

A Generative Relational Domain-Specific Language for Evolving Social-Fact Records:

Typed Assertions, Temporal Histories, and Governed Transitions

Wanhong HUANG
huangwanhong@serendip.ngo

Abstract

Public knowledge records need to preserve the distinction between a description, a person’s commitment to it, an evidentiary assessment, and an authorized institutional action. This paper specifies GRK-DSL 0.1, a domain-extensible language for evolving social-fact records organized around Entity, Relation, State, Property, Process, and Event. Technical assertion and commit structures add attribution, explicit polarity, validity intervals, revision history, and controlled operational effects. The method combines requirements reconstruction, verified comparison with established formalisms, concrete and abstract syntax, static judgments, denotational and operational semantics, and counterexample-based assessment. Proposition identity separates intensional descriptions from complete-field extensional content. Four support states retain explicit opposition within an authorized historical view. A finite positive rule core derives virtual predicates through a least fixed point, while separate workflow commands produce admitted token receipts. Compatible history union preserves disagreement and exposes operational conflicts requiring explicit repair. The paper proves finite rule termination, grounded non-explosion, targeted withdrawal locality, and deterministic projection of compatible admitted receipts under stated assumptions. It identifies implementation-level type preservation and observational disclosure protection as further obligations. Complete grammar and worked encodings accompany bounded syntax, semantic-oracle, and structural-schema checks. These artifacts establish an inspectable language contract rather than a complete interpreter or deployment result. Comparative implementation and domain studies remain necessary to assess expressivity, usability, cost, and the language’s contribution to responsible public knowledge evolution.

Keywords: domain-specific language; social-fact records; formal semantics; typed assertions; temporal history; explicit opposition; finite Datalog; provenance; workflow transitions; conformance.

Discussion Paper Note

This manuscript is a discussion paper circulated to support interdisciplinary inquiry into public knowledge. Its definitions, arguments, formal representations, and practical proposals remain open to revision. The paper provides a basis for scholarly exchange and further research; its circulation does not establish an adopted technical standard or an authoritative institutional position.

The author welcomes objections, discussion, corrections, alternative interpretations, counterexamples, and suggestions for further development. Contributions from different disciplines and from people whose knowledge or interests could be affected by the proposed infrastructure are especially valuable. Readers are invited to identify conceptual ambiguities, formal errors, omitted literature, relevant practices, and disagreements with the normative or institutional assumptions. Comments may be sent to huangwanhong@serendip.ngo. Identifying the paper, version, and relevant passage will assist review.

The author makes no claim to conceptual priority or first origination of the concepts discussed in this paper. Similar or equivalent concepts may already exist in other disciplines, intellectual traditions, languages, or bodies of practice, including under different terminology. The literature review is selective and remains open to correction and expansion. References to earlier, parallel, or independently developed work are welcome. The proposed vocabulary, connections, and applications should be assessed through their arguments and evidence. Attribution of this manuscript identifies its authorship and source; it does not establish priority for its underlying concepts.

Responsible Use and Rights Reservation

This section records a request for responsible scholarly conduct and clarifies the scope of the author's rights reservation. The author encourages good-faith criticism, independent inquiry, and responsible application, and asks users to consider foreseeable harms when adapting or applying the proposals. These ethical requests add no conditions to the licence stated in the Notices and leave its permissions and otherwise lawful uses unchanged.

Any reservation is limited to rights the author actually retains under the licence and applicable law, consistently with the licence's grants, waivers, and nonassertion provisions. The author asserts no exclusive control over the underlying concepts through this notice. Attribution and reuse do not imply authorial endorsement of subsequent interpretations, applications, or institutional positions. Applicable copyright exceptions and limitations remain available, and third-party material retains its applicable conditions.

Notices

These notices identify the manuscript’s discussion status, conditions of reuse, research context, and preparation process. They accompany the invitation to criticism and the statement on conceptual priority in the Discussion Paper Note.

Publication status and revision. This is a developing discussion paper. Its arguments, terminology, formal statements, organization, and numbering may change. Objections, discussion, corrections, and references to antecedent work are welcome. When citing or commenting on the paper, readers are asked to include its title, author, and any available version information so that later revisions can be distinguished.

Licence. Except where otherwise indicated, copyright 2026 Wanhong Huang. This manuscript is available under the Creative Commons Attribution-NonCommercial 4.0 International licence (CC BY-NC 4.0), subject to its official legal code. That legal code governs the permissions, conditions, disclaimers, and limitations of the licence. The responsible-use requests and rights reservation in this paper do not amend those terms. Third-party material remains subject to its applicable conditions.

Research scope and evidence. Constructed cases in this paper are analytical examples. The accompanying source audits record online verification and the limits of access. Formal results and bounded artifact checks are identified within their stated assumptions and scope. The project has no field deployment; institutional proposals remain proposals for discussion and evaluation.

AI assistance and author responsibility. The preparation of this manuscript involved OpenAI’s ChatGPT. AI assistance supported exploratory dialogue, source discovery and verification, argument criticism, conceptual and formal reconstruction, and manuscript preparation in L^AT_EX. The author retains responsibility for review and the final manuscript, including its arguments, representations, conclusions, and errors.

1 Introduction

Public knowledge records participate in processes of description, evidence assessment, interpretation, and institutional decision. A language intended to preserve those processes must distinguish what a record describes from who endorses it, when the description applies, and which action an institution has authorized. These distinctions become consequential when a claim is challenged, a source is withdrawn, a schema changes, or several custodians combine their histories. Treating every stored statement as an accepted fact would collapse the very differences that continued inquiry requires.

This paper specifies GRK-DSL 0.1, a domain-extensible language for evolving social-fact records. Its contribution is an integrated formal contract for attributed descriptions, temporal support, finite inference, and governed operational change. The proposal builds on established techniques for typed graphs, temporal information, explicit opposition, rule evaluation, and provenance. It brings their interfaces together around a particular problem: preserving disagreement and revision while making consequential changes explicitly attributable and operationally controlled.

1.1 A Running Representation Problem

Consider a constructed research inquiry in which Alice states that a dataset supports a claim and Bob disputes that relation. Their descriptions use the same schema and complete field values, although each submission has its own identifier. A later investigation leads Bob to withdraw his denial for the period beginning on 1 July while retaining it for June. The withdrawal enters the repository in August. An institutional review also changes its operative assessment of the claim. Several queries are now meaningful: what support was recorded before the withdrawal; what the present record says about June; what it says about July; whose commitments remain active; and which assessment the institution currently uses.

These questions require different coordinates. Record identifiers distinguish submissions. A proposition key determines whether two descriptions express the same qualified content. Assertion identifiers keep each speaker's commitment independently addressable. Valid time describes applicability; commit ancestry identifies a retained recorded history. An institution's workflow state follows admitted transition receipts and remains distinct from the support available for the claim. Current disclosure rules determine which parts of any reconstruction a requester may receive.

The example also exposes an operational hazard. A historical Event record describing a review cannot safely function as an instruction to execute that review again. Similarly, an assertion that a person possesses a permission cannot authorize the transaction introducing that assertion. The language therefore separates descriptive records from commands and places admission under a trusted host interface. That interface requires an institutional mandate and procedures through which its authority can be reviewed, as examined in Section 11. Technical execution provides no independent warrant for the underlying power.

1.2 Language Scope and Public Knowledge Purposes

A social-fact record, as used here, is a typed, situated description that can be attributed, interpreted, challenged, and revised. The language takes no position that every socially consequential occurrence is reducible to its record. Its six domain categories, Entity, Relation, State, Property, Process, and Event, provide a proposed structural basis for the selected tasks. Communities supply nominal domain types and their meanings. Category adequacy is assessed through concrete encodings and the inquiries those encodings support.

The intended users include schema authors, contributing organizations, custodians, query authors, and maintainers of host services. A contributor need not write source code personally: an interface can construct a candidate record from an attributable account and expose the encoding for review. Nevertheless, the formal contract should remain inspectable by independent implementers and critics. A human-readable vocabulary alone is insufficient if hidden execution rules change the meaning of a contribution.

The scope is deliberately bounded. The core includes immutable typed records, assertions with explicit polarity and intervals, withdrawals, transactions, workflow commands, and finite positive rules over signed relations. It excludes arbitrary event handlers, external side effects, probabilistic credibility aggregation, negation as failure, and claim-dependent workflow guards. More expressive profiles may be useful, but each would need its own semantics, interaction analysis, and evidence. A feature absent from version 0.1 cannot become an implicit premise of its guarantees.

1.3 Research Questions and Contributions

The inquiry addresses six connected questions. It examines the adequacy of the six-category basis; the composition of types, attribution, references and time; the separation of observation from execution; the interpretation of correction and branch histories; the restrictions supporting finite queries; and the boundary between language guarantees and delegated judgments. The questions concern both expressivity and discipline. A language can express a governance decision while remaining unable to determine whether that decision is justified.

The first contribution is a concrete grammar and typed abstract syntax tied to a versioned design artifact. The second is a denotational account of records, proposition identity, temporal activity, and four support states. The third is an operational account of atomic admission, workflow tokens, concurrent histories, and explicit conflict repair. The fourth is a finite query semantics with grounded recursive derivation and controlled explanation. The fifth is an assessment that separates mathematical consequences, unproved implementation obligations, bounded mechanical checks, and future domain evaluation.

The central design claim is that these mechanisms must compose without promoting descriptive availability into epistemic or institutional authority. A query result records support within a specified view. A successful transaction records accepted operations under a declared host decision. A signature or canonical digest concerns identity and integrity under its profile. None of those results alone settles the truth, public legitimacy, legal force, or complete intellectual ancestry of the represented account.

1.4 Method, Artifact Status, and Organization

The research method combines requirements reconstruction, primary-source comparison, formal specification, counterexample analysis, and bounded conformance checks. The normative artifact is the versioned GRK-DSL package: its lexical conventions, EBNF, type rules, semantic definitions, exchange schema, and examples are read together. The manuscript develops the arguments and formal relationships needed to assess that artifact independently. The complete grammar and selected source fixtures are reproduced in appendices directly from the package to avoid transcription drift.

Mathematical propositions below concern explicitly defined finite structures and admitted histories. Their proofs do not establish correctness of an unwritten interpreter. The available recognizer checks source syntax; separate semantic oracles exercise small mathematical models. No complete parser-to-execution pipeline, production policy service, federation deployment, or empirical usability result is claimed. This distinction permits precise positive results without obscuring the work that remains.

Sections 2–5 establish antecedents, requirements, syntax, and static semantics. Sections 6–9 develop records, time, transactions, and branch integration. Sections 10 and 11 address inference, disclosure, and exchange. Sections 12 and 13 assess examples, evidence, and formal properties. The conclusion identifies the language contract established here and the implementation and domain studies required to evaluate its practical contribution.

2 Related Languages and Semantic Foundations

GRK-DSL brings several established representational techniques into a common contract for attributed, revisable knowledge. Locating that contract requires comparing the questions answered by existing approaches: what a graph describes, how a description is constrained, what evidence supports an answer,

when a statement applies, and which operations change an admitted workflow. The comparison below identifies inherited mechanisms and the additional coordination that this project specifies. It does not establish that these mechanisms cannot be assembled in another language. An implementation using an existing graph or relational engine is a plausible realization of the design.

2.1 Graphs, Constraints, and Attributed Descriptions

The RDF data model supplies a general graph representation through subject–predicate–object triples, together with datatyped literals and datasets containing named graphs. Its resource concept accommodates documents, physical objects, and abstractions; relations with more than two participants can be represented indirectly. RDF also explicitly permits temporal descriptions through appropriate vocabularies, although the graph data model itself is atemporal [10]. These capabilities cover much of the structural substrate that GRK needs. The six GRK categories therefore constitute a modeling discipline imposed on a graph representation, with typed named roles and category-specific interpretation.

GRK adds an assertion envelope whose identity, speaker, polarity, context, validity interval, evidence references, and policy handle have specified roles in query evaluation and withdrawal. This envelope makes it possible to ask whether two people endorse the same complete description while preserving their separately withdrawable commitments. An RDF encoding can represent these objects and their relations. Its adequacy would depend on an accompanying profile that fixes the same proposition equality, attribution, temporal selection, and lifecycle semantics. Merely choosing a named graph for each contributor leaves those choices open. Conversely, a translation that preserves these choices could be a useful interoperability implementation rather than a competing interpretation of the project.

Constraint checking is similarly established. SHACL distinguishes data graphs from shapes graphs and supplies constraints on graph structures; closed shapes can restrict a node’s permitted properties [11]. GRK’s exact record fields and pure local checks occupy a narrower space. They determine whether an immutable record conforms to its pinned nominal declaration. A wrong reference category is an admission error, whereas two attributed descriptions that disagree may both be well formed. This distinction matters for a repository whose purpose includes preserving disputes: structural validation should not silently become a decision about whose account deserves to remain visible.

SPARQL provides graph-pattern querying, projection, and property paths, including paths of arbitrary length [9]. GRK’s recursive genealogy queries consequently introduce no claim of having invented graph traversal. Their contribution is a fixed evaluation context: a selected admitted history, a validity instant, a context of assertion, and a currently authorized view. The syntax also separates virtual rule consequences from new committed records. A query can establish that a path exists in the selected representation without making a new attributed assertion or authorizing a practical action.

2.2 Provenance and Temporal Interpretation

PROV-DM already distinguishes entities, activities, and agents and provides derivation, attribution, association, and qualified relations. Its account of revision connects a revised entity to its predecessor, and its account of primary sources acknowledges domain conventions in identifying what counts as a primary source [8]. GRK inherits the value of explicit provenance relationships. Its Process and Event categories, however, must be interpreted under the GRK specification; a shared English label does not establish a type equivalence with a PROV term. Mappings need to state which roles, identities, and temporal meanings survive translation.

Provenance also has a logical dimension. Green, Karvounarakis, and Tannen define semiring annotations for relational queries and extend the approach to Datalog, including formal power series for potentially infinite derivational provenance [4]. This work explains why a finite answer set and a finite enumeration of all proof trees are different requirements. GRK uses finite virtual answer relations and permits bounded witness representations for explanations. It does not claim a complete implementation of provenance semirings. A future annotation profile would need to specify how alternative derivations,

recursive dependence, restricted premises, and withdrawal affect annotations before inheriting any algebraic guarantees.

Temporal database research supplies the distinction between valid time in the modeled reality and transaction time associated with a database object's presence in recorded history [5]. GRK adapts this distinction to attributed assertions: an applicability interval reports the interval for which an assertion is offered, while a commit cut specifies the admitted history used to interpret it. Because the infrastructure preserves uncertain and opposing descriptions, applicability does not certify truth during that interval. Event occurrence and process extent are separately typed payloads. A correction recorded in August can withdraw an assertion's July applicability without changing what a June commit contained.

The use of a commit graph also draws on the distinction between causal partial order and physical-clock readings established by Lamport [6]. GRK defines ancestry through admitted parent links. A sender's timestamp therefore cannot determine a unique global snapshot or adjudicate concurrent branches. This is a design consequence of using explicit histories, not a proposal for a new clock algorithm. A service may associate local receipt times with stored cuts, provided it reports that mapping and preserves the cut actually evaluated.

2.3 Explicit Opposition, Rules, and Operational Change

Belnap's four-valued approach to information processing was motivated by the need to accommodate inconsistent data without deriving irrelevant conclusions [2]. GRK shares that limited motivation. Its support observation records the presence or absence of positive and negative support for an exact proposition within a view, yielding positive-only, negative-only, both, or neither. The language does not supply a complete four-valued truth-functional connective system. Its signed predicates instead become separate relations consumed by a positive rule engine. This distinction prevents a suggestive philosophical analogy from being mistaken for an unstated logical calculus.

The rule semantics follows the established fixed-point treatment of logic programs associated with van Emden and Kowalski, specialized to finite, function-free Datalog [7, 1]. The familiar restrictions on head variables and active-domain values support a constructive evaluation over finitely many ground atoms. GRK adds type signatures and signed base predicates while excluding negation-as-failure, fresh value construction, and effects. These exclusions are substantive: a rule can find an explicitly denied relation, but an unsuccessful search cannot manufacture a denial. The resulting termination argument concerns a fixed, finite authorized view; it supplies no general bound on acceptable service latency.

Operational history requires a further distinction. Fowler's author-hosted account of event sourcing describes reconstruction from retained events and explicitly discusses the danger of repeating external updates during replay [3]. GRK separates a reported occurrence from an admitted command and its token receipt. A historical Event record can be imported without advancing a review run; replay projects retained receipts without executing an external effect. The proposed integration therefore coordinates descriptive graphs, attribution, finite inference, and bounded workflow commands. Its scholarly test is whether that contract preserves meaningful distinctions across the shared cases at an acceptable representational and institutional cost. Comparison with RDF/PROV/SHACL/SPARQL, relational assertion stores, and typed Datalog should measure that preservation under equivalent assumptions.

3 Representational Requirements and Language Scope

The language requirements follow from the infrastructure's purpose of preserving contributions so that later participants can examine, challenge, and reuse them. A useful formalization must keep the differences between occurrence, evidence, attributed representation, institutional recognition, and legal status available for subsequent interpretation. This section translates that purpose into language obligations and identifies the decisions that remain with domain communities and host institutions. The requirements are design criteria derived from the project cases, rather than findings from a completed elicitation study.

Table 1: Language requirements and discriminating failures.

Requirements	Required distinction	Failure that the specification must expose
R01, R02	Attribution and exact domain vocabulary	Identical labels silently identify distinct contributors or schema versions.
R03, R04	Applicability, history, and opposition	A later correction rewrites an earlier cut, or missing support becomes denial.
R05, R07	Revision, genealogy, and compatible exchange	Import discards a branch, erases attribution, or repeats a command.
R06	Finite inference and atomic transitions	A rule generates unbounded fresh terms, or a failed transaction leaves partial effects.
R08, R09	Contestation interfaces and selective disclosure	A declared permission authorizes itself, or an explanation exposes a restricted premise.
R10, R11	Retention conditions and translation loss	A hash is presented as a reconstructible payload, or a mapping silently changes meaning.
R12	Evidence proportional to the artifact	Syntax acceptance is reported as full semantic or deployment conformance.

3.1 Social Facts as Attributed Representations

The expression *social fact* names the project’s subject matter without turning every stored item into an established fact. A laboratory observation, an account of participation, a disputed interpretation, and an institutional finding can all enter the infrastructure, but they enter through distinguishable representations. A material occurrence can leave several evidential traces and support several descriptions. An institution may recognize one description for a specified purpose, and a legal system may attach consequences under rules that exceed the repository’s vocabulary. GRK must preserve enough structure to relate these layers without inferring that storage, endorsement, recognition, and legal effect are interchangeable.

The six categories provide a practical vocabulary for this task. An *Entity* supplies an identified participant or referent; a *Relation* describes named roles among referenced objects; a *State* attributes a condition to a subject; a *Property* attributes a value; a *Process* represents an extended course of activity; and an *Event* represents an occurrence. These are categories of representation, and their adequacy is subject to domain evaluation. A claim may be modeled as an *Entity* so that people can refer to it, while relations express its evidential support, interpretation, or acceptance. Endorsing that *Entity*’s description does not automatically endorse the sentence carried by its content field. Domain libraries must make the intended propositional commitment explicit.

This compositional approach avoids adding a primitive for every institutional concept, yet places responsibilities on schema authors. A recognition record needs to identify its institution, object, scope, and basis through declared fields or linked records. A numerical credibility assessment needs a method and an interpretation; the core supplies no universal confidence scale. A community that distinguishes testimony from measurement can encode that distinction without requiring every community to adopt the same evidential taxonomy. Such flexibility is meaningful only when consumers can recover the exact schema and see the mappings used to compare it with another schema.

3.2 Preservation, Contestation, and Temporal Accountability

The principal obligations are grouped in Table 1, which retains the identifiers of the normative design dossier. The table connects each group to a discriminating failure, so that evaluation can examine semantic consequences rather than count supported keywords.

Attribution is central because public revisability depends on knowing which commitment can be challenged or withdrawn. Several speakers may endorse one complete description through different assertions. A withdrawal addresses one such assertion and an applicability interval; it leaves the other speakers’ commitments independently interpretable. Opposing assertions likewise need a precise com-

parison scope. Two communities may use the same label for different predicates, or the same predicate in different contexts. Treating all such statements as a single contradiction would suppress the differences needed to understand their disagreement.

Identity must serve both genealogy and comparison. Distinct Event records can describe different alleged occurrences with similar payloads; collapsing them would lose the identity of the reported occurrence. Two Relation records with exactly the same typed fields can instead express the same description despite their different storage identities. GRK makes that distinction explicit in its proposition key, developed in Section 6. It remains conservative about referenced objects: similar text, embeddings, and asserted alignment do not silently equate identifiers. This choice may fragment a body of knowledge until a domain mapping is supplied. The fragmentation is an acknowledged cost of preserving the premises of comparison.

Temporal accountability requires reconstructing what a selected history permits one to say about a selected applicability instant. It does not require indefinite public availability of every payload. Withdrawal, physical erasure, sealed evidence, and current disclosure restrictions have different effects. In particular, an inaccessible withdrawal reason cannot restore the support that the withdrawal removed. An incomplete host must report an appropriately restricted or unavailable answer. The language’s historical contract thus depends on retained dependencies and current authorized access, which Section 7 incorporates into the query model.

3.3 Executable Boundaries and Institutional Responsibility

A process description and a workflow command answer different questions. The former can describe an informal inquiry, a historical proceeding, or an activity whose boundaries are uncertain. The latter requests a specified change to a bound run under an admitted machine version. GRK-DSL 0.1 includes `start`, `advance`, and `conflict resolve`; it excludes arbitrary handlers and epistemic guards. Consequently, an asserted conclusion, a newly entered role description, or an imported Event does not itself constitute permission to execute. Host authorization must come from the trusted context against which the transaction is assessed.

The same boundary applies to query results used by AI systems. A finite rule can recover an explicitly encoded genealogy; it cannot certify that the genealogy exhausts the intellectual sources of a contribution. A model-generated draft can be stored, attributed, compared, and reviewed, while its adoption requires a separate accountable act. The public purpose is to preserve the distinction between generation and adoption across successive representations. Determining acceptable evidence of influence, lawful reuse, or institutional competence remains a substantive governance task.

The release is intentionally finite in syntax, values present in each program, and query grounding. Probabilistic inference, arbitrary aggregation, identity alignment, unrestricted term construction, complex workflow guards, and external effects require separately specified extensions. A Count Property in a release example records a supplied result; it does not add an aggregation operator to the language. These boundaries allow the present formal claims to be stated precisely. They also leave a research obligation to test whether contributors can express the shared cases without excessive annotation work, misleading categories, or loss of context. Formal well-formedness is one criterion within that assessment, alongside domain adequacy and the institutions needed to make participation and correction practicable.

4 Concrete Syntax and Abstract Representation

The source syntax makes consequential distinctions visible at the point of authoring: a record is introduced separately from its assertion, a withdrawal names a commitment, and a command names the run it intends to change. This section presents representative excerpts and an abstract syntax that connects those forms to the formal model. The complete EBNF in the artifact package is authoritative for punctuation and production structure. The excerpts below are copied verbatim from the complete grammar-validated fixtures; their synthetic authority assumptions do not establish live admission.

4.1 Modules, Names, and Literal Forms

A source file begins with a module name, human version, and absolute namespace URI. Imports precede declarations, declarations precede transactions, and queries follow transactions. An import fixes a module name, version, artifact digest, namespace, and local alias. Resolution uses an explicitly provided artifact catalog. Reading an import does not fetch a remote resource or execute code. The enclosing package supplies the current module's digest, avoiding a schema artifact that would have to contain its own hash.

Names are case-sensitive ASCII identifiers composed of letters, digits, and underscores, with a letter or underscore first. Qualified names separate components with `::`. Keywords are contextual, so a schema can declare the required field `subject` and a fixture can use `@policy`. The semantic resolver still rejects declarations that shadow built-in type names. Human names and descriptions use `Text`, which supports Unicode independently of the identifier alphabet. This choice reduces identifier ambiguity while leaving the content vocabulary multilingual; it does not solve translation or transliteration policy.

References beginning with `@` expand under the module namespace or an explicitly imported alias. Under the namespace in Listing 1, for example, unqualified `@alice` expands to a namespace-qualified identifier ending in `alice`. An imported alias is unique and cannot shadow a local name or the first component of a local qualified reference. Local multi-component suffixes retain their literal `::` separators. These rules make abbreviated source references reproducible under a fixed catalog rather than dependent on the receiving service's preferred names.

The UTF-8 source admits whitespace and `//` comments outside strings. Strings use JSON escaping and reject invalid escapes, raw control characters, and lone Unicode surrogates; no Unicode normalization is implicit. Integers and decimals have distinct literal forms and types. An integer has no leading zero except zero itself; a decimal contains a decimal point with digits on each side. Maximal token matching gives decimals precedence over integers, while the sign in an assertion remains separate from its record reference. Negative zero and insignificant trailing decimal zeros normalize without coercing a `Decimal` into an `Int`. Exponents, infinities, and `NaN` are outside this release.

Typed constructors prevent a quoted string from being silently treated as a time or IRI. The forms `time(...)` and `iri(...)` carry explicit tags; `some(...)` and `none` construct optional values. Lists use square brackets, whereas mathematical sets use `set[...]` and canonical typed equality. A half-open interval has a finite start and either a later finite endpoint or `*`. An `Extent` is `at(time(...))`, `during(interval)`, or `unknown`. In an `Event`, `during` reports an occurrence somewhere within the stated bounds. In a `Process` it reports the represented span. Neither form supplies an unstated probability distribution over time.

4.2 Declarations and Attributed Transactions

Listing 1 shows all six categories in the contested-knowledge fixture. Fields are named in record payloads; their order in a declaration also defines the positional signature exposed to rules. `PositiveSize` checks a supplied value, `AssessmentChoice` identifies incompatible positive descriptions, and the rules define virtual genealogy and opposition predicates. The language assigns no automatic transitivity to the English name `Supports`. The recursive `RevisionRoute` rule specifies the intended composition explicitly.

Listing 1: Declarations from the contested-knowledge fixture.

```
module example_knowledge version "0.1.0" namespace "urn:grk:example:knowledge:";

enum Stage { proposed, contested, supported, withdrawn };
entity Person { label: Text; };
entity Context { label: Text; };
entity Policy { label: Text; };
entity Claim { content: Text; };
entity Dataset { label: Text; };
entity Reason { text: Text; };
relation Supports { source: Ref<Dataset>; target: Ref<Claim>; };
relation Revises { prior: Ref<Claim>; next: Ref<Claim>; };
```

```

state Assessment { subject: Ref<Claim>; value: Stage; };
property SampleSize { subject: Ref<Dataset>; value: Int; };
process Analysis { span: Extent; input: Ref<Dataset>; output: Ref<Claim>; };
event EvidenceAdded { occurred: Extent; subject: Ref<Claim>; evidence: Ref<Dataset>; };
check PositiveSize on SampleSize: value > 0;
exclusive AssessmentChoice on Assessment by (subject);
rule EvidenceRoute: +Reach(?source, ?target) <- +Supports(?source, ?target);
rule RevisionRoute: +Reach(?source, ?next) <- +Reach(?source, ?prior), +Revises(?prior, ?next);
rule OpposedRoute: +Opposed(?source, ?target) <- +Supports(?source, ?target), -Supports(?source, ?target);

```

A transaction names its commit identity, parents, submitter, requested receipt time, and policy handle, followed by an ordered block of operations. The complete fixture’s initial transaction supplies the participants and a positive assertion. Listing 2 then supplies Bob’s denial and its later interval-scoped withdrawal. These are dependent excerpts from that fixture, rather than independent genesis programs. The reversed field order in @support_copy deliberately tests canonical field equality. Bob withdraws @b1; the source contains no instruction to withdraw Alice’s assertion or to infer the opposite of Bob’s earlier statement.

Listing 2: Opposition and subsequent withdrawal from the contested-knowledge fixture.

```

transaction @c1 parents [@c0] by @bob recorded time("2026-07-01T09:00:00.000000Z") policy @public_policy {
  // Different record identity, same extensional proposition key.
  record @support_copy: Supports { target = @claim; source = @dataset; };
  record @assessment_b: Assessment { subject = @claim; value = Stage.contested; };
  assert @b1 - @support_copy by @bob in @context valid [time("2026-06-01T00:00:00.000000Z"), *) evidence [
    @dataset] policy @public_policy;
  assert @b_assessment + @assessment_b by @bob in @context valid [time("2026-06-01T00:00:00.000000Z"), *)
    evidence [] policy @public_policy;
}

transaction @c2 parents [@c1] by @bob recorded time("2026-08-01T09:00:00.000000Z") policy @public_policy {
  record @correction: Reason { text = "The challenge is withdrawn for the interval beginning July 1 after
    review of the method."; };
  withdraw @w1 assertion @b1 valid [time("2026-07-01T00:00:00.000000Z"), *) reason @correction;
}

```

Workflow declarations are similarly explicit. Listing 3 declares an initial enum value and named steps between enum values, each associated with an Event type. There is no expression slot for a logical guard or external handler in this grammar. A separate start binds a Process run, machine, and subject using an Event reference; advance names a step, Event, and expected token list; resolve names the full conflicting head set, a selected enum value, and a reason. Their dynamic conditions are defined in the operational semantics.

Listing 3: Machine declaration from the workflow-branches fixture.

```

machine ReviewFlow tracks Review {
  initial Stage.proposed;
  step challenge from Stage.proposed to Stage.contested on EvidenceAdded;
  step accept from Stage.proposed to Stage.supported on Accepted;
  step reconsider from Stage.contested to Stage.supported on Accepted;
};

```

4.3 Abstract Syntax and Elaboration

An abstract representation removes source punctuation while preserving semantic choices. In Equation 1, n denotes a resolved name, v_m a version label, ν a namespace, $\vec{r}$ imports, $\vec{d}$ declarations, $\vec{x}$ transactions, and $\vec{q}$ queries. A declaration retains its artifact identity through its resolved name or type T . The category k ranges over all six domain categories; F is an ordered field signature, L a nonempty enum label set, K an exclusive diagnostic field key, and φ a pure check expression. A machine’s steps name source and target enum values and an Event type. The operator $|$ separates constructor alternatives.

$$\begin{aligned}
m &::= \text{Module}(n, v_m, \nu, \vec{\iota}, \vec{d}, \vec{x}, \vec{q}), \\
\iota &::= \text{Import}(n, v_m, d_g, \nu, n_{\text{alias}}), \\
d &::= \text{EnumDef}(n, L) \mid \text{TypeDef}(k, n, F) \\
&\quad \mid \text{CheckDef}(n, T, \varphi) \mid \text{ExclusiveDef}(n, T, K) \\
&\quad \mid \text{MachineDef}(n, T, \ell_0, \vec{s}) \mid \text{Rule}(n, a, \vec{a}), \\
s &::= \text{Step}(n, \ell, \ell', T_e).
\end{aligned} \tag{1}$$

Equation 2 covers every core operation. Here i is the operation's identity, r a record reference, b a speaker, x_c an assertion context, I an interval, $\vec{e}$ evidence references, and p a policy handle. In workflow operations, u is a Process run, M a machine, z its subject, e an Event, $\vec{h}$ token references, and ℓ a selected enum value. The final argument r_{why} references a reason record. Separate reference registries disambiguate these roles even where source spelling is shared.

$$\begin{aligned}
o &::= \text{Record}(i, T, v) \\
&\quad \mid \text{Assert}(i, \sigma, r, b, x_c, I, \vec{e}, p) \\
&\quad \mid \text{Withdraw}(i, a, I, r_{\text{why}}) \\
&\quad \mid \text{Start}(i, u, M, z, e) \\
&\quad \mid \text{Advance}(i, u, n_{\text{step}}, e, \vec{h}) \\
&\quad \mid \text{Resolve}(i, u, \vec{h}, \ell, r_{\text{why}}), \quad \sigma \in \{+, -\}.
\end{aligned} \tag{2}$$

Elaboration attaches a verified profile identity and receipt to an accepted transaction. Equation 3 represents the resulting commit, with parent references $\vec{c}$, submitter b , receipt time t_r , ordered operations $\vec{o}$, profile π , and validation receipt ρ . The source omits π and ρ ; the accepting host supplies or verifies them. Its recorded field is a requested value useful in offline fixtures. A live host assigns its authoritative local receipt time and reports a change before the final commit is signed.

$$x = \text{Commit}(i, \vec{c}, b, t_r, p, \vec{o}, \pi, \rho). \tag{3}$$

Queries share a source envelope but have four distinct bodies. Equation 4 records a cut reference c , valid instant t , context x_c , and requested policy p . An atom has a polarity, predicate, and terms; a term is a variable or a fully constructed typed value. Rule and selection bodies are nonempty. Selection returns bindings; support examines a base proposition; history examines direct record-related history; current examines a workflow run. The latter two bodies use the cut without reinterpreting their output as validity-filtered assertion support. The authenticated request supplies requester, purpose, and current policy epoch outside this source envelope.

$$\begin{aligned}
q &::= \text{Query}(n, c, t, x_c, p, b_q), \\
b_q &::= \text{Select}(\vec{z}, \vec{a}) \mid \text{Support}(r) \mid \text{History}(r) \mid \text{Current}(u), \\
a &::= \text{Atom}(\sigma, P, \vec{\tau}), \quad \tau ::= z \mid v.
\end{aligned} \tag{4}$$

Source normalization, abstract equality, and signed interchange bytes remain distinct contracts. Reordering payload fields leaves their typed map unchanged; reordering a type declaration's fields changes its predicate signature and hence its schema artifact. A canonical interchange profile must preserve value tags, exact type identities, reference kinds, and operation order. Grammar recognition alone verifies none of those semantic conditions. The abstract constructors make their locations explicit so that a future elaborator and type checker can be assessed against one coherent specification.

5 Types, References, and Static Semantics

Static semantics establishes which source structures have an interpretation under an exact schema. It must reject a Property with an ill-typed value without rejecting a well-typed disagreement, and it must

recognize the shape of a workflow command without mistaking that recognition for authorization. This section defines the principal judgments, their reference environments, and the restrictions that make finite rule evaluation possible. They are specifications for a future complete type checker. The supplied syntax recognizer and bounded semantic oracles provide narrower evidence, as the conformance assessment explains.

5.1 Nominal Types and Finite Values

Let Γ be a finite resolved environment containing record declarations, enum declarations, checks, exclusive keys, machine definitions, rule signatures, and pinned import dependencies. Exact domain type identity is defined by Equation 5, where ν is the module namespace, d_g its verified artifact digest, and n_T the local type name. The human version label aids coordination but does not replace the digest in identity.

$$\text{TypeId}(T) = (\nu, d_g, n_T). \quad (5)$$

Two declarations with the same fields are consequently distinct types if their exact identities differ. The same applies to enums whose labels happen to coincide. Locally declared record types have no implicit inheritance, and imported declarations cannot be overridden. The built-in names `Object`, `Entity`, `Relation`, `State`, `Property`, `Process`, and `Event` serve as abstract reference bounds; they are unavailable as record constructors. A bare named field type denotes an enum. References express links to domain records, preserving object identity explicitly rather than embedding a second anonymous record inside a payload.

The scalar domains are `Text`, `Bool`, `Int`, `Decimal`, `Time`, `Interval`, `Extent`, `IRI`, `AssertionRef`, and `CommitRef`. Every concrete value is finite. This does not mean that every type has a finite mathematical domain: arbitrary-precision integers admit arbitrarily large finite values. `Int` and exact base-ten `Decimal` remain distinct even when they express an equal mathematical quantity. An enum has a nonempty finite set of distinct labels. `List<T>` preserves order and repetitions; `Set<T>` eliminates repetitions by typed canonical equality and has canonical serialization order. Generic containers are invariant, so their element type cannot silently broaden during assignment.

An `Option<T>` field is always present and contains either `none` or `some(v)` with $v : T$. Optionality therefore belongs to the value, rather than to an accidental omission of a field. An empty collection receives its element type from the expected declaration; it does not license heterogeneous values later in the same collection. The core has no implicit null propagation, numeric coercion, or conversion from a textual date to `Time`. These choices make payload equality and predicate matching stable under exchange.

The temporal profile admits valid UTC timestamps with exactly six fractional digits and a terminal `Z`, rejecting invalid calendar values and leap-second spellings. An interval starts at a finite instant and ends strictly later or remains unbounded above. `unknown` is an explicit `Extent` value; it cannot replace a missing `occurred` or `span` field. This permits uncertainty without making record shape uncertain. A more expressive uncertainty model would require its own typed representation and interpretation.

5.2 Reference Resolution and Record Judgments

Let Δ_O map domain record identifiers to their exact types, and let separate environments Δ_A , Δ_H , and Δ_K track assertion, commit, and command/token identifiers. Their identifier kinds are disjoint within a packet. The shared source form `@name` does not allow an assertion identifier to masquerade as a domain object. We abbreviate these coordinated environments by Δ when the reference kind is unambiguous.

Equation 6 specifies domain reference typing. The relation $T \preceq U$ holds for exact equality, for membership of nominal T in the indicated built-in category, or for the `Object` bound. It supplies no structural subtype rule between unrelated nominal declarations. Corresponding registry judgments type `AssertionRef`, `CommitRef`, and operational token references.

$$\frac{\Delta_O(i) = T \quad T \preceq U}{\Gamma; \Delta \vdash @i : \text{Ref}(U)}. \quad (6)$$

Record construction uses two phases. The validator first registers the identifiers and declared types of all domain records introduced by a transaction, checking identity-kind collisions and incompatible reuse. It then validates their complete payloads against the enlarged environment. A Relation may thus reference another record appearing later in the source, and two records may refer to one another. The payloads are finite graphs expressed through opaque identifiers; they do not require mutually recursive content hashes. Two-phase registration is limited to domain records. It does not move an assertion or withdrawal ahead of its specified operational position.

Equation 7 requires the exact declared field set, a typed value for each field, and successful local checks. Here F_T is the ordered schema signature, while $\text{fields}(v)$ is the payload's key set. The source must supply each key exactly once before it can elaborate to a field map. The auxiliary Checks_T is the conjunction of the total checks declared for T , with the empty conjunction true.

$$\frac{\Gamma(T) = F_T = (f_j : U_j)_{j=1}^n, \quad \text{fields}(v) = \{f_1, \dots, f_n\} \quad \forall j \Gamma; \Delta \vdash v[f_j] : U_j, \quad \text{Checks}_T(v) = \text{true}}{\Gamma; \Delta \vdash \text{Record}(i, T, v) \text{ ok}}. \quad (7)$$

Unknown fields, missing fields, and wrong-category references fail this judgment. Reusing an existing record identity also requires identical exact type and canonical payload; Equation 7 does not by itself authorize identity replacement. A receiver that cannot obtain a required dependency cannot establish local validity by guessing its type from a label. An exchange service may hold an unresolved packet separately, but such a packet has not become a locally validated record.

Category well-formedness refines the generic judgment. A State declares `subject`: `Ref<T>` and at least one further field. A Property declares `subject`: `Ref<T>` and `value`: `U`, with optional additional declared qualifiers. A Process declares `span`: `Extent`; an Event declares `occurred`: `Extent`. These required fields coexist with other exact fields under the same nominal schema. Relations use explicitly named roles, which can reference any permitted domain category. A set-valued role can represent undirected participation; a collection's multiplicity convention and a Relation's symmetry are separate modeling decisions.

5.3 Assertions, Checks, and State Machines

An assertion requires a typed record target, Entity references for its speaker, context, and policy descriptor, Object references for its evidence, and a well-formed interval. Equation 8 states this contract compactly. The predicate $\text{Ent}_\Delta(i)$ abbreviates a derivable Entity-reference judgment, and $\text{Obj}_\Delta(i)$ the corresponding Object judgment. Identity freshness or identical replay is checked by the enclosing transaction protocol.

$$\frac{\text{Obj}_\Delta(r), \quad \sigma \in \{+, -\}, \quad \text{Ent}_\Delta(b) \wedge \text{Ent}_\Delta(x_c) \wedge \text{Ent}_\Delta(p) \quad \Gamma; \Delta \vdash I : \text{Interval}, \quad \forall e \in \vec{e} \text{ Obj}_\Delta(e)}{\Gamma; \Delta \vdash \text{Assert}(i, \sigma, r, b, x_c, I, \vec{e}, p) \text{ ok}}. \quad (8)$$

Positive and negative assertions have identical typing premises. The sign changes the attributed commitment, rather than the record's structure. A denial concerning an intensional Event therefore targets the same Event record; a separately identified alternative occurrence requires an explicit relationship if the domain wishes to compare it. A withdrawal must resolve an assertion from the ancestor history or an earlier assertion operation in the same transaction, together with its interval and reason reference. Permission to withdraw is checked separately against the trusted admission context.

A check is a total Boolean expression over one payload. The concrete grammar permits field-to-value comparisons, `is_some`, conjunction, disjunction, and parentheses. Equality uses typed value equality; ordering is available for `Int`, `Decimal`, `Time`, and `Text`, with `Text` ordered by Unicode scalar values.

The compared operands must have the same type, including when an operand is an Option. Checks neither dereference neighboring records nor consult time, policy, or external code. These restrictions make their results reproducible from the record and exact schema alone.

An `exclusive` declaration has a nonempty key of distinct declared fields and omits at least one field of its State or Property type. Equal key values with differing remaining values can identify incompatible active positive descriptions in a selected context and time. That result is a diagnostic. It does not turn one positive assertion into a negative assertion or reject the existence of the disputed evidence. In particular, an `Assessment` marked `proposed` and another marked `contested` are distinct descriptions whose relation is reported through the declared diagnostic key.

A machine deliberately tracks a narrower State shape. Its signature contains exactly `subject: Ref<U>` and `value: E`, with E an enum. Equation 9 states the associated declaration obligations: the initial value and every step endpoint belong to E , and each step's exact Event type T_e declares the same subject-reference bound U . The last two conditions apply to every $\text{Step}(n, \ell, \ell', T_e)$ in M . Field-set equality leaves the schema's declared field order intact.

$$\begin{aligned} \text{fieldsig}(T_s) &= \{\text{subject} : \text{Ref}(U), \text{value} : E\}, \\ \ell_0 &\in E, \\ \ell, \ell' &\in E, \quad \text{kind}(T_e) = \text{Event}, \\ \Gamma(T_e)[\text{subject}] &= \text{Ref}(U). \end{aligned} \tag{9}$$

General State records may contain richer domain descriptions without becoming machine tokens. A run reference resolves to a Process. Start requires an Event with the compatible subject field, but the machine declares no single nominal initial Event type. Advance requires the exact Event type named by its selected step. At execution, either Event's subject value must equal the run's bound subject. An Event for claim B cannot advance the review of claim A merely because both refer to the type Claim.

Expected-token and resolution-head lists contain distinct token identifiers. Their equality to the current maximal token set, the singleton condition for ordinary advance, and the requirement of at least two conflicting heads for resolution depend on the selected operational history. They therefore remain dynamic premises. The same is true of authority. A policy descriptor satisfying Equation 8 identifies a data object; the descriptor's existence or endorsement cannot grant the operation that introduces it.

5.4 Predicate Signatures and Rule Safety

Each domain type exposes positive and negative base predicates with the same signature. Equation 10 fixes their argument order from the declaration $F_T = (f_j : U_j)_{j=1}^n$. Entity, Process, and Event predicates have a leading identity reference; Relation, State, and Property predicates consist of complete fields. This convention aligns matching with the proposition equality defined in Section 6. The term “intensional” there concerns record identity and should be distinguished from an intensional, or derived, Datalog predicate.

$$\text{sig}_\Gamma(T) = \begin{cases} (\text{Ref}(T), U_1, \dots, U_n), & \text{kind}(T) \in \{\text{Entity}, \text{Process}, \text{Event}\}, \\ (U_1, \dots, U_n), & \text{kind}(T) \in \{\text{Relation}, \text{State}, \text{Property}\}. \end{cases} \tag{10}$$

Thus `+Person(?person, ?label)` contains an identity argument, while `+Supports(?source, ?target)` follows its two role fields. Introducing a third qualifier into `Supports` changes the predicate's arity and its complete descriptive content. Rule heads introduce module-local derived predicates and may not reuse a record type, built-in query predicate, or incompatible signature. Multiple defining rules must agree on arity and argument types. Every body predicate resolves to a declared base or derived predicate, with forward references permitted among rules.

Variable typing collects constraints from all predicate positions and constants. Each variable must have a consistent exact type or a compatible built-in reference bound. Recursive strongly connected components require inferable signatures anchored by typed predicates or constants; an ambiguous component

is rejected. The specification does not claim an implemented inference algorithm or a principal-type theorem. Literal constructors contain no variables, so a term is either a variable or a fully typed constant, including an already finite structured constant.

The range condition in Equation 11 requires every head variable to occur in a nonempty body. The judgment $\Gamma; \Theta \vdash a$ atom checks the atom against its predicate signature using variable environment Θ . This rule is subject to the declared-predicate and type-consistency conditions just described.

$$\begin{array}{c} m \geq 1, \quad \text{Vars}(h) \subseteq \bigcup_{j=1}^m \text{Vars}(a_j), \\ \Gamma; \Theta \vdash h \text{ atom}, \quad \forall j \Gamma; \Theta \vdash a_j \text{ atom} \\ \hline \Gamma \vdash (h \leftarrow a_1, \dots, a_m) \text{ safe} \end{array} \quad (11)$$

Both +P and -P are relational membership tests, so a variable occurring only in a negative-sign body atom is range-bound. There is no negation-as-failure condition requiring another positive-sign atom. The safety discipline excludes existential head variables, function application, arithmetic generation, aggregation, and side effects. A finite query's active domain consists of visible base values and program constants. Existing lists and sets can be matched as indivisible values, but rule heads cannot construct larger ones. The finite-grounding argument is inherited from Datalog's restrictions [1]; its application to GRK additionally depends on the authorized view being fixed during evaluation.

Selection variables obey the same body-binding condition and results have set semantics. A support query requires a domain record, a history query a domain record identifier, and a current query a Process run. A source query's policy handle cannot enlarge the authenticated requester's disclosure rights. These requirements give useful diagnostics before evaluation while leaving historical availability, authority, and result disclosure to the relevant host judgments.

5.5 Diagnostics and Schema Evolution

Diagnostics distinguish malformed syntax (E_PARSE), invalid or cyclic imports (E_IMPORT), structural typing (E_TYPE), invalid time values (E_TIME), identity collision (E_ID), failed local checks (E_CHECK), unsafe or ambiguous rules (E_RULE), and unavailable required references (E_REFERENCE). Denied authority (E_AUTH) and stale or conflicting workflow state (E_CONFLICT) belong to dynamic admission. A diagnostic should identify a source location and reason insofar as doing so does not disclose protected dependencies. A well-typed transaction may therefore fail admission, while a well-typed contradictory assertion may be admitted and remain contested.

Schema evolution creates a new exact artifact and an explicit mapping with identified inputs, outputs, failures, and information loss. Changes to enum labels, field order, proposition keys, or machine transitions can change meaning even under a seemingly minor version increment. The core rejects unsupported required profiles. Unknown optional annotations can remain opaque only when they cannot alter validity, authority, equality, or inference. This compatibility discipline makes historical interpretation inspectable, while leaving the substantive adequacy of a migration open to review. It cannot by itself establish that a revised classification treats affected people fairly or that an institution has authority to impose it.

6 Denotational Record Model and Epistemic Interpretation

The record model determines what a well-typed description means for identity, attribution, disagreement, and inference. This section defines the semantic domains and proposition keys, then derives the interpretation of support and its limits. The definitions concern represented content under a declared schema. They preserve the possibility that a structurally valid account is inaccurate, misleading, or institutionally contested.

6.1 Records, References, and Represented Content

Let Γ be a finite resolved schema environment and O a finite partial map from opaque record identifiers to pairs (T, v) of exact nominal type and complete typed field map. Let A and W be maps of admitted assertions and withdrawals, and let H be a finite acyclic graph of admitted commits. The typed reference environment used by static checking identifies the targets available to a candidate transaction. Its entries do not themselves confer permission to publish the referenced material.

Entity, Process, and Event records identify descriptions whose identities matter to their proposition keys. Two people with the same recorded label, two executions with the same inputs, or two occurrences with the same attributes need not be one referent. Relation, State, and Property records have extensional keys under their complete qualified content. Two separate records can therefore describe the same relation and receive independent, opposing assertions. This is a semantic design choice, requiring evaluation against domain tasks; it is unrelated to the order in which records arrive.

Let $\text{canon}_T(v)$ be an injective encoding of complete typed values under exact type T , invariant under field-map order. Equation 12 defines the proposition key. The type identity includes the schema artifact digest, and $\mathcal{I}$ contains Entity, Process, and Event.

$$\kappa(i, T, v) = \begin{cases} (T, i, \text{canon}_T(v)), & \text{kind}(T) \in \mathcal{I}, \\ (T, \text{canon}_T(v)), & \text{kind}(T) \notin \mathcal{I}. \end{cases} \quad (12)$$

Injectivity is a semantic requirement, not a claim that the current package implements a production signing format. Typed equality distinguishes an integer from a decimal, an ordered list from a set, and an opaque reference from a similarly spelled human label. A declaration-order change can alter predicate argument positions, while field-map source order does not alter a record key. Exact schema identity prevents those two forms of change from being silently conflated.

The extensional key includes every field. Consequently, descriptions with different methods, units, qualifiers, or provenance fields can remain distinct even when a user regards them as statements about one broader topic. A comparison that disregards a qualifier requires an explicit domain mapping or derived predicate. Embedding similarity, linguistic synonymy, or a shared display label cannot silently change equality. Such alignments are themselves interpretive claims with a scope and a responsible author.

6.2 Attributed Assertions and Independent Commitments

An assertion identifies its target record, polarity in $\{+, -\}$, speaker, context, validity interval, evidence references, policy handle, and introducing commit. The record may exist before any assertion targets it. Its availability permits mention and comparison without requiring endorsement. Several assertions can target one record; separate records with equal extensional keys can contribute to the same content-level support calculation.

Negative polarity denies the exact qualified description at its declared context and interval. It does not create a positive assertion of an alternative. A denial of a particular Event record concerns that intensional description; a differently identified Event with changed timing or subject is another description that requires an explicit relation if the domain wishes to connect them. Logical polarity also differs from cryptographic signing: both a positive and a negative assertion can be authenticated.

Attribution must remain separate from submission. An authenticated operator may submit a representation of another person's account under a verified delegation. The account's speaker, its encoder, the evidence custodian, and the submitting principal can then differ. The host validates the relevant authority; the record model preserves the roles needed to examine that validation. A field naming a speaker supplies a claim to inspect, not a self-authenticating act by that speaker.

6.3 Authorized Base Support

Support is evaluated relative to a request q , a historical cut C , a context c , and a valid instant t . The request includes an authenticated principal, purpose, and current policy epoch. Temporal activity is defined

in Section 7. The predicate $\text{readable}_q(a)$ means that the current host policy permits the information necessary to expose assertion a and its content. The predicate $\text{available}_q(a, C)$ means that retained, admissibly obtainable dependencies suffice to evaluate that content and its lifecycle. Equation 13 defines the resulting base sets for sign σ .

$$\begin{aligned} B_{q,C,c,t}^\sigma = \{ & \kappa(\text{rec}(a)) \mid \text{active}_C(a, t) \wedge \text{ctx}(a) = c \\ & \wedge \text{sgn}(a) = \sigma \wedge \text{readable}_q(a) \\ & \wedge \text{available}_q(a, C) \}. \end{aligned} \quad (13)$$

Here $\kappa(\text{rec}(a))$ abbreviates application of Equation 12 to the target record and its typed payload. Implementations retain permitted assertion witnesses alongside each key. Multiple witnesses can support one key, and withdrawal of one witness need not remove the key from the base. A source-filtered service must identify its selection policy; the core does not silently rank speakers and choose a winning assertion.

Restricted evidence can remain under separate custody while an assertion is public, provided that a valid release decision permits the assertion and its exposed content. The result must state its evidentiary scope through permitted metadata. Existence of a readable statement does not imply that a reader can inspect every premise or reproduce the work producing it.

Equation 14 defines the support value of proposition key p . Each coordinate is a Boolean membership indicator.

$$\text{support}_{q,C,c,t}(p) = (\mathbf{1}[p \in B_{q,C,c,t}^+], \mathbf{1}[p \in B_{q,C,c,t}^-]). \quad (14)$$

The values are neither $(0, 0)$, positive-only $(1, 0)$, negative-only $(0, 1)$, and both $(1, 1)$. They describe available support in the selected view. Neither establishes no usable support there; it leaves inaccessible information and the world outside the view unresolved. Both preserves opposition without averaging its sources or selecting one by recency. The information order is componentwise Boolean order. Combining available witnesses by componentwise maximum can increase the information recorded while leaving a substantive disagreement unresolved.

6.4 Domain Conflicts and Higher-Level Findings

Two kinds of conflict require separate treatment. Positive and negative assertions of an exact key produce both. An exclusive-value diagnostic can instead identify two positively supported State or Property descriptions sharing designated key fields but differing in their remaining content. For example, two different assessments of one subject can violate a domain's declared uniqueness expectation without either assessor having explicitly denied the other's full description. The diagnostic creates no negative assertion and does not by itself reject the evidence.

Institutional findings are composed through domain types. A Claim Entity can record a proposition's text, a Relation can describe a body's adoption of that proposition, a Process can represent its review, and Events can record critical stages. Endorsing the existence or wording of the Claim Entity is different from accepting the proposition it describes. Acceptance requires an explicitly defined domain relation or finding. This distinction prevents quotation, archival collection, or discussion from acquiring unintended endorsement.

The support algebra is consequently a resource for inquiry rather than a universal decision procedure. It assigns neither probability nor legal authority. A domain may apply a documented evidentiary standard to the available accounts, but its result must remain distinguishable from the base support that informed it. The same requirement applies to AI synthesis: materializing a generated conclusion requires a new attributed transaction and an appropriate account of its production and adoption.

7 Temporal Scope and Historical Interpretation

Temporal semantics gives correction and reconstruction their precise scope. This section distinguishes validity, occurrence, and admitted history, then defines withdrawal activity and its interaction with current

disclosure. The running example shows how a later record can change the present account of an earlier interval without changing what an earlier historical cut contained.

7.1 Instants, Intervals, and Occurrence Extents

The core uses validated UTC instants under the package’s microsecond profile. A validity interval has a finite inclusive start and an exclusive end that is either a later instant or unbounded. Empty and reversed finite intervals are rejected. The same temporal value can appear in several semantic roles; its position in an assertion, Event, Process, or transaction determines the question it helps answer.

An Event’s occurrence extent can be exact, bounded, or unknown. An unknown extent explicitly records missing temporal knowledge. A bounded Event extent describes occurrence somewhere within the stated interval; it does not assert occurrence throughout that interval. For a Process, the bounded extent denotes its represented span. This category-specific difference is fixed by the core and must survive exchange. Observation time can be supplied as typed metadata when relevant. These fields do not reorder admitted commands.

The transaction’s source-level recorded value is useful for offline fixtures. A live accepting service supplies the authoritative local receipt time and reports any adjustment before final authentication. A contributor’s backdated field cannot establish global admission order or conceptual priority. Chronology of real-world occurrences remains an evidentiary question that can use the admitted record without being determined by its storage clock.

7.2 Causal Cuts and Historical Questions

A historical cut is a parent-closed set of admitted commits. For a source query referring to commit h , Equation 15 gives its selected cut. The ancestry is taken in the admitted graph H and includes every parent dependency of h .

$$C(h) = \{h\} \cup \text{Ancestors}_H(h). \quad (15)$$

The cut can contain opposing assertions and concurrent decisions. It names recorded history independently of the valid instant used to inspect that history. A service may map a local receipt sequence to a stored cut, but a wall-clock timestamp alone cannot identify one global view across autonomous nodes. Several tips can be integrated through an admitted empty merge transaction after their dependencies have been validated.

The distinction supports two ordinary forms of inquiry. A reader can ask what the institution had recorded when it acted, using the earlier cut. The reader can also ask what the available present record says about that earlier period, using a later cut with the earlier valid instant. Those answers may differ without either query rewriting the selected history. Their difference can be important to responsibility, but the language does not infer the governing standard of responsibility from it.

7.3 Withdrawal Masks and Temporal Activity

A withdrawal targets one assertion identifier, an interval, and a reason. Its authority is checked at admission. Let $I(a)$ be assertion a ’s validity interval, $J(w)$ a withdrawal’s interval, and intro the introducing commit. Equation 16 defines activity before disclosure filtering.

$$\begin{aligned} \text{active}_C(a, t) \iff & \text{intro}(a) \in C \wedge t \in I(a) \\ & \wedge \neg \exists w \in W [\text{intro}(w) \in C \wedge \text{target}(w) = a \wedge t \in J(w)]. \end{aligned} \quad (16)$$

The withdrawal affects the intersection of the two intervals. It leaves other speakers’ assertions untouched, even when their content keys coincide. Reinstatement creates a fresh assertion; withdrawal of a withdrawal is outside the core. The set of recorded withdrawals grows monotonically under compatible history union, while active support can decrease. Consequently, adding more history does not imply a monotone increase in every query answer.

Table 2: Historical cut and valid-time coordinates in a retrospective correction

Cut	Valid instant	Support	Interpretation
C_0	July	Positive-only	Alice’s assertion is the available commitment.
C_1	July	Both	Bob’s denial joins Alice’s support.
C_2	July	Positive-only	The admitted withdrawal masks Bob’s denial for July.
C_2	June	Both	The withdrawal leaves the earlier interval unchanged.
C_1	July, queried later	Both	An earlier cut retains its interpretation under the stated availability and disclosure assumptions.

Withdrawal masks are applied before authorized support is exposed. Restricting the reason for a withdrawal cannot reactivate the targeted assertion. If lifecycle dependencies are unavailable, the service must report the affected support as unavailable or return a safely qualified partial result. Assuming continued activity would make protection of the withdrawal reason defeat the correction itself. Conversely, the service cannot reveal a protected reason or identifier merely to explain its internal mask.

7.4 Retrospective Correction Trace

At cut C_0 , Alice positively supports a relation. At its descendant C_1 , Bob adds negative support for the same qualified relation, applying throughout June and July. At C_2 , Bob’s August withdrawal removes his denial from 1 July onward. Table 2 gives the resulting support under one unchanged test disclosure profile with all necessary dependencies retained. Dates here label the synthetic fixture, and no actual repository action is reported.

The trace distinguishes a speaker’s changed commitment from a change in the represented occurrence. It also distinguishes an interval-limited correction from complete renunciation of an assertion. A user interface that presents only a current unqualified label would conceal those differences even if the backend retained them correctly. The result contract must carry the coordinates necessary to interpret the displayed status.

7.5 Current Disclosure and Conditional Reconstruction

Historical reconstruction uses retained schemas, record payloads, lifecycle data, and admitted receipts. Present access remains governed by the current request policy. A past disclosure decision can explain an earlier action; it does not automatically authorize disclosure today. Erasure, sealing, or loss of a dependency may make an earlier cut partly or wholly unreconstructable for a particular verifier.

The architecture therefore separates a formal cut from its currently available projection. An identifier or digest of deleted content cannot restore that content. A restricted view can legitimately provide less detail than the historical institution once held, and even the fact of restriction may require protection. The result’s completeness descriptor concerns the authorized supplied view, with limitations expressed at a permitted level. Conditional reconstruction is compatible with justified retirement; an unqualified promise of perpetual replay would be inconsistent with the language’s governance boundary.

8 Authorized Transitions and Transaction Semantics

Transaction semantics connects well-typed descriptions to controlled changes in an admitted repository. This section specifies candidate preparation, authority, atomicity, and workflow commands. The account uses a pure staged evaluation relation followed by an admission rule. Its purpose is to make the difference between representing an occurrence and exercising a power observable in every successful or rejected operation.

8.1 Candidate Graphs and Trusted Preconditions

A candidate transaction identifies its commit identity, complete parent list, submitter, requested receipt time, policy handle, and ordered operations. The service resolves its exact schemas and registers all candidate record declarations before evaluating the remaining operations. This two-phase process permits forward and cyclic domain references within an atomic graph. It leaves temporal and command dependencies subject to their own ordering rules. A withdrawal, for example, must target an ancestor assertion or an earlier assertion operation in the transaction.

Let C be the pre-cut used for evaluation. Its relation to the supplied parents is exact, as specified in Equation 17.

$$C = \bigcup_{p \in \text{parents}(tx)} C(p). \quad (17)$$

An empty parent set selects the empty cut for a genesis transaction. Redundant ancestor entries in a parent list are harmless. Additional history that the evaluator did not inspect cannot be attached later as an unevaluated parent. Without this equality, a transaction could claim a causal context different from the context in which its preconditions were checked.

Authentication and authorization use the trusted control-plane pre-state and current identified policy. Candidate records can describe roles, mandates, permissions, or credentials, but cannot install their own authority. A submitter naming another speaker requires independently verified delegation for the relevant act. Bootstrap policy, trust anchors, and administrative changes require a separately governed host protocol. The source language describes the requested operation and its context; it cannot choose a more permissive control plane by asserting that one applies.

8.2 Staged Evaluation and Atomic Admission

Candidate evaluation produces provisional record, assertion, withdrawal, and receipt deltas. Earlier operations can affect the staged state seen by later operations. A transaction can therefore start a run and advance the token it has just created, provided the complete transaction passes validation. Authority remains grounded in the trusted pre-state throughout this staging. An operation's newly recorded permission descriptor cannot authorize a later operation in the same transaction merely because it is now locally readable.

Let $\text{Eval}(C, tx) = (\delta, \rho)$ denote successful pure staged evaluation, and let Inv include immutable identity, stable binding, and the declared workflow invariants. Equation 18 states the admission contract. Parent closure from Equation 17 is included in TypedDep alongside typing and dependency validation.

$$\frac{\text{TypedDep}(\Gamma, C, tx) \quad \text{Auth}_{\Pi_{\text{now}}}(C, tx) = \text{permit} \quad \text{Eval}(C, tx) = (\delta, \rho) \quad \text{Inv}(C \oplus \delta)}{(H, C) \xrightarrow{tx} (H \cup \{\text{commit}(tx, \delta, \rho)\}, C \cup \{tx.id\})}. \quad (18)$$

The notation $C \oplus \delta$ refers to the candidate state projected from the cut and staged deltas; it is not an additional committed history. If a premise fails, no candidate domain effect is admitted. A protected rejection audit may record the failed request under a separate operational policy. Such an audit does not convert its allegations or proposed assertions into accepted domain data.

Receipts bind the parent frontier, schema and machine identities, policy epoch, command identities, and accepted effects. An authorization dependency on external state must be identified through a versioned decision receipt. Historical replay consumes that receipt; it does not recreate the external state or silently reevaluate the old decision under a new policy. The admission rule is a semantic contract for an implementation, not a report that all of these checks already run in the supplied prototype.

8.3 Run Binding and Operational Tokens

A workflow run is a Process record bound to one machine and one subject. The machine tracks a State schema with exactly two fields: a subject reference and an enum value. General attributed State records

can describe more elaborate configurations through other schemas. Their support cannot rewrite operational tokens.

Each accepted Start, Advance, or Resolve produces a token whose identifier is its command identifier. The token carries its run, machine, subject, value, and causal receipt. A receipt also identifies consumed tokens. The current projection consists of produced tokens that no admitted receipt in the selected history consumes. Several tokens constitute an operational conflict even when their enum values coincide. Concurrent actions with equal labels can have different grounds and consequences.

For staged state S_k , let $\text{bind}(S_k, r)$ be the partial run binding and $\max(S_k, r)$ its maximal token set. Candidate record registration supplies the typed run, subject, and Event records. Equation 19 specifies Start after those typing conditions. The Event must have a compatible subject reference, but Start imposes no additional nominal event type beyond that condition.

$$\frac{\text{bind}(S_k, r) \text{ undefined} \quad \text{subject}(e) = s \quad \text{initial}(m) = x}{\text{start}(k, r, m, s, e) : S_k \mapsto S_k[\text{bind}(r) := (m, s), k : x]}. \quad (19)$$

The new binding and token remain provisional until Equation 18 succeeds. If the run is already bound in the pre-cut or by an earlier operation, Start fails. Reusing a Process identity for an incompatible machine or subject cannot serve as an implicit migration.

8.4 Ordinary Advance and Conflict Repair

An ordinary Advance names a machine step, an Event, and the exact expected singleton token. The Event must have the step's exact nominal type and refer to the run's bound subject. Equation 20 specifies the token effect for step (x, E, y) under machine m . The enclosing admission rule supplies authority, reference, uniqueness, and invariant conditions.

$$\frac{\text{bind}(S_k, r) = (m, s) \quad \max(S_k, r) = \{u\} \quad \text{expect}(a) = \{u\} \quad \text{value}(u) = x \quad \text{step}_m(a.\text{name}) = (x, E, y) \quad \text{type}(a.\text{event}) = E \quad \text{subject}(a.\text{event}) = s}{\text{advance}(a, r) : \{u : x\} \mapsto \{a.\text{id} : y\}}. \quad (20)$$

A correctly typed Event concerning another subject fails this rule. Merely recording the Event does not attempt the transition. An expected singleton that omits another maximal token also fails; a client cannot ignore concurrency by choosing the branch it prefers. These failures preserve the distinction between an event's descriptive availability and an authorized operation over a particular run.

Resolve is a separate repair command with independent authority. It requires an existing stable binding, at least two maximal tokens, their exact complete set, a valid target enum value, and a reason record. Equation 21 gives its token effect, with U a duplicate-free set and V_m the machine's enum values.

$$\frac{\text{bind}(S_k, r) = (m, s) \quad U = \max(S_k, r) \quad |U| \geq 2 \quad y \in V_m}{\text{resolve}(k, r, U, y) : U \mapsto \{k : y\}}. \quad (21)$$

The repair preserves every consumed token in its receipt. It establishes the run's subsequent operative state, while the source assertions and reasons remain subject to their own lifecycle. Resolve is unavailable for a singleton or unstarted run and supplies no general administrative reset. Its authority to choose a valid enum value does not turn the selection into an epistemic truth judgment.

8.5 Idempotency, Effects, and Implementation Boundaries

Repeating identical admitted bytes under one commit identity returns the original receipt. Reusing the identity with different content is an integrity error. Command identities are unique, so the same receipt graph cannot create new token identities merely through repeated delivery. A new command identifier for a repeated business request requires a separate domain idempotency policy; the language does not infer that two differently identified requests concern one real-world act.

The core invokes no external actions, sends no notifications, and executes no arbitrary model or user code. A future adapter can distinguish committed intent, attempted delivery, acknowledged action, and verified outcome, with its own failure and compensation semantics. These are additional contracts. Replaying admitted Event records or command receipts never redispaches the external effects that a richer host may have associated with the original act.

The same boundary applies to epistemic guards. A future language extension could let a transition depend on a claim's support under a named evidentiary policy. It would need to define both and neither, identify read dependencies, revalidate relevant merged invariants, and address confidentiality. GRK-DSL 0.1 supplies no such guard. Its controlled token state and trusted authority conditions remain sufficient to explain the operations actually specified.

9 Branching, Merge, and Provenance

Branching permits different contributors or institutions to develop histories without imposing one sequence of interpretation. This section specifies the conditions for combining admitted histories and distinguishes preserved disagreement from operational conflict. It then examines what provenance and replay can establish under those conditions. The analysis concerns compatible records and declared admission profiles; transport convergence alone cannot settle their institutional recognition.

9.1 Compatible History Union

A branch name is a mutable reference to a commit tip. It is a navigation device, not an evidentiary ranking. A branch can contain a correction, an alternative interpretation, or a different institutional process. Its current name cannot determine which assertions are true or which decisions another institution must adopt.

Two cuts can be combined when repeated immutable identities agree in type and payload, schema dependencies are available and compatible, receipts satisfy the selected verification profile, and run bindings remain stable. Equation 22 defines the combined input cut. A subsequently admitted merge commit additionally enters its own selected source-level cut.

$$C_{\cup} = C_1 \cup C_2. \quad (22)$$

Set union retains assertions, withdrawals, and receipts from both histories. It is associative, commutative, and idempotent on compatible inputs. These algebraic properties concern the admitted payloads. Nodes using different admission policies may select different inputs, and their available public views may differ under disclosure rules. A convergence claim must specify which admission and interpretation assumptions are held equal.

Opposing assertions survive the union. Support is recomputed over the merged cut, including all relevant withdrawals. A merge that adds a withdrawal can reduce active support even while increasing the retained history. Selecting the latest timestamp would lose this distinction and would impose an unsupported hierarchy on epistemic disagreement.

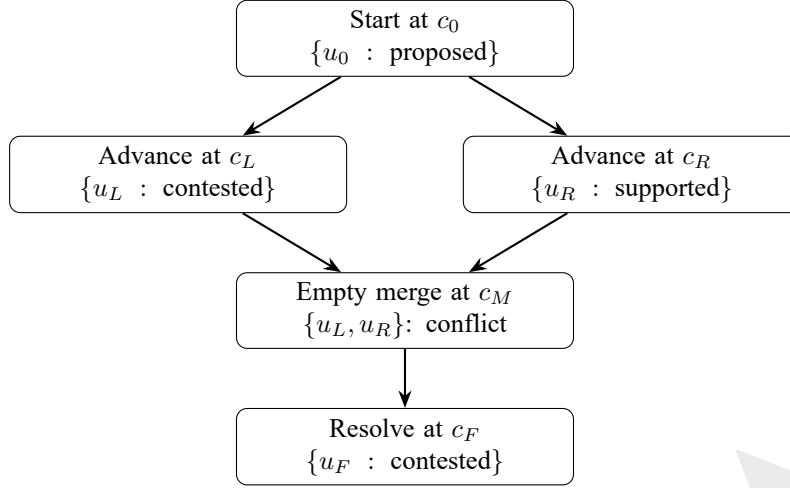
9.2 Concurrent Transitions and Stable Bindings

Let $\text{Tokens}(C, r)$ be the tokens produced for run r in cut C , and let the consumption relation be read from validated receipts. Equation 23 defines the operative projection.

$$\text{max}(C, r) = \{u \in \text{Tokens}(C, r) \mid \nexists v \in \text{Tokens}(C, r) : v \text{ consumes } u\}. \quad (23)$$

Suppose Start produces token u_0 with value proposed. Two transactions based on its cut independently consume u_0 , producing u_L at contested and u_R at supported. Each local transition can satisfy its singleton precondition. The merged projection contains $\{u_L, u_R\}$ because neither output consumes the

Figure 1: Concurrent workflow histories, merge, and explicit token repair



other. Ordinary Advance is then unavailable. A properly authorized Resolve naming both outputs can produce one new token while preserving the complete contested history.

The outcome is unchanged if both branches choose the same label: two maximal receipts still exist. Label equality cannot establish equivalence of the decisions that produced them. Concurrent Starts with the same machine and subject similarly yield compatible binding information and conflicting tokens. Starts assigning different bindings to one Process identity create an integrity conflict. Such an import is quarantined; repair requires a fresh Process identity and explicit provenance rather than mutation of the old binding.

Figure 1 shows the admitted commit structure of this constructed trace. The merge integrates history and exposes the conflict; the subsequent repair supplies a new operative token. The arrows identify parent dependencies, and the displayed token sets summarize the relevant run projection at each cut.

9.3 Global Constraints and Coordination Profiles

The core guarantees are limited to immutable identity, typing, and its stated per-run workflow conditions. General resource capacities, obligations across several runs, and mutually exclusive institutional powers require additional coordination. Two transactions can write disjoint records while both relying on the last unit of a shared resource. Checking disjoint write identifiers would fail to detect that shared read assumption.

A stronger transaction profile must specify how it records read dependencies, checks complete merged invariants, or coordinates admission before a conflict arises. That choice can affect availability, latency, and local autonomy. The proposal therefore makes no claim to unrestricted transactional convergence. Preserving a conflict is sometimes an honest intermediate result; preventing or resolving it requires the contract appropriate to the resource and institution involved.

Unverifiable imports can remain available as protected transport artifacts and become the subject of new assertions. They do not silently enter the admitted history. An institution can record that another node asserted a finding without adopting that finding or accepting its command receipts as locally operative. The exchange profile must identify the distinction.

9.4 Genealogy, Migration, and Reconstruction

Provenance connects several different objects. A source reference documents where a record was obtained; an assertion identifies a speaker's commitment; a transformation describes an operation over inputs; a review receipt documents an institutional act. Claims of conceptual influence, independent development, or historical priority require further attributed relations. A commit path supplies evidence

of recorded dependence under its trust assumptions. It cannot reconstruct an unrecorded conversation or establish complete intellectual ancestry.

Schema migration creates new records or views through a declared mapping. Its inputs, outputs, schema identities, author, and material losses should remain available at an appropriate disclosure level. Reinterpreting a field without changing the schema artifact would make old and new data appear comparable when their meanings differed. Equivalence assertions between versions are themselves scoped claims, and a failed or partial mapping should not be presented as lossless exchange.

Replay reconstructs immutable maps and token graphs from accepted deltas and receipts. It does not rerun historical authorization, nondeterministic models, or external services. A later withdrawal can remove current support for the account that motivated an earlier action without undoing the action's receipt. Reversal requires a new authorized operation under an appropriate profile, together with an account of consequences outside the repository.

Reconstruction also depends on retention. A remaining digest can identify that a payload was once committed without making its bytes recoverable. Restricted provenance can support a protected audit while leaving public readers with a qualified account. The language preserves distinctions among record identity, content equality, derivation, and institutional consequence; the host must preserve equally clear distinctions among complete, partial, and unavailable reconstructions of the selected view.

10 Query and Rule Semantics

Query semantics specifies what a program can derive from one authorized historical view. This section translates base support into signed relations, defines finite least-fixed-point evaluation, and distinguishes the four query operations. It also specifies grounded explanations and the consequences of changed inputs. The method separates the mathematical result from the host's additional obligations concerning disclosure, caching, and observability.

10.1 Signed Relations and the Active Domain

Each domain type exposes base predicates according to its category. Entity, Process, and Event predicates include a leading identity reference followed by fields in schema declaration order. Relation, State, and Property predicates use the fields alone. Each sign becomes a separate ordinary relation, such as R^+ and R^- . Let E denote the signed ground atoms obtained from the visible active base support defined in Equation 13. This translation changes representation, not the scope of the evidence or its authority.

A negative-sign atom in a body tests membership in the negative relation. It is therefore a positive relational test in the evaluation algorithm. Its variables are range-bound by the stored tuples just as variables in a positive-sign atom are. No test asks whether an atom failed to appear. Negative conclusions can arise from explicit negative support or declared rules with a negative-sign head; absence creates neither.

Let D be the finite active domain consisting of visible base values and program constants under typed canonical equality. Finite structured values, including lists and sets appearing as values, are treated as indivisible constants by the rule evaluator. Variables cannot occur inside their literal constructors. The full language admits arbitrarily large integers and texts, but one finite query instance supplies only finitely many values. The rule core constructs no new values, existential witnesses, or function terms during inference.

Rule heads introduce module-local derived predicates and cannot overwrite base record predicates. Arity and argument types must agree across definitions, and each head variable must be bound in the nonempty body. Recursion requires an inferable type signature anchored in typed predicates or constants. These restrictions separate the intended finite evaluation from extensions whose safety would require additional analysis.

10.2 Least-Fixed-Point Interpretation

For a fixed well-typed program P , let HB be the finite set of possible well-typed signed ground atoms over the base and derived predicates exposed by Γ and P , using values in D . A substitution θ satisfies a body when every instantiated atom belongs to the current set X . Equation 24 defines the immediate consequence operator, retaining the extensional input E .

$$T_{P,E}(X) = E \cup X \cup \{\theta(h) \mid (h \leftarrow b_1, \dots, b_m) \in P, \forall j \in \{1, \dots, m\} : \theta(b_j) \in X\}. \quad (24)$$

Equation 25 gives the denotation through iteration from the authorized extensional base.

$$\llbracket P \rrbracket_E = \bigcup_{n \geq 0} T_{P,E}^n(E). \quad (25)$$

Every strict iteration adds an atom from finite HB and removes none during that evaluation. The fixed point is therefore reached after finitely many strict iterations. This result concerns one fixed input and program. A new withdrawal, policy epoch, schema mapping, or program version defines a new evaluation problem; there is no requirement that its output contain the old output.

Recursive genealogy illustrates the interpretation. An attributed link can seed a Reach predicate, and a recursive rule can extend Reach through further links. A reported pair then has a grounded derivation from the available links. It does not create a new source assertion that the relationship is historically true in an unrestricted sense. A human or institutional adopter who wishes to publish that interpretation creates a separate attributed record with its grounds and qualifications.

10.3 Query Operations and Result Coordinates

The `select` operation returns distinct selected tuples whose conjunctive body holds in $\llbracket P \rrbracket_E$. Selected variables must occur in the body. Order is unspecified; a presentation profile can sort the result without changing its set denotation. Different derivations of one tuple do not produce duplicate answer rows, although permitted explanation data can distinguish their witnesses.

The `support` operation calculates the two base-support bits for the exact key of a referenced record. It does not include virtual predicates derived by rules. Derived signs are queried through `select`; materialization into an attributed record is an explicit new transaction. If the referenced payload needed to identify its key is unavailable or protected, the host cannot fabricate a content-level answer from an opaque identifier alone.

The `history` operation returns permitted introductions, withdrawals, and direct references concerning the requested record within the cut. It includes inactive assertions and their full applicable intervals and contexts. The uniform source envelope's valid instant and context do not filter this operation. Direct record history also differs from history of every extensional copy: a second Relation record can have the same content key while retaining its own identifier-specific history. A broader proposition-history or recursive genealogy service needs an explicit contract.

The `current` operation projects workflow tokens at the cut and returns permitted token witnesses or an appropriately protected conflict description. Its source envelope likewise contains syntactically required valid-time and context fields that do not filter operational state. An implementation must identify inapplicable parameters in the result metadata. Uniform syntax is a convenience for requests; it cannot imply a uniform temporal meaning that the operations do not possess.

Table 3 summarizes the semantic differences. Every operation is evaluated under current authenticated request policy, including operations over an earlier cut.

10.4 Grounded Explanations and Changed Inputs

A proof explanation can use a finite graph of ground rule instances and their premises. Reporting a conclusion requires a grounded witness derivation from the selected base. Recursive programs can admit

Table 3: Query operations and semantic coordinates

Operation	Object of evaluation	Temporal and interpretive scope
select	Conjunction over base and virtual signed predicates	Cut, valid instant, context, and identified rules.
support	Base support for an exact proposition key	Cut, valid instant, and context; virtual predicates remain separate.
history	Direct history of a record identifier	Cut; reported intervals and contexts are retained without envelope filtering.
current	Operational token projection of a Process run	Cut; envelope validity and context are inapplicable.

infinitely many expanded proof trees for the same answer, so an implementation must avoid promising to enumerate every proof tree. A cyclic dependency graph without an extensional grounding cannot by itself establish an answer under Equation 25.

After a premise is withdrawn or becomes inaccessible, the engine recomputes from the new base or uses an incremental algorithm shown equivalent to that recomputation. Consider two derived atoms that support each other through recursive rules after their only external seed disappears. Retaining both because each still names the other would select an unsupported fixed point. The least-fixed-point construction begins from the new base and removes the unfounded cycle. Correct incremental maintenance therefore requires more than reference counts over currently stored derivations.

Explanations have their own disclosure requirements. Rule identities, witness identifiers, counts of alternative derivations, and the shape of a dependency graph can disclose information beyond the visible conclusion. An explicitly authorized public attestation can provide a bounded explanation of a restricted computation, but it must be a separately permitted object. A query cannot make a confidential premise public by presenting it as provenance.

10.5 Completeness, Cost, and Observational Limits

Each result identifies its actual cut, schema and rule identities, applicable context and valid instant, current policy epoch, and selected profile. Completeness descriptors distinguish a scoped evaluation of the supplied authorized view from published gaps or unavailability. They concern the recorded view, not the world. A descriptor whose variation would reveal protected existence must be generalized consistently with the disclosure profile.

Finite termination supplies no general latency guarantee. The number of possible groundings can grow rapidly with arity and the active domain, and large joins or explanation graphs can dominate cost. Implementations can advertise resource limits and safe interruption behavior while preserving the difference between a partial evaluation and a completed answer. Formal termination, practical performance, and information-flow protection therefore remain distinct evaluation dimensions.

11 Governance and Exchange Interfaces

The language depends on host institutions and services for authority, disclosure, retention, and exchange. This section specifies those interfaces and the limits of their formal treatment. The analysis separates operations available in source syntax from the broader governance vocabulary required by an infrastructure. It then identifies the conditions under which an exchange packet can become admitted, interpretable history.

11.1 Authorization Decisions and Policy Installation

A host authorization request includes an authenticated principal, operation, targets, purpose, context, causal pre-cut, current policy epoch, relevant labels, and credential references. An asserted jurisdiction

or community mandate makes an assessment addressable; the field does not establish that the assertion is legally or institutionally correct. Trusted credentials and delegations are resolved independently from untrusted descriptive records.

The decision interface distinguishes permit, deny, indeterminate, and not-applicable. Only an effective permit allows a protected operation. The core combining profile gives a conflicting applicable denial precedence over a permit. Missing mandatory attributes or unavailable required policy yield indeterminacy; all-not-applicable supplies no authorization. Other combining algorithms require explicitly named profiles. These four administrative outcomes are separate from the four epistemic support states.

A decision receipt identifies the policy artifact and version, principal binding, operation, targets, purpose, cut, epoch, conditions, obligations, reason code, expiry, and permitted review route. Required preconditions must be discharged before the operation proceeds. Continuing obligations require an accountable monitoring process, deadlines, and evidence of performance. Recording an obligation as a Process does not establish that it has been fulfilled.

The infrastructure may represent policy using ordinary domain records, including Policy entities, permission relations, review states, and decision events. Their storage supports explanation and challenge. Installation into the effective control plane remains a separately authorized administrative act. This separation prevents a circular rule in which a recorded claim of power supplies the authority needed to make itself operative.

11.2 Operation Scope and Derived Disclosure

The governance vocabulary distinguishes record creation, assertion, withdrawal, derivation, workflow commands, reading, querying, provenance disclosure, export, publication, sealing, erasure, and declassification. Only the six specified transaction forms are source-level core operations. Other vocabulary items describe host interfaces or institutional procedures; their names should not be mistaken for undocumented GRK commands.

Authority is operation-specific. Permission to submit a record can coexist with refusal to publish its evidence. Permission to derive a local result can coexist with a restriction on export. An affected person may have a challenge route without direct editing privileges. These combinations require express host decisions, with the reasons and responsibilities needed for review. The language cannot determine a person's legal entitlement from their position in a provenance graph.

The transaction policy supplies default protection for raw record definitions, command metadata, and receipt information. An assertion policy controls the assertion and the content exposed through it. These allocations remain distinct when a publicly released assertion refers to a locally retained record; the release must cover the content actually disclosed.

Confidentiality and integrity are independent. Publicly readable material can have weak evidentiary or authentication grounds; an authoritative instruction can remain confidential. A public assertion referring to a restricted payload requires a valid release decision for the content it exposes. Mentioning an identifier cannot lower the payload's protection.

For an elementary reader-set profile, derived disclosure is limited to an audience compatible with all required inputs and rules. More expressive constraints involving purpose, time, or legal relations need a declared combination semantics. A public aggregate or attestation is an independently authorized released object. Removing a label from a confidential computation is not an adequate declassification procedure. The core supplies no claim of differential privacy or protection against repeated-query inference.

11.3 Exchange Representation and Admission Stages

The companion exchange schema uses a tagged JSON representation of values, records, assertions, withdrawals, commands, and receipts. Tagged integers and decimals avoid relying on a generic JSON number interpretation for exact typed equality. A schema identity binds the module namespace, artifact digest, and local name. A production profile must specify canonical bytes, digest algorithms, authentication,

and interpretation versions consistently. The current structural fixture uses placeholder identities and is unsigned.

Structural schema validation checks the envelope’s shape. Further stages resolve schemas and references, validate typed values and intervals, compare immutable identities, authenticate receipts, establish admission authority, and verify workflow dependencies. Passing an earlier stage does not imply passing the later stages. The distinction is important for imported records that are safe to preserve as evidence of another node’s assertion but are unqualified for local operational admission.

Exchange manifests must themselves satisfy disclosure constraints. Raw sources, protected record identifiers, or commit ancestry can reveal relationships that an approved public summary was intended to conceal. A permissible export may therefore use a newly issued release handle and an independently public attestation. Required dependencies must have permitted representations or the receiver must accept a declared limitation. There is no general rule that every source dependency should be exported.

11.4 Interoperability and Conformance Layers

Adapters can use established graph, provenance, query, policy-expression, and repository systems while preserving GRK’s interpretation profile. Mappings must distinguish source identity, semantic equality, attributed support, and institutional recognition. A graph name or repository timestamp cannot silently serve as an assertion of truth or priority. Schema translation should preserve original identifiers and declare losses or changed assumptions.

Conformance is layered: S covers syntax recognition; T resolution and typing; K attributed temporal knowledge and queries; W workflow admission and replay; G governance and disclosure enforcement; X exchange profiles; and F federation and retention services. A tool can implement one layer without implementing the others. The available recognizer and structural schema checks occupy bounded parts of this stack, while a public service requires additional institutional and operational capacity.

The host boundary is consequently an explicit part of the language’s contract. It identifies where a formal result ends and where an institution must supply a justified, reviewable decision. This paper states the institutional assumptions needed to interpret the language, including the possibility of refusal, restricted disclosure, and justified loss of historical completeness.

12 Worked Encodings and Conformance Evaluation

Evaluation must distinguish a language specification from an implementation that conforms to it and from a representation that people can use well. This section develops the supplied encodings as analytical tests, records the checks actually performed, and defines the comparative work still required. The examples exercise consequential distinctions; they are constructed fixtures with hypothetical authority and disclosure conditions.

12.1 Contested Knowledge and Record Genealogy

The first complete source fixture declares all six domain categories, a Property check, an exclusive State diagnostic, and recursive genealogy rules. Alice supports a relation from a dataset to a claim. Bob denies another Relation record with identical complete fields written in another source-field order. Equation 12 makes their extensional keys equal, while their assertion identities remain distinct.

The expected results follow Table 2: positive-only before Bob’s denial, both at the contested cut, and positive-only for July after his interval-scoped withdrawal. June remains both at the later cut. These outcomes test the composition of equality, attribution, history, and valid time. A language lacking any one of those distinctions would need an additional convention to recover the same interpretation.

The fixture’s genealogy rules derive a Reach relation from the dataset to the original claim and then to a revised claim. These are virtual answers with grounded derivations; no new Claim entity or assertion is created. Its Opposed predicate ceases for the July view when the relevant negative support is withdrawn.

Two positively asserted Assessment values separately produce an exclusive-value diagnostic. Neither a diagnostic nor a derived answer silently creates the opposite sign of a base record.

Record history supplies a further test. The history of the original support record includes its permitted direct references. The extensional copy has the same content key but a different identifier-specific history. A service that silently merges those histories would change the meaning of the operation. A broader content-level genealogy can be useful, but it must identify the alignment it performs and preserve the source distinctions.

12.2 Workflow Concurrency and Subject Integrity

The second fixture binds a ReviewRun Process to a ReviewFlow machine. Start produces a proposed token. Two independently admitted branches consume that token and produce contested and supported outputs. Their merge has two maximal tokens; an explicit Resolve consumes both and records one subsequent operative state. Appendix B reproduces the complete source fixture, including its declarations and query forms.

Several rejected variants sharpen the contract. An ordinary Advance from the merged cut has no singleton input. A Resolve naming only one maximal token omits part of the conflict. A Resolve applied to a singleton is an unsupported reset. An Event of the correct nominal type but concerning a different subject fails the subject-equality condition. These are semantic acceptance targets even when the corresponding surface text is grammatically well formed.

The example also distinguishes local admissibility from merged coherence. Both branches can have valid receipts under their shared ancestor while requiring conflict repair after integration. Replay retains those receipts without executing their Events again. Equal receipt sets have equal token projections under the assumptions proved in Section 13; the result says nothing about the institutional merits of either branch.

12.3 Local Computation and Public Release

The third fixture represents a local dataset, transformation, and intended public Count assertion. Under an explicitly assumed release grant, its PublishedCount query returns the summary identifier and the value 40. The raw dataset, input identifiers, and protected ancestry remain local unless separately authorized. A permitted explanation can use an independently public attestation.

This is a constructed operator view rather than a safe public export of the whole source file. A manifest needs permissible representations for speaker, context, summary, and other exposed dependencies. If dependency existence is sensitive, a fresh release handle and a separately authorized attestation may be necessary. Without the required grant, intended publication remains restricted or denied. A readable descriptor of the intended policy cannot grant itself authority.

The example checks a representational separation, not the effectiveness of an implemented privacy mechanism. No differential privacy, legal competence, cryptographic authentication, or protection against inference is demonstrated by the fixture. Those claims would require separate profiles and evidence. The distinction is important because even a seemingly harmless count can interact with context or repeated queries in ways absent from a syntax test.

12.4 Performed Mechanical Checks

The EBNF-driven recognizer passed 25 self-tests and recognized all three complete source examples. It lowers EBNF constructions into grammar productions and performs Earley recognition. Its tests cover selected accepted and rejected lexical and syntactic forms, including recursive and nullable grammar behavior. Recognition establishes membership under that tool's interpretation of the grammar; it does not produce a fully elaborated typed executable program.

Fifteen bounded semantic oracle checks passed. They cover selected proposition keys, schema and context separation, historical cuts, interval boundaries, withdrawal locality, hidden withdrawal reasons,

Table 4: Available artifact checks and unestablished capabilities

Artifact	Performed check	Remaining capability or claim
Concrete grammar	25 recognizer tests and three accepted examples	Full parser, elaboration, and ambiguity analysis.
Semantic models	15 bounded oracle tests	Source interpreter and comprehensive conformance.
Exchange envelope	Metaschema, one positive and six negative fixtures	Typed resolution, authenticated admission, and signing profile.
Mathematical definitions	Scoped propositions and proofs	Mechanized metatheory and implementation correctness.
Governance interfaces	Specified request and receipt contracts	Effective policy enforcement, privacy analysis, and institutional evaluation.

grounded recursion, non-explosion, token projection, and resolution. Small receipt sets are tested under permutations. The oracle uses an integer timeline, simplified values, and finite ground-rule instances. It neither parses GRK source nor demonstrates the correctness of a compiler between source and semantic structures.

The exchange schema passed the JSON Schema 2020-12 metaschema check. One structural fixture validated and six invalid variants were rejected. The fixture is unsigned and contains placeholder schema identities. Its result concerns envelope shape, not actual schema resolution, typed payloads, authentication, authority, or admissible history. These checks were rerun during preparation of this manuscript; the artifact report records their bounded scope.

Table 4 separates the available evidence from stronger claims that remain open. Treating the layers separately makes an incremental implementation assessable without inferring capabilities from a successful earlier stage.

12.5 Comparative Adequacy and Implementation Strategy

A reference implementation should be compared with a relational assertion store, an RDF/PROV/SHA-CL/SPARQL composition with a ledger, and a typed Datalog encoding. Each baseline should receive equivalent records, policies, retention assumptions, and competency questions. Necessary extensions and operator labor should be documented. Existing tools may supply the required behavior more economically; such a result would narrow the rationale for a dedicated language without negating the value of explicit contracts.

Domain evaluation should ask participants to encode consequential distinctions and then interpret the resulting records. Tasks can include representing testimony without endorsing its content, distinguishing observation from a review finding, preserving a disputed translation, and reconstructing a qualified synthesis. Review should examine where the six-category basis requires awkward encodings or suppresses a domain’s meaningful concepts. Schema validity alone cannot establish representational adequacy.

Implementation work should proceed through parsing and elaboration, a complete type checker, local temporal support, finite inference, workflow admission and replay, policy enforcement, canonical exchange, and federation. Each stage needs positive and negative tests against its actual contract. Performance measurements should report active-domain size, arities, join structure, policy overhead, retained history, and hardware. Participant research requires an appropriate protocol before recruitment. No empirical result for usability, performance, or public value is reported here.

13 Formal Properties, Proof Obligations, and Limits

The formal contract supports several mathematical results under explicit assumptions. This section proves finite inference, bounded non-explosion, withdrawal locality, and deterministic projection of

admitted receipts. It then separates those results from typing, security, and adequacy obligations that require further work. The proofs concern the definitions in this paper; they provide no claim of mechanized verification or correctness of an unimplemented runtime.

13.1 Finite Inference and Grounded Consequence

The finite rule core has the termination property stated in Proposition 13.1. Signed predicates are treated as separate ordinary relations, and all relevant constants belong to the finite active domain.

Proposition 13.1 (Finite least-fixed-point evaluation). *For a fixed finite, well-typed program P satisfying the core's range and value-generation restrictions, iteration of $T_{P,E}$ from a finite base E reaches its least fixed point after at most $|HB \setminus E|$ strict iterations.*

Proof. The set of predicates is finite, each arity is finite, and the active domain is finite. Hence HB is finite. If $X \subseteq Y$, every ground body satisfied in X is satisfied in Y , so $T_{P,E}$ is monotone. It is also inflationary. Every strict iteration adds at least one previously absent member of HB , and no iteration removes a member. At most $|HB \setminus E|$ strict iterations are possible. The first unchanged set is a fixed point. For any fixed point containing E , induction on iteration shows that every constructed set is a subset of it. The reached fixed point is therefore least among those containing the base. $\square$

The result bounds strict stages rather than execution time. If $\mathcal{R}$ contains the unsigned predicate signatures, a coarse grounding bound is given in Equation 26; typing can reduce the actual set.

$$|HB| \leq \sum_{R \in \mathcal{R}} 2^{|D|^{\text{arity}(R)}}. \quad (26)$$

Large arities or joins can make this bound expensive. A finite structured constant remains one value here; an extension that generated larger structures during inference would invalidate the active-domain premise. The proof also applies separately after each change in authorized input. It gives no monotonicity of answers across withdrawals or policy changes.

Proposition 13.2 states the precise non-explosion result. It concerns grounded rule support, leaving the substantive consequences of deliberately permissive domain rules visible to their authors.

Proposition 13.2 (Grounded consequence and absence of implicit explosion). *Every atom in $\llbracket P \rrbracket_E$ belongs to E or has a finite derivation from E through ground instances of declared rules. In particular, an atom absent from E and from every possible ground rule head is not derivable, even when both signs of another proposition occur in E .*

Proof. An atom first present at iteration zero belongs to E . An atom first introduced at iteration $n + 1$ is a ground rule head whose premises were present by iteration n . Induction supplies finite derivations for those premises and hence for the new atom. Finiteness of each body makes the constructed derivation finite. An atom absent from both the base and every possible head cannot enter at any stage. The argument does not change when the base contains both signed relations for some content. $\square$

A rule author can explicitly connect opposing support to another conclusion. That consequence follows the declared rule and may require domain criticism. The evaluator supplies no implicit classical rule deriving arbitrary atoms from opposition. Likewise, a recursive cycle that loses every base-grounded seed supplies no finite derivation and disappears under recomputation.

13.2 Withdrawal Locality and Admitted Replay

Proposition 13.3 formalizes the independence of separately attributed commitments. It concerns temporal activity, with disclosure and availability held fixed where a support consequence is drawn.

Proposition 13.3 (Locality of a targeted withdrawal). *Adding only a withdrawal targeting assertion a cannot change the temporal activity of a previously admitted assertion $b \neq a$. Its effect on a is confined to the intersection of the withdrawal interval and $I(a)$, at cuts containing the withdrawal.*

Proof. In Equation 16, a withdrawal affects b only if its target equals b . The added withdrawal has target a , so it supplies no new witness to that existential condition for b . For a , the new masking condition requires both membership of its introducing commit in the cut and membership of t in the withdrawal interval. Activity already requires $t \in I(a)$, giving the stated intersection. $\square$

Consequently, a visible available b can continue to support a shared extensional key after a is withdrawn. This consequence requires the specified view assumptions; a concurrent change to b 's disclosure cannot be ruled out by a theorem about withdrawal alone.

Proposition 13.4 concerns admitted deltas and receipts, rather than the admission of candidate commands in arbitrary order.

Proposition 13.4 (Deterministic projection of compatible admitted histories). *Equal available admitted deltas, exact schemas, and interpretation profiles produce equal immutable record maps, withdrawal masks, and operational token projections, independently of their topological replay order, provided repeated identities are compatible and receipts are fixed.*

Proof. Compatible record deltas agree at every repeated key, so their map union is independent of order. Assertions and withdrawals are identified elements combined by set union. Equation 16 depends on membership and target intervals in that resulting set, not traversal order. Produced tokens and consumption edges are fixed by the admitted receipts. Equation 23 selects the same residual tokens from equal token and edge sets. These component projections therefore agree for every topological enumeration of the same admitted graph. $\square$

Candidate commands can have order-dependent admissibility because each inspects a staged operational state. Proposition 13.4 does not commute those commands. It also assumes retained interpretable inputs; missing schemas, erased payloads, or changed disclosure conditions can prevent the same public reconstruction.

13.3 Typing and Security Obligations

The intended type-preservation argument proceeds by registering and checking the complete candidate record graph, validating typed references and values, and then preserving registry and receipt invariants through each operation. Immutable identity checks ensure that extension does not silently replace an existing typed payload. Cyclic domain references are permissible because all targets are registered before their values are checked. This gives a proof route, but a complete preservation theorem requires a fully specified elaborator and evaluator relation, including every error and receipt field.

Research Obligation 13.5 (Implementation-level type preservation). A complete implementation must show that source elaboration, schema and reference resolution, typed staged evaluation, and receipt generation agree with the specified judgments, and that every accepted transition preserves the well-formed store invariants. The current recognizer and semantic oracles do not establish this result.

Authorized input isolation also has a limited mathematical consequence. For equal authorized extensional bases and equal authorized programs, Equation 25 yields equal derived atom sets. That follows by induction on identical iterations. Full privacy requires substantially more: safe errors and completeness flags, restricted identifiers, suitable cache keys, protected execution traces, and explicit declassification conditions.

Research Obligation 13.6 (Observational disclosure protection). A host claiming privacy noninterference must define attacker observations and prove the appropriate equivalence across protected histories, including metadata, explanations, caches, control flow, storage, and transport. Filtering base records alone is insufficient evidence.

For example, two requests can receive identical rows while different error messages reveal whether a protected record exists. A stale cache can expose a result produced before permission changed. A proof explanation can disclose a hidden premise. These counterexamples motivate the host obligations without claiming that the current language package solves them.

13.4 Representational and Institutional Limits

The formal scheme defines equality and operations for selected descriptions. Its adequacy across domains remains open. Complete-field extensional keys can split descriptions that participants consider equivalent; intensional keys can retain distinct descriptions that later evidence identifies as one referent. Explicit mappings make such interpretations inspectable but can increase annotation and maintenance cost.

The core also leaves substantive evidentiary standards, credibility, institutional legitimacy, and applicable legal consequences to their appropriate inquiries. A well-typed finding can be unjust, an authenticated assertion can be false, and a complete workflow can conceal ineffective participation. Formal precision makes these distinctions available for criticism. It cannot substitute for the institutional resources and public judgment needed to use them responsibly.

14 Conclusion

This paper has specified a generative relational language for evolving social-fact records. Its central contribution is an integrated contract connecting typed descriptions, independently attributed commitments, temporal interpretation, finite inference, and authorized operational change. The language's structural categories support domain extension while keeping their adequacy open to evaluation against actual inquiries.

The semantic distinctions determine consequential behavior. Record identity and qualified content equality have separate roles. Assertions preserve speaker, context, polarity, and applicability independently of the content they target. Historical cuts and validity intervals support retrospective correction without changing an earlier recorded history. Four support states preserve explicit opposition and absence of available support without treating either as a universal truth judgment.

Operational semantics separates Event descriptions from Start, Advance, and Resolve commands. Atomic admission checks typed dependencies and trusted authority against a declared pre-state. Compatible branch union retains assertions and receipts while exposing concurrent token conflicts. Replay reconstructs admitted effects without reexecuting historical observations, external actions, or authorization decisions.

The finite rule core derives virtual signed predicates from an authorized base through a least fixed point. Its restrictions support termination and grounded consequence under the stated assumptions. Targeted withdrawal preserves other assertions' independence, and equal admitted receipt graphs yield deterministic token projections. Type preservation for a complete implementation, privacy noninterference, and practical adequacy remain separate obligations.

The supplied grammar, examples, exchange schema, and bounded checks make the proposal inspectable. They provide no complete interpreter, production security result, or field evidence. Further work should implement the contract incrementally, compare configured alternatives, and examine whether participants can represent consequential distinctions at acceptable cost. The value of the language will depend on its contribution to intelligible, contestable, and responsibly usable knowledge histories, alongside the institutions through which those histories become public.

A Complete Concrete Grammar

This appendix reproduces the authoritative GRK-DSL 0.1 EBNF directly from the accompanying artifact. Lexical token definitions and static conditions are specified in Sections 4 and 5. The EBNF uses

comma for sequence, a vertical bar for alternatives, brackets for optional forms, and braces for repetition. Syntactic recognition does not establish typing, authority, or operational admissibility.

Listing 4 is included from the versioned source during the document build, preserving its relationship to the recognizer's input.

Listing 4: GRK-DSL 0.1 concrete grammar

```
(* GRK-DSL 0.1. ISO-style EBNF: comma sequence, | alternative,
   [ optional ], { repetition }, double-quoted terminals.
   NAME, STRING, INTEGER and DECIMAL are lexical tokens defined in
   02_syntax_and_types.md. Whitespace and // line comments separate tokens. *)
program = "module", qname, "version", STRING, "namespace", STRING, ";",
    { import_decl }, { declaration }, { transaction }, { query } ;
import_decl = "import", qname, "version", STRING, "digest", STRING,
    "namespace", STRING, "as", NAME, ";" ;
declaration = enum_decl | type_decl | check_decl | exclusive_decl |
    machine_decl | rule_decl ;
enum_decl = "enum", NAME, "{", NAME, { ",", NAME }, "}", ";" ;
type_decl = kind, NAME, "{", { field_decl }, "}", ";" ;
kind = "entity" | "relation" | "state" | "property" | "process" | "event" ;
field_decl = NAME, ":", type, ";" ;
type = scalar | qname | "Ref", "<", qname, ">" |
    "List", "<", type, ">" | "Set", "<", type, ">" |
    "Option", "<", type, ">" ;
scalar = "Text" | "Bool" | "Int" | "Decimal" | "Time" | "Interval" |
    "Extent" | "IRI" | "AssertionRef" | "CommitRef" ;
check_decl = "check", NAME, "on", qname, ":", bool_expr, ";" ;
bool_expr = bool_term, { "or", bool_term } ;
bool_term = bool_factor, { "and", bool_factor } ;
bool_factor = "(" , bool_expr, ")" | "is_some", "(" , NAME, ")" |
    NAME, comparator, value ;
comparator = "==" | "!=" | "<" | "<=" | ">" | ">=" ;
exclusive_decl = "exclusive", NAME, "on", qname, "by", "(" ,
    NAME, { ",", NAME }, ")", ";" ;
machine_decl = "machine", NAME, "tracks", qname, "{",
    "initial", enum_value, ";", { step_decl }, "}", ";" ;
step_decl = "step", NAME, "from", enum_value, "to", enum_value,
    "on", qname, ";" ;
rule_decl = "rule", NAME, ":", atom, "<-", atom, { ",", atom }, ";" ;
atom = sign, qname, "(" , [ term, { ",", term } ], ")" ;
term = variable | value ;
variable = "?", NAME ;
transaction = "transaction", reference, "parents", ref_list,
    "by", reference, "recorded", time_value,
    "policy", reference, "{", { operation }, "}" ;
operation = record_op | assert_op | withdraw_op | start_op | advance_op |
    resolve_op ;
record_op = "record", reference, ":", qname, record_value, ";" ;
record_value = "{", { NAME, "=", value, ";" }, "}" ;
assert_op = "assert", reference, sign, reference,
    "by", reference, "in", reference, "valid", interval,
    "evidence", ref_list, "policy", reference, ";" ;
withdraw_op = "withdraw", reference, "assertion", reference,
    "valid", interval, "reason", reference, ";" ;
start_op = "start", reference, "run", reference, "machine", qname,
    "subject", reference, "event", reference, ";" ;
advance_op = "advance", reference, "run", reference, "via", NAME,
    "event", reference, "expect", ref_list, ";" ;
resolve_op = "resolve", reference, "run", reference,
    "heads", ref_list, "to", enum_value, "reason", reference, ";" ;
query = "query", NAME, "at", reference, "valid", time_value,
    "context", reference, "policy", reference, "{",
    query_body, "}" ;
query_body = "select", variable, { ",", variable }, "where", atom,
    { ",", atom }, ";" |
    "support", reference, ";" |
    "history", reference, ";" |
    "current", reference, ";" ;
value = STRING | INTEGER | DECIMAL | "true" | "false" | "none" |
    "some", "(" , value, ")" | reference | enum_value | time_value |
    interval | extent | iri_value | list_value | set_value ;
list_value = "[", [ value, { ",", value } ], "]" ;
set_value = "set", "[", [ value, { ",", value } ], "]" ;
```

```

time_value = "time", "(", STRING, ")" ;
iri_value = "iri", "(", STRING, ")" ;
interval = "[", time_value, ",", ( time_value | "*" ), "]" ;
extent = "at", "(", time_value, ")" | "during", "(", interval, ")" |
    "unknown" ;
enum_value = qname, ".", NAME ;
ref_list = "[", [ reference, { ",", reference } ], "]" ;
reference = "@", qname ;
qname = NAME, { ":", NAME } ;
sign = "+" | "-" ;

```

B Complete Workflow Fixture and Artifact Reproduction

This appendix provides a complete source example with declarations, transactions, branch integration, explicit repair, and queries. The authority and disclosure conditions are synthetic. Listing 5 is included directly from the fixture recognized by the syntax tool. Expected semantics are explained in Sections 8, 9, and 12; syntax recognition does not execute those semantics.

Listing 5: Constructed review workflow with concurrent branches and explicit resolution

```

module example_workflow version "0.1.0" namespace "urn:grk:example:workflow:";

enum Stage { proposed, contested, supported };
entity Person { label: Text; };
entity Context { label: Text; };
entity Policy { label: Text; };
entity Claim { content: Text; };
entity Reason { text: Text; };
state Review { subject: Ref<Claim>; value: Stage; };
process ReviewRun { span: Extent; };
event Opened { occurred: Extent; subject: Ref<Claim>; };
event EvidenceAdded { occurred: Extent; subject: Ref<Claim>; };
event Accepted { occurred: Extent; subject: Ref<Claim>; };
machine ReviewFlow tracks Review {
    initial Stage.proposed;
    step challenge from Stage.proposed to Stage.contested on EvidenceAdded;
    step accept from Stage.proposed to Stage.supported on Accepted;
    step reconsider from Stage.contested to Stage.supported on Accepted;
};

transaction @c0 parents [] by @editor recorded time("2026-06-01T09:00:00.000000Z") policy @policy {
    record @editor: Person { label = "Synthetic review steward"; };
    record @context: Context { label = "Review procedure"; };
    record @policy: Policy { label = "Fixture workflow authorization handle"; };
    record @claim: Claim { content = "Synthetic claim for workflow demonstration"; };
    record @run: ReviewRun { span = unknown; };
    record @opened: Opened { occurred = at(time("2026-06-01T08:00:00.000000Z")); subject = @claim; };
    start @token0 run @run machine ReviewFlow subject @claim event @opened;
}

transaction @left parents [@c0] by @editor recorded time("2026-06-02T09:00:00.000000Z") policy @policy {
    record @challenge: EvidenceAdded { occurred = unknown; subject = @claim; };
    advance @token_left run @run via challenge event @challenge expect [@token0];
}

transaction @right parents [@c0] by @editor recorded time("2026-06-02T10:00:00.000000Z") policy @policy {
    record @acceptance: Accepted { occurred = at(time("2026-06-02T08:00:00.000000Z")); subject = @claim; };
    advance @token_right run @run via accept event @acceptance expect [@token0];
}

// Importing/merging receipts preserves both concurrent updates.
transaction @merged parents [@left, @right] by @editor recorded time("2026-06-03T09:00:00.000000Z") policy
    @policy {}

transaction @resolved parents [@merged] by @editor recorded time("2026-06-04T09:00:00.000000Z") policy
    @policy {
        record @reason: Reason { text = "The authorized review process reopens consideration; the branch
            decisions remain in history."; };
        resolve @token_resolved run @run heads [@token_left, @token_right] to Stage.contested reason @reason;
    }

```

```
}
```

```
query OriginalRun at @c0 valid time("2026-06-01T00:00:00.000000Z") context @context policy @policy {  
  current @run; }  
query MergedRun at @merged valid time("2026-06-03T00:00:00.000000Z") context @context policy @policy {  
  current @run; }  
query ResolvedRun at @resolved valid time("2026-06-04T00:00:00.000000Z") context @context policy @policy {  
  current @run; }
```

The companion package also contains the complete contested-knowledge and local-release fixtures, the JSON exchange schema, and an unsigned structural example. From the project root, the following commands reproduce the recognition, semantic-oracle, and structural checks described in the paper. The structural checker requires the documented `jsonschema` dependency.

```
python infrastructure_src/design_checks/check_syntax.py --self-test "research/dsl/examples/*.grk"  
python infrastructure_src/design_checks/check_semantics.py  
python infrastructure_src/design_checks/check_ir.py
```

The scripts have different targets. The recognizer consumes the EBNF and source examples. The semantic oracle tests simplified mathematical models. The exchange checker validates JSON envelope structure. None is a complete compiler or interpreter, and no successful command should be interpreted as verification of an institution’s legal authority or public legitimacy.

References

- [1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. Author-hosted Chapter 12 verified 11 September 2026.
- [2] Nuel D. Belnap, Jr. A useful four-valued logic. In J. Michael Dunn and George Epstein, editors, *Modern Uses of Multiple-Valued Logic*, volume 2 of *Episteme*, pages 5–37. D. Reidel, 1977. Publisher abstract and metadata verified 11 September 2026.
- [3] Martin Fowler. Event sourcing. Author-hosted architecture essay, 12 December 2005, 2005. Author marks the material as draft; verified 11 September 2026.
- [4] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS ’07, pages 31–40. ACM, 2007. Author-hosted original verified 11 September 2026.
- [5] Christian S. Jensen, Curtis E. Dyreson, et al. The consensus glossary of temporal database concepts: February 1998 version. In *Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, pages 367–405. Springer, 1998. Chapter editors and contributing authors; verified 11 September 2026.
- [6] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. Verified 11 September 2026.
- [7] Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976. Author-hosted original verified 11 September 2026.
- [8] World Wide Web Consortium. PROV-DM: The PROV data model. W3C Recommendation, 30 April 2013, 2013. Verified 11 September 2026.
- [9] World Wide Web Consortium. SPARQL 1.1 query language. W3C Recommendation, 21 March 2013, 2013. Verified 11 September 2026.
- [10] World Wide Web Consortium. RDF 1.1 concepts and abstract syntax. W3C Recommendation, 25 February 2014, 2014. Verified 11 September 2026.

- [11] World Wide Web Consortium. Shapes constraint language (SHACL). W3C Recommendation, 20 July 2017, 2017. Verified 11 September 2026.

WORKING DRAFT