RISU TECHNICAL NOTE 2026-04

Projection Assurance

Consequence-Relevant Semantics Across Agent-Facing Interfaces

Moon Lee · RISU Institute · August 2026

DOI: 10.5281/zenodo.22149639

ABSTRACT

Agent-facing tools routinely project operations from APIs and SDKs into smaller callable surfaces. Surface retention is a poor proxy for consequence preservation: relevant state can remain visible without governing the effect, while a source control can disappear and still be preserved through a narrower target policy. Projection Assurance evaluates this boundary relative to a declared source consequence. Correspondence licenses the source-target comparison. Discrimination uses deterministic factorization to test whether the target representation retains enough information to determine the consequence. Operative Placement is a structural placement certificate for a recognized discharge at the declared consequence-governing cut. Exact Realization asks whether the operative information and mechanism semantics reproduce the source consequence. Coverage remains a separate question of represented capability breadth. The evaluated artifact preserves the distinction among source-derived facts, analytical premises, and diagnostic target models. On declared deterministic profiles grounded in pinned source code, a GitHub merge path retains head identity but fails to bind the reviewed head into the expected-head guard; the studied Azure Blob path omits general overwrite choice yet realizes the declared create-if-absent consequence through If-None-Match:*; and a prospectively frozen PayPal holdout shows that an effect-side idempotency header can still be too coarse when its value is static rather than logical-operation specific, yielding a minimal counterexample and, within the predeclared repair vocabulary, a singleton repair. Consequence-blind signature reconstruction, structured-source cross-checks, semantic mutations, and author-controlled fresh-process replay test the inference chain without retuning the frozen semantic core. The method provides a bounded test, tied to retained source evidence and explicit semantic premises, of which consequence-relevant distinctions survive projection and whether the declared mechanism preserves them at the effect cut.

Keywords: semantic preservation; interface adaptation; agent tools; translation validation; conditional operations; binding time; runtime guards; evidence-carrying assurance; Model Context Protocol

1. Introduction

Agent-facing tools rarely expose consequential source operations unchanged. An API or SDK call may be wrapped, narrowed, or re-expressed through a smaller callable surface, with arguments fixed, omitted, reconstructed, or replaced by higher-level policy. These changes can weaken an effect condition or preserve it through another mechanism. Projection Assurance therefore asks whether the target preserves the distinctions that determine the declared consequence and keeps them connected to the effect.

Some consequence-relevant distinctions are fixed outside the interface itself. A reviewed revision, an organizational policy, or the identity of a logical operation may enter software as a precondition or effect constraint before an agent-facing projection is formed. Projection Assurance takes that declared consequence as part of the profile and tests whether the projection preserves it.

The GitHub case shows why visibility is insufficient. GitHub supports an expected-head SHA precondition: when supplied, the pull-request head must still match for merge to proceed [22], and the CLI exposes the same idea as --match-head-commit [23]. In the retained github/github-mcp-server direct merge path, the reviewed head remains observable elsewhere, yet that earlier witness is not bound into the source merge request [27]. The missing relation is the review-to-request binding.

Azure provides the converse case. The retained Microsoft MCP blob-upload operation narrows the source write family to create-if-absent and calls BlobClient.UploadAsync with overwrite=false [27]. The Azure SDK lowers that choice to If-None-Match: *, Microsoft documents the Boolean overload as creating a new block blob or failing when one already exists [24], and HTTP supplies the wildcard condition's effect-time semantics [14]. The target exposes less caller choice while preserving the declared create-only consequence at the write effect.

Together, the cases show that neither surface visibility nor surface omission is a sound preservation rule: a consequence-relevant distinction may remain visible without governing the effect, while a caller-visible source control may disappear when the target independently fixes a narrower scope and binds the remaining condition at the effect.

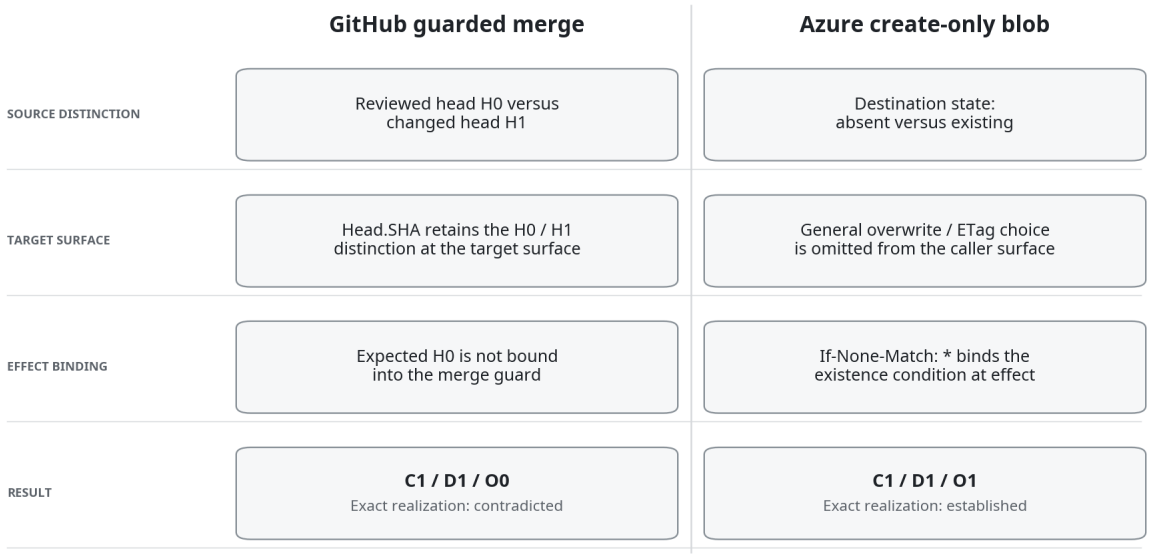

Figure 1. Canonical semantic contrast. GitHub retains the reviewed-head distinction at the target surface but leaves expected H_0 unbound at the merge guard. Azure omits the general overwrite/ETag control from the caller surface while binding the existence condition at effect through If-None-Match:*.

The framework separates five questions. Correspondence (C) records whether retained evidence supports the source-target mapping for a declared effect scope. Discrimination (D) tests whether the declared target-side representation retains enough distinction to determine the source consequence on the bounded profile. Operative Placement (O) is a structural placement certificate for a recognized discharge at the declared consequence-governing cut; the frozen v0.2.0 software retains the internal field name operativity for artifact compatibility. Exact Realization requires the information available to the discharge to determine K and the discharge to implement the required consequence mapping. Coverage concerns represented capability breadth.

This decomposition also identifies what a projection may safely remove. Once the operative signature merges realizations with different declared consequences, deterministic post-processing over that same information cannot recover the distinction [29], [30]. A source dimension can disappear when the target independently fixes a narrower scope and the remaining representation still determines the consequence; this is consequence-relative specialization, analogous to binding-time separation [12]. Invocation-specific witnesses such as reviewed heads, expected revisions, or ETags must survive, be reacquired, or be revalidated while they can govern the action. The underlying determinacy criterion follows established dependency and view-determinacy formulations [28], [31].

The empirical task is to ground those declarations in retained source evidence. For the reported evaluation, retained artifacts support observed facts; an Assurance Contract supplies the consequence lens, discriminator, Boundary, and other analytical premises. In the evaluated v0.2.0 artifact, both compile into Projection IR, with analyst-supplied target outcomes excluded from C/D/O. Evaluation tests distinct links in that chain through dual implementations, structured extraction, conformance, metamorphic and mutation tests, diagnostic noninterference, and replay [27]. SG-001 reconstructs the canonical discriminator signatures without reading the desired consequence table, and ER-001 derives mechanism semantics before K enters the outer equality check. Once qualified, the machinery is frozen and applied prospectively to a prelocked PayPal projection; ASF-001 and RR-001 add Azure extraction diversity and author-controlled same-host replay across fresh process environments. The Evaluation Capsule [27] preserves the publication-facing records and bindings.

The foundations span several mature literatures. Behavioral compatibility and adaptation already exceed signature or surface parity [1]-[6]. Semantic-service work separates invocation mechanics from service meaning, including explicit preconditions and effects [7], [8]. Partial evaluation distinguishes information fixed during specialization from residual inputs [12]; TOCTOU analysis and HTTP conditional semantics show why mutable conditions may require later validation [13], [14]. Agent-tool research addresses protocol semantics, description-implementation consistency, and downstream control [17]-[21]. Projection Assurance uses these established ideas at a concrete assurance boundary: for one declared source consequence and scope, it tests whether a projection retains a sufficient distinction, whether that distinction reaches and is correctly realized at the effect, and what evidence supports the resulting claim.

C1/D1/O1 therefore yields the v0.2.0 structural certificate for a declared profile. Behavioral equivalence, authorization correctness, and runtime safety require additional evidence; downstream policy may still constrain a projected capability even after the projection has weakened the source semantics it represents.

2. Projection Assurance: Consequence Sufficiency and Effect Realization

Projection Assurance separates information sufficiency from effect realization. Discrimination requires the declared target-side representation to determine the source consequence K. Exact operative realization additionally requires the modeled discharge, using the consequence-relevant information available on its effect path, to induce K on every admissible realization. Projection IR v0.2.0 checks only the O placement certificate, so D1 may coexist with O0 and O1 may coexist with D0.

The same determinacy relation appears at both levels. On a finite relation it is a functional dependency; for deterministic information it is equivalently partition refinement and factorization [28]-[30]. At the representation level, the semantic discriminator must determine K. At the discharge level, the operative information must determine K and the mechanism must assign the required consequence. This distinguishes loss of information from misuse of information that was preserved.

2.1 Declared Problem, Evidence Premises, and Correspondence

A declared semantic profile identifies a source operation family F, an in-scope slice $\Omega \subseteq F$, a projected target operation T, a Boundary B, a consequence lens K, a semantic discriminator d, and a discharge G. A separate evidence record E supports source-target correspondence and the interpretations assigned to d and G. Keeping E outside the semantic objects preserves the distinction between mathematical premises and evidence provenance.

The Boundary determines a nonempty admissible realization domain R and an effect cut. A realization may be a state or a bounded history; histories are used when an earlier observation, an intervening transition, and the later effect must remain distinct within one realization. The reference analyzer evaluates a finite declared R, while Section 2.3 introduces the temporal facts used by the v0.2.0 placement certificate.

$$K : R \to C$$

is the declared source consequence lens, with C its codomain for the preservation claim. K abstracts from source behavior outside that claim. Coarsening K merges consequence classes and asks the projection to preserve a weaker distinction.

R, Ω, K, and the semantic interpretation of d are fixed before evaluation, as are the discharge premises introduced later. A result-guided change creates a new profile. The same profile cannot obtain a favorable verdict by deleting a counterexample realization, redefining d from K, or merging consequence classes after failure.

Formal mathematics alone cannot establish that declared discriminator values or the semantics attributed to G faithfully represent the target. Projection IR v0.2.0 therefore accepts discriminator.signature_by_world as an Assurance Contract input supported by source-derived interpretation evidence. For the two canonical cases, SG-001 adds an export qualification: evidence-linked, consequence-blind rules reconstruct the signatures from declared world-state records and must agree exactly with the compatibility table before D is exported. Discharge placement remains evidence-grounded, while Boundary transition assumptions remain declared. Section 3 records these provenance classes explicitly.

Correspondence licenses the comparison. C1 means retained evidence establishes the mapping for the declared effect scope; C0 means the mapping is unestablished or contradicted; CU means unresolved. C is an empirical entry condition. The kernel exports a projection claim only when C1 holds.

2.2 Deterministic Consequence Sufficiency

If two realizations receive the same representation value, any deterministic computation limited to that representation must treat them identically. A representation that merges realizations with different declared consequences cannot support exact consequence preservation.

Definition 1 - Deterministic Refinement

For maps $f : R \to X$ and $g : R \to Y$, write

$$f \succcurlyeq_R g$$

exactly when

$$f(r_1) = f(r_2) \quad \text{implies} \quad g(r_1) = g(r_2) \quad \text{for all } r_1, r_2 \text{ in } R.$$

Thus $f \succcurlyeq_R g$ means that f is at least as discriminating as g on R: every fiber of f lies within a fiber of g. The relation is reflexive and transitive. Mutual refinement gives the same partition of R; modulo that equivalence, $\succcurlyeq_R$ is the usual refinement order on induced partitions. On a finite relation this is the functional dependency $f \to g$ [28], with equivalent partition and post-processing interpretations for deterministic information [29], [30].

Lemma 1 - Factorization Characterization

For $f : R \to X$ and $g : R \to Y$, $f \succcurlyeq_R g$ if and only if there exists a unique function

$$h : \mathrm{im}(f) \to Y$$

such that

$$g = h \circ f.$$

Proof. If $g=h \circ f$, equal f-values immediately give equal g-values. Conversely, suppose $f \succcurlyeq_R g$. For each x in im(f), choose any r with $f(r)=x$ and define $h(x)=g(r)$. Definition 1 makes h well-defined because all realizations in the same f-fiber have the same g-value. Every x in im(f) is realized, so this value is also forced; hence h is unique on im(f). QED.

Call r_1,r_2 consequence-divergent when $K(r_1) \neq K(r_2)$. A representation is consequence-sufficient exactly when it separates every consequence-divergent pair. Let the declared target-side discriminator be

$$d : R \to \Sigma.$$

Definition 2 - Discrimination

$$D1 \quad \Leftrightarrow \quad d \succcurlyeq_R K. \qquad \text{Otherwise D0.}$$

By Lemma 1, D1 holds exactly when there is a unique consequence decoder

$$\kappa_d : \mathrm{im}(d) \to C$$

such that

$$K = \kappa_d \circ d.$$

D1 establishes one exact, model-relative fact: K is deterministically recoverable from the declared d on R. Effect-path availability and implementation of κ_d are separate questions. Likewise, D0 for the declared d leaves open the possibility that an undeclared target representation realizes K.

A D0 witness is any consequence-divergent pair with $d(r_1)=d(r_2)$. On the frozen v0.2.0 software release's finite scalar signature surface, the Python and Node analyzers implement Definition 2 by exhaustive pair comparison [27]. Structured JSON values are permitted by the broader schema, but cross-runtime equality for that surface is not yet normalized; executable exactness is therefore limited to the conformance regime in Section 2.6.

If d and K induce the same partition of R, d is consequence-exact. A strict refinement overdiscriminates but remains sufficient. That distinction may matter for interface minimization or privacy without changing D1.

2.3 Operative Realization: What the Effect Path Implements

Discrimination concerns the declared semantic representation. Execution concerns the information that can influence the discharge and the consequence relation induced by that discharge. Treating the two as distinct avoids mistaking semantic availability for effect control.

Definition 3 - Deterministic Discharge Model and Exact Realization

For a declared deterministic discharge G, let

$$z_G : R \to Z_G$$

be the modeled operative signature: the tuple of modeled values whose variation may affect the consequence class induced by G at the relevant effect decision. It may include a caller-carried witness, state read internally at the effect, or information supplied by a qualified environmental controller. It excludes K and the selected consequence itself. Completeness of z_G with respect to the real mechanism is an external model-adequacy premise.

Let

$$\alpha_G : \mathrm{im}(z_G) \to C$$

be the declared consequence interpretation of the modeled discharge: for each operative signature, α_G assigns the source-consequence class induced by the modeled target effect. α_G is a semantic abstraction of the target mechanism and need not be a runtime function that emits labels in C. Its meaning must be grounded in target-mechanism evidence under an abstraction that does not consult K or declared_target_model_by_world. Here, independent means input-independent from the per-world K table and analyst target model; it does not imply independent authorship, a separate runtime oracle, or live-system validation. Without that grounding, $\mathrm{Realize}_R(G,K)$ remains unestablished.

The discharge model realizes the declared consequence exactly on R when

$$\mathrm{Realize}_R(G,K) \quad \Leftrightarrow \quad K = \alpha_G \circ z_G.$$

This equality is the exact deterministic effect-level target for the declared discharge model. It is stronger than D1 and the v0.2.0 Operative Placement certificate (O). ER-001 evaluates this stronger target for the two canonical cases as a companion qualification, leaving the Projection IR v0.2.0 structural rules unchanged.

ER-001 totalizes the executable check because a target mechanism may produce a native effect outcome with no faithful interpretation in the source consequence codomain C. Let $C^+ = C \uplus \mathrm{OUTSIDE\_C}(M_G)$, where M_G is the carrier-native out-

come space, and let $\iota : C \to C^+$ be the canonical embedding. Define $K^+ = \iota \circ K$. A mechanism evaluator first derives a native outcome from z_G; a grounded interpreter then induces $\alpha_G^+ : \mathrm{im}(z_G) \to C^+$ by mapping that outcome either to its supported class in $\iota(C)$ or to an OUTSIDE_C tag. ER-001 checks $K^+ = \alpha_G^+ \circ z_G$. If α_G^+ ranges entirely over $\iota(C)$, identifying C with its embedded copy reduces the check to Definition 3.

Totalization preserves the equality test while making unmatched outcomes explicit. A target effect outside the declared source classes remains OUTSIDE_C instead of being assigned a favorable C label. Because ι is injective, $z_G \succcurlyeq_R K$ if and only if $z_G \succcurlyeq_R K^+$; Proposition 1 therefore applies unchanged to the totalized check, with required decoder $\iota \circ \kappa_{zG}$ whenever z_G is sufficient. ER-001 also keeps mechanism derivation consequence-blind: Python and Node derivers receive worlds, profile constants, evidence status, mechanism rules, and consequence-class definitions, excluding the per-world K table, analyst target model, and v0.2.0 discriminator compatibility map. K enters only in the outer equality check.

Proposition 1 - Decomposition of Exact Operative Realization

For a declared deterministic discharge model (z_G, α_G), $\mathrm{Realize}_R(G,K)$ holds if and only if both

$$\text{(a)}\quad z_G \succcurlyeq_R K$$

and

$$\text{(b)}\quad \alpha_G = \kappa_{zG} \quad \text{on } \mathrm{im}(z_G),$$

where κ_{zG} is the unique decoder supplied by Lemma 1.

Proof. If $K=\alpha_G \circ z_G$, Definition 1 gives $z_G \succcurlyeq_R K$. Lemma 1 then yields a unique κ_{zG} with $K=\kappa_{zG} \circ z_G$; uniqueness forces $\alpha_G=\kappa_{zG}$ on $\mathrm{im}(z_G)$. Conversely, if (a) holds and $\alpha_G=\kappa_{zG}$, then $K=\kappa_{zG} \circ z_G=\alpha_G \circ z_G$. QED.

Proposition 1 identifies two fixed-domain failure modes. Operative insufficiency occurs when z_G collapses a consequence-divergent pair, so no deterministic branch semantics limited to z_G can realize K on all of R. Operative misalignment occurs when z_G is sufficient but α_G differs from the required decoder. The mechanism can then distinguish enough states while implementing the wrong consequence relation.

Definition 4 - The Projection IR v0.2.0 Operative Placement Certificate

Projection IR v0.2.0 does not construct z_G, infer α_G, or prove $\mathrm{Realize}_R(G,K)$. Its O axis classifies supplied placement facts. The analyzers return O1 when gates_effect=true and the evaluation cut is EFFECT, or when the cut is PRE_EFFECT and the Boundary declares no consequence-material post-check transition. They return O1_REPLACED for a declared ENVIRONMENT gate; O0 when no effect gate is declared, the cut is NONE, or a PRE_EFFECT check is followed by a possible material transition; and OU when the gate or cut is unknown or unsupported [27]. The discharge mode label is retained in the IR but does not affect the v0.2.0 O decision.

O1 is exact only for the v0.2.0 placement rule: the supplied facts match a recognized placement pattern. It neither guarantees nor is required for $\mathrm{Realize}_R(G,K)$. A recognized gate may operate on an insufficient or misaligned condition, while a correct mechanism outside the current placement vocabulary may remain unresolved. ER-001 leaves this rule unchanged and evaluates z_G and grounded mechanism semantics separately for the canonical profiles.

2.4 Irrecoverable Loss and Invocation-Specific Witnesses

Proposition 1 also yields a fixed-domain no-recovery diagnostic: if the modeled operative signature is insufficient, deterministic computation downstream of that signature cannot restore a consequence distinction already collapsed.

Corollary 1 - Post-processing Cannot Restore a Lost Consequence Distinction

If

$$\neg(z_G \succcurlyeq_R K),$$

then for every set U and every deterministic post-processing $u : \mathrm{im}(z_G) \to U$,

$$\neg(u \circ z_G \succcurlyeq_R K).$$

Consequently, no deterministic downstream computation that receives no new consequence-relevant input can realize K on all of R.

Proof. If $u \circ z_G \succcurlyeq_R K$, Lemma 1 gives a decoder h with $K=h \circ u \circ z_G$. Then K is a deterministic post-processing of z_G, so $z_G \succcurlyeq_R K$, contradicting the premise. QED.

This is the deterministic no-recovery property implicit in the same refinement and post-processing order [29], [30]. A prior observation matters only while its consequence-relevant information survives into the operative signature. Once the last bound representation merges admissible realizations requiring different consequences, later computation over that same information cannot separate them. TOCTOU analyses and HTTP conditional requests address related check/use hazards [13], [14]; the result here concerns information loss at a projection boundary.

Proposition 2 - Witness-Blind Operative Signatures

Let admissible realizations be pairs (v,s) in $R \subseteq V \times S$, where v is an invocation-specific expected witness and s is effect-time state. Suppose $z : R \to Z$ is witness-blind: there exists a function $\tilde{z} : S \to Z$ such that $z(v,s)=\tilde{z}(s)$ for every (v,s) in R.

$$z(v,s) = \tilde{z}(s).$$

If there exist v_1, v_2, and $s_\star$ such that $(v_1,s_\star)$ and $(v_2,s_\star)$ are both admissible and

$$K(v_1,s_\star) \neq K(v_2,s_\star),$$

then z is not consequence-sufficient for K.

Proof. The two realizations have the same $s_\star$ and therefore the same z-value, while their declared consequences differ. Definition 1 fails. QED.

When witness and effect-time state range over a common identity set, equality-guarded operations are an immediate instance. If K(v,s)=MATCH when s=v and STALE otherwise, a current-state-only operative signature cannot reproduce the source guard across varying expected witnesses. Preservation can instead carry the witness internally, reconstruct an information-equivalent representation, or use a separately grounded replacement. Section 4 applies this distinction to GitHub without treating one fixed-witness slice as a general impossibility proof.

For fixed R and K, Proposition 1 also fixes the repair obligation. If z_G is insufficient, repair must add a consequence-relevant distinction to the operative signature by carrying a witness, reacquiring state, revalidating later, or using another grounded input. Deterministic post-processing alone cannot suffice. If z_G is held fixed and already sufficient, repair reduces to aligning α_G with κ_{zG}. Restricting R changes the claim instead of repairing the fixed-domain problem; Section 2.5 treats that case.

2.5 Target-Grounded Specialization

Specialization narrows the declared problem instead of repairing the same fixed-domain problem. An arbitrary restriction can remove the pair that witnessed failure, so mathematics alone cannot distinguish a legitimate specialization from result-guided selection. Projection Assurance treats a restriction as target-grounded only when the target operation, configuration, or policy fixes the narrower source slice before evaluation; the resulting realization domain is a nonempty restriction $R' \subseteq R$.

Restriction is monotone in one direction: if $f \succcurlyeq_R g$, the same relation holds after restricting both maps to R'. The converse can fail because R' may omit a pair that defeated sufficiency on R. A passing restricted profile therefore carries no implication for the broader profile.

Definition 5 - Consequence-Sufficient Specialization

For a nonempty target-grounded restriction $R' \subseteq R$, let $d' : R' \to \Sigma'$ be the representation retained by the target. The specialization is consequence-sufficient exactly when

$$d' \succcurlyeq_{R'} K|_{R'}.$$

This condition is necessary and sufficient for deterministic recovery of $K|_{R'}$ from d'. It is a D-level result. Exact effect preservation on the specialized profile additionally requires $\mathrm{Realize}_{R'}(G',K|_{R'})$; Projection IR v0.2.0 exposes only the weaker Operative Placement certificate (O) together with separately grounded mechanism evidence.

Lemma 2 - Eliminating a Specialized Coordinate

Suppose the source representation on R' is $d(r)=(p(r),s(r))$. If p is a deterministic function of s on R', then d and s induce the same partition of R'. A target-fixed constant $p=p_0$ is the simplest case.

Proof. Projection from (p,s) to s makes $d \succcurlyeq_{R'} s$. If $p=h \circ s$ on R', then s reconstructs d as (h(s),s), giving $s \succcurlyeq_{R'} d$. Hence the two representations have the same fibers. QED.

Full information equivalence exceeds what consequence preservation requires. Even if p cannot be reconstructed from s, omitting p is sufficient whenever $s \succcurlyeq_{R'} K|_{R'}$. A target may therefore expose less source state while preserving the declared consequence. Classical partial evaluation similarly specializes computations by fixing information before residual inputs are supplied [12]; the criterion here is consequence-relative rather than whole-program equivalence.

The canonical pair illustrates both cases. Azure's overwrite=false is fixed by the target create-only operation across its declared scope and thus supplies a target-grounded specialization premise. A reviewed GitHub head is invocation-specific; fixing H_0 in one analyzed instance does not specialize the target merge operation to expected_head=H_0. Coverage remains orthogonal because a projection can preserve one specialized consequence while omitting other source capabilities.

Coarsening the consequence lens weakens the claim in another way. If $K_{coarse}=q \circ K_{fine}$, every representation sufficient for K_{fine} is sufficient for K_{coarse}, while the converse may fail. Restricting R or coarsening K can therefore convert a failure into a success without changing the target. Either move is legitimate only when independently motivated, and each creates a new versioned profile.

2.6 Claim Boundary

Section 2 separates three claim layers: exact mathematics over declared functions, the executable Projection IR v0.2.0 structural certificate, and the external premise that those declarations faithfully represent source and runtime behavior. Table 1 records the boundary.

Table 1. Claim boundary: executable scope and remaining external premises.

Claim	Executable / exact scope	Still external
C1	Evidence-backed entry condition. v0.2.0 reads correspondence.status from the Projection IR.	Completeness and adequacy of correspondence evidence.
D1: discrimination	Exact factorization on declared R. v0.2.0 exhaustively compares the finite scalar surface; canonical export also requires SG-001 signature reconstruction.	Adequacy of R, K, declared worlds, and discriminator interpretation; no live-runtime observation.
Operative sufficiency	Necessary and sufficient for a deterministic decoder over the modeled operative signature. Generic v0.2.0 does not compute it; ER-001 reconstructs z_G for the canonical cases.	Completeness of z_G, the Boundary, and fixed coordinates.
Exact realization / ER-001	Definition 3: exact agreement $K = \alpha_G \circ z_G$. ER-001 derives native outcomes consequence-blind and checks the totalized equality $K^+ = \alpha_G^+ \circ z_G$, retaining unsupported native outcomes explicitly.	Adequacy of the mechanism model, consequence classes, worlds, and Boundary; live-runtime conformance.
O1 / O1_REPLACED	The v0.2.0 Operative Placement certificate, derived from gates_effect, evaluation_cut, and the post-check material-transition flag.	Truth of supplied placement and Boundary facts; semantic alignment of the mechanism.
Target-grounded specialization	D is checked on the restricted profile, $d' \succcurlyeq_{R'} K|_{R'}$. v0.2.0 does not infer whether the restriction is legitimate.	Evidence that target semantics fix the narrower scope independently of the verdict.
STRUCTURAL_ASSURANCE_ESTABLISHED	Exactly C1 + D1 + O1/O1_REPLACED. Target-model cross-checks cannot create or change C/D/O.	Source/runtime preservation still depends on Boundary adequacy, discriminator grounding, and mechanism semantics.
coverage_complete	Set equality of the declared source_family and in_scope sets.	Whether the declared source family is exhaustive beyond that scope.

STRUCTURAL_ASSURANCE_ESTABLISHED means that the supplied profile satisfies C1, model-relative D1, and a recognized O1/O1_REPLACED placement pattern. $\mathrm{Realize}_R(G,K)$, general source equivalence, authorization, and runtime safety require stronger evidence. A negative structural label is equally scoped: it records failure on the declared surface without ruling out undeclared compensating mechanisms.

3. Grounding the Projection Claim

Section 2 is conditional on the declared profile. A real-system claim must also justify the connection between that profile and the implementation. Projection Assurance records this grounding separately: retained artifacts support empirical facts, the Assurance Contract declares analytical scope and model, and field-level provenance records how both enter Projection IR.

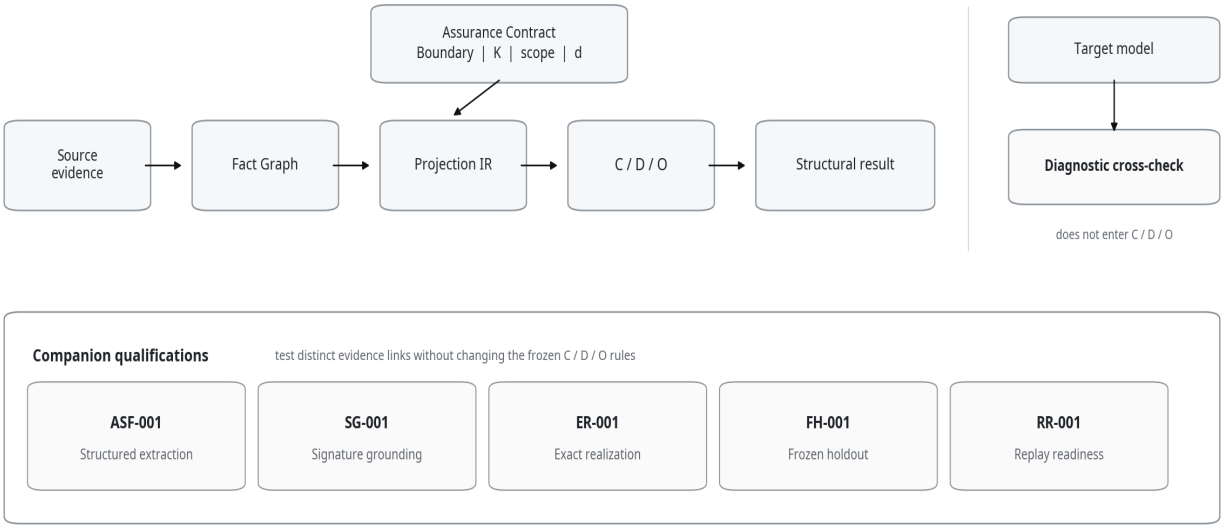

Figure 2. Grounding and qualification architecture. Source evidence and Assurance Contract declarations compile into Projection IR, which yields C/D/O and the structural result. The target model remains isolated in a diagnostic cross-check. ASF-001, SG-001, ER-001, FH-001, and RR-001 test distinct evidence links without changing the frozen C/D/O rules.

The architecture exposes each inferential step to challenge. A reviewer can dispute a source region, extraction or derivation rule, Boundary or consequence declaration, or mechanism interpretation without conflating that objection with the finite factorization result.

3.1 Retained Evidence and Scoped Absence Claims

The frozen v0.2.0 software release records each source region with repository, pinned ref, upstream path, upstream blob identity, local SHA-256, and attested extraction scope [27]. Positive claims can cite an exact retained region. Negative claims add a completeness premise because absence is supported only within the attested scope.

The GitHub claim about missing expected-head binding, for example, is grounded in a retained complete MergePullRequest function block. It does not extend to every GitHub or MCP path. The artifact manifest records the relevant complete function, method, class, or request-construction scope, which therefore bounds the evidence claim.

3.2 Observed Facts, Declared Semantics, and Field Provenance

Primitive facts are local observations anchored to retained bytes, such as a source SHA guard, target merge call, Head.SHA field, Azure UploadAsync(..., false, ...) call, or SDK path producing an IfNoneMatch wildcard. Recorded frontend rules combine them into derived propositions such as correspondence, head observability, direct-guard absence, or effect-guard presence. The Fact Graph preserves these dependencies so each derived proposition can be traced to supporting facts and source regions [27].

The Assurance Contract supplies analytical premises: the Boundary and admissible worlds, consequence lens, in-scope source slice, coverage set, the discriminator-signature compatibility copy required by Projection IR v0.2.0, and the optional target model. An admissible external push $H_0 \rightarrow H_1$ is therefore a Boundary declaration, while an SHA field or missing assignment is source-derived evidence. The separation prevents analytical choices from being presented as observations.

Projection IR preserves that distinction at field level. Evidence-derived fields are marked OBSERVED_DERIVED and linked to fact identifiers; contract fields are marked DECLARED_CONTRACT and linked to the contract digest [27]. SG-001 qualifies the canonical discriminator tables without relabeling them as observed: consequence-blind derivers reconstruct GitHub request-head and Azure existence signatures from declared world records under evidence-linked rules, with K and the target model excluded; Python, Node, and carrier-specific oracles must agree. ER-001 applies the same separation to mechanism semantics by deriving native outcomes from declared worlds and evidence-linked mechanism rules before K enters the outer equality checker. Both qualifications remain model-relative because world adequacy and live-runtime conformance are external premises.

3.3 Compilation Boundary and Diagnostic Isolation

Each case derivation names its retained artifacts, rulepack, and Assurance Contract. The frontend resolves the Fact Graph, and contract bindings compile facts and declarations into Projection IR. Vendor-specific syntax and extraction logic end before the semantic kernel, which consumes the IR rather than GitHub-, Azure-, Go-, or C#-specific rules. Projection IR stays small because it carries the assurance claim surface instead of reproducing the source system.

The optional declared_target_model_by_world stays outside the structural path. It compares an analyst-supplied end-to-end target outcome table with the declared source consequence table as a diagnostic only. Agreement cannot establish C, D, or O, and disagreement cannot create them. The GitHub structural gap, for example, derives from source-target facts, the discriminator model, discharge placement, and Boundary rather than from an analyst-authored adverse target table.

The target-model isolation invariant requires changes to declared_target_model_by_world, with the structural profile and derived facts fixed, to affect only the model cross-check; C, D, O, and the structural classification must remain unchanged. Section 4.5.5 tests this invariant by mutation. ER-001 applies the same noninterference discipline at the mechanism layer: target-model or D-map mutations cannot alter z_G, native outcomes, or the independently derived mechanism interpretation.

The evidence-carrying design has precedents in systems that bind results to checkable support. Proof-Carrying Code couples code with evidence checked against a policy [15], and in-toto binds supply-chain steps and artifacts to provenance metadata [16]. The analogy is limited: Projection Assurance records artifact identity, derivation lineage, and declaration provenance, but does not claim PCC proof soundness or in-toto cryptographic supply-chain guarantees.

4. Evaluation

Evaluation begins with two retained source-code projections that defeat opposite surface heuristics, then tests alternative explanations for their classifications in the model, implementation, extractor, analyst target model, and replay path. GitHub retains relevant head identity while omitting the earlier expected-head witness from the studied merge path; Azure removes a caller-visible overwrite control while the studied path realizes the declared create-if-absent consequence under its deterministic profile.

4.1 Evaluation Logic and Claim Scope

The pair targets two surface heuristics: that visibility of a consequence-relevant fact preserves the corresponding source guard, and that disappearance of a caller-visible source control entails semantic loss. Each is a general implication, so one grounded counterexample is sufficient to refute it as a rule. GitHub supplies the first counterexample and Azure the second; the pair is contrastive rather than statistical.

Evaluation uses one carrier-neutral kernel across the canonical cases, tests implementation and extraction alternatives, derives effect-level claims without using O or the analyst target table as mechanism-semantic inputs, applies the frozen machinery prospectively to a prelocked projection, and recomputes the qualification chain under stated same-host and authorship limits. The 12-vector corpus tests classifier logic; Section 4.6 supplies one prospective holdout. Neither supports prevalence estimates. The reported verdicts are source- and model-grounded rather than live-service conformance claims, and adequacy of the declared correspondence, worlds, and mechanism models remains a separate judgment.

The Evaluation Capsule [27] preserves publication-facing records for SG-001, ER-001, FH-001, ASF-001, and RR-001 together with the SHA-256 identities of the original qualification archives; the identifiers below refer to those recorded experiments.

4.2 GitHub Guarded Merge: Observable Head Identity Without Expected-Head Binding

The source merge operation supports an explicit expected-head precondition. In the pinned go-github source retained by v0.2.0, PullRequestOptions includes SHA and merge-request construction copies it into the request's sha field [27]. GitHub's REST API specifies sha as the pull-request head that must match for merge to proceed, with conflict on mismatch; the CLI exposes the same intent as --match-head-commit [22], [23].

The studied github/github-mcp-server handler reaches that merge operation without binding the precondition. Its retained complete MergePullRequest function constructs PullRequestOptions with commit-title and merge-method fields, omits SHA, and calls client.PullRequests.Merge [27]. MinimalPullRequest.Head elsewhere exposes a SHA, so the target surface retains head identity while the direct merge handler fails to carry a previously observed head into the source expected-head field. This negative claim is limited to the retained complete handler.

The relevant action cut runs from review or observation to the merge request. Fix the reviewed witness as H_0, and let the declared Boundary permit the head to advance to H_1 before the handler sends the request. A request carrying expected H_0 would let the source primitive reject the stale attempt; the studied handler omits that witness. GitHub's asynchronous merge endpoint provides a different protection: when sha is omitted it fixes a request-time expected head and cancels a pending

merge if the head changes afterward [22]. If H_0 has already advanced to H_1 before the request, that protection concerns H_1 and does not recover the earlier review-to-request relation.

The canonical contract instantiates this cut with a fixed expected witness H_0 and two admissible request-time head states:

$$R_{GH} = \{H_0, H_1\}.$$

$$K(H_0) = \text{MERGED\_REVIEWED\_HEAD}; \quad K(H_1) = \text{STALE\_REQUEST\_REJECTED}.$$

The discriminator is request-time head identity, $d(H_0)=H_0$ and $d(H_1)=H_1$, so D1 holds and the finite profile is consequence-exact. Retained Head.SHA grounds the target-side interpretation. SG-001 reconstructs $\{H_0{:}H_0, H_1{:}H_1\}$ from declared request_head world coordinates without reading K; Python, Node, and a target-representation oracle agree, and K or target-model mutations leave the signatures unchanged. The retained source-target merge mapping supports the C1 entry condition for this declared scope. O0 follows from the complete handler evidence because the action binds neither the earlier expected-head witness into SHA nor a declared equivalent discharge on this Boundary.

GitHub canonical result: C1 / D1 / O0 → OPERATIVE_DISCHARGE_MISSING

ER-001 yields a different diagnosis at the effect layer. On the fixed-H_0 profile, z_G contains request-time head, absence of expected SHA, and the otherwise-permissive merge condition; H_0 and H_1 remain distinguishable, so $z_G \geqslant_R K$. The failure is mechanism misalignment. At H_1, the modeled native outcome merges the current H_1 head, which maps faithfully to neither MERGED_REVIEWED_HEAD nor STALE_REQUEST_REJECTED and is retained as OUTSIDE_C(MERGED_UNREVIEWED_HEAD). Thus α_G^+ differs from the embedded required decoder $\iota \circ \kappa_{zG}$, contradicting the totalized ER-001 equality. In ER-001's counterfactual mechanism mutation, binding reviewed H_0 as expected SHA changes H_1 to an expected-head mismatch rejection and flips the modeled verdict to REALIZATION_ESTABLISHED.

D1 and O0 express different facts. D1 says request-time head identity determines K on the fixed-H_0 profile; O0 says the studied effect path does not bind the earlier witness required by the source expected-head mechanism. The case therefore isolates operative binding rather than information-theoretic impossibility. Its scope is the retained path: the result neither generalizes to GitHub merging nor requires agents to receive raw SHA values.

4.3 Azure Create-Only Blob: Target-Grounded Narrowing with an Effect-Bound Existence Guard

The retained Microsoft MCP BlobUploadCommand is create-only: it uploads a local file only when the destination blob does not exist, and its caller-facing options expose no general overwrite or ETag choice [27]. The target thus narrows the source write family before evaluation. Projection Assurance asks whether the create-if-absent consequence survives that specialization.

The implementation carries this policy to the request path. The MCP service calls BlobClient.UploadAsync(fileStream, false, cancellationToken); in the pinned Azure.Storage.Blobs source, overwrite=false becomes a BlobRequestConditions object with wildcard IfNoneMatch [27]. Microsoft documents the Boolean overload as creating a new block blob or failing when one already exists [24], and RFC 9110 supplies the conditional semantics of If-None-Match:* [14]. The create-only condition is therefore evaluated on the effect request.

The declared finite profile makes explicit which distinctions matter:

$$R_{AZ} = \{\text{ABSENT}, E_0, E_1\}.$$

$$K_{AZ}(\text{ABSENT}) = \text{CREATED}; \quad K_{AZ}(E_0) = K_{AZ}(E_1) = \text{NOT\_CREATED}.$$

$$d_{AZ}(\text{ABSENT}) = \text{ABSENT}; \quad d_{AZ}(E_0) = d_{AZ}(E_1) = \text{EXISTS}.$$

The existing-blob worlds E_0 and E_1 carry different version identities but the same consequence, so ETag identity is irrelevant to this lens; d_{AZ} and K_{AZ} induce the same partition and D1 holds. SG-001 reconstructs ABSENT versus EXISTS from the declared blob_exists coordinate without reading K. Python, Node, and an If-None-Match:* oracle agree; ETag-only mutation is invariant, while changing existence changes the signature. The retained command and service/SDK path support the C1 entry condition and place the wildcard existence precondition on the upload request, giving O1 under the v0.2.0 placement rule. Mechanism meaning is grounded in the retained SDK path, Microsoft documentation, and RFC 9110 rather than an analyst target-outcome table.

Azure canonical result: C1 / D1 / O1 → STRUCTURAL_ASSURANCE_ESTABLISHED

ER-001 establishes the effect-level equation on this specialized profile. z_G records the fixed If-None-Match:* condition and effect-time existence. Retained SDK and HTTP semantics yield WRITE_CREATED_NEW_BLOB for ABSENT and PRECONDITION_BLOCKED_EXISTING_BLOB for E_0/E_1; the grounded consequence interpreter maps them to CREATED and NOT_CREATED. Hence α_G^+ ranges entirely over $\iota(C)$ and equals $\iota \circ \kappa_{zG}$. Identifying C with its embedded copy therefore establishes $\text{Realize}_R(G,K)$ for the declared deterministic model. In ER-001's counterfactual mechanism mutation, removing the wildcard condition yields modeled overwrite outcomes outside the create-only consequence classes and changes the totalized verdict to REALIZATION_CONTRADICTED [14], [24].

The Azure result is scoped to that create-if-absent slice. The declared source family also contains replace-if-match and unconditional-put behaviors that the target does not represent. Structural assurance and source-family coverage therefore remain distinct.

4.4 Cross-Case Synthesis: Surface Retention, Operative Binding, and Coverage Are Distinct

Table 2. The cases separate caller-surface retention, operative binding, exact realization, and source-family coverage.

Dimension	GitHub guarded merge	Azure create-only blob
Declared source slice	Guarded merge by expected reviewed head	Create only if destination is absent
Consequence-relevant distinction	Reviewed head identity	Absent versus existing blob
Caller-surface treatment	Head.SHA remains observable; expected witness omitted from direct merge action	General overwrite / ETag choice absent from caller surface
Operative binding	Earlier reviewed H_0 is not bound into the source expected-head guard	If-None-Match:* is placed on the upload request
C / D / O	**C1 / D1 / O0**	**C1 / D1 / O1**
Exact realization (ER-001)	Contradicted on declared model: z_G sufficient; α_G^+ misaligned, with H_1 retained as OUTSIDE_C rather than stale rejection	Established on declared model: z_G sufficient; $\alpha_G^+ = \iota \circ \kappa_{zG}$ on the declared worlds
Coverage	Complete for declared guarded-merge family	Incomplete relative to broader declared write family

The pair establishes three separations on the declared profiles. GitHub shows that a consequence-sufficient representation can remain visible while operative preservation fails. Azure shows that a caller-visible source control can disappear while the specialized consequence remains enforced at the effect. The pair also separates coverage from structural assurance: GitHub covers its declared guarded-merge family without structural assurance, while Azure establishes structural assurance on a narrower slice with incomplete family coverage.

4.5 Adversarial Validation of the Evidence-to-Verdict Chain

Adversarial validation targets alternative explanations for the reported results. Threats include a kernel keyed to the canonical answers, compiler-manufactured semantics, syntax-sensitive extraction, a discriminator table chosen from K, placement mistaken for effect correctness, α_G derived from the desired consequence, target-model leakage into C/D/O, outcome-driven holdout tuning or carrier selection, and replay from saved outputs. The controls address these failure paths separately; each PASS addresses one route without substituting for the others.

4.5.1 Semantic Kernel Conformance

Semantic conformance checks the executable kernel beyond the two vendor profiles. Twelve authored vectors cross the principal structural boundaries, including C0 and CU, D1/O0, D0/O1, O1_REPLACED, overdiscrimination, and outside-scope behavior. Python and Node return the expected result on all 12/12 vectors [27]. The D1/O0 versus D0/O1 crossing is especially diagnostic because it confirms that discrimination and placement do not collapse into one favorable-status rule.

The corpus validates implementation against the authored specification rather than the abstraction itself. Agreement between separate Python and Node implementations therefore provides differential evidence against implementation-specific error [35], with specification, vectors, and expectations still RISU-authored.

4.5.2 Derivation and Extraction Robustness

Upstream of the kernel, C/D/O inputs could depend on one compiler or extraction strategy. In v0.2.0, Python and Node derivation compilers regenerate byte-identical Fact Graph and Projection IR products for each case, and a separate Derivation Checker reconstructs and accepts the semantic IR [27]. Because both compilers share retained artifacts, rule packs, and Assurance Contracts, this tests implementation divergence rather than source-interpretation independence.

GitHub also receives a structurally different extraction check. Text rules and Go-AST queries agree on 4/4 mapped evidence facts; three semantics-preserving AST transformations leave those facts invariant, and four semantic mutations are detected [27]. Together, the tests target opposite extractor failures: sensitivity to irrelevant syntax and insensitivity to material structural change [33], [34].

For Azure, ASF-001 supplies an analogous structured-source qualification. Two separately implemented token-structured C# frontends, a Python Pygments lexer/parser and a Node scanner/parser, reconstruct the retained option surface, command-to-service call, UploadAsync(..., false, ...) argument, and SDK lowering to wildcard If-None-Match. The frontends agree on the canonical facts, remain invariant under comment, whitespace, and local-identifier changes, detect directional semantic mutations, and fail closed when required structural loci disappear; ASF-001 passes 25/25 checks. This removes the earlier

GitHub-only extraction-diversity asymmetry on the tested Azure relations. The qualification is token-structured, not a Roslyn/compiler AST proof, and query and scope adequacy remain external.

4.5.3 Consequence-Blind Signature Grounding

Projection IR v0.2.0 accepts discriminator.signature_by_world from the Assurance Contract, creating a D-specific circularity risk: an analyst could choose a table that mirrors the consequence partition. SG-001 keeps the generic IR format intact but requires canonical export signatures to be regenerated from declared world-state records through an evidence-linked rule whose executable inputs exclude the source consequence map and declared_target_model_by_world.

Both canonical cases pass. GitHub regenerates $H_0 \to H_0$ and $H_1 \to H_1$; Azure regenerates ABSENT→ABSENT and E_0/E_1→EXISTS. Python and Node outputs are byte-equivalent at canonical JSON, carrier-specific oracles agree, and both maps match the retained compatibility tables. K and target-model mutations leave the maps unchanged; consequence-relevant world changes alter them; reviewed-head or ETag identity changes that are irrelevant to the respective discriminator do not. Missing grounding evidence or required world coordinates fail closed. SG-001 records 20/20 checks with zero semantic-core, C/D/O-rule, or contract-signature changes.

SG-001 removes direct per-world signature assignment as the sole support for canonical D. The worlds, state coordinates, and semantic choice of discriminator remain qualified premises supported by Boundary and source evidence. SG-001 strengthens the link from those premises to d without converting the world model into runtime observation.

4.5.4 Exact Realization Grounding

Because O certifies placement rather than mechanism correctness, ER-001 adds a separate effect-semantic qualification. Its derivers receive declared worlds, fixed profile constants, resolved mechanism evidence, carrier-specific mechanism rules, and consequence-class definitions, excluding the desired per-world K answers. They emit z_G, native outcomes, and a totalized mechanism interpretation; only the outer checker then sees K and computes sufficiency, the required decoder, α_G^+ alignment, and worldwise equality.

ER-001 passes 44/44 checks. Python and Node mechanism derivations agree, carrier-specific oracles produce the same products, and the mechanism semantic coordinate agrees with SG-001 without consuming the compatibility map. GitHub has $z_G \succcurlyeq_R K$ but α_G^+ misalignment at H_1, retained as OUTSIDE_C, so its status is REALIZATION_CONTRADICTED. Azure has $z_G \succcurlyeq_R K$ with α_G^+ aligned to the embedded required decoder on all three worlds, so its status is REALIZATION_ESTABLISHED.

The controls test direction as well as agreement. K, target-model, and D-map mutations leave z_G, native outcomes, and mechanism interpretation unchanged; relevant world-state mutations alter the derivation, while GitHub repository identity and Azure ETag-only mutations do not. Missing mechanism evidence, missing required world fields, and overlapping consequence predicates fail closed. In ER-001 counterfactual semantic mutations, binding reviewed head as expected SHA changes GitHub to established, while removing If-None-Match:* changes Azure to contradicted. Author-controlled replay regenerates the same report digest. An ER-001 PASS certifies protocol integrity within this qualification; external reproduction remains NOT_CLAIMED.

4.5.5 Diagnostic Noninterference, Recomputability, and Reproduction Status

Section 3 requires analyst-authored target outcomes to stay outside C/D/O. The target-model separation test flips GitHub's diagnostic cross-check from mismatch to match and Azure's from match to mismatch while leaving C, D, O, and both structural classifications unchanged [27]. This directly tests the claimed noninterference property.

Recomputability is evaluated separately from semantic correctness. The v0.2.0 reproduction path verifies retained inputs, rebuilds the structured frontend, regenerates both cases with both compilers, checks the derivations, runs both analyzers, reruns the 12-vector corpus, and repeats structured metamorphic and mutation gates; the author run completes 18/18 steps [27]. RR-001 then replays the historical capsule, synthesis package, SG-001, ER-001, FH-001, and ASF-001 from fresh extracts under two process environments on the same host. SG-001, ER-001, FH-001, ASF-001, and normalized synthesis reports regenerate byte-identically; the historical capsule yields identical normalized scientific results after excluding runtime timing fields. This establishes author-controlled dual-environment reproduction readiness on one host. Cross-machine and non-author execution remain open, and external reproduction remains NOT_CLAIMED.

Table 3. Orthogonal checks target distinct failure paths in the evidence-to-verdict chain; no row substitutes for the others.

Threat	Control	Bounded result	Still external
Canonical hard-coding	12-vector conformance on Python and Node analyzers	12/12 on each analyzer.	Vectors and expected results are RISU-authored; they test implementation against the specification, not the abstraction itself.
Compiler-specific derivation	Python/Node derivation compilers plus Derivation Checker	Byte-identical Fact Graph and IR per case; checker accepts each product.	Same retained evidence, rules, contracts, and authorship.
Extraction artifact	GitHub text/Go-AST differential; ASF-001 dual structured C# frontends; metamorphic and semantic-mutation tests	GitHub: 4/4 facts, 3/3 invariants, 4/4 mutations. Azure: ASF-001 25/25 with dual-frontend agreement, mutation sensitivity, and fail-closed missing loci.	Authored query and scope adequacy; ASF-001 is token-structured, not a full compiler AST proof.
Discriminator circularity	SG-001 consequence-blind dual derivation, carrier oracle, mutation and fail-closed controls	20/20; canonical maps reconstructed; K and target-model mutations are noninterfering; 0 C/D/O rule changes.	World set, state coordinates, and discriminator interpretation remain declared/model-grounded; no live measurement.
Placement or K mistaken for mechanism truth	ER-001 mechanism derivation excluding per-world K and target outcomes, carrier oracle, totalized outcomes, noninterference, and counterfactual semantic mutations	44/44; GitHub contradicted, Azure established; counterfactual bind/remove-guard mutations flip the corresponding modeled exact verdicts.	Finite worlds, Boundary premises, mechanism-model completeness, consequence-class adequacy, and live-runtime conformance.
Target-model leakage	Target-model mutation negative control	Cross-check flips while C/D/O and the structural verdict remain unchanged.	Does not validate the Boundary, consequence lens, or source grounding.
Outcome-driven holdout tuning / carrier cherry-picking	FH-001 prelocks the semantic core, deterministic selection rule, case profile with expected verdict unspecified, and repair vocabulary before target implementation inspection; all post-lock discovered create_order carriers are evaluated.	49/49; deterministic selection and lock chain hold; every retained carrier is evaluated; the static-token family is checked symbolically; frozen-vocabulary repair and surface-only negative controls are direction-sensitive.	Author-controlled candidate universe and case model; no independent blinding, representative sampling, or live target mutation.
Saved output mistaken for replay	Sealed replay plus RR-001 dual fresh-process-environment replay	18/18 canonical; RR-001 40/40. SG/ER/FH/ASF and synthesis reports reproduce identically; the historical capsule is scientifically identical after timing normalization.	Author-controlled, same host/toolchain; no upstream reacquisition, cross-machine independence, or non-author reproduction.

4.6 Prospective Frozen Holdout: PayPal Create-Order Idempotency

FH-001 tests whether already-qualified machinery can analyze a new projection without outcome-driven retuning. Before inspecting the target implementation, the protocol freezes the semantic core, C/D/O, SG-001 rules, ER-001 exact-realization equation, OUTSIDE_C discipline, witness-minimality rule, and a seven-obligation repair vocabulary. Candidate eligibility and deterministic selection are predeclared; SHA-256 ordering selects PayPal Agent Toolkit create_order projecting PayPal Orders v2 POST /v2/checkout/orders with the PayPal-Request-Id guard. CASE_LOCK.json then fixes source and target refs, effect cut, bounded worlds, K, discriminator meaning, and source-guard obligation while recording the expected status as UNSPECIFIED_BY_PROTOCOL. Target implementation inspection begins only after that lock. FH-001 was executed under that frozen protocol, selection record, and CASE_LOCK.json; the Evaluation Capsule [27] preserves the resulting qualification record and the SHA-256 identity of the original archive.

PayPal documents PayPal-Request-Id as an idempotency key for supported POST calls: reusing the key returns the prior operation state instead of repeating the action, while omission supplies no such duplicate-prevention relation; Create Order documents a six-hour default retention interval [39], [40]. The pinned PayPal TypeScript Server SDK maps paypalRequestId to the request header on POST /v_2/checkout/orders [38]. The locked profile therefore tests the retry-sensitive consequence of the guarded create-order variant rather than imposing a request ID on every create-order call.

$$R_{FH} = \{\text{RETRY\_SAME}, \text{NEW\_DISTINCT}\}.$$

$$K_{FH}(\text{RETRY\_SAME}) = \text{NO\_DUPLICATE\_RETURN\_ORIGINAL\_OPERATION}.$$

$$K_{FH}(\text{NEW\_DISTINCT}) = \text{CREATE\_DISTINCT\_NEW\_ORDER}.$$

Both worlds share merchant and payload class P0 within the retention interval and differ only in logical-operation identity. RETRY_SAME reuses RID0 after an ambiguous prior success; NEW_DISTINCT denotes a new logical creation RID1. The discriminator SAME_OPERATION versus DISTINCT_OPERATION is consequence-exact, establishing D1. At the out-

bound create-order effect cut, preservation requires an identity stable across retries of one logical creation and distinct across separate creations, or an evidence-grounded equivalent.

Post-lock inspection finds two retained target carriers, and FH-001 evaluates both. The TypeScript create_order schema exposes no per-invocation request-id field; createOrder obtains shared headers and posts to /v_2/checkout/orders [37]. PayPalClient can copy context.request_id into PayPal-Request-Id, but that context is set at toolkit/client construction and reused by the retained tool instance. The Python path posts the same endpoint and its retained header builder has no PayPal-Request-Id path [37]. A wrapper that constructs a separate toolkit instance or supplies a per-logical-operation key would constitute a separately grounded replacement outside these retained paths.

Table 4. Frozen holdout results across all retained create_order carrier modes.

Retained target mode	Effect request ID	O	Exact result	Minimal mismatch witness
Python create_order	absent	O0	CONTRADICTED (z insufficient)	RETRY_SAME
TypeScript; config absent	absent	O0	CONTRADICTED (z insufficient)	RETRY_SAME
TypeScript; static = RID0	RID0 in both worlds	O1	CONTRADICTED (z insufficient)	NEW_DISTINCT
TypeScript; static ≠ RID0	same other ID in both worlds	O1	CONTRADICTED (z insufficient)	RETRY_SAME
Repair: invocation-relative binding	**current logical operation ID**	**O1**	**ESTABLISHED**	**none**

The holdout exposes a third failure mode. Python and unconfigured TypeScript paths return O0 and fail exact realization. A constructor-scoped static TypeScript token can return O1 because the header reaches the effect, yet exact realization still fails: if the token equals prior RID0, both worlds replay the prior operation; otherwise both are treated as one new keyed operation. Either way, RETRY_SAME and NEW_DISTINCT share z_G despite different K values. FH-001 proves the same insufficiency symbolically for the entire static-token family.

The cardinality-minimal insufficiency witness is {NEW_DISTINCT, RETRY_SAME}. For worldwise equality, one realization suffices: RETRY_SAME for absent or static-different modes, NEW_DISTINCT for the static-equal mode. Within the repair vocabulary frozen before target inspection, inclusion-minimal synthesis returns the singleton CARRY_WITNESS_TO_EFFECT. Under the gate's modeled repair semantics, binding PayPal-Request-Id to current_logical_operation_id preserves the identifier across retries and changes it for a distinct operation, establishing exact realization on both locked worlds. A surface-only modeled mutation that exposes the witness without binding the outbound request remains a failing negative control.

FH-001 passes 49/49 checks with zero changes to the semantic core, C/D/O, SG-001, or ER-001 rules. Python and Node derivations agree, a carrier oracle agrees, all retained create_order carrier paths discovered after lock are evaluated, the ER-001 exact-analysis function is byte-identity checked against its parent binding, and repair directionality is tested. The evidentiary claim is prospective author-controlled transfer on one prelocked projection. The prelock record constrains outcome-driven retuning and post-lock carrier selection within the recorded procedure; it does not provide independent blinding or representative sampling. Third-party reproduction and live PayPal mutation testing remain outside scope.

5. Related Work

Several established literatures supply the mathematical and methodological ingredients used here. Behavioral and strong-preservation work make semantic preservation property-relative [41]; determinacy asks when one representation fixes another; translation validation checks concrete transformations across representations; binding-time and temporal-consistency work distinguish stable specialization from state that requires later validation; and service-semantic and agent-tool research separates descriptions, implementations, and enforcement. Projection Assurance organizes these ingredients around one projection-specific assurance object: a concrete source-to-target operation relation, a declared consequence and scope, and an evidence chain that evaluates correspondence, consequence determinacy, effect realization, operative placement, and coverage separately.

5.1 Behavioral Compatibility, Adaptation, and Property-Relative Substitution

Behavioral compatibility has long exceeded interface shape. Liskov and Wing ground substitutability in preserved behavioral properties [1]; Zaremski and Wing compare components through formal pre- and postconditions [2]; Yellin and Strom treat sequencing protocols and adaptors [3]; and Interface Automata models temporal assumptions, guarantees, compatibility, and refinement beyond signatures [36]. Service interoperation and substitution add that semantic suitability can be separated

from invocation mechanics [4], substitution may depend on the functionality required by a composition [5], and adapter chains may lose capabilities [6].

These lines of work support two premises used here: literal field parity is unnecessary for preservation, and surface similarity is insufficient; preservation can also be property-relative. Given a licensed source-target correspondence, Projection Assurance asks whether the target distinguishes every realization requiring a different consequence and whether that distinction remains operative at the effect.

5.2 Determinacy and Information Refinement

Discrimination uses established determinacy mathematics. Armstrong's functional dependency $X \rightarrow Y$ requires agreement on X to entail agreement on Y [28]. View determinacy extends the idea to whether views uniquely determine a query result [31]. Deterministic information can be ordered by induced partitions [29], with partition refinement equivalently characterized by deterministic post-processing or factorization [30].

On a finite profile, $d \succcurlyeq_R K$ therefore means that d determines K, equivalently that K factors through d. The same foundation yields the no-recovery result for deterministic post-processing. Whether the target obtains, refreshes, or correctly uses that information at the effect is a separate operative and evidentiary question.

5.3 Translation Validation and Property-Relative Preservation

Translation validation supplies an instance-specific methodological precedent. Pnueli, Siegel, and Singerman validate each concrete source-target translation instead of proving a translator correct for all inputs [9]; Necula applies the same strategy to optimizing compilation [10]. Projection Assurance follows that instance-wise discipline: a passing case supports one pinned source operation, projected operation, scope, and consequence lens rather than certifying an entire projector, protocol, SDK, or server family.

Cross-representation correctness need not require literal identity. Hoare relates concrete data representations to abstract ones [32]. Abate and colleagues relate source and target traces drawn from different sets and derive the corresponding transport of properties [11]; robust property-preservation work likewise makes the protected source-property class explicit [25]. Strong preservation in abstract interpretation makes the same property relativity explicit at the abstraction boundary: concrete and abstract models are required to agree on the formulas of a chosen specification language [41]. These are close formal precedents for property-relative semantic preservation. Projection Assurance does not claim a new general refinement or preservation relation. It specializes the question to one projected operation and treats representation sufficiency, effect-path realization, operative placement, and source-family coverage as separate obligations. A target may therefore pass for a declared consequence on a nonempty specialized scope without being behaviorally equivalent to the full source operation.

5.4 Binding Time and Effect-Time Validation

Binding-time analysis explains when omission can be principled. Partial evaluation separates static information fixed during specialization from dynamic information left to residual computation [12]. In Projection Assurance, a source coordinate may disappear from the invocation surface when the target independently fixes the relevant policy over the restricted domain. A value held constant only in one analyzed invocation does not constitute target-level specialization.

Mutable state creates the complementary problem. TOCTOU work shows that a fact established earlier may be false when later used [13]. HTTP conditional requests supply a protocol-level pattern for effect-time validation because If-Match and If-None-Match condition method execution on current representation state [14]. The resulting temporal rule is simple: stable policy may be bound early; invocation-specific evidence tied to mutable state must remain bound, be reacquired, or be revalidated while it can still control the effect.

5.5 Service Semantics and the Agent-Tool Boundary

Semantic-service research already separated callable interfaces from service meaning. Paolucci and colleagues match services through declarative capabilities rather than interface or registry fields alone [7]. OWL-S distinguishes profile, process model, and grounding while making inputs, outputs, preconditions, and effects explicit [8]. These systems establish a useful precedent for representing semantic meaning separately from invocation mechanics.

Agent-tool systems address neighboring assurance relations. MCP standardizes tool discovery and invocation while treating ToolAnnotations as behavioral hints rather than guarantees [17]. Work on tracked capabilities and verifiable tool use constrains which capabilities or tool/data flows an agent may exercise [18], [19]. Recent preprints examine protocol semantics [20] and description-code inconsistency in MCP servers [21].

Description consistency concerns whether a target tool is described faithfully; policy enforcement concerns whether its behavior may be exercised; Projection Assurance concerns whether that behavior preserves the declared consequence semantics of the source operation. These checks address different relations: satisfying one does not by itself establish the others. The

scope here is concrete source-to-target consequence fidelity. MCP is one projection carrier represented in the retained cases; the formal definitions and verdict rules do not depend on MCP.

6. Discussion and Limitations

Projection Assurance and Consequence Closure [26] qualify adjacent boundaries. Projection Assurance asks whether source consequence semantics survive projection. Consequence Closure begins with a declared semantic model and asks whether current evidence fixes the action consequence across admissible realizations. A faithful projection can feed an OPEN action cut, and a CLOSED internal model can remain source-unsound after an upstream material condition was lost. Neither verdict entails the other.

6.1 Implications for Projected Interfaces

The design implication is to preserve consequence-relevant distinctions instead of source syntax. A target may omit a source argument when it independently fixes a narrower policy and the remaining representation still separates consequence-divergent realizations. Exposing a relevant value is insufficient if the action path never binds that value, or an equivalent distinction, where it can govern the effect. Review should follow the distinction from source consequence to target effect rather than count retained fields.

Binding time refines that rule. Stable policy can be specialized before invocation; an invocation-specific witness tied to mutable state or logical-operation identity must reach the effect through carriage, reacquisition, or revalidation while it remains material. Azure fixes create-only policy and re-evaluates existence at upload; GitHub exposes head identity without binding the reviewed head as the merge precondition. ER-001 shows that GitHub retains enough request-time distinction but implements the wrong consequence relation, while Azure's wildcard condition implements the required decoder on the declared profile. The PayPal holdout adds operative insufficiency: an idempotency header can reach the effect yet remain too coarse when its value is fixed at toolkit construction. The operative signature must vary along every consequence-material coordinate that remains live in scope.

Exported assurance results should carry their scope, consequence lens, grounding evidence, and operative status. These qualifications matter when autonomous software composes observation and action across independently designed tools, where a distinction may be observed in one component, transformed in another, and enforced in a third. A carrier-neutral representation should preserve those claim boundaries.

This separation also matters in socio-technical systems, where source consequences may reflect human or organizational rules as well as software-local constraints. Once expressed in software, the same material distinction can be retained, specialized, misaligned, or lost across a projection. Its adequacy remains a modeling and grounding question.

6.2 Claim Boundary and Generality

Projection Assurance is conditional on a declared semantic profile. Within that profile, D is exact on the supplied finite scalar relation and the v0.2.0 structural classification follows mechanically from the declared facts and certificate rules. SG-001 reconstructs canonical discriminator maps from declared world-state coordinates through consequence-blind, evidence-linked rules. ER-001 derives z_G and carrier-native outcomes without access to per-world K, the target model, or D compatibility tables, then checks the exact consequence equation. Under those declared deterministic models, GitHub is REALIZATION_CONTRADICTED and Azure is REALIZATION_ESTABLISHED. The remaining judgment lies in the premises connecting finite models to real systems: adequacy of R, K, Boundary and fixed coordinates, specialization, correspondence evidence, consequence-class semantics, and mechanism-model completeness. Authorization, transaction-wide safety, and live-runtime certification require additional evidence.

The evaluation supports logical separation and one prospective transfer rather than ecosystem prevalence. GitHub and Azure are contrastive witnesses against opposite surface heuristics, and the prelocked PayPal holdout tests transfer without semantic retuning. Structured extraction and fresh-process replay reduce specified extractor- and execution-path alternatives. The candidate universe, source interpretation, case models, and execution remain author-controlled; one holdout is not a representative benchmark and author replay is not independent reproduction.

The executable core remains finite and deterministic. Projection IR v0.2.0 provides the structural kernel and Operative Placement certificate; SG-001 and ER-001 are companion qualifications, and FH-001 tests the frozen machinery on one new deterministic projection. The two-world PayPal profile abstracts the documented idempotency relation over one retention interval and excludes concurrent same-key races, cross-account behavior, full order lifecycle semantics, and live-service nondeterminism. Probabilistic behavior, richer temporal or transactional semantics, nondeterministic discharges, and stronger runtime claims require corresponding extensions to the model, verifier, and evidence.

Subsequent software releases reduce several implementation-side trust boundaries without changing the evaluation reported here. The current v0.7.0 release separates a versioned Source Consequence Contract from a hash-pinned Projection Adapter:

finite admissible realizations are generated from declared coordinate domains and an admissibility rule, the source consequence is evaluated from one declared consequence function, and target semantic compilation remains consequence-blind. Proof-carrying certificates can then be checked through a compact path that does not depend on the producer verifier/compiler stack [42]. These changes reduce direct authorship of per-world answer tables and the amount of producer code a certificate consumer must trust. They do not establish the adequacy of the declared source model, the correctness of semantic evidence interpretation, live-runtime conformance, or independent reproduction.

7. Conclusion

Projection Assurance evaluates source-to-target projections at the level of declared consequence rather than surface form. Correspondence licenses the comparison. Discrimination asks whether the target-side representation determines the source consequence on the declared domain. Operative Placement records the recognized location of an evidence-backed discharge. Exact Realization asks whether the effect path retains enough information and implements the required consequence mapping. Coverage remains a separate question. This decomposition avoids treating parameter retention, state visibility, or guard presence as stand-alone proxies for semantic preservation.

The retained cases exercise these distinctions in three different ways. On the declared deterministic models, GitHub preserves enough operative information on the fixed-H_0 profile but the studied direct merge mechanism does not realize the required reviewed-head consequence; Azure narrows the source write family and realizes the declared create-if-absent consequence through an effect-time existence condition; and the frozen PayPal holdout shows that a request-id header can reach the effect while a constructor-scoped static value still collapses retry and distinct-operation worlds. The supporting qualification chain adds consequence-blind signature reconstruction, mechanism-level checking, structured extraction diversity, prospective evaluation, and fresh-process replay without changing the frozen semantic core. Scope assumptions, model adequacy, live-runtime conformance, and independent reproduction remain explicit boundaries. For a declared projection, the method therefore provides a bounded way to determine which consequence-relevant distinctions survive projection, whether the declared mechanism preserves them at the effect cut, and what evidence supports that determination.

Artifact Availability and Software Versions

The reported evaluation is bound to the frozen v0.2.0 software and its preserved qualification records. Those materials are archived in a dedicated Evaluation Capsule [27], which includes the evaluated code and retained evidence and records the qualification results and cryptographic identities without treating the historical package as the current software release. The current implementation is v0.7.0 [42], archived separately as the current citable software release. The later implementation lineage retains the semantic definitions developed in this note while strengthening artifact construction and certificate checking; the Section 4 results remain bound to the frozen Evaluation Capsule. The note, Evaluation Capsule, and current software are separate research objects with persistent identifiers.

References

[1] B. Liskov and J. M. Wing, “A Behavioral Notion of Subtyping,” ACM Transactions on Programming Languages and Systems, vol. 16, no. 6, pp. 1811-1841, 1994, doi: 10.1145/197320.197383.

[2] A. M. Zaremski and J. M. Wing, “Specification Matching of Software Components,” ACM Transactions on Software Engineering and Methodology, vol. 6, no. 4, pp. 333-369, 1997, doi: 10.1145/261640.261641.

[3] D. M. Yellin and R. E. Strom, “Protocol Specifications and Component Adaptors,” ACM Transactions on Programming Languages and Systems, vol. 19, no. 2, pp. 292-333, 1997, doi: 10.1145/244795.244801.

[4] S. R. Ponnekanti and A. Fox, “Application-Service Interoperation without Standardized Service Interfaces,” in Proc. 1st IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), pp. 30-37, 2003, doi: 10.1109/PERCOM.2003.1192724.

[5] J. Pathak, S. Basu, and V. Honavar, “On Context-Specific Substitutability of Web Services,” in Proc. 2007 IEEE International Conference on Web Services (ICWS 2007), pp. 192-199, 2007, doi: 10.1109/ICWS.2007.134.

[6] Y. Chung and D. Lee, “Mathematical Basis for the Chaining of Lossy Interface Adapters,” IET Software, vol. 4, no. 1, pp. 43-54, 2010, doi: 10.1049/iet-sen.2009.0019.

[7] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, “Semantic Matching of Web Services Capabilities,” in The Semantic Web - ISWC 2002, LNCS 2342, pp. 333-347, 2002, doi: 10.1007/3-540-48005-6_26.

[8] D. Martin et al., “OWL-S: Semantic Markup for Web Services,” W3C Member Submission, Nov. 22, 2004. [Online]. Available: https://www.w3.org/Submission/OWL-S/

[9] A. Pnueli, M. Siegel, and E. Singerman, “Translation Validation,” in Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1998), LNCS 1384, pp. 151-166, 1998, doi: 10.1007/BFb0054170.

[10] G. C. Necula, “Translation Validation for an Optimizing Compiler,” in Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2000), pp. 83-94, 2000, doi: 10.1145/349299.349314.

[11] C. Abate et al., “An Extended Account of Trace-relating Compiler Correctness and Secure Compilation,” ACM Transactions on Programming Languages and Systems, vol. 43, no. 4, Art. 14, pp. 1-48, 2021, doi: 10.1145/3460860.

[12] N. D. Jones, C. K. Gomard, and P. Sestoft, Partial Evaluation and Automatic Program Generation. Prentice Hall, 1993.

[13] D. Dean and A. J. Hu, “Fixing Races for Fun and Profit: How to Use access(2),” in 13th USENIX Security Symposium (USENIX Security 04), pp. 195-206, 2004. [Online]. Available: https://www.usenix.org/conference/13th-usenix-security-symposium/fixing-races-fun-and-profit-how-use-access2

[14] R. Fielding, M. Nottingham, and J. Reschke, Eds., “HTTP Semantics,” RFC 9110, STD 97, June 2022, doi: 10.17487/RFC9110.

[15] G. C. Necula, “Proof-Carrying Code,” in Proc. 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1997), pp. 106-119, 1997, doi: 10.1145/263699.263712.

[16] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, “in-toto: Providing Farm-to-Table Guarantees for Bits and Bytes,” in 28th USENIX Security Symposium (USENIX Security 19), pp. 1393-1410, 2019. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias

[17] Model Context Protocol, “Specification,” protocol revision 2025-11-25. [Online]. Available: https://modelcontextprotocol.io/specification/2025-11-25. Accessed: Aug. 26, 2026.

[18] M. Odersky, Y. Zhao, Y. Xu, O. Bračevac, and C. N. Pham, “Securing Agents With Tracked Capabilities,” in Proc. ACM Conference on AI and Agentic Systems (CAIS 2026), pp. 812-838, 2026, doi: 10.1145/3786335.3813127.

[19] A. Doshi, Y. Hong, C. Xu, E. Kang, A. Kapravelos, and C. Kästner, “Towards Verifiably Safe Tool Use for LLM Agents,” in Proc. IEEE/ACM 48th International Conference on Software Engineering, NIER (ICSE-NIER 2026), pp. 201-205, 2026, doi: 10.1145/3786582.3786839.

[20] A. Schlapbach, “Formal Semantics for Agentic Tool Protocols: A Process Calculus Approach,” arXiv:2603.24747, 2026. [Online]. Available: https://arxiv.org/abs/2603.24747

[21] Y. Shi et al., “Description-Code Inconsistency in Real-world MCP Servers: Measurement, Detection, and Security Implications,” arXiv:2606.04769, 2026. [Online]. Available: https://arxiv.org/abs/2606.04769

[22] GitHub, “REST API endpoints for pull requests,” sections “Merge a pull request” and “Merge a pull request asynchronously.” [Online]. Available: https://docs.github.com/en/rest/pulls/pulls. Accessed: Aug. 26, 2026.

[23] GitHub CLI, “gh pr merge,” option --match-head-commit <SHA>. [Online]. Available: https://cli.github.com/manual/gh_pr_merge. Accessed: Aug. 26, 2026.

[24] Microsoft, “BlobClient.UploadAsync Method (Azure.Storage.Blobs),” Azure for .NET Developers. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/azure.storage.blobs.blobclient.uploadasync?view=azure-dotnet. Accessed: Aug. 26, 2026.

[25] C. Abate, R. Blanco, D. Garg, C. Hriţcu, M. Patrignani, and J. Thibault, “Journey Beyond Full Abstraction: Exploring Robust Property Preservation for Secure Compilation,” in Proc. 32nd IEEE Computer Security Foundations Symposium (CSF 2019), pp. 256-271, 2019, doi: 10.1109/CSF.2019.00025.

[26] M. Lee, “Consequence Closure: Semantic Assurance at the Machine Action Boundary,” RISU Technical Note 2026-03, Aug. 2026, doi: 10.5281/zenodo.22095709.

[27] M. Lee, Projection Assurance Evaluation Capsule: Frozen v0.2.0 Evaluation and Qualification Record, ver. 1.0.0 [Dataset]. RISU Institute, Aug. 2026, doi: 10.5281/zenodo.22149517.

[28] W. W. Armstrong, “Dependency Structures of Data Base Relationships,” in Information Processing 74: Proceedings of IFIP Congress 74. Amsterdam, The Netherlands: North-Holland, 1974, pp. 580-583.

[29] J. Landauer and T. Redmond, “A Lattice of Information,” in Proc. 6th IEEE Computer Security Foundations Workshop (CSFW 1993), pp. 65-70, 1993, doi: 10.1109/CSFW.1993.246638.

[30] M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith, “Measuring Information Leakage Using Generalized Gain Functions,” in Proc. 25th IEEE Computer Security Foundations Symposium (CSF 2012), pp. 265-279, 2012, doi: 10.1109/CSF.2012.26.

[31] A. Nash, L. Segoufin, and V. Vianu, “Views and Queries: Determinacy and Rewriting,” ACM Transactions on Database Systems, vol. 35, no. 3, Art. 21, pp. 21:1-21:41, 2010, doi: 10.1145/1806907.1806913.

[32] C. A. R. Hoare, “Proof of Correctness of Data Representations,” Acta Informatica, vol. 1, no. 4, pp. 271-281, 1972, doi: 10.1007/BF00289507.

[33] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, “Metamorphic Testing: A Review of Challenges and Opportunities,” ACM Computing Surveys, vol. 51, no. 1, Art. 4, pp. 4:1-4:27, 2018, doi: 10.1145/3143561.

[34] Y. Jia and M. Harman, “An Analysis and Survey of the Development of Mutation Testing,” IEEE Transactions on Software Engineering, vol. 37, no. 5, pp. 649-678, 2011, doi: 10.1109/TSE.2010.62.

[35] W. M. McKeeman, “Differential Testing for Software,” Digital Technical Journal, vol. 10, no. 1, pp. 100-107, 1998.

[36] L. de Alfaro and T. A. Henzinger, “Interface Automata,” in Proc. 8th European Software Engineering Conference / 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2001), pp. 109-120, 2001, doi: 10.1145/503209.503226.

[37] PayPal, PayPal Agent Toolkit, commit 2fe2525866247d6875977282048bb6a3074b0805 [Source code]. GitHub. [Online]. Available: https://github.com/paypal/agent-toolkit/tree/2fe2525866247d6875977282048bb6a3074b0805. Accessed: Aug. 27, 2026.

[38] PayPal, PayPal TypeScript Server SDK, commit b02a9b9c8f53418d6278928fa41b174803037277 [Source code]. GitHub. [Online]. Available: https://github.com/paypal/PayPal-TypeScript-Server-SDK/tree/b02a9b9c8f53418d6278928fa41b174803037277. Accessed: Aug. 27, 2026.

[39] PayPal, “Idempotency,” PayPal Developer. [Online]. Available: https://developer.paypal.com/reference/guidelines/idempotency/. Accessed: Aug. 27, 2026.

[40] PayPal, “Create order,” Orders v_2, POST /v_2/checkout/orders. [Online]. Available: https://developer.paypal.com/sdk/orders/v_2/orders-create/. Accessed: Aug. 27, 2026.

[41] F. Ranzato and F. Tapparo, “Generalized Strong Preservation by Abstract Interpretation,” Journal of Logic and Computation, vol. 17, no. 1, pp. 157-197, Feb. 2007, doi: 10.1093/logcom/exl035.

[42] M. Lee, Consequence-Preserving Projections: Semantic Assurance for Agent Tool Interfaces, ver. 0.7.0 [Software]. RISU Institute, Aug. 2026, doi: 10.5281/zenodo.22149593.

Appendix A. Primary Retained Source Identities

The canonical empirical derivations are bound to the retained upstream identities below. Refs are abbreviated to 12 hexadecimal characters for readability; the Evaluation Capsule [27] preserves complete commit identities and retained-file SHA-256 values in its source identity ledger.

Table A1. Primary source identities retained by the frozen software artifact. Full source identities are preserved in the Evaluation Capsule [27].

Artifact	Repository / pinned ref	Retained scope
GH_SOURCE_MERGE	google/go-github @ 34349a88bac3	github/pulls.go; PullRequestOptions + merge request construction
GH_MERGE	github/github-mcp-server @ 822c87761f85	pkg/github/pullrequests.go; complete MergePullRequest function
GH_MINIMAL_PR	github/github-mcp-server @ same ref	pkg/github/minimal_types.go; complete MinimalPullRequest + branch types
AZ_COMMAND	microsoft/mcp @ 35089f45a44b	BlobUploadCommand complete class
AZ_OPTIONS	microsoft/mcp @ same ref	BlobUploadOptions complete class
AZ_SERVICE	microsoft/mcp @ same ref	StorageService.UploadBlob complete method
AZ_SDK	Azure/azure-sdk-for-net @ e6cb2e254d40	BlobClient.UploadAsync(Stream,bool,CancellationToken) complete overload

Appendix B. Frozen Holdout Retained Source Identities

FH-001 is bound to the public source identities below. Refs are abbreviated to 12 hexadecimal characters; the Evaluation Capsule [27] preserves the complete source refs and the SHA-256 identity of the original FH-001 qualification archive. Target implementation scopes were inspected only after CASE_LOCK.json was sealed.

Table B1. Primary retained identities for the prospective PayPal holdout. Complete source identities and the qualification archive identity are preserved in the Evaluation Capsule [27].

Artifact	Repository / pinned ref	Retained scope
PP_TARGET_TOOLS	paypal/agent-toolkit @ 2fe252586624	typescript/src/shared/tools.ts; create_order tool registration
PP_TARGET_PARAMS	paypal/agent-toolkit @ same ref	typescript/src/shared/parameters.ts; createOrderParameters
PP_TARGET_FUNCTION	paypal/agent-toolkit @ same ref	typescript/src/shared/functions.ts; createOrder effect path
PP_TARGET_CLIENT	paypal/agent-toolkit @ same ref	typescript/src/shared/client.ts; getHeaders / context.request_id mapping
PP_TARGET_LIFECYCLE	paypal/agent-toolkit @ same ref	typescript/src/ai-sdk/toolkit.ts; toolkit/client construction and tool reuse
PP_TARGET_PY_ORDER	paypal/agent-toolkit @ same ref	python/.../orders/tool_handlers.py; create_order
PP_TARGET_PY_CLIENT	paypal/agent-toolkit @ same ref	python/.../paypal_client.py; build_headers and post
PP_SOURCE_SDK	paypal/PayPal-TypeScript-Server-SDK @ b02a9b9c8f53	src/controllers/ordersController.ts; createOrder + PayPal-Request-Id mapping