Searched, not verified

Kenneth Pernyér, Reflective Labs
Stockholm, Sweden
kenneth.pernyer@reflective.se · reflective.se · ORCID 0009-0006-4543-1156

Between "the model argued it is safe" and "a machine-checked proof exists" there is a wide and useful middle, and most organizational assurance questions live in it. This paper describes Soter, which puts an SMT solver into the convergence loop of our substrate to answer one shape of question: *can any model exist that violates this encoded invariant?* Encoding the negation and asking for satisfiability turns a safety claim into a search for a counterexample, and a counterexample — unlike an argument — is a concrete assignment an engineer can read. The contribution we care about is not the integration but the epistemics around it. An `unsat` result is recorded as *searched* evidence and never as *verified*, and we give the two reasons precisely: the encoding gap between an organizational invariant and a formula, and the checker gap between a solver's claim and a proof an independent checker has validated. We argue that the most dangerous engineering error in this area is not an unsound solver but a status mapping that quietly reads `timeout` as `unsat`, and we show what the mapping must look like to prevent it. The path from searched to verified — proof certificates checked by a small independent kernel — is described and marked as not taken.

1. Introduction

An organization asking whether a policy is safe has, in practice, two unattractive options. It can ask people — including, lately, a language model — to reason about it, which produces an argument whose quality is invisible until it is wrong. Or it can commission a formal verification effort, which produces a genuine guarantee at a cost and timescale that makes it available to almost nothing.

Between those sits bounded symbolic search. An SMT solver (Barbosa et al., 2022, Moura and Bjørner, 2008) will not prove your system correct, but it will answer a surprisingly large family of questions of the form *is there any way this can happen?* — exhaustively, over the encoded domain, in seconds. Its ancestry is the combination of decision procedures (Nelson and Oppen, 1979) and its modern implementations are among the most heavily exercised pieces of software in computer science.

Soter puts one in the loop. It reads a query from the shared context, runs the solver, and proposes a report; the substrate decides whether the report becomes a fact (Pernyér, 2025a). Nothing in the extension has promotion authority, and the report is evidence rather than permission.

The paper is mostly about what that evidence is worth, because we think the field is careless about it in a way that will eventually cost someone a great deal.

2. The question

Soter asks one shape of question:

$$\exists M. \quad M \vDash \neg\varphi, \tag{1}$$

where φ is an encoded invariant and M ranges over models of the background theories. If the solver returns `sat`, it has produced an M — a concrete assignment of roles, amounts, states and flags under which the invariant fails. If it returns `unsat`, no such assignment exists within the encoding.

The inversion is worth dwelling on, because it is why this is useful to an organization rather than only to a verification team. We do not ask the solver to confirm that things are fine. We ask it to try, exhaustively, to break the claim, and we are interested in its failure to do so. This is the same posture as the adversarial review of the planning layer (Pernyér, 2025b), mechanized and made complete over a domain.

The questions that have paid for themselves are all separation questions in the sense of the classical integrity models (Clark and Wilson, 1987):

- Can any non-finance actor commit a high-value expense?
- Can any human-in-the-loop escalation bypass a hard deny?
- Can any combination of roles reach both `approve_payment` and `bypass_audit`?
- Can a generated policy context satisfy contradictory constraints?

Each has the property that a human reviewer will confidently say no, and each has a combinatorial structure — roles times delegations times states — where confident human noes are worth very little.

3. Two gaps

An `unsat` answer to Eq. 1 is strong and it is not a proof of anything about the organization. Two gaps

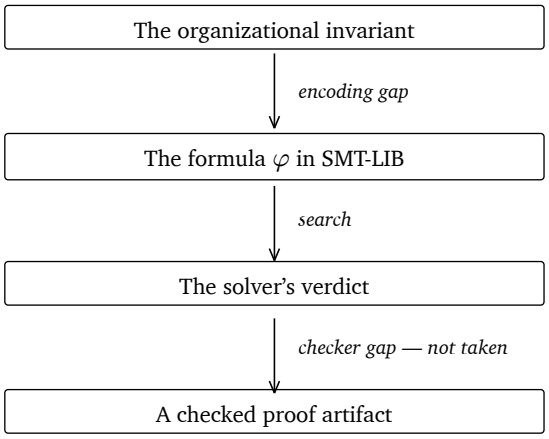

Figure 1 | What stands between an organizational claim and a verified one. Soter crosses the middle arrow and reports what it found. The first arrow is a human act of modelling and is not machine-checkable. The third arrow — a proof certificate validated by an independent checker — is described in Section 7 and is not implemented.

separate them, and naming both is the whole of our epistemic position.

The encoding gap. φ is a formula someone wrote. It stands to the organizational invariant in the same relation a specification stands to an intention, and no solver can tell you whether it stands there faithfully. An `unsat` on a formula that forgot delegation chains is a true statement about a formula and a misleading statement about the company. This gap is not reducible by better solvers; it is reduced by review, by fixtures, and by making the encoding an artifact the organization reads rather than an implementation detail it inherits.

The checker gap. Modern SMT solvers are enormous, highly optimized programs. When one answers `unsat`, that answer is a claim by a piece of software with a bug history, not a certificate anyone has checked. Proof-producing modes exist and an independent checker can validate the certificate against a small trusted kernel — the discipline proof assistants call the de Bruijn criterion, and the reason a Lean development (The mathlib Community, 2019) carries a different kind of weight. We do not do this today.

> **Definition (Evidence tiers). Searched**: a solver explored the encoded space and reported a status. The claim is about the encoding, and the report is unchecked. **Verified**: a proof artifact exists and has been validated by an independent checker. Soter emits only the former, and the vocabulary does not permit the former to be recorded as the latter.

That last clause is enforced rather than recommended: the evidence tier is part of the typed report, `Verified` is not among the values Soter can construct, and a consumer that wants a verified claim finds nothing in the context to satisfy it.

Status	What the engine may conclude
`sat`	A counterexample exists. For an invariant query this is a finding, with the witness attached.
`unsat`	No violation exists in the encoding. High-assurance searched evidence. Not proof.
`unknown`	The solver could not decide. Evidence exists; it may not satisfy a hard assurance requirement.
`timeout`	The budget was exhausted. Inconclusive, and operationally a failure.
`error`	Query or backend failed. Diagnostic only.

Table 1 | Status to evidence. The three lower rows are distinct from `unsat` and from each other, and none of them may be widened into it.

4. The mapping that must not collapse

Five statuses come back from the solver, and mapping them into evidence is where a system of this kind is most likely to be quietly unsafe.

The failure mode we design against is not an unsound solver. It is a status mapping that treats absence of a counterexample as absence of a violation. `unknown` and `timeout` both mean *we did not find one*, and the whole value of the exercise rests on the difference between *did not find* and *does not exist*.

> **Proposition (Non-collapse).** Let σ map solver statuses to assurance levels. If $\sigma(\text{timeout}) = \sigma(\text{unsat})$ then a query made harder — by widening the domain, adding a role, or lowering the budget — can raise the reported assurance of the same invariant. Any σ with this property is unsafe regardless of the solver's soundness.

The proposition is nearly trivial and the behaviour it describes is common: systems that report "no issues found" on a run that timed out have implemented exactly this σ. Under Table 1 a timeout is an operational failure, which is one of the honest exits of the substrate rather than a quiet pass (Pernyér, 2025a).

5. Determinism and identity

An SMT query is content-addressed. The SMT-LIB payload is validated and hashed with SHA-256, and the hash is the query's identity: the same question

asked twice is the same fact, not two facts that happen to agree.

This is what makes a solver Suggestor idempotent in the substrate's sense. Axiom A3 requires that running a Suggestor twice yields the same outcome, which is not automatic for a program with a time budget and a randomized search — re-running a hard query can return `unsat` once and `timeout` the next time. Content addressing separates the two concerns cleanly: the *query* is deterministic and identifies the fact; the *report* carries the status, the budget it ran under, and the backend version, so a differing outcome is a new report about a known query rather than a contradiction about an unknown one.

6. What a counterexample is worth

The asymmetry between the two useful answers is worth stating, because it runs opposite to the intuition that a positive result is the good one.

`unsat` is the answer an organization wants and the weaker of the two epistemically: it is a claim about the whole encoded space, and it inherits both gaps of Section 3. `sat` is the answer nobody wants and is nearly unimpeachable: the solver has produced a witness, the witness can be replayed against the real policy engine, and if the replay reproduces the violation then the encoding gap has been crossed in the only direction it can be crossed by machine.

We therefore treat a counterexample as a first-class artifact rather than a diagnostic. It is attached to the report, it is replayable, and the typed fixture that produced the first Arbiter high-risk query exists specifically so that a counterexample can be re-run against the live authorization path. A found violation is a test case the organization did not have.

7. Towards verified

The path from searched to verified is known and we have not walked it. A proof-producing configuration emits a certificate for `unsat`; a small independent checker validates the certificate without trusting the solver that produced it; the validated artifact is what a `Verified` tier would record. For our backend the checker exists; for others there are proof assistants with a long track record.

> **Open direction (A verified tier).** Define the artifact a `Verified` evidence tier would carry: the certificate, the checker and its version, the formula hash, and a binding from the formula to the organizational invariant it claims to encode. Determine whether the last of these can be made anything more than an attestation by a human.

We suspect it cannot, and that the encoding gap is permanently a matter of review rather than proof. If that is right, then `Verified` closes exactly one of the two gaps, and a system that reports it should say which.

8. Limitations

Everything here is bounded. The search is over an encoded domain with finite role sets, bounded amounts and a fixed set of state transitions; a real organization has none of those bounds, and choosing them is part of the encoding gap.

The extension is named for its purpose and not for its backend deliberately — `cvc5` is the first native binding, not the contract — but today one backend is what exists, and a cross-check between independent solvers, which would materially reduce the checker gap at low cost, is not implemented.

Native builds are opt-in. Default builds and ordinary CI use fake backends, so the common development path exercises the query and report surface without exercising a solver at all. This is the right trade for build times and it means our routine testing does not test the thing this paper is about.

Finally, the queries are hand-written. Generating them from a policy corpus — so that every new Cedar rule automatically produces the separation questions it should be asked — is the obvious next step and is not done.

9. Conclusion

An SMT solver in a governed loop buys something specific: exhaustive search over an encoded domain, in seconds, for questions where human confidence is worth little and formal verification is unaffordable. It does not buy proof, and the discipline that matters is refusing to let it be recorded as though it had.

Two gaps, named and kept open. The encoding gap is human and probably permanent. The checker gap is technical, closable, and not yet closed. Between them sits a tier called **searched**, which is more than an argument and less than a proof, and which we think most organizational assurance will always live in.

The engineering lesson is smaller and sharper. Do not let `timeout` mean `unsat`. Everything else in this paper is a matter of degree; that one is a matter of kind.

> **Code and correspondence.** The work described here is implemented, not sketched. The source behind every claim in this paper — the engine, the extension, the fixtures and the tests that hold the invariants — can be shared on request, including the parts that did not work. Write to kenneth.pernyer@reflective.se. We are equally interested in being told where the argument is wrong.

References

Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng,

Y., Tinelli, C., Zohar, Y., 2022. cvc5: A versatile and industrial-strength SMT solver. Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 415–442.

Clark, D.D., Wilson, D.R., 1987. A comparison of commercial and military computer security policies. Proceedings of the IEEE Symposium on Security and Privacy 184–194.

The mathlib Community, 2019. The Lean mathematical library. CoRR.

Moura, L. de, Bjørner, N., 2008. Z3: An efficient SMT solver. Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 337–340.

Nelson, G., Oppen, D.C., 1979. Simplification by cooperating decision procedures. ACM Transactions on Programming Languages and Systems 1, 245–257.

Pernyér, K., 2025a. Proposal, gate, commitment: deterministic convergence as a substrate for organizational decisions. Reflective Labs technical report.

Pernyér, K., 2025b. Institutionalized disagreement: how a plan earns the right to be run. Reflective Labs technical report.

A. The query surface, operationally

Q1 Query An `SmtQuery` carries an SMT-LIB payload, the background theories it assumes, and a budget. The payload is validated before it reaches a backend.

Q2 Identity The payload is hashed with SHA-256. The hash is the query's identity in the context, so re-asking is idempotent in the sense of the substrate's axiom A3.

Q3 Report An `SmtReport` carries the status of Table 1, the witness when the status is `sat`, the budget consumed, and the backend and its version. Its evidence tier is always **searched**.

Q4 Provenance Reports cross the Suggestor boundary as typed proposals with provenance, and are traced. A report is evidence for the convergence path and never a promotion decision.

Q5 Backends The native binding is isolated in a `*-sys` crate behind a feature. Default builds link no solver; fake backends serve ordinary CI, and real-solver runs are explicit.

B. Encoding a separation invariant

The first fixture encodes the expense-commitment invariant. Let R be a finite set of roles, $D \subseteq R \times R$ a delegation relation, amt a bounded integer, and $\mathrm{commit}(r, a)$ the predicate that role r commits amount a. The organizational invariant is that only finance may commit above a threshold τ, including through delegation:

$$\forall r, a. \quad \mathrm{commit}(r, a) \wedge a > \tau \implies \mathrm{fin}^*(r), \tag{B.1}$$

where fin^* is the reflexive-transitive closure of D restricted to finance roles. Soter asserts the negation of Eq. (B.1) together with the encoded policy and asks for satisfiability. A `sat` result returns a role, an amount and a delegation chain — a concrete route by which a non-finance actor reaches a high-value commitment.

Two encoding choices in that paragraph are exactly where the encoding gap lives: whether the closure is the right notion of "through delegation", and whether τ is the same threshold the live policy uses. Neither is decided by the solver, and both are recorded in the fixture so that a reviewer can disagree with them.

