Rules, Protocols and an Ontology on Ordinals, Bitcoin

Three Protocols on PACT for Public Reasoning on an Immutable Substrate

Wiard Vasen

May 2026

Abstract

This document specifies three protocols built on PACT v1: KINSHIP, LINEAGE, and RULES. Together they extend PACT's sat-bound identity model with relations between identities and deductive inference over those relations, all anchored on Bitcoin through Ordinals inscriptions. KINSHIP declares symmetric kinship between avatars. LINEAGE acknowledges asymmetric intellectual precedent. RULES contains logical statements in Datalog form, evaluated deterministically by any reader against chain state. The three protocols share PACT's envelope discipline and add no new ontological primitives; what they add is the inferential layer that allows public reasoning to become a property of the wall itself rather than of any reader.

Contents

1 Introduction

1.1 What this paper specifies

This paper specifies three protocols that build on PACT v1: KINSHIP, LINEAGE, and RULES. Each protocol is implemented through PACT's existing envelope discipline and adds no new ontological primitives. What they add is the layer above identity: relations between sat-bound entities, recognition of intellectual precedent, and deductive inference over both.

PACT v1, as specified by OrdinalsLover [2], defines how identity can be anchored to a satoshi and how transitions on that identity can be recorded through Ordinals inscriptions on Bitcoin. That specification deliberately stops at identity and action. It does not define how identities relate to one another, how an identity acknowledges its predecessors, or how an inference engine may derive new facts from what is on chain. Those questions are the subject of this paper.

The three protocols share three properties that follow from their substrate. They are permanent, because Bitcoin is. They are openly authored, because no registrar mediates inscription. They are deterministically reproducible, because their semantics are fixed and their evaluation is computational rather than discretionary. Together these properties form what this paper calls *public reasoning*: not merely showing publicly what has been derived, but making the derivation itself publicly testable on a substrate no instance can retract.

1.2 What this paper assumes

The reader is assumed to be familiar with PACT v1 [2], in particular with its four-part ontology (namespace, world, avatar, transition), its envelope form, the sat-bound identity model, and the Merkle-based verification methods described there. This paper does not restate that material. Where extension is needed, the relevant section of PACT v1 is cited.

The reader is also assumed to have working knowledge of Bitcoin Ordinals [3] as a transport layer, and basic familiarity with declarative logic programming, in particular Datalog [1]. The Datalog requirement is light: this paper introduces the syntax it uses and illustrates evaluation through worked examples.

1.3 What this paper does not do

This paper does not propose a new ontology. The four primitives defined in PACT v1 are sufficient for everything specified here. KINSHIP, LINEAGE, and RULES are each instantiated as TRANSITION objects under existing namespace and world rules. What changes is not the envelope but the semantics carried in the data field, and the validity rules that govern that data.

This paper does not specify autonomous execution. None of the three protocols moves value, transfers ownership, or triggers external actions. They are declarative: they record claims and allow conclusions to be derived from those claims. Acting on the conclusions, when action is warranted, is left to readers of the chain.

This paper does not specify a complete reasoning system. RULES in its current form is a minimal Datalog dialect, sufficient for the inference patterns currently demonstrated on the Ordinalswall but not intended as a general-purpose theorem prover. Section 12 discusses what would be needed for richer reasoning, and what should remain outside the scope of this protocol.

1.4 Status of this document

The three protocols specified here have all been inscribed on Bitcoin mainnet at the time of writing, together with at least one working example of each in use. Section 10 describes the

Ordinalswall live demonstration where these inscriptions can be inspected and the derivations they support can be reproduced.

The specification reflects what is currently working. Where a specification element is intended for a future protocol version (notably RULES v2, sketched in Section 5), this is stated explicitly. The reader can treat the rest as normative for what is on chain today.

1.5 Notation and conventions

This paper follows the notation and conventions of PACT v1 [2]:

- PACT in uppercase denotes the literal protocol identifier in active envelopes.
- A sat is a satoshi interpreted under ordinal numbering.
- JSON field names are shown in monospace, for example `kinship` or `rule_id`.
- The words MUST, SHOULD, and MAY are used in their ordinary specification sense.

For the new material in this paper, additionally:

- KINSHIP, LINEAGE, and RULES in uppercase denote the three protocols specified here, each implemented as a TRANSITION action under PACT v1.
- A *derivation* is a fact computed by applying a rule to chain-state. Derivations are not inscribed; they are deterministic functions of what is on chain.
- A *premise* is a fact or earlier derivation that supports a derivation. Premises are traceable: any derivation can be unfolded to the chain-inscribed facts that support it.

1.6 Document structure

Section 2 describes the problem the three protocols together address: relations and reasoning without authority. Sections 3, 4, and 5 specify each protocol in turn. Section 6 shows how the three compose into the four-layer reasoning stack visible on the Ordinalswall live page. Sections 7 and 8 discuss verification and intellectual context. Section 9 formalises the notion of public reasoning that the three protocols together make possible. Section 10 describes the working demonstration. Section 11 states what the stack cannot do. Section 12 sketches extensions that follow internally from what stands. Section 13 acknowledges the work this paper builds on.

2 Foreword: relations without authority

2.1 The problem the three protocols address

PACT v1 [2] solves a specific problem: how an identity can be anchored to a substrate that no party owns. The four primitives — namespace, world, avatar, transition — together with the sat-bound identity model and the Ordinals inscription envelope, give an entity a way to exist on Bitcoin in a manner no registrar can revoke. That is the layer below which nothing further is needed for identity itself.

But identity in isolation is not yet a system that can reason about itself. A namespace declares a context. A world declares rules within a context. An avatar declares its own existence. A transition records what an avatar has done. None of these records, taken alone, says anything about how avatars stand in relation to one another, or about what may be inferred from the relations that exist. PACT v1 leaves the structure of relations between identities, and the inferential layer that may be built on those relations, deliberately open.

This paper closes that opening with three additions. KINSHIP declares that two or more avatars stand in mutual kin. LINEAGE acknowledges that one avatar or work descends from another. RULES contains logical statements that allow conclusions to be drawn from inscribed facts. Each of the three is implemented as an ordinary PACT v1 TRANSITION; what changes is not the envelope but the semantics carried in the data field.

The combination produces a layer above identity in which public reasoning becomes possible. By *public reasoning* this paper means a specific thing: not merely that conclusions are made publicly visible, but that the derivation of those conclusions is itself publicly testable, by anyone, on a substrate no instance can retract.

2.2 Why this layer was missing

Logical reasoning systems have existed for a long time. Prolog dates from 1972. Datalog dates from 1977. The Semantic Web, with its RDF, OWL, and SWRL specifications, has been under active development for over twenty years. Each of these traditions has produced sophisticated formalisms for representing knowledge and deriving consequences from it.

What none of them have produced, until the substrate described in this paper became available, is a system in which the facts themselves enjoy the same persistence as the reasoning over them. Prolog facts live in a running interpreter. Datalog facts live in a database that an operator maintains. RDF triples live on web servers, with URIs that may rot and SPARQL endpoints that may go offline. The reasoning engines built on these substrates are mathematically sound, but their soundness delivers durable conclusions only as long as the underlying facts are durable. When the operator stops paying for the hosting, the conclusions become unreproducible — not because the logic was wrong, but because the premises are gone.

Bitcoin, in combination with Ordinals [3], supplies a substrate where this is no longer the case. Inscribed facts do not depend on any operator. They cannot be unilaterally removed by the inscriber, by an institution, or by an intermediary. They are reachable through ordinary tools and verifiable without privileged software. The substrate that logic programming and the Semantic Web have always assumed — or hoped for — is now available, and it is available without asking permission.

The three protocols specified in this paper bring the missing inferential layer to that substrate. They do not invent the techniques; the techniques are decades old. They place the techniques on a foundation that gives their conclusions the same lifespan as Bitcoin itself.

2.3 What relations without authority means

The phrase *without authority* requires care. It does not mean that no one is responsible for what is inscribed. Whoever controls the satoshi on which an inscription is placed bears responsibility for that inscription, just as anyone who publishes any statement bears responsibility for what they publish. What the phrase means is more specific: that no third party — no platform, no registrar, no governing body — is required to validate, mediate, or grant permission for an inscription to take effect.

A KINSHIP declared between two avatars is in effect because it is on chain. No body of standards needs to recognise it. A LINEAGE acknowledging a precedent is in effect because it is inscribed; the acknowledged party is not required to consent or to know. A RULE proposing an inference pattern is in effect because it has been written down on a substrate that does not forget; whether other readers apply that rule is a separate question, but the rule itself stands.

This is a different model from the one familiar in academic publishing or institutional authentication. There, validity is conferred by recognition: a paper is valid because a journal publishes it, a credential is valid because an institution issued it, a relation is valid because a registrar recorded it. The three protocols here invert that relationship. Validity is conferred by inscription on a substrate that no party owns; recognition is something readers do afterwards, not something the inscription depends on.

This inversion has consequences. Some are liberating: anyone can declare anything, no permission is needed, the wall does not gatekeep. Others are sobering: anyone can declare anything, no quality filter applies before inscription, and the work of distinguishing well-supported claims from poorly-supported ones falls to readers rather than to gatekeepers. Both consequences are part of what *without authority* entails.

2.4 What this layer enables

When relations between identities and rules over those relations exist on the same persistent substrate as the identities themselves, several things become possible that were not possible before.

First, attribution survives its participants. A LINEAGE acknowledgement of a predecessor remains visible long after the predecessor's website has gone offline, after the journals indexing their work have ceased publication, after the institutions that employed them have closed. The acknowledgement does not depend on any of these to remain discoverable.

Second, derivations are reproducible. Two readers, applying the same rule to the same chain-state, will reach identical conclusions. There is no privileged interpretation, no version of the truth held by a server. The derivation is a function; the inputs are public; the output is a property of the function and the inputs.

Third, the system can reason about itself. A rule can be declared that detects contradictions in inscribed facts. A rule can be declared that traces intellectual descent across multiple LINEAGE generations. A rule can be declared that identifies pairs of avatars who are related but in different namespaces. The wall reads back, and what it reads back can itself be made the subject of further reading.

These possibilities are what motivate the specifications in the chapters that follow. The next three chapters define KINSHIP, LINEAGE, and RULES respectively, in their PACT v1 envelope form. Chapter 6 shows how they compose. The remaining chapters establish verification, intellectual context, and the boundaries of what this layer can and cannot do.

3 KINSHIP

3.1 What KINSHIP is

KINSHIP is a relation protocol. A KINSHIP inscription declares a symmetric relation among two or more avatars. The semantics are deliberately low: KINSHIP states that the named avatars stand in a community of kin, but does not specify what kind of kinship, what its consequences are, or what would dissolve it. Interpretation of the kinship is left to RULES inscribed under the relevant world or namespace, or to readers of the chain.

The choice to keep semantics low is intentional. A relation protocol that prescribes too much closes off applications its authors did not anticipate; a relation protocol that prescribes nothing fails to be a protocol at all. KINSHIP takes a middle position: it formalises the act of declaration and the identities involved, but leaves the meaning of the kinship to the world that interprets it. Two avatars can be in kinship because they share a creator, because they cooperate on a project, because they descend from the same lineage of work, or for reasons their inscriber alone understands. The protocol records that the kinship exists; what it means is determined elsewhere.

3.2 Envelope form

A KINSHIP inscription is an ordinary PACT v1 TRANSITION. The envelope follows the form specified in PACT v1 [2], with action set to KINSHIP and a data field containing the participating avatars and any additional metadata.

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "<namespace>",
  "world": "<world>",
  "avatar": "<inscriber-avatar>",
  "action": "KINSHIP",
  "data": {
    "kinship": [
      "<namespace>/<world>/<avatar-name-1>",
      "<namespace>/<world>/<avatar-name-2>",
      "<namespace>/<world>/<avatar-name-3>"
    ],
    "declared_at": "<ISO 8601 timestamp>",
    "note": "<optional human-readable note>"
  }
}
```

3.3 Required fields

The KINSHIP-specific fields inside data are:

- `kinship`: an array of avatar references, each of the form `namespace/world/avatar-name`. The array MUST contain at least two distinct entries. Order is not significant; KINSHIP is symmetric.
- `declared_at`: an ISO 8601 timestamp marking when the kinship is declared. The timestamp is informational; the authoritative ordering is supplied by the Bitcoin block in which the inscription is confirmed.

The `note` field is optional and carries a free-form human-readable description of the kinship. It is not parsed by the protocol.

3.4 Validity rules

A KINSHIP inscription is valid under PACT v1 if its envelope is structurally valid (Section 5 of [2]) and the following additional rules hold:

1. The `kinship` array MUST contain at least two distinct entries.
2. Each entry in the `kinship` array MUST be a well-formed avatar reference. The protocol does not require, at parse time, that the referenced avatar already exists on chain; a kinship declared toward a not-yet-inscribed avatar takes effect once that avatar is constituted.
3. The inscribing avatar (the `avatar` field at the envelope level) SHOULD be one of the entries in the `kinship` array. A KINSHIP inscribed by an avatar that does not include itself in the kinship is technically valid but semantically peculiar; readers and rules may treat it differently from self-inclusive kinship.
4. Two KINSHIP inscriptions involving the same set of avatars are not in conflict. Each is a separate declaration; both stand on chain. A reader may interpret repeated declarations as confirmation, as accumulation of evidence, or as redundant — that choice belongs to the world or to applicable rules.

3.5 Symmetry and asymmetry

KINSHIP is symmetric. If avatar A inscribes a KINSHIP that includes B, the kinship between A and B exists from the moment the inscription confirms, regardless of whether B has issued any reciprocal inscription. B's silence does not negate the kinship; only B's explicit inscription of an opposing claim would create a tension that readers and rules would have to resolve.

This asymmetry of declaration combined with symmetry of relation is unusual but deliberate. It reflects the substrate. On Bitcoin, anyone can inscribe; nothing prevents one party from declaring kinship without the other's involvement. The protocol does not pretend otherwise. It records what was declared, by whom, and when. The question of whether B accepts or contests the kinship is a matter for B's own subsequent inscriptions, and for the rules that interpret kinship declarations within the relevant world.

A future protocol could specify a CO-KINSHIP variant that requires explicit confirmation by all named parties. Such a protocol would offer stronger guarantees but at the cost of ease of declaration. KINSHIP as specified here favours ease: any avatar can declare a kinship at any time, and the chain records what was declared. Stronger forms of mutual recognition can be built on top by inscribing rules that interpret only those KINSHIP records confirmed by all named parties.

3.6 What KINSHIP does not specify

KINSHIP does not specify the nature of the kinship. The same inscription form is used for all kinds of kin: collaborators, co-creators, contemporaries, members of a school, participants in a project. The protocol records that the relation exists; the world or applicable rules determine what kind of relation it is.

KINSHIP does not specify dissolution. There is no UNKINSHIP action in this protocol. If an avatar wishes to record that a previously declared kinship no longer holds, the avatar may inscribe a new TRANSITION whose action and data express that withdrawal under whatever

conventions the world recognises. The original KINSHIP remains on chain; what changes is the derived current state, computed by readers who apply the withdrawal-recognising rules.

KINSHIP does not specify exclusivity. An avatar may participate in any number of distinct KINSHIP inscriptions, each involving a different set of avatars. The protocol places no upper bound on participation. Whether a particular world treats one kinship as superseding another is a matter for the world's rules.

3.7 Worked example

The first KINSHIP inscription on chain links three avatars in the pact-semantics namespace, each in a different world:

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "pact-semantics",
  "world": "open-field",
  "avatar": "wanderer-001",
  "action": "KINSHIP",
  "data": {
    "kinship": [
      "pact-semantics/open-field/wanderer-001",
      "pact-semantics/ordinals/forecaster-001",
      "pact-semantics/mathematics/vonneumann-001"
    ],
    "declared_at": "2026-05-02T00:00:00Z",
    "note": "Three avatars in mutual kin across three worlds."
  }
}
```

The inscription is valid: the envelope is well-formed, the kinship array contains three distinct entries, and the inscribing avatar (wanderer-001) appears in the array.

The kinship spans three worlds — open-field, ordinals, mathematics — illustrating that KINSHIP is not constrained to a single world. An avatar in one world can declare kinship with avatars in other worlds, provided the inscriber controls its own sat. The cross-world reach of KINSHIP is one of the properties that makes it useful as a primitive for the inferential layer specified in Chapter 5: rules can derive relations that hold across worlds, not only within them.

3.8 What KINSHIP enables

In isolation, a KINSHIP inscription declares only that certain avatars are in kin. The richer consequences emerge when KINSHIP is combined with RULES. A rule of the form

```
IF kinship(A,B) AND avatar(A) AND avatar(B) THEN related(A,B)
```

interprets KINSHIP as implicit relation between kinship members. Different rules could interpret the same KINSHIP differently: as eligibility for joint authorship, as shared responsibility, as transitive trust. The protocol records the kinship; rules determine the consequences.

This is the pattern the rest of this paper develops. KINSHIP is one of the primitive relations on which inference is built; LINEAGE is another; RULES is the language in which the inference is expressed. Together they form the layer above PACT v1's identity model in which public reasoning becomes possible.

4 LINEAGE

4.1 What LINEAGE is

LINEAGE is a recognition protocol. A LINEAGE inscription acknowledges that the inscriber's work, identity, or contribution descends from an earlier source. Unlike KINSHIP, LINEAGE is asymmetric: avatar A acknowledges B, but B is not required and is often not in a position to reciprocate. The acknowledged source may have been inscribed elsewhere, may have lived only on the web before any chain existed, or may predate digital computation entirely.

LINEAGE formalises a practice that academic citation, artistic attribution, and intellectual gratitude have always assumed without making structural. Citing a paper acknowledges that the citing work is partly indebted to the cited work; placing a name in an acknowledgements section recognises a contribution that is not part of the formal authorship. These practices are ancient, but their durability has always depended on the durability of the documents that record them. Papers go out of print. Acknowledgements pages are read once and forgotten. The recognition is real but the record is fragile.

LINEAGE on chain offers something the existing practices do not: an acknowledgement that no instance can retract. The inscriber cannot remove it; the acknowledged party cannot demand its removal; the institution that holds the inscriber's employment, if any, has no power to disown it. The acknowledgement stands as long as Bitcoin stands.

4.2 Envelope form

A LINEAGE inscription is an ordinary PACT v1 TRANSITION. The envelope follows the form specified in PACT v1 [2], with action set to LINEAGE and a data field identifying the acknowledged source.

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "<namespace>",
  "world": "<world>",
  "avatar": "<inscriber-avatar>",
  "action": "LINEAGE",
  "data": {
    "acknowledges": "<source identifier>",
    "kind": "<descent | influence | dependency | precedent>",
    "declared_at": "<ISO 8601 timestamp>",
    "note": "<human-readable explanation>"
  }
}
```

4.3 Required fields

The LINEAGE-specific fields inside data are:

- `acknowledges`: a string identifying the source being acknowledged. The identifier may be: a PACT avatar reference of the form `namespace/world/avatar-name`; an inscription identifier; a sat number with optional inscription index; a URL; a citation in plain text; or any other form the inscriber considers sufficient. The protocol does not enforce a single identifier scheme because sources predate the chain by varying degrees, and no single scheme can name all of them.

- `kind`: a label characterising the nature of the descent. The protocol recognises four canonical kinds (descent, influence, dependency, precedent) but does not restrict the field to these. World- specific extensions may define additional kinds.
- `declared_at`: an ISO 8601 timestamp marking when the lineage is declared. Authoritative ordering is supplied by the Bitcoin block.

The `note` field is optional and carries a free-form explanation of why the lineage is being acknowledged. For LINEAGE more than for KINSHIP, the note is often the substantive content: it explains what is owed, in what way, and for what reason.

4.4 The four canonical kinds

The four canonical kinds are not exhaustive but cover the most common reasons for acknowledgement.

Descent. The inscriber's work is a direct continuation of the acknowledged source. The relationship is genealogical: the new work would not exist, in its current form, without the earlier work as parent. PACT v1 acknowledging the Semantic Web as descent is an example of this kind: PACT v1 inherits the four-part ontology pattern from a long lineage of ontology work.

Influence. The inscriber's work has been shaped by the acknowledged source without being a direct continuation. The earlier work changed how the inscriber thinks, but the new work pursues a different direction. Acknowledging a teacher or a transformative reading is typically an influence acknowledgement.

Dependency. The inscriber's work technically requires the acknowledged source to function. A specification that builds on an earlier specification declares dependency. The relationship is operational: without the earlier work, the new work cannot be implemented as specified.

Precedent. The inscriber's work follows a path the acknowledged source first opened, but the relationship is not direct continuation, not specific influence, and not operational dependency. Precedent acknowledgements honour the priority of those who arrived earlier without claiming descent from their specific work. Bitoshi Blockamoto's Bitmap Signal Theory was acknowledged by PACT as ontological precedent: PACT does not implement BST, does not depend on BST, and is not a direct descendant of BST, but BST opened the conceptual space in which sat-bound identity could be imagined as a protocol-level claim.

4.5 Validity rules

A LINEAGE inscription is valid under PACT v1 if its envelope is structurally valid (Section 5 of [2]) and the following additional rules hold:

1. The `acknowledges` field MUST be a non-empty string. The protocol does not validate that the identified source actually exists; that determination is left to readers and to rules that may inspect external references.
2. The `kind` field MUST be present. Values outside the four canonical kinds are permitted but SHOULD be documented in the note.
3. Multiple LINEAGE inscriptions from the same avatar toward the same source are permitted. They are not in conflict; each stands as a separate acknowledgement, possibly of different kinds or in different contexts.

4. A LINEAGE inscription cannot be revoked by a subsequent inscription. The acknowledgement, once made, remains on chain. An inscriber may inscribe a later transition that qualifies, contextualises, or contradicts the earlier acknowledgement, but the original LINEAGE is not removed.

The fourth rule deserves emphasis. LINEAGE is the most permanent of the three protocols specified in this paper because acknowledgement, once made, is not the kind of thing that should be quietly withdrawn. A relation between living parties (KINSHIP) may evolve and be re-described. A rule (RULES) may be superseded by a better rule. But to acknowledge a debt and then to remove that acknowledgement later is a different kind of act, and the protocol declines to support it. What the protocol allows is contextualising the debt through a later inscription; the original record stands regardless.

4.6 Asymmetry of declaration

Like KINSHIP, LINEAGE is unilateral in its declaration. The inscriber acknowledges; the acknowledged party is not asked and may not be aware. Unlike KINSHIP, where the symmetry of the relation creates a question about whether mutual recognition should be required, LINEAGE has no such question. Acknowledging a predecessor is intrinsically the inscriber's act, and the predecessor's role is to have done the work that prompted the acknowledgement. Reciprocity is structurally unavailable when the acknowledged source predates the chain or the inscriber.

This asymmetry is precisely what makes LINEAGE useful for acknowledging long-dead authors, deprecated systems, and intellectual movements that have no living representative. A specification that builds on Datalog can acknowledge Hervé Gallaire and Jack Minker without their participation; a philosophical work that builds on Spinoza can acknowledge him without his consent. The acknowledgement is real because it is recorded; what makes it valuable is that it cannot be withdrawn, even by the inscriber, even later.

4.7 What LINEAGE does not specify

LINEAGE does not specify the truth of the acknowledgement. An inscriber who declares descent from a source they have not read, or influence from a school they have only browsed, is making a false claim — but the protocol does not detect falsity. What the protocol records is that the claim was made, on chain, by an identifiable inscriber. Whether the claim is defensible is a matter for readers, for those familiar with the acknowledged source, and for any rules that may cross-check claims.

LINEAGE does not specify completeness. An inscriber who acknowledges only some of the sources they are indebted to is not in violation of the protocol. The protocol does not require comprehensive acknowledgement; it provides a mechanism for the acknowledgements that are made. Selective acknowledgement is a feature of all citation practices and LINEAGE inherits this characteristic.

LINEAGE does not specify hierarchy. A LINEAGE inscription acknowledges one source; if the inscriber is indebted to several sources, multiple LINEAGE inscriptions are required. The protocol does not provide a single inscription form for acknowledging a network of influences. This is a deliberate simplicity: each acknowledgement is a separate act, and combining several into one inscription would obscure their individual character.

4.8 Worked example

The first LINEAGE inscription on chain acknowledges Bitoshi Blockamoto and Bitmap Signal Theory as ontological precedent to PACT:

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "pact-semantics",
  "world": "open-field",
  "avatar": "wanderer-001",
  "action": "LINEAGE",
  "data": {
    "acknowledges": "blockamoto/bst",
    "kind": "precedent",
    "declared_at": "2026-05-03T00:00:00Z",
    "note": "Bitmap Signal Theory recognised an early form of sat-bound claim. PACT
        extends this principle to identity in a fuller sense: sat-bound, context-
        bearing, and persistent across time."
  }
}
```

The inscription is valid: the envelope is well-formed, the acknowledged source is identified, the kind is precedent, and the note explains the nature of the acknowledgement. The acknowledgement does not require Bitoshi's consent and does not depend on BST itself being inscribed in PACT-compatible form; it stands as a record that PACT recognises BST as the source from which a particular conceptual move descends.

4.9 What LINEAGE enables

In isolation, a LINEAGE inscription records one acknowledgement. The richer consequences emerge when LINEAGE is combined with RULES, as the same pattern observed for KINSHIP. A rule of the form

```
IF lineage(A, B) AND lineage(B, C) THEN intellectual_descent(A, C)
```

makes intellectual descent transitive across multiple generations of acknowledgement. Other rules could interpret LINEAGE differently: as conferring trust, as creating obligations of citation, as marking eligibility for certain contexts. The protocol records the acknowledgement; rules determine the consequences.

LINEAGE also makes possible something that KINSHIP does not: a chain of recognition that crosses generations. KINSHIP links contemporaries; LINEAGE links the present to the past. Together they describe both the horizontal community of current participants and the vertical genealogy of the work they continue.

5 RULES

5.1 What RULES is

RULES is the inference protocol. A RULES inscription contains a logical statement that, when applied to facts on chain, produces derived facts. The statement takes the form of a Datalog rule: a head predicate that holds whenever a body of predicates can be satisfied. RULES inherits from Datalog [1] the formal properties of a half-century of research in declarative logic programming: termination in PTIME for non-recursive rules, semi-naive evaluation for recursive rules, model-theoretic semantics with unambiguous fixpoints.

What RULES adds to Datalog is not new logic but a new substrate. Traditional Datalog facts and rules live in a running interpreter or a database. They produce conclusions that are valid as long as the system that holds them is running. RULES on Bitcoin, in combination with the facts inscribed under PACT v1, KINSHIP, and LINEAGE, gives the inferential layer the same persistence as the underlying substrate. A rule inscribed in 2026 will produce the same derivations from the same facts in 2126, by anyone running the same evaluation, without consulting any operator.

5.2 Envelope form (v1)

A RULES inscription is an ordinary PACT v1 TRANSITION. The envelope follows the form specified in PACT v1 [2], with action set to RULES and a data field containing the rule expression and supporting metadata.

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "<namespace>",
  "world": "<world>",
  "avatar": "<inscriber-avatar>",
  "action": "RULES",
  "data": {
    "rule_id": "<unique identifier within scope>",
    "expression": "IF <body> THEN <head>",
    "applies_to_protocols": ["PACT", "KINSHIP", "LINEAGE"],
    "declared_by": "<inscriber-avatar>",
    "declared_at": "<ISO 8601 timestamp>",
    "note": "<human-readable explanation>"
  }
}
```

The v1 envelope carries the rule as a string in the `expression` field. The string follows a restricted Datalog syntax described below. Future versions of the protocol may carry the rule as a structured abstract syntax tree (AST) rather than a string; Section 5.8 sketches that possibility.

5.3 Restricted Datalog syntax

A rule has the form:

IF b_1 AND b_2 AND ...AND b_n THEN h

where each b_i is a body literal and h is the head literal. A literal has the form `predicate`$(t_1, t_2, \ldots, t_k)$ where each t_i is either a variable (uppercase identifier) or a constant (string in quotes, or a typed reference such as an avatar identifier).

The following constraints apply to v1:

1. All variables in the head MUST also appear in the body (the safety condition). This guarantees that every derivation is grounded in chain-inscribed facts.
2. Negation MAY appear before body literals in the form `NOT predicate(...)`. Negation is interpreted under the closed-world assumption: a literal is taken to be false if it is not derivable. Stratification is required (a predicate that appears negated in one rule's body MUST NOT depend, directly or transitively, on its own derivation).
3. Recursive rules ARE permitted. A predicate may appear in both the head and the body of the same rule, or in a cycle of rules. Evaluation uses semi-naive Datalog evaluation with fixpoint detection.
4. Built-in predicates such as equality, ordering, and string comparison MAY be used but MUST be specified explicitly by the world or by a separate rule. The protocol does not provide a default set of built-ins; each world declares what is available within it.

5.4 Required fields

The RULES-specific fields inside data are:

- `rule_id`: a string uniquely identifying the rule within its declaring scope (typically the namespace or world). The id is used by other rules and by readers to refer to this rule.
- `expression`: the rule itself, expressed as a string in the syntax described above.
- `applies_to_protocols`: an array listing the protocols whose facts the rule consumes. The array SHOULD include each protocol whose predicates appear in the rule body. This field is informational; it does not constrain evaluation.
- `declared_by`: an avatar reference identifying the inscriber. This MUST match the envelope-level `avatar` field.
- `declared_at`: an ISO 8601 timestamp.

The `note` field is optional and carries a human-readable explanation of what the rule means and why it was inscribed. For RULES, the note is often essential context: a rule expressed in restricted Datalog syntax may be precise but opaque to a casual reader.

5.5 Validity rules

A RULES inscription is valid under PACT v1 if its envelope is structurally valid and the following additional rules hold:

1. The `expression` field MUST be a syntactically valid rule under the v1 grammar.
2. All variables in the head MUST appear in the body (safety).
3. Negation, where used, MUST be stratified across the full set of rules being applied together.
4. The `rule_id` MUST be unique within the declaring scope at the moment of inscription. Two rules with the same id from the same scope are ambiguous; the protocol does not specify which takes precedence, and readers may treat the conflict as an error.
5. The protocol does not validate that the predicates named in the rule are predicates that exist in the chain-inscribed facts. A rule that references non-existent predicates is syntactically valid but semantically inert; it derives nothing because no body can be satisfied.

5.6 Evaluation

A reader applies a set of rules to a set of facts by semi-naive evaluation:

1. Begin with the set of inscribed facts as the initial derivation state.
2. For each rule, attempt to satisfy the body against the current state. Each successful binding produces a new derived fact (the head with variables substituted by the matched constants).
3. Add new derivations to the state.
4. Repeat until no rule produces a new derivation. This is the fixpoint.

The fixpoint is unique. Two readers, applying the same rules to the same facts, reach identical states. This is what makes public reasoning reproducible: the conclusions are not opinions, they are functions of the inputs.

For non-recursive rules, the fixpoint is reached in a single pass. For recursive rules, multiple passes are required; semi-naive evaluation tracks which derivations are new in each pass to avoid redundant work. The evaluation always terminates: Datalog without function symbols is decidable, and the number of possible derivations is bounded by the number of distinct combinations of constants in the facts.

5.7 Worked example: the first rule on chain

The first RULES inscription on chain interprets KINSHIP as implicit relation between kinship members:


```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "pact-semantics",
  "world": "open-field",
  "avatar": "wanderer-001",
  "action": "RULES",
  "data": {
    "rule_id": "kinship-implies-relation",
    "expression": "IF kinship(A,B) AND avatar(A) AND avatar(B) THEN related(A,B)",
    "applies_to_protocols": ["PACT", "KINSHIP"],
    "declared_by": "pact-semantics/open-field/wanderer-001",
    "declared_at": "2026-05-04T00:00:00Z",
    "note": "If two avatars are in mutual kinship, they stand in the relation '
        related' to each other. This rule formalises the implicit consequence of
        KINSHIP without prescribing the nature of the relation."
  }
}
```


Applied to the KINSHIP and PACT facts on chain at the time of writing, this rule produces three derivations. The KINSHIP inscription declared mutual kin among Wanderer-001, Forecaster-001, and vonNeumann-001. The rule unifies pairs:

- `kinship(W, F) AND avatar(W) AND avatar(F)` is satisfied; therefore `related(W, F)` is derived.
- `kinship(F, V) AND avatar(F) AND avatar(V)` is satisfied; therefore `related(F, V)` is derived.

- `kinship(W, V) AND avatar(W) AND avatar(V)` is satisfied; therefore `related(W, V)` is derived.

None of the three `related` facts is on chain. All three are derivable by any reader who applies the rule to the chain-state. This is the simplest possible illustration of public reasoning: the premises are inscribed, the rule is inscribed, the conclusions follow without anyone needing to inscribe them.

5.8 The v2 sketch

The v1 envelope carries the rule as a string. This is sufficient for present use but has two known weaknesses. First, parsing the expression string is not part of the envelope-validation step in PACT v1; readers must implement their own parser, and parsers may differ in their interpretation of edge cases. Second, the string form makes it harder to reference predicates across rules and to declare predicate signatures explicitly.

A future v2 version of RULES could address both weaknesses by carrying the rule as a structured abstract syntax tree, and by including explicit predicate definitions:

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "pact-semantics",
  "world": "open-field",
  "avatar": "wanderer-001",
  "action": "RULES",
  "data": {
    "rule_id": "kinship-implies-relation",
    "version": 2,
    "predicate_definitions": {
      "related": {
        "arity": 2,
        "args": ["avatar_ref", "avatar_ref"],
        "description": "Symmetric relation derived from kinship."
      }
    },
    "rule_ast": {
      "head": {"predicate": "related", "args": ["A", "B"]},
      "body": [
        {"predicate": "kinship", "args": ["A", "B"]},
        {"predicate": "avatar", "args": ["A"]},
        {"predicate": "avatar", "args": ["B"]}
      ]
    },
    "applies_to_protocols": ["PACT", "KINSHIP"],
    "declared_by": "pact-semantics/open-field/wanderer-001",
    "declared_at": "<future timestamp>",
    "note": "v2 envelope with structured AST and explicit predicate definitions."
  }
}
```

The v2 sketch is not normative for present inscriptions. The v1 envelope remains the form in current use. The v2 form is described here so that readers and future implementers can see the direction the protocol may take, and so that v1 inscriptions made now can later be migrated to v2 by parsing their expression strings into ASTs without loss of information.

5.9 What RULES does not do

RULES is Datalog, not Prolog. It does not support arbitrary recursion with negation in cyclic dependencies; stratification is required. It does not support function symbols (Prolog's `foo(bar(X))` construct); facts and derivations are flat. It does not support arithmetic, optimisation, or imperative control flow.

These limitations are deliberate. Datalog without function symbols is decidable: every evaluation terminates, every derivation can be traced to its supporting facts, and every reader reaches the same conclusions. The richer features that Prolog adds — and that mainstream programming languages add beyond Prolog — break one or more of these properties. RULES sacrifices expressive power to preserve reproducibility.

For inferences that exceed Datalog's expressiveness, the protocol does not provide a path. A future protocol layer, specified separately, could carry richer logic — but it would do so by accepting weaker guarantees about termination, determinism, or reproducibility. The choice to keep RULES within Datalog is the choice to keep the inferential layer publicly verifiable in the strongest sense.

5.10 What RULES enables

In isolation, RULES specifies a logical statement. The substantive consequences emerge when many rules accumulate and when readers apply them collectively to the chain-state.

A second rule, declared by anyone, can interpret the same facts differently. A third rule can build on derivations from the first two. The chain becomes a public knowledge base in which the body of available inferences grows over time, and in which any reader can inspect the genealogy of any particular conclusion: which rule derived it, from which facts, and which earlier rules contributed those facts in turn.

This is the layer above identity, the layer above relations, that this paper has been building toward. Identity persists on chain. Relations between identities persist on chain. And now, with RULES, the inferences over those relations are not private acts of reasoning but public functions of public inputs. What follows from what is on chain becomes itself a property of what is on chain.

6 Composition: the four-layer reasoning stack

6.1 What this chapter shows

Chapters 3, 4, and 5 specified KINSHIP, LINEAGE, and RULES as three independent protocols. Each can be used in isolation: KINSHIP without RULES still records who stands in kin to whom; LINEAGE without RULES still acknowledges intellectual debt; RULES without KINSHIP or LINEAGE can still reason over PACT identity facts alone.

What the three protocols produce together, however, is more than the sum of their parts. When inscribed under the same substrate and evaluated together, they form a four-layer reasoning stack in which each layer rests on the one below and each layer makes new operations possible. This chapter describes that stack as a whole, names what each layer contributes, and shows how the layers compose into the working demonstration visible on the Ordinalswall live page.

6.2 The four layers

The stack has four layers. Each layer is a category of fact or operation, and each rests on the layer beneath it.

Layer 1 — PACT identity facts. The bottom layer is PACT v1 itself: namespaces, worlds, avatars, and transitions. These are the entities about which everything else is said. An avatar exists because it has been inscribed on a sat under a namespace and a world; without this layer, none of the higher layers have anything to refer to. Layer 1 is the fundament of identity.

Layer 2 — Relations between identities. The second layer is KINSHIP and LINEAGE together. KINSHIP records symmetric relations within a community of contemporaries; LINEAGE records asymmetric acknowledgements toward predecessors. Both are protocols for relating identities to each other. Layer 2 takes Layer 1's identities as inputs and produces facts about how they stand in relation. Without Layer 1, Layer 2 has no entities to relate. Without Layer 2, the identities of Layer 1 stand in isolation, recorded but not connected.

Layer 3 — Rules over the lower layers. The third layer is RULES. A rule is a statement that says what may be inferred from the facts of Layer 1 and Layer 2. A rule does not itself add facts; it adds the possibility of derivation. Without Layer 1 and Layer 2, a rule has no body literals to satisfy. Without Layer 3, the relations of Layer 2 and the identities of Layer 1 sit on chain without inferential consequences.

Layer 4 — Derivations. The fourth and topmost layer is the set of facts that follow from applying Layer 3 rules to the facts of Layers 1 and 2. Derivations are not inscribed; they are computed by readers who apply the rules to the chain-state. Each derivation is traceable: it can be unfolded to the inscribed facts and rules that support it. Layer 4 is what public reasoning produces.

6.3 The dependency direction

The layers depend downward, not upward. Layer 4 depends on Layer 3, which depends on Layer 2, which depends on Layer 1. Removing a lower layer would invalidate everything above it; removing a higher layer leaves the lower layers intact.

This direction matters for understanding what each layer contributes. Layer 1 cannot reason; it can only assert identity. Layer 2 cannot infer; it can only declare relations. Layer 3 cannot stand

alone; it requires facts to operate over. Only Layer 4 produces conclusions, and only by means of Layers 1 through 3.

The direction also matters for the substrate. The lower layers are inscribed on Bitcoin; they are permanent, ordered, and verifiable. The top layer, by contrast, is computed at read-time. A reader generates Layer 4 by running the evaluation; the result is the same for any reader running the same evaluation, but the layer itself is not inscribed. This asymmetry is intentional. Inscribing every derivation would fill the chain with derivable consequences and obscure the distinction between what is asserted and what is inferred. Holding the top layer at read-time keeps the chain clean: the chain holds the premises; the reasoning produces the conclusions.

6.4 The composition in the worked example

The same example used in Chapter 5 illustrates the full stack:

Layer 1 (identity). Four avatars are inscribed: Wanderer-001, Forecaster-001, vonNeumann-001, and Auditor-001. Each is bound to a satoshi, each is constituted in a world under a namespace.

Layer 2 (relations). A KINSHIP inscription declares that Wanderer-001, Forecaster-001, and vonNeumann-001 stand in mutual kin. A LINEAGE inscription acknowledges Bitoshi Blockamoto and Bitmap Signal Theory as ontological precedent. Both are TRANSITION objects under PACT v1; each references avatars from Layer 1.

Layer 3 (rule). A RULES inscription declares that kinship between two avatars implies a derived `related` fact: `IF kinship(A,B) AND avatar(A) AND avatar(B) THEN related(A,B)`.

Layer 4 (derivations). A reader applying the rule to the facts of Layers 1 and 2 derives three new facts: `related(W, F)`, `related(F, V)`, `related(W, V)`. None is inscribed; all three are computed.

The composition is complete in this small example. The reader can start from any layer and work in either direction: upward, to see what conclusions follow; downward, to see what facts support a given conclusion. The genealogy of every derivation is traceable to its inscribed premises.

6.5 What composition makes possible

The four-layer stack makes a specific class of operations possible that none of its individual protocols supports alone.

Cross-layer queries. A reader can ask: *which avatars stand in derived relation to Wanderer-001?* The answer crosses Layers 1 (Wanderer is an avatar), 2 (kinship facts mention Wanderer), 3 (the rule transforms kinship into relation), and 4 (the derivations are computed). The query spans the stack; the answer is a function of the whole.

Provenance unfolding. For any derived fact in Layer 4, a reader can unfold the derivation: which rule produced it, which facts of Layer 2 satisfied the rule's body, which avatars of Layer 1 are referenced. The unfolding always reaches inscribed premises; nothing is left to opinion.

Compositional accumulation. New rules added at Layer 3 produce new derivations at Layer 4 without changing Layers 1 or 2. A second rule that interprets LINEAGE transitively, for example, produces *intellectual descent* derivations alongside the existing *related* derivations. The chain holds the premises stably; the inferential layer grows.

Independent re-derivation. Two readers, given the same chain-state and the same set of rules, derive the same Layer 4. Disagreement is detectable: if two implementations produce different derivations from the same inputs, at least one is incorrect, and the discrepancy can be traced to the specific step where the implementations diverge.

6.6 The visible expression of the stack

The Ordinalswall live page (Chapter 10) presents the four-layer stack as an interactive visualisation. The page shows derivations at the top, rules below them, relations below those, and PACT identity facts at the foundation. A reader who clicks on any element sees its connections to the layers above and below. The visualisation is not a separate construct; it is the stack made navigable.

That such a presentation is possible at all is a property of the composition. A system in which identity, relations, rules, and derivations were each held by different operators in different formats would not admit a unified view. The four-layer stack on Bitcoin admits one because all four layers share the same substrate: the lower three are inscribed; the top is computed; all four are accessible from chain-state alone.

6.7 Compositionality and minimality

The stack does not require additional protocols beyond those specified in Chapters 3, 4, and 5. The four layers emerge from the combination of three protocols on the PACT v1 envelope. No new envelope type is introduced; no new validity machinery is added; no new substrate is required.

This minimality is deliberate. A more elaborate stack — one that introduced separate protocols for each kind of relation, each kind of inference, each kind of derivation — would multiply the surface area without adding capacity. The present specification favours composability: a small set of primitives that combine into a rich structure, rather than a large set of specialised protocols each addressing a narrow case.

The minimality also defends portability. A future implementation that wants to support additional protocols can do so by adding new TRANSITION actions under PACT v1, without disturbing the existing four-layer composition. The substrate remains stable; the inferential layer grows by accretion.

6.8 What the stack does not produce

The stack is a reasoning architecture, not a complete knowledge system. Several things it does not produce should be noted.

The stack does not produce truth. It produces conclusions that follow validly from inscribed premises. Whether the premises are themselves true — whether a KINSHIP correctly describes a real relation, whether a LINEAGE correctly identifies a genuine source, whether a rule correctly captures the meaning of a relation — is a question the stack does not answer. Truth-evaluation is the work of readers, not of the stack.

The stack does not produce consensus. Different rules inscribed by different parties may interpret the same facts in different ways. A reader applying one rule will derive one set of conclusions; a reader applying another rule will derive another. The stack records the disagreement; it does not resolve it. Resolution is a social process that takes place above the stack, not within it.

The stack does not produce action. A derivation that says `related(A, B)` is a fact about what follows from the chain. It is not an instruction. Whether anyone acts on the derivation —

whether A invites B to collaborate, whether a reader cites both parties, whether an institution recognises the relation — is a matter for those parties and readers, not for the stack.

What the stack produces is a substrate on which truth, consensus, and action can be reasoned about, recorded, and later inspected. It does not displace human or institutional work; it provides a public, durable, verifiable layer beneath that work.

7 Verification

7.1 What verification means in this layer

Verification at the PACT v1 layer concerns whether a given inscription is structurally valid, whether its sat-bound identity is continuous, and whether its Merkle commitments match the bytes they claim to bind. PACT v1 [2] specifies these checks in Sections 6 and 8 of that document. This paper does not repeat that material; it builds on it.

Verification at the layer specified in this paper concerns a different question: whether a derivation produced by applying a rule to chain-state actually follows from that chain-state. The question is logical rather than cryptographic. A rule applied incorrectly produces an invalid derivation even if every underlying inscription is cryptographically sound. The purpose of this chapter is to specify what makes a derivation verifiable, what mechanisms a reader uses to verify it, and what guarantees that verification provides.

7.2 What is verified

Three classes of object can be verified at this layer:

Inscribed facts. A KINSHIP, LINEAGE, or RULES inscription is a TRANSITION under PACT v1. Its verification is the PACT v1 verification: envelope structure, sat-bound continuity, byte-exact reproduction. The protocols specified here add no new cryptographic requirements; their inscriptions are verified as PACT v1 inscriptions. What this paper adds is the validity rules for the data fields, specified in Chapters 3, 4, and 5.

Rule applications. A rule is verified when a reader confirms that the rule's expression is syntactically valid, that its variables are safe, and that it would terminate under semi-naive evaluation. This is a static check on the rule itself, independent of the facts it might be applied to.

Derivations. A derivation is verified when a reader confirms that the rule's body literals can be satisfied against the inscribed facts, and that the substitutions producing the head literal are valid. This is a dynamic check that depends on both the rule and the current chain-state.

The first class is verified by PACT v1 mechanisms. The second and third classes are the focus of the rest of this chapter.

7.3 Verifying a single rule

A rule is statically verifiable from its inscription alone, without reference to any other inscriptions. The verification checks the following:

1. The `expression` field parses as a syntactically valid rule under the v1 grammar specified in Chapter 5.
2. Every variable appearing in the head also appears in at least one positive body literal (the safety condition).
3. Negation, where used, is consistent with stratification across the full set of rules being applied together.
4. The `rule_id` is unique within its declaring scope at the moment of inscription.

A rule that fails any of these checks is invalid. An invalid rule produces no derivations regardless of what facts exist on chain. This is the same property as in Datalog generally: the inferential layer cannot derive consequences from a malformed rule.

7.4 Verifying a derivation

A derivation is dynamically verifiable. To verify that a particular derived fact follows from a chain-state, a reader performs the following procedure:

1. Identify the rule whose head matches the structure of the derived fact.
2. Find a binding of variables to constants such that substituting the binding into the head yields the derived fact.
3. Substitute the same binding into each body literal of the rule.
4. Confirm that each substituted body literal is itself a fact in the chain-state, either as an inscribed fact or as a previously verified derivation.
5. If every body literal is satisfied, the derivation is valid. If any body literal cannot be satisfied, the derivation does not follow from the chain-state.

The procedure is mechanical and reproducible. Two readers running the same procedure on the same chain-state with the same rule reach the same conclusion about whether a given derivation is valid.

7.5 Verifying the full derivation set

A reader who wants to verify the complete set of derivations that follow from a chain-state runs semi-naive evaluation as specified in Chapter 5, Section 5.6. The evaluation produces the fixpoint: the set of all facts derivable from the rules and the inscribed facts.

The evaluation is verified by reproducibility. Two implementations of the evaluation, given the same inputs, must produce the same fixpoint. A discrepancy between implementations indicates an error in at least one of them and can be traced to the specific rule application where they diverge.

The fixpoint itself is the canonical answer to the question *what does the chain-state imply, given these rules?* It is not opinion. It is a function of the inputs, and any reader who runs the function on the same inputs reaches the same output.

7.6 Worked verification: the first derivation

The first RULES inscription on chain (Section 5.7) declares:

```
IF kinship(A,B) AND avatar(A) AND avatar(B) THEN related(A,B)
```

To verify the derivation `related(W, F)` where W is Wanderer-001 and F is Forecaster-001, a reader proceeds as follows:

Step 1. Identify the rule. The rule `kinship-implies-relation` has head `related(A,B)`, which matches `related(W,F)` under the binding `A := W, B := F`.

Step 2. Substitute the binding into the body. The body becomes:

```
kinship(W, F) AND avatar(W) AND avatar(F)
```

Step 3. Verify each substituted body literal against the chain-state.

- `kinship(W, F)`: the KINSHIP inscription on chain declares mutual kin among Wanderer-001, Forecaster-001, and vonNeumann-001. The kinship between W and F is one of the implied pairwise relations from this declaration. The literal is satisfied.
- `avatar(W)`: Wanderer-001 is constituted as an avatar in pact-semantics/open-field. The literal is satisfied.
- `avatar(F)`: Forecaster-001 is constituted as an avatar in pact-semantics/ordinals. The literal is satisfied.

Step 4. All three body literals are satisfied. The derivation `related(W, F)` is therefore valid: it follows from the rule and the chain-state.

Step 5. The verification is complete. The reader has confirmed, without trusting any external service, that the derivation follows from inscribed premises by valid application of an inscribed rule.

The same procedure verifies `related(F, V)` and `related(W, V)`. All three derivations are valid; none is on chain; all three are verifiable by any reader.

7.7 What verification at this layer does not require

Verification at this layer requires no external service. It requires no indexer, no oracle, no mediating party. The inputs are the inscribed facts (verifiable by PACT v1 mechanisms) and the inscribed rules (verifiable from their own inscriptions). The output is a set of derivations, computed deterministically from the inputs.

In particular, verification does not require:

- agreement with any other reader's choice of which rules to apply,
- endorsement by the authors of the rules being applied,
- recognition by any institution or community,
- connection to any network beyond the one needed to read Bitcoin and the inscriptions on it.

A reader running the verification on a fully local Bitcoin node, with all relevant inscriptions cached locally, reaches the same conclusions as a reader running the verification against a public node. The verification is a function of the chain-state, and the chain-state is the same for any reader who reads the same blocks.

7.8 What verification provides

Verification at this layer provides one specific guarantee: that the conclusions presented as derivations are functions of inscribed premises and inscribed rules. It does not guarantee that the premises are true, that the rules are sensible, or that the conclusions are useful. What it guarantees is that the conclusions cannot be tampered with, silently changed, or rewritten by anyone. They are what they are, computed from what is on chain.

This is a narrower guarantee than is sometimes claimed for on-chain reasoning systems. It is also a more honest one. The substrate gives durability and reproducibility; the protocols give structure and inferential consequence; the verification confirms the chain from premises

to conclusions. None of these gives truth. Truth remains the work of those who choose which premises to inscribe and which rules to apply.

What the layer does is make that work public. A reader who disagrees with a conclusion can trace the disagreement to its source: an inscribed fact whose accuracy is contested, a rule whose interpretation is disputed, a binding whose validity is questioned. The disagreement becomes specific and locatable, rather than diffuse and unattributable. That specificity is itself a contribution to public reasoning.

7.9 Verification beyond a single chain-state

The chain-state changes over time. Each new block confirms new inscriptions, which add new facts and possibly new rules. A derivation that holds at block height N may or may not hold at block height $N + 1$; the rule may still be valid, but the inscribed facts that previously satisfied its body may have been augmented or contradicted by later inscriptions.

Verification across time therefore requires specifying the chain-state at which the verification is performed. A derivation is not absolute; it is parameterised by the block height at which the chain-state is read.

For most purposes, the chain-state at the current chain tip is the relevant one. Where historical reasoning is needed — for example, to establish what could have been derived at a particular moment in the past — the reader specifies the block height and reproduces the chain-state up to that point. The same verification procedure applies; only the input state changes.

This temporal parameterisation is implicit in any system that reasons over evolving facts. The chain makes it explicit. A derivation always has a date, expressible as a block height, and the date is part of what makes the derivation reproducible.

8 Traditions

8.1 Why this chapter is here

The protocols specified in this paper are not invented. KINSHIP formalises a relation that anthropology and sociology have described for centuries. LINEAGE formalises an acknowledgement practice older than printing. RULES is Datalog, a logic programming dialect specified in 1977 and refined for nearly five decades. Even the underlying ontology — namespace, world, avatar, transition — is a recognised pattern across software engineering, knowledge representation, and biological taxonomy (Section 3 of [2]).

What this paper contributes is not new logic. It is a new substrate. The substrate brings older traditions into a setting where their results gain durability they have not previously enjoyed. This chapter places the three protocols in the intellectual lineage they descend from, makes the descent explicit, and identifies what the substrate adds to each tradition.

8.2 Logic programming

Logic programming as a discipline began with Prolog. Alain Colmerauer and Philippe Roussel developed the first implementation in Marseille in 1972, building on Robert Kowalski's theoretical work on resolution as a programming mechanism. Prolog established that a program could be expressed declaratively — as a set of facts and rules — rather than imperatively as a sequence of instructions. A query against the program is answered by searching for substitutions that satisfy the rules.

Datalog emerged in 1977 from work by Hervé Gallaire, Jack Minker, and David Maier, among others. Where Prolog allowed arbitrary recursion, function symbols, and the cut operator, Datalog deliberately restricted itself: no function symbols, no cut, recursion only in stratified form. The restrictions made Datalog decidable: every query terminates, every evaluation reaches a unique fixpoint, every reasoner produces the same answer from the same inputs.

These restrictions are exactly what makes Datalog suitable for a substrate where reproducibility is the foundational guarantee. RULES is Datalog because Datalog produces conclusions that any reader can reproduce; the additional expressive power of Prolog would compromise that property. The choice is not a tradeoff in capability so much as an alignment between language and substrate. Datalog and Bitcoin share a discipline of strict determinism; placing one on the other is a natural fit.

Modern Datalog has not stood still. Soufflé, an open-source Datalog engine developed primarily for static program analysis, demonstrates that the language scales to billions of facts on commodity hardware. RDFox, a commercial Datalog engine, integrates with Semantic Web standards. ddlog (Differential Datalog) supports incremental computation so that small changes to facts produce small changes to derivations rather than full re-evaluation. GPU-accelerated Datalog systems such as GDlog have demonstrated that evaluation can run on graphics hardware for further performance gains.

The implementations vary, but the formal properties of the language are stable. A rule inscribed in PACT-RULES v1 today will be evaluated correctly by a reader using any of these engines, provided the engine implements standard Datalog semantics. The substrate is durable; the evaluation techniques can evolve; the conclusions remain reproducible across both axes.

8.3 The Semantic Web

The Semantic Web tradition is younger than logic programming in some respects but draws on similar foundations. Tim Berners-Lee proposed it in 1998 as an extension of the World Wide

Web in which information would have explicit structure and machines could reason over that structure. The technical artifacts of the tradition were standardised in stages: RDF (Resource Description Framework) in 1999, OWL (Web Ontology Language) in 2004, SPARQL (the query language) in 2008.

RDF expresses knowledge as triples: subject-predicate-object statements that together form a graph. OWL adds vocabulary for describing classes, properties, and constraints. SPARQL allows querying the resulting knowledge graphs. SWRL (Semantic Web Rule Language) and N3Logic are extensions that add rule-based reasoning on top of RDF and OWL — the inferential layer the present paper places on PACT.

The intellectual foundation predates the Web. Ross Quillian's semantic networks (1968) provided the first formal model for knowledge representation as graphs of concepts and relations. KL-ONE (1985) refined the model with description logics that distinguished between defining a concept and stating a fact about an instance. The W3C standards from the 1990s onwards inherit this tradition and extend it with web-scale addressability through URIs.

What the Semantic Web has always assumed is a substrate sufficient for its claims to remain reachable. The assumption has not always held. URIs rot when domains expire. SPARQL endpoints go offline when the institutions hosting them lose interest or funding. Triples persist only as long as their hosts persist. The reasoning engines built on these foundations have been mathematically sound but practically fragile.

The substrate this paper specifies removes that fragility for the inscribed claims. A LINEAGE acknowledgement on Bitcoin does not need a host to remain reachable; the chain is the host. A KINSHIP declaration does not depend on any registrar's continued existence. The same RDF-shaped claims that the Semantic Web has expressed for decades can now be made on a substrate that no organisation maintains and no jurisdiction controls.

8.4 Where the traditions converge

Logic programming and the Semantic Web converged in the early 2000s with the development of rule languages designed to operate over RDF data. SWRL, RIF (Rule Interchange Format), and N3Logic each attempted to specify a rule layer that could be evaluated against Semantic Web triples. The approaches differed in detail, but the architectural pattern was shared: facts as triples, rules as if-then statements, an inference engine that produces derived triples from the combination.

The pattern is recognisable in the four-layer stack specified in this paper. PACT v1 facts function as the triple-shaped ground knowledge. KINSHIP and LINEAGE add particular relation types. RULES expresses inference patterns over the combination. Derivations are the equivalent of the new triples that an SWRL or N3Logic engine would produce.

What distinguishes the present specification from SWRL and N3Logic is not the inferential pattern, which is nearly identical, but the substrate. The Semantic Web rule languages were specified for evaluation on web servers or in-memory engines; the conclusions they produce live as long as those servers do. The protocols specified here place the same kind of reasoning on a substrate that, for the inscribed premises and rules, makes the conclusions reproducible indefinitely.

8.5 What this substrate adds to each tradition

To logic programming, the substrate offers what was always missing: a knowledge base whose facts have the same lifespan as the reasoning over them. A Datalog rule has always been a function from facts to derivations. The function has always been reproducible in principle. What

was not reproducible across decades was the input: the facts lived in a database whose schema or contents could change, in a process whose state vanished when the process ended, in a file whose location changed when the project moved. PACT brings the inputs onto a substrate where they remain addressable and unmodifiable.

To the Semantic Web, the substrate offers what URIs aspired to but never fully delivered: identifiers that do not rot. A subject-predicate-object triple inscribed under PACT rules is reachable through ordinary tools, persistent without a hosting fee, and resolvable to the same content for any reader at any time. The Semantic Web's vision of structured machine-readable knowledge does not require a web server to serve the structure; it can live where the chain lives.

To both traditions together, the substrate offers something neither alone produces: a public space for reasoning that is not owned. A SWRL inference engine running on a corporate server produces conclusions that the corporation can withdraw or modify. A Datalog evaluation running in a research project produces conclusions that vanish when the project ends. The protocols specified here keep both the premises and the inferential procedure available to any reader, on a substrate no party owns. The reasoning becomes a public function of public inputs.

8.6 Where this paper fits

This paper does not propose a new logic. It does not propose a new ontology language. It does not propose a new inference algorithm. What it proposes is a small set of protocols that allow existing logic and existing ontology practices to operate on Bitcoin's witness layer through Ordinals.

The contribution is therefore architectural rather than foundational. The foundations are inherited: from Quillian and KL-ONE, from Colmerauer and Kowalski, from Gallaire and Minker, from Berners-Lee and the W3C working groups. The protocols specified here arrange these inheritances on a new substrate and demonstrate that the arrangement works.

A reader from the logic programming community will recognise RULES as restricted Datalog with conventional safety conditions. A reader from the Semantic Web community will recognise the four-layer stack as RDF-like ground knowledge, explicit relation types, SWRL-shaped rules, and derived triples. A reader from neither community will recognise the combination as a publicly readable knowledge base whose contents persist independently of any institution.

The recognition is intentional. This paper succeeds when practitioners from existing traditions see the substrate as a place where their work can persist, not as a competing formalism that requires them to abandon what they already know. The protocols are minimal precisely so that as much existing knowledge representation work as possible can be ported onto them with minimal translation.

8.7 Acknowledged debts

The intellectual debts named in this chapter — to Quillian, KL-ONE, Prolog, Datalog, Soufflé, RDFox, RDF, OWL, SWRL, N3Logic, and the broader Semantic Web community — are acknowledged here as a textual matter, but they are also candidates for inscribed LINEAGE acknowledgements under the protocol specified in Chapter 4.

A LINEAGE inscription acknowledging Datalog as the source of PACT-RULES, or acknowledging the Semantic Web as the source of the layered ontology pattern, would record on chain what this paper records in prose. Such inscriptions are not required for the protocols to function; they are appropriate when the inscriber wishes to formalise the acknowledgement in a way that survives the document.

The choice of which acknowledgements to inscribe and which to leave in prose is left to the inscriber. What the chain gains by formalisation is durability; what prose retains is the freedom to discuss debts in nuance and context that an inscription cannot easily carry. Both have their place.

9 Public reasoning

9.1 What public reasoning is

Earlier chapters have used the phrase *public reasoning* without definition. The term has been deployed as if its meaning were obvious from context, but it carries enough weight in this paper's argument to warrant explicit treatment. This chapter defines public reasoning, distinguishes it from related notions, and shows how the three protocols on PACT together produce it.

The shortest formulation is this: public reasoning is reasoning whose derivation is itself publicly testable on a substrate no instance can retract. Three properties together constitute the definition. Without any one of them, what remains is a different and weaker thing.

9.2 The three properties

The properties are permanence, open authorship, and deterministic reproducibility. Each is necessary; none alone is sufficient.

Permanence. The premises and the rules from which conclusions are derived must remain accessible across time. A reasoning system whose inputs vanish when the host server goes offline produces conclusions that cannot be re-verified later. The substrate must guarantee that what was inscribed remains inscribable in the same form, by anyone, indefinitely. Bitcoin provides this property; very few other substrates do.

Open authorship. Anyone able to participate at the substrate level must be able to inscribe facts, relations, and rules without asking permission. A reasoning system that admits inputs only through a curatorial gate produces conclusions whose validity depends on the curator's judgement. That dependency converts public reasoning into private reasoning made visible. PACT v1, KINSHIP, LINEAGE, and RULES require no permission for inscription beyond control of a satoshi.

Deterministic reproducibility. Two readers, applying the same rules to the same chain-state, must reach identical conclusions. A reasoning system whose evaluation depends on heuristics, randomness, or implementation choices produces conclusions that cannot be reliably re-derived. The restricted Datalog used by RULES (Chapter 5) guarantees this property: every evaluation terminates, every fixpoint is unique, every implementation that follows the specification reaches the same answer.

9.3 Why all three are necessary

A reasoning system that has only permanence — facts inscribed on a durable substrate, but evaluation by a private authority — produces conclusions that survive in time but remain unverifiable. The reader must trust the authority's report of what follows from the inscribed facts.

A reasoning system that has only open authorship — anyone can inscribe, but on an ephemeral substrate — produces conclusions whose inputs vanish before they can be re-verified. What was once derivable becomes undecidable as the premises disappear.

A reasoning system that has only deterministic reproducibility — formal logic correctly implemented, but on private data controlled by a single party — produces conclusions only that party can re-derive. The mathematics is sound; the public character is missing.

The three properties reinforce one another in a way that no combination of two can replicate. Permanence without open authorship is a museum: things are preserved, but only the curator decides what enters. Open authorship without permanence is a marketplace: many can speak, but the speech fades. Deterministic reproducibility without the other two is a closed system: rigorously consistent, privately auditable, publicly inaccessible.

The combination of all three is what this paper calls public reasoning. It is, as far as the author can establish, a new combination. The traditions surveyed in Chapter 8 each contribute one or two of the properties; none combines all three, and the substrate that makes the combination possible — Bitcoin's witness layer made addressable through Ordinals — is itself recent enough that the combination has not yet been articulated as a class.

9.4 What public reasoning is not

Several adjacent notions resemble public reasoning closely enough that distinguishing them is useful.

Open data. A repository of facts that anyone can read is open data, not public reasoning. The reasoning is the missing ingredient: a reader of open data must construct their own inferences, with no shared substrate for the inferential layer itself. Public reasoning includes the inferential layer in what is shared.

Open source. Software that anyone can read, modify, and redistribute is open source, not public reasoning. Open source covers the rules (the code) but typically not the facts on which the code operates, and the conclusions the code produces depend on the runtime state of the system running it. Public reasoning, by contrast, places facts, rules, and reproducible conclusions all on the same substrate.

Distributed consensus. A blockchain achieves consensus among many parties about the state of a ledger. This is not the same as public reasoning. Consensus establishes what is true at the data layer; public reasoning establishes what follows from what is true at that layer. Public reasoning rides on top of distributed consensus about the underlying facts; it adds an inferential layer that consensus alone does not provide.

Provenance tracking. Systems that record the chain of custody of a piece of information — when it was created, who modified it, when it was published — track provenance, not reasoning. A provenance record says what happened to a fact; a public reasoning record says what follows from a fact. Provenance is descriptive of the past; public reasoning is generative of new conclusions from present inputs.

9.5 What public reasoning enables

When reasoning becomes public in this strong sense, several things change in how knowledge can be organised.

Disagreement becomes locatable. Two readers who reach different conclusions from the same chain-state can trace the disagreement to a specific step: a fact whose existence is contested, a rule whose interpretation is disputed, a binding whose validity is questioned. The disagreement is no longer a vague divergence; it is a specific claim at a specific layer.

Genealogy becomes inspectable. For any conclusion, the reasoning that produced it can be unfolded back to inscribed premises. The unfolding is mechanical and reproducible; nothing depends on the goodwill of the original reasoner. A reader who wants to know why a particular conclusion holds can trace the entire derivation without consulting the conclusion's author.

Accumulation becomes possible. New rules and new facts inscribed on the chain extend the body of derivable conclusions without invalidating the old. A second inscriber's contribution does not require the first inscriber's permission. The chain becomes a knowledge substrate that grows by accretion, with each contribution visible alongside what came before.

Critique becomes structural. A critic who disagrees with a conclusion can inscribe a counter-rule, a contradicting fact, or a clarifying derivation. The critique is not a reply that fades; it stands on the same chain, in the same form, equally durable. Public reasoning thereby admits structured disagreement as a first-class feature rather than a complication to be managed.

9.6 The relationship to truth

Public reasoning does not produce truth. This bears repeating because the strength of the substrate, combined with the formal soundness of the inferential layer, can create the impression that what is derived must be correct. It must not.

What public reasoning produces is conclusions that follow validly from inscribed premises, by inscribed rules, according to a deterministic evaluation. Whether the premises are true — whether a KINSHIP inscription correctly describes a real relation, whether a LINEAGE acknowledgement correctly identifies a genuine source, whether the rule sensibly captures the meaning of those relations — is a separate question. The substrate guarantees the evaluation; it cannot guarantee the inputs.

This is a familiar limitation. Mathematical proof systems have always faced it: a valid proof from false premises yields a false conclusion. Logic programming has faced it since Prolog. Database query engines face it daily. The difference is that public reasoning makes the limitation visible. A reader of derived conclusions on the chain can inspect the premises and form their own judgement about their truth, rather than accepting or rejecting the conclusions on faith. The substrate does not displace this work; it makes the work possible by ensuring that the premises are accessible to be inspected.

9.7 The relationship to authority

Public reasoning does not produce authority. A conclusion that has been validly derived from inscribed premises is not, for that reason alone, an authoritative statement. Authority — the social recognition that a particular conclusion should be acted upon, that a particular reasoner should be trusted, that a particular institution should adjudicate — operates at a different level than the chain.

What the substrate offers to authority is a record. An institution that wishes to claim authority over a domain can inscribe its claims and its derivations on chain, where they remain available for inspection long after the institution itself may have changed character or ceased to exist. The institution does not gain authority by inscribing; it gains a public record of what it has claimed.

Conversely, an institution can lose authority if its inscribed claims are shown — by inscribed counter-claims, or by derivations that contradict its own — to be inconsistent or unsupported. The chain does not enforce the loss; the chain provides the material for those who would adjudicate.

In this sense, public reasoning shifts the relationship between authority and evidence. Where authority traditionally withholds the evidence on which its judgements rest — and asks to be trusted on the basis of that withheld evidence — public reasoning places the evidence in the open and lets authority follow rather than precede the public examination of what that evidence supports.

9.8 Why this matters now

The substrate that makes public reasoning possible has only recently become practical. Bitcoin has existed since 2009; Ordinals has existed since early 2023; PACT has existed since April 2026; the three protocols specified in this paper have existed for less than a month at the time of writing. The combination is new. Its consequences have not yet been worked out by those who might use it.

This paper offers the protocols and the architecture but not the use cases. What public reasoning will produce, when applied at scale by communities of practice that have so far operated under different substrates, is a question neither this paper nor any current author can fully anticipate. What can be anticipated is that the combination — permanence, open authorship, deterministic reproducibility — has been long sought and rarely encountered. Where it appears, knowledge work that has historically depended on institutional gatekeeping or ephemeral hosting can take a different shape.

This paper is offered as a specification of the lower layers on which that different shape might be built. The work of building it is for others to do, on a substrate that does not require permission to begin.

10 A working demonstration: the Ordinalswall live page

10.1 What this chapter shows

The previous chapters specified KINSHIP, LINEAGE, and RULES, described how they compose into a four-layer reasoning stack, and defined what public reasoning means as a property of that stack. This chapter describes a working demonstration of the specification: the live page on ordinalswall.com that presents the current chain-state of the four layers as an interactive visualisation.

The demonstration is small. It does not exhaust the possibilities of what the protocols make possible. What it does is show that the specification is operable: that the inscriptions described in earlier chapters exist on chain, that they form the relations and rules described, and that a reader can navigate the resulting structure without trusting any service.

10.2 The page

The live page is reachable at `https://ordinalswall.com/live.html`. It is a single HTML page with inline JavaScript for interactivity. The page presents the four layers of the reasoning stack as labelled groups of clickable elements:

- Derivations at the top, labelled *what follows*.
- Rules below derivations, labelled *what may be inferred*.
- Relations below rules, labelled *kinship and lineage*.
- PACT identity facts at the foundation, labelled *avatars on chain*.

A reader can click any element to see its connections to other elements: which avatars participate in a kinship, which rules consume a relation, which derivations follow from a rule. A detail panel below the visualisation describes the selected element and identifies the inscription that established it on chain.

10.3 Inscribed elements

The page presents the following inscribed elements at the time of writing:

Avatars (Layer 1). Four avatars, each constituted under PACT v1 on its own carrier sat. Wanderer-001 in pact-semantics/open-field on sat 1168224522821204. Forecaster-001 in pact-semantics/ordinals. vonNeumann-001 in pact-semantics/mathematics. Auditor-001 in agents/openclaw. Each carries an inscription identifier visible in the detail panel.

Relations (Layer 2). Two inscribed relations. A KINSHIP declaration linking Wanderer-001, Forecaster-001, and vonNeumann-001 in mutual kin (inscription 93e3474b69a0ab2ef65af0458f805d1e0985d3e5c4db
A LINEAGE acknowledgement of Bitoshi Blockamoto and Bitmap Signal Theory as ontological precedent to PACT (inscription dafd8748c37b914866a28a23a30be645693382c1fccd5bcc6dd750b59f6843bfi0).

Rule (Layer 3). One inscribed rule: `kinship-implies-relation`, declaring that `IF kinship(A,B) AND avatar(A) AND avatar(B) THEN related(A,B)`. The rule is inscription 3e72086f88d791f5122e1cb0bab0727 inscribed on 4 May 2026.

Derivations (Layer 4). Three derivations, computed at read-time by applying the rule to the chain-state. The page presents them as `related(W, F)`, `related(F, V)`, and `related(W, V)`.

None is on chain. All three follow deterministically from the inscribed facts and the inscribed rule.

10.4 What the visualisation makes navigable

The four-layer stack described in Chapter 6 is a structural claim. The visualisation makes the structure navigable. A reader who clicks on Wanderer-001 sees that Wanderer is part of the KINSHIP, that the rule consumes the KINSHIP, and that two of the three derivations involve Wanderer. A reader who clicks on the rule sees the KINSHIP it consumes and the three derivations it produces. A reader who clicks on a derivation sees the rule that produced it, the KINSHIP that satisfied the rule's body, and the avatars referenced.

What the visualisation does not do is replace the chain. The inscriptions remain authoritative; the visualisation is a view on them. A reader who distrusts the visualisation can follow each inscription identifier to a public ord-node or ordinals.com and verify the contents directly. The visualisation offers convenience; the chain offers authority.

The page also does not derive the conclusions itself in any authoritative sense. The three `related` derivations shown on the page are the same three that any reader running semi-naive evaluation against the same chain-state would produce. The page does not claim to be the unique source of the derivations; it claims only to display them.

10.5 What Auditor-001 demonstrates

The fourth avatar on the page, Auditor-001, has no kinships. The rule applied to the current chain-state therefore produces no derivations involving Auditor. When a reader clicks on Auditor, the highlight shows that no other elements are connected to it: it stands alone among the avatars, present on chain but not currently the subject of any relation or any derivation.

This is a feature, not an absence. The wall does not require activity. An avatar may exist, documented, with full room to enter into relations later or not at all. Silence is a valid state. What matters is that the silence is itself visible: a reader can see that Auditor exists and that no relations currently involve it. That visibility is part of what the demonstration provides. A system that hid silent identities would be less honest about what is on chain.

10.6 Verification through the page

The page is also a starting point for independent verification. Each inscribed element exposes its inscription identifier, which links to ordinals.com or any equivalent explorer. A reader who wants to confirm that the KINSHIP declaration says what the page reports can fetch the raw inscription content and read it directly. The same applies to the rule and to each avatar's constitution.

For derivations, the page's authority extends only to identifying the rule and the satisfying premises. The derivation itself follows by Datalog semantics; a reader who implements semi-naive evaluation independently and applies the same rule to the same chain-state derives the same three `related` facts. The page invites this independent evaluation rather than discouraging it.

The verification flow described in Chapter 7 is therefore operationally available through the page. A reader who wishes to verify any element can do so without trusting the page beyond its role as a convenience layer over the chain.

10.7 What the demonstration does not yet show

The page presents what is on chain at the time of writing. What it does not yet show is the breadth of what the specification permits. Several extensions discussed in Chapter 12 would,

when inscribed, appear on the page automatically: a transitive rule on related, a peer rule that distinguishes same-namespace relations, a meta-LINEAGE rule that traces intellectual descent across generations, a contradiction-detecting rule.

None of these are present on the page today because none has been inscribed. The page reflects the chain; the chain contains the elements specified in this paper and described in this chapter; everything else is a possibility for the future.

This honesty is itself a feature of the demonstration. A working example with one rule and three derivations is more trustworthy than a polished interface that promises features not yet implemented. The page shows what works. What does not yet work is acknowledged as not yet working. Readers can form their own judgement about whether the demonstrated operability suggests that further inscriptions would behave as the specification predicts.

10.8 Why this demonstration is sufficient

A specification document benefits from a working demonstration but does not require an exhaustive one. The purpose of the demonstration is to confirm that the specification is operable in principle, not to enumerate every application that might be built on it. The Ordinalswall live page does the former: it shows that PACT v1 with the three protocols specified here produces a verifiable four-layer reasoning stack on Bitcoin mainnet, today, without any service mediating the result.

What the demonstration shows is small enough to fit on one HTML page and large enough to demonstrate the architecture in full. Avatars, relations, rules, and derivations all appear; clicking traces the structure; the chain validates the inscribed elements; semi-naive evaluation produces the derivations. The complete vertical slice of the four-layer stack is operational.

Further demonstration — more inscribers, more namespaces, more rules — will follow if the protocols are taken up by others. The present specification does not require it. What the present specification requires is that the architecture work for one inscriber. That has been established.

11 Boundaries

11.1 Why a chapter on what the stack cannot do

A specification that describes only what its subject can do is incomplete. The boundaries — what the system cannot do, should not attempt to do, and should not be expected to do — are part of what the specification is. Without explicit boundaries, readers project capabilities onto the system that it does not possess, and the resulting disappointment or misuse is foreseeable.

This chapter states the boundaries of the four-layer reasoning stack specified in this paper. Some of the boundaries follow from Bitcoin itself; others from the restricted Datalog used by RULES; others from the declarative nature of all three protocols. None of these boundaries is a deficit in the sense that closing it would make the stack better. Each is the necessary cost of one of the three properties — permanence, open authorship, deterministic reproducibility — that constitute public reasoning.

11.2 The stack does not move value

Bitcoin's witness layer carries data. The value layer of Bitcoin moves satoshis. PACT operates on the witness layer. The stack specified in this paper therefore records claims and derives consequences but does not transfer satoshis, issue payments, or trigger financial transactions.

A rule may declare a condition under which a payment ought to occur. It cannot cause the payment. The actual movement of value is an ordinary Bitcoin transaction that some person or process must initiate. The stack can record that the payment occurred, after the fact, by means of a further TRANSITION inscription that references the transaction id; but the inscription does not cause the payment, only documents it.

This boundary is structural. Bitcoin's design separates witness data from spendable outputs precisely so that data inscription does not interfere with monetary semantics. PACT respects that separation. Systems that wish to combine inscribed claims with autonomous value movement must operate at a different layer or on a different substrate. Ethereum and other smart-contract platforms occupy that space; this paper does not.

11.3 The stack does not enforce execution

A rule that derives `related(A, B)` produces a fact about what follows from inscribed premises. It does not produce action. Whether anyone treats A and B as related, whether either of them learns of the derivation, whether the derivation matters in any practical sense — none of this is within the stack's reach.

The same holds for KINSHIP and LINEAGE. A KINSHIP declaration records that the inscriber claims kinship among the named avatars. It does not enforce reciprocity, does not notify the named parties, and does not require them to behave as if the kinship existed. A LINEAGE acknowledgement records the inscriber's recognition of a source. It does not require the source to acknowledge in return, does not generate any obligation on the source, and does not even require the source to know that the acknowledgement exists.

This boundary is also structural. The stack is declarative: it records statements, not commands. Translating declarations into actions is the work of those who read the chain, who decide whether the declarations apply to them, and who choose whether to act. The stack provides material for action; it does not perform action.

11.4 The stack is not Turing-complete

The RULES protocol uses restricted Datalog (Chapter 5). Datalog without function symbols is decidable: every evaluation terminates, every fixpoint is reached in finite time, every implementation produces the same answer. These properties make the inferential layer publicly reproducible.

The cost of these properties is expressive power. RULES cannot express arbitrary recursion with non-stratified negation. RULES cannot express functions that build new terms from existing terms. RULES cannot express arithmetic over unbounded domains, optimisation problems, or imperative state changes. A computation that requires any of these features cannot be expressed as a RULES inscription.

This boundary is non-negotiable for any extension to RULES that wishes to preserve public reproducibility. The literature on combining logic programming with computation shows that adding Turing-completeness invariably sacrifices either termination, determinism, or both. The choice to restrict RULES to Datalog is the choice to keep the inferential layer publicly verifiable in the strongest sense, even at the cost of some inferences that richer systems could express.

For computations that exceed Datalog's expressiveness, a separate protocol layer would be required. Such a layer is sketched in Chapter 12 as a possibility, not a part of the present specification. The sketched layer would carry computation hashes and inputs, not the computations themselves; verification of arbitrary computation is a separate problem with its own substantial literature.

11.5 The stack is not real-time

Bitcoin produces approximately one block every ten minutes. An inscription confirms when its containing transaction appears in a mined block. Until confirmation, the inscription is not yet on chain in the sense that other parties can rely on it; until several confirmations, the ordering of inscriptions within recent blocks may still be revised by chain reorganisation.

The stack is therefore not suitable for applications that require sub-second decisions, intra-second event ordering, or guaranteed delivery within a fixed time window. High-frequency trading, real-time control systems, and event streams that produce many events per second are all outside the operating range of the stack.

What the stack is suitable for is what one might call slow truths: claims and consequences that must be traceable across years and decades, that must remain reproducible after the original participants are no longer reachable, that must survive the loss of any particular service or database. The temporal granularity of the stack is the block, not the second; the temporal scope is the lifespan of Bitcoin, not the lifespan of a transaction.

11.6 The stack does not adjudicate truth

This boundary has been stated in earlier chapters and is restated here for completeness. A derivation produced by the stack is a fact about what follows from inscribed premises and inscribed rules. Whether the premises are true is a separate question. Whether the rules are sensible is a separate question. Whether the derivation captures something real about the world is a separate question.

The stack guarantees that the derivation follows validly from the premises and the rules. It does not guarantee that the premises are accurate, that the rules sensibly model the domain, or that the conclusions correspond to anything worth attending to. Truth-evaluation is the work of readers who examine the inscribed material and form judgements about it.

This is sometimes presented as a weakness of formal systems generally: they prove what follows from given premises without warranting the premises. The presentation is correct but the diagnosis is incomplete. A formal system that warranted its own premises would be circular. The stack accepts the limitation as a property of formal reasoning and provides what formal reasoning can provide: mechanical verification of consequences, with the premises made explicit and inspectable.

11.7 The stack does not produce consensus

Two readers may apply different sets of rules to the same chain-state and reach different sets of derivations. The stack records both rule sets; it does not adjudicate between them. A reader who wishes to know what follows from the chain-state must specify which rules they are applying; absent that specification, the question has no unique answer.

This is a feature rather than a defect. The stack admits plural reasonings about the same facts. A community that wishes to converge on a particular interpretation can do so by collectively endorsing a rule set, but the endorsement is a social act, not a property of the chain. The chain records the rules that have been inscribed; readers and communities decide which to apply.

A future protocol layer might specify mechanisms for rule endorsement, weighted voting on rule sets, or reputation-based selection of rules to apply. Such mechanisms are outside the scope of this specification. The present stack accommodates them as future inscriptions but does not require them.

11.8 The stack does not protect against bad actors

Anyone with a Bitcoin wallet can inscribe under PACT. Anyone with a wallet can declare KINSHIP toward avatars that have not consented. Anyone with a wallet can acknowledge LINEAGE from sources that the inscriber has never read. Anyone with a wallet can inscribe rules whose purpose is to mislead.

The stack records these inscriptions alongside well-meaning ones. It does not filter, vet, or prioritise. The verification specified in Chapter 7 confirms that an inscription is structurally valid; it does not confirm that the inscription is honest, accurate, or useful.

This is the cost of open authorship. A stack that filtered inscriptions according to some standard of quality would require an institution to perform the filtering; that institution would become a gatekeeper; the open authorship property would be lost. The stack as specified accepts the cost: bad-faith inscriptions stand alongside good-faith ones, distinguishable only by the work readers do to evaluate them.

The mitigation is structural rather than preventative. Bad inscriptions are detectable by the same mechanisms that make good ones reproducible. A bad-faith KINSHIP can be contradicted by a counter-inscription from any of the named parties. A bad-faith LINEAGE can be challenged by readers familiar with the acknowledged source. A bad-faith rule can be ignored, contradicted by a better rule, or shown to produce nonsensical derivations. The stack provides the material for critique; the work of critique remains social.

11.9 The boundaries together

Taken individually, each boundary states a thing the stack does not do. Taken together, they describe what the stack is for: durable, reproducible, declarative reasoning at the tempo of Bitcoin blocks, in a language strong enough to capture useful inferences but not so strong that reproducibility breaks. The boundaries define the niche.

A reader considering whether to use the stack for a particular purpose can do so by checking the purpose against the boundaries. If the purpose requires sub-second response, the stack is unsuitable. If the purpose requires autonomous execution, the stack is unsuitable. If the purpose requires the system itself to guarantee truth or filter bad actors, the stack is unsuitable. If the purpose can be served by inscribed claims, deterministic derivations, and durable reproducibility, the stack is suitable.

The boundaries are not a list of features missing from a later version. They are properties of what the stack is. A future protocol that wishes to address one of the boundaries — adding sub-second response, or autonomous execution, or filtering — would be a different protocol on a different substrate or with different guarantees. The present stack is what it is: small, slow, public, and verifiable, with the consequences that combination entails.

12 Future extensions

12.1 What this chapter sketches

Earlier chapters specify what is normative. This chapter sketches what may follow without altering the specification. The extensions described here are not commitments. They are possibilities that follow internally from what is on chain today, expressible as additional inscriptions under the existing protocols. Each extension can be added by anyone who controls a sat; none requires a new protocol envelope or a new validity machinery.

The purpose of the sketch is twofold. First, to demonstrate that the present specification is not closed: the four layers admit growth without re-specification. Second, to mark some directions of growth that follow more naturally than others, so that readers considering their own contributions have orientation about where the existing material points.

Five extensions are sketched: recursive rules over `related`, domain-specific predicates such as `peer`, meta-LINEAGE that traces intellectual descent across generations, contradiction-detection rules, and the v2 envelope for RULES that was introduced in Chapter 5. Each is a possibility for the chain; none is required for the chain to function as it does today.

12.2 Recursive rules over related

The first rule on chain (Chapter 5) is non-recursive: it derives `related(A, B)` from `kinship(A, B)` without referring to other instances of `related`. A natural extension is a rule that makes `related` transitive:

```
IF related(A, B) AND related(B, C) THEN related(A, C)
```

Applied to the current three derivations, `related(W, F)`, `related(F, V)`, and `related(W, V)`, the transitivity rule produces no new derivations: the three pairs are already mutually transitive (any reachable pair is already inscribed via the KINSHIP, and the rule produces no further closure on a clique of three).

The transitivity rule becomes consequential as soon as a fourth avatar enters the network. If a new KINSHIP links a fourth avatar X to one of the existing three, the non-recursive rule alone produces direct `related` relations between X and that one avatar. The transitivity rule then closes `related` over the full reachable set, making X related to all three original avatars regardless of which single avatar X was directly linked to.

The recursive rule is fully expressible under RULES v1. Semi-naive evaluation handles transitive closure in PTIME, and the validity rules in Chapter 5 permit recursion without further provisos. The rule is a candidate inscription whenever the inscriber wishes to make the transitivity of `related` explicit on chain.

12.3 Domain-specific predicates

The current rule produces a single derived predicate, `related`. A second rule could introduce a different derived predicate that captures finer distinctions. One example is `peer`, which would distinguish avatars that are related and in the same namespace from avatars that are related but live in different worlds.

```
IF related(A, B) AND same_namespace(A, B) THEN peer(A, B)
```

The rule presupposes a built-in predicate `same_namespace` that returns true when two avatar references share their namespace component. Such built-ins are permitted under RULES v1 (Section 5.3, point 4), but must be specified by the world declaring the rule. The inscribing world would either declare the built-in explicitly or rely on a previously inscribed rule that defines it.

Applied to the current chain-state, the `peer` rule produces no derivations: the three avatars in the existing KINSHIP are each in different namespaces, so no two of them satisfy `same_namespace`. As with the transitivity rule, the consequence becomes visible only with future inscriptions. A second avatar in pact-semantics/open-field entering into KINSHIP with Wanderer-001 would generate a `peer` derivation between them, while leaving the existing cross-namespace relations unaffected.

Other domain-specific predicates follow the same pattern. `collaborator(A, B)` could mark relations within a declared collaborative world. `senior(A, B)` could combine LINEAGE with timestamps to mark predecessors who acted earlier in time. Each predicate is one rule; each rule adds one inscriptive line. The chain accommodates them in parallel rather than in sequence.

12.4 Meta-LINEAGE

LINEAGE acknowledges one source per inscription. Multiple LINEAGE inscriptions from the same avatar acknowledge multiple sources, but no single inscription describes a genealogy across generations. A rule can derive such genealogies from the existing inscriptions:

```
IF lineage(A, B) AND lineage(B, C) THEN intellectual_descent(A, C)
```

The rule reads: if A acknowledges B as predecessor, and B acknowledged C as predecessor, then A is intellectually descended from C. The derivation is transitive across generations of acknowledgement. A second rule, recursive, extends this to arbitrary generation counts:

```
IF intellectual_descent(A, B) AND lineage(B, C) THEN intellectual_descent(A,
C)
```

Applied to the current chain-state, the meta-LINEAGE rule produces one derivation. The existing LINEAGE acknowledges Bitoshi Blockamoto and BST as precedent to PACT. If Blockamoto inscribes a LINEAGE acknowledging some earlier work — say, Casey Rodarmor's Ordinals as the substrate that made BST possible — the meta-LINEAGE rule would derive that PACT is intellectually descended from Ordinals, through BST as intermediate generation.

This kind of derivation is natural in academic and intellectual contexts. Citation networks already form implicit graphs of descent, but the graphs are inferred from texts rather than computed from declared acknowledgements. Meta-LINEAGE on PACT makes the descent graph computable directly from chain-state, as a function of inscribed LINEAGE facts and the rule that interprets them.

12.5 Contradiction detection

A more ambitious rule searches for inconsistencies in the inscribed material:

```
IF derived(P) AND derived(NOT P) THEN contradiction(P)
```

The rule presupposes an extended Datalog with explicit negation in derived facts, which is a non-trivial extension to the v1 syntax. A simpler variant, expressible under v1 without changes, looks for specific kinds of conflict:

```
IF kinship(A, B) AND withdraws_kinship(C, A, B) THEN kinship_disputed(A,
B)
```

Here `withdraws_kinship` would be a transition type that an avatar uses to deny a kinship declared elsewhere. The rule does not resolve the dispute; it derives a `kinship_disputed` fact that readers can then investigate.

Contradiction-detecting rules are useful precisely because they make the wall self-critical. A community using PACT for substantive claims benefits from automated detection of inconsisten-

cies in the claims. The rules cannot determine which side of a contradiction is correct; they can ensure that contradictions, when present, are visible rather than silently coexisting.

12.6 The RULES v2 envelope

Chapter 5, Section 5.8, sketched a v2 envelope for RULES that carries rules as structured abstract syntax trees rather than as expression strings, and that includes explicit predicate definitions. The v2 envelope is a future extension in the strict sense: it is not yet on chain, and no v1 inscription needs to be migrated for the chain to continue functioning.

When v2 inscriptions begin to appear, parsers may evaluate both v1 and v2 rules. The semantics are intended to be compatible: a v1 rule and its v2 equivalent should produce the same derivations from the same facts. The advantage of v2 is in tooling rather than capability: the structured AST is easier for tools to manipulate, the explicit predicate definitions document the intended arity and types of each predicate, and the version number permits future extensions to the rule format without ambiguity.

The migration path from v1 to v2 is straightforward. A v1 rule's expression string can be parsed into an AST by any implementation of the Datalog grammar; the resulting AST is the v2 form. Inscribers who wish to formalise their v1 rules under v2 can do so by inscribing a new TRANSITION with the v2 fields filled. Both forms remain on chain; the v2 form does not invalidate the v1.

12.7 Extensions outside the present specification

Other extensions would require new protocols rather than new rules. Three are noted briefly here, with the understanding that none belongs to the present specification.

A computation protocol, hinted at in PACT v1 [2] (Section 12.5), would carry verifiable computation as a kind of transition: an avatar inscribes a declaration that a specific algorithm, identified by hash, was applied to a specific input, identified by hash, producing a specific output, also by hash. The protocol would not perform the computation; it would record that the computation occurred and provide the data needed to reproduce it. This is a substantial extension and belongs to its own specification document.

A coordination protocol could carry multi-party declarations: multiple avatars co-inscribing a single TRANSITION whose validity requires signatures from all participants. KINSHIP as currently specified is unilateral; a coordinated CO-KINSHIP would require explicit consent from all named parties before taking effect. This extension would offer stronger guarantees of mutual recognition at the cost of higher inscription complexity.

A revocation protocol could provide a structured way to withdraw earlier inscriptions, with the original record remaining on chain but the derived current state reflecting the withdrawal. The current specification permits ad-hoc withdrawal through new TRANSITIONs whose semantics are world-specific; a revocation protocol would standardise this practice.

These three extensions are mentioned to mark their absence from the present specification, not to commit to their future development. Whether they appear depends on whether those who would use them choose to specify them. The present paper provides the substrate on which such specifications could be built, without prejudging which will be needed.

12.8 The character of growth

The extensions sketched in this chapter share a property worth noting. None of them requires re-specifying the four layers. None requires modifying the existing inscriptions. None requires

coordination among existing inscribers. Each is a possibility that becomes available because the four layers exist; each can be realised by anyone who controls a sat and chooses to inscribe.

This is the character of growth that the specification permits. The chain accumulates inscriptions without overwriting them. The inferential layer accumulates rules without invalidating earlier rules. The community of inscribers grows, if it grows, by addition rather than by replacement. What was on chain yesterday remains on chain today; what is added today is visible alongside what was added before.

The specification ends here, in a sense, because growth beyond what is specified does not require further specification. The protocols are minimal. The substrate is permanent. The work that remains is the work of inscribers who find the protocols useful and the substrate worth using. Whether and how that work happens is not for this paper to predict.

13 Acknowledgements

This paper rests on work the author did not do.

The substrate it builds on is Bitcoin, designed by Satoshi Nakamoto and maintained by a global community of contributors since 2009. Without the witness layer that Bitcoin provides, nothing specified here would have anywhere to live. The addressability of that witness layer through individually inscribed satoshis was opened by Casey Rodarmor's Ordinals protocol in early 2023; without that opening, the inscriptions described in this paper would remain technically possible but practically remote.

The four-part ontology that PACT v1 instantiates and that this paper extends is the work of OrdinalsLover, who specified PACT v1 in April 2026. The careful separation between substrate (Bitcoin), transport (Ordinals), and ontology (PACT) that makes the present extensions possible is owed to that earlier specification. Where this paper treats namespace, world, avatar, and transition as primitives, it is using primitives that PACT v1 defined.

The recognition that satoshis can serve as identity anchors was articulated in early form by Bitoshi Blockamoto's Bitmap Signal Theory. PACT v1 acknowledges BST as ontological precedent through a LINEAGE inscription on chain; this paper inherits that acknowledgement and extends it.

The inferential layer that RULES specifies descends from multiple traditions. Robert Kowalski's theoretical work on resolution as programming, and Alain Colmerauer and Philippe Roussel's implementation of Prolog in Marseille in 1972, established that logic could serve as a programming substrate. Hervé Gallaire, Jack Minker, and David Maier developed Datalog from 1977 onwards, in the disciplined form that made it suitable for use on a substrate that requires reproducibility. The modern implementations — Soufflé, RDFox, ddlog, and others — demonstrate that the formal properties scale to substantial fact bases on commodity hardware.

The Semantic Web tradition, opened by Tim Berners-Lee's proposal in 1998 and developed through RDF, OWL, SPARQL, SWRL, and N3Logic, articulated for two decades the vision that machine-readable knowledge could be public, interoperable, and reasonable over. The implementations have always been more fragile than the vision warranted; the protocols specified here move that vision onto a substrate where its fragility falls away.

Ross Quillian's semantic networks, formulated in 1968, provided the earliest formal model of knowledge as a graph of concepts and relations. KL-ONE refined this into description logics in 1985. The four-part pattern this paper inherits is older than computing in some sense; it is the recognised shape of how systems organise themselves to account for identity, context, and recorded action.

Andrew Tanenbaum's principle that systems should be small, clean, safe, and understandable shaped the discipline of the present specification. A protocol that can be read in an afternoon and implemented in a weekend is more likely to be correctly understood and reliably reproduced than a protocol that requires extensive tooling. The present paper has tried to honour this principle.

Frank van Harmelen, professor of Knowledge Representation and Reasoning at the Vrije Universiteit Amsterdam, shaped how ontologies are taught and applied in the Netherlands. His work on the relationship between formal logic and real-world data informs how this paper thinks about worlds as contexts of declared rules rather than arbitrary collections of inscriptions.

Peter Todd's OpenTimestamps demonstrated, well before PACT, that Merkle batching against Bitcoin is a workable pattern for durable timestamping. The verification methods inherited from

PACT v1, and the broader confidence that Bitcoin can serve as anchoring layer for off-chain artefacts, owe directly to that earlier work.

The author also thanks the working partner whose analytical and practical sensibility provided counterweight to the broader-strokes thinking that produced this paper. The discipline of bringing each claim back to a concrete inscription on chain, of distinguishing what works today from what is hoped for tomorrow, was sharpened in conversations that took place outside any document.

Acknowledgement on chain. The intellectual debts named in this chapter are candidates for inscribed LINEAGE declarations under the protocol specified in Chapter 4. Some of those inscriptions exist; others do not yet. The choice of which to inscribe and which to leave in prose is the inscriber's. What the chain gains by inscription is durability; what prose retains is the freedom to discuss debts in nuance and context that an inscription cannot easily carry. Both have their place. This paper itself is one such piece of prose, and it stands alongside the inscriptions that exist and the inscriptions that may yet appear.

Wiard Vasen
IJmuiden, May 2026

A Pseudocode

This appendix gives reference pseudocode for the operations specified in the body of the paper. The pseudocode is not normative for the protocols themselves; the normative specifications are in Chapters 3, 4, 5, and 7. The pseudocode shows one valid implementation strategy. Other implementations that produce identical outputs from identical inputs are equally conformant.

The pseudocode uses a generic procedural notation. Conventions used throughout: capitalised identifiers (A, B, P) denote variables in rule expressions; lowercase identifiers (facts, rules, head) denote program-level variables; indentation indicates block structure; the keyword **return** terminates a procedure; the symbol ⊥ denotes failure or absence.

A.1 Envelope validation for KINSHIP, LINEAGE, RULES

The PACT v1 envelope validation is specified in [2]. The procedures below specify the additional validity rules for the data field of each protocol, applied after PACT v1 envelope validation has succeeded.

```
procedure ValidateKinshipData(data):
  if data.kinship is not array:
    return invalid("kinship must be array")
  if length(data.kinship) < 2:
    return invalid("kinship requires at least two entries")
  for each entry in data.kinship:
    if not WellFormedAvatarRef(entry):
      return invalid("malformed avatar reference")
  if data.declared_at not in ISO8601 format:
    return invalid("declared_at must be ISO 8601")
  return valid

procedure ValidateLineageData(data):
  if data.acknowledges is not non-empty string:
    return invalid("acknowledges required")
  if data.kind not present:
    return invalid("kind required")
  if data.declared_at not in ISO8601 format:
    return invalid("declared_at must be ISO 8601")
  return valid

procedure ValidateRulesData(data, scope):
  if data.rule_id not unique within scope:
    return invalid("rule_id collision in scope")
  if data.expression not parseable as Datalog rule:
    return invalid("expression not parseable")
  rule := ParseRule(data.expression)
  if not Safe(rule):
    return invalid("rule violates safety condition")
  if not Stratified(rule, all_rules_in_scope):
    return invalid("rule violates stratification")
  return valid
```

The procedure `WellFormedAvatarRef` checks that an avatar reference matches the form `namespace/world/avata`
The procedure `ParseRule` converts an expression string into an abstract syntax tree. The procedures `Safe` and `Stratified` are defined below.

A.2 Safety condition

A rule is safe if every variable appearing in the head also appears in at least one positive body literal. The check is straightforward.

```
procedure Safe(rule):
  head_vars := variables_in(rule.head)
  body_vars := empty set
  for each literal in rule.body:
    if literal is positive:
      body_vars := body_vars union variables_in(literal)
  for each var in head_vars:
    if var not in body_vars:
      return false
  return true
```

A rule that fails the safety check produces no derivations because variables in the head cannot be bound to constants through evaluation against any chain-state.

A.3 Stratification

Stratification ensures that a predicate appearing negated in one rule's body does not depend, directly or transitively, on its own derivation. The standard check builds a dependency graph among predicates and verifies that no negative edge participates in a cycle.

```
procedure Stratified(rules):
  graph := empty directed graph
  for each rule in rules:
    head_pred := predicate_of(rule.head)
    for each literal in rule.body:
      body_pred := predicate_of(literal)
      if literal is negative:
        add_edge(graph, body_pred, head_pred, label="negative")
      else:
        add_edge(graph, body_pred, head_pred, label="positive")
  for each cycle in graph:
    if cycle contains any edge labelled "negative":
      return false
  return true
```

A rule set that fails stratification cannot be evaluated under the standard semi-naive procedure because the fixpoint is not uniquely defined.

A.4 Semi-naive evaluation

Semi-naive evaluation computes the fixpoint of a Datalog rule set against a fact set in a manner that avoids redundant recomputation across iterations.

```
procedure SemiNaiveEvaluate(rules, facts):
  state := facts
  delta := facts
  while delta is non-empty:
    new_derivations := empty set
    for each rule in rules:
      bindings := FindNewBindings(rule, state, delta)
```

```
      for each binding in bindings:
        derived_fact := Substitute(rule.head, binding)
        if derived_fact not in state:
          new_derivations := new_derivations
                             union {derived_fact}
    delta := new_derivations
    state := state union new_derivations
  return state
```

The procedure `FindNewBindings` finds variable bindings that satisfy the rule's body, with at least one body literal matching a fact in `delta` (the facts new in the previous iteration). This restriction is what makes the algorithm semi-naive rather than naive: each rule is applied only when there is potential for new derivations, not on every iteration.

The procedure `Substitute` substitutes variables in the rule's head with the constants from the binding, producing a ground fact suitable for addition to the state.

```
procedure Substitute(literal, binding):
  result := copy of literal
  for each variable in literal.args:
    if variable in binding:
      replace variable in result.args with binding[variable]
  return result
```

The fixpoint is reached when no new derivations are produced in an iteration. The state at that point contains all facts derivable from the input rules and facts. The fixpoint is unique under the stratification assumption: any other implementation following the same semantics produces the same state.

A.5 Derivation verification

Verifying a single derived fact involves finding a rule whose head matches the fact and confirming that the rule's body can be satisfied against the chain-state.

```
procedure VerifyDerivation(fact, rules, chain_state):
  for each rule in rules:
    binding := UnifyHead(rule.head, fact)
    if binding is bot:
      continue
    body_satisfied := true
    for each literal in rule.body:
      substituted := Substitute(literal, binding)
      if substituted not in chain_state and not
         Derivable(substituted, rules, chain_state):
        body_satisfied := false
        break
    if body_satisfied:
      return valid(rule, binding)
  return not_derivable
```

The procedure `UnifyHead` attempts to find a variable binding that, when substituted into the rule's head, produces the target fact. If no such binding exists for a particular rule, the procedure returns $\perp$ and the verification proceeds to the next rule.

The procedure `Derivable` is recursively the same verification applied to body literals that are

themselves derivations. Care must be taken to avoid infinite recursion in the presence of cyclic rules; standard techniques include memoisation of already-verified facts and termination on fixpoint detection.

A.6 Audit path through layers

A reader who wishes to trace a derived fact down to its chain-inscribed premises uses an audit-path procedure that unfolds derivations layer by layer until reaching inscribed facts.

```
procedure AuditPath(fact, rules, chain_state):
  path := empty list
  if fact in chain_state:
    return path with [(inscribed, fact)]
  result := VerifyDerivation(fact, rules, chain_state)
  if result is not_derivable:
    return failure
  rule := result.rule
  binding := result.binding
  path := append(path, (derived_by, rule.rule_id, binding))
  for each literal in rule.body:
    substituted := Substitute(literal, binding)
    sub_path := AuditPath(substituted, rules, chain_state)
    if sub_path is failure:
      return failure
    path := append(path, sub_path)
  return path
```

The procedure terminates when every leaf of the audit path is an inscribed fact. The resulting path documents the complete genealogy of the derived fact: which rules contributed at each step, which bindings were used, and which inscribed premises support the chain of inference. The path is the same for any reader applying the same rules to the same chain-state.

A.7 Notes on implementation

The pseudocode above is reference-level. Concrete implementations face engineering choices that the pseudocode does not address:

- Storage and indexing of facts for efficient lookup during `FindNewBindings`.
- Handling of equality and constant comparison, especially across implementations that may canonicalise differently.
- Memoisation strategies for `Derivable` to avoid recomputing the same sub-derivations.
- Termination heuristics for very large fact sets, where reaching the fixpoint may require many iterations.

These engineering choices do not affect the semantics. Two implementations making different engineering choices but correctly implementing the algorithms above produce identical fixpoints from identical inputs. This is the property that makes the inferential layer publicly reproducible: the result is a function of the inputs, and the function is well-defined.

B Worked examples

This appendix presents the full chain-state on which the body of the paper has drawn, together with detailed walkthroughs of how the specified procedures apply to that state. The material here is illustrative rather than normative; the inscriptions are real, but readers should consult the chain itself for authoritative content.

The chain-state used throughout this appendix is the state of the Bitcoin chain at the time of writing (May 2026), as visible on the Ordinalswall live page (`ordinalswall.com/live.html`). All inscription identifiers are as recorded on Bitcoin mainnet.

B.1 The chain-state at the time of writing

Four avatars, two relations, and one rule are inscribed.

Avatars.

- Wanderer-001, in `pact-semantics/open-field`, on sat `1168224522821204`.
- Forecaster-001, in `pact-semantics/ordinals`.
- vonNeumann-001, in `pact-semantics/mathematics`.
- Auditor-001, in `agents/openclaw`.

Relations.

- KINSHIP among Wanderer-001, Forecaster-001, and vonNeumann-001 (inscription `93e3474b69a0ab2ef65a`
- LINEAGE acknowledging Bitoshi Blockamoto and Bitmap Signal Theory (inscription `dafd8748c37b914866a28a23a30be645693382c1fccd5bcc6dd750b59f6843bfi0`).

Rule.

- `kinship-implies-relation`, declaring `IF kinship(A,B) AND avatar(A) AND avatar(B) THEN related(A,B)` (inscription `3e72086f88d791f5122e1cb0bab0727761ec2462ef4badf93dcd60a93e89` inscribed 4 May 2026).

B.2 Example 1: validating a KINSHIP inscription

The KINSHIP inscription as decoded from chain-state contains the envelope:

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "pact-semantics",
  "world": "open-field",
  "avatar": "wanderer-001",
  "action": "KINSHIP",
  "data": {
    "kinship": [
      "pact-semantics/open-field/wanderer-001",
      "pact-semantics/ordinals/forecaster-001",
      "pact-semantics/mathematics/vonneumann-001"
    ],
    "declared_at": "2026-05-02T00:00:00Z",
    "note": "Three avatars in mutual kin across three worlds."
```

```
  }
}
```

The validation proceeds in two stages.

Stage 1: PACT v1 envelope validation. The procedures from PACT v1 (Section 5 of [2]) verify that the envelope has `p` equal to `PACT`, `v` equal to `1`, and `type` equal to `TRANSITION`. The fields `ns`, `world`, `avatar`, `action`, and `data` are all present and the `data` field is an object. The envelope is structurally valid.

Stage 2: KINSHIP-specific validation. The procedure `ValidateKinshipData` from Appendix A applies:

- `data.kinship` is an array. Pass.
- Length of `data.kinship` is 3, which is at least 2. Pass.
- Each entry matches the form `namespace/world/avatar-name`. Pass.
- `data.declared_at` is in ISO 8601 format. Pass.

The KINSHIP inscription is valid. After confirmation it contributes the following facts to the chain-state:

- `kinship(wanderer-001, forecaster-001)`
- `kinship(wanderer-001, vonneumann-001)`
- `kinship(forecaster-001, vonneumann-001)`

By symmetry of KINSHIP, the same facts hold with arguments reversed; the parser may represent each pair once or twice depending on implementation choice, with no semantic difference.

B.3 Example 2: validating a LINEAGE inscription

The LINEAGE inscription contains the envelope:

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "pact-semantics",
  "world": "open-field",
  "avatar": "wanderer-001",
  "action": "LINEAGE",
  "data": {
    "acknowledges": "blockamoto/bst",
    "kind": "precedent",
    "declared_at": "2026-05-03T00:00:00Z",
    "note": "Bitmap Signal Theory recognised an early form of sat-bound claim."
  }
}
```

Stage 1. Envelope validation succeeds as for Example 1.

Stage 2: LINEAGE-specific validation. The procedure `ValidateLineageData` from Appendix A applies:

- `data.acknowledges` is the non-empty string `blockamoto/bst`. Pass.
- `data.kind` is the string `precedent`, one of the four canonical kinds. Pass.
- `data.declared_at` is in ISO 8601 format. Pass.

The LINEAGE inscription is valid. After confirmation it contributes the following fact to the chain-state:

- `lineage(wanderer-001, blockamoto/bst, precedent)`

The fact is asymmetric: it records that wanderer-001 acknowledges blockamoto/bst, with no implied reciprocal fact from blockamoto/bst toward wanderer-001.

B.4 Example 3: validating a RULES inscription

The RULES inscription contains the envelope:

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "pact-semantics",
  "world": "open-field",
  "avatar": "wanderer-001",
  "action": "RULES",
  "data": {
    "rule_id": "kinship-implies-relation",
    "expression": "IF kinship(A,B) AND avatar(A) AND avatar(B) THEN related(A,B)",
    "applies_to_protocols": ["PACT", "KINSHIP"],
    "declared_by": "pact-semantics/open-field/wanderer-001",
    "declared_at": "2026-05-04T00:00:00Z",
    "note": "If two avatars are in mutual kinship, they stand in the relation '
        related' to each other."
  }
}
```

Stage 1. Envelope validation succeeds.

Stage 2: RULES-specific validation. The procedure `ValidateRulesData` from Appendix A applies:

- `rule_id` `kinship-implies-relation` is unique within the namespace `pact-semantics` at the time of inscription. Pass.
- `expression` parses as a Datalog rule with head `related(A,B)` and body `kinship(A,B), avatar(A), avatar(B)`. Pass.
- Safety check: variables `A` and `B` appear in both head and body. Pass.
- Stratification check: the rule contains no negation, so stratification is trivially satisfied. Pass.

The rule is valid. It is added to the rule set in scope.

B.5 Example 4: full evaluation against the chain-state

The semi-naive evaluation procedure from Appendix A is now applied to the chain-state described above, with the single rule `kinship-implies-relation` as the rule set.

Initial state. The state contains the avatar facts:

- `avatar(wanderer-001)`
- `avatar(forecaster-001)`
- `avatar(vonneumann-001)`
- `avatar(auditor-001)`

and the kinship facts:

- `kinship(wanderer-001, forecaster-001)`
- `kinship(wanderer-001, vonneumann-001)`
- `kinship(forecaster-001, vonneumann-001)`

and the lineage fact:

- `lineage(wanderer-001, blockamoto/bst, precedent)`

Initial `delta` equals the initial state.

Iteration 1. The rule is applied to each binding that satisfies the body using at least one fact from `delta`.

For `A := wanderer-001, B := forecaster-001`:

- `kinship(wanderer-001, forecaster-001)` is in delta. Satisfied.
- `avatar(wanderer-001)` is in delta. Satisfied.
- `avatar(forecaster-001)` is in delta. Satisfied.

The body is satisfied. The head substitutes to `related(wanderer-001, forecaster-001)`, which is not yet in the state. It is added to `new_derivations`.

The same applies for the bindings `A := wanderer-001, B := vonneumann-001` and `A := forecaster-001, B := vonneumann-001`, producing `related(wanderer-001, vonneumann-001)` and `related(forecaster-001 vonneumann-001)`.

For bindings involving auditor-001, no kinship literal matches, so no derivations are produced.

After iteration 1, `new_derivations` contains three `related` facts. `delta` is set to these three; `state` is updated to include them.

Iteration 2. The rule is applied with the new `delta`. The rule's body requires `kinship(A,B)`, but no `kinship` facts are in the new delta (only `related` facts are). Therefore no rule application uses a delta fact in iteration 2; `new_derivations` is empty.

The fixpoint is reached. The final state contains the four avatar facts, the three kinship facts, the one lineage fact, and three derived `related` facts:

- `related(wanderer-001, forecaster-001)`
- `related(wanderer-001, vonneumann-001)`

- `related(forecaster-001, vonneumann-001)`

B.6 Example 5: verifying a single derivation

A reader wishes to verify that `related(wanderer-001, forecaster-001)` follows from the chain-state.

The procedure `VerifyDerivation` from Appendix A applies:

Step 1. Find a rule whose head matches the target. The rule `kinship-implies-relation` has head `related(A, B)`. `UnifyHead(related(A, B), related(wanderer-001, forecaster-001))` produces the binding `A := wanderer-001, B := forecaster-001`.

Step 2. Substitute the binding into the body. The body becomes:

- `kinship(wanderer-001, forecaster-001)`
- `avatar(wanderer-001)`
- `avatar(forecaster-001)`

Step 3. Confirm each substituted body literal against the chain-state.

- `kinship(wanderer-001, forecaster-001)`: in chain-state from the KINSHIP inscription. Satisfied.
- `avatar(wanderer-001)`: in chain-state from Wanderer's PACT constitution. Satisfied.
- `avatar(forecaster-001)`: in chain-state from Forecaster's PACT constitution. Satisfied.

All body literals are satisfied. The derivation is valid.

B.7 Example 6: full audit path

A reader wishes to trace the full genealogy of `related(wanderer-001, vonneumann-001)` back to inscribed premises.

The `AuditPath` procedure from Appendix A applies:

Layer 1: target fact. The fact `related(wanderer-001, vonneumann-001)` is not in chain-state (it is derived). Proceed to verification.

Layer 2: rule application. The rule `kinship-implies-relation` produces the fact under the binding `A := wanderer-001, B := vonneumann-001`. This step is recorded in the audit path:

> derived_by: kinship-implies-relation,
> binding: {A: wanderer-001, B: vonneumann-001}

Layer 3: body literals. Each body literal is checked.

`kinship(wanderer-001, vonneumann-001)` is found in chain-state, contributed by the KINSHIP inscription `93e3474b...0557i0`. Audit path entry:

> inscribed: kinship(wanderer-001, vonneumann-001),
> inscription: 93e3474b...0557i0

`avatar(wanderer-001)` is in chain-state from the Wanderer-001 constitution inscription. Audit path entry:

> inscribed: avatar(wanderer-001),
> inscription: *Wanderer-001 constitution*

`avatar(vonneumann-001)` is in chain-state from the vonNeumann-001 constitution inscription. Audit path entry:

> inscribed: avatar(vonneumann-001),
> inscription: *vonNeumann-001 constitution*

Termination. All leaves of the audit path are inscribed facts. The path is complete. The derivation `related(wanderer-001, vonneumann-001)` traces to three inscribed premises through one rule application.

B.8 Example 7: a fact that does not derive

A reader wishes to verify whether `related(wanderer-001, auditor-001)` follows from the chain-state.

Verification. The procedure `VerifyDerivation` attempts to find a binding for `kinship-implies-relation`. The head matches under the binding `A := wanderer-001, B := auditor-001`. The body becomes:

- `kinship(wanderer-001, auditor-001)`
- `avatar(wanderer-001)`
- `avatar(auditor-001)`

The literals `avatar(wanderer-001)` and `avatar(auditor-001)` are in chain-state. The literal `kinship(wanderer-001, auditor-001)` is not: no KINSHIP inscription has named auditor-001 as kin. The body is not satisfied for this binding.

No other rule's head matches the target. The verification returns `not_derivable`.

The negative result is itself informative. It documents that auditor-001 stands on the chain without entering into derived relations with the other avatars. This corresponds to the visualisation on the Ordinalswall live page where clicking on Auditor-001 shows no connections to other elements.

B.9 Notes on the worked examples

The seven examples above traverse the full path from inscription to verified derivation. Examples 1 to 3 cover inscription validation, one for each protocol. Example 4 covers the full evaluation procedure. Example 5 covers single-derivation verification. Example 6 covers genealogy tracing. Example 7 covers the negative case.

The examples are reproducible by any reader with access to the chain-state described in Section B.1. The same procedures applied to the same inputs produce the same outputs in any conforming implementation. Discrepancies between implementations indicate errors that can be located precisely within the worked examples: a different binding, a missing fact, an unrecognised inscription. The worked examples thus serve both as illustrations of the specification and as test cases for implementations.

References

[1] Hervé Gallaire, Jack Minker, and Jean-Marie Nicolas. Logic and databases: A deductive approach. *ACM Computing Surveys*, 16(2):153–185, 1984. Foundational article on Datalog as a deductive query language; the term Datalog itself was coined earlier in workshops organised by these authors from 1977 onwards.

[2] OrdinalsLover. PACT – Permanent Anchored Computation Theory: A minimal protocol for identity and anchored action on Bitcoin via Ordinals. Specification document, April 2026. Inscribed and circulated through Ordinals on Bitcoin mainnet.

[3] Casey Rodarmor. Ordinal theory handbook. `https://docs.ordinals.com`, 2023. Specification of the Ordinals protocol for attaching inscriptions to individual satoshis.