/2786805.2786825.

https://squareslab.github.io/materials/SmithOverfitting2015.pdf

Scope: Primary evidence motivating independent semantic tests.

[11] Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. (2024). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? ICLR.

https://arxiv.org/abs/2310.06770

Scope: Primary repository-task evaluation precedent.

[12] Hoeffding, W. (1963). Probability Inequalities for Sums of Bounded Random Variables. JASA 58(301), 13–30. DOI: 10.1080/01621459.1963.10500830.

https://doi.org/10.1080/01621459.1963.10500830

Scope: Fixed-family bounded-loss precision planning.

[13] Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., and Roth, A. (2015). The Reusable Holdout: Preserving Validity in Adaptive Data Analysis. Science 349(6248), 636–638.

https://pubmed.ncbi.nlm.nih.gov/26250683/

Scope: Adaptive evaluation caveat; no reusable-holdout implementation claimed.

[14] Architect X13 (2026). Eidara Continuum Bible. Internal source edition, sections 2.4.1–2.4.2, p. 10.

https://eidara.ai/science-philosophy/

Scope: Canonical motivation; GR-2 is a new proposed formalism. Public URL provides context, not the private source text.