

Goal-Relative Assurance of Dynamically Generated Execution Traces

Robert Betts
Martyn Riley
Jamie Riley-Betts

Riley Betts Ltd
`robert.betts@rileybetts.ai`

11 September 2026

Extending *Trust and Proof* for Agentic Process Orchestration

Riley Betts Ltd working paper, intended for publication at <https://rileybetts.ai/>.

Abstract

Traditional process assurance obtains much of its tractability from process invariance. A process is specified, approved and repeatedly executed against different cases. Individual cases may introduce different facts, evidence, exceptions and dependencies, but the process itself remains substantially stable. Assurance can therefore focus on whether an execution conformed to an approved process and whether the assumptions surrounding that execution were satisfied.

Agentic orchestration changes this structure. For consequential processes, such as regulated client onboarding, both the execution trace and the assumptions on which its acceptance depends may vary from instance to instance. An agent may select evidence, tools, branches, ordering, repetitions, escalations and external dependencies dynamically. The assurance problem can therefore no longer be expressed adequately as proof that a fixed process was followed.

This paper develops a goal-relative formulation. Let G_a denote the governing goal for cases of type a , x_i the actual execution trace for instance i , and A_i the assumptions and dependencies materially associated with that execution. The claim $P(G_a, x_i \mid A_i)$ states that there is an acceptable proof or evidentiary object establishing that execution x_i satisfies G_a relative to the execution boundary A_i . That is not the proof inside discharge. This alone is insufficient. A proof can be valid relative to its declared boundary while that boundary omits assumptions that governance requires for the goal. We therefore distinguish a mandatory goal-assurance floor A_G , an instance-specific proof-support set A_g^i , and the actual execution boundary A_i . Write:

$$A_g^i \models A_G$$

when the instance-specific proof-support set satisfies the mandatory assurance floor. Hygiene is separate:

$$A_g^i \subseteq A_i$$

Boundary Sufficiency requires both:

$$A_g^i \models A_G \wedge P(G_a, x_i \mid A_g^i)$$

Discharge is stricter. It holds only if those hold, the remaining residue is acceptable under the governing policy, and:

$$A_g^i \subseteq A_i$$

Residual trust is consequently goal-relative. $R(A_g^i)$ is the residue of the proof-support set. $R(A_i)$ is the residue of the execution boundary. From $A_g^i \subseteq A_i$ it does not follow that $R(A_g^i) \subseteq R(A_i)$. A valid proof does not empty $R(A_g^i)$. $P(G_a, x_i \mid A_g^i)$ does not entail $R(A_g^i) = \emptyset$. The local-discharge conjunct of Betts (2026), $\neg \text{Need}(T, x \mid A)$, is not restated here, because the execution is not a member of the support. In agentic systems the execution can itself introduce, replace or discharge assumptions, making the proof boundary partly endogenous to the execution. The governed object is therefore not a process definition but an instance-specific assurance model connecting goal, execution, proof support, execution boundary and residual trust. The paper does not claim runtime assurance, goal-oriented specification, dynamic assurance, proof-carrying artefacts, or residual trusted assumptions as new. The contribution is a formally checked goal-relative assurance model for dynamically generated consequential execution. Discharge holds only if proof validity, floor satisfaction, residual-trust acceptability, and boundary hygiene all hold. The first three are distinct. Hygiene is a further conjunct of that only-if. A run may partly construct the boundary against which it is later accepted. That situation is set out in §§12–15. The companion development does not define the boundary as a result of the execution. It machine-checks the model’s central identities and supplies countermodels for the claimed non-entailments.

Keywords: goal-relative assurance; agentic orchestration; proof boundary; trust residue; process assurance; governance; formal verification; machine-checked metatheory

How to cite: Betts, R., Riley, M. and Riley-Betts, J. (2026) ‘Goal-Relative Assurance of Dynamically Generated Execution Traces’. Riley Betts Ltd. Available at: <https://rileybetts.ai/papers/goal-relative-assurance>

Terms used with a fixed sense in this paper are collected in the glossary before the references. That glossary is the subset of the series master glossary, `master-glossary.md`, that this paper uses. The shared senses are fixed there.

Contents

1	Introduction	5
2	The assurance object has changed	5
3	Goal, execution and boundary	6
4	Why $P(G_a, x_i \mid A_i)$ is not enough	7
5	The mandatory goal-assurance floor A_G	8
6	The instance-specific proof-support set A_g^i	9
7	Sufficiency and minimality	9
8	Boundary Sufficiency	10
9	Boundary authority	11
10	Actual execution boundary A_i	12
11	Goal-relative trust residue	13
11.1	Execution residue	13
11.2	Goal-relative residue	13
11.3	Trust inherited from the mandatory floor	14
12	The execution can construct its own boundary	14
13	Dynamic Boundary Sufficiency	15
14	The temporal dimension	16
15	Two temporal scales	16
16	Goal-relative execution equivalence	17
17	Goal-relative trust-dominance	18
18	Risk: valid proof, invalid boundary	19
19	Risk: silent boundary substitution	19
20	Risk: self-justifying execution	20
21	Risk: residue migration	21
22	Regulated client onboarding	21
23	From process specification to goal specification	22
24	Proof-carrying execution	23
25	Goal Discharge Condition	23

26 Relationship to prior work	24
26.1 Runtime certification and dynamic assurance	26
26.2 Goal-oriented requirements and declarative process models	26
26.3 Runtime assurance	26
26.4 Proof-carrying code	26
26.5 Trust and trusted computing bases	26
27 The extended assurance structure	27
27.1 Fixed artefact or bounded mechanism	27
27.2 Deterministic instance-specific process	27
27.3 Agentic process	27
27.4 Boundary-constrained agentic process	28
27.5 Dynamic agentic process	28
28 Consequences for governance	28
28.1 Goal validity	28
28.2 Minimum-boundary validity	29
28.3 Execution validity	29
28.4 Proof-support sufficiency	29
28.5 Boundary integrity	29
28.6 Residual-trust acceptability	29
29 Open formal questions	30
29.1 Uniqueness of A_g^i	30
29.2 Semantic assumptions	30
29.3 Composition	31
29.4 Temporal validity	31
29.5 Goal-relative residue ordering	31
29.6 Policy over residue	31
29.7 What the structure does not decide	32
30 Conclusion	32
31 Glossary	33
32 References	34

1 Introduction

Conventional process assurance lives off invariance. A process is specified, approved and reused. Individual cases introduce different facts, evidence and exceptions, but the path itself remains substantially stable. Assurance can therefore ask whether this run followed the approved process and whether the assumptions surrounding that run were satisfied.

Agentic orchestration removes that invariant. For consequential work such as regulated client onboarding, both the execution trace and the assumptions on which its acceptance depends may be chosen per instance. An agent may select evidence, tools, branches, order and external dependencies dynamically. Proof that a fixed process was followed is then the wrong object.

This paper extends Betts (2026). That work treats a proof as boundary-relative and residual trust as what remains of the boundary after local discharge. Here the execution is no longer a member of the support, and the boundary may be constructed during the run. The stable objects above the execution are the governing goal and a mandatory assurance floor.

A valid proof is not enough if it answers a weaker question than governance required. Discharge holds only if four conditions hold: proof validity, floor satisfaction, residual-trust acceptability, and boundary hygiene. The first three are distinct. Hygiene is required separately, and it is not part of that independence result. The unit of assurance is the instance model that links those objects, not the process definition.

This paper does not claim runtime assurance, goal-oriented specification, dynamic assurance, proof-carrying artefacts, or residual trusted assumptions as new. Its contribution is a formally checked goal-relative assurance model for dynamically generated consequential execution. Sections 12–15 set out the case in which a run may partly construct the boundary against which it is later accepted. The companion development does not define that dependence. The model distinguishes an independently governed mandatory assurance floor A_G , the concrete instance support A_g^i used to establish the goal, the wider execution boundary A_i , and the residual trust $R(A_g^i)$. Discharge is the four-conjunct only-if above. Proof validity, floor satisfaction, and residual-trust acceptability are the distinct conditions.

A companion Lean 4 development machine-checks the model’s central identities and implications and supplies explicit countermodels for the claimed non-entailments and failure modes. Machine-checked internal validity of the framework is not empirical or normative validity of an instantiation.

The paper therefore does not claim that formal methods for autonomous systems are new; that Lean establishes that a real onboarding system is safe; that the framework derives the correct assurance floor, the correct Need relation, or an institution’s acceptable residual trust; or that the open composition and semantic-equivalence questions of §29 have been solved.

The next section shows why the object itself has changed.

2 The assurance object has changed

A conventional process is designed around repeatability.

Suppose a regulated institution classifies a prospective client as type a . Regulation and institutional policy determine a process:

$$x_a$$

For clients c_1 , c_2 , and c_3 of that type, the prescribed execution is substantially invariant:

$$x(c_1) = x(c_2) = x(c_3) = x_a$$

The clients are not invariant.

Their identities, ownership structures, jurisdictions, source documents, sanctions results, risk indicators and exceptional circumstances differ. These differences produce different assumptions and dependencies:

$$A_1 \neq A_2 \neq A_3$$

The assurance claims can therefore be represented as:

$$P(x_a \mid A_1)$$

$$P(x_a \mid A_2)$$

$$P(x_a \mid A_3)$$

The process remains fixed while its proof boundary varies.

This already means that identical process execution does not entail identical assurance. Two clients can traverse the same workflow while the acceptance of their outcomes rests on different evidence and different residual trust.

Agentic orchestration removes the remaining invariant.

An agent may determine which evidence to collect, which sources to query, which checks are material, which steps require repetition, which specialists or humans to consult, and the order in which those actions occur. For two clients of the same type:

$$A_1 \neq A_2$$

and now potentially:

$$x_1 \neq x_2$$

The system is no longer simply executing a predefined process against varying cases. It is constructing an execution appropriate to each case.

Goal-oriented requirements and declarative process models already allow more than one path to a common goal. The narrower point here is that those generated executions may also carry different proof supports and different actual execution boundaries, and that those differences remain governance-relevant even when the governing goal is met. The assurance object must therefore move from the process definition to the particular execution and its instance-specific assurance model.

3 Goal, execution and boundary

The first extension is to distinguish what must be achieved from what the system actually did.

Let:

$$G_a$$

be the governing goal or obligation for a case of type a .

Let:

$$x_i$$

be the actual execution trace for instance i .

Let:

$$A_i$$

be the assumptions and dependencies materially associated with that execution.
The claim relative to the execution boundary becomes:

$$P(G_a, x_i \mid A_i)$$

read as:

there exists an acceptable proof or evidentiary object establishing that execution x_i satisfies governing goal G_a , relative to the execution boundary A_i .

This is not proof validity. Proof validity, defined in §25, is $P(G_a, x_i \mid A_g^i)$, the same goal and execution relative to the proof-support set. Hygiene does not make the two formulas one claim.

This separates three objects:

- G_a : what must be true for cases of type a ;
- x_i : what this execution actually did;
- A_i : what acceptance of that conclusion depends upon for this execution.

Two executions may differ:

$$x_1 \neq x_2$$

and their boundaries may differ:

$$A_1 \neq A_2$$

while both satisfy the same governing goal:

$$P(G_a, x_1 \mid A_1)$$

$$P(G_a, x_2 \mid A_2)$$

The invariant has moved from **process identity** to **goal satisfaction**.

That move is not itself new. Goal-oriented requirements engineering and declarative process models already separate desired properties from fixed procedural paths. What matters for consequential agentic execution is that two runs can satisfy the same governing goal while resting on different proof supports and different actual execution boundaries. Those differences remain governance-relevant even when the outcomes are identical.

It is not necessary for every valid execution to be identical. It is necessary for every consequential execution to establish the required goal against a support that satisfies the independently governed floor.

4 Why $P(G_a, x_i \mid A_i)$ is not enough

The formulation above contains a critical risk.

If A_i is merely the set of assumptions and dependencies associated with the execution, then the system may produce a proof that is valid relative to its own declared boundary while that boundary is itself inadequate.

Formally:

$$P(G_a, x_i \mid A_i)$$

does not establish that A_i contains everything governance requires before G_a may be discharged.

This matters particularly when the same autonomous system participates in:

1. selecting the execution;
2. selecting evidence;
3. selecting tools and dependencies;
4. constructing the proof boundary; and
5. presenting the resulting assurance claim.

Without an independently specified minimum boundary, the agent can become both the generator of the action and, indirectly, the author of the standard against which that action is judged.

A proof can therefore be internally valid and still be insufficient for governance.

This requires another distinction.

5 The mandatory goal-assurance floor A_G

For a governing goal G_a , there may be assurance requirements that must be satisfied in every acceptable proof of that goal.

Call this set:

$$A_G$$

A_G is the **mandatory goal-assurance floor**.

It is not the set of concrete assumptions or dependencies an individual execution happened to use. It is the set of assurance requirements imposed by the governing definition of G_a .

For regulated onboarding, elements might include requirements concerning:

- authoritative customer identity;
- correct customer classification;
- sanctions status;
- beneficial ownership where applicable;
- provenance of required evidence;
- applicable jurisdiction;
- applicable policy or regulatory version;
- required approval authority.

The precise contents are determined by the governing obligation, not by the agent's chosen execution.

This distinction matters because a requirement and the mechanism that satisfies it are not necessarily the same object. If A_G requires authoritative identity evidence, one permissible execution may satisfy that requirement through one approved identity provider and another through a different authoritative source.

We therefore write:

$$A_g^i \models A_G$$

when the concrete, instance-specific proof-support set A_g^i satisfies the mandatory assurance requirements in A_G . Satisfaction is covering. Every requirement in A_G is instantiated by some

assumption in A_g^i . It is not membership of a requirement's name in the support. Requirements and assumptions are different types.

The agent is free to construct a different execution and may satisfy a requirement through different permitted mechanisms. It is not free to silently weaken the requirement itself.

6 The instance-specific proof-support set A_g^i

The mandatory floor does not necessarily contain everything needed for a particular case.

An individual client may introduce material circumstances requiring additional evidence or dependencies.

Let:

$$A_g^i$$

be an instance-specific **proof-support set**: a candidate set of concrete assumptions and dependencies used as the boundary of the proof claim. It is not defined as the unique sufficient set, and it is not defined by minimality.

The proof claim relative to that candidate is:

$$P(G_a, x_i \mid A_g^i)$$

That claim may fail. If it holds, the candidate is sufficient for the proof. Sufficiency is P , not the definition of A_g^i .

For the proof to be admissible for the governing purpose it must also hold that:

$$A_g^i \models A_G$$

For a simple case, the concrete support in A_g^i may do no more than instantiate the requirements of A_G . A case containing material exceptions may require additional assumptions or dependencies:

$$A_g^i = S_i \cup E_i$$

where S_i is the concrete support satisfying the mandatory floor and E_i is a set of additional assumptions introduced by the facts of instance i .

For example, an ambiguous sanctions match may add an identity-resolution assumption. A complex ownership chain may add evidence concerning intermediate beneficial owners. A jurisdictional feature may add a further dependency.

The proof-support set is therefore **goal-relative and instance-specific**.

7 Sufficiency and minimality

Sufficiency and minimality should be kept separate.

A proof-support set A_g^i is sufficient for the proof claim when:

$$P(G_a, x_i \mid A_g^i)$$

A sufficient set need not be unique and need not be minimal. A larger set containing a sufficient subset is still sufficient for P .

It is sufficient for the governing purpose only when it also satisfies:

$$A_g^i \models A_G$$

A proof-support set may additionally be minimal for the proof claim if:

$$\forall B \subset A_g^i, \neg P(G_a, x_i \mid B)$$

No proper subset is then sufficient to establish the same claim for that execution.

Minimality is not required by the framework. There may be several incomparable minimal support sets, and the logically minimal set need not by itself satisfy the assurance requirements imposed by governance. A proof may establish a proposition from fewer assumptions than the institution is willing to accept as a sufficient basis for the consequential decision.

The distinction is therefore:

- A_G specifies what governance requires;
- A_g^i contains the concrete proof support used for this instance;
- A_i is the wider actual execution boundary.

Inclusion of the proof-support set in the execution boundary is hygiene, required for discharge:

$$A_g^i \subseteq A_i$$

It does not follow from P or from $\models$. That is shown in §10.

The execution boundary may contain implementation or operational dependencies that are not material to proof of this particular goal. Such dependencies may still matter to security, privacy, resilience or another governing goal. Materiality is always relative to the property currently being established.

8 Boundary Sufficiency

The distinction above yields a governance condition.

A consequential proof claim should not be accepted merely because:

$$P(G_a, x_i \mid A_g^i)$$

holds.

It must also establish:

$$A_g^i \models A_G$$

Together:

$$A_g^i \models A_G \wedge P(G_a, x_i \mid A_g^i)$$

Call this **Boundary Sufficiency**.

Boundary Sufficiency: The proof-support set satisfies the mandatory assurance requirements of the goal, and the proof of the goal holds relative to that set. This is necessary for discharge. It is not discharge.

This prevents an important class of failure. A proof may be valid relative to its declared boundary while that boundary is insufficient for the governing purpose.

The two conjuncts are independent.

$$P(G_a, x_i \mid A_g^i)$$

does not imply:

$$A_g^i \models A_G$$

and:

$$A_g^i \models A_G$$

does not imply:

$$P(G_a, x_i \mid A_g^i)$$

The proof asks:

Does this execution establish the goal relative to these assumptions?

Boundary Sufficiency asks:

Does the boundary supporting that proof satisfy what governance requires before this goal may be discharged?

Both are necessary.

9 Boundary authority

Boundary Sufficiency raises another question: who defines A_G ?

If the autonomous system that constructs x_i can also redefine A_G , the protection is largely circular.

The mandatory goal-assurance floor therefore requires an authority outside the particular execution whose validity it constrains.

Depending on the domain, that authority may derive from:

- legislation;
- regulation;
- institutional policy;
- formally approved risk appetite;
- contractual obligation;
- safety specification;
- externally governed technical standard.

This does not mean that A_G must be permanently static.

Regulations change. Policies change. Evidence standards change. New mechanisms can discharge assumptions that previously required trust.

It means that a change to A_G is a **governance event**, not an execution decision. The same support and the same covering can fail $\models$ under one floor version and succeed under another. That flip is not an execution result. It is a change of the requirement set.

An agent may choose:

$$x_i$$

and may construct:

$$A_g^i$$

It must not, in the same ungoverned decision, choose the floor, choose the covering, or define the floor from the support it has just built.

If it may replace A_G by the empty requirement set, every support satisfies:

$$A_g^i \models A_G$$

If it may weaken covering, a public inference satisfies an authoritative-identity requirement without an approved source.

If the floor is defined as whatever this A_g^i already covers, then:

$$A_g^i \models A_G$$

holds for every A_g^i , and Boundary Sufficiency collapses to:

$$P(G_a, x_i \mid A_g^i)$$

This creates a separation between **goal authority** and **execution autonomy**. Goal authority fixes A_G and the covering. Execution autonomy may choose x_i and A_g^i .

10 Actual execution boundary A_i

The wider execution boundary remains important.

Let:

$$A_i$$

contain all material assumptions and dependencies introduced or relied upon by the actual execution.

Boundary Sufficiency does not imply:

$$A_g^i \subseteq A_i$$

That inclusion is hygiene, not a theorem of P or of $\models$. Discharge requires it. It may fail: the declared support can name assumptions the execution never used.

When hygiene holds, the proof-support set is typically a proper subset of the execution boundary:

$$A_g^i \subset A_i$$

An execution may invoke:

- a model;
- a planner;
- a retrieval service;
- an external registry;
- a document parser;
- a human reviewer;
- an identity provider;

- a policy evaluator;
- an execution service.

Some of these dependencies may be necessary for proving G_a . Others may be operationally present without being material to that particular proof.

The distinction between A_g^i and A_i allows both to remain visible.

This avoids two opposite errors.

First, treating every technical dependency as equally relevant to every assurance claim produces an unhelpfully large boundary.

Second, considering only the assumptions directly mentioned in the proof can hide operational dependencies capable of invalidating the proof or introducing other consequential risks.

Both boundaries matter, but they answer different questions.

11 Goal-relative trust residue

The distinction among A_G , A_g^i , and A_i also refines the concept of trust residue.

The original construction (Betts, 2026) defines:

$$R(A) = a \in A : \text{Need}(T, a \mid A \setminus a)$$

For an agentic execution there are now several useful residues.

11.1 Execution residue

$$R(A_i)$$

is the residue of the execution boundary A_i .

11.2 Goal-relative residue

$$R(A_g^i)$$

is the residue of the proof-support set used for this execution. That set need not be a minimal sufficient set.

This is particularly important for consequential decisions.

If:

$$a \in R(A_g^i)$$

then acceptance of G_a for execution x_i remains dependent on trust in a .

The Trust Displacement Result (Betts, 2026) still applies. From:

$$P(G_a, x_i \mid A_g^i)$$

it does not follow that:

$$R(A_g^i) = \emptyset$$

Proof of the goal relative to A_g^i is not global elimination of residual trust in that support. The local-discharge conjunct of Betts (2026), $\neg \text{Need}(T, x \mid A)$, is not restated here, because x_i is not a member of A_g^i . The result is limited in the same way as in Betts (2026): it does not assert that the residue is always nonempty.

$A_g^i \subseteq A_i$ does not imply:

$$R(A_g^i) \subseteq R(A_i)$$

Residue is not monotone. An assumption still needed in A_g^i may be discharged by a dependency that sits in A_i but not in A_g^i . The goal-relative residue is then not a subset of the execution residue.

What does follow is only:

$$R(A_g^i) \subseteq A_g^i \subseteq A_i$$

The inclusion of residues would hold if Need were context-independent. This paper does not assume that.

11.3 Trust inherited from the mandatory floor

Because A_G is now a set of assurance requirements rather than concrete execution dependencies, $R(A_G)$ should not be treated as a residue in the same sense as $R(A_g^i)$.

Instead, a mandatory requirement may constrain the kinds of concrete dependencies permitted to satisfy it. If the institution requires evidence from an authoritative registry, for example, the instantiated support in A_g^i may contain that registry and the resulting residue may include trust in its authority or operation.

The relevant object remains:

$$R(A_g^i)$$

but some members of that residue are present because of the way A_G is instantiated.

This exposes trust embedded in the institution's definition of sufficient assurance without confusing the semantic requirement with the concrete dependency that satisfies it.

Even a perfectly verified autonomous execution may therefore bottom out in trust inherited from the governing assurance floor.

Display that as two members of one support. Let A_g^i contain an agent-chosen inference and an institution-chosen registry. The inference may fail Need relative to the rest of A_g^i and drop out of the residue, while the registry remains residual. Then:

$$public_inference \notin R(A_g^i)$$

$$approved_registry \in R(A_g^i)$$

Local discharge of an agent dependency does not empty $R(A_g^i)$.

The trust has not necessarily remained in the agent. It may reside in the institution's chosen basis for accepting the goal.

12 The execution can construct its own boundary

Agentic orchestration introduces another difference from conventional workflow execution.

The process can introduce assumptions as a consequence of the path it selects.

Suppose an execution chooses to query an external registry. It introduces:

$$a_{registry}$$

Another execution asks a human compliance officer to resolve an ambiguity:

$$a_{human}$$

Another invokes a model-based classifier:

$$a_{model}$$

Another relies on a specialist external service:

$$a_{service}$$

The process therefore does not merely execute inside a predetermined boundary. It modifies the boundary. An action taken during execution can change the boundary. More explicitly, a run is a finite sequence of actions and boundaries:

$$A_i^0 \rightarrow x_i^1 \rightarrow A_i^1 \rightarrow x_i^2 \rightarrow A_i^2 \rightarrow \dots \rightarrow A_i^n$$

where actions taken during execution introduce, discharge or replace assumptions and dependencies.

The final proof-support set is meant to be selected from a boundary that the execution has partly constructed. The run does not force that selection. Discharge still requires $A_g^i \subseteq A_i$, and §10 records that the inclusion can fail.

This makes the proof boundary **endogenous to the execution**.

Adaptive systems have long been able to change configuration and dependencies at runtime. The distinctive problem is not that dependencies can change. It is that a dynamically generated execution can introduce, remove, or replace dependencies that subsequently form part of the support or wider boundary against which that same consequential execution is accepted.

That endogenous construction is a central risk in autonomous process assurance.

13 Dynamic Boundary Sufficiency

If A_i evolves during execution, Boundary Sufficiency cannot necessarily be checked only once at the beginning.

At the point at which the system seeks to discharge G_a , it must establish that the final proof-support set still satisfies the mandatory assurance floor:

$$A_g^i \models A_G$$

and:

$$P(G_a, x_i \mid A_g^i)$$

If execution has changed the meaning, provenance or validity of an element of A_g^i , the earlier satisfaction of the boundary condition may no longer be sufficient.

This suggests a dynamic form:

$$BS(G_a, x_i, t)$$

where Boundary Sufficiency is evaluated at the point of consequential discharge.

For a long-running process, it may be necessary to preserve evidence that:

$$A_g^i(t) \models A_G(t)$$

under the policy and regulatory state applicable at the relevant time.

This creates a temporal assurance problem as well as a structural one.

14 The temporal dimension

Let:

$$(G_a(t), A_G(t), x_i(t), A_g^i(t), A_i(t), R(A_g^i(t)))$$

describe the assurance structure of instance i at system time t . The time argument is a function. It is not a superscript, so t is not an instance index and not a case type. Superscripts remain instance indices.

Several dimensions can change independently. Equality in one coordinate does not force equality in the others. In particular, any of

$$G_a, A_G, x_i, A_g^i, A_i, R(A_g^i)$$

may differ between t and $t + 1$.

A model update may change the generated execution without changing the goal.

A policy update may change A_G while the visible process remains identical.

A new data source may alter the proof boundary without changing the final decision.

A new formal proof or cryptographic mechanism may discharge a residual trust dependency while leaving the execution path unchanged.

Consequently:

$$x_i(t) = x_i(t + 1)$$

does not imply:

$$A_i(t) = A_i(t + 1)$$

and does not imply:

$$R(A_g^i(t)) = R(A_g^i(t + 1))$$

Likewise, identical outcomes do not imply equivalent assurance.

This means that assurance provenance must preserve not only what happened but the boundary relative to which the action was accepted at that time.

15 Two temporal scales

There are two distinct temporal questions.

The first is **execution time**. During one execution, indexed here by τ , the agent may construct and modify its boundary:

$$A_i^0 \rightarrow x_i^1 \rightarrow A_i^1 \rightarrow x_i^2 \rightarrow \dots \rightarrow x_i^n \rightarrow A_i^n$$

The question is whether the proof-support boundary remains sufficient as this particular execution develops.

The second is **system time**, indexed by t , across successive executions, model versions, policy versions and regulatory regimes.

The mandatory floor itself may change:

$$A_G(t) \text{ need not equal } A_G(t + 1)$$

while a class of observable decisions remains unchanged.

The two changes should not be collapsed. An execution may begin under one policy state and finish under another. A model or tool may change between two otherwise similar cases. A previously acceptable source may cease to satisfy a mandatory assurance requirement.

For long-running or consequential processes, assurance provenance therefore needs to identify both the boundary constructed during the execution and the version of the governing assurance floor against which discharge was accepted.

This creates a governance question that proof alone cannot answer: when A_G changes during an execution, which version governs discharge?

16 Goal-relative execution equivalence

Agentic systems require a notion of equivalence that does not depend on identical traces.

Two executions may be operationally different:

$$x_i \neq x_j$$

but equivalent with respect to a governing goal.

Write:

$$x_i \equiv_G x_j$$

when both executions satisfy G_a relative to their instance supports. The subscript on $\equiv_G$ names that relation. It is not a second goal:

$$P(G_a, x_i \mid A_g^i) \wedge P(G_a, x_j \mid A_g^j)$$

This is a relation on the pairs (x_i, A_g^i) and (x_j, A_g^j) , not on traces alone. It does not require $A_g^i = A_g^j$, and it does not require a shared support that proves both.

This is not ordinary trace equivalence. The executions may differ ($x_i \neq x_j$) and may use different evidence, different orderings and different mechanisms.

Goal-relative equivalence does not give assurance equivalence. It may be that:

$$x_i \equiv_G x_j$$

while:

$$R(A_g^i) \neq R(A_g^j)$$

The executions satisfy the same goal but rely on different residual trust. That is where trust-dominance becomes useful.

A further distinction is useful. Let $o(x)$ denote the externally observable outcome of execution x . Outcome equality does not give goal-relative equivalence, and goal-relative equivalence gives neither equal supports nor equal residues.

$$o(x_i) = o(x_j)$$

does not imply:

$$x_i \equiv_G x_j$$

$$x_i \equiv_G x_j$$

does not imply:

$$A_g^i = A_g^j$$

$$x_i \equiv_G x_j$$

does not imply:

$$R(A_g^i) = R(A_g^j)$$

Operational monitoring usually observes outcomes. Governance of dynamically generated execution requires the supports and the residues on the right.

17 Goal-relative trust-dominance

Suppose:

$$P(G_a, x_i \mid A_g^i)$$

and:

$$P(G_a, x_j \mid A_g^j)$$

and both satisfy Boundary Sufficiency:

$$A_g^i \models A_G$$

$$A_g^j \models A_G$$

Write:

$$A_g^j \prec_t A_g^i$$

when:

$$R(A_g^j) \subsetneq R(A_g^i)$$

This is a relation on the residues of the goal-supporting boundaries, not on the named supports, not on the traces, and not on the outcomes. It does not require:

$$A_g^j \subseteq A_g^i$$

The supports may be incomparable under inclusion while one residue is still a proper subset of the other. Comparable here means a shared universe of assumptions, so that residue inclusion is well-typed. Two architectures built on genuinely different assumption names need not be comparable at all.

Execution j then establishes the same governing goal while retaining a proper subset of the residual trust dependencies retained by execution i .

This does not establish that j is globally safer, cheaper, fairer or otherwise superior.

It establishes only that, with respect to the trust relation being analysed, its goal-supporting boundary retains less residual trust.

This is a more useful meaning of "more trustless" than a Boolean system label.

An agent may have several permissible ways of satisfying G_a . Where the residues are comparable under inclusion, trust-dominance is one principled dimension on which those alternatives can be evaluated.

18 Risk: valid proof, invalid boundary

The most important failure mode distinguished by the model can now be stated directly.

Proof validity is $P(G_a, x_i \mid A_g^i)$. Boundary insufficiency is that proof holding while the support actually used does not meet the floor:

$$P(G_a, x_i \mid A_g^i) \wedge \neg(A_g^i \models A_G)$$

$P(G_a, x_i \mid A_i)$ may also hold. That is the claim relative to the execution boundary. It is not this proof, and it does not put a covering support inside A_i .

The strong form is that no support drawn from the execution meets the floor:

$$\forall A \subseteq A_i, \neg(A \models A_G)$$

No acceptable A_g^i then exists inside A_i , so the proof inside discharge cannot be valid against a covering support.

A weaker operational form is also a failure. A sufficient support may exist inside the execution boundary, yet the system may still fail to use one:

$$(\exists A \subseteq A_i, A \models A_G) \wedge \neg(A_g^i \models A_G)$$

Here A_g^i is the support actually used. The existential A is an unused alternative inside A_i . The governance claim is invalid because the support used for the proof does not satisfy the mandatory assurance requirements.

This is **boundary insufficiency**.

It is particularly dangerous in autonomous systems because the output may carry all the visible characteristics of a proved or verified decision.

The failure is not necessarily:

- incorrect inference;
- incorrect execution;
- incorrect proof;
- failed policy evaluation.

The failure is that the proof answered a weaker question than the governing institution required.

That is a distinct assurance failure.

Ordinary logging, policy checks, or successful proof generation may look the same in this case as in a successful discharge. The model distinguishes them.

19 Risk: silent boundary substitution

A second failure occurs when the execution appears to satisfy the required boundary but silently substitutes one dependency for another.

Suppose A_G requires authoritative identity evidence.

An execution may replace an approved registry with a probabilistic identity inference from public data.

The resulting execution may still produce convincing evidence for identity, but the substitution changes the assurance structure.

If the governing requirement is represented semantically rather than merely as a component name, the question becomes whether the replacement genuinely discharges the same requirement.

This demonstrates why A_G cannot ultimately be only a syntactic set of implementation components.

Its members must represent **assurance requirements at an appropriate semantic level**.

Implementation choices may then discharge those requirements through different mechanisms.

This is analogous to the earlier requirement that trust-dominance operate over a shared universe of appropriately individuated assumptions.

20 Risk: self-justifying execution

A third failure is specific to systems with substantial autonomy.

If the system can:

- select x_i ;
- select A_g^i ;
- interpret G_a ;
- decide whether A_G has been satisfied; and
- authorise the resulting consequential action,

then the assurance loop may become self-justifying.

The system effectively argues:

I selected this path, selected the evidence against which the path should be judged, determined that the evidence was sufficient, and therefore authorised the result.

The precise collapse is this. If A_G is defined as the set of requirements already instantiated by A_g^i , then:

$$A_g^i \models A_G$$

is true for every such pair, and:

$$A_g^i \models A_G \wedge P(G_a, x_i \mid A_g^i)$$

if and only if:

$$P(G_a, x_i \mid A_g^i)$$

Formal proof inside that loop does not restore the floor. The floor is no longer an independent object.

The deterministic elements that define and enforce:

$$A_G$$

and validate:

$$A_g^i \models A_G$$

therefore become particularly important.

This is not an argument that every such validation must be conventional software. It is an argument that the authority establishing the minimum assurance requirement must not be silently collapsible into the same unconstrained decision that it is intended to govern.

21 Risk: residue migration

A fourth failure is subtler.

An architectural change may remove one trusted dependency while introducing another.
Suppose:

$$R(A_g^1) = a, b$$

and after a change:

$$R(A_g^2) = a, c$$

Neither:

$$R(A_g^2) \subset R(A_g^1)$$

nor:

$$R(A_g^1) \subset R(A_g^2)$$

holds.

The new architecture is not more or less trustless under $\prec_t$.

Trust has moved.

This is **residue migration**.

A scalar trust score may conceal the change by assigning both boundaries the same numerical value. A set-based representation exposes that a human approval dependency b , for example, may have been replaced by a model-provider dependency c .

For regulated systems, that change can be material even where the count of residual assumptions is unchanged and observable process performance improves.

22 Regulated client onboarding

The following is an illustration of the objects, not a demonstration that any particular onboarding system is safe. Need, covering, A_G and Acceptable_G are supplied, not derived.

Consider a simplified client type a .

Its governing goal might be:

$$G_a = \text{"client is eligible to be onboarded under the applicable legal and institutional requirements"}$$

The mandatory assurance floor might include semantic requirements for:

$$A_G = \text{identity, classification, sanctions, ownership, applicable_policy}$$

For client i , the facts may introduce additional assumptions:

$$E_i = \text{source_of_funds, enhanced_i_identity_resolution}$$

so that:

$$A_g^i = S_i \cup E_i$$

The agent constructs an execution:

$$x_i$$

which obtains identity evidence from provider p_1 , queries a sanctions source p_2 , traverses an ownership registry p_3 , asks a human to resolve one ambiguity and applies policy version v .

The actual execution boundary A_i also includes the orchestration model, retrieval mechanisms, policy evaluator, execution controls and other technical dependencies.

The consequential decision should not be accepted merely because the agent reports:

$$P(G_a, x_i \mid A_i)$$

The institution needs evidence that:

$$A_g^i \models A_G$$

and:

$$P(G_a, x_i \mid A_g^i)$$

It can then inspect:

$$R(A_g^i)$$

to understand where acceptance of this particular onboarding decision still depends on trust. Another client of the same type may follow:

$$x_j \neq x_i$$

with:

$$A_g^j \neq A_g^i$$

while still satisfying:

$$P(G_a, x_j \mid A_g^j)$$

This is legitimate variation if both executions satisfy Boundary Sufficiency. The illustration does not show that a mandated sequence has been retired. A required step can remain inside the goal or the floor, as §23 records. What the example keeps stable is the goal and its mandatory assurance floor.

23 From process specification to goal specification

This suggests a deeper architectural transition.

Traditional workflow engineering tends to specify:

perform steps 1, 2, 3, 4 and 5 in this order.

Agentic orchestration increasingly specifies:

establish G_a subject to these constraints.

That moves assurance from **procedural conformity** toward **goal-relative proof**.

The procedural approach obtains confidence by constraining the path.

The goal-relative approach permits path variation but requires each generated path to carry sufficient evidence that it satisfied the governing obligation.

This does not imply that process constraints disappear.

Some steps may themselves be mandated and therefore form part of A_G or the definition of G_a .

The important distinction is that variability is explicit rather than treated as an exception to an otherwise fixed process.

24 Proof-carrying execution

A consequential agentic execution should therefore produce more than:

$$result$$

and more than:

$$result + auditlog$$

Conceptually, it should carry an assurance object sufficient to reconstruct:

$$G_a$$

$$A_G$$

$$x_i$$

$$A_g^i$$

$$A_i$$

$$P(G_a, x_i \mid A_g^i)$$

$$R(A_g^i)$$

and, where material:

$$R(A_i)$$

This can be described as **proof-carrying execution**.

The analogy is with proof-carrying code (Necula, 1997). That work showed that an untrusted producer can supply machine-checkable evidence that an artefact satisfies a fixed safety policy. The present object is different, and the term is used as an extension of that pattern rather than as a claim that proof-carrying artefacts are new.

The concern is not merely whether supplied code satisfies a safety policy. It is whether a dynamically generated sequence of consequential actions satisfies a governing goal relative to a boundary that itself may have been partly constructed during execution.

The assurance object must therefore expose both the proof and the conditions under which the proof is meaningful.

25 Goal Discharge Condition

The framework can be summarised by a necessary condition on discharge.

Let:

$$\text{Discharge}(G_a, x_i)$$

be the claim that execution x_i may be accepted as having discharged governing goal G_a . That claim is relative to the instance support, the floor, and the residue policy. It is not a function of G_a and x_i alone.

Four conditions are required. The first three are independent of one another. Boundary hygiene is the fourth. It is not part of that independence result.

First, **proof validity**:

$$P(G_a, x_i \mid A_g^i)$$

Second, **floor satisfaction**:

$$A_g^i \models A_G$$

Third, **residual-trust acceptability**:

$$\text{Acceptable}_G(R(A_g^i))$$

Fourth, **boundary hygiene**:

$$A_g^i \subseteq A_i$$

A sound discharge claim holds only if all four hold. The converse is not claimed. Meeting the four conditions does not by itself constitute discharge. Further conditions, including those in §28, may still fail.

None of the three conditions entails the other two.

A proof may be valid while its boundary fails to satisfy the governing assurance floor:

$$P(G_a, x_i \mid A_g^i) \wedge \neg(A_g^i \models A_G)$$

The boundary may satisfy the governing floor while the proof itself fails:

$$A_g^i \models A_G \wedge \neg P(G_a, x_i \mid A_g^i)$$

And a proof may be valid against a sufficient boundary while the remaining trust is unacceptable for the consequence being authorised:

$$P(G_a, x_i \mid A_g^i) \wedge A_g^i \models A_G \wedge \neg \text{Acceptable}_G(R(A_g^i))$$

The rule deliberately does not require:

$$R(A_g^i) = \emptyset$$

A consequential decision may legitimately depend on residual trust. What matters is that the residue is explicit and acceptable under the governing policy.

That is not a relaxation of Betts (2026). It is Trust Displacement applied to A_g^i : a sound discharge policy may not treat P as emptying the residue. Empty residue remains possible. It is not required, and it is not implied by proof validity.

Proof establishes the property. Boundary Sufficiency establishes that the proof answers the question governance actually requires. Residual-trust acceptability determines whether the remaining trust is acceptable for the consequence being authorised.

26 Relationship to prior work

The constructions $P(x \mid A)$, $R(A)$ and $\prec_t$ (Betts, 2026) were developed independently of Bella's Goal Availability and minimal-trust framework (Bella, 2007). Subsequent examination of Bella (2007) revealed an important conceptual convergence.

Bella (2007) identifies assumptions required for a formal guarantee that a participant cannot itself verify, describing these as that participant's minimal trust. This provides independent confirmation of the underlying observation that formal guarantees do not remove every trust

dependency surrounding a claim. The existence of residual or minimal trusted assumptions is not claimed here as novel.

The constructions are not identical. Betts (2026) models proof explicitly as boundary-relative through $P(x \mid A)$, defines the remaining demand for trust as a residue $R(A)$, introduces $\prec_t$ as a partial ordering over comparable proof boundaries according to inclusion of their residual trust sets, and proves the Trust Displacement Result: local discharge of trust in the subject does not entail $R(A) = \emptyset$. This paper keeps R and $\prec_t$ on A_g^i . It restates displacement as:

$$P(G_a, x_i \mid A_g^i) \not\Rightarrow R(A_g^i) = \emptyset$$

because the execution is no longer a member of the support.

Bella's principle is goal availability: a formal guarantee a participant can invoke. The question this paper then asks is what happens when the execution itself is not invariant.

In an agentically orchestrated process:

$$x_i \neq x_j$$

may hold while both executions are intended to discharge the same governing goal G_a . Their proof-support boundaries may also differ:

$$A_g^i \neq A_g^j$$

The invariant therefore moves from process identity to goal satisfaction.

This paper extends Betts (2026) by separating the governing goal from the generated execution, introducing the mandatory goal-assurance floor A_G , requiring the instance-specific proof-support set to satisfy that floor, and treating the execution boundary as something that may be partly constructed during execution.

The central objects are therefore:

$$G_a$$

$$A_G$$

$$x_i$$

$$A_g^i$$

$$A_i$$

and:

$$R(A_g^i)$$

linked by:

$$P(G_a, x_i \mid A_g^i)$$

and:

$$A_g^i \models A_G$$

The contribution claimed here is a formally checked goal-relative assurance model for dynamically generated consequential execution. Discharge is an only-if with four conjuncts: proof validity, floor satisfaction, residual-trust acceptability, and boundary hygiene. The companion development checks that only-if, and shows that the first three are independent. It does not show four-way independence.

26.1 Runtime certification and dynamic assurance

Rushby (2008) addresses systems whose functionality may be modified or synthesised at runtime and moves part of certification toward runtime evidence and arguments. Dynamic safety- and assurance-case research similarly treats assurance arguments as objects that must evolve with the operating system and environment (Denney, Habli and Pai, 2015; Calinescu et al., 2018).

Rushby already writes certification as claims, evidence, and argument, with assumptions as explicit hypotheses of the claim, and with a separate judgement that the residual risk is acceptable. Calinescu et al., with Kelly, rebuild a Goal Structuring Notation argument, assumptions included, after a runtime reconfiguration. Those results are not this paper’s externally fixed floor, its covering relation, the result that hygiene is entailed by neither proof nor covering, or the result that support inclusion does not yield residue inclusion. The question formalised here is whether the concrete support of the proof for a particular generated execution satisfies an independently authoritative assurance floor, whether that support sits inside the execution boundary, and what trust remains after that proof.

26.2 Goal-oriented requirements and declarative process models

Goal-oriented requirements engineering already treats goals as the stable objects from which multiple implementations may be derived (Dardenne, van Lamsweerde and Fickas, 1993; van Lamsweerde, 2001). Declarative process models likewise permit many traces to satisfy a common constraint set (van der Aalst, Pesic and Schonenberg, 2009).

The paper therefore does not rest on a simple fixed-workflow-versus-agentic-goal dichotomy. Its narrower contribution is that dynamically generated executions may have different proof supports and different actual execution boundaries while satisfying the same governing goal, and that these differences remain governance-relevant even when outcomes are identical.

26.3 Runtime assurance

Runtime assurance architectures commonly enforce safety properties around advanced or untrusted controllers using monitors, verified safety mechanisms, or fallback controllers (Schierman et al., 2020).

The present paper asks a different question. Even if the execution demonstrably satisfies a property, what assumptions make that demonstration acceptable, did those assumptions meet the independently specified assurance requirement, and is the remaining trust acceptable for the consequence?

26.4 Proof-carrying code

Proof-carrying code demonstrates that an untrusted producer can supply machine-checkable evidence that an artefact satisfies a policy (Necula, 1997).

The paper extends that pattern from a supplied artefact toward an instance-specific execution whose assurance context may itself evolve during generation. “Proof-carrying execution” is used in that sense in §24: as an analogy and extension, not as a claim that proof-carrying artefacts are new.

26.5 Trust and trusted computing bases

Prior work has long made trust assumptions explicit. Rangan (1988; 1992) formalised the trust requirements of protocols and sought minimal trust axioms for distributed mechanisms. Birrell, Lampson, Needham and Schroeder (1986) made the trust relations implied by authentication structure visible, rather than assuming a single global trusted authority. Thompson (1984) showed that trust may sit outside the artefact under inspection: verifying source does not

establish the corresponding property of an executable if the compiler is compromised. McCune et al. (2008) treat TCB minimisation as establishing the same security-sensitive property on a smaller trusted base. Monniaux and Boulmé (2022) show that even a formally verified compiler still rests on a surrounding trust base, including language modelling and external algorithms. Bella (2007) identified a participant’s minimal trust as the assumptions required for a guarantee that the participant cannot itself verify.

These are antecedents for residual or minimal trusted assumptions, not competitors for $R(A)$ or $\prec_t$. The contribution is to make residual trust an explicit coordinate in the assurance model of an individual consequential execution and to relate it to an independently governed assurance floor.

The claim is not that previous computing systems lacked dynamic or probabilistic behaviour. Probabilistic algorithms, cryptography, compilers, distributed protocols, statistical systems and data analytics have long complicated assurance. The distinction considered here is that an autonomous system may select consequential actions and the paths by which those actions become justified. Boundary construction can therefore become part of operational decision-making.

Residual trust is consequently no longer only something identified when analysing a mechanism. It becomes part of the assurance state of an individual consequential action.

27 The extended assurance structure

The progression can now be stated compactly.

27.1 Fixed artefact or bounded mechanism

$$P(x \mid A)$$

27.2 Deterministic instance-specific process

$$P(G_a, x_i \mid A_g^i)$$

where:

$$x_i = x_a$$

but:

$$A_g^i$$

varies.

27.3 Agentic process

$$P(G_a, x_i \mid A_g^i)$$

where both:

$$x_i$$

and:

$$A_g^i$$

vary.

27.4 Boundary-constrained agentic process

$$A_g^i \models A_G \wedge A_g^i \subseteq A_i$$

and:

$$P(G_a, x_i \mid A_g^i)$$

27.5 Dynamic agentic process

$$A_i^0 \rightarrow x_i^1 \rightarrow A_i^1 \rightarrow \dots \rightarrow x_i^n \rightarrow A_i^n$$

with final discharge only if:

$$A_g^i \models A_G$$

and:

$$P(G_a, x_i \mid A_g^i)$$

and:

$$\text{Acceptable}_G(R(A_g^i))$$

and:

$$A_g^i \subseteq A_i$$

The unit of assurance has therefore become:

$$(G_a, A_G, x_i, A_g^i, A_i, R(A_g^i))$$

rather than simply:

$$process$$

28 Consequences for governance

The six-tuple

$$(G_a, A_G, x_i, A_g^i, A_i, R(A_g^i))$$

is not a single score. It produces at least six separate governance questions. They are not six names for one check. The model does not show that each can fail while the other five hold. Goal selection is taken as given. Discharge in §25 is an only-if on proof, floor satisfaction, residual-trust acceptability, and hygiene. Meeting that only-if is not a claim that the questions in this section have been answered. §25 already says that further conditions, including those below, may still fail.

28.1 Goal validity

Was the correct governing goal G_a selected for the case?

A later proof that x_i satisfies G_a is worthless if G_a was the wrong obligation. Treating a prospective client as a low-risk natural person when the facts required a beneficial-ownership goal is not a proof failure. It is a goal-selection failure. The model takes G_a as given. Governance must still decide that the given G_a is the one the case required.

28.2 Minimum-boundary validity

Was the correct mandatory assurance floor A_G applied?

Even with the right goal, the floor can be the wrong version, the empty set, or a floor defined from the support the agent already chose. Section 9 treated that last construction as circular: Boundary Sufficiency then collapses to proof. This question is therefore not whether some floor was named. It is whether the independently authoritative floor for this G_a , under the applicable policy version, was the one applied. An audit of x_i does not answer it. Equal traces can sit under unequal floors.

28.3 Execution validity

Did the generated trace x_i satisfy G_a ?

This is the question P asks. It is necessary. It is not the whole of discharge. A system can generate a convincing trace, even a valid proof of that trace, and still fail every other question in this section. The converse also holds. The floor can be met while the execution fails to establish the goal.

28.4 Proof-support sufficiency

Did A_g^i satisfy A_G and contain the additional case-specific support required for this instance?

This is floor satisfaction together with the instance extras of §6. Boundary Sufficiency requires both:

$$A_g^i \models A_G$$

and:

$$P(G_a, x_i \mid A_g^i)$$

The conjuncts are independent. A support can prove G_a while missing a mandatory requirement. A support can meet the floor while the proof fails. The strong failure of §18 is that no subset of A_i meets the floor. The weaker operational failure is that a sufficient support exists inside A_i but was not the one used.

28.5 Boundary integrity

Does A_i accurately expose the material dependencies introduced or relied upon during execution?

Hygiene requires:

$$A_g^i \subseteq A_i$$

That inclusion is not a theorem of P or of $\models$. The declared support can name assumptions the execution never used. The execution can introduce dependencies that never appear in A_i . If the recorded boundary is incomplete, later inspection of floor satisfaction and residue is inspection of a partial record. Observability of the trace is therefore not observability of the boundary.

28.6 Residual-trust acceptability

Is $R(A_g^i)$ acceptable for a decision of this consequence?

Proof does not empty the residue. Acceptable_G is a parameter, not a theorem of P . Two executions can be goal-equivalent and Boundary Sufficient while their residues differ, migrate, or fail the institution's policy. A scalar trust score can conceal that movement by assigning both residues the same number. The question is not whether residual trust is zero. It is whether the remaining set is explicit and acceptable for this consequence.

A system may succeed at five and fail at the sixth. The six questions therefore cannot be collapsed into a single dashboard indicator, a single proof object, or a single process-conformance check.

This is why observability alone is insufficient.

An audit log may accurately reconstruct x_i while saying nothing about whether the applicable A_G was used, whether A_g^i met that floor, whether A_i exposed the material dependencies, or whether the remaining trust residue was acceptable.

Likewise, a proof object may establish $P(G_a, x_i \mid A_g^i)$ while failing to establish that the correct G_a was selected, that the correct A_G was used, that the support sat inside the execution boundary, or that Acceptable_G held.

The failure modes of §§18–21 are particular ways these questions come apart. Boundary insufficiency is a failure of proof-support sufficiency. Silent substitution is a failure of minimum-boundary validity when the floor is only a component name. Self-justifying execution is a failure of authority over A_G . Residue migration is a change that proof and outcome equality need not detect.

Governance is therefore not merely evidence accumulation.

It is management of the relationship among these objects. The institution that can reconstruct the trace, and even check a proof, has not yet asked the other four questions. For consequential agentic execution, those questions remain part of the decision.

29 Open formal questions

Several aspects require further work.

29.1 Uniqueness of A_g^i

A proof may have multiple incomparable minimal support sets.

If:

$$A_g^{i,1}$$

and:

$$A_g^{i,2}$$

are both minimal, neither need be a subset of the other.

A second source of non-uniqueness appears under composition: two supports can both satisfy a union floor while remaining incomparable, even when one of them is a single shared covering of both requirements.

The framework may therefore need to represent a family of minimal proof-support sets rather than assume uniqueness.

29.2 Semantic assumptions

A_G cannot ultimately be only a set of component names.

The assurance requirement "authoritative identity evidence" may be discharged by several mechanisms. A semantic mapping between requirements and concrete dependencies will be required.

29.3 Composition

A complex goal may decompose:

$$G = G_1 \wedge G_2 \wedge \dots \wedge G_n$$

Each subgoal may have its own assurance floor. The floors compose by union:

$$A_G = A_{G_1} \cup A_{G_2} \cup \dots \cup A_{G_n}$$

Floor satisfaction splits over that union. For a candidate support A_g :

$$A_g \models A_G \leftrightarrow A_g \models A_{G_1} \wedge A_g \models A_{G_2} \wedge \dots \wedge A_g \models A_{G_n}$$

This does not determine a unique A_g . A single shared assumption can satisfy both requirements of a two-part floor, and a split pair can also. Both of the following can hold, with neither support a subset of the other:

$$A_g^{shared} \models A_G$$

$$A_g^{split} \models A_G$$

$$\neg A_g^{shared} \subseteq A_g^{split}$$

$$\neg A_g^{split} \subseteq A_g^{shared}$$

Sharing is a fact about concrete supports, not about how floors union.

What remains open is composition when each subgoal is proved relative to a different support: whether $P(G_1 \wedge G_2, x \mid A)$ follows from $P(G_1, x \mid A_g^1)$ and $P(G_2, x \mid A_g^2)$ with $A_g^1 \neq A_g^2$, and which of several sufficient supports should be used for the composed goal.

29.4 Temporal validity

Evidence and assumptions can expire.

A proof-support set sufficient at time t may not remain sufficient at $t + 1$.

The validity interval of both P and members of A_g^i may therefore need explicit representation.

29.5 Goal-relative residue ordering

Trust-dominance currently assumes a shared universe of appropriately individuated assumptions.

Cross-mechanism comparisons will require semantic equivalence between assurance requirements before set inclusion can be meaningful.

29.6 Policy over residue

$$\text{Acceptable}_G(R(A_g^i))$$

introduces a normative layer that proof alone cannot determine. It is a parameter of the framework, not a predicate derived from P or from $\models$.

This is likely where institutional risk appetite, regulation and governance policy enter the formal structure.

29.7 What the structure does not decide

Need, covering, the contents of A_G , and Acceptable_G are parameters. Once they are fixed, the framework relates proof, floor satisfaction, hygiene and residue. It does not prove that a given Need is the right one, that a given floor is the right floor, or that a given residue policy is the right policy.

The machine-checked identities and countermodels are a claim about the internal validity of the parameterised model. They are not a claim that a particular choice of those parameters correctly captures regulation, institutional policy, risk appetite, or a real onboarding system.

R is formed on A_g^i , not on A_G . Treating $R(A_G)$ as a residue in the sense of Betts (2026) is well-typed only if the floor has been collapsed onto assumption names.

There is no uniqueness theorem for A_g^i . Several incomparable minimals can exist for one proof, and several incomparable supports can satisfy one union floor.

30 Conclusion

Agentic orchestration changes the object that process assurance must govern.

In conventional automation, individual cases vary while an approved process remains substantially invariant. The process provides the stable object against which executions are judged.

In agentic automation, both the execution trace and its proof boundary may vary by instance. The execution may also introduce the dependencies on which its own acceptance later rests.

Goal-relative assurance therefore requires a stable element above the execution: the governing goal and its mandatory assurance floor.

The resulting model is:

$$A_g^i \models A_G \wedge A_g^i \subseteq A_i$$

with:

$$P(G_a, x_i \mid A_g^i)$$

and a goal-relative trust residue:

$$R(A_g^i)$$

The critical insight is that a valid proof is not sufficient if it proves the right-looking claim against the wrong boundary.

For consequential autonomous action, governance must establish four conditions: **proof validity**, **floor satisfaction**, **residual-trust acceptability**, and **boundary hygiene**. The first three are independent of one another. Hygiene is required separately and is not part of that independence result.

$$\text{Discharge}(G_a, x_i)$$

only if:

$$P(G_a, x_i \mid A_g^i) \wedge (A_g^i \models A_G) \wedge \text{Acceptable}_G(R(A_g^i))$$

with $A_g^i \subseteq A_i$. Meeting those conditions is necessary. It is not the definition of discharge. The unit of assurance becomes:

$$(G_a, A_G, x_i, A_g^i, A_i, R(A_g^i))$$

That six-tuple is the unit of a formally checked goal-relative assurance model, not merely a process definition plus a proof. A companion development checks the model's identities and non-entailments.

This moves residual trust from the margins of system assurance into the decision itself.

The paper does not claim that runtime assurance, goal-oriented specification, dynamic assurance, proof-carrying artefacts, or residual trusted assumptions are themselves new. The contribution is their particular decomposition around dynamically generated consequential execution, and especially the proposition that proof validity is insufficient unless the support of that proof satisfies an independently authoritative assurance floor.

The question is no longer only whether the generated execution achieved its goal.

It is whether it achieved that goal **against a boundary sufficient to justify the consequence**.

31 Glossary

This glossary is a subset of the series master glossary. Entries here are the senses fixed in this paper. It also uses $P(x \mid A)$, $\text{Need}(T, x \mid A)$, $R(A)$, local discharge, the Trust Displacement Result, and $\prec_t$ as defined in Betts (2026); those entries are not repeated except where this paper changes them.

Governing goal G_a . What must be true for cases of type a . System time is written as a function, $G_a(t)$, not as a subscript.

Execution x_i . What instance i actually did.

Actual execution boundary A_i . The material assumptions and dependencies the execution introduced or relied upon.

Mandatory goal-assurance floor A_G . The assurance requirements the governing definition of G_a imposes on every acceptable proof of that goal. A change to A_G is a governance event, not an execution decision. $R(A_G)$ is not a residue in the same sense as $R(A_g^i)$.

Instance-specific proof-support set A_g^i . A candidate set of concrete assumptions and dependencies used as the boundary of the proof claim. Not the unique sufficient set, and not defined by minimality. $A_g^i = S_i \cup E_i$ splits floor-instantiating support from assumptions introduced by the facts of the instance.

Covering, $\models$. $A_g^i \models A_G$ when every requirement in A_G is instantiated by some assumption in A_g^i .

Boundary hygiene. $A_g^i \subseteq A_i$. Required for discharge. Not entailed by P or by $\models$.

Boundary Sufficiency. $A_g^i \models A_G \wedge P(G_a, x_i \mid A_g^i)$. Necessary for discharge. Not discharge. $BS(G_a, x_i, t)$ is that condition at the point of consequential discharge.

Goal authority and execution autonomy. Goal authority fixes A_G and the covering. Execution autonomy may choose x_i and A_g^i .

Execution residue $R(A_i)$ **and goal-relative residue** $R(A_g^i)$. Residues of the execution boundary and of the proof-support set. $A_g^i \subseteq A_i$ does not imply $R(A_g^i) \subseteq R(A_i)$.

Trust-dominates, $\prec_t$. Written on the boundaries: $A_g^j \prec_t A_g^i$ when $R(A_g^j) \subsetneq R(A_g^i)$. The relation compares residues. It does not require $A_g^j \subseteq A_g^i$.

Trust Displacement Result, as restated. $P(G_a, x_i \mid A_g^i) \not\Rightarrow R(A_g^i) = \emptyset$. Local trust discharge remains $P(x \mid A) \rightarrow \neg \text{Need}(T, x \mid A)$, as in *Trust and Proof*. It is not restated for x_i , because the execution is not a member of the support.

Endogenous boundary. A boundary partly constructed by the execution later accepted against it.

Execution time τ and system time t . τ indexes change inside one run. t is system time, written as a function argument, as in $A_G(t)$ and $A_g^i(t)$. It is not a superscript and not a case type.

Process identity and goal satisfaction. The invariant moves from sameness of process to satisfaction of one governing goal by executions that may differ.

Procedural conformity and goal-relative proof. Confidence from a constrained path, against confidence from evidence that a generated path satisfied the goal.

Discharge(G_a, x_i). The claim that x_i may be accepted as having discharged G_a . It holds only if proof validity $P(G_a, x_i \mid A_g^i)$, floor satisfaction $A_g^i \models A_G$, residual-trust acceptability $\text{Acceptable}_G(R(A_g^i))$, and boundary hygiene $A_g^i \subseteq A_i$ all hold. The first three are independent of one another. Hygiene is the fourth condition and is not part of that independence result. Meeting the four conditions does not by itself constitute discharge. Acceptable_G is a parameter.

Boundary insufficiency. A valid proof whose support does not meet the floor.

Silent boundary substitution. A requirement in A_G appears met by a mechanism that does not discharge it.

Self-justifying execution. The same decision selects the path, the support, and the reading of the floor, so covering collapses.

Residue migration. Residual dependencies are replaced, and neither residue is a proper subset of the other, so $\prec_t$ does not apply.

Proof-carrying execution. An execution that carries the goal, the floor, the trace, both boundaries, the proof, and the residues needed to reconstruct the claim.

32 References

- Bella, G. (2007) *Formal Correctness of Security Protocols*. Berlin: Springer. DOI: 10.1007/978-3-540-68136-6.
- Betts, R. (2026) ‘Trust and Proof’. Riley Betts Ltd. Available at: <https://rileybetts.ai/papers/trust-and-proof>
- Birrell, A.D., Lampson, B.W., Needham, R.M. and Schroeder, M.D. (1986) ‘A global authentication service without global trust’, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 223–230.
- Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M.U., Habli, I. and Kelly, T. (2018) ‘Engineering trustworthy self-adaptive software with dynamic assurance cases’, *IEEE Transactions on Software Engineering*, 44(11), pp. 1039–1069. DOI: 10.1109/TSE.2017.2738640
- Dardenne, A., van Lamsweerde, A. and Fickas, S. (1993) ‘Goal-directed requirements acquisition’, *Science of Computer Programming*, 20(1–2), pp. 3–50. DOI: 10.1016/0167-6423(93)90021-G

- Denney, E., Habli, I. and Pai, G. (2015) ‘Dynamic safety cases for through-life safety assurance’, *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*, vol. 2, pp. 587–590. DOI: 10.1109/ICSE.2015.199
- McCune, J.M., Parno, B., Perrig, A., Reiter, M.K. and Isozaki, H. (2008) ‘Flicker: an execution infrastructure for TCB minimization’, *Proceedings of EuroSys 2008*.
- Monniaux, D. and Boulmé, S. (2022) ‘The trusted computing base of the CompCert verified compiler’, in *Programming Languages and Systems: ESOP 2022*. Lecture Notes in Computer Science, 13240. Cham: Springer, pp. 204–233. DOI: 10.1007/978-3-030-99336-8_8.
- Necula, G.C. (1997) ‘Proof-carrying code’, *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL ’97)*, pp. 106–119. DOI: 10.1145/263699.263712
- Rangan, P.V. (1988) ‘An axiomatic basis of trust in distributed systems’, *Proceedings of the IEEE Symposium on Security and Privacy*. DOI: 10.1109/SECPRI.1988.8112.
- Rangan, P.V. (1992) ‘An axiomatic theory of trust in secure communication protocols’, *Computers & Security*, 11(2), pp. 163–172. DOI: 10.1016/0167-4048(92)90043-Q.
- Rushby, J. (2008) ‘Runtime certification’, in Leucker, M. (ed.) *Runtime Verification: RV 2008*. Lecture Notes in Computer Science, 5289. Berlin: Springer, pp. 21–35. DOI: 10.1007/978-3-540-89247-2_2
- Schierman, J.D., DeVore, M.D., Richards, N.D. and Clark, M.A. (2020) ‘Runtime assurance for autonomous aerospace systems’, *Journal of Guidance, Control, and Dynamics*, 43(12), pp. 2205–2217. DOI: 10.2514/1.G004862
- Thompson, K. (1984) ‘Reflections on trusting trust’, *Communications of the ACM*, 27(8), pp. 761–763.
- van der Aalst, W.M.P., Pesic, M. and Schonenberg, H. (2009) ‘Declarative workflows: balancing between flexibility and support’, *Computer Science — Research and Development*, 23(2), pp. 99–113. DOI: 10.1007/s00450-009-0057-9
- van Lamsweerde, A. (2001) ‘Goal-oriented requirements engineering: a guided tour’, *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE 2001)*, pp. 249–263. DOI: 10.1109/ISRE.2001.948567