

Higher E-Graphs for Futamura Projections: Coherence Obstructions, Descent, and Verified Deployment

Anonymous

October 22, 2026

Abstract

We formulate a proof-relevant semantics of self-applicable specialization by combining higher e-graph proof spaces with the Futamura projections. After aligning compiler endpoints, coherence of independently constructed correctness certificates is the inhabitation of a path space between paths. We identify its fundamental-group obstruction and, when inhabited, its space of choices as a double-loop space. A parameterized example over the circle shows that fiberwise coherence need not globalize. Initiality preserves complete solution families, while descent assembles compatible local witnesses in slices of an infinity-topos. We separate these semantic results from self-application termination, fault attribution, and cost optimization. A finite reproducibility package checks the illustrative examples; it does not implement a self-applicable compiler generator or certify the general theorems in a proof assistant.

1. Problem and contribution

Partial evaluation produces programs whose behavior agrees with a program supplied with some of its inputs. Self-application yields the Futamura projections: residual programs, compilers, and compiler generators [3,4]. A correctness equation does not determine the space of its proofs. A distributed implementation additionally requires compatibility across interfaces, while resource optimization distinguishes implementations with equivalent behavior.

This paper combines the proof-space and initial-functor semantics of higher e-graphs [1] with the specialization and distribution questions of [2]. We give an exact obstruction to comparing independently obtained compiler certificates, identify the space of comparisons when it is inhabited, and construct a parameterized obstruction that survives pointwise solvability. We then formulate initiality and descent as soundness criteria for specification transformations and distributed witnesses. These are applications and combinations of established homotopy and topos theory; no new general cofinality, descent, or Squier theorem is claimed.

The principal result is Theorem 3.2 together with its relative form, Theorem 5.2. The self-generation equation follows conditionally from ordinary partial-evaluation correctness (Proposition 2.2). We distinguish that equation from a claim that all projection constructions stop. We also separate verification from fault attribution and semantic equality from resource cost. The manuscript [1] and proposal [2] are both labelled *Editorially reviewed* by their publisher; [2] explicitly claims no new theorem.

Conventions. Spaces are homotopy types, denoted by $\mathcal{S}$. An infinity-category means an $(\infty, 1)$ -category unless specified otherwise. Products, limits, pullbacks and diagrams are homotopy coherent. Cartesian means finite-product-preserving, not finite-limit-preserving. For paths, $r \cdot s$

means first r , then s . Universes are fixed so that indexing categories are small. All existence assertions about witnesses mean inhabitation, not a specified algorithm for finding them.

2. Partial evaluation and its witness refinement

Definition 2.1 (Admissible self-application). Fix a set C of program representations embedded in a data set D with pairing, an evaluator $E : C \times D \rightarrow D$, and a partial specializer $\mu : C \times D \rightarrow C$. Write $E_p(d) = E(p, d)$ and $E_p(s, d) = E_p(\langle s, d \rangle)$. Whenever $\mu(p, s)$ is defined, require equality of partial functions

$$E_{\mu(p,s)}(-) \simeq E_p(s, -). \quad (\text{PE})$$

Here extensional equality includes the domain of definition and returned values. Let $m \in C$ implement μ : $E_m(p, s) = \mu(p, s)$ as partial functions. Self-application is admissible only for the particular expressions assumed defined below. In a typed implementation these conditions require compatible code encodings and stage types. This definition does not make the evaluator total or computably decide its termination domain.

For an interpreter $i \in C$ and source representation $p \in D$, assume the needed calls exist and put

$$r_{i,p} = \mu(i, p), \quad c_i = \mu(m, i), \quad g = \mu(m, m).$$

Proposition 2.2 (Projections and self-generation). Under Definition 2.1,

$$E_{r_{i,p}}(d) \simeq E_i(p, d), \quad E_{c_i}(p) = \mu(i, p), \quad E_g(i) = c_i, \quad E_g(m) = g. \quad (1)$$

The first equality compares partial computations. The remaining displayed values exist under the stipulated specialization assumptions.

Proof. Apply (PE) to (i, p) , (m, i) , and (m, m) , respectively. The implementation equation for m gives the middle equalities. Finally

$$E_g(m) = E_{\mu(m,m)}(m) = E_m(m, m) = \mu(m, m) = g.$$

Definedness follows because (PE) equates partial functions, including their domains. $\square$

Equation (1) is an extensional consequence, not a new fixed-point theorem. It gives no uniqueness of its proof, runtime bound, or prohibition on further projection schemes; see [4]. If representations are taken modulo renaming, the equalities must consistently use that quotient at every stage.

Definition 2.3 (Witness refinement). A witness refinement supplies a small cartesian infinity-category T for the pure value and constraint theory, chosen expressions representing the relevant operations, and sound correctness paths in its mapping spaces. For specialization it includes a specified lift $\varepsilon_{p,s}$ of (PE), stable under the substitutions being used. Uniform reasoning requires a coherent family of such witnesses. A collection of unrelated pointwise proofs is insufficient. This is additional structure on the extensional setup.

For programs of interface $A \rightarrow B$, put $P_{A,B} = \text{Map}_T(A, B)$. Programs are points, correctness transformations are paths, and comparisons between transformations are paths between paths. They are respectively 1-, 2-, and 3-cells in T . A directed presentation can instead retain noninvertible 2-cells and pass to equality semantics afterward. Partial execution and effects require explicit domain or effect semantics; the category of all partial functions is not being declared cartesian. Categorical models separating programs from evaluations are developed in [7].

For translator $c : C_L \rightarrow C_M$ and semantic maps $e_L : C_L \rightarrow B$, $e_M : C_M \rightarrow B$, the correctly typed certificate is $e_M c \Rightarrow e_L$. In an uncurried presentation it is $\text{eval}_M(c \times \text{id}) \Rightarrow \text{eval}_L$. The semantic maps and their witness spaces must be specified; a raw equality of partial functions does not select a canonical intensional certificate.

3. The obstruction to Futamura coherence

Let Q be a space of compiler expressions, P a space of their semantic expressions, and $e : Q \rightarrow P$ the chosen interpretation. Let $c_2, c_3 \in Q$ denote the compiler obtained directly by F2 and the compiler obtained by evaluating F3 on the interpreter. A witness refinement supplies

$$\psi : c_3 \rightsquigarrow c_2, \quad \kappa_2 : e(c_2) \rightsquigarrow b, \quad \kappa_3 : e(c_3) \rightsquigarrow b,$$

where $b \in P$ is the required behavior. The certificates κ_2, κ_3 may have been constructed independently.

Definition 3.1 (Coherence space). Set $a = e(c_3)$, $\beta = \text{ap}_e(\psi) \cdot \kappa_2$, and

$$\text{Coh}(\kappa_3, \beta) = \text{Path}_{\text{Path}_P(a,b)}(\kappa_3, \beta). \quad (2)$$

Thus a coherence compares parallel certificates after their compiler endpoints have been aligned. Defining $\kappa_3 := \beta$ provides one certificate by transport. Comparing an independently produced κ_3 is a further obligation.

Theorem 3.2 (Existence and choices). For any space P , points a, b , and paths $\alpha, \beta : a \rightsquigarrow b$:

1. $\text{Coh}(\alpha, \beta)$ is inhabited exactly when $[\alpha \cdot \beta^{-1}] = 1$ in $\pi_1(P, a)$.
2. After choosing one coherence Θ , there is an equivalence $\text{Coh}(\alpha, \beta) \simeq \Omega^2(P, a)$. This equivalence depends on the choices.
3. Consequently the set of components of the coherence space, when inhabited, is a torsor for $\pi_2(P, a)$. Choosing Θ identifies it with that group. For $k \geq 1$, its based homotopy groups are $\pi_{k+2}(P, a)$.

Proof. Put $H = \text{Path}_P(a, b)$. Concatenation with α^{-1} on the right identifies H with $\Omega(P, a)$, taking α to the constant loop up to the standard inverse homotopy. The points α, β are in the same component exactly when their difference loop is null-homotopic, proving (1). Given $\Theta : \alpha \rightsquigarrow \beta$ in H , the maps $\gamma \mapsto \gamma \cdot \Theta^{-1}$ and $\ell \mapsto \ell \cdot \Theta$ identify $\text{Path}_H(\alpha, \beta)$ with $\Omega(H, \alpha)$. The preceding equivalence identifies this loop space with $\Omega^2(P, a)$. Taking homotopy groups proves (3); changing the chosen coherence translates the component identification. $\square$

Corollary 3.3. If P is a 1-type, an inhabited coherence space is contractible. If $P = K(A, 2)$ for a nonzero abelian group A , the coherence space from the constant path to itself is the discrete space A . Thus existence need not imply a unique component of choices.

Example 3.4 (The unfilled diamond). Take a graph with vertices s, u, v, t and edges $s \rightarrow u \rightarrow t$, $s \rightarrow v \rightarrow t$. Its realization is a circle. Its two routes from s to t determine a nontrivial difference loop, although all vertices have the same image in π_0 . Attaching a disk along that loop supplies a comparison. Therefore equality classes and specified correctness edges alone do not force a comparison between their pastings. This is the proof-space distinction of [1, Proposition 5.1], applied to compiler certificates.

Proposition 3.5 (A conditional construction). Suppose a chosen fragment of the directed specialization calculus has a convergent 2-polygraph presentation. Suppose each generating

confluence admits a sound semantic 3-cell with its stated boundary. Then every pair of parallel derivations in its free (2,1)-category admits a semantic comparison generated by those cells.

Proof. Squier completion supplies a homotopy basis for a convergent 2-polygraph [6, Theorem 4.3.2]. Every parallel pair is connected by a composite of generating 3-cells, inverses and whiskerings. Interpret these operations and the specified sound generators in the target. $\square$

This criterion requires a presentation covering the relevant substitution and structural equations; it is not automatically a theorem for arbitrary cartesian term rewriting. Convergence and sound semantic fillings must be established. Higher uniqueness requires higher cells. Nor do the labels F2 and F3 alone define a pseudomonad: unit, multiplication, their types and the requisite coherence must first be supplied.

4. Initiality and complete solution spaces

Let $\mathcal{X}$ be an infinity-topos and $M : T \rightarrow \mathcal{X}_U$ a product-preserving model. A cartesian term diagram $D : J \rightarrow T$ has solution family

$$\text{Sol}_M(D) = \lim_{j \in J} MD(j). \quad (3)$$

It can include candidate code, values, correctness paths and boundaries between those paths. Specifying which data are in D determines what its solutions certify.

For $u : J \rightarrow K$, use the comma category $J_{/k}$ of arrows $u(j) \rightarrow k$. Write $|A| = \text{colim}_A *$. Call u initial if $|J_{/k}| \simeq *$ for every k , and n -initial if $\tau_{\leq n}|J_{/k}| \simeq *$.

Theorem 4.1 (Specification invariance). If $D \simeq D'u$ and u is initial, then $\text{Sol}_M(D') \rightarrow \text{Sol}_M(D)$ is an equivalence. For $n \geq 0$, n -initiality is equivalent to preservation of limits under restriction for every $H : K \rightarrow \mathcal{S}_{\leq n}$; it also implies this preservation for diagrams of n -truncated objects in $\mathcal{X}_U$.

Proof. For initial u , apply the initial-functor theorem to each generalized-point functor $\text{Map}_{\mathcal{X}_U}(V, -)$; these preserve limits and jointly detect equivalences. For the truncated assertion, in $\text{Fun}(K, \mathcal{S}_{\leq n})$ the left Kan extension of 1_J is $Y_n(k) = \tau_{\leq n}|J_{/k}|$. Restriction on limits is

$$\text{Map}(1_K, H) \longrightarrow \text{Map}(Y_n, H).$$

By Yoneda this is an equivalence for all H exactly when $Y_n \rightarrow 1_K$ is an equivalence. Generalized points prove the implication for slices. $\square$

This is [1, Theorems 4.1 and 4.3] specialized to compiler specifications, with the standard cofinality background of [5, Section 4.1]. No claim is made that an arbitrary correct compiler transformation is initial. A compiler application must construct the diagram comparison and verify its comma categories.

For a cartesian $F : C \rightarrow D$, [1, Theorem 3.1] supplies

$$C \xrightarrow{\eta_F} \int_D Y_F \xrightarrow{p_F} D, \quad Y_F(d) = |C_{/d}|. \quad (4)$$

Here η_F is initial and p_F is the left fibration classified by Y_F . This retains spaces of represented presentations over d , rather than just components. Its cartesian restriction uses product-preserving left Kan extension. The unique-up-to-contractible-choice factorization need not have contractible fibers. Neither (4) nor Theorem 4.1 supplies a terminating saturation algorithm.

5. Parameterized coherence and descent

Definition 5.1 (Relative coherence object). Let $P \rightarrow U$ be an object of $\mathcal{X}/U$ and let a, b be sections. Form the relative path object

$$H = U \times_{P \times_U P} P,$$

using (a, b) and the diagonal. For sections $\alpha, \beta : U \rightarrow H$, put

$$W = U \times_{H \times_U H} H. \quad (5)$$

Its sections are coherent comparisons of α and β over the entire parameter object. These are homotopy pullbacks, not set-theoretic equalizers.

Theorem 5.2 (Base change and descent of coherence). For every $f : V \rightarrow U$, f^*W is the relative coherence object of the pulled-back data. If $q : U' \rightarrow U$ is an effective epimorphism with Čech nerve $U'_\bullet$, then

$$\mathrm{Map}_{\mathcal{X}/U}(U, W) \simeq \lim_{[k] \in \Delta} \mathrm{Map}_{\mathcal{X}/U}(U'_k, W). \quad (6)$$

For $\mathcal{X} = \mathcal{S}$, the fiber at u is the coherence space of Theorem 3.2 for the corresponding fiber data. The same statement holds after applying any point's inverse-image functor.

Proof. Pullback preserves the finite limits defining (5), giving base change. Effectiveness means that U is the colimit of its Čech nerve, also in the slice. Mapping into W sends this colimit to the limit in (6). Pulling back to a point gives the iterated path space of Theorem 3.2. $\square$

Proposition 5.3 (Pointwise coherence need not globalize). There exist fiberwise inhabited coherence objects over $U = S^1$ with no global section, although sections exist after restriction to contractible open arcs.

Proof. Work in spaces. Set $P = U \times K(\mathbb{Z}, 2)$ and let $a = b$ be the constant basepoint section. Then $H \simeq U \times K(\mathbb{Z}, 1) \simeq U \times S^1$. Choose α to be the graph of id_{S^1} and β the graph of a constant map. Every fiber of (5) is a path space between two points of the connected circle, hence inhabited. A section of W would be a homotopy from id_{S^1} to a constant map, contradicting their actions on π_1 . Both maps are homotopic on each contractible arc. Thus local sections exist, but no compatible complete descent datum can exist. $\square$

This also explains why proving a compiler comparison separately for each input or environment does not prove a coherent family of comparisons. One must work in the appropriate parameter slice or space of structured transformations.

6. Verified distribution and guarded execution

Choose a site (N, J) of contexts with restriction maps and a genuine Grothendieck topology. Set $\mathcal{X} = \mathrm{Sh}_\infty(N, J)$. A family $Z \rightarrow U$ may encode local executions, boundary values, and correctness witnesses for a residual program. Theorem 5.2 applies to its coherence constraints, while the same descent proof applies directly to Z .

Corollary 6.1. Local certified executions with a complete compatible descent datum determine a global certified solution. Equivalences of solution families can be checked after effective-epimorphic pullback. Initial specification transformations preserve these families by Theorem 4.1.

Proof. Apply (6) to Z and use descent for objects, which makes such pullback conservative. Then apply Theorem 4.1. $\square$

Certificate soundness is a separate hypothesis. An accepted attestation, agreement certificate, or probabilistic proof must be related to the required computation under its actual trust or error model. Sheafification alone does not prove that relation. Nor does a quorum cardinality alone prove stability under base change and composition of covers.

For guarded execution, require the family of guards to cover the parameter object and provide compatible semantic and state-transfer data on overlaps. In a general topos a guard need not have a Boolean complement. A fallback case must therefore be specified as part of a covering or operational construction. Descent proves compatible behavior; it does not synthesize a scheduler or a runnable state-transfer implementation.

Proposition 6.2 (Obstruction is not attribution). Failure of global constraint satisfaction, without further trusted observations, does not identify a dishonest participant.

Proof by counterexample. Let honest components report their local binary relations $x = y$, $y = z$, and $x \neq z$. Each relation is satisfiable and restricts to the full set $\{0, 1\}$ on either single variable. No joint assignment satisfies all three. Thus locally nonempty constraints with agreeing marginal supports can be globally inconsistent with no dishonesty. This is not a compatible family of chosen sections of a sheaf; it is the absence of such a choice. $\square$

The connection to contextuality [8] is an obstruction analogy. Fault attribution additionally requires a model connecting observed contradictions to identifiable actions. The counterexample distinguishes marginal compatibility from the full descent datum required by Corollary 6.1.

7. Cost on presentations

Let R be a set of implementation records, $\sigma : R \rightarrow P$ their semantic interpretation, and $b \in P$ a specification. A certificate can be a path $\sigma(x) \rightsquigarrow b$ together with further required data. Given cost $c : R \rightarrow [0, \infty]$ and latency $\ell : R \rightarrow [0, \infty]$, define

$$\inf\{c(x) : x \in R, \text{Cert}(x, b) \neq \emptyset, \ell(x) \leq L\}. \quad (7)$$

The infimum need not be attained or computable. Cost lives on implementation records; it is not assumed invariant under semantic equivalence.

Proposition 7.1. For $\rho : R \rightarrow Q$, a cost factors through its image $\rho(R)$ exactly when it is constant on each fiber of ρ . Moreover, an exactly additive nonnegative path cost on a groupoid assigns zero to every arrow.

Proof. Fiberwise constancy is precisely the condition that $\bar{c}(\rho(x)) = c(x)$ be well defined. For a groupoid arrow r , exact additivity gives $0 = c(1) = c(r) + c(r^{-1})$; nonnegativity forces both terms to vanish. $\square$

The second assertion concerns exact additivity, not arbitrary lax cost models. It explains why directed migration costs should not be identified with invertible semantic certificates. Lawvere enrichment may model directed distance, but does not make all semantically equivalent implementations zero-cost deployments.

Proposition 7.2 (Correctness gives no speedup bound). Under quotation facilities, the wrapper specializer

$$\mu_{\text{wrap}}(p, s) = \ulcorner d \mapsto E_p(s, d) \urcorner$$

satisfies (PE) while retaining interpretation of p at residual runtime.

Proof. Executing the wrapper invokes the evaluator on the captured p, s and dynamic d , giving exactly the right-hand side of (PE). No interpretive work has been eliminated by this construction. $\square$

If specialize-then-distribute and distribute-then-specialize yield semantic expressions a_{SD}, a_{DS} with certificates $\eta_{SD} : a_{SD} \rightsquigarrow b$ and $\eta_{DS} : a_{DS} \rightsquigarrow b$, their semantic comparison is $\eta_{SD} \cdot \eta_{DS}^{-1}$. A syntactic rewrite realizing it requires completeness or an explicit derivation in the chosen calculus. Their cost difference has no predetermined sign. Coherence of independently produced comparisons is again governed by Theorem 3.2.

8. Evidence, scope and research obligations

The accompanying reproducibility package contains exact finite checks for the unfilled and filled diamond, a toy arithmetic specializer, the inconsistent binary constraints, and equivalent expressions with different operation counts. They validate the examples, not the general theorems. The toy specializer does not implement F2/F3 or a self-applicable compiler generator.

The remaining constructive task is to exhibit a typed self-applicable witness calculus, prove its operational soundness, identify the generating confluences for a usable fragment, and check initiality for concrete specification transformations. A complete compiler, higher saturation procedure, dependent-syntax completeness theorem, global optimizer, or Byzantine fault-localization theorem is not supplied. The topological examples delimit exactly what the stated hypotheses support.

Authorship and automation. Anonymous, at the commissioning researcher’s request. An AI assistant assisted with formulation, literature checking, proofs, typesetting and finite-example code. The researcher authorized preparation and submission; independent human verification of every proof is not asserted. No proof-assistant certification or independent referee review is claimed. Editorial and publication status are determined by the journal.

References

1. Anonymous. *Higher E-Graphs: Cartesian Factorization, Proof Spaces, and Topos Semantics*. Higher Categorical Semantics, 2026. Editorially reviewed manuscript. [Article](#).
2. Third Projection Collective. *Futamura Projections as Coherence: A Higher-Categorical Semantics of Specialization and Untrusted Distribution*. Higher Categorical Semantics, 2026. Editorially reviewed proposal. [Article](#).
3. Y. Futamura. *Partial Evaluation of Computation Process—An Approach to a Compiler-Compiler*. Higher-Order and Symbolic Computation 12, 381-391, 1999; reprint of the 1971 paper. [DOI](#).
4. R. Gluck. *Is there a fourth Futamura projection?* PEPM 2009, 51-60. [Publisher identifier and author record](#). DOI: 10.1145/1480945.1480954.
5. J. Lurie. *Higher Topos Theory*. Princeton University Press, 2009, Sections 4.1 and 6.1-6.2. [Text](#).
6. Y. Guiraud and P. Malbos. *Polygraphs of finite derivation type*. Mathematical Structures in Computer Science 28(2), 155-201, 2018, Theorem 4.3.2. [Author version](#).
7. D. Pavlovic. *Monoidal computer I: Basic computability by string diagrams*. Information and Computation 226, 94-116, 2013. [Author version](#).
8. S. Abramsky and A. Brandenburger. *The sheaf-theoretic structure of non-locality and contextuality*. New Journal of Physics 13, 113036, 2011. [Author version](#).

Appendix A. Typed interfaces and the scope of self-application

For a typed language, the interface of specialization should be displayed as a family

$$\mu_{S,A,B} : \text{Code}_L(S \times A, B) \times S \rightharpoonup \text{Code}_L(A, B).$$

The code object describes representations of programs; it need not be the exponential of the denotations. A semantic partial operation can be presented by a domain inclusion and a map

$$\text{Adm}_{S,A,B} \hookrightarrow \text{Code}_L(S \times A, B) \times S, \quad \hat{\mu}_{S,A,B} : \text{Adm}_{S,A,B} \rightarrow \text{Code}_L(A, B).$$

This presentation does not make membership in the domain decidable. It also does not give a total evaluator for all programs. The specialization witness lives over this domain and relates the behavior of the residual code to the behavior with the static argument fixed. The intended semantics of partial functions or effects determines what equality of those behaviors means.

An interpreter and a specializer must themselves be represented at suitable code interfaces before they can be supplied to specialization. Equation (1) uses an external common encoding to keep these indices legible. A typed implementation must exhibit the embeddings, quotation maps, pairing conventions and stage indices that make each application legal. A universal code representation within an untyped language and a universe of all semantic objects are different constructions. The former does not assert the latter.

This separation also prevents a common proof-relevance mistake. A discrete set of source strings has only discrete identity types. Nontrivial rewrite paths belong to a separately constructed semantic or syntactic witness space; they are not literal identities between distinct strings. The map from representations to that space is part of the refinement. An optimizer can retain a concrete representative and its cost while interpreting it in a higher equality space.

For a family of interpreters or interfaces, coherence must be established in a space of structured families, or in the relevant slice, rather than inferred from its evaluation at each point. Such spaces can themselves be limits enforcing naturality and substitution. Proposition 5.3 demonstrates the logical necessity of this distinction without depending on any programming-language implementation.

Appendix B. Geometry of the obstruction and its finite check

Theorem 3.2 involves two different questions. First, the loop $\alpha \cdot \beta^{-1}$ must have trivial class in $\pi_1(P, a)$. Second, after a filling has been chosen, the other fillings form a space equivalent to $\Omega^2(P, a)$. Killing a particular fundamental-group element therefore does not force all higher choices to be contractible. For $P = K(\mathbb{Z}, 2)$ there is no fundamental-group obstruction, while the components of fillings form a copy of $\mathbb{Z}$ after a base filling is selected.

In the diamond example, order vertices as (s, u, v, t) and edges as (su, ut, sv, vt) . The cellular boundary is

$$\partial_1 = \begin{pmatrix} -1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

It has rank three over $\mathbb{Q}$. The cycle representing the difference of the two routes is

$$z = (1, 1, -1, -1)^\top, \quad \partial_1 z = 0.$$

Before a face is attached, the first Betti number is $4 - 3 = 1$. After attaching one face with boundary $\partial_2(1) = z$, it becomes $4 - 3 - 1 = 0$. The exact-rational code verifies these ranks and the chain identity $\partial_1\partial_2 = 0$.

The proof about the fundamental group does not rely on inferring it from first homology in general. Here the graph is explicitly homeomorphic to a circle, and filling its boundary once gives a disk. In other examples, vanishing first homology would not suffice to prove that a proof loop is null-homotopic. A verifier claiming full coherence must check the actual requested higher cell or use a sound presentation theorem.

The parameterized example in Proposition 5.3 has a related geometric description. Up to homotopy, $W \rightarrow S^1$ is the fibration of paths ending at a fixed point of the circle, with projection given by their initial point. Its total space is contractible, its fibers have components indexed by $\mathbb{Z}$, and a section would contract the circle. Local path choices on arcs differ by winding numbers on overlaps. This is an obstruction to choosing the fillings coherently; it is not an obstruction to filling any one selected fiber.

Appendix C. A finite specialization example

The executable example uses integer expressions generated by

$$t ::= n \mid s \mid d \mid (t + t) \mid (t \times t).$$

Here s is static and d dynamic. The evaluator recursively computes integer values. The specializer substitutes a literal for s , retains d , and folds a binary operation exactly when both specialized operands are literals. A dynamic residual expression may subsequently be evaluated with any value of d .

Proposition C.1. For every expression t and integers s, d , this specializer satisfies

$$\text{eval}(\text{spec}(t, s), d) = \text{eval}(t, s, d).$$

Proof. Induct on t . Literal and variable cases follow from the definitions. For an addition or multiplication, the induction hypotheses identify the values of the specialized operands. If both are literals, constant folding returns their corresponding operation. Otherwise residual evaluation performs that same operation. These cases exhaust the grammar. $\square$

The finite checks evaluate seven structurally different expressions over all static and dynamic inputs in $\{-3, -2, -1, 0, 1, 2, 3\}$. They compare original and residual values and verify that the residual contains no occurrence of s . This is 343 original/residual comparisons. Exact arithmetic avoids floating-point equivalence assumptions.

This example is deliberately small. Its specializer is an external Python procedure; the object language cannot represent that procedure. Consequently these tests establish no executable F2/F3 instance or self-application claim. Proposition 2.2 is a conditional mathematical statement about an implementation satisfying Definition 2.1, not an empirical claim about this toy language.

For cost, take $t_1 = d + 1$ and $t_2 = (d + 0) + 1$. Their values agree for all integer inputs, while a direct evaluator performs one and two arithmetic operations respectively. The code checks this difference. It also checks an expression with static arithmetic for which specialization removes operations, and verifies that a wrapper evaluator retains the original expression's operations. The example demonstrates why specialization correctness and useful optimization require separate evidence.

Appendix D. Inconsistent supports and reproducibility

The support counterexample enumerates all eight assignments to $(x, y, z) \in \{0, 1\}^3$. The relations on contexts $\{x, y\}$, $\{y, z\}$, and $\{x, z\}$ are respectively equality, equality, and inequality. Each has two local assignments, and its projection to either one-variable overlap is $\{0, 1\}$. The intersection of all three lifted relations on the joint context is empty.

No chosen assignment on every pair can agree on all overlaps. If there were such a family, the ordinary sheaf of assignments would glue it to a triple satisfying all relations, contradicting the enumeration. Thus agreeing marginal supports and compatible selected sections must not be conflated. The same logical distinction appears in the contextuality literature [8], although this example is used solely as a finite constraint countermodel.

The publication package includes `checks.py`, `checks-result.json`, the full manuscript in Markdown and LaTeX, the journal-length version, PDF, citation metadata, build instructions, and a SHA-256 manifest. Run the example checks with Python 3.10 or newer:

```
python checks.py
```

The script uses only the Python standard library and exits with an error if an assertion fails. Its JSON output reports cellular ranks, specialization comparison counts, local-support and global-assignment counts, and cost comparisons. Rebuilding the PDF requires Pandoc, a LaTeX installation with the listed packages, and `make`; the mathematical checks do not require those tools. PDF timestamps can differ between builds, so reproducibility of the content and example results is distinguished from byte-for-byte PDF reproduction.

The source papers' supplied citation metadata are retained in `references.json`. No private submission receipt, session credential, contact address, or editorial correspondence is part of the public reproducibility package.