Proposal, gate, commitment

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

Autonomy is cheap. A competent model, a tool loop and a queue produce an autonomous agent in an afternoon. Trust is the expensive part, and the usual way to earn it — observe the system, measure how often it is right, extrapolate — gives a statistical claim about the past. An organization needs a structural claim about the future. I describe Converge, a deterministic multi-agent runtime built on the opposite ordering: agents may propose anything, and only the engine decides. Shared state is an append-only context ordered by inclusion; agents are inflationary, idempotent, commuting maps over it; and every proposal crosses an explicit promotion gate that checks authority, schema and confidence before it becomes a fact. Nine invariants — monotonicity, determinism, idempotency, commutativity, termination, consistency, starvation freedom, confluence and observability — are enforced by the engine, not measured after it. I show that monotonicity and gate commutation give local confluence, that a budget-bounded well-founded measure gives termination, and that the two together yield a unique normal form by Newman's lemma. Determinism is therefore a theorem about the substrate, not a property I hope the scheduler preserves. Every run ends in one of four honest exits — converged, budget exhausted, policy blocked, escalated — each carrying its own provenance. The claim is bounded, and worth stating precisely: invalid states are not improbable. They are unrepresentable.

1. Introduction

Modern organizations fail at automation in a specific, unglamorous way: their systems do not agree with themselves. One service acts on a customer record that another has already superseded. An agent summarizes a decision that was never actually taken. An automation completes a workflow that a policy would have stopped, had anyone asked it. The failure is rarely a wrong prediction. It is a commitment made without the authority to make it.

That matters because a business is a commitment system. It binds authority and resources under uncertainty, across time, across functions, under governance and under audit. The question a language model answers well — *can the model produce a good answer?* — is not the question an organization has to answer, which is *can we act on this without creating hidden liabilities?* A model that is right 97% of the time is an excellent assistant and a poor institution. Institutions earn trust not because they rarely err, but because the conditions under which they can err are known, bounded and recorded.

Converge is my answer to the second question. It is a deterministic, context-driven multi-agent runtime in which agents read a shared context and propose facts, and the engine — not the agent — decides what gets promoted. The governing sentence is short: **agents are allowed to propose; only the system is allowed to decide.** Everything else in this paper is the machinery that makes that sentence enforceable instead of decorative.

1.1. It was never only about language models

People keep mistaking Converge for an orchestration layer over a language model. It is not, and the distinction is why the guarantees in this paper are available at all.

The architecture is a mixture of experts in the original sense: a gate routes a problem to whichever local expert is competent for it and combines what comes back (Jacobs et al., 1991). That is not the sense the phrase acquired once it was absorbed into language-model architecture, where the experts are feed-forward blocks inside one network and the gate is a softmax trained end to end (Dai et al., 2024, Shazeer et al., 2017). My experts are heterogeneous and external: a convex optimizer, an SMT solver, a vector index, a graph traversal, a fuzzy evaluator, a small locally-hosted model, and a frontier language model when the problem is genuinely linguistic. The gate is not learned. It is policy, and it's the promotion gate of Section 2.

Two things follow from that. The first is economic. A large class of the questions an organization actually asks — is this schedule feasible, does this configuration satisfy the constraints, are these two records the same entity, what does this policy actually entail — isn't best answered by a language model. It's answered better, more cheaply, and with a certificate, by something older. An SMT solver returns unsatisfiable with a proof. A solver returns an optimum with a dual certificate. A language model returns fluent text with a probability attached. Models from the big labs are

© 2025 Reflective Labs™. All rights reserved

important and will stay important, and I use them where the problem is one they're actually good at. But treating them as the right instrument for every subproblem is an artifact of their convenience, not their fit.

There's a sharper way to say the first point, and it's the sentence the rest of this paper defends: **language models belong in coordination, not in commitment.** They're very good at reducing the cost of reaching a shared picture — synthesizing evidence across sources, surfacing an assumption two stakeholders disagree about without knowing it, generating options with the trade-offs made explicit. They are not authoritative signatories. A commitment has to answer *who authorized this*, *on what evidence*, and *under what constraints* precisely; a probabilistic output can approximate those answers and cannot guarantee them. The failure that matters isn't a hallucination in a summary. It's a system where a model emits something commitment-shaped, the organization treats it as a commitment, and no commitment was ever actually formed. That's an architectural error, not a prompting error, and the fix is a type boundary rather than a better prompt: the probabilistic and the authoritative must not share a type, and there is no code path from the first to the second that skips the gate.

The second point is structural, and it's the one that makes the mathematics possible. Heterogeneous external experts can be *required* to commute, to be idempotent, and to declare what they read — obligations you can impose on a solver at its interface and cannot impose on a layer inside a network. The nine invariants of Section 3 hold precisely because the experts sit outside the engine, behind a gate, rather than inside a model, behind a softmax. Everything I prove in this paper rests on that choice.

1.2. Contribution

I claim no new mathematics. The results I lean on — the Knaster–Tarski fixed-point theorem (Tarski, 1955), Newman's lemma (Newman, 1942), the determinacy of Kahn networks (Kahn, 1974) — predate the field they're applied to here by decades. The contribution is the choice of substrate: a multi-agent runtime for organizational decisions should be built so these theorems *apply*, and the price of applying them — append-only context, commuting Suggestors, an explicit gate, and a run that halts rather than guesses — is one an organization should be glad to pay.

Section 2 defines the substrate and the three-stage lifecycle from observation to fact. Section 3 states the nine invariants. Section 4 formalizes the convergence loop and its four exits, and Section 5 derives uniqueness of the normal form. Section 6 places Converge against the layer above it and against the papers that follow this one, and Section 8 says plainly what the guarantee does not cover.

2. The substrate

Let $\mathcal{F}$ be a set of *facts*: immutable, attributed, timestamped claims about the organization. A *context* is a finite set of facts, and contexts are ordered by inclusion.

> **Definition (Context lattice).** The context lattice is the complete lattice $(\mathcal{K}, \subseteq, \cup, \cap)$ with $\mathcal{K} = \mathcal{P}(\mathcal{F})$. I write $K \sqsubseteq K'$ for $K \subseteq K'$ and read it as *K' is at least as informed as K*.

Think of it as a room of experts around a shared whiteboard. Each expert reads the board and proposes additions. Nobody may erase. When nobody has anything new to say, the room has converged. The whiteboard is the context, the experts are Suggestors, and the discipline that makes the room an institution rather than a meeting is that no expert writes on the board directly.

2.1. Three stages, two kinds of object

The expert that reads that whiteboard is a **Suggestor**, and it's the only kind of participant the engine knows about.

> **Definition (Suggestor).** A Suggestor is a map $s : \mathcal{K} \to \mathcal{P}(\mathcal{P})$ from a context to a set of proposals. At registration it declares what it reads, what it accepts, what it costs and what it produces. It has no other interface: it cannot write to the context, and it cannot address another Suggestor.

A Suggestor proposes; only the engine extends. Write $\delta_s(K) = s(K)$ for the proposals and Φ_s for their promotion into the context. Letting p range over $\delta_s(K)$,

$$\Phi_s(K) = K \cup \{ f_p : \gamma(p, K) = \text{promote} \}. \quad (1)$$

A proposal $p \in \mathcal{P}$ is a tuple (f_p, s, R_p, c_p): a candidate fact, the identity and version of the Suggestor that produced it, the subset $R_p \sqsubseteq K$ it actually read, and a confidence $c_p \in [0, 1]$. Proposals are drafts; facts are commitments. The boundary between them is the only place in the system where authority gets exercised.

Confidence isn't decoration — it's what lets heterogeneous experts share one context without a scheduler arbitrating between them. A greedy heuristic that finds a good but unproven answer caps its confidence below one, because it can't prove optimality. A solver that proves optimality reports one. Both proposals may get promoted, both may sit in the context at once, and downstream Suggestors — including the language model that eventually writes the explanation — select by confidence. A fast approximate answer and a slow exact one coexist rather than race: the Formation doesn't wait for the solver, and

© 2025 Reflective Labs™. All rights reserved

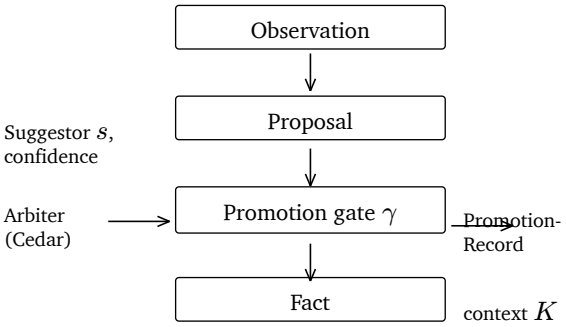

Figure 1 | The three-stage lifecycle. A Suggestor turns observation into a proposal; the promotion gate γ turns a proposal into a fact, or does not. Authority is evaluated at the gate, never inside the Suggestor, and every decision the gate makes — promote, block, escalate — is written to a `PromotionRecord` with full provenance.

it doesn't have to discard the solver's answer when it arrives.

2.2. Context is the API

Suggestors don't talk to each other. No message, no call, no handoff — a Suggestor reads the context and proposes to the context, and that's the whole of its social life.

This is the architectural decision most multi-agent systems get wrong, and it fails in a slow, expensive way rather than an obvious one. When agents address each other directly, a participant's behavior comes to depend on who spoke to it, in what order, and with what phrasing. Explainability degrades from *which facts led to this decision* to *which agent said what to whom, and when*. Evaluation gets brittle, because the unit under test is a conversation rather than a function of context.

Here's the mathematical consequence, and it's the one this paper actually needs. If s_1 could send a message to s_2, the order they ran in would be observable in the result, and the commutativity invariant of Section 3 would be false by construction — not hard to enforce, just meaningless to state. Because the context is the only channel, each Φ_s is a function of the context alone, and commutativity becomes something I can require, check and rely on. Context is the API, and the invariants are what that decision buys.

2.3. The promotion gate

The gate γ is a total function from a proposal and a context to a decision.

$$\gamma : \mathcal{P} \times \mathcal{K} \to \{\text{promote}, \text{block}, \text{escalate}\}. \quad (2)$$

It applies three checks in order. *Authority*: is this Suggestor, at this version, permitted to assert this fact type in this context? *Schema*: does the candidate fact match the declared structure of its type? *Confidence*: is the proposal's confidence above the threshold this fact type and this policy require?

That authority check is a boundary, not just a step, so it's worth being precise about it. The gate **poses** the authority question. It doesn't answer it. Policy evaluation — in my deployment, Cedar (Cutler et al., 2024) — lives outside the foundation, as an extension that registers as a Suggestor like any other. The engine owns the governed promotion path and nothing else. Making the kernel a policy engine would put the thing most likely to change inside the thing that must not. A proposal that fails authority is blocked; one that fails confidence is escalated; one that fails schema is a defect in the Suggestor and gets rejected outright.

Two things follow from putting the gate outside the Suggestor rather than inside it. First, Suggestors can be sampled, non-deterministic, model-backed and occasionally wrong without threatening any invariant, because nothing a Suggestor produces is load-bearing until the gate has admitted it. Second, authority is enforced in exactly one place, so the answer to *who was allowed to do this?* is a policy evaluation with a recorded outcome, not an archaeology of call sites.

3. Nine invariants

The engine enforces nine invariants. They aren't design goals, and they aren't scored after the fact — each is a checkable predicate on the transition relation, and a run that violates one halts instead of continuing.

1. **Monotonicity** — context only grows.
2. **Determinism** — the same context yields the same result.
3. **Idempotency** — running a Suggestor twice yields the same outcome.
4. **Commutativity** — Suggestor order doesn't affect the final context.
5. **Termination** — every run converges in finitely many steps.
6. **Consistency** — no contradictory facts coexist in a context.
7. **Starvation freedom** — every eligible Suggestor eventually executes.
8. **Confluence** — parallel branches merge to a single truth.
9. **Observability** — every effect is recorded in the ledger.

Appendix A restates the nine in the operational form the engine checks; Appendix B sketches the two proofs. The first two are worth a closer look here, because they're the ones most often claimed and least often delivered.

3.1. Monotonicity

> **Axiom (Monotonicity).** For every Suggestor s and context K, $K \sqsubseteq \Phi_s(K)$, where Φ_s is the promotion

© 2025 Reflective Labs™. All rights reserved

of s's proposals into K. Context only grows: no edits, no deletions, and therefore no lost provenance.

Monotonicity is what makes a run auditable, but its real work is mathematical. Because each Φ_s is inflationary and order-preserving, the cycle operator

$$F(K) = K \cup \bigcup_{s \in S} \Phi_s(K) \tag{3}$$

is monotone on a complete lattice, so it has a least fixed point above any starting context K_0 (Tarski, 1955),

$$K^* = \bigcap \{X : K_0 \sqsubseteq X,\ F(X) \sqsubseteq X\}. \tag{4}$$

Eq. 4 is the closure of the organization's knowledge under its own Suggestors and its own policy: everything this fleet, at this version, under this policy, can conclude from what's currently known — and nothing else.

3.2. Determinism

Axiom (Determinism). The same context produces the same result. If $K \xrightarrow{*} K_1$ and $K \xrightarrow{*} K_2$ with K_1, K_2 irreducible, then $K_1 = K_2$.

Determinism here is a property of the *system*, not of any individual Suggestor. Individual Suggestors may sample, may call a non-deterministic model, may return different proposals on different days. What's deterministic is the fixed point reached from a given context, by a given fleet, at a given version, under a given policy — the same guarantee a Kahn network gives for dataflow, where the scheduling of nodes can't be observed in the output (Kahn, 1974).

4. The convergence loop

Definition (Formation). A Formation is a finite set of registered Suggestors run by the engine against one shared context until none of them can add anything — the fixed point of Eq. 3. A Formation is declarative: Suggestors are registered, not sequenced. There's no step one.

A cycle wakes every eligible Suggestor in the Formation, collects proposals, evaluates each at the gate, merges the promoted facts, and tests for convergence. Writing Δ_n for the set of facts promoted in cycle n,

$$K_{n+1} = K_n \cup \Delta_n, \quad \Delta_n \subseteq \mathcal{F}, \tag{5}$$

the run has converged at n when $\Delta_n = \varnothing$: every eligible Suggestor has spoken, and none of them added anything the context didn't already contain.

4.1. Termination is budgeted, not hoped for

Termination has two independent sources, and that's deliberate. The first is structural: by monotonicity the context only grows, so the set of proposals a Suggestor can still make that are new to the context keeps shrinking. Formally, let $\mu(K) \in \mathbb{N}$ count the *open questions* in K — facts marked as requiring elaboration and not yet elaborated. Every promotion resolves at least one open question, or gets discarded as a no-op, so μ strictly decreases along the step relation and is bounded below. Hence, for every K,

$$\exists n \le \mu(K) : \quad K \xrightarrow{n} K^* \not\to . \tag{6}$$

The second source is operational, and it doesn't trust the first: every run carries a budget $B = (t, \text{tokens}, \text{cycles})$, and the engine halts when any component runs out. A theorem about a measure is a good reason to believe a loop terminates. A budget is what you want when the theorem's premise turns out to have been mis-implemented.

Exit	Condition and what the operator receives
Converged	$\Delta_n = \varnothing$. Fixed point reached; decision complete, with the full derivation.
Budget exhausted	B spent before $\Delta_n = \varnothing$. Partial evidence surfaced honestly, marked partial.
Policy blocked	$\gamma = \text{block}$ on a proposal the decision depends on. Logged with the Cedar rule that blocked it.
Escalated	Confidence below threshold. Full paper trail handed to a named human authority.

Table 1 | The four honest exits. The set is exhaustive by construction: the loop has no other terminating branch, and no exit is silent. *Converged* is the only one that means the system is finished; the other three mean it stopped, and say why.

4.2. Four honest exits

A run doesn't simply end. It ends in one of exactly four ways, each a first-class outcome carrying trace, provenance and explicit authority.

What matters about Table 1 is what it leaves out. There's no fifth exit where the system produces a confident answer because the honest one wasn't available, and no path where a blocked proposal quietly gets downgraded to an assumption. Systems that can't stop safely eventually commit to something nobody authorized. The four exits are how Converge stops instead.

4.3. Consistency as a gate, not a report

Consistency is the one invariant I can't establish by construction, because contradiction is a property of

© 2025 Reflective Labs™. All rights reserved

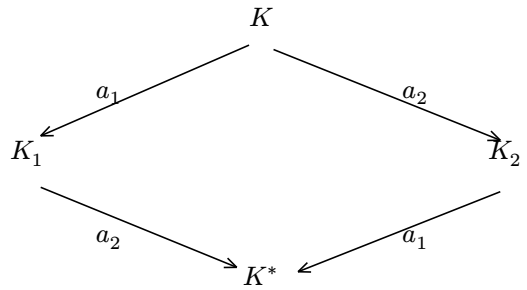

Figure 2 | Local confluence. Two Suggestors eligible on the same context K commute: the branch that promoted a_1 first and the branch that promoted a_2 first are joined by the remaining step, and both reach the same K^*.

content, not shape. The engine treats it as an admission condition on the merge: the facts promoted in a cycle are admitted only if the resulting context contains no contradictory pair under the organization's own contradiction relation $\perp$,

$$K_{n+1} = \begin{cases} K_n \cup \Delta_n & \text{if no pair } f \perp g \\ \text{escalate} & \text{otherwise,} \end{cases} \tag{7}$$

where f and g range over $K_n \cup \Delta_n$. Two proposals that answer the same question differently aren't a contradiction, and it's worth being exact about the difference. A greedy schedule at confidence 0.65 and a proven-optimal schedule at 1.0 are facts about two different objects — *a* feasible schedule and *the* optimal one — and they coexist happily, ordered by confidence. A contradiction is a pair of promoted facts that can't both be true of the organization, and $\perp$ is where the organization says which pairs those are.

Escalation is the point. A system that resolves contradictions silently has chosen, on the organization's behalf, which of two conflicting claims about it is true. Eq. 7 hands that choice back, which is the behavior you'd expect from a competent employee, not just a competent engine.

5. What the invariants buy

5.1. A unique normal form

Idempotency and commutativity give local confluence directly. If $K \to K_1$ by promoting a_1's proposals and $K \to K_2$ by promoting a_2's, both branches close on $K_1 \cup K_2$. That's the diamond of Figure 2, and it's why *which Suggestor went first* is an implementation detail rather than a semantic one.

Local confluence alone isn't enough — the classical counterexamples are all non-terminating — but with termination, it is.

> **Theorem (Newman (Newman, 1942)).** A terminating, locally confluent rewriting system is confluent, and every object has a unique normal form.

Invariant	Enforced by	On violation
Monotonicity	Append-only fact store	Reject write
Idempotency	Content-addressed facts	Deduplicate
Commutativity	Set-union merge of promoted facts	Reject Suggestor
Termination	Well-founded μ and budget B	Halt run
Consistency	Admission check, Eq. 7	Escalate
Observability	Ledger append per gate decision	Halt run

Table 2 | Six of the nine invariants and where they're enforced. *Enforced by* names the mechanism that makes the violation unrepresentable; where that's impossible, the engine halts rather than degrading silently.

Together with Eq. 6, this gives the determinism axiom as a *theorem* rather than an assumption: the normal form is unique, so scheduling, parallelism and retry can't change the answer. That's a stronger claim than a benchmark can make, and it's the claim an auditor actually needs.

5.2. Enforcement points

Table 2 is the honest version of a guarantee table. Three of the six are structural — a violating state simply can't be built — and three are checks that stop the run. I prefer the first kind, and I've tried to be clear about which is which.

6. The loop and the formations

Converge answers exactly one question: *is this proposal allowed to become a fact?* It doesn't decide which experts should have been asked, in what shape, or in what order, and it deliberately knows nothing about the layers above it. That separation has a short statement: **Converge is the loop; Organism is the formations.**

The loop is what this paper describes: cycle, gate, merge, test for a fixed point, exit honestly. It's the same loop no matter which experts get convened, which is what makes it provable. Converge owns the Formation as a mechanism — the contract a Suggestor signs, the gate it has to cross, the fixed point the engine runs to. What it doesn't own is the compo-

© 2025 Reflective Labs™. All rights reserved

sition: which specialists to convene for this intent, decomposed into which sub-problems, under what budget, and whether the goal is even well-formed. That's a different problem with a different character. It's selection under uncertainty, it benefits from learning, and it has no fixed point of its own. Putting it inside the engine would make the engine's determinism conditional on a learned policy, and the guarantee would evaporate. So it lives one layer up, in Organism — with the intent pipeline, the adversarial review that challenges a plan before it's committed, and the huddle where several models debate — and reaches Converge only as a set of registered Suggestors and a budget.

The specialists themselves are the subject of the papers that follow this one, and each returns to the substrate defined here rather than restating it: Organism on intent and formation composition; Ferrox on constraint solving, exposing CP-SAT, min-cost flow and MIP as Suggestors; Mnemos on knowledge and recall, with hybrid vector and lexical retrieval; Soter on bounded symbolic search over `cvc5`, stronger than argument and weaker than proof; Prism on closed-form analytics and, separately, on fuzzy inference; Crucible on models that must be fit to data rather than authored, and on training as a Formation in its own right; and Arbiter on authorization, where policy evaluation participates as a Suggestor instead of getting absorbed into the kernel. Each is an expert behind the gate. None of them changes the loop.

7. Related work

Conflict-free replicated data types get convergence from the same algebraic core: commutativity, associativity and idempotency of the merge (Shapiro et al., 2011). My context is a grow-only set CRDT with a gate in front of it, and the gate is the difference. A CRDT is designed never to reject, because in a replicated store rejection means a lost write. A decision substrate has to reject, because silent acceptance means a lost contradiction. Ledger ordering follows Lamport's happens-before (Lamport, 1978). Cedar supplies the policy language and its analyzability (Cutler et al., 2024), which matters here because a gate you can't reason about statically is a gate you have to test empirically — the exact regress this design is trying to escape. For the rewriting results I follow the standard treatment (Baader and Nipkow, 1998). The failure mode I'm designing against — systems that accrete implicit, unowned coupling until nobody can say what the system believes — is catalogued as technical debt in machine-learning systems (Sculley et al., 2015). On the mixture-of-experts lineage I take the older reading (Jacobs et al., 1991) rather than the architectural one (Shazeer et al., 2017), for the reasons given in Section 1.1.

8. Limitations

Four limits are worth stating plainly, because a guarantee whose boundaries are vague is a marketing claim.

First, monotonicity isn't free. A context that only grows will grow without bound, and the engine's compaction of resolved subgraphs is an engineering answer to a semantic problem, not a theorem. Compaction preserves the ledger, but a compacted context is no longer literally the object the proofs range over.

Second, the contradiction relation $\perp$ is supplied by the organization. The engine guarantees that detected contradictions escalate. It doesn't guarantee that the organization correctly said what a contradiction is.

Third, the invariants constrain the substrate, not the Suggestors. Nothing here stops a Suggestor from proposing a fact that's well-formed, authorized, consistent, confident, and wrong. The claim is bounded, and I'll state it in bounded form: invalid *states* are impossible. Invalid *beliefs* are merely attributable.

Fourth, determinism is relative to a pinned fleet, a pinned policy and a pinned set of provider versions. Change any of the three and you have a different system, whose fixed point may differ. That's a feature — it's what makes replay meaningful — but it means *deterministic* describes a configuration, not Converge in the abstract.

9. Conclusion

I've described a runtime whose safety argument doesn't depend on measurement. Model shared organizational state as an append-only context lattice, Suggestors as inflationary and commuting proposal generators, and authority as an explicit gate in front of the merge, and three classical results become applicable: a least fixed point exists (Tarski, 1955), every run reaches it in finitely many steps under a budget, and that fixed point is unique (Newman, 1942). Determinism is then a consequence rather than a hope, and the scheduler — the part of any multi-agent system that's hardest to test and easiest to get wrong — becomes semantically invisible.

The cost is real. Suggestors have to be written to commute, context has to stay append-only, contradictions have to escalate a run that a more permissive system would have finished, and every commitment pays for a policy evaluation and a ledger append. I think that's the right trade wherever a decision binds authority or resources, because there, an answer that's usually right and occasionally, silently wrong is worse than no answer at all.

None of this required a bigger model. It required deciding what the loop is, what's allowed to enter it, and what happens when it stops — and then refusing to let anything else in. Converge is the loop; Organism is the formations; the specialists are experts behind a

© 2025 Reflective Labs™. All rights reserved

gate. If a system can't explain why it did something, it can't be trusted to do it again. Autonomy is cheap. Trust is engineered. This is the engineering.

> **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

Baader, F., Nipkow, T., 1998. Term Rewriting and All That. Cambridge University Press.

Cutler, J.W., Disselkoen, C., Eline, A., He, S., Headley, K., Hicks, M., Hietala, K., Ioannidis, E., Kastner, J., Mamat, A., McAdams, D., McCutchen, M., Rungta, N., Torlak, E., Wells, A., 2024. Cedar: A new language for expressive, fast, safe, and analyzable authorization. Proceedings of the ACM on Programming Languages 8.

Dai, D., Deng, C., Zhao, C., Xu, R.X., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., Xie, Z., Li, Y.K., Huang, P., Luo, F., Ruan, C., Sui, Z., Liang, W., 2024. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. arXiv preprint.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E., 1991. Adaptive mixtures of local experts. Neural Computation 3, 79–87.

Kahn, G., 1974. The semantics of a simple language for parallel programming. Information Processing 74: Proceedings of the IFIP Congress 471–475.

Lamport, L., 1978. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21, 558–565.

Newman, M.H.A., 1942. On theories with a combinatorial definition of "equivalence". Annals of Mathematics 43, 223–243.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., Dennison, D., 2015. Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems (NeurIPS) 2503–2511.

Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M., 2011. Conflict-free replicated data types. Stabilization, Safety, and Security of Distributed Systems (SSS) 386–400.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., Dean, J., 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. International Conference on Learning Representations (ICLR).

Tarski, A., 1955. A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5, 285–309.

© 2025 Reflective Labs™. All rights reserved

A. The nine invariants, operationally

Each invariant is stated as the engine checks it. S is the Suggestor fleet at a pinned version, K ranges over contexts, γ is the promotion gate of Eq. 2, and $\to$ is the step relation of Eq. 5.

A1 Monotonicity For all $s \in S$, $K \sqsubseteq \Phi_s(K)$. Checked on every write: the fact store rejects any operation that isn't an append.

A2 Determinism For a pinned fleet, policy and provider set, the normal form K^* reached from K is unique. Established by A3–A5 and Newman's lemma; verified in CI by replaying recorded runs under shuffled schedules.

A3 Idempotency $\Phi_s(\Phi_s(K)) = \Phi_s(K)$. Enforced by content-addressing facts, so a re-proposed fact is the same fact, and the second promotion is a no-op.

A4 Commutativity For eligible s_1, s_2, $\Phi_1(\Phi_2(K)) = \Phi_2(\Phi_1(K))$. Enforced by the merge: a Suggestor whose effect isn't expressible as a set union of promoted facts can't be registered.

A5 Termination $\mu(K) \in \mathbb{N}$ strictly decreases on every non-discarded step, and independently the budget B bounds time, tokens and cycles.

A6 Consistency No $f, g \in K$ with $f \perp g$. Checked before the merge is committed, per Eq. 7; a violation escalates rather than resolves.

A7 Starvation freedom Every Suggestor eligible at cycle n is woken at some cycle $m \geq n$ before convergence is declared. Enforced by a fair scheduler over the eligible set.

A8 Confluence If $K \xrightarrow{*} K_1$ and $K \xrightarrow{*} K_2$ then there is K_3 with $K_1 \xrightarrow{*} K_3$ and $K_2 \xrightarrow{*} K_3$.

A9 Observability Every gate decision appends a `PromotionRecord` carrying the Suggestor identity and version, the context subset read, the proposal, the policy consulted and the outcome. A cycle with a missing record halts the run and marks it untrustworthy.

B. Proof sketches

B.1. Local confluence

Let s_1, s_2 be eligible at K, with $K \to K_1 = K \cup \Delta_1$ and $K \to K_2 = K \cup \Delta_2$. Monotonicity gives $K \sqsubseteq K_1$ and $K \sqsubseteq K_2$, and since eligibility and gate admission are themselves monotone in the context, s_2 remains eligible at K_1 and s_1 at K_2, with the same proposals admitted. Applying the other Suggestor yields

$$\Phi_2(K_1) = K \cup \Delta_1 \cup \Delta_2 = \Phi_1(K_2) = K_1 \cup K_2, \tag{B.1}$$

using idempotency to absorb any fact proposed by both. Eq. (B.1) closes the diamond of Figure 2, so $\to$ is locally confluent.

The monotonicity of gate admission is the load-bearing step, and it's an obligation on policy, not a property of set union: a Cedar policy that denies in a larger context what it permitted in a smaller one breaks the argument. The policy linter rejects rules like that.

B.2. Uniqueness of the normal form

By A5 the relation $\to$ is terminating, and by the previous sketch it's locally confluent. Newman's lemma (Newman, 1942) gives confluence, and a confluent terminating relation has unique normal forms: if $K \xrightarrow{*} K_1$ and $K \xrightarrow{*} K_2$ with both irreducible, confluence supplies a common K_3 reachable from each, and irreducibility forces $K_1 = K_3 = K_2$. Axiom A2 follows.

© 2025 Reflective Labs™. All rights reserved