Self-Improving Intelligence

Representation, Discovery, and Recursive Revision

Komorebi P.
8 October 2026

First consolidated edition
Version 1.0.0

Abstract

Improving a solver is not the same as improving the procedure that produces its successors. This first edition develops a contract-relative account in which representations, experiments, repair languages, evidence, and costs remain explicit as discovery procedures change. Exact finite synthesis produces diagnostic policies with execution and complete-storage guarantees, including uncertainty about one fixed experiment model. A checked compiler shortcut avoids exhaustive search when a candidate attains proved lower bounds. A counterexample-guided learner then synthesizes an internal pruning rule. A further acquisition procedure constructs a proof-checked equation library that reduces later synthesis work while preserving the exact candidate and counterexample transcript. Repair-relative weight certificates extend the diagnostic method beyond singleton-success repairs. The final integration tracks which guarantees remain applicable when semantics, grammar order, repairs, witnesses, or resource prices change. A complete resource ledger separates preservation of meaning from reductions in particular event counts and from end-to-end improvement. The contribution is an integrated theoretical framework, conditional proofs, and executable finite witnesses. It does not establish general intelligence, autonomous invention of its trusted substrate, or indefinite recursive acceleration.

Scope. Self-contained mathematical arguments and exact finite reference constructions. Prior experimental histories are not used as evidence for the proposed integrated system. First edition freezes this theoretical scope. Independent peer review and proof-assistant verification are not claimed.

Contents

Reading the first edition ii
First-edition scope and dependency guide iii
I The object that improves 1
1 From a better answer to a better investigator 1
2 Related work and the proposed contribution 2
3 A protected comparison for improvement procedures 3
4 A small world with reusable diagnostic structure 4
II Discovering and transporting repair 5
5 A residual constrains a diagnosis 5
6 One investigation, one repair, and two resource bounds 6
7 Investigating uncertainty in the experiment model 8
8 Selecting a better procedure without hiding an oracle 11
9 Transporting an investigation, not merely its result 12
10 Growing a repair language without losing its old scope 13
III Improving the procedure that discovers repairs 14
11 A certificate can remove the need for exhaustive search 14
12 The cost of certifying a better compiler 16
13 A bounded selection of a better discovery procedure 17
IV Learning a compiler's rejection rule 20
14 A failed search representation can itself be repaired 20
15 Composing a rule from primitive syntax 21
16 When an unfinished read can close a search branch 23
17 Pruning depends on what counts as a repair 25
18 What the synthesized compiler actually gains 26
V Improving how discovery rules are acquired 29
19 The next retained object is a search representation 29
20 Learning equations without confusing probes with proofs 29
21 A certified quotient preserves the investigation's evidence 30
22 A repair-relative certificate that can be updated cheaply 33
23 One chain from acquired equations to an improved investigator 35
VI Keeping a revision's guarantees applicable 37
24 A certificate belongs to a changing realization 37
25 One resource ledger for the complete improvement 39
VII Evidence and recursive continuation 41
26 Evidence that survives candidate selection 41
27 Sequential evidence for an improved discovery procedure 42
28 The complete cost of becoming a better improver 44
29 A finite certificate for repeated improvement 45
30 Acceleration is an additional hypothesis 46
VIII A bounded route toward general intelligence 48
31 A proposed learned architecture 48
32 The compiled diagnostic improvement 49
33 Generality requires more than semantic relabeling 51
34 Open problems beyond the first edition 52
35 Conclusion 53
A A complete diagnostic trace 54
B Reference computations and evidence scope 54
C Claim map and the scope of the first edition 55
References 56

Reading the first edition

The central question is what has changed when a system becomes better at making itself better. More stored answers can improve a solver without improving its learning procedure. A richer repair language can make a correction expressible without making it discoverable. An improved search procedure can find useful corrections without establishing that its own descendants will search better still. The paper keeps these distinctions visible within one account.

A conceptual reading can follow Sections 1–4, the interpretations after the formal results, and Part VIII. A mathematical reading proceeds through diagnosis and uncertain-model synthesis, then certified compiler revision, grammar synthesis, acquired search quotients, repair-relative certificates, sequential evidence, and recursive progress. Part VI binds the guarantees to their changing inputs and consolidates the resource ledger. The running instrument is finite and inspectable. Appendix A gives a complete diagnostic trace, and the source package contains the parent and descendant compilers, frozen procedure, rule-synthesis, and equation-library records, exact checks, and an itemized review record.

Relation to earlier work. *Machine Understanding* specifies what representations must preserve across inquiry and explanation [1]. *Stratified Recursive Improvement Closure* (SRIC) supplies the earlier distinction between task, improvement-process, and epistemic improvement, together with repair transport and evidence-preserving lineages [2]. Neuralese and Operator-Symbolic Language (OSL) distinguish provisional latent computation from an explicit semantic interface [3, 4]. This paper consolidates their theoretical roles. It neither reproduces their experimental histories nor imports a complete-system claim from them. Every formal statement used below is proved here or identified as a cited foundation.

What is new in this manuscript. The proposed integration makes the improvement procedure itself a charged, transferable computational object. Its observations must support an appropriate repair, its decisions must fit one executable policy, and its acceptance must refer to a preserved evaluation contract. The finite constructions sharpen those obligations. Partition refinement, decision trees, dynamic programming, statistical concentration, and simulation arguments have classical foundations. Their appearance here is not a priority claim for the underlying mathematics.

Conventions. The principal constructive models are finite and deterministic. A stochastic law appears only with an explicit probability model. A protected predicate means a declared operational constraint, not a proof of moral adequacy. An exact reference check verifies the implemented finite case. A proof supplies the general conditional argument. No benchmark score, test count, or version number substitutes for the other.

Symbol	Meaning
S, U, L	A target solver, an improvement procedure, and a reusable library.
Ξ, B	An evaluation contract and its resource allowance.
$\mathcal{R}_\Xi(U, L; B)$	Expected improvement produced by the procedure, not an oracle over its unselected candidates.
$B_0 \subseteq X$	Possible current states of a finite repair problem.
e, r, P	An experiment, a terminal repair, and an executable diagnostic tree.
$F_h(B_0)$	Nondominated execution/storage pairs for successful trees of depth at most h.
σ, τ	Correspondences of states and operations in a declared transfer.
$\mathcal{K}, W_{h,k}$	A safe continuation set and states guaranteeing k progress events within h attempts.

First-edition scope and dependency guide

This edition closes an editorial and mathematical scope, not the open problem of general intelligence. It develops one finite chain in which a solver, its diagnostic compiler, an internal search rule, and the representation used to acquire such rules are separate objects. Each generated object is accompanied by the assumptions needed to interpret it.

Layer	Constructed or acquired here	Still supplied
Repair	Executable diagnostic tree	Experiment model and successful-repair meanings.
Compiler	Selected or pruned discovery procedure	Candidate skeletons, lower-bound proofs and witness checker.
Rule acquisition	Arithmetic pruning expression and counterexamples	Grammar, feature registry and teacher.
Acquisition representation	Checked equations and ordered representatives	Arithmetic semantics, primitive proof rules and equation learner.
Continuing validity	Replayed certificates and explicit invalidation	Dependency-complete registry and sound local checking base.

The principal formal dependency runs from finite repair synthesis to transport and guarded compiler revision, then from sound equations to ordered-trace preservation, and finally to composition of the acquired rule with the diagnostic compiler. Repair-weight pruning is a separately scoped extension, not a consequence of the singleton certificate. Statistical lineage evidence and resource inequalities add independent premises to any claim of repeated improvement.

How to interpret the demonstrations. The finite programs generate or acquire the indicated objects. Their control problems and arithmetic languages are analytical constructions. Exact checks test the implemented domains; prose proofs establish the general conditional statements. No historical agent experiment supplies a missing premise. A smaller event count is not a universal runtime result, and a generated rule is not evidence that its teacher or proof language invented itself.

Review route. Read Sections 3, 6, 21, 23, 24, and 25 for the connected core. The supplied statement review identifies the premises and proof route for every formal result. The source map records the actual depth of reference checking. Open questions in Section 34 remain open research within the program rather than unfilled requirements for this edition.

Part I

The object that improves

1 From a better answer to a better investigator

A system encounters a failure, changes its representation, and succeeds on the next attempt. Something useful has happened. The system may have learned a fact, recovered a missing variable, or selected a more suitable operation. Yet the next unfamiliar failure may leave it no better equipped than before. It has a repaired answer without a reusable account of repair.

A stronger change would preserve the way the failure was investigated. The system might discover that its representation merged two situations with different future consequences. It might learn to seek an observation that separates those situations, to reject an inadequate family of repairs, or to reuse a diagnostic method in another notation. The object retained is then not merely a solution. It is part of a procedure for acquiring solutions.

The recursive question begins one step further out. Can the system improve that procedure, use the revised procedure to construct another improvement, and retain a defensible comparison across those changes? The question is not answered by putting an agent's source code in its own context. Editability is an affordance. Productive revision is a property of what the permitted edits actually achieve.

We take general intelligence as a motivating problem rather than a theorem premise. A broad reasoner should acquire useful capabilities across substantially different tasks without receiving their solutions inside an uncharged interpreter. Chollet's account of skill-acquisition efficiency emphasizes scope, priors, and experience rather than demonstrated skill alone [5]. Our narrower contribution concerns one possible mechanism behind such acquisition: reusable procedures for diagnosing failures, choosing informative actions, constructing corrections, and preserving their scope.

1.1 Three distinct achievements

Task improvement changes how well a solver performs a declared task. *Process improvement* changes how effectively an improvement procedure produces useful solvers from comparable starting conditions. *Epistemic improvement* changes how reliably the evaluation process distinguishes genuine gains from misleading evidence. This separation is inherited from SRIC [2]. A system can advance on one axis without advancing on the others.

A larger archive, for example, can improve retrieval while the procedure that extends the archive remains unchanged. A stronger checker can reject more invalid proposals while producing no additional valid proposals. A faster search may exploit a defect in its evaluation. None of these possibilities makes improvement meaningless. They identify the claim that must be tested.

Recursive improvement also differs from sustained acceleration. A sequence of positive gains can converge to a finite limit. A sequence of better algorithms can consume increasing development costs. A procedure can improve on one task family and deteriorate on another. The substantive claim is not that a feedback loop exists, but that the loop produces useful and transferable changes under a stable comparison.

1.2 The route developed here

The paper moves from representational failure to a finite repair problem, from that problem to a compiled procedure, and from the procedure to transferable and statistically accountable improvement. It then treats the

succession of procedures as another finite control problem. This last step makes a common quantifier visible: the system needs one policy that can carry its claims forward, not a different hypothetical policy for every desired property.

The route does not require every useful correction to be formally proved beneficial on all future tasks. Exact semantic certificates and statistical performance evidence have different jobs. Nor does it require each system to invent its own computational substrate. A finite search language may be supplied. The theory must disclose what that language already makes possible and what the learner actually selects or extends.

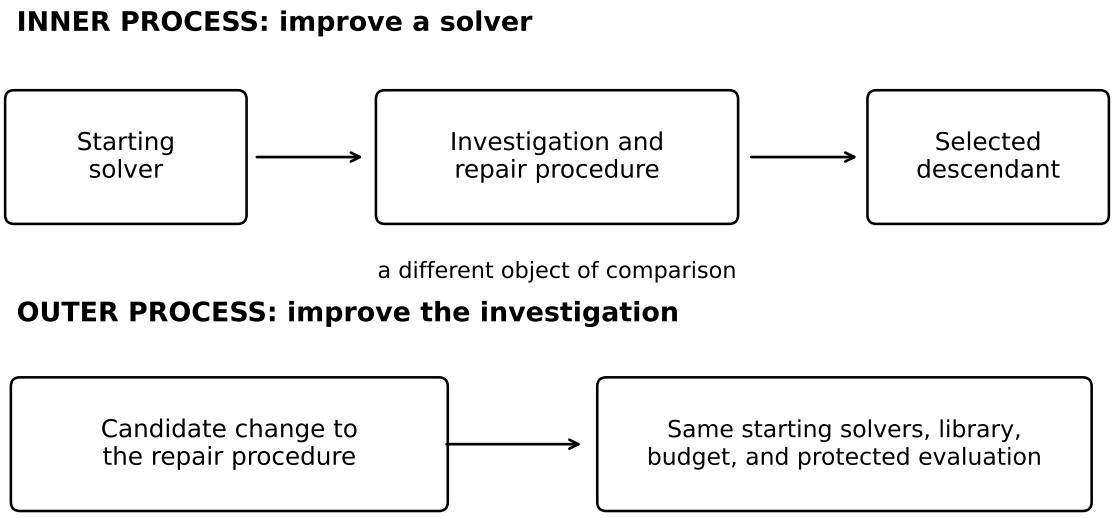


Figure 1: The inner process repairs a solver. The outer process changes how repair is attempted. Arrows indicate proposed information flow, not an implemented neural agent.

2 Related work and the proposed contribution

Learning how to learn has a substantial history. Learned optimizers can acquire update rules from optimization problems [6]. Model-Agnostic Meta-Learning trains parameters from which a small number of updates can adapt to a new task [7]. These approaches show why adaptation mechanisms deserve direct study. Neither the phrase meta-learning nor the presence of an outer objective, by itself, establishes an indefinitely improving outer procedure.

DreamCoder couples learned program libraries to neural search guidance [8]. Its relevance is the interaction between a language for expressing solutions and the ability to search that language. SEAL instead learns to produce adaptation data and update directives whose value is evaluated after the resulting parameter update [9]. These are different places at which an improvement process can become learnable.

The Gödel-machine formulation treats a self-rewrite as admissible after a proof of usefulness within encoded assumptions [10]. The Darwin Gödel Machine uses empirical evaluation and an archive of modified coding agents [11]. They should not be conflated. A proof-based criterion and a benchmark-based criterion support different guarantees and have different costs.

More recent systems sharpen the distinction between a task agent and its modifier. Hyperagents makes both roles editable and studies transfer of improvement strategies [12]. Its reported experiments include improvements to modification procedures and transfer across tested domains. The archived variants and their

selection protocol remain part of the evaluation setup. The general expressiveness of the editable program is not itself evidence that efficient discovery occurs on every computable task.

The September 2026 AIDE[2] preprint studies repeated modification of a research agent's harness and reports transfer beyond its selection tasks [13]. Its separate comparison of a discovered agent as an outer-loop improver is explicitly inconclusive relative to the strong reference. This distinction is directly relevant: improvements in research performance and improvements in driving the next improvement loop are related hypotheses, not interchangeable observations. These public works are research context. Their empirical findings are not treated as evidence for the architecture proposed here.

Causal abstraction provides a formal account of mechanism-preserving interpretation [14]. Adaptive data analysis studies why repeated access to evaluation information can invalidate ordinary statistical reasoning [15]. Both are necessary neighbors. A self-improver must know what its representations mean and what its evidence permits it to conclude.

2.1 What this paper adds to that discussion

The intended contribution is an integrated realization contract. A candidate improvement identifies its target, information access, experiment family, repair language, controller, retained evidence, and charged resources. It also identifies the comparison under which the procedure itself is said to improve. The exact results concern finite cases of that contract. The architectural hypotheses concern whether learned proposal and acquisition mechanisms can satisfy it economically on wider tasks.

This is not a claim that program libraries, recursive search, causal abstraction, or guarded updates originated with Komorebi. The aim is to connect them without allowing the cost or uncertainty of one stage to disappear inside another. A valid final repair is weaker than a valid procedure for finding it. A transferable repair is weaker than a transferable investigation. A safe accepted edge is weaker than a lineage with a continuing path to further improvement.

3 A protected comparison for improvement procedures

Let S denote a solver implementation, U an improvement procedure, and L the library available to that procedure. The distinction is functional rather than physical. They may occupy one executable program. Their different roles are retained so that changes can be attributed rather than hidden in a single checkpoint name.

Definition 3.1 (Improvement contract)**.** A contract Ξ specifies a distribution $\mathcal{H}$ of starting-solver/task pairs, an initial information and library boundary, permitted observations and modifications, a resource allowance B, protected predicates, and an evaluation rule $J_\tau(S) \in [0, 1]$. The improvement procedure returns one selected descendant or the unchanged baseline. Selection must use only the admitted development information. The final evaluation is not available as an oracle for choosing retrospectively among all candidates.

Write $U(S, \tau, L; B)$ for the selected result, including the procedure's randomness where applicable. Its productivity is

$$\mathcal{R}_\Xi(U, L; B) = \mathbb{E}_{(S,\tau)\sim\mathcal{H}}\big[J_\tau(U(S, \tau, L; B)) - J_\tau(S)\big]. \tag{1}$$

Invalid execution and out-of-budget behavior receive the treatment fixed by Ξ, not an exclusion chosen after seeing the outcome. The score may be a primary coordinate of a larger profile. Resource, validity, and group-specific constraints need not be folded into one scalar.

The value in (1) lies in $[-1, 1]$. A procedure that may return the baseline need not achieve a nonnegative true value: noisy selection can still choose a worse descendant. A nonnegative certificate requires additional evidence. The mere existence of a fallback does not make every selector good at using it.

3.1 Fixed-library and full-system comparisons

A comparison of $\mathcal{R}_\Xi(U', L; B)$ with $\mathcal{R}_\Xi(U, L; B)$ isolates process change at a common library. A comparison with (U', L') measures the combined descendant. Both are useful, but they answer different questions. A crossed comparison of (U, L), (U', L), (U, L'), and (U', L') can expose interaction between procedure and library. There is no general additive decomposition that makes their contributions independent.

The same discipline applies to base models, training histories, tools, and stored controllers. A changed foundation model is a changed component. A foreign-agent test must supply a valid adapter or charge discovery of one. Calling two targets unfamiliar does not make them distributionally independent of the development process.

Counterexample 3.2 (More knowledge without a better improver)**.** Consider four equally likely task names. A fixed procedure returns a stored solution when one exists and otherwise leaves the solver unchanged. A library containing one answer yields a smaller expected gain than a library containing all four. The procedure has not changed. Holding the library fixed gives identical productivity for the two copies of the procedure. The full system improved, but this example contains no improvement of the improvement rule.

3.2 What persists across a recursive step

A lineage state can be written schematically as

$$Z_t = (S_t, U_t, L_t, E_t, \Xi_t, \rho_t),$$

where E_t records evidence and rejected alternatives, and ρ_t records remaining resources and scope. A proposed modification may target any admitted subset. A comparison across different Ξ_t needs a declared bridge. It cannot be justified merely because both scores are called accuracy.

The public record should distinguish which changes were selected by the system, which were supplied by an external designer, and which were only proposed. A successful finite search within a fixed grammar is a genuine computational result. It is not evidence that the grammar was invented, that a broader domain was covered, or that the outer search process improved itself.

4 A small world with reusable diagnostic structure

Consider eight cells with exactly one latent defect, indexed by $j \in \{0, \dots, 7\}$. The solver must repair that cell while preserving the others. Its old representation retains only the number of defects, so every episode initially looks identical.

A diagnostic query names a subset A and returns $\mathbf{1}\{j \in A\}$. The reference language contains eight singleton queries and three bit-coordinate queries. Repair r_i succeeds exactly when $i = j$. Because repair is irreversible within the episode, a procedure may commit only when every surviving possibility justifies its choice. Otherwise it may continue investigating or abstain.

The model is fully declared. Setting each known cell as the defect on a calibration copy would permit learning the complete query table in $8(8 + 3) = 88$ observation calls. The code instead generates that table from the analytical specification and records it as supplied information. It does not claim to have executed those calibrations or inferred a hidden environment.

Equality-first testing asks whether $j = 0$, then $j = 1$, and so on. Its worst case needs seven queries, identifying the last cell by elimination. The three bit-coordinate queries need only three. Each procedure then performs one repair. Both complete binary trees have eight leaves and fifteen stored records. Faster execution has not made the controller free.

Part II

Discovering and transporting repair

5 A residual constrains a diagnosis

A residual records disagreement between a prediction and an observation, between a proposed operation and a required invariant, or between two routes of translation. It is evidence against a conjunction of assumptions. It does not uniquely identify which assumption failed.

For example, an unexpected output can arise because a state variable was discarded, a transition was modeled incorrectly, the observation was noisy, or an action had a side effect omitted from the model. These explanations can agree on the current discrepancy and disagree about the next useful experiment. An automatic instruction to enlarge the representation after every error would confuse missing information with faulty dynamics.

Let a hidden case θ determine a set $\mathcal{R}(\theta)$ of successful repairs. The set may have several members. Diagnosis need only obtain enough information to choose a repair in the appropriate set. It need not identify every hidden property of θ.

Proposition 5.1 (Feedback indistinguishability limits repair)**.** *Suppose two cases have disjoint successful-repair sets and induce transcript laws P_0, P_1 under a fixed admitted investigation protocol. A repair selector observes only the transcript and independent private randomness. Under equal prior probabilities, its average probability of successful repair is at most*

$$\frac{1 + d_{\mathrm{TV}}(P_0, P_1)}{2}.$$

In particular, identical transcript laws rule out certain success in both cases.

Proof. From the selected repair construct a binary decision: report case zero if the repair belongs to $\mathcal{R}(0)$, and case one otherwise. Success in case zero implies a correct decision. Success in case one also implies a correct decision because the two repair sets are disjoint. Thus repair success is at most the success of this binary test. For a randomized test with probability $f(o) \in [0, 1]$ of reporting zero, the latter success is

$$\tfrac{1}{2}\big(1 + \mathbb{E}_{P_0} f - \mathbb{E}_{P_1} f\big) \le \tfrac{1}{2}(1 + d_{\mathrm{TV}}(P_0, P_1)).$$

The inequality follows by summing the positive differences of the two finite probability laws. The same argument applies to any measurable transcript space with the usual total-variation definition. □

The bound concerns the entire transcript, not each observation marginal in isolation. Correlations between observations can identify a case even when every separate marginal is uninformative. Randomized action selection is already covered when it is part of the declared protocol and its observable record.

Nor does the bound identify the most informative protocol. The learner must choose experiments before receiving their outcomes. A protocol that distinguishes the cases may exist while being inadmissible, too expensive, or absent from the current language. The finite synthesis below makes those possibilities separate outcomes.

5.1 Repair sufficiency is not full identification

Suppose $\mathcal{R}(\theta_1) = \{a, c\}$ and $\mathcal{R}(\theta_2) = \{b, c\}$. The cases are distinct, yet repair c works in both. A method can stop without learning which case occurred. Conversely, pairwise overlap of repair sets does not guarantee one repair for an entire uncertainty set. The three sets $\{a, b\}$, $\{b, c\}$, and $\{a, c\}$ intersect pairwise and have empty total intersection. The stopping condition must refer to the whole surviving set.

This matters for representation discovery. A useful latent state may preserve exactly those distinctions needed for future repair decisions rather than every property of the underlying process. Whether that state remains sufficient after new repair operations become available is a separate question.

5.2 Repairing a failed representation

Let X be a finite set with deterministic operations $F_u : X \to X$ and output labels $q(x)$. Begin with a partition P_0 that preserves both the old retained labels and q. Define the next partition by splitting a block whenever two states in it have successors in different current blocks under some permitted operation. Equivalently,

$$x \equiv_{P_{t+1}} y \quad \Longleftrightarrow \quad x \equiv_{P_t} y \text{ and } F_u(x) \equiv_{P_t} F_u(y) \text{ for every } u.$$

The construction uses a complete transition description. Having a few observed contradictions does not give a learner that table for free.

Proposition 5.2 (Least stable refinement of a retained representation)**.** *The splitting construction stabilizes after at most $|X| - |P_0|$ strict refinements. Its final partition is the coarsest refinement of P_0 on which every F_u has a well-defined quotient operation. It preserves every finite continuation of the retained labels and outputs.*

Proof. A strict refinement increases the number of nonempty blocks, which is bounded by $|X|$. At a fixed point, equivalent states have equivalent successors, so the quotient operations are well-defined. Induction on word length then preserves all retained labels after every finite word. For minimality, let R be any stable partition refining P_0. If R refines P_t, stability of R implies that R-equivalent states have successors in the same P_t blocks, so they remain together in P_{t+1}. Induction shows that R refines every P_t and hence the terminal partition. □

This is a standard partition-refinement argument expressed as a repair of an old representation. It is not necessarily the smallest representation for the outputs alone: P_0 may retain distinctions that are no longer needed. Discarding those distinctions requires a different contract. The continuation and cost perspectives are developed more fully in *Machine Understanding* [1].

A counterexample pair is useful because it proves that a current block is too coarse under the stated transition model. It does not specify how a learned encoder can acquire the missing coordinate from its actual inputs. The mathematical quotient and its realizable acquisition procedure remain separate objects.

6 One investigation, one repair, and two resource bounds

We now give a finite model in which that acquisition problem can be decided exactly. The result is deliberately restricted, but it charges the complete investigation rather than only a selected final answer.

Definition 6.1 (Finite repair experiment model)**.** A model has a finite nonempty state set X, a finite experiment set $\mathcal{E}$, and a finite repair set $\mathcal{A}$. Experiment e is allowed on a domain $D_e \subseteq X$, changes the state by $f_e : D_e \to X$, and reports $o_e : D_e \to O_e$. Its execution cost $c_e \geq 0$ is fixed and known. Repair a has fixed cost $c_a \geq 0$ and succeeds on a declared set $G_a \subseteq X$. All costs are in one stated unit. A nonempty set $B_0 \subseteq X$ records the possible current states.

A chosen experiment must be uniformly allowed: $B_0 \subseteq D_e$. After observing o, the possible *post-experiment* states are

$$B_{e,o} = \{f_e(x) : x \in B_0,\ o_e(x) = o\}. \tag{2}$$

Only nonempty branches are retained. A terminal repair is justified when $B_0 \subseteq G_a$. If the original state or an earlier obligation matters, it must be retained as an immutable tag in X. Merely changing the current state into a known state is not recovery of its history.

A diagnostic program is a finite rooted tree. An internal record names an experiment and branches on its possible report. A leaf names a repair justified on that branch's surviving states. We count one storage record per node of the complete tree. Shared subroutines, compressed code, and graphs with merged suffixes belong to other storage models. The present count is chosen because it is exact and unambiguous.

Let $W(P, B_0)$ be the largest execution cost of a complete path, including its terminal repair. Let $M(P)$ be the number of stored tree records. These costs are not independently minimized. A single tree has to realize both.

For horizon h, let $\mathcal{F}_h(B_0)$ be the nondominated attainable pairs (w, m). A pair dominates another when it is no larger in either coordinate and strictly smaller in at least one. Let $T(B_0) = \{(c_a, 1) : B_0 \subseteq G_a\}$. Then

$$\begin{aligned} \mathcal{F}_0(B_0) &= \operatorname{Pareto} T(B_0), \\ \mathcal{F}_h(B_0) &= \operatorname{Pareto}\left(T(B_0) \cup \bigcup_{e:B_0 \subseteq D_e} \left\{\left(c_e + \max_o w_o,\ 1 + \sum_o m_o\right) : (w_o, m_o) \in \mathcal{F}_{h-1}(B_{e,o})\ \forall o\right\}\right). \end{aligned} \tag{3}$$

An empty child frontier makes that experiment unavailable for a successful complete tree. An empty union is not a zero-cost solution.

Theorem 6.2 (Exact finite repair synthesis with compiled storage)**.** *For every finite model, nonempty B_0, and integer $h \geq 0$, recurrence* (3) *yields exactly the nondominated cost pairs of successful diagnostic trees of depth at most h. Every retained pair has an executable witness. A work/storage allowance (b, m) is feasible if and only if some retained pair is componentwise at most (b, m).*

Proof. At horizon zero only a justified terminal repair can succeed, giving $T(B_0)$. Assume the claim at $h - 1$. A successful tree at horizon h either stops immediately or has one uniformly allowed first experiment. Its child at each possible report is successful on exactly the post-experiment support in (2). The execution cost is the root cost plus the worst child cost. The storage is the root record plus the sum over every child, including children not traversed in a particular execution. Conversely, choosing a successful child for each report constructs a successful root tree with those costs. Replacing a dominated child by a dominating one never increases the root maximum or sum. Thus Pareto pruning loses no feasible allowance. Induction proves both completeness and witness construction. □

The theorem is an application of finite dynamic programming, not an efficient general solution to scientific discovery. Its state subsets and child combinations may grow exponentially. It assumes the transition, observation, legality, and success tables are correct. It also does not include the cost of constructing those tables in its online path cost. Acquisition of the model, compiler execution, and validation are separately charged development resources.

6.1 A more informative question can be a worse repair question

Counterexample 6.3 (Entropy reduction need not minimize repair work)**.** There are four equally likely cases. Repair a succeeds in $\{0, 1, 2\}$ and repair b succeeds in $\{3\}$. A balanced query separates $\{0, 1\}$ from $\{2, 3\}$. Another query separates $\{0, 1, 2\}$ from $\{3\}$. Both cost one unit and leave the state unchanged. The balanced

query has the larger entropy reduction, but its second branch still needs another observation. The unbalanced query supports a correct repair after one observation in every case. Maximizing information about the entire hidden case is not the same objective as minimizing the cost of a successful correction.

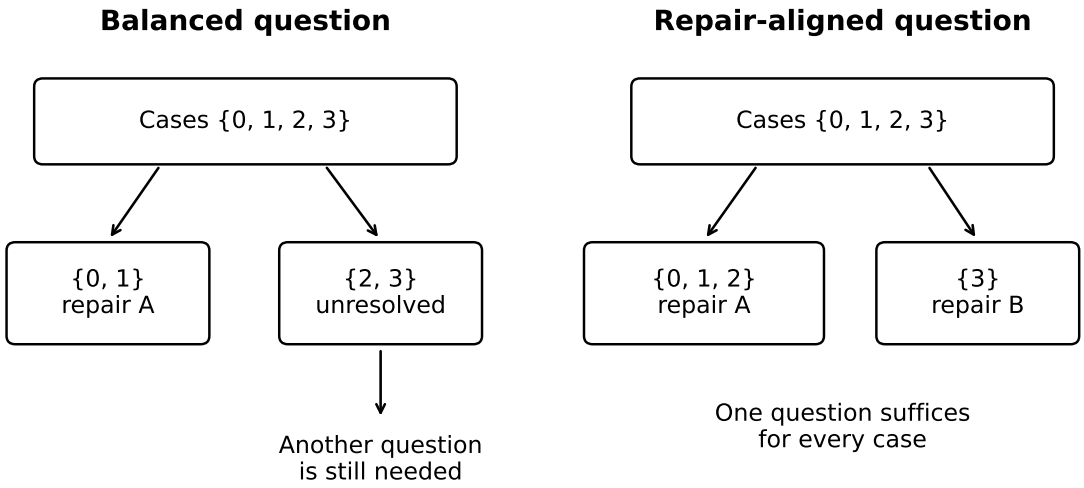


Figure 2: Two equally priced questions about four cases. The balanced split leaves one branch with incompatible repairs. The repair-aligned split identifies a valid action immediately. The figure represents exact sets, not measured agent behavior.

The example suggests a learning target more specific than general curiosity. A useful acquisition policy should distinguish alternative *decisions* when needed, preserve unresolved alternatives when they matter later, and avoid spending work on irrelevant details merely because those details are easy to predict.

6.2 What a compiled witness proves

The reference implementation returns a concrete tree together with its cost pair. A separate interpreter checks its branch coverage, the domains of its experiments, the success of each leaf, and both resource counts. The checker does not infer that the environment model is true. It checks the relation between a specified model and the proposed procedure. This is the same reason an OSL semantic check and a learned Neuralese realization need separate evidence [4, 3].

7 Investigating uncertainty in the experiment model

The first synthesis theorem assumes that the experiment table is known. This is a substantial informational premise. A reasoner may know which experiments it can request while being uncertain about their outcomes, side effects, or interpretation. Optimizing one guessed table can produce a perfectly checked procedure for the wrong instrument.

A useful first extension retains several explicit possibilities rather than replacing uncertainty by one convenient estimate. Let Θ be a finite nonempty set of candidate models with a common state set X, experiment names, repair names, and nonnegative costs. Model θ specifies D_e^θ, f_e^θ, o_e^θ, and G_a^θ. The

unknown model is fixed during an investigation. The initial information is a nonempty set $B \subseteq \Theta \times X$. The procedure observes reports, not the model label or the hidden physical state.

An experiment is allowed only when it is allowed under every pair still possible. Its update is

$$B_{e,o} = \{(\theta, f_e^\theta(x)) : (\theta, x) \in B,\ x \in D_e^\theta,\ o_e^\theta(x) = o\}. \tag{4}$$

The model tag remains unchanged. Observations can exclude a model, but the algorithm cannot switch the underlying model between steps merely because doing so produces a convenient worst case. A terminal repair a is justified when $x \in G_a^\theta$ for every $(\theta, x) \in B$.

This is a finite form of planning under model uncertainty. Robust dynamic programming provides an established neighboring literature [17]. The present construction is not an application of a rectangular transition-uncertainty assumption. It explicitly remembers a fixed model label, including correlations between that model's predictions at different times.

Theorem 7.1 (Repair synthesis under a fixed unknown model)**.** *Construct a finite repair model on $\widetilde{X} = \Theta \times X$ by*

$$\widetilde{f}_e(\theta, x) = (\theta, f_e^\theta(x)), \quad \widetilde{o}_e(\theta, x) = o_e^\theta(x), \quad \widetilde{G}_a = \{(\theta, x) : x \in G_a^\theta\},$$

with the corresponding tagged domains and the common action costs. Applying Theorem 6.2 to this model gives exactly the nondominated work/storage pairs of report-dependent trees that succeed for every initial pair in B, within the declared horizon. Every retained tree is valid for the actual system whenever its model and initial state belong to B.

Proof. A tree receives the same experiment names and report alphabet in both descriptions. For each experiment and report, its possible tagged successor states are exactly (4). Its terminal repair is valid on the tagged support exactly when it is valid for every retained model/state pair. Consequently the report histories, path costs, branch counts, and stopping conditions agree node by node. Induction on tree depth gives a cost-preserving correspondence of all successful trees. The finite synthesis theorem then supplies the frontier and its witnesses. If the actual pair starts in B, each actual observation leaves its actual successor in the updated support, so the executed terminal repair is valid for that pair. □

This theorem decides a finite robust investigation problem. It does not learn an arbitrary model class or guarantee that the class contains the truth. Its negative result is also contract-relative. An empty frontier may mean that the admitted experiments cannot resolve the uncertainty, that a needed calibration is unavailable, or that the horizon is too short. It is not evidence that every possible experiment would fail.

7.1 One hidden bit and two sensor conventions

Counterexample 7.2 (Known-model repair without a common unknown-model policy)**.** Let $x \in \{0, 1\}$ be the cell requiring repair and let $\theta \in \{0, 1\}$ be a fixed sensor convention. A nondestructive query reports $o = x \oplus \theta$ at cost one. Repair r_i succeeds exactly when $x = i$ and costs one. If θ is known, one query followed by the appropriate repair always succeeds. If all four (θ, x) pairs are initially possible, report zero leaves $(0, 0)$ and $(1, 1)$, while report one leaves $(0, 1)$ and $(1, 0)$. Repeating the same query gives no further information. No finite tree using only this query can guarantee a repair, although each known model admits a two-unit procedure.

Add a permitted reference measurement that reports θ, leaves x unchanged, and costs two. The procedure can calibrate, make the ordinary query, and choose $r_{o\oplus\theta}$. Its worst-case work is four and its complete binary tree has seven records. Either measurement alone leaves both repairs possible. The two measurements are

therefore both necessary in this example, regardless of their order. The executable reference construction returns exactly $(4, 7)$ at horizons at least two.

The reference measurement is an added information channel. It is not a renamed latent coordinate and not a free inference from the original report. Its authority, reliability, and cost belong to the model. If the physical reference is faulty, the exact guarantee no longer follows. The same issue would arise if a learner excluded the true sensor convention after an unreliable observation.

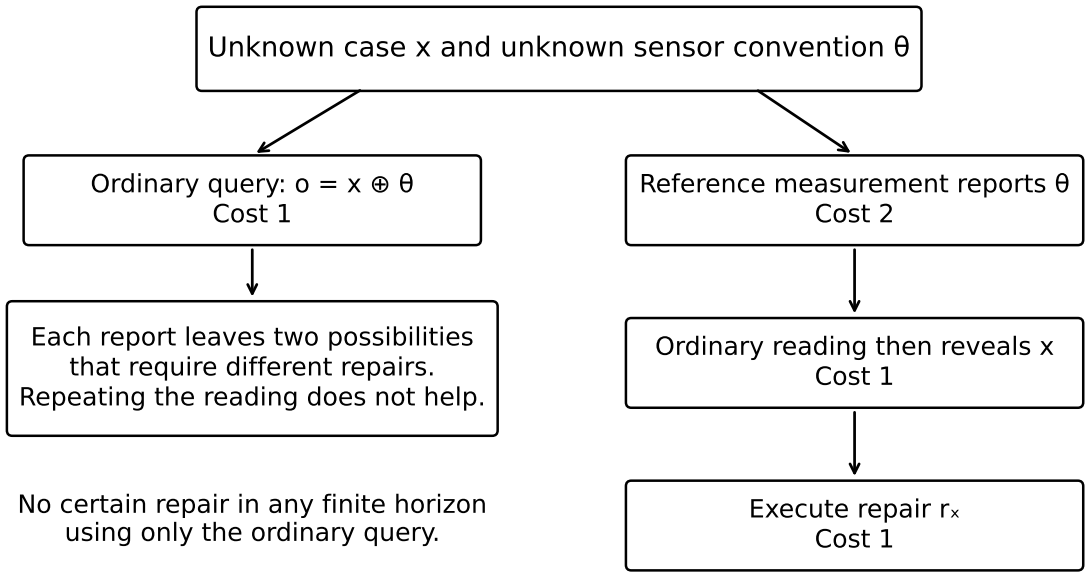


Figure 3: An ordinary reading can identify the repair under a known sensor convention yet fail under two possible conventions. A separately specified reference measurement resolves the missing distinction. The figure is an analytical model, not a calibration experiment on an agent.

7.2 Model refinement should be driven by repair relevance

There is no general need to identify the whole model before acting. Several retained models may agree on a successful repair even while disagreeing about irrelevant outputs. The tagged support retains this possibility because its stopping condition is a common successful repair, not a singleton model label. Conversely, a calibration that removes many candidate models can still leave the decisive repair ambiguity intact.

A model learner therefore has two different targets. It can seek a more accurate description, or it can seek enough justified information to produce an affordable valid repair. The former can support the latter, but the two costs need not be minimized by the same investigation. The learner should retain the unresolved alternatives that matter to future decisions rather than certify an entire environment from one successful correction.

The finite construction also makes the cost of caution visible. The tagged state set can have $|\Theta||X|$ members, and the support search can be substantially larger than a search under one assumed model. Representation refinement, model acquisition, and compiler improvement are consequently coupled research problems. A more reliable model can increase the computational burden of planning until a better discovery procedure exploits its structure.

8 Selecting a better procedure without hiding an oracle

The finite compiler can itself be the object modified by an outer process. One candidate may change the experiment order. Another may change the cost model, add a repair constructor, or select a different representation for the uncertainty set. Their effects should not all be described as the same kind of learning.

For a fixed finite collection $\{U_1, \ldots, U_N\}$, suppose a declared assessment gives estimates $\widehat{R}_i$ of true productivities R_i. This assessment can be exact on a fully specified finite distribution or statistical under additional assumptions. Choosing a candidate must use the assessment actually available at selection time, not a hidden final score inspected afterward.

Proposition 8.1 (Selection under uniform assessment error)**.** *If $|\widehat{R}_i - R_i| \le \varepsilon$ for every candidate and $\hat{\imath}$ maximizes $\widehat{R}_i$, then*

$$R_{\hat{\imath}} \ge \max_i R_i - 2\varepsilon.$$

If a baseline is included and the selected candidate's estimated advantage exceeds $2\varepsilon + \gamma$, its true advantage exceeds γ.

Proof. For a true maximizer i^*, $R_{\hat{\imath}} \ge \widehat{R}_{\hat{\imath}} - \varepsilon \ge \widehat{R}_{i^*} - \varepsilon \ge R_{i^*} - 2\varepsilon$. Apply the same two error bounds to the selected candidate and the baseline for the second assertion. □

The uniform premise is the essential obligation. A small error for each candidate considered in isolation does not automatically control a candidate selected after adaptive access to the same evaluation record. Section 26 addresses an explicit alternative based on conditional validity.

8.1 Changing the selector and changing the language

An improved selector over a fixed candidate family can produce higher gains without making any previously impossible repair expressible. Conversely, a richer language can increase the best achievable gain while a weak selector never finds the new repair. These are distinct forms of headroom, already separated in the earlier SRIC treatment [2].

For a fixed budget, let $R^*_{\mathcal{G}}$ be the largest productivity among procedures expressible in a declared language $\mathcal{G}$, with installed data and interpreter costs held fixed. The gap $R^*_{\mathcal{G}} - R(U)$ is search or selection headroom *within that language*. Comparing $R^*_{\mathcal{G}'}$ with $R^*_{\mathcal{G}}$ concerns expressibility and resources. Neither upper envelope is automatically available to the learner.

A zero observed gain from one selector update is not a proof of saturation. A genuine saturation statement needs either a complete finite search or an upper bound close enough to the incumbent value. Otherwise the failure may reflect the proposer, the evidence channel, the search allowance, or the evaluation.

8.2 The compiled instrument improver

In the eight-cell instrument, the supplied language admits both singleton and bit queries. Recurrence (3) can construct a depth-three diagnosis. The exported program replaces the equality-first diagnostic policy in an otherwise unchanged repair system. It does not contain the hidden defect index of the next episode.

Under a cap of three diagnostic calls and one repair call, the equality-first procedure certifies and repairs the first three cells, then abstains on the other five. On the uniform finite case distribution, its productivity from a zero-success baseline is $3/8$. The depth-three procedure certifies all eight cases, giving productivity 1. These are exact properties of stipulated procedures. The compiler selected a better procedure within a supplied finite language. It did not invent the language, learn a general visual representation, or improve the compiler that performed this search.

That last distinction is deliberate. This first construction improves the diagnostic procedure, not its compiler. Part III then changes the compiler, using certified early termination to construct the same optimal diagnostic controllers with fewer model-record reads. The finite selector that installs that compiler remains supplied. An archive entry saying that a descendant is more reflective supplies none of these separate claims.

9 Transporting an investigation, not merely its result

A repair can commute with a representation map even when the procedure used to discover it cannot be transferred. The investigation may depend on an unavailable observation, on a source-specific reset, or on the precise meaning of a stopping condition. A transport contract must include those actions and their evidence.

Consider source and target finite repair models. Let $\sigma : X \to X'$ be a bijection, $\tau : \mathcal{E} \to \mathcal{E}'$ a bijection of experiments, $\psi : \mathcal{A} \to \mathcal{A}'$ a bijection of repairs, and $\eta_e : O_e \to O'_{\tau(e)}$ a bijection of report labels. Assume

$$\begin{aligned} \sigma(D_e) &= D'_{\tau(e)}, & \sigma f_e(x) &= f'_{\tau(e)}\sigma(x), \\ \eta_e(o_e(x)) &= o'_{\tau(e)}(\sigma x), & \sigma(G_a) &= G'_{\psi(a)}, \end{aligned} \tag{5}$$

and matching execution costs. These assumptions concern supplied semantics. Their discovery or learned approximation is an additional problem.

Theorem 9.1 (Exact transport of diagnostic procedures)**.** *Under* (5)*, relabeling every experiment, branch, and repair transports successful diagnostic trees bijectively between the source support B_0 and target support $\sigma(B_0)$. Execution cost, complete tree storage, and depth are preserved. In particular,*

$$\mathcal{F}_h(B_0) = \mathcal{F}'_h(\sigma(B_0)).$$

The same conclusion holds for every selected tree, not only a cost-optimal tree.

Proof. A terminal repair is valid on B_0 exactly when its image is valid on $\sigma(B_0)$. For an internal experiment, domain preservation gives uniform admissibility. Transition and observation preservation yield

$$\sigma(B_{e,o}) = B'_{\tau(e),\eta_e(o)}.$$

Apply induction to every child. The report bijection preserves the number of children, the maximum execution cost, and the sum of stored child records. Inverse maps give the converse correspondence. The equality of frontiers follows because costs and dominance are unchanged. □

The theorem covers a complete explanation of how the repair is found. It is stronger than equality of the repaired output but weaker than a claim that any learned representation admits such a correspondence. A known map costs something to store and execute. In the toy instrument, an arbitrary cell permutation requires a label adapter. A permutation of eight names is not discovered merely by allocating an eight-row table.

9.1 Approximate performance transfer

Suppose a source comparison gives productivity advantage Δ to U' over U. Let $\widetilde{U}', \widetilde{U}$ be actual transported target procedures. If each procedure's productivity differs from its source value by at most ε, then

$$\mathcal{R}'(\widetilde{U}') - \mathcal{R}'(\widetilde{U}) \geq \Delta - 2\varepsilon. \tag{6}$$

This follows by subtracting the two inequalities. A common source of the premise is an exact task correspondence plus a bounded evaluation-law shift. For any fixed gain function in $[-1, 1]$, changing the

task law by total variation δ changes its expectation by at most 2δ. A further pointwise gain discrepancy η therefore gives $\varepsilon = \eta + 2\delta$.

This elementary estimate does not certify admissibility or sure success. A rare unauthorized action can have small average loss and still violate a zero-tolerance constraint. Such constraints require their own preservation check. The same is true of uncharged translation work: an equal-output target procedure can fall outside the budget after the adapter is added.

Counterexample 9.2 (Output transfer with no transferable investigation)**.** A source can inspect a hidden bit for one unit and then choose one of two disjoint successful repairs. The target has the same hidden states and repair effects, but no operation that reveals the bit. The final repair meanings correspond exactly. The source diagnostic policy does not transport, and no target policy can guarantee success from the two-state uncertainty set. The missing object is an information channel, not a better translation of the repair names.

10 Growing a repair language without losing its old scope

A persistent discrepancy can be a limitation of the candidate language rather than the search procedure. Repeatedly optimizing coefficients cannot produce a nonlinear term in a language that expresses only affine functions. Reordering uninformative experiments cannot separate two cases whose transcripts are always identical. A learner must distinguish an unsuccessful search from a certified absence of a suitable expression.

A conservative extension embeds each old program into a new language with the same meanings on the old domain. That semantic property is valuable, but it does not imply nonregression under a fixed resource allowance. The new interpreter may consume more time, memory, or evidence before executing the embedded old program.

Proposition 10.1 (The resource price of conservative extension)**.** *Suppose an embedding preserves every old procedure's successful behavior and adds at most $d \geq 0$ execution work and $s \geq 0$ stored records. Every old allowance (b, m) is then feasible in the new language under allowance $(b + d, m + s)$. Feasibility at the unchanged allowance follows only when the embedded procedure fits that allowance. Semantic conservativity alone supplies no such guarantee.*

Proof. Embed a witness for the old allowance. By hypothesis its behavior is preserved and its costs are bounded by the shifted pair. The unchanged-budget assertion requires the corresponding inequalities. For the final claim, an old one-unit procedure can be embedded with identical output into a new interpreter requiring two units of startup work. It remains semantically available but cannot execute within the old one-unit allowance. □

A practical extension should retain a directly executable old path when possible, then route only justified cases through the new machinery. The routing operation also has a cost and an information requirement. A context-dependent grammar is not a solution if deciding the context already requires solving the original problem.

10.1 Error localization and new constructors

The useful role of packets and residuals is to make this diagnosis structured. A packet can declare a domain, a retained distinction, an uncertainty set, an operation interface, and unresolved conditions. A residual identifies which preservation equation or evidence constraint failed. A constructor adds a new kind of candidate repair. The theory does not assume that a packet is an independent physical component or that its name supplies a new algorithm.

Part III

Improving the procedure that discovers repairs

11 A certificate can remove the need for exhaustive search

The compiled tree in the first example is an improved diagnostic policy. This section changes the procedure that constructs such trees. The distinction is concrete: the parent and descendant receive the same new problem table and return equally valid optimal controllers, but they need not perform the same amount of discovery work.

The parent is the exact frontier compiler of Theorem 6.2. It examines a finite set of support/horizon states and combines successful subtrees. That generality is useful, but it can do unnecessary work when one inexpensive candidate already meets a proved global lower bound. A descendant compiler can first propose such a candidate, check it independently, and stop only if its checked cost proves that further search cannot improve either protected resource coordinate.

This combines a proposal mechanism with a decisive rejection or acceptance condition. Synthesis from partial programs and counterexamples has established precedents [18]. The particular addition here is a certificate that can close an *optimality* search, rather than merely establish that one candidate is valid. Neither the proposal template nor the lower-bound argument is assumed to have been invented by the system.

11.1 The scope in which a short proof is decisive

Consider $n \geq 1$ possible immutable cases. Every admitted query is total, nondestructive, binary-valued, and costs one unit. Every available repair costs one unit and succeeds on exactly one case. At least one repair is available for each case. The target is certain successful repair from the complete set of n possibilities. A literal controller stores one record per query or repair node. Redundant unary query nodes are permitted but have their full costs.

Theorem 11.1 (Checked lower-bound closure of a diagnostic search)**.** *Every successful controller in this scope has worst-case execution work at least*

$$1 + \lceil \log_2 n \rceil$$

and storage at least $2n - 1$*. If an independently checked candidate attains both bounds and respects the requested horizon, its cost pair is the entire nondominated frontier. Returning this candidate is therefore equivalent in the protected work/storage objective to completing the exhaustive synthesis.*

Proof. Every case produces one deterministic query history. Two different cases cannot terminate at the same repair leaf because no repair succeeds in both. Thus there are at least n nonempty leaves. A tree of binary query depth d has at most 2^d leaves, so $d \geq \lceil \log_2 n \rceil$. Every path also executes a unit-cost repair. For storage, let l be the number of leaves and let i_1, i_2 count unary and binary internal nodes. Counting edges in a finite rooted tree gives $i_1 + 2i_2 = i_1 + i_2 + l - 1$, hence $i_2 = l - 1$. Total records are $2l - 1 + i_1 \geq 2n - 1$. A successful candidate attaining both coordinatewise lower bounds dominates every other attainable cost pair. Its independent validity check and horizon check make it a feasible witness, so the Pareto frontier contains only that pair. □

The immutable-case premise is essential. An experiment that resets every physical state into one safely repairable state can defeat a case-identification lower bound when only the current state matters. Multiple

cases admitting a common repair can do the same. The guard must therefore verify the actual contract before applying the bound. This is not a universal theorem that diagnosis always requires identifying the hidden state.

A candidate that is valid but misses one lower bound does not close the search. Nor does failure of the proposal rule prove that no optimum exists. In both cases the descendant falls back to the general compiler. The certificate licenses an early return, not an early claim of impossibility.

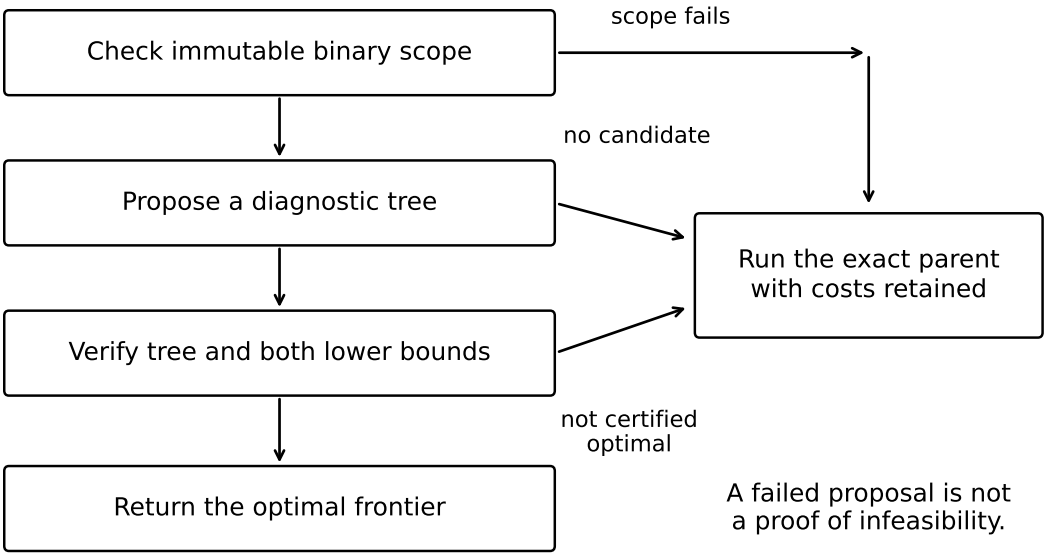


Figure 4: The descendant changes discovery control flow. A checked candidate must attain both global lower bounds before the compiler can stop early. Failure of a guard, proposal, or bound check invokes the unchanged exact compiler. The sequential reference implementation charges unsuccessful preliminary work.

11.2 The inexpensive proposal is not the proof

The reference proposer repeatedly chooses a query whose two current branches have the smallest maximum cardinality, breaking ties by the installed query order. It searches the public observation table rather than recognizing privileged names such as “bit zero.” It constructs a tree recursively, stopping when a unique justified repair remains. It aborts a proposed branch when the remaining binary depth cannot distinguish its cases.

Balancing is a supplied heuristic. It need not minimize repair work in general, as Counterexample 6.3 already shows. It is useful in the narrower singleton-repair scope because the required distinctions coincide with case identification. Even there, a locally balanced split need not extend to a globally optimal tree for an arbitrary query library. Correctness comes from the finished witness and the lower bound, not from the heuristic’s name.

The checker reconstructs every possible support, checks all reports, and independently replays every initial case. A fabricated terminal label, missing branch, insufficient horizon, or altered repair effect is rejected. This is a finite model checker authored within the project. It is not a proof of the physical observation table or an independent proof-assistant kernel.

12 The cost of certifying a better compiler

The unit used in the following comparison is a consumed record of the supplied transition/report table. A repeated access counts again. The unit is chosen because it is explicit and can be instrumented in the unchanged parent compiler and the descendant. It is not a claim to count every Python operation, allocation, set-membership test, processor instruction, or unit of energy.

Both procedures start with cold search caches and the same structurally validated table. Shared input-shape validation is recorded separately. The descendant additionally charges its semantic guard, candidate construction, and completed witness check. The parent charges its complete frontier search and the same witness-checking procedure for its returned frontier. Model acquisition, source storage, and the finite procedure-selection stage remain separate ledger entries.

Proposition 12.1 (Read cost of the certified binary-family shortcut)**.** *Let $n = 2^k$ with $k \geq 1$. Suppose there are q total unit-cost binary immutable queries, and the balanced proposer produces a perfect depth-k tree. In the stipulated implementation, the semantic guard reads every query row once, the proposer scans all q queries at each internal support, and verification performs one support reconstruction and one complete case replay. The total number of transition/report record reads is*

$$R_{\text{shortcut}} = nq(k+1) + 2nk. \tag{7}$$

The usual hidden-cell family with all singleton queries and k coordinate queries satisfies the premise, including after arbitrary cell relabeling, report-bit flips, or reordering of the query list.

Proof. The guard uses nq reads. At each of the k query levels, the current supports partition the n cases, so scanning all queries uses nq reads per level. Every case traverses k query nodes in the support reconstruction and again in the independent replay, contributing $2nk$. Summing gives (7). In the hidden-cell family, every nonsingleton support reached by coordinate splits is a subcube. A remaining coordinate divides it equally, while a singleton query cannot give a smaller maximum branch. Thus the balancing rule chooses an equal split until singleton leaves are reached. When a two-case support admits a singleton tie, either split reaches singleton leaves. Relabeling the cases or reversing report names preserves all branch cardinalities and this argument. □

The result is about a particular implemented access pattern. Caching rows, recognizing trusted structure symbolically, or using another cost unit changes the formula. A specialized cube compiler supplied with certified coordinate semantics can avoid the general search too. The result does not establish superiority to that informed comparator or to every available synthesis algorithm.

12.1 Some verification work cannot disappear

Proposition 12.2 (A table-access floor for universal scope certification)**.** *Suppose a deterministic validator receives an arbitrary table of $N \geq 1$ independently variable records only through individual record reads. It must accept every table whose records all satisfy a fixed per-record predicate and reject every table with a violating record. Assume each record position admits both a satisfying and a violating value. On every accepted table, the validator must read all N positions at least once.*

Proof. Suppose it accepts a satisfying table while leaving position j unread. Change only that position to a violating value. Every record the validator reads has the same value, so determinism gives the same sequence of reads and the same acceptance. This contradicts rejection of every table containing a violation. Therefore no accepted table can have an unread position. □

In the compiler's unrestricted table interface, an unread entry could contain a destructive transition or an additional report value. The full semantic scope cannot be certified merely by recognizing a few apparently balanced rows. Trusted construction metadata or a proof from a different representation can change the interface and reduce this cost, but that metadata then becomes a charged assumption or a separate proof object.

This identifies a useful bottleneck. Improving search can make model access and verification dominate the remaining computation. It does not make every stage improve at the same rate. Nor does it establish that this bottleneck is permanent, since a learned representation might eventually support cheaper certificates. Such a representation would need its own acquisition and soundness argument.

13 A bounded selection of a better discovery procedure

We now make the persistent change explicit. The candidate catalogue contains the unchanged parent compiler, a guarded first-nontrivial-query proposer with exact fallback, and a guarded balanced proposer with exact fallback. The guard, proposal constructors, lower-bound proof, checker, and fallback are supplied. The selection procedure runs each candidate on the four-cell and eight-cell development problems, with the same tables and cold caches. It selects the candidate with the smallest total number of charged record reads and writes a policy record before evaluating any additional problems.

The developmental totals are 10,256 reads for the parent, 10,380 for the guarded first-query rule, and 488 for the guarded balanced rule. The selector therefore installs the balanced rule. These totals concern a fixed, fully enumerated development catalogue. They do not imply a statistical generalization guarantee over an unknown problem distribution. The selected artifact changes the control flow of future compilations without containing the hidden state of a future repair episode.

Cases	Parent compiler reads	Descendant reads	Returned (work, storage)
2	16	16	(2, 3)
4	352	88	(3, 7)
8	9,904	400	(4, 15)
16	576,128	1,728	(5, 31)

Every paired entry includes complete checking of the returned controller. The sixteen-cell problem is evaluated only after the policy record is fixed. Additional cold cases remove singleton queries, restrict the initial support to six possibilities, reverse operation order, and change cell and report labels. These vary the finite structure available to the compiler. They are not evidence of transfer to an unrelated scientific domain.

Under a common cap of 2,000 table reads, the descendant returns a checked optimal sixteen-cell controller. The parent reaches the cap without returning its complete frontier. Its outcome is recorded as *incomplete within the allowance*, not as a proof that no controller exists. Without that cap, the parent also returns the same optimal cost pair. The difference lies in discovery work rather than repair validity or online diagnostic depth.

13.1 What was selected and what remains supplied

The persisted policy chooses a proposal rule from a fixed catalogue. It is not a stored solution to the next hidden-cell episode. It therefore modifies an improvement procedure rather than only adding another repaired answer. The compiler accepts arbitrary supplied tables, applies its semantic guard, and retains the exact fallback outside the shortcut's demonstrated scope.

This is nevertheless a restricted form of process learning. The catalogue does not invent a new lower-bound theorem or discover a new programming primitive. It does not improve the selector that searched the catalogue. A second call with the same candidates and the same criterion returns the same selected rule. One genuine improvement of a discovery procedure is not an indefinite sequence of improving outer procedures.

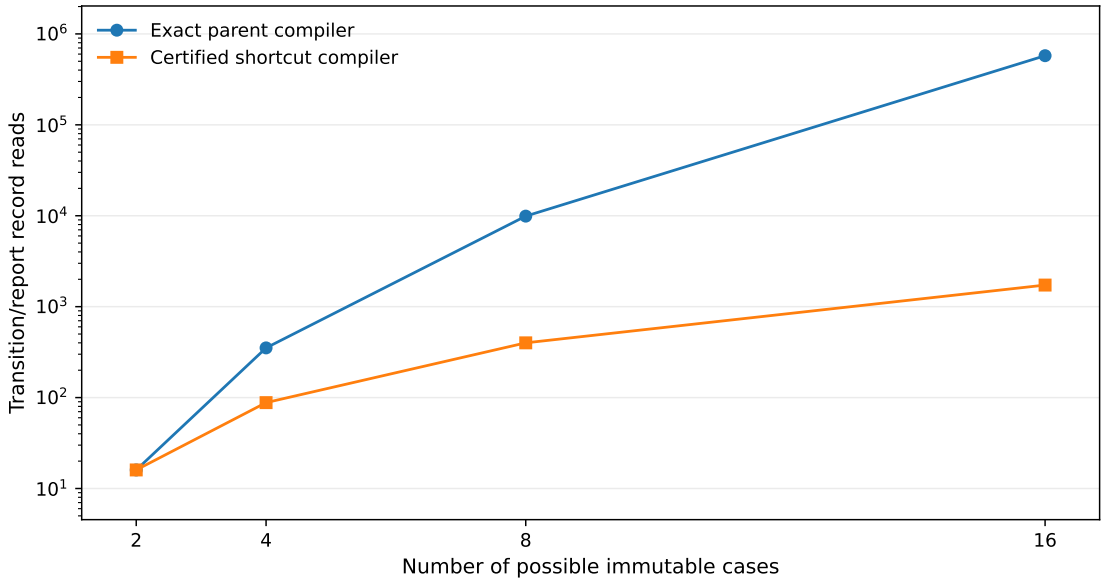


Figure 5: Charged transition/report record reads on the exact binary family, including witness checking. The vertical axis is logarithmic. Both compilers return the same optimal online work and stored tree size. These deterministic algorithm counters are not measured agent-learning performance or total CPU instructions.

The distinction gives the next research target a concrete form. A learned proposer would need to generate useful shortcut templates or representations not simply chosen from this fixed list, while the checker verifies their stated scope. A learned evaluator would need to identify when existing cost or success measurements omit a consequential requirement. Such changes would alter additional parts of the realization and need their own preserved comparison.

13.2 Fallback protects meaning, not free execution

Counterexample 13.1 (A semantically protected shortcut with a slower fallback)**.** Give the compiler four immutable cases and only four singleton queries. The horizon permits the three-query equality tree. A balanced proposal produces a valid controller but cannot attain the two-query information lower bound, since the query library does not supply the needed split. The descendant therefore performs its preliminary guard and proposal work and then runs the exact parent. It returns the same frontier but consumes more table reads. Semantic nonregression has not established work nonregression.

The counterexample is included in the shipped comparison. The declared development objective prefers the shortcut on its development family, not on every possible input. Changing the mixture of structured and fallback cases can change which procedure is preferable. Attributing all gains to the new procedure while excluding its failed proposal attempts would invalidate the comparison.

There is a conditional way to bound this risk through scheduling rather than prediction. The following result is intentionally separate from the sequential implementation above.

Proposition 13.2 (Bounded slowdown from a certified two-worker race)**.** *Suppose a parent and a shortcut worker can be paused after each unit of a common work measure and resumed without losing progress. Each may announce completion only after producing a result certified for the same target. Alternate one unit of each worker and stop at the first certified completion. If their standalone completion costs are* $D_0, D_1 \in \mathbb{N}_{>0} \cup \{\infty\}$ *and at least one is finite, at most* $2\min\{D_0, D_1\}$ *worker units are executed. If each dispatched unit costs at most an additional* $c \geq 0$ *scheduler units in the same measure, total charged work is at most* $2(1 + c)\min\{D_0, D_1\}$.

Proof. After $2d$ dispatched worker units, each worker has received exactly d units of progress. Set d to the smaller finite standalone completion cost. That worker has completed its certification by this point, so the race has stopped no later. Multiplying the number of dispatches by the maximum combined work per dispatch gives the overhead bound. □

The theorem does not provide resumability for an arbitrary compiler, bound the memory occupied by both workers, or make their setup free. The present reference compiler uses sequential fallback, not this race. The companion suite checks the abstract schedule bound separately on finite worker tapes. A systems implementation would need to include state checkpointing, dispatch cost, shared-resource contention, and both workers' storage before claiming this guarantee.

Taken together, the results distinguish three achievements: a heuristic frequently proposes a good answer, a certificate establishes that the answer closes the search, and an implementation retains an affordable fallback when the heuristic fails. None of these can substitute for the others.

13.3 A complete receipt for the sixteen-cell compilation

The selected policy is evaluated on a fresh sixteen-cell table after its bytes are fixed. The new table has twenty query rows: sixteen singleton queries and four coordinate queries. The next hidden defect is not supplied to either compiler. Both methods construct a complete controller that will work for any of the sixteen cases.

The descendant's recorded access account is

Compiler stage	Consumed transition/report records
Semantic scope guard	320
Proposal construction across four levels	1,280
Support reconstruction by the checker	64
Independent replay of all cases	64
Total	1,728

The separate common input-shape validation consumes another 320 records before the measured compilation begins. It is not included in the 2,000-read compiler allowance. Counting that shared setup on both sides gives 2,048 reads for the descendant and 576,448 for the parent. No particular implementation is permitted to hide its extra guard inside an uncharged setup step.

The three-candidate development comparison itself consumed 21,124 model-record reads. It is part of the cost of selecting the descendant, not part of one later compilation. For the declared sixteen-cell workload the per-use read saving is 574,400, so those developmental reads are recovered within one use in that same narrow accounting unit. This does not repay an unspecified human research cost, prove a wall-clock speedup on every machine, or charge the creation of the candidate templates. Source sizes and measured runtimes are recorded separately rather than silently converted into model-record reads.

The result concerns cheaper discovery of equally valid controllers. It does not establish an improved selector or an independently discovered compiler catalogue.

Part IV

Learning a compiler's rejection rule

14 A failed search representation can itself be repaired

The preceding selector chooses among three supplied compiler strategies. It changes which algorithm will be used, but does not compose a new decision rule. We now move one step inside a compiler. A small learner constructs a predicate that tells the compiler when it can stop inspecting an unpromising query. The learned object is neither a hidden-cell answer nor a completed diagnostic tree. It is an operation on the compiler's own partial information, reused while constructing trees for further problems.

The construction is deliberately restricted. Its experiment tables, compiler skeleton, primitive arithmetic language, and correctness argument remain supplied. Within that boundary, the learner is not given the complete predicate as one named strategy. It first encounters a representation that cannot express its assigned distinction, adds a specified missing feature, and synthesizes a formula from counterexamples. A structural certificate then determines whether the formula may safely prune later searches.

This separates three questions that are easy to conflate: whether the examples determine a suitable formula, whether the formula is valid outside those examples, and whether executing it actually reduces the relevant cost. The reference construction answers each under a different explicit contract.

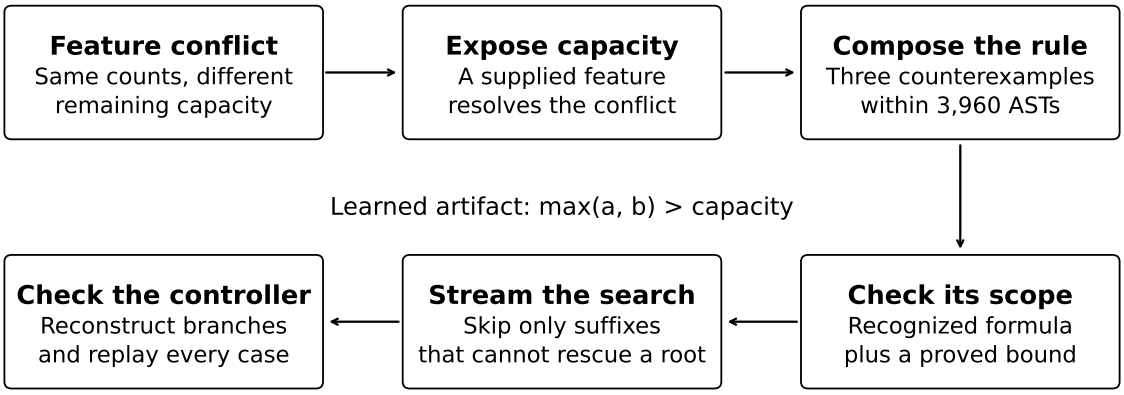


Figure 6: One complete bounded improvement. A feature conflict justifies exposing the supplied capacity feature. Counterexamples select a composed arithmetic expression. A narrow structural check links that expression to a proved scope, after which it can skip rows inside later compilations. The grammar, teacher, scope proof, and compiler skeleton remain outside the learned component.

14.1 The information available inside a search

A compiler is considering a binary query on a support of m possible immutable cases. It reads the query's public model rows in a fixed order. After a prefix, a of the inspected cases have report zero and b have report

one. There are $m - a - b$ rows left to read. If this query is the first of at most $d \geq 1$ remaining queries, either child can contain at most

$$C = 2^{d-1}$$

cases when distinct cases need distinct repair leaves. The local cardinality premise $m \leq 2C$ is checked before a root is investigated. An overfull support cannot fit the remaining binary depth, even before any rows of the proposed query are read.

The initial search representation contains (a, b, m) but omits C. This can lose a distinction relevant to the search itself. For example, $(a, b, m) = (0, 3, 4)$ cannot have a completion whose two branch sizes are at most two. It can have a completion with branch sizes $(1, 3)$ when the allowed child capacity is four. The read prefix is the same. The permitted future investigation is different.

Proposition 14.1 (Feature sufficiency for a search decision)**.** *Let $\mathcal{P}$ be a finite set of search contexts, let $r : \mathcal{P} \to Z$ be their available feature map, and let $\ell : \mathcal{P} \to \{0, 1\}$ be the required decision label. A deterministic rule $f : Z \to \{0, 1\}$ satisfying $fr = \ell$ exists if and only if ℓ is constant on each nonempty fibre of r. A pair in one fibre with different labels rules out every such rule, irrespective of its program length.*

Proof. If f exists, equal features have the same output under f and hence the same label. Conversely, assign to each attained feature value its common label. This is well defined when the fibre condition holds. Values outside the image of r can be assigned arbitrarily. A conflicting pair violates necessity for every possible f. □

The criterion is elementary factorization, not a new general learning theorem. Its role is to distinguish insufficient information from inadequate search effort. In the finite construction, exhaustive search for a larger formula over (a, b, m) cannot fix the conflict. The supplied feature registry contains the remaining-depth capacity C. Exposing it resolves this particular obstruction. Computing C from an already known depth is not a physical experiment and does not identify the next episode's hidden defect.

Counterexample 14.2 (The same prefix requires different rejection decisions)**.** For the two contexts $(a, b, m, C) = (0, 3, 4, 2)$ and $(0, 3, 4, 4)$, the initial feature vector (a, b, m) is identical. In the first, three cases already occupy a child with capacity two. In the second, assigning the remaining case to the other child gives a capacity-feasible completion. Thus the predicate "no capacity-feasible completion exists" cannot be computed from the initial features alone. Adding the capacity makes this specific decision representable. It does not establish that the actual query library can distinguish every resulting child.

This is a small instance of representational revision inside a discovery algorithm. The target of learning is not only a model of the external task. It is a model of which search continuations remain possible under the compiler's own constraints. A richer representation is useful because a named counterexample requires it, rather than because extra variables are assumed to improve intelligence in general.

15 Composing a rule from primitive syntax

The enlarged feature vector is (a, b, m, C). The reference language has six integer atoms, $a, b, m, C, 0, 1$, three binary arithmetic operations, max, min, and addition, and the five comparisons $>, <, \geq, \leq, =$. A candidate is a Boolean comparison with at most five syntax nodes. No general program execution is admitted. The interpreter evaluates only these finite arithmetic trees.

Integer atoms have size one. A binary arithmetic expression has size three. A comparison of two atoms has size three, while a comparison with one arithmetic child has size five. For commutative arithmetic, operand pairs are generated in one fixed order. This produces 180 size-three formulas and 3,780 size-five formulas,

or 3,960 candidates in all. Different syntaxes can express the same function. No semantic deduplication is assumed in these counts.

This is a bounded instance of syntax-guided synthesis, in which the candidate language and a semantic requirement are specified separately [19]. Counterexamples guide the choice of candidate. Learning invariants from examples, counterexamples, and implication constraints is also an established research direction [20]. The present learner does not implement the full ICE framework and makes no claim to discover general loop invariants. It learns one rejection expression under a supplied search contract.

15.1 What the teacher knows

Development uses the 198 count-prefix contexts with $C \in \{1, 2, 4\}$, $2 \le m \le 2C$, and $a, b \ge 0$, $a + b \le m$. For each context, the teacher enumerates the possible numbers of report-one cases among the unread rows. It labels the context as reject precisely when no such completion can put at most C cases in either branch. The teacher therefore solves a small finite feasibility problem. It is not reading a label generated by the final formula.

After each returned counterexample, the learner retains that context and its label. It selects the earliest formula in the fixed syntax enumeration that agrees with every retained example. The finite teacher either returns another disagreement or checks the formula against the complete developmental domain. The definition of that domain, the generation of its labels, and all rejected candidates are part of the development account.

Theorem 15.1 (Finite counterexample refinement within a grammar)**.** *Let $\mathcal{G}$ be a finite nonempty set of deterministic candidate programs, let $\mathcal{P}$ be a finite domain, and let ℓ be its specified labeling. Suppose a teacher either returns a point where the proposed program differs from ℓ or certifies agreement on all of $\mathcal{P}$. A learner that proposes only candidates consistent with every retained counterexample rejects each proposed program at most once. It returns a correct program on $\mathcal{P}$, or certifies that none in $\mathcal{G}$ meets the labeling, after at most $|\mathcal{G}|$ teacher calls. With a fixed total order and the earliest-consistent selection rule, any returned program is the earliest one that agrees on the entire domain.*

Proof. A counterexample removes the current program because it must be included in the constraints for every later proposal. Hence that program cannot be proposed again. A program agreeing with ℓ on all of $\mathcal{P}$ is never removed by a valid counterexample. There are only $|\mathcal{G}|$ candidates, so after at most that many distinct proposals the teacher has accepted one or all candidates are excluded. If a returned program were preceded by a globally correct program in the fixed order, the earlier program would remain consistent and would have been selected instead. This proves the ordering assertion. □

The result concerns the declared finite domain. It is not a generalization bound and supplies no efficiency guarantee for obtaining the teacher's answers. A teacher timeout is not a certificate. If the grammar does not contain a suitable program, enlarging the grammar changes the search problem and its cost.

15.2 The synthesized expression and its provenance

The reference learner encounters three disagreements. Its first candidate, $a > a$, fails on a prefix with too many ones. The next proposed candidate, $b > a$, rejects a prefix that can still be completed. The third, $b > C$, misses a prefix with too many zeros. The fourth teacher proposal is

$$\rho(a, b, m, C) = \mathbf{1}\{\max(a, b) > C\}. \tag{8}$$

The formula uses five syntax nodes and passes all 198 developmental contexts. The enumeration generates all 3,960 AST roots and considers 190 of them, including candidates rejected directly by retained examples.

There are 282 retained-example checks, 210 full-domain teacher comparisons, 344 completion-count checks to construct the labels, and 1,898 interpreted syntax-node visits in the learner's recorded comparisons. The full grammar has 19,440 literal node occurrences. Shared in-memory subexpressions can reduce actual storage, so this last quantity is not a measured byte count.

These numbers describe an exact finite synthesis, not a statistical training cohort. The formula is serialized and frozen before evaluation on larger supports, additional capacities, and changed row and query orders. No new episode's defect index or solved controller is placed in the formula. It contains only the reusable relation between two observed prefix counts and remaining capacity.

The learner discovers the composition of supplied primitives. It does not invent addition, maximum, the teacher, the resource model, or the eventual safety proof. This boundary is important: enumerating a finite grammar is more expressive than choosing among the earlier three named compiler policies, but it remains a bounded search space chosen in advance.

16 When an unfinished read can close a search branch

A finite fit is insufficient grounds for deleting possibilities from a later search. Pruning is stronger than proposing. A bad proposal can be rejected by a finished-witness checker. A bad pruning rule can remove the only successful controller and leave nothing for that checker to inspect. We therefore attach the generated expression to a separate scope argument.

Proposition 16.1 (Exact obstruction to a capacity-feasible completion)**.** *Let C, m, a, b be integers with $C \geq 1$, $2 \leq m \leq 2C$, $a, b \geq 0$, and $a + b \leq m$. A binary completion of the remaining $m - a - b$ reports with both final branch counts at most C exists if and only if $\max(a, b) \leq C$. Consequently, a prefix satisfying $\max(a, b) > C$ can be rejected before its remaining rows are read, without rejecting any completion that satisfies these two capacity constraints.*

Proof. Counts do not decrease as rows are read, so $a > C$ or $b > C$ makes a feasible completion impossible. Conversely, suppose $a, b \leq C$. Let $r = m - a - b$. There are $C - a$ available places in the zero branch and $C - b$ in the one branch. Both are nonnegative and their sum is $2C - a - b \geq r$. Distribute the r remaining cases between these two capacities, filling either branch up to its capacity and putting any remainder in the other. This gives the required completion. The argument concerns possible report assignments, not whether an installed query has that assignment. □

The theorem applies to arbitrary positive integer capacities. The connection to query depth uses $C = 2^{d-1}$. An actual search separately checks that both observed branches are nonempty, since a redundant unary query cannot be part of a controller attaining the minimal literal storage. It also verifies all later subtrees. Capacity feasibility is necessary, not sufficient, for actual repair solvability.

16.1 A deliberately narrow certificate checker

The executable gate accepts only ASTs that normalize by swapping a comparison's sides or maximum's arguments to $\max(a, b) > C$. It rejects other finite-consistent formulas rather than claim to prove arbitrary arithmetic identities. Thus the universal justification is the preceding prose proof, coupled to a project-local structural checker and a runtime scope guard. This is not a Lean certificate or an independent theorem-proving service.

Counterexample 16.2 (Agreement on training prefixes does not establish scope)**.** The formula $\max(a, b) > \min(C, 4)$ agrees with (8) on every developmental context because every developmental capacity is at most four. At $(a, b, m, C) = (0, 5, 16, 8)$ it rejects, while a completion with final branch sizes $(8, 8)$ exists. A

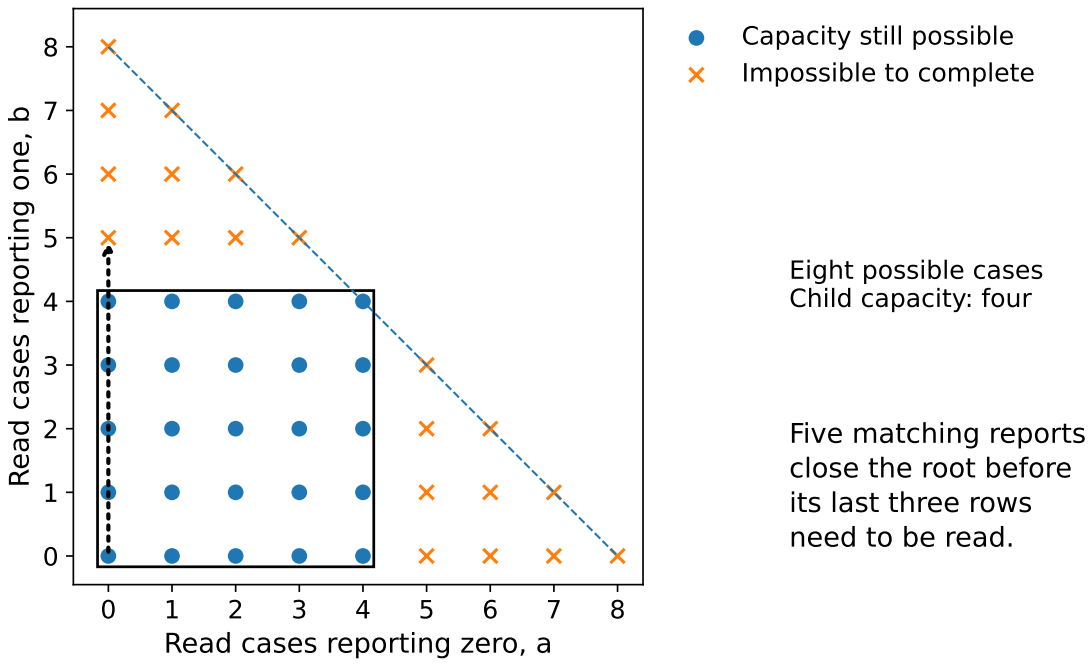


Figure 7: An exact count-prefix model with eight cases and child capacity four. A prefix outside the square $a, b \le 4$ cannot be completed within that capacity. Prefixes also satisfy $a + b \le 8$. Surviving the count test does not guarantee that the installed query library can separate the children. The dotted arrow is one unfinished read that can stop immediately after its fifth equal report.

larger input has not merely changed the frequency of familiar cases. It has reached a scope on which the bounded-cap formula was never justified. The structural gate does not accept that formula.

This counterexample also separates candidate selection from admissibility. A learner may prefer a formula for reasons of fit or cost. The compiler may install it as a pruning rule only under the corresponding proof obligation. Changing the rule's semantics, the available features, or the repair-success contract requires revisiting that obligation.

16.2 The same search with fewer model-row reads

The reference compiler first checks the immutable, unit-cost, binary-query and singleton-repair scope of Theorem 11.1. It then conducts an ordered depth-first feasibility search at the information lower depth $d_0 = \lceil \log_2 |B_0| \rceil$. The full-scan version reads every current row of a candidate query before rejecting an overfull child. The streaming version consults the certified prefix rule after each read. Both use the same query order, report order, remaining depth, and stopping rules. If a root has capacity-feasible children but a child cannot be completed using the actual library, the search backtracks to the next root.

A successful controller is separately checked and attains both lower bounds. When no controller exists at d_0, a requested horizon equal to d_0 permits a genuine finite infeasibility result. A longer requested horizon instead invokes the unchanged general compiler. Failure of the initial scope guard also invokes that compiler. Exhaustion of a model-read allowance returns incomplete, never infeasible merely because the allowance ran out.

Theorem 16.3 (Semantics-preserving streaming pruning)**.** *Under the checked binary immutable singleton-repair scope, fixed query and row orders, and identical search and fallback rules described above, replacing*

full root scans by the certified predicate (8) *returns the same first successful lower-depth controller, or the same exhaustive failure at that depth. With unlimited completion of any required common fallback, both compilers return the same protected frontier. The streaming version uses no more transition/report model-row reads, and the difference is exactly the number of unread suffix rows skipped at rejected roots. Scope checks and final witness verification are included identically in both read accounts.*

Within a root scan, after a prior prefix has passed, the predicate can be evaluated by comparing only the newly incremented count with C. This lowering preserves every rejection decision.

Proof. By Proposition 16.1, every root rejected early would be rejected after its full scan. A root not rejected early has its complete rows read and produces the same two child supports in the same order. Induction on remaining depth therefore gives the same child searches, backtracking choices, first completed controller, and lower-depth failure. Nonempty binary splits strictly reduce support, and the depth decreases, so the finite search terminates. Any successful tree can omit redundant unary roots without losing case information in this immutable unit-cost scope. The search consequently covers all relevant lower-depth controllers. The checked lower bounds justify the same optimal frontier on success. Identical failure conditions invoke the same exact fallback on the same problem.

At every visited root, either the same rows are read or one suffix of rows that cannot affect the rejection decision is omitted. All descendant supports actually entered are identical. Summing omitted suffix lengths gives the stated read difference. This excludes timeouts, under which the methods can finish at different allowances but neither asserts false infeasibility.

For the lowering, both prior counts are at most C, since otherwise the scan would already have stopped. One report increments exactly one count. The unchanged count remains at most C, so the maximum exceeds C exactly when the incremented count does. This implication requires the stated loop invariant. □

The read comparison isolates one alteration to one compiler. It is not a claim that every new search order beats the earlier balanced proposer, nor that fewer table reads imply shorter processor runtime. Interpreting a five-node formula at each prefix adds work. The supplied lowering replaces that interpretation by one integer comparison, but it too has a cost. The code records both versions and checks that they choose identical trees.

Counterexample 16.4 (Local capacity without a completing query library)**.** Four cases admit singleton repairs. The only available query divides them into $\{0, 1\}$ and $\{2, 3\}$ and leaves the state unchanged. At depth two, the root satisfies the $2/2$ capacity test. Repeating the only query cannot separate either pair, so no successful depth-two tree exists. Accepting the first capacity-feasible root as a finished certificate would be wrong. The streaming compiler searches its children, records the failure, and concludes infeasibility only after exhausting the actual library.

17 Pruning depends on what counts as a repair

The singleton scope is not cosmetic. A repair procedure need not determine the hidden case if one action succeeds on several cases. A change to the repair language can therefore invalidate the interpretation of raw case counts as required leaves, even when the experiment model is unchanged.

Let B be a finite nonempty immutable-case support and let $G_r \subseteq B$ be each repair's success set restricted to it. Define $\kappa(B)$ to be the smallest number of repair-success sets whose union covers B. Set $\kappa(B) = \infty$ when no such cover exists. This number concerns the repaired objective, not the number of hidden states. Computing it can itself be expensive.

Proposition 17.1 (Repair-relative leaf requirements)**.** *For finite immutable cases with binary deterministic queries, any certain successful diagnostic tree of query depth at most d has at least $\kappa(B)$ nonempty repair*

leaves. Consequently $\kappa(B) \le 2^d$ is necessary for success, and literal tree storage is at least $2\kappa(B) - 1$. These conditions are not sufficient for success under a specified query library.

Proof. Every initial case reaches one repair leaf. The set of cases reaching that leaf is contained in its repair's success set. The success sets of the attained leaves therefore cover B, so the number of leaves is at least $\kappa(B)$. A binary tree of depth d has at most 2^d leaves. The edge-count argument from Theorem 11.1 gives at least twice the number of nonempty leaves minus one literal records, with any unary nodes adding further storage. The preceding one-query example has $\kappa(B) = 4$ and $d = 2$, satisfying both numerical lower bounds while lacking a completing tree. Thus the necessary conditions are not sufficient. □

For singleton repairs, $\kappa(B) = |B|$, recovering the earlier argument. With a universal repair, $\kappa(B) = 1$ even when many cases remain unknown. The executable streaming path retains the easier singleton guard rather than quietly substituting a hard cover computation whose cost would be uncharged. The broader proposition indicates how a different pruning contract could be developed, not a hidden extension of the present code.

Counterexample 17.2 (A successful repair language can make identification unnecessary)**.** Keep the four-case query table but add one unit-cost repair that succeeds in every case. The exact optimum is now one repair call and one stored record, with no queries. A compiler that carries over the old "four cases require four distinct leaves" assumption can discard this optimum. The reference guard rejects the changed singleton scope and the general compiler returns the zero-query solution. The old theorem remains true. Its premise no longer describes the new language.

This is the preservation problem in recursive revision. A learned optimization acquires authority from a contract. Subsequent language growth can change that contract. Keeping the old formula while discarding its premise is not improvement. A retained shortcut should therefore carry its feature definition, semantic assumptions, recognized proof form, and fallback behavior together.

18 What the synthesized compiler actually gains

The learner freezes (8) after the small development domain. It is then used on the hidden-cell query family through 64 cases. The primary comparison is between the full-scan ordered search and the same ordered search with prefix pruning. It therefore isolates the effect of the generated rule. The earlier balanced wrapper is retained as a separate comparison, not silently treated as identical to this full-scan baseline.

Cases	Full-scan reads	Streaming reads	Rows avoided	Added comparisons
2	12	12	0	2
4	66	65	1	25
8	308	277	31	141
16	1,384	1,129	255	681
32	6,192	4,641	1,551	3,137
64	27,744	19,457	8,287	14,209

All listed compilers include the semantic scope guard and final witness check. The common input-shape validation is separately recorded and identical. For example, the 64-case query table has 70 queries and its shared shape check consumes 4,480 model rows for each method before the reported compilation allowance. The final controllers have the same literal trees in the primary comparison. Their execution and storage costs are unchanged: seven online units and 127 stored records in the 64-case case.

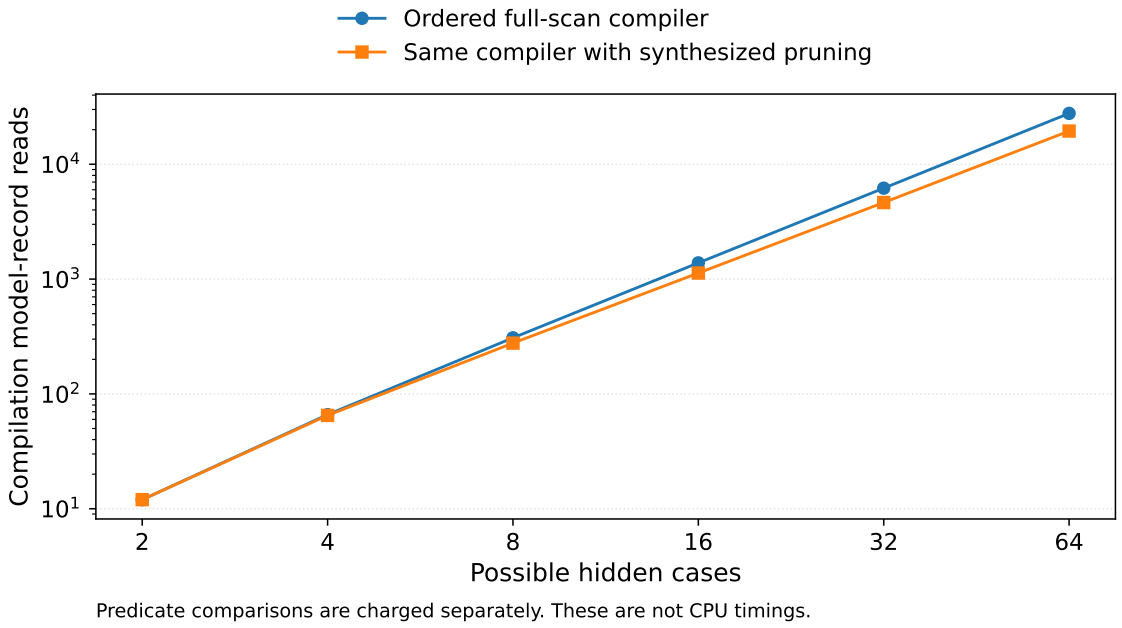


Figure 8: Transition/report model-record reads for the same ordered compiler before and after installing the synthesized prefix rule. Scope and final witness checks are charged in both curves. Common input-shape validation is separately reported. The vertical axis is logarithmic. The plot is an exact finite cost comparison, not a timing ratio or evidence of a broad neural-learning gain.

18.1 A resource improvement is not a universal speedup

In the interpreted version, each prefix evaluates five AST nodes. The checked lowering uses one integer comparison instead. Relative to a baseline that already performs the complete-root capacity test, a conservative extra-work account charges every new prefix comparison without crediting any removed end-of-scan check. At 64 cases this adds 14,209 comparisons while avoiding 8,287 model rows. At two cases no row is avoided and the new comparisons are pure overhead.

If a model-row read has declared price $w > 0$ and a new comparison has price one, the additional per-use saving under this conservative two-unit account is

$$\Delta(w) = w\,(R_{\text{full}} - R_{\text{stream}}) - N_{\text{compare}}.$$

For the 64-case row it is positive when $w > 14209/8287$. At $w = 1$ it is negative. The price is a modeling choice, not a measured statement about Python or a physical machine. Interpreted AST visits, installed code, grammar construction, labels, and development search remain additional recorded coordinates. No weighted sum is presented as a universal cost of intelligence.

If development has total charged cost D in a chosen common accounting unit and the same future use has saving $\Delta(w) > 0$, N repetitions repay that cost only when $N\Delta(w) > D$. This is the amortization argument already used earlier, applied to the learned compiler rule. It does not authorize ignoring development because deployment might eventually repeat often.

18.2 Transport and negative controls

The formula contains counts and a remaining-depth capacity rather than cell names or query identifiers. Renaming cases and flipping binary report meanings preserve the capacity predicate. A complete search

uses the model rows to determine its actual branches, and its witness is checked anew. Reordering queries can change which equally optimal controller is encountered first. Reordering rows can change how early an impossible prefix becomes visible. Correctness is preserved under the certified scope, but the number of saved reads need not be invariant.

A four-case equality-only library provides another boundary. It has no controller at the information lower depth, so a larger requested horizon invokes the general compiler. The new method can spend preliminary work before that fallback. On a changed language with a universal repair, the semantic guard rejects the pruning scope and the general optimum is recovered. These are specified behaviors, not successes relabeled after observing the results.

18.3 The precise recursive claim

The first construction improved a diagnostic controller. The second selected a better compiler from named policies. The present construction synthesizes and installs an internal decision rule that changes how future controllers are discovered. It also records a representational obstruction within that compiler and a permitted feature refinement that removes it. This is a bounded inductive program-synthesis step, not neural training or a renamed cache of solved episodes.

The construction does not demonstrate that the grammar enumerator becomes better at inventing later grammars, that the learner discovers its own theorem checker, or that its task family becomes generally intelligent. The primitive operations, teacher, feature registry, scope proof, and search skeleton are substantial prior structure. The gain nevertheless occurs at the claimed location: future searches omit work because of a reusable expression composed from evidence, and the conditions under which that omission remains justified are checked rather than presumed.

Part V

Improving how discovery rules are acquired

19 The next retained object is a search representation

Part IV changes a decision inside the diagnostic compiler. Its counterexample-guided learner still traverses a fixed list of arithmetic candidates. A further change is possible without pretending that the learner invents its own computational foundations: it can acquire a reusable representation of that candidate space, then use the representation to reduce the work of synthesizing later rules.

The new object is neither a completed diagnostic tree nor the target expression $\max(a, b) > C$. It is a library of checked equations and an ordered map from grammar expressions to earlier equivalent representatives. Learning the library does not require the labels of a later synthesis task. Once the representation has been constructed and validated, subsequent tasks still begin with empty task-specific counterexample sets.

This direction has direct precedents. Equality saturation represents equivalent expressions and supports program optimization and synthesis [21]. Ruler automatically infers rewrite rules and uses equality saturation within that inference process [22]. The present implementation is intentionally smaller. It uses a typed arithmetic language, finite proposals, a local derivation checker, and ordered enumeration rather than an e-graph. It does not claim a new rewrite-inference paradigm or superiority to those systems. Its purpose is to make one additional level of the improvement process executable and to connect the acquired search representation to the paper's unchanged semantic and resource contracts.

19.1 Four objects and two meanings of success

Let $\mathcal{G} = (p_0, \ldots, p_{N-1})$ be a finite ordered grammar of pure, total expressions. A target task supplies contexts $z \in D$ and desired values $f(z)$. An ordinary learner uses counterexamples to find the first grammar expression that agrees with all of D. A separate equational learner observes expressions on a development probe set and proposes equations. A checker admits only equations supported by a derivation in the declared arithmetic theory.

There are therefore two distinct successes. A task candidate may agree with a complete finite target domain while remaining unjustified outside it. An equational certificate may establish agreement for every integer substitution, even though it says nothing about whether the resulting program solves a particular task. The latter can safely reduce the former search, but cannot manufacture its missing target evidence.

20 Learning equations without confusing probes with proofs

The reference language uses exact integers, the total operations +, min, max, and the comparisons $<$, $\leq$, $>$, $\geq$, and $=$. A finite schema language contains 135 terms built from variables x, y, small integer literals, Boolean literals, and at most one binary operation. The proposer evaluates those schemas on the nine contexts $(x, y) \in \{1, 2, 3\}^2$.

Two schemas with the same probe vector become a proposed equation, not an accepted rule. This produces 1,068 proposals. The supplied proof procedure proves 336 using integer arithmetic identities. The other 732 are not admitted. Failure of that limited procedure is not a theorem that an equation is false.

The derivation interface has six elementary rule kinds: commutativity where valid, addition of zero, idempotence of minimum and maximum, reversal of an order comparison, reflexive comparison, and exact evaluation of literal constants. It also checks replacement at an explicit subtree position. A proof of an

equation contains two derivations ending in the same expression. A separate replayer checks every position, side condition, and endpoint. This is a project-local equational checker, not a proof-assistant kernel or a general arithmetic decision procedure.

Counterexample 20.1 (A probe-equivalent rule that fails on a new input)**.** On the development inputs $x \in \{1, 2, 3\}$, the terms $\max(x, 1)$ and x agree. At $x = 0$ they differ. The proposed equation is not admitted by the proof checker. By contrast, $\max(x, 0) = x$ holds on all nonnegative integers but not on all integers. A checker using the stronger integer-wide contract may fail to admit a rule that would be valid under a narrower natural-number contract. Rejection, counterexample, and scope restriction remain different outcomes.

20.1 What the learner contributes and what it is given

The proposer generates the equations from evaluations rather than receiving the final equation list. The library is serialized before the later task targets are evaluated. The primitive syntax, probe schedule, arithmetic axioms, proof search, and normalization engine remain supplied. Their presence does not negate the acquired representation. It does prevent describing the construction as discovery of integer arithmetic or autonomous improvement of the equational learner itself.

The complete accepted library occupies 69,518 bytes in its stated compact JSON serialization. This is a serialization measurement, not an estimate of an optimal theory description. It contains redundant equations. A supplied full normalizer is retained as a stronger comparison, so the results do not depend on comparing only against deliberately repeated syntax.

21 A certified quotient preserves the investigation's evidence

Write $[\![p]\!](z)$ for an expression's value in context z. Let $n(p)$ be a deterministic representation produced by finitely many checked substitutions of valid equations. We only require

$$n(p) = n(q) \quad \Longrightarrow \quad [\![p]\!](z) = [\![q]\!](z) \quad \text{for every admitted } z.$$

The reverse implication is unnecessary. Distinct normal forms may denote the same function. Missing such an equivalence wastes an optimization opportunity but does not remove a valid candidate.

Theorem 21.1 (Sound ordered quotienting from checked equations)**.** *Assume the expression language is pure and total, every admitted schema equation is valid for all well-typed substitutions in the common semantic domain, and each normalization terminates after valid contextual replacements. For each p_i, keep its earliest grammar representative*

$$\rho(i) = \min\{j : n(p_j) = n(p_i)\}.$$

Then $\rho(i) \le i$, p_i and $p_{\rho(i)}$ agree on every admitted context, and every pointwise specification realizable in $\mathcal{G}$ is realizable in its retained representatives. No confluence or complete semantic-equivalence decision procedure is required.

Proof. A schema identity remains true under a well-typed substitution. Purity and totality make replacement inside any expression context preserve its value. Induction over a finite derivation gives equality between an expression and its normalized result. Two expressions with the same result therefore have equal values by transitivity. The defining set for $\rho(i)$ contains i, giving existence and $\rho(i) \le i$. Replacing a satisfying expression by its representative preserves every pointwise requirement. The argument uses the soundness of each actual derivation, not uniqueness or completeness of normalization. □

In the reference implementation, a successful rewrite must decrease a size-and-structural-order measure. Rewriting introduces no free variables or literals outside the finite input-and-library vocabulary, and never

increases size. Only finitely many terms lie below the initial size over that vocabulary, so there is no infinite decreasing rewrite sequence. The index keeps the original earliest AST, not the normalized AST. This preserves the reference learner's output convention and its program interpreter cost on every teacher proposal.

Theorem 21.2 (Preservation of ordered counterexample-guided acquisition)**.** *Consider a learner that scans a finite ordered grammar, retains every counterexample, tests candidates against those examples in a fixed order, and submits only consistent candidates to a deterministic teacher. The teacher returns the first disagreement in a fixed complete finite context order, or accepts. Under Theorem 21.1, scanning only the earliest representatives produces exactly the same submitted ASTs, counterexamples, and final accepted AST or grammar-exhaustion outcome. With the original AST interpreter retained, it performs no more retained-example evaluations or interpreter-node visits. Its teacher calls and teacher-evaluation work are unchanged. Quotient construction, storage, and dispatch costs are separate.*

Proof. Consider a removed expression p_i and its earlier representative p_j. If the reference learner accepted p_j, it never reaches p_i. Otherwise a retained example refuted p_j, either before its teacher call or as a result of that call. That example is still retained when p_i is reached. Semantic equality means the same example refutes p_i, so the reference learner never submits p_i to the teacher. Deleting that unsuccessful consistency check leaves the sequence of proposals unchanged. Induction over the ordered scan gives the same candidate AST at every teacher call. The deterministic teacher then supplies the same first counterexample or acceptance. Removed candidates contribute only nonnegative evaluation work, while submitted expressions and their evaluations remain identical. If the finite grammar is exhausted, both learners have rejected the same possible behaviors. □

The theorem concerns an extensional target and a fixed candidate order. A teacher that accepts based on source identity, or a learner that forgets old counterexamples, does not satisfy it. Nor does equality of return values make two candidates equal in execution cost, memory use, or causal side effects. To preserve a different resource-constrained objective, one must keep appropriate nondominated representatives or include those observables in the equivalence contract. The reference deliberately preserves the first original AST rather than selecting an uncharged cheaper replacement.

21.1 Cold reacquisition and additional finite task families

The 3,960-expression grammar reduces to 1,362 retained representatives. Reacquiring the original capacity predicate starts with no task counterexamples and no installed copy of the answer. The original scan considers 190 candidates, while the quotient scan considers 76. Both submit the same four candidate ASTs, receive the same three counterexamples, and return the identical five-node expression. Retained-example checks fall from 282 to 112. Total interpreter-node visits in that task fall from 1,898 to 1,376. Teacher comparisons remain 210.

The frozen quotient is next used for nine finite targets involving sums, extrema, comparisons, and equality, followed by a parity target that is not expressible on its complete declared domain. These are changed target functions in the same arithmetic language, not a statistical sample of unrelated research domains. The first nine use all 256 contexts in $\{0, 1, 3, 5\}^4$. The parity control uses all contexts in $\{0, 1, 2, 3\}^4$, so more than one even value is present.

Finite target	Original interpreter visits	Quotient interpreter visits
Sum exceeds capacity	4,611	3,517
Minimum below threshold	7,762	5,553
Maximum equals a value	26,407	10,116
Sum equals a value	29,357	12,311
Parity outside the grammar	54,045	18,728
Complete ten-target batch	196,384	85,162

Every submitted candidate and counterexample trace agrees exactly between the two learners. The parity target produces grammar exhaustion in both. A bounded grammar's failure is not a proof that the target function is uncomputable.

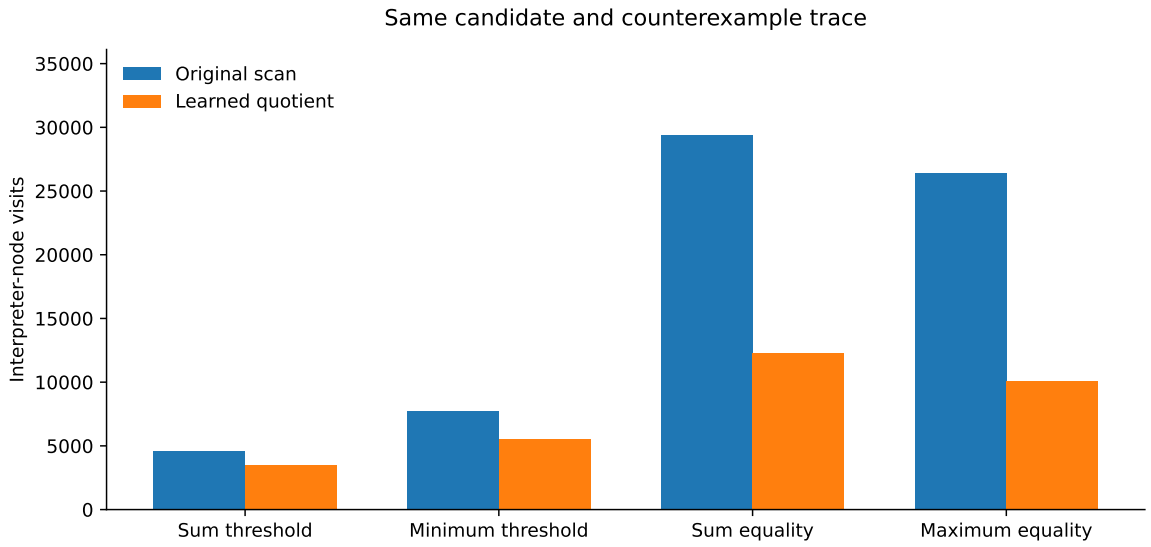


Figure 9: Repeated task-synthesis evaluations are reduced while the teacher trace remains fixed. Values are exact interpreter-node counts in the stated finite cases. Equation discovery, checking, quotient construction, and stored library costs are not included in these bars and are reported separately.

21.2 The stronger normalizer and the cost of preparing the shortcut

A supplied full arithmetic normalizer retains 1,352 representatives, ten fewer than the learned library. It therefore removes at least as much semantic redundancy under this implementation's normal-form relation. The learned library is not claimed to outperform that stronger prior. This control identifies the result as automatic acquisition of a useful search representation from the supplied reasoning substrate, not a new best-in-class synthesis algorithm.

The library-development ledger records 1,215 probe evaluations, 3,519 probe interpreter-node visits, 1,068 proposed equations, and 584 successful checked axiom steps. Proof search also attempts proposals it cannot establish. Applying the admitted library to the whole grammar costs 19,440 normalizer-node visits and 518,140 schema-match attempts, with 4,798 successful replacements. Grammar generation, proof search, bookkeeping, and serialization remain distinct charged phases. These event counts do not encode every Python instruction.

Let C_{prepare} be the complete preparation cost in a declared common resource model, including failed proposals and certificate checking, and let d_t be the subsequent saving on task t after including dispatch. Reuse is beneficial in that model exactly when

$$\sum_{t=1}^{T} d_t > C_{\text{prepare}}.$$

This is the earlier amortization identity applied to acquisition itself. The reference does not convert unlike counters into a universal wall-clock claim. Even one setup coordinate, 518,140 schema matches, exceeds the 522 interpreter-node visits saved in one cold capacity-rule run. A short workload or an expensive matcher can make the full modification undesirable.

21.3 A cached proof representation has a scope

The quotient records the grammar's contents, order, and declared integer semantics. Loading a cache requires checking those bindings and reconstructing its representative map from the admitted proofs. Adding syntax or changing order requires rebuilding the index. Extending the grammar can still reuse an equation library when every substituted operation is pure, total, and retains the relevant types and meanings.

Counterexample 21.3 (The same operator spelling with changed semantics)**.** The identity $x+0 = x$ is valid for ordinary addition. Suppose a revised interpreter uses the same printed symbol but executes $a \oplus b = a + b + 1$. Then $x \oplus 0 \neq x$. A cache certified for the former interpreter cannot be imported merely because its expression strings still parse. A semantic identifier helps detect ordinary version mismatch, but an identifier falsely attached to a changed implementation is not itself proof of faithful execution.

22 A repair-relative certificate that can be updated cheaply

The singleton pruning result remains valid within its original scope. To move beyond that scope, the compiler must account for what a repair can accomplish, not merely count hidden cases. An exact minimum repair cover can be expensive. A feasible lower-bound witness is often enough to exclude an impossible branch.

Let X be a finite immutable state set and let $S_r \subseteq X$ be the success set of repair r. Choose nonnegative rational weights w_x satisfying

$$\sum_{x \in S_r} w_x \leq 1 \quad \text{for every permitted repair } r. \tag{9}$$

This is a feasible dual weight assignment for unweighted set cover. We use only its elementary weak-duality property, not an assumption that the best assignment has been found [23].

Proposition 22.1 (Repair-relative weight and prefix obstruction)**.** *Suppose a total immutable deterministic binary diagnostic tree succeeds on nonempty support B and has L reachable leaves and maximum query depth h. Under* (9)*,*

$$\sum_{x \in B} w_x \leq L \leq 2^h.$$

At a candidate query root with $h \geq 1$ remaining query levels, if the cases already read into either outcome branch have weight greater than 2^{h-1}, no assignment of its unread reports can make that branch repairable within the remaining depth. This implication is valid even when one repair succeeds in several states.

Proof. Immutable deterministic execution partitions B into its nonempty reachable leaf supports. Each leaf support is contained in the success set of its chosen repair. Nonnegativity and (9) give weight at most one for

every leaf, so summing gives the first inequality. A binary tree of depth at most h has at most 2^h leaves, by induction on h. At a candidate root, each child has at most $h - 1$ query levels. Unread cases can only add nonnegative weight to an outcome branch, so a prefix already exceeding its leaf capacity cannot be rescued by any completion. □

Theorem 22.2 (Streaming search with a checked repair-weight witness)**.** *Fix total immutable binary unit-cost experiments, unit-cost repairs with arbitrary fixed success sets, an ordered initial support, a repair order, an experiment order, a finite query horizon, and a weight vector satisfying* (9)*. Consider depth-first search that first checks for a common successful repair and otherwise explores informative query roots in that order, rejecting child supports whose weight exceeds their remaining binary leaf capacity. Replacing each complete root scan by a scan that stops on the first overweight prefix preserves the exact first returned tree or finite-horizon infeasibility result. It never increases model-record reads during search. These statements do not establish a Pareto-optimal storage frontier or free weight arithmetic.*

Proof. Proposition 22.1 shows that an overweight support has no successful tree at its depth. A root stopped on an overweight prefix would still be rejected after a complete scan because all further contributions are nonnegative. Every root not stopped early is partitioned identically, with the same row and outcome orders. Induction over depth and the ordered root scan therefore yields identical first successful children and identical returned trees. A query with one reachable outcome leaves the immutable support unchanged, so deleting such a query never destroys the existence of a depth-bounded successful tree. The search over the remaining roots and repairs is finite and complete for existence. Early stopping deletes only unread suffixes of roots the complete scan also rejects, proving the read comparison. The final witness checker and weight-scope checker remain required and are charged separately. □

22.1 Constructing a feasible witness without solving the whole cover problem

A simple supplied constructor processes states in order. It raises the current state's weight by the smallest remaining slack among repairs containing that state, and updates those repair loads. An uncovered state may receive a finite weight of one. Every step preserves nonnegativity and all repair constraints. The result is checked independently by resumming every repair's weight. It can be a weak certificate. In particular, the all-zero assignment is always feasible and excludes nothing.

For example, on three cases with success sets $\{1, 2\}$, $\{2, 3\}$, and $\{1, 3\}$, the ordered greedy constructor returns total weight one. The feasible fractional assignment $w_1 = w_2 = w_3 = 1/2$ has total $3/2$ and proves that at least two leaves are required. Thus the supplied constructor does not optimize the bound. Both assignments are valid and neither creates a missing experiment.

The construction is not itself learned in this revision. It is a semantic extension showing how the compiler can continue when successful repair no longer means exact state identification. A learner may later propose a stronger weight assignment, but the same inequalities would still govern admission. The work of producing and checking those weights cannot be hidden inside one cheap predicate call.

22.2 Paired repairs change the task, not just its notation

Consider $2g$ possible hidden cases divided into g pairs, where g is a power of two. One repair succeeds on both members of each pair. The query language contains equality questions and bits of the pair index. The greedy weight constructor gives weight one to the first member of each pair and zero to the second, with total weight g. A depth $\log_2 g$ tree is both necessary and sufficient, so the diagnostic-plus-repair work is $1 + \log_2 g$ and a full binary witness uses $2g - 1$ records.

Hidden cases	Repair groups	Full scan reads	Streaming reads	Work / records
4	2	20	18	2 / 3
8	4	152	120	3 / 7
16	8	864	608	4 / 15
32	16	4,416	2,864	5 / 31

These columns count search reads only. For the 32-case problem, each method additionally pays 1,152 scope-table reads and 256 witness-checking reads, giving 5,824 versus 4,272 total model reads. Weight construction, repair-success-set checks, rational additions, and comparisons have their own ledger. The streaming search makes 2,941 capacity comparisons rather than 1,073. Both methods return the identical literal tree.

This is not a generalization score obtained by making the task easier. The comparison is between two discovery procedures on the same paired-repair task. The difference between that task and the earlier singleton task explains why the earlier cardinality certificate would be invalid here.

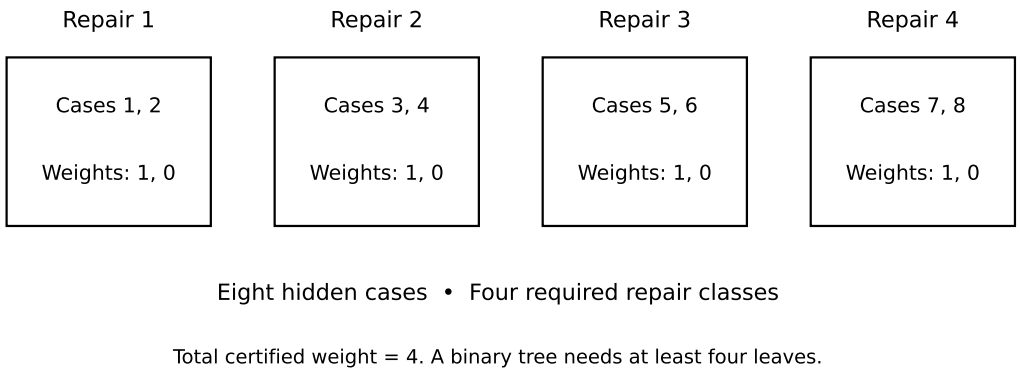


Figure 10: Eight hidden cases need only four repair distinctions when each repair succeeds on a pair. The displayed weights certify a four-leaf lower bound. They do not claim that every feasible weight assignment is optimal or that every query library can realize the bound.

Counterexample 22.3 (A newly admitted repair invalidates the old lower-bound certificate)**.** In an eight-case paired model, the original weights have total four. Add a repair that succeeds on every case. Its weight load is four, violating (9). The old certificate must be rejected. Recomputing a feasible assignment gives total one, and the compiler returns the universal repair with no query, work one, and one stored record. The success of that repair does not retrospectively validate the obsolete weight assignment.

A capacity-feasible prefix still does not guarantee that available experiments can distinguish the required repair classes. The implementation retains a model with only a constant report: its cover weights fit a nominal depth bound, but no successful tree exists. A lower-bound certificate is a reason to reject some candidates, not a generator of a missing experiment.

23 One chain from acquired equations to an improved investigator

The levels can now be connected without substituting one kind of evidence for another. The equational learner produces a checked library. The library prepares an ordered search representation. That representation

reduces the repeated evaluation work of acquiring the earlier pruning rule and other finite targets. The rule's separate scope certificate then permits its use in the diagnostic compiler. Each stage retains the predecessor's semantic comparison.

Proposition 23.1 (Composition of acquisition and compiler preservation)**.** *Assume the ordered quotient satisfies Theorem 21.2, the resulting task expression passes the unchanged compiler-scope checker, and the original and optimized compilers satisfy the common-domain premises of Theorem 16.3. If all stages complete, quotient-guided rule acquisition followed by optimized compilation returns the same learned expression and the same diagnostic controller as ordinary rule acquisition followed by the reference ordered compilation. In a common additive resource model, writing K for extra preparation cost, A, A' for the two acquisition costs, and D, D' for the two compilation costs, the composite saves work exactly when*

$$(A - A') + (D - D') > K.$$

The semantic conclusion does not imply the cost inequality, and a budget interruption does not imply infeasibility.

Proof. Theorem 21.2 gives the same acquired AST and teacher transcript. Passing the unchanged scope checker places that expression in the stated compiler-preservation theorem, which gives the same returned controller. The reference total is $A + D$ and the modified total is $K + A' + D'$. Subtracting gives the displayed necessary and sufficient cost condition. Neither semantic theorem bounds K or assigns exchange rates among distinct resource units. An interrupted finite search has not exhausted its candidates and therefore cannot use completeness to infer infeasibility. □

23.1 The recursive achievement and its next boundary

The retained change is now used to produce future discovery rules more economically in a specific evaluation coordinate. It is not merely a better diagnostic result. The learner's search representation has changed, while the pointwise target, oracle transcript, and final generated program remain fixed. This is a bounded meta-improvement with an explicit implementation and cost boundary.

It does not require that a useful self-improver invent every foundational component. It does require us to locate those foundations honestly. Here the schema proposer, primitive axioms, checking engine, and ordered-quotient construction are given. The acquired equation library and grammar quotient are generated. A supplied stronger normalizer can outperform the learned library, and a short workload can fail to repay its preparation cost. Extending the result to learned physical semantics, larger generated languages, uncertain arithmetic, or repeated improvements of the library learner itself requires additional work.

23.2 The boundary of the first edition

The finite construction now covers repair discovery, compiler revision, synthesized pruning, acquired search representations, and repair-relative scope extension. Part VI consolidates their dependency and resource contracts. A first edition need not claim that general intelligence follows or that the outermost learner has become unboundedly self-improving. Those are distinct research questions. The completed scope is the explicit finite chain and the evidence required to interpret it.

Part VI

Keeping a revision's guarantees applicable

24 A certificate belongs to a changing realization

A proof can remain mathematically correct while ceasing to justify an installed shortcut. The equation library was proved for pure, total integer expressions. The ordered quotient was constructed for one grammar order. A repair-weight certificate was checked against particular success sets. A resource comparison assigned costs to specified events. None of those objects is licensed to retain its old interpretation after every component bearing the same name has changed.

This is the final integration problem for the first edition. The earlier sections provide distinct certificates for the outputs of different stages. A continuing system must also identify which of those certificates remain applicable after a revision. The appropriate response is not to repeat every discovery from scratch, nor to regard an old passing record as timeless. It is to retain the actual dependencies, reject unsupported reuse, and recheck the affected obligations before relying on them again.

24.1 A declared dependency graph

Let an artifact state be $a = (a_1, \dots, a_d)$. Its components may include operation semantics, a grammar and its order, an equation library, experiment tables, repair-success sets, a returned controller, and a resource model. A finite directed acyclic graph has one vertex v for each claimed guarantee. Its incoming vertices $\mathrm{pa}(v)$ identify guarantees used as premises. Its direct dependency set $D_v \subseteq \{1, \dots, d\}$ identifies the artifact components used directly. Each vertex carries a witness w_v and a local verification rule.

The dependency declaration must be *semantically complete*: the meaning of the claim and the local rule is determined by the components in D_v and the stated parent claims. Hidden environment assumptions cannot be left outside this declaration merely because the implementation does not know how to check them. They can instead be recorded as explicit external premises. Verifier soundness, the interpretation of primitive arithmetic, and the faithfulness of the supplied model to an external process are not established by naming them in a registry.

Suppose a revision replaces a by a'. Mark a vertex directly stale whenever one of its declared component values differs. Let Δ be the union of those vertices and all their descendants in the dependency graph. Equivalently, in a topological order, mark v whenever

$$a|_{D_v} \neq a'|_{D_v} \quad \text{or} \quad \mathrm{pa}(v) \cap \Delta \neq \varnothing.$$

Here equality refers to the declared artifact value, not merely an unchanged version label. This procedure is conservative. A changed premise may leave the conclusion true, but the old evidence is not reused without a further argument.

Theorem 24.1 (Dependency-complete reuse across a revision)**.** *Assume a finite acyclic dependency graph, semantically complete direct dependencies, and sound local verification rules. Suppose the old witnesses establish every old claim from the declared external premises. After a revision, the old claims outside Δ remain established by their old witnesses. If every vertex in Δ is re-established in topological order using the revised artifact values and already established parent claims, all revised claims hold. If an affected vertex is not re-established, no dependent claim is justified by this reuse rule alone. These statements are conditional on the external premises applicable to the revised realization.*

Proof. A vertex outside Δ has unchanged direct inputs and no parent in Δ. Inducting through a topological order, its parents remain established with the same meanings, so dependency completeness leaves its old verification argument applicable. For an affected vertex, assume its parents have already been established in the revised realization. Soundness of the newly checked local rule then establishes its revised claim. A second induction yields the result when every affected vertex is re-established. Without that step, the rule has an unsupported premise and supplies no conclusion along that dependency. This is a limitation of the reuse argument, not a proof that the dependent claim is false. □

The argument is elementary induction over declared dependencies. Its role here is to connect the exact preservation results to their continued use. It does not compute the complete dependency graph from arbitrary source code. A dependency cycle requires an additional justified rule, such as an independently proved invariant. The finite replayer in this edition rejects an unsupported cycle rather than interpreting mutual assertions as evidence.

Counterexample 24.2 (An unchanged name hides a changed premise)**.** A stored claim asserts that two binary values u and v agree. Initially both are zero. A registry records only u as a dependency and continues to validate that its stored value is zero. Change v to one without changing u. The declared check still passes, but the claimed equality is false. Recording and replaying the incomplete registry cannot repair its omitted premise. Dependency completeness is a mathematical assumption, not a consequence of serialization or a checksum.

24.2 The scopes that actually have to be carried

The implemented constructions require different bindings. An equation witness includes the supported arithmetic semantics and the local derivation rules. A quotient includes the equation library and the full ordered grammar. Exact candidate-trace preservation additionally requires the extensional teacher, fixed context order, and retained counterexamples. The compiler rule has its own integer-prefix and singleton-repair scope. Weighted pruning instead checks every repair-success set and the nonnegative weights. A returned tree must be checked against the actual experiment effects, observations, repair meanings, and horizon.

Cost and evidence claims add other dependencies. A reduction in model reads is not a price-independent reduction in total work. An audit comparing a frozen candidate cannot be carried unchanged to a new candidate merely because the score field has the same name. Statistical independence and commitment order may depend on events outside the program's observation boundary. The manifest must state whether those conditions are guaranteed by the controlled construction, supplied by a trusted interface, or not established.

The reference certificate manager therefore has several outcomes. A claim is *valid* when its declared inputs match and its local evidence is replayed successfully. It is *stale* when a direct input changed, *blocked* when a required parent is unavailable or different, *rejected* when its local record fails, and *missing* when no witness was supplied. A timeout is not successful replay. Nor is a valid resource assessment necessarily favorable: the assessment may correctly report a negative saving.

The reference manager uses exact canonical snapshots for direct input comparisons and content identifiers for linking parent evidence. Those identifiers aid integrity checks. They do not establish who produced a witness or whether an external assumption is true. The manager and dependency registry are supplied components. This edition does not claim that the learner derived its own trusted verification base.

24.3 A repair-language change through the complete record

The first-edition demonstration checks the retained equation library, its ordered quotient, the capacity rule's supported form, a repair-weight witness, a returned weighted diagnostic tree, and a conditional resource assessment. The equation and diagnostic branches remain distinct. A valid capacity expression is not silently substituted for the weighted compiler's separate certificate.

In the 32-case paired-repair example, the accepted weight total is sixteen and the returned controller has work five and 31 records. Add a repair that succeeds in every case. The old weight certificate becomes stale, and direct rechecking confirms that it fails the new repair constraint. Its dependent controller and cost claims cannot rely on it through the old chain. The arithmetic equations and ordered grammar quotient remain applicable because their declared premises have not changed.

Rebuilding the weight witness gives total one. The compiler then returns a universal repair with work one and one record. The new controller is checked against the new repair language. The old cost comparison is deliberately not reissued: a claim about the work of producing the former controller does not become a claim about this different search simply because both controllers are valid. The appropriate repair of that claim is a new comparison under a stated workload.

Changing only the resource prices has the opposite pattern. The semantic certificates remain applicable, while the cost conclusion needs to be recomputed. Changing the grammar order invalidates the ordered quotient and its recorded acquisition lineage, not the independently proved arithmetic identities. These cases illustrate selective invalidation without equating it with a complete theory of arbitrary self-modification.

25 One resource ledger for the complete improvement

The manuscript's numerical comparisons concern several different units. A model-record read, an interpreter-node visit, an arithmetic comparison, a stored tree record, and a serialized byte are not interchangeable. The purpose of a complete ledger is to preserve those distinctions until a specific execution or economic model justifies a conversion.

Phase	Recorded object or event	What the count does not supply
Model and language acquisition	Supplied experiment tables, grammar, feature registry, primitive axioms	An executed calibration of an unknown physical process, or free discovery of those foundations.
Equation acquisition	1,068 proposals, 336 admitted equations, 584 successful checked axiom steps	Proof completeness, a generally optimal equation set, or uncharged failed attempts.
Search representation	3,960 grammar expressions, 1,362 representatives, 518,140 schema-match attempts	Free preparation, hardware-time units, or superiority to the 1,352-representative supplied normalizer.
Task-rule acquisition	1,898 versus 1,376 interpreter-node visits for the capacity rule	All preparation costs, or a different teacher transcript. The retained four proposals agree.
Compiler execution	Query-model reads, weight arithmetic, capacity comparisons, scope checks, witness replay	A universal conversion of avoided reads into elapsed time.
Installed information	Equation-library serialization of 69,518 bytes, original programs, quotient records	Minimal description length, physical peak memory, or a byte-free interpreter.
Online repair	Experiment and repair calls, plus complete stored tree records	Only the visited path's storage, or the cost of discovering the controller.
Revision management	Bound snapshots, witness checks, parent comparisons, and replacement evidence	Free invalidation detection or a trusted explanation of undeclared external assumptions.

The exact per-phase records remain in the source package. The table does not add unlike events into one total. In particular, retained storage is normally a capacity coordinate. Treating it as an additive expense

requires a stated storage-duration or installation convention. Peak memory combines by its own rule, not by automatically summing all sizes ever observed.

25.1 Which gains survive a change of resource prices?

Fix an additive ledger with $d \geq 1$ named event types. Let $K \in \mathbb{R}^d_{\geq 0}$ be extra setup, let $s_i \in \mathbb{R}^d$ be the old-minus-new per-use event vector for workload type i, and let n_i be its nonnegative integer multiplicity. The net event saving is

$$\Delta(n) = \sum_i n_i s_i - K. \tag{10}$$

A price vector $w \in \mathbb{R}^d_{\geq 0}$ specifies conversions into one declared additive resource measure. Its priced saving is $w \cdot \Delta(n)$. The prices are modeling assumptions, not learned hardware laws inferred from the event counts.

Proposition 25.1 (Resource dominance and declared price uncertainty)**.** *For a fixed ledger and workload, $w \cdot \Delta(n) \geq 0$ for every nonnegative price vector w if and only if $\Delta(n)$ is componentwise nonnegative. If admissible prices form the convex hull of a nonempty finite set W_0, then*

$$\min_{w \in \mathrm{conv}(W_0)} w \cdot \Delta(n) = \min_{w \in W_0} w \cdot \Delta(n).$$

Thus strict improvement throughout that price set holds exactly when every listed vertex gives a strictly positive margin. The claims concern the same workflow, workload, and complete additive ledger.

Proof. Componentwise nonnegativity makes every nonnegative weighted sum nonnegative. If coordinate j is negative, the jth coordinate vector is a nonnegative price vector giving a negative sum, proving the converse. Every point of the specified convex hull is a convex combination of its listed points. Linearity makes its priced saving the same combination of their savings, which cannot be smaller than their minimum. A listed minimizer attains that minimum, proving the equality and the strict criterion. □

This is ordinary linear accounting, not a new optimization principle. It prevents a system from asserting a price-independent gain after improving only one coordinate. The deterministic workload count must also be separated from an expectation over future tasks. Proposition 28.1 addresses the latter under its stated assumptions. Changing the task mixture can change both the amount and the sign of an improvement.

Counterexample 25.2 (One valid controller, opposite resource conclusions)**.** For the 32-case weighted-search comparison, the old-minus-new search-event vector is

$$(1552,\ 1552,\ -1868)$$

in model-record reads, rational additions, and capacity comparisons. Scope checking and final witness checking are equal for both procedures and cancel in this particular difference. Prices $(1, 1, 1)$ give a saving of 1,236 units. Prices $(1, 0, 1)$ give a loss of 316 units, and prices $(0, 0, 1)$ give a loss of 1,868 units. The identical returned controller remains valid in every comparison. None of the price choices was measured as a universal property of a processor.

The zero setup in that isolated execution comparison is not a statement that preparing the whole improvement was free. For an explicitly hypothetical additional setup charge of 2,000 comparison units and prices $(1, 1, 1)$, one use loses 764 units while two uses save 472. The purpose of this small calculation is to expose the dependence on the declared ledger. The actual equation-discovery, proof-search, quotient-construction, and revision-management records must be included when assessing the entire first-edition workflow.

Part VII

Evidence and recursive continuation

26 Evidence that survives candidate selection

A self-improvement procedure has an unusual incentive to find weaknesses in its own measurements. It searches repeatedly, retains favorable results, and changes the way later candidates are generated. A validation number can therefore become part of the training information even when it is never called a training label. This is a central issue in adaptive data analysis [15].

The correct distinction is between development evidence used to generate a candidate, selection evidence used to choose among candidates, and final evidence used to assess the selected procedure. A private test repeatedly consulted by an outer selector is still selection information for that outer process. Keeping it hidden from the inner solver does not make it independent of the entire lineage.

A practical record should identify what was frozen before each assessment, which outputs were exposed, and which claim the assessment supports. When a procedure changes a checker, both the checker change and the candidate it would approve cannot receive their only justification from the new checker's own approval. That restriction does not forbid improving evaluators. It prevents the target of validation from silently changing the validation standard in the same step.

Proposition 26.1 (Conditional error budgets across an adaptive lineage)**.** *Let $\mathcal{H}_{t-1}$ be the information available before attempt t. Suppose the event A_t that attempt t accepts a false improvement claim satisfies*

$$\Pr(A_t \mid \mathcal{H}_{t-1}) \le \delta_t \quad \textit{almost surely},$$

for deterministic $\delta_t \ge 0$ with $\sum_{t\ge1} \delta_t \le \delta$. Then

$$\Pr\left(\exists t \ge 1 : A_t\right) \le \delta.$$

The candidate-generating process and the time at which attempts stop may depend on the earlier history.

Proof. Taking expectations gives $\Pr(A_t) \le \delta_t$. The union bound gives the result for any finite number of attempts. The infinite statement follows from continuity of probability for the increasing finite unions. A history-dependent stop can only reduce the set of attempts whose acceptance events occur. □

No independence between attempts is needed. The conditional premise is stronger than a marginal guarantee for a candidate considered before adaptive selection. It must be justified by the actual sampling or verification mechanism. A budget such as $\delta_t = \delta/[t(t+1)]$ is summable, but writing that schedule does not make an invalid test conditionally valid.

26.1 One sufficient fresh-audit construction

Freeze a candidate and its comparator using the prior history. Draw n_t fresh evaluation cases conditionally independently from the declared distribution and record paired score differences $Y_i \in [-1, 1]$. Let Δ_t be their conditional mean and $\widehat{\Delta}_t$ their average. The bounded-variable concentration inequality gives

$$\Pr\left(\widehat{\Delta}_t - \Delta_t \ge r \mid \mathcal{H}_{t-1}\right) \le \exp(-n_t r^2/2).$$

Consequently, accepting a claim $\Delta_t \geq \gamma$ only when

$$\widehat{\Delta}_t - \sqrt{2\log(1/\delta_t)/n_t} \geq \gamma \tag{11}$$

meets the preceding conditional error requirement for that claim. This is a fixed-sample sufficient construction. The candidate and n_t may depend on past data, but n_t is fixed before this audit is examined. Inspecting the same sample repeatedly and stopping at a favorable value requires a time-uniform method or a separately justified spending rule. Time-uniform concentration is an established subject [16].

Formula (11) can be conservative and expensive. It is included to make the premise operational, not as a claim that this test is optimal. If evaluator randomness, shared tasks, or dependence between runs changes the sampling unit, the analysis must change with it. Thousands of messages from one trained lineage do not automatically become thousands of independent improvement trials.

Counterexample 26.2 (Reused evidence can reward memorization)**.** A task has N equally likely names and an externally fixed binary answer for each. After the answers on a development subset become available, a candidate stores them and gives the wrong answer on every other name. It is perfect on that subset and has true accuracy equal only to the subset's fraction of the task universe. An outer loop that keeps selecting by the same subset can improve its reported score without improving beyond the information it has exposed. Renaming the subset a holdout does not restore independence.

26.2 Proof obligations and performance evidence

A semantic proof can establish that a proposed operation preserves a specified relation. A program checker can establish that a finite tree respects the supplied domains and budgets. A fresh performance comparison can support usefulness on a declared task law. None of these establishes the truth of every environmental premise. Their evidence types should remain separate in an improvement receipt.

The constructive tests shipped with this paper are deterministic finite enumerations. They are not samples from an AGI population, and they do not use (11) to manufacture confidence intervals. Their role is to check the implementations of the stated finite arguments.

27 Sequential evidence for an improved discovery procedure

The deterministic compiler comparison above can be checked on its whole declared family. A learned discovery procedure evaluated on fresh stochastic tasks would need another kind of evidence. It may also be expensive enough that the auditor would like to inspect results sequentially and stop once the evidence is adequate. The fixed-sample rule in Section 26 does not automatically justify that stopping policy.

We give a small time-uniform alternative whose finite arithmetic is directly inspectable. Before an audit begins, freeze the candidate procedure, its comparator, their installed information, the task law, the common resource cap, and a paired score $Y_t \in [-1, 1]$. Timeouts and invalid outcomes must have a stipulated score rather than disappear from the sample. Fix a proposed improvement margin $\gamma \in [0, 1)$.

The null condition is

$$\mathbb{E}[Y_t \mid \mathcal{H}_{t-1}] \leq \gamma \quad \text{for every audit step } t, \tag{12}$$

where $\mathcal{H}_{t-1}$ includes all information available before the next score. Conditionally independent cases from a fixed task law with mean at most γ satisfy this condition. An auditor cannot change the task distribution toward favorable cases and retain that interpretation without a new argument.

Choose a nonnegative bet λ_t, measurable from $\mathcal{H}_{t-1}$, with $\lambda_t \leq 1/(1+\gamma)$. Define

$$E_0 = 1, \qquad E_t = E_{t-1}\big(1 + \lambda_t(Y_t - \gamma)\big). \tag{13}$$

The variable E_t is an evidence process, not a posterior probability that the candidate is better. Predictability means choosing the bet before observing the score to which it is applied.

Theorem 27.1 (An anytime-valid paired improvement audit)**.** *Under* (12) *and the stated predictable bet constraints, E_t is a nonnegative supermartingale. For every $\delta \in (0,1)$,*

$$\Pr\left(\sup_{t\geq 0} E_t \geq 1/\delta\right) \leq \delta.$$

Consequently an audit may reject the stated null at its first crossing of $1/\delta$, while choosing its stopping time from the observed audit history. The result does not require a predetermined number of cases or independence beyond the conditional-mean premise.

Proof. Since $Y_t - \gamma \geq -(1+\gamma)$, every factor in (13) is nonnegative. Conditional expectation and predictability give

$$\mathbb{E}[E_t \mid \mathcal{H}_{t-1}] = E_{t-1}\big(1 + \lambda_t(\mathbb{E}[Y_t \mid \mathcal{H}_{t-1}] - \gamma)\big) \leq E_{t-1}.$$

Let τ be the first threshold crossing. For a finite N, the stopped process at $\tau \wedge N$ still has expected value at most one, by induction on the stopped conditional expectations. On $\{\tau \leq N\}$ its value is at least $1/\delta$, while it is nonnegative otherwise. Therefore $\Pr(\tau \leq N) \leq \delta$. Letting N increase proves the result for any crossing time. □

This is the elementary nonnegative-supermartingale method underlying time-uniform evidence, not a new general concentration theorem [16]. The contribution here is its explicit paired comparison and its placement within the recursive resource and information contract. In particular, an audit observation is a complete comparable attempt by each discovery procedure, not an arbitrary message emitted within one attempt.

27.1 Why seeing the outcome before choosing the bet fails

Counterexample 27.2 (Hindsight betting creates false evidence)**.** Let Y_t be independent fair signs and $\gamma = 0$. Choose $\lambda_t = 1$ when the current sign is positive and zero otherwise. This invalid rule doubles E_t on a positive score and leaves it unchanged on a negative one. Within four scores it crosses the threshold four whenever at least two scores are positive, an event of probability $11/16$. This exceeds the nominal bound $1/4$. Each individual bet lies in the numerical range $[0,1]$. What fails is predictability.

A data-dependent stopping rule is allowed by the theorem. A data-dependent choice that looks at the *current* outcome before staking on that outcome is not. The software can validate the numerical range of a bet, but it cannot infer the provenance of an externally supplied bet from that number alone. A usable audit receipt must record the commitment order and the sampling contract.

The same issue arises when a candidate is edited after seeing an audit result. The resulting process no longer evaluates one frozen procedure under its original interpretation. It can start a new comparison, debit a new conditional error budget, and retain the earlier result as development evidence. It cannot silently reuse the old evidence process as though the candidate had not changed.

27.2 Evidence across recursive generations

For attempt j, condition on the complete earlier lineage, freeze its candidate, and initialize a fresh valid audit at $E_0^{(j)} = 1$ with error allowance δ_j. If $\sum_j \delta_j \leq \delta$, Proposition 26.1 bounds the probability of ever accepting a false margin claim by δ. Each audit can stop sequentially, and the next candidate may depend on completed earlier audits. The premise remains conditional validity for each actual attempt.

Exact certificate checks and sequential evidence still have different jobs. A table-defined compiler can be checked for admissibility and optimality on its formal input without making a claim about a population of tasks. A stochastic audit can support a productivity comparison on that population without proving every generated program correct. The recursive system needs both whenever its claim combines semantic validity and statistical usefulness.

The new reference suite enumerates finite rational null paths and confirms the crossing bound in those cases. It also checks the hindsight counterexample. No live learner was evaluated with this test, and the deterministic compiler counts are not retroactively presented as independent random samples.

28 The complete cost of becoming a better improver

A procedure can save work on each future task and still be a poor investment when its discovery and validation are expensive. Conversely, a costly method can be worthwhile when it is reused sufficiently often. Both observations are ordinary resource accounting. They become especially important when a claimed improvement shifts work from inference into training, into a translator, or into an external oracle.

Use a resource vector rather than an unexplained token total:

$$C = (C_{\text{model}}, C_{\text{search}}, C_{\text{validate}}, C_{\text{install}}, C_{\text{execute}}, C_{\text{store}}).$$

These coordinates need not share units. Model construction may involve environment observations. Search and validation may involve processor work. Installation may require conversion or compilation. Execution work concerns one deployment. Storage concerns the retained library and complete controller. A scalar comparison needs declared conversion weights or a common physical budget, not an arbitrary sum of unlike quantities.

For the hidden-cell instrument, a compiled tree has fifteen records even though a successful depth-three run visits only four. The bit-query API is installed for both procedures in the matched comparison. Its table representation, if used, occupies $n(n + \log_2 n)$ Boolean entries for the chosen power-of-two family. The reported three-versus-seven diagnostic comparison does not imply that inspecting a large subset costs one machine instruction in every implementation. It uses one sensor call as its stipulated unit.

Proposition 28.1 (Amortization of a reusable process improvement)**.** *In a fixed additive resource unit, suppose installing a new procedure has incremental cost $C \geq 0$ relative to the old one, and its expected per-use saving on an unchanged task distribution is $d > 0$. If there are N uses and no additional costs or changes in the task law, its expected cumulative saving is $Nd - C$. It is strictly resource-improving exactly when $N > C/d$.*

Proof. Linearity of expectation gives expected total use savings Nd, without requiring independence between uses when each has the stated expectation. Subtract the fixed incremental cost and solve the strict inequality. □

The formula is not a prediction of future demand. Distribution drift, maintenance, growing context, retrieval overhead, or a changed interface can alter d or add costs. The value of the repair library depends on whether later problems actually admit its operations.

28.1 Larger search spaces can lower productivity

Counterexample 28.2 (More expressibility with worse fixed-budget discovery)**.** A search language has N candidates, exactly one useful. A procedure samples uniformly with replacement $B \geq 1$ times and recognizes usefulness perfectly after each sample. Its probability of finding the useful candidate is $1 - (1 - 1/N)^B$. Adding an additional useless candidate preserves every old expression and strictly decreases this probability. The best attainable repair is unchanged, while the implemented discovery procedure becomes worse under the same sampling budget.

This does not argue against grammar growth. It identifies what the learner must improve alongside growth: proposal probabilities, indexing, modular reuse, or informative pruning. The search space and the search distribution are different parts of the agent.

An archive has a related ambiguity. The best true value among retained artifacts cannot decrease when an old artifact remains available under the same score. But a deployed selector can perform worse as retrieval becomes harder. The monotonicity of a hypothetical maximum is not the monotonicity of actual use. Resource-limited archive management belongs inside the recursive system rather than outside its reported budget.

29 A finite certificate for repeated improvement

A valid local change can destroy the ability to make another valid change. A new representation may discard the evidence needed to verify future proposals. A controller may consume the remaining budget while preserving the current task score. A shortcut can bypass an interface on which later transports depend. Recursive continuation therefore needs its own object.

Let Z be a finite set of complete lineage configurations. A configuration includes the currently selected solver and improver, retained libraries and evidence, the active comparison contract, and relevant resource state. The following theorem assumes this configuration is observed. Unknown configurations require belief-state modeling or a different synthesis result. No unrestricted introspection is inferred.

Each z has a finite set of authorized attempts $A(z)$. An attempt has a nonempty set $\mathrm{Post}(z, a)$ of possible successor configurations. This can represent all outcomes of a stochastic proposal or an abstract uncertainty set. A safe set $\mathsf{Safe} \subseteq Z$ encodes the declared semantic, evidence, authority, and resource conditions. A pause or terminal state may have a zero-progress self-loop. Actual expensive attempts must debit resources in Z, or the model must explicitly state their replenishment.

Define

$$K_0 = \mathsf{Safe}, \qquad K_{n+1} = \{z \in K_n : \exists a \in A(z),\ \mathrm{Post}(z, a) \subseteq K_n\}.$$

The descending finite sequence reaches a fixed point $\mathcal{K}$. It is the set from which some policy keeps every successor safe indefinitely. This can include doing nothing forever. Safety alone is not recursive improvement.

For each transition, a progress mark $p(z, a, z') \in \{0, 1\}$ is supplied by a checked criterion. A mark of one means the specified process-improvement claim has been established under the same comparison. It is not assigned merely because a file changed. Define

$$W_{0,0} = \mathcal{K}, \qquad W_{0,k} = \varnothing \quad (k > 0), \tag{14}$$
$$W_{h,k} = \{z \in \mathcal{K} : \exists a \in A(z),\ \mathrm{Post}(z, a) \subseteq \mathcal{K}, \quad z' \in W_{h-1,\max\{0,k-p(z,a,z')\}} \quad \forall z' \in \mathrm{Post}(z, a)\}.$$

Here h is the number of remaining attempts and k the number of still-required progress events. A pause action lets a policy that finishes early use the remaining horizon without changing the result.

Theorem 29.1 (Common-policy recursive progress)**.** *A configuration belongs to $W_{h,k}$ if and only if there exists one authorized policy that keeps all outcomes in $\mathcal{K}$ and guarantees at least k checked progress events within h attempts. A policy depending on the current configuration and the remaining pair (h, k) suffices. The finite recurrence gives a witness when feasible and a complete negative answer within the stated model when infeasible.*

Proof. At horizon zero, zero remaining progress is possible exactly in $\mathcal{K}$, and positive remaining progress is impossible. Assume the claim for $h - 1$. A policy at z must choose one initial action before its outcome is

known. Every possible successor must remain in $\mathcal{K}$ and must admit a continuation meeting the remaining target $\max\{0, k - p\}$. By induction these conditions are exactly the memberships in (14). Conversely, choose a witnessing first action and a witnessing continuation for each observed successor. The resulting policy works on every path. Induction proves necessity, sufficiency, and the stated memory requirement. Finiteness makes the construction terminating. □

The use of one first action is important. The weaker claim that every possible outcome has some favorable action, selected after secretly learning that outcome, is not an implementable policy. Likewise, independently feasible semantic, resource, and evaluation certificates must refer to the same attempt and successor relation.

The theorem is complete only for its finite catalogue and its information boundary. It does not create a promising next candidate or identify the true productivity of an arbitrary program. Its progress labels may come from exact finite evaluation or from a separate evidence layer. With probabilistic labels, the conclusion about actual progress holds on the event that the relevant certificates are valid. Proposition 26.1 can bound failure of that event under its own assumptions.

29.1 Safe rejection can prevent a worst-case progress guarantee

Counterexample 29.2 (Repeated opportunity is not bounded sure progress)**.** At a safe configuration, an attempt either produces a genuine improvement or rejects the candidate and returns to the same configuration. Both outcomes are possible at every attempt. A safe pause is also available. For every finite h, there is a path with h consecutive rejections, so no policy guarantees one improvement within h attempts. With an independent positive success probability, eventual success can nevertheless occur with probability one. The worst-case finite theorem and that probabilistic statement answer different questions.

29.2 Why local improvement does not imply recursive viability

A second finite example has a proposal that raises the current score but deletes the only retained witness needed to validate any subsequent proposal. Its destination lies outside Safe when witness availability is protected. If witness availability is omitted, it may remain safe but have no productive continuation. The result depends on the contract, not the flattering name assigned to the edit.

This is a compact version of the closure perspective in SRIC [2]. The present contribution is to connect that perspective to a concrete diagnostic tree, a resource witness, and a specified productivity comparison rather than treating closure as a free consequence of successful self-modification.

30 Acceleration is an additional hypothesis

Recursive improvement can be meaningful without being indefinitely accelerating. Its measured pace depends on useful gains, the cost of finding and checking them, available opportunities, and the chosen scale of evaluation. A normalized task score has a ceiling. A changing suite can continually introduce new headroom, but its successive normalized scores are not automatically comparable.

Proposition 30.1 (A fixed-margin ceiling on a fixed score)**.** *Suppose a lineage has a fixed measure $R_t \in [a, b]$, every accepted progress event increases it by at least $\gamma > 0$, and all other transitions are nondecreasing in that same measure. Then the number of progress events is at most*

$$\left\lfloor \frac{b - a}{\gamma} \right\rfloor.$$

An infinite sequence of positive increments is compatible with bounded total improvement when no common positive lower bound is imposed.

Proof. After k progress events, telescoping and nondecrease give $R_t \geq R_0 + k\gamma \geq a + k\gamma$. Since $R_t \leq b$, the bound follows. For the second assertion, start at zero and use increments 2^{-t} for $t \geq 1$. Every increment is positive and their partial sums remain below one. □

If nonprogress transitions can decrease the measure, the first bound need not hold: repeatedly falling and recovering can generate arbitrarily many positive increments. Those recoveries are not unlimited cumulative progress. A moving evaluator creates a different problem. It may be appropriate, but it requires bridges to preserve the meaning of the comparison.

A simple accounting model can express the additional conditions for acceleration. Let g_t be an expected useful gain and $c_t > 0$ the complete cost of the corresponding discovery and acceptance cycle. Improvement productivity per unit resource is g_t/c_t. Raising g_t alone does not raise that ratio when c_t rises faster. A larger reachable repair set is similarly insufficient when proposal quality falls or validation becomes the limiting cost.

SRIC's bottleneck perspective is useful here: improving one stage can expose another stage as the dominant limitation [2]. The next bottleneck might be information acquisition rather than proposal generation, or evidence quality rather than repair expressibility. A theory of acceleration would need to specify how those bottlenecks move and which resources support their resolution. A claim of universal acceleration does not follow from the existence of one productive cycle.

Part VIII

A bounded route toward general intelligence

31 A proposed learned architecture

The mathematical contracts do not specify a complete learning algorithm. They do identify the components that a credible algorithm must make learnable and the evidence required when those components change. The intended architecture has one active solver, a recurrent working state, a persistent library of typed operations, an acquisition policy, a proposal mechanism, and an admission procedure. These are roles in one realized computation, not a requirement for a separate large model for every role.

31.1 What would learn

Let r_ϕ encode the retained state, $G_{\theta,a}$ describe proposed operations, π_ω choose observations or experiments, and p_ψ propose candidate programs or parameter updates. A development episode can modify $\phi, \theta, \omega, \psi$ through a declared learning rule. A persistent program library can also change by adding, replacing, or removing checked operators. Temporary deliberation without a retained modification is not called a new learned generation here.

The learning signals need separate purposes. Prediction losses train anticipation of future outputs. Intervention-consistency losses test whether the proposed operations match their intended effects. Acquisition rewards favor useful reductions of repair uncertainty after charging experiment cost. Proposal rewards evaluate the actual selected descendant rather than fluency of its rationale. Preservation checks protect specified older capabilities or interfaces. There is no universal set of weights that turns these distinct obligations into one sufficient scalar objective.

A latent computation may suggest that a particular variable matters. A typed checkpoint should then declare the variable's domain, the operations it supports, the evidence used to infer it, and the continuations on which it was checked. Its translator must not read the answer that the checkpoint claims to predict. Its complexity is charged. OSL is a proposed semantic consumer for such checkpoints, while Neuralese concerns the learned latent producer [4, 3]. This paper does not implement their complete learned bridge.

Proposition 31.1 (Local operation agreement preserves finite continuation)**.** *Let D be invariant under every source operation F_a, let $q : D \to Z$, and suppose*

$$qF_a = G_a q \quad \text{on } D$$

for every admitted a. If the required output satisfies $y = \bar{y}q$ on D, then for every finite operation word w,

$$qF_w = G_w q, \qquad yF_w = \bar{y}G_w q.$$

Proof. The empty word gives the identity. Assume the relation holds for a prefix. Domain invariance allows application of the next local equation, giving $qF_aF_w = G_aqF_w = G_aG_wq$. Induction establishes the first identity and composing with $\bar{y}$ gives the second. □

The proposition is a specification for meaningful reuse, not a method for learning q or G_a. Invariance matters because an unchecked operation can leave the domain where the equations hold. Output preservation matters because commuting hidden states need not retain the task variable. Actual causal use matters because a probe can decode an appropriate state that the solver does not use.

31.2 What a complete attempt would do

A bounded iteration would begin with an observed task failure and an explicit set of plausible causes. The active acquisition policy chooses an admitted experiment. Its result updates those causes and the current state. A proposer supplies a candidate repair or a new constructor when the old language is insufficient. A separate execution checks the candidate's declared interface, resource usage, and behavior. A selected candidate is assessed on evidence not used to construct it. If admitted, its actual changes and scope are stored, including the procedure that found it.

A later outer attempt may change the acquisition policy, proposer, or selection code. It must then be assessed as an improver from comparable starting conditions, not only as the latest solver. An improvement may be retained provisionally for a particular domain rather than promoted globally. Failed hypotheses remain useful evidence when their assumptions and reasons for rejection are preserved.

Candidate evaluation can use the same kind of mathematical tools as the proposer without sharing the candidate's unchecked assumptions. Independence here means a specified information and authority separation, not a mystical absence of common software or training ancestry. External audit data and the authority to redefine a protected score are not supplied to a candidate as editable inputs for the same transaction.

31.3 A learned policy is not an unconstrained interpreter

A sufficiently powerful interpreter can hide the entire solution behind a small label. The remedy is not to forbid interpreters by definition. It is to state their installed information, runtime, storage, and access. This is the resource lesson of *Machine Understanding* [1]. The diagnostic tree is intentionally explicit so that this accounting can be performed exactly in the worked case.

Nor should neural implementation be required simply for appearance. A library of executable operators and a learned experiment selector can be a meaningful adaptive system. Conversely, updating a large weight tensor does not automatically establish a useful representation, a causal mechanism, or a transferable improvement procedure. The architecture is evaluated by its realized operations and the origin of their improvement.

32 The compiled diagnostic improvement

We can now state the exact mathematical achievement of the running example. It is deliberately smaller than the full learned architecture, but complete enough to expose the relevant distinctions.

For $n = 2^k$ cells, assume a single defective cell, binary reports, singleton queries, and k bit-coordinate queries. All queries leave the cell unchanged and cost one diagnostic call. A correct final repair costs one additional call. Distinct cells require distinct terminal repairs. There are no false repair commits. A literal tree record contains one experiment or repair identifier plus its child references.

Theorem 32.1 (Optimal diagnosis in the hidden-cell family)**.** *Every certain diagnosis needs at least k binary queries in the worst case. The bit-query procedure attains this bound and uses $2n - 1$ records in a full binary tree. Any procedure using singleton queries alone needs at least $n - 1$ queries in the worst case, and equality-first testing attains that bound with the same number of tree records. Thus their worst total execution costs, including repair, are $k + 1$ and n.*

Proof. A binary tree of depth less than k has fewer than $2^k = n$ leaves. Every successful leaf can identify at most one cell because its repair must be correct for every remaining case. Therefore depth at least k is required. The k coordinate bits identify every cell and give a complete binary tree with n leaves and $n - 1$ internal nodes. For singleton tests, an adversary can answer no on each tested cell while at least two untested cells remain. Fewer than $n - 1$ distinct singleton queries therefore cannot guarantee identification. Equality-first

testing uses one leaf for each positive answer and one final leaf for the last remaining cell, again totaling $2n - 1$ records. Adding the terminal repair gives the claimed work counts. □

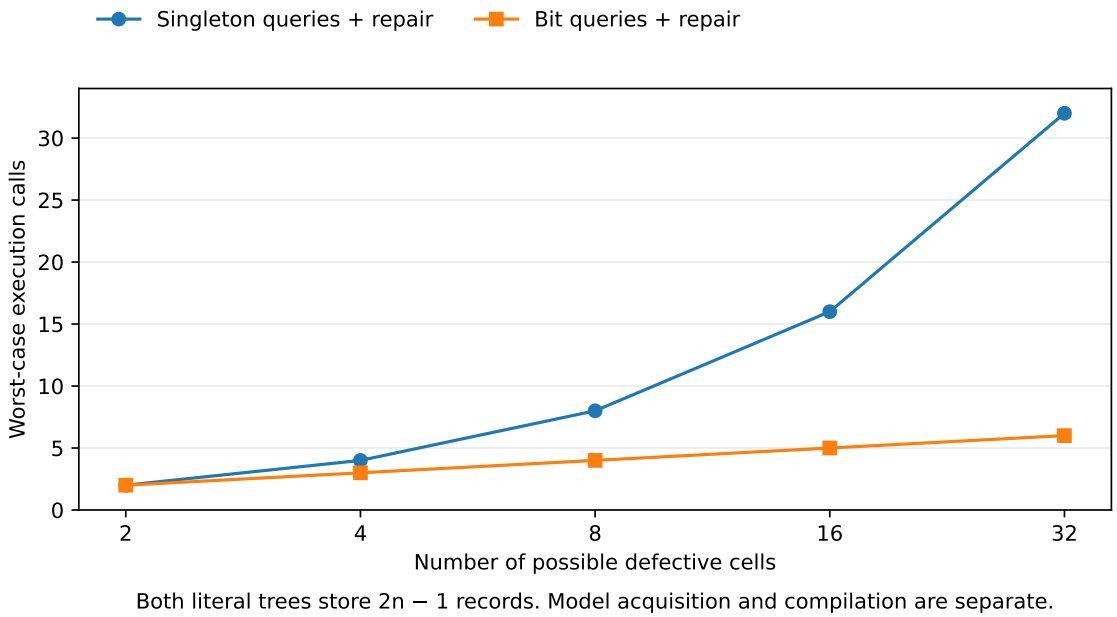


Figure 11: Exact total execution costs for the power-of-two cell family. Singleton testing requires $n - 1$ diagnostic queries, while the admitted bit queries require $\log_2 n$. A final repair adds one call to each plotted curve. Both full trees have $2n - 1$ stored records. The curves omit separately charged model acquisition, compilation, and adapters.

32.1 What is actually compiled

The reference compiler receives an observation table and the set of successful terminal repairs. It computes the frontier in Theorem 6.2 and returns an optimal tree under a chosen work/storage allowance. At $n = 8$ it returns a four-call, fifteen-record witness. A separately coded interpreter checks every initial cell and every branch. The reference comparison constructs the equality-first baseline independently, rather than comparing the compiler to a weakened version of its own output.

The selected tree is an executable diagnostic policy for future episodes. It is not a saved answer for the calibration case. In a new episode, the hidden cell is unavailable to the policy. Only reports from its queries determine the branch and eventual repair. A cell permutation supplies a controlled foreign interface when the query sets, terminal repairs, and adapter are transported together. Theorem 9.1 explains that transfer. The tests do not claim that the adapter was learned without correspondence data.

32.2 The measured object is a procedure

Under the earlier four-call cap, the equality-first improver repairs $3/8$ of the uniform eight-cell cases and otherwise abstains. The compiled improver repairs all cases. The repaired solvers themselves are simple: each now knows the correct cell-specific operation. What improved is the procedure that acquires that case-specific correction. The library and sensor language are unchanged.

This original controller-level construction does not by itself improve the compiler. The compiler and acquisition constructions in Parts III–V provide the additional changes to compilation and rule acquisition.

Each comparison retains its own starting procedure, supplied information, and cost unit. Their coexistence does not convert the original eight-cell demonstration into evidence of indefinite self-improvement.

32.3 Controls that expose the boundary

Removing the bit-query family changes the synthesis problem and restores the singleton lower bound. Omitting the adapter breaks foreign-label replay even when the tree shape is unchanged. Deliberately choosing an incorrect leaf causes the independent checker to reject the certificate. Reducing the allowed tree records below fifteen rules out the certain eight-cell solution in this literal binary-tree model. These are failures of distinct requirements, not reasons to reinterpret the target after a run.

The repair-aligned query example in Section 6 supplies a different control. There the compiler need not identify the exact hidden case. It should stop when one repair works for every surviving case. This prevents full identification from being hard-coded as the universal objective of inquiry.

33 Generality requires more than semantic relabeling

The exact transport theorem is useful because it proves what is preserved when a correspondence exists. It is not evidence that the system can discover correspondences efficiently across arbitrary tasks. A relabeled hidden-cell world remains the same informational problem. Transfer from diagnosis to theorem discovery, planning, or empirical model construction needs additional argument.

A credible generality study would vary several things separately: the state distinctions needed for future behavior, the allowed experiment language, the repair constructors, the information available to the learner, and the resource costs. It would also separate unfamiliar instances of a known schema from unfamiliar combinations of schemas and from an operation family that the old language cannot express.

The proposed mechanism is not that every task reduces to binary cell location. It is that many tasks can expose a useful pattern: a currently retained representation fails to support a consequential distinction, an experiment can discriminate among causes, and a reusable operation can resolve the resulting class of failures. Whether that pattern is broad enough and economical enough to contribute to general intelligence is a research hypothesis.

33.1 The next decisive learning comparison

The most informative next study would compare a parent and descendant improvement procedure on the same family of fresh starting solvers. It would hold installed libraries and base-model access fixed in the primary contrast, then add a separate full-system contrast that includes any learned library. A stronger ordinary program-library learner should be included, since library learning itself is established [8].

The development system would be allowed to acquire or revise an encoder, experiment selector, and operator library in bounded simulated environments. Evaluation would include longer continuations, new operator compositions, and changed observation formats. Each change should have an explicit semantic target and a resource ledger. Removing acquisition adaptation, repair reuse, or recurrent state separately would locate which mechanism produces a gain. Those are proposed experiments, not results reported in this paper.

The desired endpoint is not merely that the descendant solves more familiar cases. It is that its acquired improvement procedure produces better valid repairs on unfamiliar cases within the same budget, and that the gain survives adequate controls for retained answers, installed adapters, and selection leakage. A subsequent generation must repeat that comparison rather than inherit its predecessor's claim by name.

33.2 What would count against the central hypothesis

The obstruction-guided procedure may be no better than an ordinary search method given the same information. Its apparent advantage may vanish after charging its diagnostic model and checker. Its recovered operators may fail on longer compositions. A learned selector may memorize surface labels instead of identifying the causal distinction. A larger repair language may dilute search enough to erase its representational advantage. These outcomes would identify where the proposed integration fails or requires stronger assumptions.

A negative result on a restricted task family would not prove that general intelligence is impossible. A positive result would not prove AGI either. Both would help determine whether the proposed mechanism earns its place in a larger architecture.

34 Open problems beyond the first edition

The exact core rests on finite models, supplied semantics, bounded horizons, and explicit information. Several important extensions remain.

Learning the experiment model. The tagged-model theorem accounts for a fixed unknown member of a supplied finite class, including paid calibration. It does not construct that class from open-ended experience or guarantee that it contains the true process. Safe learning with noisy, misspecified, or changing observation models remains open in this framework. Model uncertainty must not be replaced by a guessed deterministic table merely to preserve an exact theorem.

Discovering a repair language. Both the original compiler and the selected shortcut use a supplied repair language. The earlier selector chooses within a supplied catalogue of compiler templates. Part IV additionally synthesizes a primitive arithmetic rule after exposing a missing feature. Its grammar, feature registry, teacher, and proof gate remain supplied. Open-ended constructor discovery needs a representation of candidate meanings, a method for proposing them, and evidence that the new language preserves old operations while making new problems tractable. Expressibility of all computable programs is not an efficient search strategy.

Acquiring stronger search representations. Part V changes the candidate space used to acquire future discovery rules. The new equations and quotient are generated, while their arithmetic meanings and proof machinery are supplied. A stronger supplied normalizer remains a useful comparator. Learning effective proof search, choosing cheaper rule subsets, and adapting the representation when a language changes are distinct next questions. A finite-consistent equation is not automatically a valid optimization.

Transfer without a supplied correspondence. The exact policy-transport theorem assumes its maps. Learning them introduces gauge ambiguity, underidentification, lossy observations, and potential answer leakage. The Neuralese-to-OSL bridge remains a separate learned realization problem, not a solved consequence of writing a commuting square [3].

Improving evaluation without changing the question. An evaluator can be genuinely defective. Correcting it requires retaining evidence of the defect, specifying the new scope, and distinguishing a change in estimated performance from a change in the underlying target. A fully unrestricted evaluator rewrite would remove the stable comparison on which the reported improvement depends.

Noisy and distributed recursive synthesis. The finite progress theorem uses fully observed configurations and worst-case successor sets. Useful systems may need probabilistic service, partial observation, or several

decision centers. Separate success probabilities do not automatically yield one policy that satisfies all constraints. A noisy extension must state both the risk accepted and the evidence supporting its transition law.

Long-run productivity. Repeated finite improvements do not determine the eventual learning curve. A richer task distribution, increasing resources, changing architectures, or new scientific observations may create further headroom. Those changes should be modeled and charged rather than absorbed into a claim of autonomous acceleration. The current paper does not settle whether general-purpose recursive improvement has increasing, constant, or diminishing returns.

34.1 Authority and bounded implementation

The formal permission to modify a local program is not authorization to change external services, access private data, or remove an institution's controls. Proposed deployments would require an explicit scope of authority and a justified handling of affected parties. The reference package does not connect to services, modify other agents, train a neural model, or conduct autonomous external research. It constructs and checks finite mathematical objects in a local process.

Similarly, stopping and refusal are legitimate outcomes when required evidence or resources are missing. An improvement claim must not become a reason to suppress those outcomes. A preserved archive can retain unsuccessful attempts without promoting them into operating authority. These commitments follow the evidence-preserving lineage perspective rather than a presumption that continued self-modification is always desirable.

35 Conclusion

A better solver, a better repair procedure, and a better process for discovering repair procedures are different objects. They can interact productively, but the interaction needs an explicit realization. General intelligence motivates the search for transferable discovery methods. It is not a premise from which those methods follow.

The finite theory keeps the relevant obligations in one chain. An investigation must acquire enough information to choose a successful repair, without necessarily identifying every hidden detail. Its execution and stored controller must fit the same resource witness. A transfer must preserve observations and admissibility, not only an output. A language extension changes which lower bounds remain valid. An adaptive lineage must retain the evidential meaning of its accepted gains and a continuing policy for any promised progress.

The constructive sequence now reaches the acquisition of discovery rules. A compiler produces better diagnostic trees. A guarded strategy reduces the search needed to find such trees. A learner composes an internal rejection predicate. A further learned equation library prepares a smaller search representation for reacquiring that predicate and other targets. Exact teacher-transcript preservation identifies what has not changed: the required evidence and the final selected AST. The additional improvement concerns repeated evaluation work, not a newly supplied answer.

Correctness and productivity remain different. A supplied full normalizer can remove more redundancy than the learned library. Preparation can cost more than a short workload saves. Streaming avoids some model reads while adding comparisons. A repair that succeeds on several cases can make an old identification bound unsound. These are not peripheral caveats. They determine what the acquired modification is and when it is worth using.

The finite constructions demonstrate specific changes in discovery behavior under declared semantics. They do not establish learned physical meaning, efficient search over every computable procedure, or an indefinitely improving sequence of outer learners. Their positive contribution is an auditable connection between representations, generated rules, checked equivalences, resource use, and recursive revision. A

claim about becoming better at improvement can now be located at a particular level, checked against its predecessor, and refused when its evidence or cost contract is missing.

The first edition also makes those claims revisable without making their premises disposable. A changed repair language can invalidate a weight bound while leaving the equation library sound. A changed resource price can reverse a cost conclusion without altering one controller transition. The dependency-complete reuse theorem states when old witnesses remain applicable, and the resource proposition states when an event saving is independent of a selected price model. Their role is to keep one actual, inspectable chain of reasoning and computation attached to every claimed gain.

A A complete diagnostic trace

Take $n = 8$ and hidden defect $j = 5$, whose binary digits are 101. One optimal tree uses the three queries

$$A_0 = \{1, 3, 5, 7\}, \qquad A_1 = \{2, 3, 6, 7\}, \qquad A_2 = \{4, 5, 6, 7\}.$$

The precise order may differ among equally optimal compiled trees. For this order, the supports are

$$\{0, \ldots, 7\} \xrightarrow{1} \{1, 3, 5, 7\} \xrightarrow{0} \{1, 5\} \xrightarrow{1} \{5\}.$$

The terminal repair is r_5. The path uses three diagnostic calls and one repair call. The complete tree stores all eight repair leaves and seven query nodes, not merely this path.

Equality-first testing asks about cells zero through five before it obtains a positive report. It uses six diagnostic calls on this particular case, then one repair. Under the three-diagnostic cap it abstains after three negative reports. The cap is fixed before the episode, not shortened for the baseline after observing which cell is defective.

For a target relabeling $\sigma(i) = 3i + 1 \pmod 8$, the defect has target label $\sigma(5) = 0$. Each query becomes $\sigma(A_r)$ and the terminal repair becomes $r'_{\sigma(5)}$. Reports and tree shape agree with the source, but the label adapter is part of the transported interface. Applying the old numeric repair label five directly in the target would be incorrect. This is why syntactic resemblance does not replace a semantic map.

A.1 A repair problem that should stop before full identification

In the four-case example, the support $\{0, 1, 2\}$ admits repair a immediately. A compiler that insists on identifying the precise case would spend unnecessary work. The successful stopping condition is the existence of one action whose success set contains the whole support. Conversely, the support $\{2, 3\}$ admits no common repair in that example. A leaf chosen there without another observation is invalid even though either repair succeeds in one of its cases.

B Reference computations and evidence scope

The package implements the finite experiment model, the two-resource recurrence, tree reconstruction, and a separate replay checker. It also implements stable partition refinement and finite recursive-progress synthesis. Exact test groups compare selected constructions with separately coded enumerations on small domains. Integer and rational arithmetic are used for mathematical acceptance checks. Plotting converts exact analytical quantities to display coordinates only.

The verification program records its exact domains, input and output hashes, command, interpreter, duration, and outcomes. The source audit inventories formal statements and checks labels, citations, and their proof environments. The build uses LaTeX and the supplied vector figures. A clean rebuild is compared by extracted PDF text, not claimed to be byte-identical across environments.

These checks are authored within this project. Alternative algorithms provide implementation cross-checks, not independent peer review. No test count establishes a universal claim outside its implemented domain. The proofs in the manuscript provide the conditional general arguments. No Lean certificate, trained agent, learned semantic bridge, or automated research run is included.

B.1 A precise reading of the finite demonstration

The observation and success tables are supplied analytical semantics. The compiler selects a tree without access to the next episode's hidden index. The checker separately receives that index when exhaustively verifying all possible episodes. That division prevents a policy from consulting hidden state during replay. It does not make the supplied model an empirically learned object.

The main generated files are the compiled cell-tree certificate, a report of source and transported replays, the cost frontiers, and the exact check results. All can be regenerated locally. The README records the commands and the review notes record any failures or limitations encountered during preparation.

C Claim map and the scope of the first edition

Claim	What the present manuscript supplies
Feedback can support a repair	A transcript-discrimination bound and a common-repair stopping condition. No automatic causal diagnosis is inferred.
A representation supports continuation	A least stable refinement and explicit operation equations. Learning the representation is additional.
A repair can be found within a budget	An exact finite work/storage recurrence and a checkable tree. It requires a supplied finite experiment model.
A discovery procedure transfers	A correspondence preserving complete investigation semantics and costs. Discovering that correspondence is additional.
A pruning rule is learned	A finite grammar synthesis, a witnessed feature conflict, and a separate scope proof. The teacher and grammar are supplied.
A compiler skips work safely	Streaming root rejection preserves the controller and avoids counted rows. Predicate work remains charged.
A candidate procedure is better	A selection bound under uniform assessment error and a protected productivity comparison. The error premise must be established.
A lineage preserves credible gains	Conditional error spending and an explicit distinction between development, selection, and audit evidence.
A lineage can improve repeatedly	An exact finite common-policy criterion for marked progress. Marks require separate valid evidence.
The process accelerates	Not established. A fixed-score ceiling and cost accounting expose additional requirements.
The system is generally intelligent	Not established. Generality and learned discovery remain research targets across unfamiliar structures.
The synthesis procedure improves	A checked equation library removes redundant candidate evaluations while preserving teacher proposals. Preparation costs remain separate.
One repair covers several cases	A feasible repair-weight certificate replaces singleton counting. Its validity is rechecked after a repair-language change.
A composed gain is affordable	The actual acquisition, compilation, checking, and preparation costs must fit one common resource model.
Old certificates survive a change	Only dependency-complete unchanged obligations can be reused without new evidence. Affected descendants require re-establishment.
A gain is independent of prices	Componentwise event dominance, or a strictly positive margin on the declared finite price polytope, supplies the precise criterion.

References

[1] Komorebi P. *Machine Understanding: Representation, Explanation, and Transfer*. Operation Komorebi, first consolidated edition, v1.0.0, 8 October 2026.

[2] Komorebi P. *Stratified Recursive Improvement Closure: Obstruction, Descent, Transport, and Evidence-Preserving Lineages*. Formal theory draft v2.4, Block 23, 3 September 2026. The reviewed source is identified in the package source map. Cited for theoretical structure, not as empirical evidence for this manuscript's architecture.

[3] Komorebi P. *Neuralese*. Operation Komorebi, Final Research Edition v1.0, 2026. Section 12 specifies the proposed semantic and continuation interface.

[4] Komorebi P. *Operator-Symbolic Language*. Operation Komorebi, Final Research Edition v1.0, 2026. Explicit semantic and operational interface.

[5] F. Chollet. On the Measure of Intelligence. 2019. https://arxiv.org/abs/1911.01547.

[6] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to learn by gradient descent by gradient descent. 2016. https://arxiv.org/abs/1606.04474.

[7] C. Finn, P. Abbeel, and S. Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *PMLR* 70:1126–1135, 2017. https://proceedings.mlr.press/v70/finn17a.html.

[8] K. Ellis et al. DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning. 2020. https://arxiv.org/abs/2006.08381.

[9] A. Zweiger, J. Pari, H. Guo, E. Akyürek, Y. Kim, and P. Agrawal. Self-Adapting Language Models. 2025. https://arxiv.org/abs/2506.10943.

[10] J. Schmidhuber. Goedel Machines: Self-Referential Universal Problem Solvers Making Provably Optimal Self-Improvements. 2003, revised 2006. https://arxiv.org/abs/cs/0309048.

[11] J. Zhang, S. Hu, C. Lu, R. Lange, and J. Clune. Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents. 2025. https://arxiv.org/abs/2505.22954.

[12] J. Zhang et al. Hyperagents. 2026. https://arxiv.org/abs/2603.19461. Public manuscript accessed 8 October 2026.

[13] D. Srikanth, B. Zhao, D. Xu, Y. Wu, and Z. Jiang. Recursive self-improvement of AI research agents. Preprint, 22 September 2026. https://arxiv.org/abs/2609.26457. The outer-loop comparison is qualified in Section 3.6.

[14] A. Geiger et al. Causal Abstraction: A Theoretical Foundation for Mechanistic Interpretability. *Journal of Machine Learning Research* 26(83):1–64, 2025. https://jmlr.org/papers/v26/23-0058.html.

[15] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth. Preserving Statistical Validity in Adaptive Data Analysis. 2014, revised 2016. https://arxiv.org/abs/1411.2664.

[16] S. R. Howard, A. Ramdas, J. McAuliffe, and J. Sekhon. Time-uniform Chernoff bounds via nonnegative supermartingales. *Probability Surveys* 17:257–317, 2020. https://doi.org/10.1214/18-PS321.

[17] A. Nilim and L. El Ghaoui. Robust Control of Markov Decision Processes with Uncertain Transition Matrices. *Operations Research* 53(5):780–798, 2005. https://doi.org/10.1287/opre.1050.0216. Cited as background on explicit model uncertainty, not as an identical uncertainty model.

[18] A. Solar-Lezama et al. Combinatorial sketching for finite programs. *ASPLOS*, pp. 404–415, 2006. https://doi.org/10.1145/1168857.1168907.

[19] M. Raghothaman and A. Udupa. Language to Specify Syntax-Guided Synthesis Problems. 2014, revised 2016. https://arxiv.org/abs/1405.5590. Specification-language account of the syntax-guided synthesis framework introduced by Alur and colleagues in 2013.

[20] P. Garg, C. Löding, P. Madhusudan, and D. Neider. ICE: A Robust Framework for Learning Invariants. In *Computer Aided Verification*, LNCS 8559, pp. 69–87, 2014. https://doi.org/10.1007/978-3-319-08867-9_5.

[21] M. Willsey, C. Nandi, Y. R. Wang, O. Flatt, Z. Tatlock, and P. Panchekha. egg: Fast and Extensible Equality Saturation. *Proceedings of the ACM on Programming Languages* 5(POPL), 2021. https://doi.org/10.1145/3434304.

[22] C. Nandi, M. Willsey, A. Zhu, Y. R. Wang, B. Saiki, A. Anderson, A. Schulz, D. Grossman, and Z. Tatlock. Rewrite Rule Inference Using Equality Saturation. *Proceedings of the ACM on Programming Languages* 5(OOPSLA), 2021. https://doi.org/10.1145/3485496. Author manuscript: https://arxiv.org/abs/2108.10436.

[23] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. Chapter 1 develops set-cover relaxations and dual bounds. Author-hosted edition: https://www.designofapproxalgs.com/.