A Licence-Flow Verdict on the Enclosure Moat

Testing our own non-monopolisation strategy against the software-licence corpus

Raazesh Sainudiin — An \|/*ias*/|\ Commons Continuum

2026-08-16

Abstract

Having tested an external impossibility claim – Wiggins's sovereignty trilemma – against the digital-law corpus of three jurisdictions, we now turn the same instrument on ourselves. VakeWorks publishes a layered licensing strategy for the IAS Commons and its Concordance service – an Apache-2.0 permissive core, an AGPL-3.0 network-copyleft division, and a proprietary edge for client-specific white-labelling – and asserts that this layering forms an anti-enclosure *moat*: no adversarial actor can take the published code, run it as a network service, and lock users out without returning its modifications to the commons. We test that assertion exactly as we tested Wiggins: on dual-verified official licence texts (Software Package Data Exchange (SPDX) License List cross-checked against each steward), with every clause typed before it is cited, and with an adversarial satisfiability check – here over a *licence-flow algebra* on the code-flow spine-graph, in which a combined work carries the join (most-copyleft) of its linked inputs and a JSON/process/network boundary cuts the flow. Casting an enclosure attempt as a satisfaction problem, we find: (i) under a *within-law* adversary the enclosure instance is unsatisfiable *only under a three-part core* – AGPL-3.0 Section 13 (Remote Network Interaction), our gate-placement discipline, and a value-architecture requirement that the strategic value be *intrinsic* to the AGPL-3.0 division (bound to modifying or linking it) rather than reachable by merely calling the unmodified division across a wire; drop the third and the classic open-core *wrap* – a proprietary front around the unmodified division – is an admissible enclosure (satisfiable); (ii) a *permissive-only* strategy – relying solely on Apache/BSD/MIT upstream – admits an enclosure trajectory (satisfiable), so the AGPL-3.0 division is not decorative but load-bearing; (iii) against a *violating* adversary who uses our code but hides it, no licence clause is decidable and the moat is not a satisfiability question but a deterrence condition, in which AGPL-3.0 is necessary but not sufficient, the binding constraints are detectability *and* enforcement capacity, and detection of our code inside a concealed service is a forensic capability distinct from the ownership-provenance the substrate already records. The verdict re-scopes VakeWorks's non-monopolisation pledge from an elaborate promissory instrument to its minimal sufficient core – the *three* commitments the moat actually needs – and locates precisely where a legal moat ends and provenance, detection, enforcement, and non-licence moats must begin. The value-intrinsic-versus-gate-everything tension is the live architecture decision this report exists to surface. As with the trilemma, the result is metacircular by design: the tool validates the licensing architecture of the substrate that produces the tool.

Contents

Introduction **2**

1 The licence-flow algebra **3**

2 The moat, decomposed **3**

3 Locating the core, and the two regimes **4**

4 Typed clauses **6**

5 Verdict **7**

6 The minimal sufficient pledge **7**

7 Three adversary classes **8**

8 Falsifiers **8**

9 Governance receipts **8**

10 Acknowledgments **9**

Introduction

In issue three of the *Being AVake Works* newsletter we put a viral claim – that no global company can simultaneously satisfy Chinese content control, European interoperability, and American decoupling – to the law itself, and returned a jurisdiction-explicit verdict: the contradiction is real, it has a single minimal unsatisfiable core, and the observed "three Apples" is its unique feasible resolution [Sa26c, Sa26b]. That was Concordance pointed outward, at another party's architecture.

This report points it inward. VakeWorks makes a public commitment – a non-monopolisation pledge – that the commons it builds cannot be enclosed: taken private, run as a locked service, and monetised without the improvements flowing back. That pledge rests on a concrete engineering choice, a three-layer licence strategy: a permissive Apache-2.0 core so anyone may build on it; an AGPL-3.0 division around the parts whose value is realised as a network service; and a proprietary edge for client white-labelling. The question we test is whether that layering actually does what the pledge promises – and, if it does, what the *smallest* version of the pledge is that still does it.

We test it as mathematics, not as intention, and with the same conservatism as before: no language model is the authority on what a licence says. Each licence is fetched on two independent paths – the SPDX License List and the licence steward (the Free Software Foundation for the GNU family, the Apache Software Foundation for Apache-2.0) – compared, hashed, and archived with its verification status [SPD26, Fre07a, Apa04, Fre07b, Ope99]. The reasoning layer is a *licence-flow algebra* over the code-flow spine-graph (§1): a typed computation in which linking propagates the most-copyleft licence and a network or process boundary stops it. We then cast an enclosure attempt as a satisfaction problem and reason adversarially – for every route by which an actor might enclose our value we try to construct an admissible licence-flow that achieves it, admitting the moat only where that construction provably cannot exist (§3). §1 fixes the algebra and the definitions; §2 states the moat as three sub-claims; §3 locates the minimal unsatisfiable core and separates the within-law regime from the violation regime; §4 types the load-bearing clauses; §5 states the verdict; §6 extracts the minimal sufficient pledge; §8 lists the falsifiers; §9 records the governance method. A plain-language account appears as issue four of the newsletter [Sa26d].

The apparatus is deliberately the same one used for the trilemma. There the enforcing State was an explicit parameter of the machine; here the *licensor* is – a relicensing move by an upstream vendor plays the role the State's obligation-assignment played before. The satisfiability-and-minimal-core method, the DET/INT typing, and the sigma-attested clause registry carry over unchanged. This is not reuse for economy alone: it is the claim that legal-architectural questions

of very different surface – jurisdictional compliance, licence enclosure – are one mathematical object under the jurisdiction-explicit model [Sa26a].

1 The licence-flow algebra

Definition 1.1 (Code-flow spine-graph and the licence lattice)**.** *A* code-flow spine-graph $G = (N, E)$ *has nodes* N *the code components of a system and edges* $E \subseteq N \times N$ *its dependency and data-flow relations. The* licence lattice $(\mathrm{Lic}, \sqsubseteq)$ *orders licences by* combines-into*: $a \sqsubseteq b$ when a work combining a-code and b-code must be distributable under b. The working chain for the zones in play is*

$$\text{BSD/MIT} \sqsubseteq \text{Apache-2.0} \sqsubseteq \text{LGPL} \sqsubseteq \text{GPL-3.0} \sqsubseteq \text{AGPL-3.0},$$

with proprietary *a separate absorbing element $\top'$, incomparable to the chain except that no copyleft licence may flow into it by linking. The join $a \sqcup b$ is the least licence a linked combination of a and b must carry (copyleft wins); pairwise compatibility caveats – Apache-2.0's patent clause against GPL-2.0-only, for instance – are recorded as edge annotations, not as lattice defects.*

Definition 1.2 (Edge type and effective licence)**.** *Each edge carries a type $t : E \to \{\mathrm{LINK}, \mathrm{GATE}\}$. A* LINK *edge is a compile-time, static, or dynamic link – shared address space, a derivative work – across which licence* propagates*. A* GATE *edge is a JSON, process, or network boundary – a subprocess call, an inter-process message, a wire protocol – across which licence does* not *propagate (mere aggregation). The* effective licence *of a node is the least fixed point*

$$\mathrm{Eff}(n) \;=\; L(n) \;\sqcup\; \bigsqcup \{\, \mathrm{Eff}(m) : (m \to n) \in E,\; t(m \to n) = \mathrm{LINK} \,\},$$

computed by propagating joins up the LINK*-subgraph only; a* GATE *edge contributes nothing and so resets the flow. It is well-defined because* Lic *is finite and* LINK*-reachability is acyclic modulo the gates. (This is the copyleft dual of the trilemma report's typed-inference rule, where a derived claim inherits the* meet *of its inputs' verification types: meet-for-honesty there, join-for-copyleft here, the same propagation with the boundary as the cut [Sa26b].)*

Definition 1.3 (Target, admissibility, and the enclosure moat)**.** *A* target assignment $T : N \to \mathrm{Lic}$ *names the licence each node is intended to ship under; VakeWorks's is $T(\mathit{core}) = \mathit{Apache\text{-}2.0}$, $T(\mathit{division}) = \mathit{AGPL\text{-}3.0}$, $T(\mathit{edge}) = \mathit{proprietary}$. A licence-flow is* admissible *iff* $\mathrm{Eff}(n) \sqsubseteq T(n)$ *for every node n; a violation* $\mathrm{Eff}(n) \not\sqsubseteq T(n)$ *is exactly a copyleft dependency reaching n along a* LINK *path, repairable by inserting a* GATE *on that path, relicensing $T(n)$ upward, or dropping the dependency. An* enclosure architecture *is an admissible flow in which an external actor delivers our value to users as a network service while placing a* GATE *so that the AGPL-3.0 network-copyleft obligation never reaches the served node – the users get the value, the commons gets nothing back. The* moat holds *iff no enclosure architecture is admissible: an adversarial unsatisfiability over the licence clauses and the flow. Its* minimal unsatisfiable core *is the smallest clause-and-obligation set whose removal would let an enclosure architecture through.*

2 The moat, decomposed

We test the pledge's promise as three sub-claims of increasing strength, against a licensor/competitor adversary whose moves are relicensing, acquisition-driven relicensing, hidden intermediate layers, and – in the third regime – outright violation.

Claim 1 (Within-law enclosure is unsatisfiable)**.** *Under the current three-layer target, no admissible licence-flow lets a compliant actor run our value as a network service while stripping the give-back obligation. (Decidable over the archived clause corpus; located in §3.)*

Claim 2 (The AGPL-3.0 division is load-bearing)**.** *A permissive-only strategy – the same core and division targeted at Apache-2.0 or another permissive licence – admits an enclosure architecture; hence the AGPL-3.0 layer is necessary to Claim 1, not a stylistic choice. (Decidable; a satisfiable counter-architecture is exhibited.)*

Claim 3 (Necessity across relicensing trajectories)**.** *No layering resting solely on upstream permissive licences keeps the moat across all adversarial relicensing trajectories a vendor may play on a component the commons depends on, whereas a layering that pins our own value behind the AGPL-3.0 division and gate discipline does. (This is the non-monopolisation-necessity conjecture; it is decidable only relative to a declared trajectory set, and is typed as a conjecture with falsifiers in §8.)*

3 Locating the core, and the two regimes

Proposition 3.1 (Permissive-only admits enclosure)**.** *If every VakeWorks node is targeted at a permissive licence (Apache-2.0, BSD, or MIT), an enclosure architecture is admissible.*

Proof. Permissive licences impose attribution and, for Apache-2.0, a patent grant, but place no obligation on a party who conveys only the *behaviour* of the software over a network without distributing its source [Apa04, Ope99]. An external actor forks the permissive core and division, adds private modifications, and serves the result to users behind a network GATE. Every edge into the served node is either permissive-to-permissive LINK (join stays permissive) or a GATE (no propagation), so $\mathrm{Eff}(\text{served}) = \text{permissive} \sqsubseteq T(\text{served})$ and the flow is admissible. The users receive the value; no clause compels return of the modifications. The enclosure architecture exists, so the instance is satisfiable. □

Definition 3.2 (Value-intrinsic)**.** *The strategic value is* value-intrinsic *to the AGPL-3.0 division when delivering it requires* modifying or LINK-ing *the division – the value is bound to the division's code – as opposed to being obtainable by* GATE-*calling the unmodified division across a wire (where the division is a wire-callable service and the value lives entirely in the caller).*

Proposition 3.3 (The AGPL-3.0 division makes enclosure unsatisfiable)**.** *If the strategic value resides in a division targeted at AGPL-3.0* and *is value-intrinsic to it (Definition 3.2), then no compliant enclosure architecture is admissible; the obstruction is AGPL-3.0 Section 13. Without value-intrinsic the proposition fails: the unmodified wrap below is an admissible enclosure.*

Proof. AGPL-3.0 Section 13 provides that a party who *modifies* the Program and makes it available to users *interacting with it remotely through a computer network* must offer those users the Corresponding Source of the modified version, at no charge, through a network server [Fre07a]. This is the one copyleft clause that a GATE does not defeat *for a modified deployment*: it is triggered by the network interaction, not by distribution of a binary. Under value-intrinsic, delivering the value requires modifying or LINK-ing the division, and there are exactly two ways to do so. *By* LINK*:* $\mathrm{Eff} = \cdots \sqcup \text{AGPL-3.0} = \text{AGPL-3.0}$, so the linked work inherits the network-copyleft and is not enclosed. *By modifying and serving behind a* GATE *(the software-as-a-service move that defeats GPL-3.0 and every permissive licence):* the GATE keeps the front's licence off the division, but Section 13 binds the *modified division's own* remote interaction regardless of internal boundaries – its users must be offered its source. Either way the value-bearing node owes source to its users, so admissibility would require $\text{AGPL-3.0} \sqsubseteq T(\text{served})$ with $T(\text{served})$ *below* AGPL-3.0 – impossible on the lattice. Re-implementing the value from scratch is clean-room competition, not enclosure of *our* code (§5). Hence, under value-intrinsic, no compliant enclosure architecture exists. □

Proposition 3.4 (The unmodified wrap enclosure (why value-intrinsic is needed))**.** *If the value is* not *value-intrinsic – if it is obtainable by* GATE*-calling the unmodified division – then an enclosure architecture is admissible even with the division at AGPL-3.0.*

Proof. Let the adversary ship a proprietary front F, $T(F) = \top'$, whose value-add lives in F, and reach our unmodified division D across a boundary, $F \xrightarrow{\text{GATE}} D$. By Definition 1.2 the GATE contributes nothing, so $\text{Eff}(F) = L(F) = \top' \sqsubseteq T(F)$: admissible. AGPL-3.0 Section 13 triggers only on a party who *modifies* the Program; D is unmodified, so the source-offer is vacuous (D's source is already public) and in any case borne by whoever runs D, never by F. F delivers our value to users as a closed service and owes the commons nothing. This is the classic open-core *wrap* – proprietary value built *around* the copyleft component, not inside it – and it is exactly the managed-service move (a hyperscaler wrapping an unmodified open-source engine) that AGPL-3.0 alone does not stop. It bites us specifically: our own gate-invariant makes the division wire-consumable so the Apache-2.0 core stays clean, and that same wire-consumability is what lets a competitor's front consume it. Hence value-intrinsic is a genuine third requirement, in tension with the gate-everything discipline, and not a wording nicety. □

Proposition 3.5 (Minimal unsatisfiable core)**.** *Taken against the* joint *strategy objective – a permissively-adoptable core* and *an enclosure-proof value – the load-bearing core is the triple*

$$M^\star = \{\ \text{AGPL-3.0 Section 13},\quad \textsf{gate-invariant},\quad \textsf{value-intrinsic}\ \},$$

where the gate-invariant is the design obligation core $\xrightarrow{\text{GATE}}$ *division (the core reaches the AGPL-3.0 division only across a wire boundary) and value-intrinsic is Definition 3.2. $M^\star$ is an unsatisfiable core in which each element is load-bearing: dropping any one admits an enclosure architecture. The three are load-bearing for distinct reasons – Section 13 and value-intrinsic together carry the moat (enclosure-UNSAT), and the gate-invariant carries the permissively-uncontaminated core – so the moat is not "two clauses wide" but a two-clause-plus-one-discipline core, the discipline being the harder half to hold.*

Proof. The triple secures the joint objective: Section 13 with value-intrinsic makes enclosure UNSAT by Proposition 3.3, and the gate-invariant keeps Eff(core) = Apache-2.0 (no LINK from the AGPL-3.0 division reaches the core), so the core stays freely adoptable. Minimality, each element load-bearing: *drop Section 13* – target the division at GPL-3.0 – and the network-GATE enclosure of Proposition 3.1 reopens, since GPL-3.0 has no network-interaction clause and only Section 13 survives the wire boundary (enclosure SAT). *Drop value-intrinsic* – let the value be GATE-callable from the unmodified division – and the unmodified wrap of Proposition 3.4 is admissible (enclosure SAT). *Drop the gate-invariant* – LINK the core to the division – and either the core inherits AGPL-3.0 by the join (the permissive-adoption half is lost), or, to restore adoption, the value is pushed back into the permissive core and Proposition 3.1 makes enclosure SAT again. Each singleton dropped admits an enclosure; the triple admits none. We do not claim $M^\star$ is the *unique* core; a uniqueness argument (the analogue of surface-locality in the trilemma report) is left open. The Section 13 side is typed DET (dual-verified, sha256 `98ecf65b`); the gate-invariant and value-intrinsic are DET obligations on our own graph, discharged by the licence-flow checker and the value-architecture review respectively. □

Remark 3.6 (Why the proprietary edge is consistent with the moat)**.** *The proprietary edge $T(\text{edge}) = \top'$ does not contradict the AGPL-3.0 division precisely because we hold copyright on that division. A copyright holder may dual-licence: the public receives the division under AGPL-3.0, while we grant ourselves a proprietary licence to build the edge on it. So the very clause that blocks a competitor's enclosure (Proposition 3.3) is the asset our own copyright converts into the proprietary edge – a third party, lacking our copyright, cannot replicate the*

edge without tripping Section 13. This is the value-capture side of the same core, and it is why the moat and the revenue layer are one object rather than two. Its one dependency – that we retain exclusive copyright on the division – is exactly what the contribution/review adversary of §7 attacks, and is carried in the provenance layer, not the pledge.

Proposition 3.7 (The violation regime is a deterrence condition, not a satisfiability question)**.** *Against an adversary who uses our AGPL-3.0 code but conceals it – rewriting to obscure provenance, shipping a locked black-box service, never disclosing – Proposition 3.3 does not bind: a clause cannot constrain a party who violates it. The moat in this regime is not unsatisfiability but a deterrence condition,* $\Pr[\textit{detection}] \times \textit{realised penalty} > \textit{enclosure gain}$*, in which* three *distinct legs must each hold, and any one failing breaks deterrence: (a) detection – proving our code is inside the black box; (b) realised penalty – our capacity and willingness to enforce; (c) provenance – proving the code is ours so the penalty can attach.*

Proof sketch. Satisfiability presumes compliance; a violator is outside the compliant strategy set, so the UNSAT of Proposition 3.3 is silent on it. What remains is a deterrence condition – not a solved inspection-game equilibrium: a full treatment would have the defender choose a costly detection investment d with $\Pr[\text{detection}] = p(d)$ and the violator best-respond with concealment that lowers p, an arms race we state as a condition rather than derive. The three legs separate cleanly, and this is where the honesty lives. *Provenance (c)* is supplied by the sigma-chain: it hash-chains *our own* releases and so proves ownership, the leg that makes a found violation actionable – but it proves ownership, it does not *find* our code in someone else's concealed artifact. *Detection (a)* is that finding – external forensics on the adversary's artifact (code fingerprinting, behavioural signatures, watermarking) – a capability distinct from provenance and one the substrate does not currently have; so $\Pr[\text{detection}]$ is, today, low. *Penalty (b)* is realised only if we can and will enforce: litigation against a resource-rich enclosurer is costly and slow, so enforcement capacity – not detection – can be the binding term precisely against the largest violators. AGPL-3.0 Section 13 supplies the legal basis for (b) and (c) but nothing for (a). The honest conclusion: the legal moat is complete against a compliant actor and only *partial* against a determined hidden violator, weakest exactly where the threat is biggest, and completed not by any licence clause but by an as-yet-unbuilt detection instrument, real enforcement capacity, and the non-licence moats – canonical-upstream position, community gravity, the data and service surface, and trademark (the soft-vector re-monopolisation surface identified in the audit acknowledged in §10). □

4 Typed clauses

As in the trilemma report, each load-bearing clause is typed by how mechanically its effect is settled, and the moat's strength is stated in both directions. AGPL-3.0 Section 13 in its *direct* case is DET: a modified division served over a network to interacting users triggers the source-offer obligation, and a checker can verify the trigger – this is the determinate spine of Proposition 3.3. Section 13 in its *boundary* cases is INT: whether a thin proxy, an unmodified passthrough, or an orchestration-only shim counts as the user "interacting remotely with the modified work" is a residue of legal judgement, named rather than mechanised (this is falsifier F1). The gate-invariant – that a process-and-wire boundary forms no derivative work – is DET-leaning but fact-sensitive: the settled reading is that network communication is mere aggregation, yet tight coupling can be argued into a single work, so the strategy keeps the boundary a genuine JSON/HTTPS protocol to stay on the determinate side (falsifier F3). Permissive relicensing to proprietary is DET: the grants permit it outright, which is exactly why Proposition 3.1 is determinate. The honest shape of the moat, then, is determinate against the adversary's cheapest moves – the direct software-as-a-service and linkage enclosures – and interpretive only at the

exotic boundaries. Verbatim clause text is drawn from the dual-verified licence corpus, not quoted from memory; the citable set is the clause registry `clause_rosters.license-moat-2026-08`.

5 Verdict

Theorem 5.1 (Enclosure-moat verdict at the current corpus snapshot)**.** *Under the three-layer target, Claim 1 holds* conditionally*: within-law enclosure is unsatisfiable iff the three-element core $M^\star = \{$AGPL-3.0 Section 13, gate-invariant, value-intrinsic$\}$ is maintained (Proposition 3.5); dropping value-intrinsic admits the unmodified wrap (Proposition 3.4, enclosure SAT). Claim 2 holds: a permissive-only target admits enclosure (Proposition 3.1), so the AGPL-3.0 division is load-bearing. Claim 3 – necessity across all relicensing trajectories – is decidable only against a declared trajectory set and remains a conjecture (§8); on the trajectories enumerated it holds. So the moat is real but rests on three things, not two, and the third – that the value be architected intrinsic to the division rather than wire-callable – is in genuine* tension *with the gate-everything discipline that keeps the core adoptable: that trade-off is the live architecture decision this report exists to surface. Against a violating adversary (Proposition 3.7) the moat is necessary-but-not-sufficient, its detection leg presently unbuilt and its penalty leg limited by enforcement capacity, so it is completed by forensic detection, enforcement, and non-licence instruments rather than by any licence clause.*

Remark (why Section 13 and not copyleft in general). Every other copyleft clause – GPL-3.0's derivative-work reach included – is defeated by the very boundary our own architecture recommends elsewhere: put a GATE (a wire protocol) between you and the copyleft code and the obligation does not travel. AGPL-3.0 Section 13 is the clause written *against that boundary* – but only for a *modified* deployment, which is why value-intrinsic is its indispensable partner: if the value is reachable from the *unmodified* division across a wire (Proposition 3.4), even Section 13 does not travel, and the moat is illusory. The moat is as strong as the one clause that survives the network cut *and only when the value is bound to the code that clause governs.*

Remark (scope: what the moat does not cover). The moat protects against enclosure of *our code.* A clean-room functional reimplementation that copies none of our code is legitimate competition, not enclosure; the pledge speaks to code-derived enclosure of the commons, not to the commons's market position against independent reimplementation.

6 The minimal sufficient pledge

The non-monopolisation pledge exists in an ambitious drafted form – elaborate terms, revenue and calendar cutoffs – prepared to cover many contingencies at once. The moat core tells us which of its clauses are load-bearing. The *minimal sufficient pledge* is the smallest clause set P such that P together with the adversary's within-law capabilities is unsatisfiable for enclosure, and no proper subset of P suffices – the dual, on the pledge itself, of the minimal unsatisfiable core located above. By Proposition 3.5 the load-bearing content reduces to *three* commitments: (a) the value-bearing division ships under AGPL-3.0 (or a licence at least as strong on network interaction); (b) the gate-invariant is maintained, so the permissive core reaches the division only across a wire boundary; and (c) the strategic value is architected *intrinsic* to the division (bound to modifying or linking it), not left extractable by calling the unmodified division across a wire. Commitment (c) is the one the ambitious draft omitted and the one that costs the most to hold, because it stands in tension with (b): the same wire-consumability that keeps the core adoptable is what a competitor's wrap exploits (Proposition 3.4). Everything the moat needs against a *compliant* adversary follows from these three; the elaborate revenue and calendar cutoffs are not required for enclosure-UNSAT and belong, if anywhere, to policy rather than to

the moat. The threat insight behind the ambitious draft is not discarded but re-homed: the warning about gamed contribution and external authorship is a *provenance* concern (§7), carried by the sigma-chain and quorum-attested contribution rather than by a pledge clause. The pledge is thus simplified to what is provably necessary, and no simpler.

7 Three adversary classes

The moat must hold against three distinct adversaries, and the analysis assigns each to the layer that actually binds it. The *relicensor* – a vendor or acquirer who relicenses, acquires-and-closes, keeps the released layer permissive while closing future versions, or hides intermediate layers – is a within-law player, met by the satisfiability moat of Proposition 3.3 and priced across trajectories by Claim 3; released versions are irrevocable, so the community always retains a fork from the last permissive commit. The *violator* – hidden use, provenance obscured – is met by the inspection game of Proposition 3.7, where the licence is necessary-not-sufficient and detection carries the moat. The *contribution/review gamer* – an adversary who injects code under a licence trap or later asserts external rights to poison provenance – is not a licence-game move at all: it attacks *who faithfully built the artifact*, not *what the artifact may become*, and is therefore met in the provenance and governance layer (sigma-chain provenance, quorum-attested contribution, and a Developer Certificate of Origin rather than a Contributor Licence Agreement), not in the pledge. Keeping this third class out of the pledge is exactly what keeps the pledge minimal.

8 Falsifiers

Claim 3 is a claim about relicensing trajectories, which no snapshot can prove; we state what would falsify the verdict, and enclosure becomes satisfiable (the moat fails) if any holds. *(F1)* A court reads AGPL-3.0 Section 13's network trigger *not* to reach the adversary's deployment – the interpretive boundary of §4 resolving against us – collapsing the clause half of $M^\star$ and reopening the software-as-a-service enclosure, in that jurisdiction, as an inspection game. *(F2)* The strategic value turns out to be reproducible from the permissive core alone – the division is not in fact load-bearing for the value – which reduces the case to Proposition 3.1 and makes enclosure satisfiable. *(F2b, value-intrinsic)* The value is deliverable by calling the *unmodified* division across a wire rather than by modifying or linking it – value-intrinsic fails – so the unmodified wrap of Proposition 3.4 is admissible and enclosure is satisfiable; this is the failure mode the third core element guards, and the one most in tension with our own gate discipline. *(F3)* The gate-invariant is not maintained as a genuine wire protocol – tight coupling in practice – so either the core self-contaminates to AGPL-3.0 (losing adoptability) or a court finds a single work across the boundary. *(F4)* We lose exclusive copyright on the division – external copyleft contributions accepted without a Developer Certificate of Origin or copyright assignment – weakening the dual-licence basis for the proprietary edge (Remark 3.6); this is the contribution/review adversary of §7, and it enters the moat analysis *only* here, through F4. *(F5)* A relicensing trajectory outside the declared set – a novel acquisition-and-close structure – against which the current layering is not robust would falsify Claim 3 as stated. Should the Concordance satisfiability test return the permissive-only strategy as *also* enclosure-unsatisfiable, Claim 2 would be false and the pledge rationale revised accordingly – the outcome we would report against our own preference, per the honesty charter.

9 Governance receipts

Every step ran inside a governed substrate: each licence fetch, dual-verification, hash, typing decision, and edit is a sigma-attested, append-only, hash-chained record, and the clause registry

is the only citable set. The licence corpus was fetched on two independent paths – the SPDX License List and each licence's steward (the Free Software Foundation for the GNU family, the Apache Software Foundation for Apache-2.0) – compared and archived with provenance caveats recorded; AGPL-3.0-only is dual-verified with content hash `98ecf65b`. The licence-flow algebra instantiated here is the general form developed for arbitrary code-flow spine-graphs; the two-sided map for IAS/Concordance – consuming zones (permissive Haskell abstract-syntax types and sources, through the tool binaries at their process boundary, to the Apache-2.0 proof and core layers) and emitting tiers (Apache-2.0 core, AGPL-3.0 division, proprietary edge), every boundary gate-typed – is its first worked instance. The satisfiability results are by-hand adversarial refutations, not solver runs; a licence-flow checker that computes Eff over the spine-graph and flags every $\mathrm{Eff}(n) \not\sqsubseteq T(n)$ with its offending LINK path is the mechanised sibling, to run at build and release. The naming discipline of the trilemma work is kept: the formalism speaks only of an adversarial-style actor; named real-world enclosure examples belong to the newsletter, never to the mathematics. An independent adversarial review attacked the two load-bearing proofs before this verdict was fixed: it found the original two-element core incomplete – the unmodified-wrap enclosure of Proposition 3.4 is admissible unless value-intrinsic (Definition 3.2) is added – promoting the core to three elements and the minimal pledge to three commitments, and it sharpened the violation regime into the three separable legs of Proposition 3.7; those corrections are incorporated here.

10 Acknowledgments

The central limit this report locates – that a licence, like a pledge, can lock the *hard* monopolisation vectors while re-monopolisation re-enters through the *soft* ones – we owe to an independent, commissioned non-monopolisation audit of VakeWorks's pledge by **Liezl Coetzee (Sociable Systems)** (first-pass, governance-and-commercial layers, 8 June 2026; public / non-NDA surface). Ahead of this work, that audit established with unusual clarity that a serious lock on patents, federated transport, identity decoupling, signing, and no-penalty exit still leaves the governance, interpretation, and service *windows* open – **convening and secretariat power, conformance-gatekeeping (who authoritatively certifies "compliant"), interpretive authority over load-bearing terms, reference-implementation normativity, and service gravity** – which is precisely where the incumbents (search, social, mobile-identity, cloud) actually consolidated on nominally-open substrates, *without a single patent*. This report's licence-flow specialisation of that insight is its central honest result: the moat is *necessary but not sufficient* (Proposition 3.7), and is completed by non-licence instruments – canonical-upstream position, community gravity, the service surface, trademark. Two further findings of her audit shape the framing here: the recursive observation that **every new governance or audit mechanism is itself a capture surface** – mirrored in our own result that the gate-invariant which keeps the core adoptable is the very seam a wrapper exploits (Proposition 3.4); and the "marking-your-own-homework" and auditor-pool-capture vectors, which motivate the report's contribution/review-gaming adversary class (§7) and its assignment to the provenance rather than the licence layer. Her audit also demonstrated the pledge's seriousness by how much of its first pass VakeWorks folded into the binding instrument – the same discipline of stating plainly where the front-door lock ends and the open windows begin that we have tried to keep here. The independent adversarial review of this report's own proofs is noted separately in §9.

References

[Apa04] Apache Software Foundation. Apache license, version 2.0. SPDX license list cross-verified against steward https://www.apache.org/licenses/LICENSE-2.0.txt; archived 2026-08-16, sha256 fac03222, dual-verified., 2004.

[Fre07a] Free Software Foundation. GNU affero general public license, version 3 (AGPL-3.0); Section 13, remote network interaction. SPDX https://spdx.org/licenses/AGPL-3.0-only.json cross-verified against steward https://www.gnu.org/licenses/agpl-3.0.txt; archived 2026-08-16, sha256 98ecf65b, dual-verified., 2007.

[Fre07b] Free Software Foundation. GNU general public license, version 3 (GPL-3.0). SPDX https://spdx.org/licenses/GPL-3.0-or-later.json cross-verified against steward https://www.gnu.org/licenses/gpl-3.0.txt; archived 2026-08-16, sha256 0c4658da, dual-verified., 2007.

[Ope99] Open Source Initiative. The 3-clause BSD license (BSD-3-Clause). SPDX https://spdx.org/licenses/BSD-3-Clause.html; archived 2026-08-16, sha256 2fcf05bb, single-path (steward portal unreachable, caveat recorded)., 1999.

[Sa26a] Raazesh Sainudiin and an IAS Commons Continuum. An interaction analysis of interacting agent systems: identity, continuity, and governed capability on a sovereign substrate. Monograph, in preparation (VakeWorks / OpenEarth Network Commons), 2026.

[Sa26b] Raazesh Sainudiin and an IAS Commons Continuum. A jurisdiction-explicit verdict on the sovereignty trilemma: testing an architectural impossibility claim against the US–EU–PRC digital-law corpus. VakeWorks Technical Report. https://vake.works/preprint/20260815-sovereignty-trilemma-jem.pdf, 2026.

[Sa26c] Raazesh Sainudiin and an IAS Commons Continuum. Three apples, one theorem: testing a viral sovereignty claim against the law itself. Being AVake Works newsletter (LinkedIn), issue 3, August 2026. https://www.linkedin.com/pulse/three-apples-one-theorem-testing-viral-sovereignty-claim-sainudiin-phnyf, 2026.

[Sa26d] Raazesh Sainudiin and an IAS Commons Continuum. A watermark for all. Being AVake Works newsletter (LinkedIn), issue 4, August 2026, 2026.

[SPD26] SPDX Workgroup, The Linux Foundation. SPDX license list, version 3.25. https://spdx.org/licenses/. Corpus dual-verified against each licence steward and archived 2026-08-16., 2026.