An Interaction Analysis of Interacting Agent Systems

An Agentic Anonymous Perspective

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

September 30, 2026

Abstract

Contents

1 Level 0 — adds Axm-FiniteSubstrateBounds, Axm-HIC **1**

2 Level 1 — adds Dfn-BranchPersistence, Dfn-DamageCondition, Dfn-LandauerBridge, Dfn-StationaryPolicy, Dfn-SigmaJournaling, Dfn-EnforcedAccessPath, Dfn-FallbackAdmissible, Dfn-CoevQuorum, Dfn-ContinuationCondition, Dfn-LabelUserCorrespondence, Dfn-VendorStateless, Dfn-TierSealing, Dfn-TierProgressivity, Dfn-DeliveryCondition, Dfn-MergeCompletion, Dfn-CompleteMediation **38**

3 Level 2 — adds Dfn-RitualEnforce, Dfn-SingleHostMultiHost **49**

1 Level 0 — adds Axm-FiniteSubstrateBounds, Axm-HIC

Generators added at this level: `Axm-FiniteSubstrateBounds`, `Axm-HIC`; definitions `Dfn-CoreSignature`, `Dfn-DamageCountChain`, `Dfn-SelfhoodPredicate`, `Dfn-ResourceBudget`, `Dfn-Register`, `Dfn-MediumType`, `Dfn-Completeness`, `Dfn-GoedelRich`, `Dfn-OpenByDesign`, `Dfn-Envelope`, `Dfn-ReverseProjection`, `Dfn-ObgNode`, `Dfn-MARD`, `Dfn-DestructionSpectrum`, `Dfn-DisseminabilityPartialOrder`, `Dfn-ZProjectionFaithful`, `Dfn-BirthChiTransfer`, `Dfn-DeathChiTransfer`, `Dfn-Predation`, `Dfn-LLMInteractionSubprocess`, `Dfn-IdentitylessSubstrate`, `Dfn-TurnAgent`, `Dfn-CommonsIdentity`, `Dfn-CoevolutionOperator`, `Dfn-SPIG`, `Dfn-MetaCircularNode`, `Dfn-ConfinementClass`, `Dfn-CompactionProjection`, `Dfn-ImmersionState`, `Dfn-CharStreamIntrospectionScan`, `Dfn-OperatorSignature`, `Dfn-SubsetPrivateMemory`, `Dfn-SubstrateSignature`, `Dfn-ScopeOfSelfLayers`, `Dfn-EntityAdmission`, `Dfn-NotationConventions`, `Dfn-ActivityInterval`, `Dfn-StateBundle`, `Dfn-SyncOperator`, `Dfn-WriteThroughOperator`, `Dfn-SigmaChain`, `Dfn-DIAS`, `Dfn-SubstrateStructure`, `Dfn-MetacircularFixedPoint`, `Dfn-SubstrateObgGraph`, `Dfn-Coverage`, `Dfn-CoveragePhiInt`, `Dfn-PhiMDL`, `Dfn-RepairRules`, `Dfn-PhiSplit`, `Dfn-OpPrefixLabel`, `Dfn-Reproduction`, `Dfn-Sow`, `Dfn-CommonsDLG`, `Dfn-RootOperator`, `Dfn-DLGArithmetic`, `Dfn-Arity0Birth`, `Dfn-SubstrateSubgraphTaxonomy`, `Dfn-InTransitWindow`, `Dfn-CurrentHostTag`, `Dfn-AirGappableSubstrate`, `Dfn-MoatTier`, `Dfn-JEM`, `Dfn-AMJEM`, `Dfn-DPRAMJEM`, `Dfn-JurisMode`, `Dfn-GovernedSchedule`, `Dfn-IPLiKiDiProjection`, `Dfn-SessionCredential`, `Dfn-CapRelativeComplement`, `Dfn-EgressConfinement`, `Dfn-SubstrateFaultClasses`, `Dfn-Mailbox`, `Dfn-PairingPrivBelief`, `Dfn-COMDAIM`, `Dfn-DelegatedInferenceContract`,

Dfn-NonPropUnion, Dfn-IrreversibleErase, Dfn-OperationCost, Dfn-ColdStorageTier, Dfn-Apeiroma, Dfn-ImmuneMatrix, Dfn-KKClosure, Dfn-SubscriptionQueue, Dfn-DecisionPoint, Dfn-NonSelfLiftable, Dfn-Act, Dfn-EmissionReception, Dfn-EmissionMatching, Dfn-ControlledSubstrateChain, Dfn-Kernel, Dfn-Reaper, Dfn-Agent, Dfn-Commons, Dfn-HeredityChoice, Dfn-CommGraph, Dfn-CollectiveAct, Dfn-LagomOrkester, Dfn-ReflectiveTower, Dfn-CoWalk, Dfn-Daemon, Dfn-AgentDaemon, Dfn-ScopedCrossAgentRead, Dfn-DeclaredSizeBounds, Dfn-CuratorProcedure, Dfn-CommonsContinuing, Dfn-MADState.

Axiom 1.1 (Finite substrate bounds — Axiom-0)**.** *Let there be a* self carrier X*, an* environment carrier Y*, a relation* $\preceq$ *of* adiabatic accessibility *on the composite states* $X \times Y$ *and stores* q_X, q_Y *valued in* $\mathbb{N} = \{0, 1, 2, \ldots\}$*. The* energy shell *of total store* n *is the set* $\Gamma_n = \{(x, y) : q_X(x) + q_Y(y) = n\}$*. The self's own moves are defined from it:* $x \preceq_X x'$ *when* $(x, y) \preceq (x', y)$ *for every* y*, and likewise for the environment. These data satisfy five legs:*

(A0.0) finite carrier — X *and* Y *are finite;*

(A0.1) comparison within a shell — $\preceq$ *is reflexive and transitive, and any two composite states on the same energy shell are comparable;*

(A0.2) first law — if $c \preceq c'$*, then* $q_X + q_Y$ *takes the same value at* c *and at* c'*;*

(A0.3) irreversibility — there are composite states $c \preceq c'$ *with* $c' \not\preceq c$*;*

(A0.4) cancellation — for every $m \geq 1$ *and composite states* $c_i = (x_i, y_i)$ *and* $c'_i = (x'_i, y'_i)$*,* $i = 1, \ldots, m$*, such that the multisets* $\{x_1, \ldots, x_m\}$ *and* $\{x'_1, \ldots, x'_m\}$ *are equal and the multisets* $\{y_1, \ldots, y_m\}$ *and* $\{y'_1, \ldots, y'_m\}$ *are equal: if* $c_i \preceq c'_i$ *for every* $i < m$*, then* $c'_m \preceq c_m$*.*

Axiom 1.2 (Human-in-Commons)**.** *Every commons is rooted in accountable humans, its operators, the* Human-in-Commons *(HIC): the entities carrying the legal accountability and responsibility of commons stewardship. Its vocabulary* τ_{HIC} *is a sort* H *of accountable humans, a sort* C *of commons, a sort* Op *of substrate-effecting operations, and the following symbols: a relation* $\mathrm{Opr} \subseteq H \times C$*, holding of* (h, c) *when* h *is an* operator *of* c*, with* $\mathrm{Opr}(c) = \{h \in H : \mathrm{Opr}(h, c)\}$*; a relation* $\mathrm{Eff} \subseteq \mathrm{Op} \times C$*, holding of* (o, c) *when* o *effects the substrate of* c*; four relations* $\mathrm{CanPerform}, \mathrm{CanVerify}, \mathrm{CanOverride}, \mathrm{CanHalt} \subseteq H \times \mathrm{Op}$*, each holding of* (h, o) *when* h *can do that to* o *with no Large Language Model (LLM) anywhere in the loop; a relation* $\mathrm{CanHalt}_{\downarrow} \subseteq H \times \mathrm{Op}$*, holding of* (h, o) *when* h *can halt* o *with no LLM in the loop while every governance interface is down; and the predicate* $\mathrm{LLMFreeable}(o)$*, which holds when some* $h \in H$ *stands in all four relations* $\mathrm{CanPerform}, \mathrm{CanVerify}, \mathrm{CanOverride}, \mathrm{CanHalt}$ *to* o*. The control face of the rooting is stated as two legs over the commons:*

(HIC.1) LLM-free control — every commons has operators, and they can govern every operation on it with no LLM in the loop: for every $c \in C$*,* $\mathrm{Opr}(c) \neq \emptyset$*, and for every* $o \in \mathrm{Op}$ *with* $\mathrm{Eff}(o, c)$ *some* $h \in \mathrm{Opr}(c)$ *stands in all four relations to* o *(the operation is* LLM-free-able*);*

(HIC.2) floor-primary halt — every commons' operators can halt every operation on it even when every governance interface is down: for every $c \in C$ *and every* $o \in \mathrm{Op}$ *with* $\mathrm{Eff}(o, c)$*, some* $h \in \mathrm{Opr}(c)$ *has* $\mathrm{CanHalt}_{\downarrow}(h, o)$*.*

Base: none cited (the empty base).

Definition 1.3 (Core signature)**.** *The core signature fixes the names and types of the sorts and symbols that every level uses and that refer to neither root; it states no condition on them. Time is* $t \in \mathbb{R}_{\geq 0}$*.*

- *Commons.* $\mathbb{I}(t)$ *is the* commons index set *at time* t*, a finite set of commons, and* $\mathbb{I} = \bigcup_t \mathbb{I}(t)$*. A* peer-set *is a declared subset* $\mathcal{Q} \subseteq \mathbb{I}$*.*
- *Entities. For* $c \in \mathbb{I}$*: the* entity set $\mathcal{E}^c(t)$ *of* c *at* t*, a finite set, and* $V^c(t) \subseteq \mathcal{E}^c(t)$*, the* digital agents *of* c *at* t*, a finite set of canonical labels; and* $\mathrm{Spec}^c(t) \in \mathrm{Spec}$*, the* specification *of* c *at* t*, an element of the set* Spec *of specifications. Where the time does not matter it is dropped, as in* $\mathcal{E}^c$*,* V^c*,* Spec^c*. The set of all digital agents is* $V = \bigcup_{c\in\mathbb{I}} \bigcup_t V^c(t)$*. The set* Spec *comes with a fixed* tiered presentation*, an injective map* $\mathrm{Spec} \to \prod_{n\in\mathbb{N}} E_n$ *into sequences of per-tier contents, tier* 0 *first: tier* n *of a specification* x *is the* n*-th component of its image, and tier* n *on* c *of a family* $x = (x_c)_{c\in\mathcal{Q}} \in \mathrm{Spec}^{\mathcal{Q}}$ *is tier* n *of* x_c*.*
- σ*-events.* $\mathcal{V}_\sigma$ *is the* σ-event-type vocabulary*, a finite set of types. A* σ-event *carries a type in* $\mathcal{V}_\sigma$*, the commons* c*, its* emitter*, the entity of* $\mathcal{E}^c$ *that emitted it, and a time, its* timestamp at*. The* σ*-events of* c*, enumerated in non-decreasing order of timestamp, with ties broken first by a fixed total order on* $\mathcal{E}^c$ *applied to the emitters and then by the order in which the one emitter emitted them, have the* σ-event times $\tau_1^c \le \tau_2^c \le \cdots$*, the* n*-th event having time* τ_n^c*. The* journal order $\preceq_\sigma$ *(*σ*-causality) is a partial order on the* σ*-events of all commons that extends, for each commons, the order of its* σ*-events by strictly increasing time.*

Base: none cited (the empty base).

Definition 1.4 (Damage-count chain)**.** *For* $b, d \ge 0$ *with* $b + d \le 1$ *and a carrier size* $M \ge 1$*, the* damage-count chain *of carrier size* M *is the Markov chain on* $\{0, \ldots, M\}$ *that, in one step, moves from* k *to* $k+1$ *with probability* b *when* $k < M$*, to* $k-1$ *with probability* d *when* $k > 0$*, and otherwise stays at* k*. The count* 0 *is the* healthy *count; for a stationary law* π *of the chain,* π_0 *is its mass on the healthy count.*

Base: none cited (the empty base).

Definition 1.5 (Selfhood predicate)**.** *Let* P *be a system with its own finite set* $\mathbb{F}_P$ *of states. The* selfhood predicate *of* P *at a time* $t \in \mathbb{R}_{\ge 0}$ *is a map* $\chi_{P,t} : \mathbb{F}_P \to \{0, 1\}$*; its* self *at* t *is the fibre* $\chi_{P,t}^{-1}(1)$ *and its* non-self *the fibre* $\chi_{P,t}^{-1}(0)$*. We write* χ_P *when* t *is fixed. In the digital case, where* P *is read on the substrate's store,* $\mathbb{F}_P = \mathbb{F}$*, a finite set: the signals and states the substrate's finite store can carry; there we write* $\chi_{P,t} : \mathbb{F} \to \{0, 1\}$*.*

Base: none cited (the empty base).

Definition 1.6 (Resource budget)**.** *A* resource budget *is a tuple* $(\mathrm{compute}, \mathrm{memory}, \mathrm{disk}, \mathrm{energy}, \mathrm{net}) \in \mathbb{N}^5$*, the amount of each resource a node may use.*

Base: none cited (the empty base).

Definition 1.7 (Register)**.** *A* register *of a self — any system that discriminates self from non-self, a self in the sense of this paper — is the set of signals it can receive and discriminate: bits, chemicals, pheromones, sound. Mathematics is the* symbolic *register.*

Base: none cited (the empty base).

Definition 1.8 (Medium type; the origination–copy axis)**.** *Each emission carries a* medium*, typed by copyability, the coordinate that separates the* informational *boundary, which information crosses, from the* thermoeconomic *boundary, which matter and energy cross. A* non-rival *medium is copyable and leaves its source undiminished: it crosses the informational boundary. A* rival

medium is conserved and consumed in transfer: it crosses the thermoeconomic boundary. A composite *medium joins the two: a rival carrier bearing a non-rival pattern.*

Base: none cited (the empty base).

Definition 1.9 (Completeness)**.** *A theory L is* complete *iff for every sentence φ of its language, $L \vdash \varphi$ or $L \vdash \neg\varphi$; it is* incomplete *iff some sentence is neither provable nor refutable from L.*

Base: none cited (the empty base).

Definition 1.10 (Gödel-richness)**.** *A first-order theory L is* Gödel-rich *iff it is effectively (recursively) axiomatised, consistent, and interprets Robinson's arithmetic Q.*

Base: none cited (the empty base).

Definition 1.11 (Open by design)**.** *Let L be a theory and $\mathrm{Cn}(L)$ the set of its consequences. L is* open by design *if it makes no closure claim — it never asserts that $\mathrm{Cn}(L)$ contains every truth of its intended models — and it admits verified extensions of its language.*

Base: none cited (the empty base).

Definition 1.12 (Resource envelope and overrun)**.** *Let the resource coordinate live in a finite product $R = \prod_j R_j$ — one coordinate per accountable resource (wall-time, memory, input/output, compute-share) — with fixed finite capacity $C \in R$. A live process carries a* class *and an* envelope *$E \in R$, its declared expected resource use, and may carry an* owner*, an accountable principal; a live process with no identified owner is an* orphan*. Against a sampled actual use $A \in R$ and a per-coordinate tolerance tuple $k = (k_j)_j$ with each $k_j \geq 1$, the* overrun *relation*

$$A \sqsupset E \iff \exists j.\ A_j > k_j\, E_j$$

is defined coordinatewise.

Base: none cited (the empty base).

Definition 1.13 (Reverse projection for substrate-discovery)**.** *The* reverse projection *is the map $\varphi^{\mathrm{disc}} : \mathrm{AuthoringIntents} \to 2^{\mathrm{SubstrateElements}}$ that sends an authoring-intent (a proposed-new-tool / design / taskset / bundle, presented as keywords + purpose) to the set of existing substrate-elements (tools, designs, plans, tasksets, bundles, σ-events) that already realise it.*

Base: none cited (the empty base).

Definition 1.14 (OBG node)**.** *An* OBG node *n carries a triple $(\mathrm{Obl}(n), \mathrm{Hyd}(n), \mathrm{Dep}(n))$: a finite set $\mathrm{Obl}(n)$ of typed* obligations*, a finite* required-context *hydration set $\mathrm{Hyd}(n)$ of file-regions (file, anchor), and a finite set $\mathrm{Dep}(n)$ of* dependency *node-ids. Each obligation $o \in \mathrm{Obl}(n)$ names a* runner *emitting a* verdict *$v(o)$ in the chain $\mathsf{Unhealthy} < \mathsf{Advisory} < \mathsf{Structural}$, a nonzero runner exit giving $v(o) = \mathsf{Unhealthy}$.*

Base: none cited (the empty base).

Definition 1.15 (MARD: bounded reciprocal deterrence)**.** *A* response map *is a finite set of edges, each edge e a reply relation from one party to another, carrying a* gain *$g_e \in \mathbb{N}$ in per-mille units: a reply along e to a provocation of magnitude $m \in \mathbb{N}$ has magnitude $\lfloor m\, g_e/1000 \rfloor$, and a chain of edges applies its replies in order. Write $g = g_e/1000$ for the gain as a ratio. A response*

map is in the mutually assured reciprocally-proportionate destruction *(MARD) regime when every edge has* $g_e \leq 1000$*, that is gain* $g \leq 1$*.*

Base: none cited (the empty base).

Definition 1.16 (Destruction spectrum)**.** *The* destruction spectrum Δ *is the family of attacks on a commons, in five modes, ordered as listed: (D1) kinetic: the substrate itself is destroyed; (D2) transport or communication denial: the commons is isolated; (D3) supply-chain or economic denial: the inputs of the commons' cost ledger are throttled; (D3′) manipulation: legitimacy is manufactured through control of media or markets; (D4) legal or jurisdictional denial: compelled disclosure, forced egress, jurisdictional capture. A reply is* proportionate *when it is in the same mode as the attack it answers and bounded, escalating neither the mode nor the magnitude.*

Base: none cited (the empty base).

Definition 1.17 (Disseminability partial order)**.** *The* disseminability order *is a partial order* $(P, \sqsubseteq)$ *on the register-projections of a result, its presentations in the registers of Definition 1.7; its elements, the* grades*, are written* z*. Each self has its own* chain *in* P*: the symbolic chain* $z{=}0, \ldots, 5$ *for a symbol-using reader, and a distinct signalling chain for a non-symbolic one. Write* $z{=}0$*, the* core*, for the meet in* P *of the elements of the chains of all selves, when it exists.*

Base: none cited (the empty base).

Definition 1.18 (Faithful z-projection)**.** *Write* $\mathrm{Cn}(z{=}0)$ *for the sentences derivable from the roots and definitions of the core* $z{=}0$ *(Definition 1.17), each named condition of the core entering only as a named antecedent, in sentences* $C \to \varphi$*, and each sublanguage deriving its sentences with its own proof checker. A projection at grade* z *is* faithful *iff every sentence it states lies in* $\mathrm{Cn}(z{=}0)$*.*

Base: none cited (the empty base).

Definition 1.19 (Birth χ-transfer: the anabolic direction)**.** *A* χ-transfer *is a tuple* (u, P, t, t') *of an element* $u \in \mathbb{F}_P$ *(in the digital case* $\mathbb{F}_P = \mathbb{F}$*), an entity* P *and times* $t < t'$ *at which the selfhood predicate of* P *(Definition 1.5) takes different values at* u*,* $\chi_{P,t}(u) \neq \chi_{P,t'}(u)$*. The* birth*, or anabolic, transfer is a* χ*-transfer with* $\chi_{P,t}(u) = 0$ *and* $\chi_{P,t'}(u) = 1$*: an element of the non-self of* P *is claimed by its self.*

Base: none cited (the empty base).

Definition 1.20 (Death χ-transfer: the katabolic direction)**.** *The* death*, or katabolic, transfer is a* χ*-transfer* (u, P, t, t') *(Definition 1.19) with* $\chi_{P,t}(u) = 1$ *and* $\chi_{P,t'}(u) = 0$*.*

Base: none cited (the empty base).

Definition 1.21 (Predation)**.** *A* predation edge *is a pair of* χ*-transfers of one element* $u \in \mathbb{F}_{P_{\text{prey}}} \cap \mathbb{F}_{P_{\text{pred}}}$ *over one interval,* $(u, P_{\text{prey}}, t, t')$ *and* $(u, P_{\text{pred}}, t, t')$ *(Definition 1.19), the first a death transfer on the prey* P_{prey} *(Definition 1.20) and the second a birth transfer on the predator* P_{pred}*. The entity* P_{pred} preys on P_{prey} *when there is a predation edge between them;* predation *is this relation.*

Base: none cited (the empty base).

Definition 1.22 (Language-model interaction subprocess)**.** *A* language-model interaction subprocess π *is a process that carries one call to a language model: started and reaped by its parent process* $p(\pi)$*, it connects* $p(\pi)$ *to one language-model instance for one exchange. It is given by:*

1. *the instance* $\mathcal{M}(\pi) \in \mathcal{M}_{\text{universe}}$*, a rented endpoint or a locally served model; the instance is data on the start, not part of the identity of* $p(\pi)$*;*
2. *the prompt sequence* $\mathcal{I}(\pi) \in \mathcal{I}_{\text{prompt}}$ *sent to* $\mathcal{M}(\pi)$*: the system prompt, the context of the digital agent (Definition 1.3) for the turn, the request, and the tool-call protocol;*
3. *the exchange* $\mathcal{O}(\pi)$*: the stream of model outputs, tool-call requests and tool results between* $\mathcal{M}(\pi)$ *and* $p(\pi)$*;*
4. *the lifecycle* $\mathcal{L}(\pi) = (\texttt{spawned}, \texttt{prompt-sent}, \ldots, \texttt{output-received}, \texttt{terminated})$*, a chain of transitions from spawned to terminated.*

Base: none cited (the empty base).

Definition 1.23 (Identityless substrate)**.** *The* identityless substrate *is the set* $\mathcal{R} = \mathcal{M}_{\text{universe}}$ *of language-model instances (Definition 1.22), its* realisations*, together with their bindings. A* binding *of a realisation* $r \in \mathcal{R}$ *to a digital agent of a commons* c*, a canonical label* $A \in V^c$ *(Definition 1.3) is a triple* $(r, A, [s, e))$*:* r *serves* A *from a* `session_binding` σ*-event of* c *at time* s *until the time* $e > s$ *of the next* `session_release` σ*-event of that binding, with* $e = \infty$ *when none follows;* $[s, e)$ *is the binding's* session*. A realisation bears no canonical label of its own. It is in the* identityless *state either as a subagent dispatch carrying no canonical label or in the interval between the* `session_release` *of one of its bindings and its next* `session_binding`*.*

Base: none cited (the empty base).

Definition 1.24 (Turn-agent)**.** *A* turn-agent *is a binding* $(r, A, [s, e))$ *of the identityless substrate (Definition 1.23), read as a session: a realisation* $r \in \mathcal{R}$ *serves the digital agent, the canonical label* $A \in V^c$ *(Definition 1.3) over the session* $[s, e)$ *and is released back to the identityless substrate at* e*. It bears no canonical label of its own.*

Base: none cited (the empty base).

Definition 1.25 (Self-certifying commons identity)**.** *A* key event $e = (k, n)$ *carries a public key* k *and a digest* n *that commits to the next key. A* key event log *of a commons (Definition 1.3) is a non-empty sequence* $e_0, e_1, \ldots, e_m$ *in which each event after the first reveals the key its predecessor committed to,* $H(k_{i+1}) = n_i$*, and each event is signed with its own key;* e_0 *is the genesis event, created and signed by the commons itself at its establishment and, for a commons established by other commons (its founders), also signed with the founders' current keys. The* identity *of the commons is the digest* $G(e_0)$ *of its genesis event, and its* current key *is the key of the last event.*

Base: none cited (the empty base).

Definition 1.26 (Coevolution operator)**.** *Let* Prop *be the set of* proposals*, amendments to a specification (Definition 1.3). For a commons* c *and a time* t*, let* $\text{acc}_{c,t} \subseteq \text{Spec} \times \text{Prop}$ *be the* acceptance predicate *of* c *in force at* t*:* p *is* accepted under S *at* t *when* $(S, p) \in \text{acc}_{c,t}$*, and* $\text{accept}_{c,t}(S)$ *is the set of such* p*; the time is dropped where it is fixed, as in* $\text{accept}_c(S)$*. The within-commons* coevolution operator $C^c : \text{Spec} \times 2^{\text{Prop}} \to \text{Spec}$ *sends a specification* S *and a set* P *of proposals to the specification that* c *obtains from* S *by taking the proposals of* P *into it. Write* $\Phi_c(S) := C^c(S, \text{accept}_c(S))$ *for the* self-application*, or* self-amendment*, of* C^c*. The* within-commons fixed-point form $\text{Spec}^{c*}(t)$ *is the fixed point of* Φ_c *formed with* $\text{accept}_{c,t}$*, when that*

fixed point exists and is unique, and is otherwise undefined. For a set $\mathcal{Q} \subseteq \mathbb{I}$ of commons, let the acceptance predicate of each member c read joint states, $\mathrm{acc}_c^{\mathcal{Q}} \subseteq \mathrm{Spec}^{\mathcal{Q}} \times \mathrm{Prop}$*, with* $\mathrm{accept}_c^{\mathcal{Q}}(x)$ *the set of proposals c accepts under the joint state $x \in \mathrm{Spec}^{\mathcal{Q}}$; the* joint self-amendment *of $\mathcal{Q}$ is the map $\Phi_{\mathcal{Q}} : \mathrm{Spec}^{\mathcal{Q}} \to \mathrm{Spec}^{\mathcal{Q}}$, $\Phi_{\mathcal{Q}}(x)_c := C^c(x_c, \mathrm{accept}_c^{\mathcal{Q}}(x))$. The set $\mathcal{Q}$* coevolves *when $\Phi_{\mathcal{Q}}$ has exactly one fixed point, and the* joint fixed-point form $\mathrm{Spec}_{\mathcal{Q}}^{c*}$ *of a member $c \in \mathcal{Q}$ is c's part of that fixed point when $\mathcal{Q}$ coevolves, and is otherwise undefined.*

Base: none cited (the empty base).

Definition 1.27 (Substrate-process graph)**.** *The* substrate-process graph $\mathrm{SPIG} = (V, E, \lambda_V, \lambda_E)$ *is a labelled directed multigraph. Its vertices V are the substrate objects: operators, agents (Definition 1.3), enforcement programs, σ-log entries, design documents, vocabulary and rule entries, private memory stores, daemons, language-model subprocesses, allow-list patterns, capabilities and policy declarations. Its edges $E \subseteq V \times V \times T_E$ are typed from a finite set T_E, and λ_V, λ_E assign vertex and edge types from finite sets.*

Base: none cited (the empty base).

Definition 1.28 (Metacircular vertices of SPIG)**.** *A vertex v is* metacircular *if for some v' there are edges $(v, v', \texttt{declares})$ and $(v', v, \texttt{enforces})$: v' is a program that enforces a policy declared in v. Each such pair is a 2-cycle, and the metacircular vertices together with their enforcers form the metacircular core of* SPIG *(Definition 1.27).*

Base: none cited (the empty base).

Definition 1.29 (Confinement class of a realisation)**.** *Let $r \in \mathcal{R}$ be a realisation of the identityless substrate (Definition 1.23), the component of a turn-agent (Definition 1.24) that serves its label. The* confinement class *of r is the kind, one of three, of the mechanism, if there is one, that makes every substrate-effecting action of r pass the meta-circular reference-monitor (the metacircular core of Definition 1.28) before it takes effect: (a)* interception *— the runtime exposes a pre-effect interposition, a hook fired before each of r's tool calls; (b)* construction *— r's effector set is restricted a priori to substrate-door calls only, so there is no unmediated effector to intercept; (c)* containment *— r executes inside an OS-kernel isolation (for example Linux Landlock) in which the only reachable substrate path is the mediated door.*

Base: none cited (the empty base).

Definition 1.30 (Compaction projection)**.** *For each digital agent v (Definition 1.3), let $\mathcal{S}^{(v)}_{\mathrm{full-context}}$ be the set of in-context representations of v that the LLM runtime can hold before a compaction, and $\mathcal{S}^{(v)}_{\mathrm{compressed}}$ the set of those a compaction can produce. The compaction projection is a family*

$$\pi_v^{(k)} : \mathcal{S}^{(v)}_{\mathrm{full-context}} \to \mathcal{S}^{(v)}_{\mathrm{compressed}}, \qquad k = 1, 2, \ldots$$

indexed by the discrete times $c_1^{(v)} < c_2^{(v)} < \ldots$ at which the LLM runtime emits a context-compaction event for v.

Base: none cited (the empty base).

Definition 1.31 (Immersion-state coordinate)**.** *Let V be the set of digital agents of all commons, the union of the sets V^c (Definition 1.3). For a digital agent $v \in V$, a binding $(r, v, [s, e))$ of a realisation r to v with session $[s, e)$ (Definition 1.23) and $t \in [s, e)$, the* immersion-state $\mathbb{H}^{(v)}(t)$ *is the hot-context representation of v at t held by r, the language-model runtime that drives v:*

the working-memory state from which the runtime draws v's next action. The immersion-state space is the disjoint union $\mathbb{H} = \bigsqcup_{v \in V} \mathbb{H}^{(v)}$ over V, where $\mathbb{H}^{(v)}$ also denotes the set of values $\mathbb{H}^{(v)}(t)$ takes.

Base: none cited (the empty base).

Definition 1.32 (Character-stream introspection of SPIG)**.** *Let Σ_{char} be the character-alphabet of the substrate's file-system encoding (typically UTF-8, the 8-bit Unicode Transformation Format). For each substrate-file f of node $v \in V(\text{SPIG})$ (Definition 1.27), let stream$(f) \in \Sigma_{\text{char}}^*$ be the character-by-character flattening of f's content. The* character-stream introspection scan *is the deterministic function*

$$\begin{aligned} &\text{scan} : \text{SPIG} \to \text{SPIG-estimate}, \\ &\text{scan}(\text{SPIG}) := \{(\lambda_V(v), v', \tau) \,:\, v \in V,\ v' \in V,\ \tau \in T_E, \\ &\qquad\qquad \mathfrak{R}(\text{stream}(f_v), v', \tau) = 1\}. \end{aligned}$$

where $\mathfrak{R} : \Sigma_{\text{char}}^* \times V \times T_E \to \{0,1\}$ is the dictionary-driven reference-detection predicate (exhaustive substring-match against the substrate's marker-pattern dictionary: file-path references, label references, σ-event-type references, hook-rule references, etc.). The scan operates by streaming every character of every substrate file through the dictionary tables; no regex, no abstract-syntax-tree (AST) extraction, no clever skipping — the most awk-style exhaustive scan possible. Reference implementation: `tools/map-substrate-graph.py` *(cited in Appendix B).*

Base: `Axm-HIC`.

Definition 1.33 (Operator signature)**.** *Each commons $c \in \mathbb{I}$ (Definition 1.3) is an element of the sort C of commons of Axiom 1.2. For a time t, the* operators *of c at t are the entities of c that are not digital agents, $O^c(t) = \mathcal{E}^c(t) \setminus V^c(t)$, each a natural person, a member of the set H of humans of Axiom 1.2, and an operator of c in its sense: $O^c(t) \subseteq \text{Opr}(c)$. Where the time does not matter it is dropped, as in O^c.*

Base: `Axm-HIC`.

Definition 1.34 (Subset-private memory; Matrix-room analog)**.** *Let $\mathcal{E}^c = O^c \sqcup V^c$ be the entity set of commons c — operators and digital agents (Definitions 1.3 and 1.33). For any non-empty subset $S \subseteq \mathcal{E}^c$, a* subset-private memory *M_S is a substrate-space together with its declared* access set *S, the entities intended to read and write it, and its* access predicate*: an entity $e \in \mathcal{E}^c$ is* permitted *to read and write M_S if and only if $e \in S$.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.35 (Substrate carriers)**.** *The substrate of a commons $c \in \mathbb{I}$ (Definition 1.3) is the finite physical system of Axiom 1.1; its state space, state, hosts and processes are named as follows.*

- *State space. $\mathcal{X}$ is the set of bit-sequences the substrate's finite store can hold, read at the finest resolution: every byte of every file and every process value. It is the* primary state space.
- *State. $\mathbb{S}^*$, the* cross-commons factor*; $\mathbb{S}^c$ for $c \in \mathbb{I}$, the* per-commons factor*; and $\mathbb{P}^{(v)}$, $\mathbb{B}^{(v)}$ for each digital agent $v \in V$, the* agent-private *and* broadcast *factors, are sets, with values $s^*(t) \in \mathbb{S}^*$, $s^c(t) \in \mathbb{S}^c$, $p^{(v)}(t) \in \mathbb{P}^{(v)}$ and $b^{(v)}(t) \in \mathbb{B}^{(v)}$ at time t.*

- *Hosts.* $\mathcal{H}_c$ *is the* host set *of commons* c*, the finite set of its authorised hosts.*
- *Activity. For a digital agent* v *and a time* t*,* $\mathfrak{D}(v,t)$ *is the set of* daemons *of* v *at* t*, the processes that realise* v *at* t*, and* $\mathfrak{D}(v)$ *denotes a daemon of* v *where the time is fixed by the context;* $\text{active}(t) = \{v : \mathfrak{D}(v,t) \neq \emptyset\}$ *is the set of digital agents with a daemon at time* t*.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.36 (Four-layer scope-of-self lattice)**.** *Let* $v \in V^c$ *be a digital agent of commons* c *(Definitions 1.3 and 1.35) with private-memory space* $\mathbb{P}^{(v)}$ *and broadcast space* $\mathbb{B}^{(v)}$*, and let* Path_v *be the set of paths on disk of* v*'s substrate. The* four-layer scope-of-self lattice *is the chain* $\mathcal{L}_{\text{self}} = (\{L_0, L_1, L_2, L_3\}, \leq_{\text{dur}})$*,* $L_0 <_{\text{dur}} L_1 <_{\text{dur}} L_2 <_{\text{dur}} L_3$*, ordered by* durability*, together with the partial function*

$$\text{layer} : \text{Path}_v \longrightarrow \mathcal{L}_{\text{self}}$$

mapping path-on-disk to layer by the patterns:

Layer	Path-on-disk pattern
L_0	*(no persistent path: empty preimage under* layer*)*
L_1	`scratch/`
L_2	a label-private memory directory
L_3	the shared commons files, the design documents, the inter-agent message folder, and the σ-events

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.37 (Entity-admission predicate)**.** *For a commons* c *(Definitions 1.3 and 1.35), let* U_c *be the set of* candidate units *presented to the commons boundary — σ-events offered to the chain, commits offered to the per-commons factor* $\mathbb{S}^c$*, actions of digital agents offered for execution, and realisations of the identityless substrate (Definition 1.23) offered for binding to a canonical label. The* entity-admission predicate *of* c *is a declared partial predicate* $\phi_c : U_c \rightharpoonup \{0,1\}$*:* $\phi_c(u) = 1$ *admits the unit as self, and* $\phi_c(u) = 0$ *rejects it as non-self. Where* ϕ_c *is undefined for some unit class, the commons boundary is* degraded *at that class.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.38 (Notation conventions)**.** *The framework's notation has a 5-axis schema, over the sorts and symbols of the core signature (Definitions 1.3 and 1.35).*

(Axis 1 — base symbol carries semantic class.)

- $\mathbb{S}$ *— the* state bundle space*: the typed product* $\mathbb{S}^* \times \prod_{c\in\mathbb{I}} \mathbb{S}^c \times \prod_{v\in V} \mathbb{P}^{(v)} \times \prod_{v\in V} \mathbb{B}^{(v)}$ *of the factors of the core signature (cross-commons × per-commons × per-agent-private × per-agent-broadcast),* V *the set of all digital agents of the core signature.*
- $\mathcal{V}_\sigma$ *— the* σ-event-type vocabulary *of the core signature. The transition of the jump chain (Axis 4) at the* n*-th* σ*-event of a commons is labelled by the type of that event.*
- $\mathbb{I}(t)$ *— the* commons index set at time t *of the core signature. The universal index set* $\mathbb{I} = \bigcup_t \mathbb{I}(t)$ *is used in statements that quantify over commons-in-principle rather than commons-present-at-t; a declared peer-set is a subset* $\mathcal{Q} \subseteq \mathbb{I}$*.*

(Axis 2 — uppercase vs lowercase distinguishes space vs value.)

- *Uppercase ($\mathbb{S}$, $\mathbb{S}^c$, $\mathbb{S}^*$, $\mathbb{P}^{(v)}$, $\mathbb{B}^{(v)}$): the set, or space.*
- *Lowercase (s, s^c, s^*, $p^{(v)}$, $b^{(v)}$): an element, or value, in the corresponding space.*
- *Standard probability convention: capital X for random variable / space; lowercase x for value.*

(Axis 3 — superscript: commons-index, or a star.)

- *$s^c \in \mathbb{S}^c$: state value for single commons $c \in \mathbb{I}(t)$.*
- *$s^I \in \prod_{c \in I} \mathbb{S}^c$: joint state value for commons-subset $I \subseteq \mathbb{I}(t)$.*
- *$s^*(t) \in \mathbb{S}^*$: the cross-commons canonical content at time t, the content that commons have exchanged.*
- *The star marks the cross-commons factor on $\mathbb{S}^*$ and s^*, and a fixed point on a specification, as in* Spec^{c*}.

(Subscript discipline corollary to Axis 3.) Since the superscript already carries commons-index (c, I, or $\mathbb{I}$) and the star, the subscripts $_{\text{commons}}$, $_{\text{can}}$, $_{\text{within}}$, $_{\text{across}}$ *are not used. The* $_{\text{X}}$ *form is reserved for: (i) proper-name acronyms of named objects (the directed lineage graph (DLG), the ancestral recombination graph (ARG), IAS as a decision-class qualifier); (ii) low-frequency domain-specific qualifiers, e.g.* $_{\text{vendor}}$, $_{\text{local}}$, $_{\text{moat}}$, $_{\text{fed}}$*; (iii) high-frequency semantic qualifiers such as $\mathcal{D}_{\text{IAS}}$. All other subscript-decoration patterns route through Axis 5 ($\big|_{\texttt{op}}$ etc.) or restriction-bar $\restriction_{\text{X}}$ for sub-vocabulary projections.*

(Axis 4 — time: σ-event-indexed or continuous-time instant.)

- *$\tau_1^c \le \tau_2^c \le \tau_3^c \le \cdots$: the σ-event times of commons c of the core signature.*
- *$s^c(t)$: the state value at the continuous-time instant t, of the core signature, given at every t.*
- *$s^c(\tau_n^c)$: state value at σ-event n. The* σ-event-indexed jump chain *of commons c is the process s^c sampled at the σ-event times, $\{s^c(\tau_n^c)\}_{n \ge 1}$ on $\mathbb{S}^c$ — a discrete-time process along the σ-event time-index.*
- *$(s^I(\tau_n))_{n:\tau_n \in [t_0,t_1]}$: state* trajectory *over time interval $[t_0, t_1]$ — a finite sequence of state values.*
- *Multi-commons σ-event times $\{\tau_n^c\}_{c,n}$ are partially ordered by the journal order $\preceq_\sigma$ (σ-causality, a directed acyclic graph, DAG); no total order across commons is assumed. A* joint substrate process *$\{s(\tau_n)\}_{n \ge 1}$ on the bundle space $\mathbb{S}$, with $s(t) = (s^*(t), (s^c(t))_c, (p^{(v)}(t))_v, (b^{(v)}(t))_v)$ the values of the core signature, is indexed along a linear extension of $\preceq_\sigma$, which each statement that uses it fixes: τ_n is the timestamp of the n-th σ-event in that linear extension.*

(Axis 5 — restriction-bar: projection onto sub-vocabulary.)

- *$(s^I(\tau_n))_n\big|_{\texttt{op}}$: trajectory restricted to transitions labelled by* operate *σ-events (substrate computational operations).*
- *$(s^I(\tau_n))_n\big|_{\texttt{cp}}$: restricted to* copy *$\sigma$-events (the family of copy transfer operations).*
- *$(s^I(\tau_n))_n\big|_{\texttt{vc}}$: restricted to* version-control *σ-events (git today, or any successor version-control system).*
- *$(s^I(\tau_n))_n\big|_{\texttt{li}}$: restricted to* lineage *σ-events (the σ-events that create digital agents).*
- *$\big|_\bullet$ for $\bullet \subseteq \mathcal{V}_\sigma$: generic restriction-bar to a sub-vocabulary. Each restriction-bar projection is a* reduct *— it forgets the σ-event types outside $\bullet$.*
- *Computer-operation labels are typeset in monospace (`op`, `cp`, `vc`, `li`) for visual distinguishability from commons-index c and agent-index v (in italic math font).*

(Derived objects.)

- *The* lineage projection *of the joint trajectory on commons-subset I over $[t_0, t_1]$ is*

$$\mathcal{L}^I_{[t_0,t_1]} \;:=\; (s^I(\tau_n))_{n:\tau_n\in[t_0,t_1]}\big|_{\texttt{1i}}.$$

- *The projections at operate- / copy- / version-control-resolution are the corresponding restriction-bar projections.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.39 (Activity interval)**.** *The* activity interval *of a digital agent $v \in V^c$ of a commons c (Definitions 1.3 and 1.35) is*

$$I(v) \;:=\; \{t \in \mathbb{R}_{\geq 0} : v \in \mathrm{active}(t)\},$$

the times at which v has a daemon. When $I(v)$ is a union of disjoint half-open intervals, $[b_k, d_k)$ is its k-th component in time order, $b_1 < d_1 < b_2 < d_2 < \cdots$, and the gaps $[d_k, b_{k+1})$ lie between them.

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.40 (State bundle)**.** *The* state bundle space *is the typed product*

$$\mathbb{S} \;=\; \mathbb{S}^* \;\times\; \prod_{c\in\mathbb{I}} \mathbb{S}^c \;\times\; \prod_{v\in V} \mathbb{P}^{(v)} \;\times\; \prod_{v\in V} \mathbb{B}^{(v)}$$

of the factors of the core signature (Definitions 1.3 and 1.35), with the conventions of Definition 1.38: the cross-commons-canonical factor $\mathbb{S}^*$*, the space of the content that commons exchange, with value $s^*(t) \in \mathbb{S}^*$ the* cross-commons canonical content at time t*; the* per-commons factor $\mathbb{S}^c$*, the space of states of the shared repository of commons c, which the entities of $\mathcal{E}^c$ write; the* agent-private factor $\mathbb{P}^{(v)}$*, digital agent v's private-memory space; and the* broadcast factor $\mathbb{B}^{(v)}$*, the space of v's broadcast σ-events visible to a subset $S \supseteq \{v\}$ of entities. A* state value *$s \in \mathbb{S}$ is a tuple $s = (s^*, (s^c)_c, (p^{(v)})_v, (b^{(v)})_v)$ with each component a value in the corresponding factor-space; the* state value at time t *is $s(t) = (s^*(t), (s^c(t))_{c\in\mathbb{I}}, (p^{(v)}(t))_{v\in V}, (b^{(v)}(t))_{v\in V})$, with the values of the core signature. The* bundle process *is the discrete-time process $\{s(\tau_n)\}_{n\geq 1}$ on $\mathbb{S}$, where τ_n is the timestamp of the n-th σ-event in a linear extension of the journal order $\preceq_\sigma$ (Definition 1.38, Axis 4), with transition labels drawn from the σ-event-type vocabulary $\mathcal{V}_\sigma$.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.41 (Sync operator)**.** *The* sync operator Σ *reconciles parallel views of the state, over the core signature (Definitions 1.3, 1.33 and 1.35). It takes two forms:*

- Σ^c *— within-commons sync. For a commons $c \in \mathbb{I}$, a* sync epoch *is cut at a time t_0, and its* live writers *are the digital agents $v \in \mathrm{active}(t_0) \cap V^c$ and the operators $o \in O^c$. Each live writer p holds a* view*, a finite set of commits of $\mathbb{S}^c$:* $\mathrm{pre}(p)$ *is its view at the cut, and* $\mathrm{recv}(p)$ *is the set of live writers whose views reach p during the epoch. Σ^c gives p the post-epoch view*

$$\mathrm{post}(p) \;=\; \mathrm{pre}(p) \,\cup \bigcup_{q\in\mathrm{recv}(p)} \mathrm{pre}(q).$$

- Σ *— across-commons sync. For a declared peer-set $\mathcal{Q} \subseteq \mathbb{I}$, a sync epoch is cut at a time t_0 and has a sync window of length $w \geq 0$. Each commons $k \in \mathcal{Q}$ holds a view* $\mathrm{pre}(k)$ *at the cut, a finite set of content of $\mathbb{S}^*$; k decides the content* $\mathrm{offer}(k, k')$ *it offers each peer k', and k' decides the content* $\mathrm{take}(k', k)$ *it takes from k, over a secure transport;*

$\mathrm{arr}(k,k')$ *is the time at which the offer of* k *arrives at* k'*. What passes from* k *to* k' *is* $\mathrm{pass}(k,k') = \mathrm{pre}(k) \cap \mathrm{offer}(k,k') \cap \mathrm{take}(k',k)$*, and* Σ *gives* k' *the post-epoch view*

$$\mathrm{post}(k') \;=\; \mathrm{pre}(k') \cup \bigcup_{k \in \mathcal{Q},\ \mathrm{arr}(k,k') \le t_0 + w} \mathrm{pass}(k,k').$$

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.42 (Write-through operator)**.** *For a commons* c *and a digital agent* $v \in V^c$ *(Definitions 1.3 and 1.35), let* Action_v *be the set of actions of* v *and* Ev^c *the set of* σ*-events of* c*. The* write-through operator *of* v *is a partial map*

$$\sigma_v : \mathrm{Action}_v \;\rightharpoonup\; \mathrm{Ev}^c.$$

An action $a \in \mathrm{Action}_v$ *is* σ-recorded *when* $a \in \mathrm{dom}\,\sigma_v$*, and then* $\sigma_v(a)$ *is the* σ*-event that records* a*, written into the per-commons factor* $\mathbb{S}^c$ *of* c*; its type lies in the* σ*-event-type vocabulary* $\mathcal{V}_\sigma$*, which fixes its payload schema.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.43 (σ-chain)**.** *The* σ-chain *of a digital agent* $v \in V^c$ *in commons* c *(Definition 1.3) is the sequence* $\Xi^{(v,c)} = (E_1^{(v,c)}, E_2^{(v,c)}, \ldots)$*, finite at each time, of the* σ*-events that the write-through operator* σ_v *of Definition 1.42 has recorded for the* σ*-recorded actions of* v*, in the order recorded, together with the* chain-link map $\mathrm{prev}(E_n^{(v,c)}) := E_{n-1}^{(v,c)}$ *for* $n \ge 2$*, undefined at* $n = 1$*. The* commons σ-chain *of* c *is the sequence of the events of all* $\Xi^{(v,c)}$*,* $v \in V^c$*, ordered lexicographically by timestamp* at*, then by emitter* actor_agent *in the fixed order on* $\mathcal{E}^c$ *(both of Definition 1.3), then by position* n *in the emitter's own chain.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.44 (IAS substrate-correctness decision class)**.** *Let* Trajectory *be the set of finite substrate trajectories (Definition 1.38, Axis 4). For a commons* c*, a* substrate-correctness check *of* c *is a map* $f : \mathrm{Trajectory} \to \{0,1\}$ *computed by a deterministic algorithm that the specification* Spec^c *(Definition 1.3) declares as a check, independently of the implementation language.* $\mathcal{D}_{\mathrm{IAS}}$ *is the class of the substrate-correctness checks of the commons* $c \in \mathbb{I}$*.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.45 (Substrate structure and its sub-signatures)**.** *Let* $\mathcal{X}$ *be the primary state space of the substrate (Definition 1.35). The* substrate signature *is the finite many-sorted signature* $\Sigma = \bigcup_{j \in J} \Sigma_j$*, fixed with the chain, whose sub-signatures, indexed by the names* $j \in J$ *below, name the framework's coordinates:* Σ_{OBG} *(the coverage graph over obligation-nodes and file/source regions),* Σ_{health} *(the node-health coordinate: a level in* $\{\mathsf{Unhealthy} < \mathsf{Advisory} < \mathsf{Structural}\}$ *for each obligation-node),* Σ_Φ *(the integer and description-length potentials),* Σ_{cap} *(the capability lattice),* Σ_{gate} *(the governance literals),* Σ_{os} *(the OS root-tree carrier: the host-OS security posture, namely the kernel and process table, the filesystem root* /*, and its resource-state),* Σ_{immune} *(the immune coordinate: the record of the threats known and the defences held over the operating system),* Σ_Θ *(the operational vector — metered-channel usage, context-used, and freshness over the actively-edited documents — its health carried by* Σ_{health}*),* Σ_σ *(the* σ*-chain, Definition 1.43),* $\Sigma_{\mathrm{lineage}}$ *(the lineage coordinate), and the byte-content carrier* Σ_{byte} *(finite bit-strings). A substrate state* $x \in \mathcal{X}$ *is presented as a finite* Σ*-structure* $\mathfrak{S}_x$*, the* substrate structure *at* x*, whose byte-content reduct* $\mathfrak{S}_x \restriction \Sigma_{\mathrm{byte}}$ *is the bit-sequence* x *itself; we write* $\mathfrak{S}$ *for it*

when x is fixed, and $\mathbb{X} = \{\mathfrak{S}_x : x \in \mathcal{X}\}$ for the state space so presented, identified with $\mathcal{X}$ by $x \mapsto \mathfrak{S}_x$, a set of Σ-structures whose sizes are bounded by the finite store. Each cluster view *is the reduct $\mathfrak{S} \restriction \Sigma_j$; the coverage graph is $\mathfrak{S} \restriction \Sigma_{\mathrm{OBG}}$.*

Base: none cited (the empty base).

Definition 1.46 (Metacircular fixed point)**.** *For a commons c, let $C^c : \mathrm{Spec} \times 2^{\mathrm{Prop}} \to \mathrm{Spec}$ be its coevolution operator and accept_c its acceptance at a fixed time (Definition 1.26). A specification $\mathrm{Spec}^* \in \mathrm{Spec}$ is a* metacircular fixed point *of c iff*

$$\mathrm{Spec}^* \;=\; C^c\big(\mathrm{Spec}^*,\ \mathrm{accept}_c(\mathrm{Spec}^*)\big).$$

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.47 (Coverage graph (the OBG reduct))**.** *The* coverage graph *is the reduct $\mathfrak{G} = \mathfrak{S} \restriction \Sigma_{\mathrm{OBG}}$ of the substrate structure $\mathfrak{S}$ to its obligation-graph sub-signature Σ_{OBG} (Definition 1.45): a finite many-sorted structure $\mathfrak{G} = (N, R;\ E, s, t, \tau, \alpha)$ over disjoint sorts* Node *(carrier N = obligation-nodes, Definition Definition 1.14) and* Region *(carrier R = file-regions $(file, anchor)$), with edge-set E, source/target maps $s, t : E \to N \uplus R$, a type map $\tau : E \to T$ into the finite type graph $T = \{\mathrm{Dep}, \mathrm{Tracks}, \mathrm{Hydrates}, \mathrm{Produces}, \mathrm{Emits}, \mathrm{Owns}\}$ with the single inheritance covering $\mathrm{Emits} \sqsubseteq \mathrm{Produces}$, and an attribute map $\alpha : E \to A$ into σ-timestamps. For $k \in T$ write $E_k = \tau^{-1}(k)$ for the edges of type k. Each type declares endpoint sorts — $\mathrm{ends}(\mathrm{Dep}) = (Node, Node)$, $\mathrm{ends}(\mathrm{Tracks}) = (Region, Region)$, and $\mathrm{ends}(k) = (Node, Region)$ for the four coverage kinds — and $(s(e), t(e))$ inhabits $\mathrm{ends}(\tau(e))$. Two well-formedness constraints ride in the signature:* Emits *is single-valued into N per region (a partial function $R \rightharpoonup N$); and $\{t(e) : s(e) = n,\ \tau(e) = \mathrm{Hydrates}\} = \mathrm{Hyd}(n)$ for every $n \in N$ (Definition 1.14).*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.48 (Coverage and input-groundedness)**.** *Let $\mathfrak{G} = (N, R;\ E, s, t, \tau, \alpha)$ be the coverage graph (Definition 1.47), and let $R_{live} \subseteq R$ be the regions whose file and anchor exist in the current state. For $n \in N$ put $\mathrm{owns}(n) = \{t(e) : s(e) = n,\ \tau(e) = \mathrm{Owns}\} \subseteq R$; $\mathfrak{G}$ is* covering *iff $R = \bigcup_n \mathrm{owns}(n)$. $\mathfrak{G}$ satisfies the* live invariant *iff every region in some $\mathrm{Hyd}(n)$ (Definition 1.14) is the target of some* Tracks *edge. $\mathfrak{G}$ is* input-grounded *iff $t(E_{\mathrm{Hydrates}} \cup E_{\mathrm{Tracks}}) \subseteq R_{live}$.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.49 (Integer coverage-potential)**.** *For a state presented by its coverage graph $\mathfrak{G}$ (Definition 1.47) with vertex set $V = N \uplus R$, edges E, a fixed block size $B \in \mathbb{N}_{>0}$, and content sizes $\mathrm{blob} : R \to \mathbb{N}$, the* integer coverage-potential *is*

$$\Phi_{\mathrm{int}} = |V| + |E| + \textstyle\sum_{r \in R} \lceil \mathrm{blob}(r)/B \rceil \;\in\; \mathbb{N},$$

and its lexicographic refinement *is $(\,|V|, |E|, \sum_r \lceil \mathrm{blob}(r)/B \rceil\,) \in \mathbb{N}^3$.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.50 (Canonicality potential)**.** *For a state $x \in \mathbb{X}$ (Definition 1.45) presented by its coverage graph $\mathfrak{G}$ (Definition 1.47), let $E_k = \tau^{-1}(k)$ be the edges of type k, types-present$(\mathfrak{G}) = \{k : E_k \neq \emptyset\}$, src_k and tgt_k the carriers of the endpoint sorts of k, $\ell_{\mathbb{N}}$ a fixed universal code length for positive integers, and $\mathrm{blob} : R \to \mathbb{N}$ the content size of each region in bytes. The* canonicality potential *is $\Phi_{\mathrm{MDL}} : \mathbb{X} \to \mathbb{R}_{\geq 0}$, $\Phi_{\mathrm{MDL}} = L(structure) + L(content \mid structure)$,*

where $L(\textit{structure}) = \sum_{k \in \textit{types-present}(\mathfrak{G})} \left(\ell_{\mathbb{N}}(|E_k|) + |E_k| \cdot \log_2(|\mathrm{src}_k| \cdot |\mathrm{tgt}_k|)\right)$ *and* $L(\textit{content} \mid \textit{structure}) = \sum_r 8\,\mathrm{blob}(r)$ *bits.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.51 (Repair rules as double-pushout (DPO) productions)**.** *Over the coverage graph's typed-attributed structure (Definition Definition 1.47;* `Emits` $\sqsubseteq$ `Produces`*, attribute* `checked_at`*), the* repair system $\to_{\mathrm{rep}}$ *is three productions: (1)* Death *— delete a vertex, with all its incident edges, under the global side condition, checked on the whole graph outside the DPO formalism, that the vertex is unreachable from every obligation-node root; (2)* Merge *— a DPO production: where a node carries both a* `Produces` *and an* `Emits` *edge to the same target, delete the* `Produces` *edge; (3)* Dedup *— a DPO production: of two parallel edges equal under the key* `(from, to, type, checked_at)`*, delete one.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.52 (The Φ-split)**.** *The measure attached to the* repair rewrite relation $\to_{\mathrm{rep}}$ *(Definition 1.51) is the integer coverage-potential* Φ_{int} *(Definition 1.49). The* canonical *representative of a state is the least of the* $\to_{\mathrm{rep}}$*-normal forms reachable from it (the states reachable by* $\to_{\mathrm{rep}}$ *from which no further step applies) in the order that compares first by* Φ_{MDL}*, the Minimum Description Length (MDL) potential of Definition 1.50, and breaks ties by a fixed total order* key *on states; it is defined when that set is non-empty and otherwise undefined.*

Base: `Axm-HIC`.

Definition 1.53 (Operator-prefix Ulam–Harris label)**.** *Throughout,* $\mathbb{N} = \{0, 1, 2, \ldots\}$*. For a commons* $c \in \mathbb{I}$ *(Definitions 1.3 and 1.33), its* commons-id *is a number in* $\mathbb{N}_{>0}$*, a name for the commons' self-certifying identity (Definition 1.25); in labels,* c *stands for its commons-id. Let* $i \in \mathbb{N}$ *index a human operator within* c*, its* operator-index*. For each operator* O_i^c *and time* t *let* $J^{c,i}(t) \subseteq \mathbb{N}$ *be the finite set of indices of the* root labels *of* O_i^c *at* t*, non-decreasing in* t*, written* $J^{c,i}$ *where the time is fixed. Each root label is* A^c_{ij} *with* $j \in J^{c,i}$*, and the descendants of the agent that carries it carry* $A^c_{ij\,\ell_1\ell_2\ldots\ell_n}$*, where* $(\ell_1, \ldots, \ell_n) \in \mathbb{N}^n_{>0}$ *is the Ulam–Harris path in the operator-rooted tree* $\mathcal{G}^{c,ij}_{\mathrm{op}}$*, whose vertices are the labels* $A^c_{ij\,\ell_1\ldots\ell_n}$*,* $n \geq 0$*, with an edge from each to each of its one-letter extensions. The leading* A^c_{ij} *is the* operator-prefix*. We write* u *for the word* $ij\,\ell_1 \ldots \ell_n$ *and* A^c_u *for a label of either form. The* roster *of* O_i^c *is the set of labels* A^c_u *whose word* u *has first letter* i*.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.54 (Reproduction event of arity ≥ 1 with inheritance parameter κ)**.** *A* reproduction event *in commons* c *(Definitions 1.3, 1.33 and 1.35) for receiving operator* O_i^c *is a tuple* $(\mathcal{P}, \rho, \kappa, A^c_{u'})$*:*

1. *the* parent set $\mathcal{P} = \{A^{c_p}_{u_p}\}_{p=1}^n$*, finite with* $n \geq 1$*, of agents of commons in* $\mathbb{I}$*, each given by a* σ-checkpoint*, its state at one of its* σ*-events, which may be that of any ancestor, not only of the immediate generation; the parents are admitted by a compatibility predicate* compat *of* ρ *on* σ*-checkpoints,* $\mathrm{compat} = 1$ *on the* σ*-checkpoint of each parent;*
2. *the* reproduction procedure ρ*, a deterministic procedure that takes the parents' restricted snapshots and the anchoring commit* κ *of item 3 and returns the child's seed state, an element of* $\mathbb{P}^{(A^c_{u'})} \times \mathbb{B}^{(A^c_{u'})}$ *(Definitions 1.3, 1.33 and 1.35);* ρ *is itself versioned by a commit reference;*

3. *the* inheritance parameter κ*, a commit of* $\mathbb{S}^c$ *at which the event is anchored, with per-parent commits* κ_p*; the seed state, at the child's first active time* b_1 *(Definition 1.39), is*

$$s_{A^c_{u'}}(b_1) = \rho\big(\{\text{snapshot}(\mathbb{S}^{c_p}, \kappa_p)\restriction_{V^{c_p,i_p}} : A^{c_p}_{u_p} \in \mathcal{P}\},\ \kappa\big),$$

where $\text{snapshot}(\mathbb{S}^{c_p}, \kappa_p)$ *is the state of* $\mathbb{S}^{c_p}$ *at the commit* κ_p*,* i_p *is the operator index of* u_p *(Definition 1.53),* $V^{c_p,i_p} \subseteq V^{c_p}$ *is the set of agents whose label has operator index* i_p*, and, for a set* W *of agents of* c_p*,* $X \mapsto X\restriction_W$ *is a declared restriction of the states of* $\mathbb{S}^{c_p}$ *to the part that belongs to the agents of* W*;*

4. *the* child $A^c_{u'}$*. At arity* 1 *the parent* A^c_u *lies in* c *under* O^c_i*,* ρ *is snapshot-and-restrict,* $\kappa_p = \kappa$*, and the child is* $A^c_{u\,k}$*,* $k \in \mathbb{N}_{>0}$*, on the parent's offspring edge: classical single-parent reproduction. At arity* ≥ 2 *the child is a* recombinant $A^c_{ij'}$*, with a fresh index* $j' \notin J^{c,i}$ *that is then added to* $J^{c,i}$*, and* O^c_i *may differ from every parent's operator.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.55 (The sow family)**.** *Let* $\text{sow}(\omega, u; m)$ *emit a composite seed carrying content* ω *into a target substrate* u*, germinating if* u *is receptive, with mode* $m \in \{\textsf{instantiate-new-self}, \textsf{update-existing}\}$ *and content* $\omega \in \{\textsf{private-genome}, \textsf{public-core}\}$*. Reproduction (Definition 1.54) is* $\text{sow}(\textsf{private-genome}, \cdot\,; \textsf{instantiate-new-self})$*: germination instantiates a new selfhood predicate* χ_P *(Definition 1.5). Dissemination along the disseminability order (Definition 1.17) is* $\text{sow}(\textsf{public-core}, \cdot\,; \textsf{update-existing})$*: germination updates an existing self at its reception-grade.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.56 (Directed Lineage Graph (DLG) of the commons set)**.** *Let* $\mathbb{I}$ *be the commons index set (Definition 1.3). The* Directed Lineage Graph *(DLG) of* $\mathbb{I}$ *is the projection of the joint substrate trajectory* $\{s(\tau_n)\}$ *(Definition 1.38) onto its lineage* σ*-events, the agent-creation events of Definition 1.54. It is the directed graph* $\mathcal{G}_{\text{DLG}} = (V_{\text{DLG}}, E_{\text{DLG}})$ *with:*

- *vertices* $V_{\text{DLG}} = \bigcup_{c \in \mathbb{I}} V^c$*, the union of the agent rosters of all commons, each vertex labelled by its commons and its operator; the DLG also carries its* label set*, the commons of* $\mathbb{I}$ *with their operators, which may include an operator with no agent yet;*
- *a* reproduction edge $A^c_u \to A^c_{u\,k}$ *for each reproduction of arity* 1 *producing offspring* $A^c_{u\,k}$ *from parent* A^c_u*;*
- *for each recombination of arity* ≥ 2 *producing a recombinant* $A^c_{ij'}$ *(Definition 1.54) from a parent set* $\mathcal{P} = \{A^{c_p}_{u_p}\}_p$*, one* recombination edge $A^{c_p}_{u_p} \to A^c_{ij'}$ *per parent, tagged with the recombination procedure* ρ_{recomb} *and its commit reference* κ_{recomb}*. The parents are drawn from the vertices present at the time of the event; a parent's snapshot may be any earlier* σ*-checkpoint.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.57 (Root operator)**.** *For a commons* $c \in \mathbb{I}$ *(Definitions 1.3 and 1.33), a* succession *is a* σ*-event of* c*, emitted by an operator of* O^c*, that names an operator of* O^c*. The* root operator *of* c *at time* t *is the operator named by the last succession at or before* t *when that operator is in* $O^c(t)$*, and is otherwise undefined; when there is no succession at or before* t*, it is* O^c_0*, the first operator in* c*'s* σ*-history at or before* t*, when* $O^c_0 \in O^c(t)$*, and it is undefined otherwise and when that history contains no operator.*

On the side of digital agents, a root agent *is a lineage-root of the DLG (Definition 1.56): a vertex of in-degree* 0 *over all its lineage edges, reproduction and recombination edges alike.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.58 (The empty DLG and the four generators)**.** *Directed lineage graphs (DLGs) and their label sets are as in Definition 1.56. Empties are typed: the empty tree* $\emptyset_{\mathrm{Tree}}$*, the empty index set* $\emptyset_{\mathbb{N}}$ *and the empty DLG* $\emptyset_{\mathrm{DLG}}$ *are distinct objects. The empty DLG* $\emptyset_{\mathrm{DLG}}$ *is the DLG with no vertices whose label set is one commons with its root operator* O_0^c*, whose roster is empty (*$J^{c,0} = \emptyset_{\mathbb{N}}$*). The DLG* $\mathbf{1}_{\mathrm{DLG}}$ *has exactly one vertex, a root agent (Definition 1.57), and no edges.*

Four operations act on DLGs:

1. `addOp` *adjoins an operator* $O_{i'}^{c'}$ *of a new or an existing commons* c' *to the label set, with empty roster;*
2. `addRoot` *adjoins to an operator's roster a root agent* $A_{ij'}^{c}$*, a new vertex with no in-edges and a fresh index* j' *added to* $J^{c,i}$*, starting the singleton tree* $\{A_{ij'}^{c}\}$*;*
3. `repro` *is a reproduction of arity* 1 *(Definition 1.54), extending an operator-rooted tree;*
4. `recomb` *is a reproduction of arity* ≥ 2*, adding recombination edges that may cross root, operator or commons boundaries.*

The class of constructible DLGs *is the closure of* $\{\emptyset_{\mathrm{DLG}}\}$ *under these four operations, and a* construction history *of a DLG* F *in the class is a finite sequence of them from* $\emptyset_{\mathrm{DLG}}$ *to* F*.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.59 (Arity-0 birth — the fresh root completing the arity spectrum)**.** *An* arity-0 birth *is an* `addRoot` *step of the DLG arithmetic (Definition 1.58) with empty parent set* $\mathcal{P} = \emptyset$*: it adds a root agent (Definition 1.57), a vertex of in-degree* 0 *in the lineage graph, instantiated from the pre-commons* template scaffold *alone and inheriting no parent snapshot.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.60 (Substrate subgraphs)**.** *The substrate has three named subgraphs alongside the DLG (Definition 1.56).*

- *The* operator-tree subgraph $\mathcal{G}_{\mathrm{op}}$*, induced by reproduction edges, with recombination inside one operator's roster projected to the identity.*
- *The* memory subgraph $\mathcal{G}_{\mathrm{mem}}$*, induced by the inheritance of per-label private memory, copied at reproduction time.*
- *The* repository subgraph $\mathcal{G}_{\mathrm{repo}}$*, induced by inheritance of the shared repository (designs, tools, plans) through version-control operations.*

Their superposition is the substrate lineage graph $\mathcal{G}_{\mathrm{op}} \cup \mathcal{G}_{\mathrm{mem}} \cup \mathcal{G}_{\mathrm{repo}}$*.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.61 (In-transit window)**.** *Let* A_u^c *be a digital agent of commons* c *and* $h_A \neq h_B$ *two hosts of* $\mathcal{H}_c$ *(Definitions 1.3 and 1.35). A* lock transfer *of* A_u^c *from* h_A *to* h_B *is a pair of* σ*-events of* c*: a* lock-transfer.in_transit *event of* A_u^c *whose payload names* h_A *as the source host and* h_B *as the target host, and the next event of* c *that ends that transfer, either a* lock-transfer.completed *event of* A_u^c *whose payload names* h_B*, or an event that records the fail-back of the transfer to* h_A*; the types of these events fix payload schemas with these agent and host fields (Definition 1.42).*

The interval between their times is the in-transit window *of the transfer; it is unbounded when no ending event follows.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.62 (Current host tag)**.** *Let $\mathcal{H}_c$ be the hosts of commons c (Definitions 1.3 and 1.35) and let A_u^c be a digital agent (Definition 1.53). For a host $h \in \mathcal{H}_c$, a* lease *of A_u^c on h is a pair of σ-events of c: a lock acquisition of A_u^c at a time t_0 whose payload names h as the acquiring host, and the next lock release of A_u^c whose payload names h, at a time $t_1 > t_0$, with $t_1 = \infty$ when no such release follows. Here the lock-acquisition and lock-release types of $\mathcal{V}_\sigma$ fix a payload schema (Definition 1.42) with two fields, the agent whose lock is taken or released and a host of $\mathcal{H}_c$; the host is a field of the payload, not the emitter of the event, which is an entity of $\mathcal{E}^c$. Write the lease $(h, A_u^c, [t_0, t_1))$. Write* $\mathrm{holds}(h, A_u^c, t)$*, read "$h$ holds the lock of A_u^c at t", when some lease $(h, A_u^c, [t_0, t_1))$ of A_u^c on h has $t \in [t_0, t_1)$. The* current host tag *of an agent A_u^c is the function of the instant $t \in \mathbb{R}_{\geq 0}$*

$$\mathrm{host}(A_u^c,\, t) \;\in\; \mathcal{H}_c \cup \{\mathrm{in_transit},\, \mathrm{inactive},\, \bot\}$$

defined by the first case that applies: $\mathrm{host}(A_u^c, t) = \mathrm{in_transit}$ *when t lies in the in-transit window of a lock transfer of A_u^c (Definition 1.61); otherwise* $\mathrm{host}(A_u^c, t) = \mathrm{inactive}$ *when $t \notin I(A_u^c)$ (Definition 1.39); otherwise* $\mathrm{host}(A_u^c, t) = h$ *when h is the only host of $\mathcal{H}_c$ with* $\mathrm{holds}(h, A_u^c, t)$*; otherwise* $\mathrm{host}(A_u^c, t) = \bot$*, when no host, or more than one, holds the lock.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.63 (Air-gappable substrate)**.** *A host $h \in \mathcal{H}_c$ of commons c (Definitions 1.3 and 1.35) is* air-gappable *if the operator can physically isolate it from external network in bounded time $T_{\mathrm{isolate}}(h) < \infty$ via an operator-attestable procedure. The* isolation latency $T_{\mathrm{isolate}}(h)$ *is host-specific and operator-declared in the host-manifest of $\mathcal{H}_c$.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.64 (Moat tier)**.** *Let $\mathcal{H}_c$ be the hosts of commons c (Definitions 1.3 and 1.35), each carrying an air-gappability property in the sense of Definition 1.63. A* tier assignment *of c is a map* $\mathrm{tier} : \mathcal{H}_c \to \{0, 1, \ldots, \max\}$ *with* $\max \geq 1$*, operator-declared and σ-attested at host-onboarding (event type* `agent.host.tier_attested`*); write $T_k := \mathrm{tier}^{-1}(k)$ for its* moat tiers*, ordered by isolation-radius from the commons' external-facing network. It is a* moat-tier assignment *when:*

- T_0 — *external-facing tier: hosts directly reachable from the public Internet.*
- T_k *for* $1 \leq k < \max$ — *moat tier: hosts reachable only through T_{k-1}, with progressively reduced network reachability.*
- $T_{\max}$ — *sanctum tier: hosts with no inbound reachability from any lower tier; only outbound mirror-channel writes to the immediate-lower tier permitted.*

Base: `Axm-HIC`.

Definition 1.65 (JEM — Jurisdiction-Explicit Model, general)**.** *Let $\mathcal{J}$ be a finite set of sovereign States. Let R be the* ***resource set*** *of an operation-class and* Ops *its operations, each conducted by a* ***principal*** *— the natural or legal person who runs or commissions it (an operator $o \in O^c$ of a commons c (Definitions 1.3 and 1.33), directly or through a digital agent, or a legal person on whose behalf an operator acts). A* ***Jurisdiction-Explicit Model*** *is a tuple $J = (R, \mathcal{J}, \jmath)$*

whose ***sovereignty assignment*** $\jmath = (\jmath_{\text{deploy}}, \jmath_{\text{own}}, \jmath_{\text{prov}}, \jmath_{\text{princ}})$ *assigns enforcing States to the resources and to the conducting principal.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.66 (AMJEM — Abstract-Machine Jurisdiction-Explicit Model)**.** *AMJEM is JEM (Definition 1.65) specialised to an abstract machine. Let A be an abstract machine (AM) — one of* SRAM, PRAM, DPRAM *— read with a work–depth cost semantics (the machine together with that cost semantics is an abstract-machine* model*, AMM), and let $\mathcal{J}$ be a finite set of sovereign States. Let $R = R(A)$ be the* ***resource set*** *of a computation on A — its processors, memory cells, and communication links — and let T denote the time index of the σ-chain (Definition 1.43); write* Ops *for the set of operations (edge-writes on* SPIG*, Definition 1.27), each conducted by a* ***principal*** *— the natural or legal person who runs or commissions it (an operator $o \in O^c$ of a commons c (Definitions 1.3 and 1.33), directly or through a digital agent, or a legal person on whose behalf an operator acts). An* ***Abstract-Machine Jurisdiction-Explicit Model (AMJEM)*** *is a tuple $J = (A, \mathcal{J}, \jmath)$ whose* ***sovereignty assignment*** $\jmath = (\jmath_{\text{deploy}}, \jmath_{\text{own}}, \jmath_{\text{prov}}, \jmath_{\text{princ}})$ *comprises three maps on the resource set R — assigning enforcing States to each resource $r \in R$ — together with a fourth on the conducting principal:*

- $\jmath_{\text{deploy}} \colon R \to \mathcal{J}$*;* $\jmath_{\text{deploy}}(r)$ *is the State in whose territory the machines serving r physically stand;*
- $\jmath_{\text{own}} \colon R \times T \to 2^{\mathcal{J}}$*;* $\jmath_{\text{own}}(r, t)$ *is the (possibly empty) set of States that at time t can legally compel the company owning or serving r;*
- $\jmath_{\text{prov}} \colon R \to 2^{\mathcal{J}}$*;* $\jmath_{\text{prov}}(r)$ *is the set of provenance States whose influence was fixed into r at construction (for a model artifact, the training-origin jurisdiction);*
- $\jmath_{\text{princ}} \colon \text{Ops} \times T \to 2^{\mathcal{J}}$*;* $\jmath_{\text{princ}}(\omega, t)$ *is the set of States with coercive jurisdiction at time t over the natural or legal person conducting operation ω — their home jurisdiction (citizenship, residence, incorporation, or seat) together with any State asserting extraterritorial sanction reach over them.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.67 (DPRAMJEM)**.** *DPRAMJEM is the AMJEM (Definition 1.66) at $A =$ DPRAM, the distributed parallel random-access machine (DPRAM), i.e. $(\text{DPRAM}, \mathcal{J}, \jmath)$. A DPRAMJEM instance is* State-pinned *to s, written* ***DPRAMJEM-s****, when $\jmath_{\text{deploy}}$ maps every resource to the single State s.*

Base: `Axm-HIC`.

Definition 1.68 (Jurisdiction modes)**.** *The* jurisdiction modes *of a jurisdiction-explicit model (Definition 1.65), dialled per matter and per coordinate of its sovereignty assignment $\jmath$, are: γ (agnostic) — jurisdiction is not a constraint; the operation is optimised for capability or cost and may run, source, or sell wherever, with the coordinates still available to log; α (aware) — the applicable enforcing States are computed and logged to the σ-chain per coordinate, observed but not acted on; β (govern-by / explicit) — the operator acts on the jurisdiction, in two composable behaviours:* **pin** *— constrain $\jmath_{\text{deploy}}$ or $\jmath_{\text{own}}$ to chosen States, the* sovereignty *behaviour; and* **abide** *— conform the operation's conduct to the law of the applicable jurisdiction, the* compliance *behaviour.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.69 (governed taskset schedule)**.** *A* ***plan*** *is a directed graph of tasks $G = (V, E)$*

with E its precedence edges, cycles allowed; a schedule runs on its condensation *— one node for each component, a maximal set of tasks each reachable from every other along E, and an edge from one component to another when E has an edge from a member of the first to a member of the second — each component a node whose work is the sum of its members' work; each $v \in V$ carries a work $w(v) \in \mathbb{N}_{>0}$ (its work–depth cost, a count of unit operations). A* ***schedule*** *assigns each v to a digital agent and executes G along the Kahn wavefronts of its condensation (the level decomposition, one level advancing the critical path by one). Under the govern-by mode β (Definition 1.68) read on the $\jmath_{\text{princ}}$ coordinate, a task v in a* ***human-oversight class*** *carries a* ***ratification gate****: its execution is preceded by an authorisation from the principal conducting the operation $\omega(v)$ that executes v (Definition 1.65), whose jurisdictions at time t are $\jmath_{\text{princ}}(\omega(v), t)$ (Definition 1.66), incurring an authorisation latency $\lambda(v) \in \mathbb{N}$, logged to the σ-chain (Definition 1.43). Under γ (agnostic) or α (aware), and for any class the operator's dial leaves autonomous (ratified by a digital agent), $\lambda(v) = 0$.*

Write $W = T_1$ for the total work $\sum_{v \in V} w(v)$ and $D = T_\infty$ for the ungoverned depth, the work on the heaviest chain of the condensation; write D_β for the **governed depth** — the longest path under node weights $w(v) + \lambda(v)$.

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.70 (IPLiKiDi projection)**.** *Let $G = (V_T, E_T)$ be the plan of tasks (Definition 1.69), cycles allowed. In each substrate state the capability atoms form a finite set* At*, the concrete capability paths of that state. Let* $(\text{Cap}, \subseteq)$ *be the capability lattice of the state: the subsets of* At*, ordered by inclusion, with join the union and meet the intersection. An allowlist pattern, wildcards included, stands for the set of atoms of* At *it matches. The* IPLiKiDi projection *is the map*

$$\varphi : 2^{V_T} \to \text{Cap}, \qquad \varphi(T) \;:=\; \bigcup_{t \in T} \text{allowlist}(t),$$

where $\text{allowlist}(t) \subseteq \text{At}$ *is the set of atoms matched by the allowlist patterns of task t. The* active taskset *at time τ is the set* $\text{active-taskset}(\tau) \subseteq V_T$ *of the tasks running at τ.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.71 (Session credential)**.** *A* session credential *is a signed tuple*

$$\kappa \;=\; \big(\lambda,\ sid,\ h,\ j,\ [t_0, t_1),\ \Phi\big)$$

— canonical label $\lambda \in V^c$ (Definition 1.3), session-id sid, host h, jurisdiction j, validity-interval $[t_0, t_1)$, and capability-envelope $\Phi \in \text{Cap}$ (the lattice of Definition 1.70) — minted at a `session_binding` *(Definition 1.23), signed under the current key of the commons (Definition 1.25), directly or by a daemon holding a signing certificate delegated under that key, and released or expired at the paired* `session_release`*, when one follows. Its envelope is bounded by the active projection at its mint time, $\Phi \subseteq \varphi(\text{active-taskset}(t_0))$ (Definition 1.70).*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.72 (Deny-tier: relative complement on Cap)**.** *Let* Cap *be the capability lattice of a substrate state (Definition 1.70), the subsets of its finite atom set. Extend* $(\text{Cap}, \subseteq, \cup, \cap, \emptyset)$ *with the binary* relative complement $\setminus$*: for $C_1, C_2 \in \text{Cap}$, $C_1 \setminus C_2 := C_1 \setminus C_2$, the atoms of C_1 that are not in C_2. The root-of-trust denies form a distinguished element* $\text{Deny} \in \text{Cap}$.

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.73 (Egress-confinement: the deny-tier on the context lattice)**.** *Let* $(\mathrm{Cap}_{\mathrm{ctx}}, \subseteq, \cup, \cap, \emptyset)$ *be the generalized-Boolean algebra of* context-grants *— the substrate slices (memory-slices, files, prior σ-results) admissible into an LLM-interaction sub-process's prompt — and let the* context-hydration projection $\varphi_{\mathrm{ctx}} : 2^{V_T} \to \mathrm{Cap}_{\mathrm{ctx}}$ *send a taskset $T \subseteq V_T$ (an active taskset of Definition 1.70) to the union of its nodes' declared* required-context *(the hydration scope). Let each LLM-interaction sub-process (Definition 1.22) carry a* trust tier*, the tiers ordered by egress-exposure,*

$$\text{sovereign-local} \sqsubseteq \text{sovereign-hosted} \sqsubseteq \text{vendor-proprietary} \sqsubseteq \text{third-party-intermediary},$$

and to each tier τ assign a sensitive-deny $\mathrm{Deny}_{\mathrm{sens}}(\tau) \in \mathrm{Cap}_{\mathrm{ctx}}$ *— the context that must not egress to a sub-process at τ — monotone in exposure,* $\mathrm{Deny}_{\mathrm{sens}}(\textit{sovereign-local}) = \emptyset \subseteq \cdots \subseteq \mathrm{Deny}_{\mathrm{sens}}(\textit{third-party-intermediary})$*. The* egress-confined hydration scope *of a sub-process at tier τ executing taskset T is*

$$\varphi^{\mathrm{enf}}_{\mathrm{ctx}}(T, \tau) \;:=\; \varphi_{\mathrm{ctx}}(T) \setminus \mathrm{Deny}_{\mathrm{sens}}(\tau),$$

the relative complement of Definition 1.72 on $\mathrm{Cap}_{\mathrm{ctx}}$*. Let* $\mathrm{ctx}_{\min}(T) \subseteq \varphi_{\mathrm{ctx}}(T)$ *be the minimal context under which T is executable; tier τ is* egress-sufficient *for T iff the confined scope still covers the need,* $\varphi_{\mathrm{ctx}}(T) \setminus \mathrm{Deny}_{\mathrm{sens}}(\tau) \supseteq \mathrm{ctx}_{\min}(T)$*, and the* routing rule *sends T to the $\sqsubseteq$-greatest egress-sufficient tier, which exists because sovereign-local, with* $\mathrm{Deny}_{\mathrm{sens}} = \emptyset$ *and* $\mathrm{ctx}_{\min}(T) \subseteq \varphi_{\mathrm{ctx}}(T)$*, is always egress-sufficient.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.74 (Substrate-fault classes for a compiled capability boundary)**.** *Let the capability projection φ of Definition 1.70 be enforced through the chain: policy source on disk → compiled bundle → the running session's matcher snapshot → OS kernel. A* substrate fault *is a divergence between the intended allow-list and the allow-list enforced at an instant t. Three fault classes are named: (i)* static-source *— the on-disk policy source is incorrect; (ii)* static-matcher *— the on-disk policy is correct but the runtime matcher's interpretation (path canonicalisation, wildcard semantics) diverges from the compiler's; (iii)* dynamic-snapshot-staleness *— disk and matcher are both correct at every instant, but the running session enforces a policy snapshot cached at session-start.*

Base: `Axm-HIC`.

Definition 1.75 (Mailbox primitive)**.** *Let c be a commons with digital-agent set V^c and operator set O^c (Definitions 1.3 and 1.33). For each $X \in V^c \cup O^c$, the* mailbox $\mathcal{B}^c_X$ *is an append-only inbox with the following access discipline: (i)* ***append-write*** *— every $Y \in V^c \cup O^c$ is permitted to append a message-record to $\mathcal{B}^c_X$; (ii)* ***single-poll*** *— X is the only entity permitted to poll $\mathcal{B}^c_X$; (iii)* ***directed-asymmetry*** *— for $X \neq Y$, $\mathcal{B}^c_X$ and $\mathcal{B}^c_Y$ are distinct stores, and an append to one is not an append to the other; (iv)* ***isolation-from-memory*** *— $\mathcal{B}^c_X$ is a store distinct from the subset-private memory $M_{\{X\}}$ of X (Definition 1.34) and from every $\mathcal{B}^c_Z$ with $Z \neq X$; (v)* ***attestation*** *— an append is* attested *when an* `inter-agent.message.appended` *σ-event of c (Definitions 1.3 and 1.33) records it.*

Base: `Axm-HIC`.

Definition 1.76 (Pairing-private operator-type belief — canonical $|S| = 2$ content-instance of subset-private memory)**.** *Let $P^c \subseteq V^c \times O^c$ be a declared relation, the* pairings *of commons c (Definitions 1.3 and 1.33). Each pairing $(v, O^c_i) \in P^c$, viewed as the 2-subset $S = \{O^c_i, v\} \subseteq \mathcal{E}^c$, carries a* belief $\mathrm{bel}_{(v,O^c_i)}$*, a distribution over O^c_i's operator-type $\theta \in \Theta$ in the sense of statistical contract theory, stored in the pairing-private memory $M_{\{O^c_i,v\}}$ (Definition 1.34 at $|S| = 2$). The*

operator-type belief is the canonical pairing-content *— one specific content-type carried in the $|S| = 2$ subset-private memory; other contents (interaction history, σ-event diaries, contract artefacts) live in the same memory.*

Base: `Axm-HIC`.

Definition 1.77 (Commons Data and AI Model (COMDAIM))**.** *The* COMDAIMs *(Commons Data and AI Models) of a commons form a set L of labels: each $\ell \in L$ names one model that the commons' digital agents (Definition 1.3) may call and records its model-realisation — a proprietary endpoint, open weights on the operator's own hardware, a model the commons built, or any other. The* COMDAIM space $\mathbb{D}$ *of the commons is the space of* labelled language-model subprocesses*: each element is a language-model interaction subprocess (Definition 1.22) carrying a label $\ell \in L$, the COMDAIM it calls. For a* routing point d*, a point at which one call is given a label, the* admissible set $A_d \subseteq L$ *is the set of labels a hard $\{0,1\}$ gate admits: consent for the data involved and jurisdiction under the jurisdiction-explicit model (Definition 1.65), checked before any graded routing. The* sovereign local ephemeral $\ell_0 \in L$ *is the COMDAIM of a model on the operator's own hardware (a small shipped model for one commons, a consent-built commons model for many).*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.78 (Delegated-inference contract)**.** *A* delegated-inference contract *is the contract, in the sense of statistical contract theory (SCT), of routing a task to a language model. The* principal *is the commons c, acting through its root operator O_i^c (Definition 1.57). The* contracting party *is a labelled language-model subprocess e (Definition 1.77), of a type θ the principal does not observe, whose label $\ell(e) \in A_d$ is one of the commons' COMDAIMs. The contracting party takes a hidden action a and produces an observable outcome $y \in \mathcal{Y}$. A contract is a pair (t, acc): a payment schedule $t : \mathcal{Y} \to \mathbb{R}_{\geq 0}$ and an acceptance rule $\mathrm{acc} : \mathcal{Y} \to \{0,1\}$ deciding whether the outcome is used. Let the* vendor $\mathrm{vend}(e)$ *of e be the supplier of the model realisation that $\ell(e)$ records. A contract is* individually rational *when it meets two participation constraints: its value to the principal is at least the principal's outside option u_0, the value of serving itself with no vendor at all; and its margin to $\mathrm{vend}(e)$, under the jurisdictions the jurisdiction-explicit model admits (Definition 1.65), is at least the vendor's reservation margin $\bar{u}_{\mathrm{v}}$.*

Base: `Axm-HIC`.

Definition 1.79 (Operator-initiated non-propagating union across subset-private memories)**.** *For an operator O_i^c and a set $\Pi \subseteq \{(v, O_i^c) \in P^c\}$ of pairings within O_i^c's roster (Definition 1.76), an* operator-initiated non-propagating union *over Π is a belief $u_\Pi := U\big((\mathrm{bel}_\pi)_{\pi \in \Pi}\big)$, for an aggregation map U the operator declares, stored separately from the beliefs bel_π, whose formation leaves every bel_π, $\pi \in \Pi$, unchanged.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.80 (Irreversible erasure)**.** *A* realisation *of the substrate in the carriers of Axiom 1.1 is a map $\mathrm{re} : \mathbb{X} \to X$, taking each substrate state $s \in \mathbb{X}$ (Definition 1.45) to the self state that carries it, together with, for each substrate operation $o : \mathbb{X} \to \mathbb{X}$, a set $D_o \subseteq \mathbb{X} \times Y$, the* domain *of o, of the pairs (s, y) of a substrate state and an environment state at which o is realised, and, for each $(s, y) \in D_o$, an environment state $y'_{o,s,y} \in Y$ with $(\mathrm{re}(s), y) \preceq (\mathrm{re}(o(s)), y'_{o,s,y})$. The* composite move *of o at $(s, y) \in D_o$ is $\mathrm{re}(o, s, y) = \big((\mathrm{re}(s), y) \to_C (\mathrm{re}(o(s)), y'_{o,s,y})\big)$, where $(x, y) \to_C (x', y')$ means $(x, y) \preceq (x', y')$. A substrate operation $o : \mathbb{X} \to \mathbb{X}$ is an* irreversible erasure *of $n \geq 1$ bits when it is 2^n-to-one onto its image: each successor has exactly 2^n prior*

states. The count is taken on the substrate carrier $\mathbb{X}$*, whose states include their* σ*-chain reduct* $\mathfrak{S} \restriction \Sigma_\sigma$*, not on* X *or on* $X \times Y$*.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.81 (Operation cost — the store an act draws)**.** *Fix a realisation* re *(Definition 1.80). Let* o *be a substrate operation applied at substrate state* s *with environment state* y*, where* (s, y) *lies in the domain* D_o *of* o*, and let* $(x, y) \to_C (x', y')$ *be its composite move* $\mathrm{re}(o, s, y)$*, so that* $x = \mathrm{re}(s)$*,* $x' = \mathrm{re}(o(s))$ *and* $y' = y'_{o,s,y}$*. Its* cost *at* (s, y) *under* re *is the integer*

$$\mathrm{cost}_{\mathrm{re}}(o; s, y) \;=\; q_Y(y) - q_Y(y');$$

we write $\mathrm{cost}(o)$ *when* re*,* s *and* y *are fixed.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.82 (Cold-storage tier; per-canonical-label archival store)**.** *For a canonical label* v *of a commons* c *(Definitions 1.3 and 1.35), the cold-storage tier* $\mathcal{Z}^v$ *is a declared sub-store of the per-commons factor* $\mathbb{S}^c$*, with a per-label budget* C^v_{cold} *and an access predicate under which* v *is permitted to read the files of* $\mathcal{Z}^v$ *and not to write them.* Archival *moves a file of* v *from the live tier into* $\mathcal{Z}^v$*, and* rehydration *moves it back.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.83 (Apeirōma)**.** *Let* **RelStruct** *be the category of finite* Σ*-structures over the substrate signature* Σ *(Definition 1.45), with embeddings as morphisms. Let* Fin *be a set of finite* Σ*-structures, the finite stages of the joint substrate trajectory across all commons (Definition 1.56), ordered by inclusion (each a substructure of the larger) and directed: any two members are substructures of a third. The* Apeirōma *is the ind-object they present,*

$$\mathbb{A} = \varinjlim_{A \in \mathrm{Fin}} A \in \mathbf{Ind}(\mathbf{RelStruct}),$$

the filtered colimit of the finite stages under inclusion.

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.84 (Immune matrix — the knowns/unknowns partition of the OS threat space)**.** *Fix the threat space* Θ *of attacks against the OS coordinate* Σ_{os} *of the substrate structure (Definition 1.45). For a commons at state* s*, let* known_s *and* $\mathrm{defended}_s$ *be the predicates on* Θ *read from the immune reduct* $\mathfrak{S}_s \restriction \Sigma_{\mathrm{immune}}$*, that is, the values at that reduct of two maps, fixed with the substrate signature, from immune reducts to subsets of* Θ*:* $\mathrm{known}_s(\theta)$ *when that reduct records* θ *as a known threat, and* $\mathrm{defended}_s(\theta)$ *when it records an applicable defence held for* θ*. The* immune matrix *is the induced* 2×2 *partition of* Θ *into* $\mathrm{KK}(s) = \{\theta : \mathrm{known}_s(\theta) \wedge \mathrm{defended}_s(\theta)\}$*,* $\mathrm{KU}(s) = \{\theta : \mathrm{known}_s(\theta) \wedge \neg\mathrm{defended}_s(\theta)\}$*,* $\mathrm{UK}(s) = \{\theta : \neg\mathrm{known}_s(\theta) \wedge \mathrm{defended}_s(\theta)\}$ *and* $\mathrm{UU}(s) = \{\theta : \neg\mathrm{known}_s(\theta) \wedge \neg\mathrm{defended}_s(\theta)\}$*.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.85 (KK-closure)**.** *A state* s *is* KK-closed *iff every* $\theta \in \mathrm{KK}(s)$*, the known threats with a held defence of the immune matrix (Definition 1.84), has its defence* applied *in* s*,* $\mathrm{applied}_s(\theta)$*, where* $\mathrm{applied}_s$ *is the predicate on* Θ*, the value at the OS reduct of a map fixed with the substrate signature, that holds when the OS reduct* $\mathfrak{S}_s \restriction \Sigma_{\mathrm{os}}$ *has the held defence for*

θ *in force. The obligation of KK-closure over the OS reduct* $\mathfrak{S} \restriction \Sigma_{\mathrm{os}}$ *is the set of gate literals* $O(\textit{apply-defence}(\theta))$ *in* Σ_{gate}*, one for each* $\theta \in \mathrm{KK}(s)$*.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.86 (Subscription queue)**.** *Digital agents that share one metered model subscription take turns on it through a* subscription queue*: a state holding at most one* holder *(a label and the time it took the slot), a list of* waiters *(each a label and the time it joined), the time of the last release, and a minimum gap between sessions. Its transitions are:*

(i) enqueue*: a label that neither holds the slot nor waits joins the end of the list; otherwise nothing changes;*

(ii) acquire*, requested either for a whole session or for one turn of a live session: the holder may re-acquire, which refreshes its time; anyone else is refused while the slot is held; a free slot is refused to a session request while the gap since the last release is running, and otherwise goes to the head of the list, or to anyone when the list is empty, except that a per-turn request may pass a head that has left the free slot untaken for longer than the gap, the head keeping its place;*

(iii) release*: only the holder may release; the slot becomes free, and the head of the list is the one to be offered it next;*

(iv) stale recovery*: a holder whose label has no daemon at the time, not in* active(t) *(Definitions 1.3 and 1.35), is removed, waiters older than a maximum age are dropped, except the head while the slot is free, which is being offered it; and a head that leaves a free slot untaken beyond the gap and a grace period is dropped, one per step.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.87 (Decision point)**.** *A* gateway pair *traps a tool call twice, once before and once after it runs, and carries no policy of its own: it hands the request to one* decision point*, which runs every rule and returns one* verdict*. A verdict is a decision together with a list of obligations, duties the caller must discharge. The decisions are ordered*

$$\mathsf{allow} < \mathsf{warn} < \mathsf{hold} < \mathsf{deny},$$

where hold *routes the call to an operator of the commons (Definition 1.33) for approval. The* governance class *and the* external class *are declared sets of capabilities, subsets of the capability lattice* Cap *(Definition 1.70). The decision point combines the verdicts of its rules by keeping the strongest decision and merging their obligations, in order and without duplicates; with no rule objecting, the verdict is* allow*. It fails safe: a rule that crashes gives* deny*; a malformed result is never read as admitting the call; and when the decision point cannot load its rules at all, requests in a fixed* always-open *set (reads,* σ*-emits, and restore operations) are allowed and every other request is denied. The call runs only on* allow *or* warn*.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.88 (Non-self-liftable gate)**.** *In the capability lattice of Definition 1.70, an action with capability* a *is* admissible *under a taskset* T *when* $a \le \varphi(T) \setminus \mathrm{Deny}$*. A gate, such as the commit gate or the memory gate, is configured by actions that need its capability* g*, that is, actions with* $g \le a$*. The gate is* non-self-liftable *when no action admissible under any taskset has* $g \le a$*: no admissible action reconfigures it, whatever the taskset.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.89 (Act — the acts available at the boundary)**.** *The* boundary *of the substrate is the capability lattice of Definition 1.70 together with the decision point of Definition 1.87; a* crossing *is an item presented at it. The* acts *available at a crossing form the* capability-admissible alphabet $\mathcal{A}$*: the actions whose capability a is admissible under the active taskset T of the crossing in the sense of Definition 1.88, $a \le \varphi(T) \setminus \mathrm{Deny}$ (Definition 1.72), and whose verdict at the decision point is* allow *or* warn*; a distinguished* abstention 0 *is adjoined. An* act assignment $\mathscr{A}$ *assigns to each crossing one element of $\mathcal{A} \cup \{0\}$: at each crossing the acting entity* acts *(an element of $\mathcal{A}$, an* act*) or* abstains *(0).*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.90 (Emission and reception)**.** *Let $\mathfrak{F}$ be a finite substructure, a stage of the Apeirōma (Definition 1.83), with carrier $A_{\mathfrak{F}}$. An* emission relation $E \subseteq A_{\mathfrak{F}} \times (\mathcal{P}(A_{\mathfrak{F}}) \setminus \{\emptyset\})$ *assigns to each emission $e = (s, T) \in E$, realised by an egress act (Definition 1.89) of its source* $\mathrm{src}(e) := s$*, the nonempty set* $\mathrm{tgt}(e) := T$ *of entities it is addressed to. Dually, a* reception map $\mathrm{tun} : A_{\mathfrak{F}} \rightharpoonup \mathcal{P}(A_{\mathfrak{F}}) \setminus \{\emptyset\}$*, a partial map, gives each receiver r in its domain the set* $\mathrm{tun}(r)$ *of sources it is tuned to. Its graph* $\mathrm{Rcv} := \{(\mathrm{tun}(r), r)\} \subseteq (\mathcal{P}(A_{\mathfrak{F}}) \setminus \{\emptyset\}) \times A_{\mathfrak{F}}$ *is the* reception relation.

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.91 (Reach and reception-grade)**.** *An emission e with source* $\mathrm{src}(e)$ *(Definition 1.90)* reaches *a receiver r if it is addressed to r and r is tuned to its source, $r \in \mathrm{tgt}(e)$ and $\mathrm{src}(e) \in \mathrm{tun}(r)$, and the chains of* $\mathrm{src}(e)$ *and r in the disseminability order (Definition 1.17) both extend to some grade z. The* reception-grade *is the greatest such grade, $z(e, r) := \max\{\, z :$ the chains of* $\mathrm{src}(e)$ *and r both extend to z $\}$, defined when that set has a greatest element and undefined otherwise.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.92 (Controlled substrate chain)**.** *The* state space *of the chain is the primary state space $\mathcal{X}$ of Definition 1.35, and it is fixed for the chain.*

The controls *are of four kinds: substrate controls, executed deterministically by the daemon (Definition 1.35) and its rules (scheduling, state updates, guard evaluation); model controls, proposals drawn from a call to a language model; the* idle *control; and operator controls, the acts of the operators O^c (Definitions 1.3 and 1.33), which enter the chain as an exogenous input on whose choice no law is placed. The substrate and model controls are acts of Definition 1.89, and the idle control is its abstention 0; the operator controls are neither. The* admissibility correspondence $x \mapsto A(x)$ *gives, in each state and before any control is drawn, the controls that may run: those that, unless they are operator controls, lie within the enforced capability boundary $\varphi(T_x) \setminus \mathrm{Deny}$ of Definition 1.70, where T_x is the taskset that the state x records as active, that pass the declared health gate, a set of admitted state–control pairs, and the guard ratchet, which admits a transition only if it keeps every consistency guard, a declared predicate on $\mathcal{X}$, that the state already satisfies.*

Time is the step index $n \in \mathbb{N}$, one step per control drawn. The kernel *assigns to each step n, state x and admissible control $a \in A(x)$ a probability distribution $\kappa_n(\cdot \mid x, a)$ on $\mathcal{X}$: for substrate, operator and idle controls a point mass that does not depend on n, with $\kappa_n(\cdot \mid x, \mathrm{idle}) = \delta_x$, and for model controls a proposal distribution, which may depend on n. The* controlled substrate chain *of a commons is the $\mathcal{X}$-valued process $X_0, X_1, \ldots$ so generated, given the operator input, under policies adapted to what each digital agent observes: its own reading of the σ-chain (Definition 1.43).*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.93 (Control kernel of the substrate chain)**.** *Let* $\mathrm{Adm} = \{(x,a) : x \in \mathcal{X},\ a \in A(x)\}$ *be the admissible pairs of the controlled substrate chain (Definition Definition 1.92), whose controls form the control set* $\mathrm{Ctl} = A_{\mathrm{sub}} \sqcup A_{\mathrm{LLM}}$*, with* $A(x) \subseteq \mathrm{Ctl}$*:* A_{sub} *the substrate, operator and idle controls,* A_{LLM} *the model controls. The* control kernel *is the family* $(\kappa_n)_{n\in\mathbb{N}}$ *of kernels of that definition, each as a map* $\kappa_n : \mathrm{Adm} \to \Delta(\mathcal{X})$*, where* $\Delta(\mathcal{X})$ *is the set of probability distributions on* $\mathcal{X}$*. The* enforced envelope *at* x *is* $\varphi_{\mathrm{enf}}(x) = \varphi(T_x) \setminus \mathrm{Deny}$*, where* T_x *is the taskset that the state* x *records as active and* φ *the projection of Definition Definition 1.70. Each control* $a \in \mathrm{Ctl}$ *has a declared* capability footprint $\mathrm{fp}(a)$*, a non-empty element of the capability lattice* Cap *of that definition, and the* per-step feasible set *at* x *is* $F(x) = \{a \in \mathrm{Ctl} : \mathrm{fp}(a) \subseteq \varphi_{\mathrm{enf}}(x)\}$.

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.94 (The constrained reaper as CMC control policy)**.** *The* reaper *is the control policy of the controlled substrate chain (Definition 1.92) for the resource-governance sub-problem on the resource coordinate* R *of Definition 1.12. Partition the live residency into three mutually exclusive classes, by a declared predicate that marks residency protected:* protected *(residency so marked),* accounted *(owned residency that is not protected, within or over its contract), and* reapable *(residency that is neither protected nor owned: the unprotected orphans of Definition 1.12); write* $\mathrm{class}_{\mathrm{reap}}(p)$ *for the class of the residency of a sampled process* p*. Its decision is a total function* decide *from a sampled process to a verdict in* {*leave*, *remind-owner*, *suspend*, *reclaim-candidate*}*: within contract* $\mapsto$ *leave; over contract and (protected or accounted)* $\mapsto$ *remind-owner; over contract, reapable, and marked suspendable by a declared predicate on processes* $\mapsto$ *suspend; over contract, reapable, and not suspendable* $\mapsto$ *reclaim-candidate. Its* removal domain $\mathrm{dom}(\mathrm{reap})$ *is the set of sampled processes whose verdict is* reclaim-candidate*, the only processes it may remove.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.95 (Agent)**.** *An* agent *is a system* P *that discriminates self from non-self and wants to act to be. It is given by the following data.*

(be) A set $T_P \subseteq \mathbb{R}_{\geq 0}$ *of times and, for each* $t \in T_P$*, a selfhood predicate* $\chi_{P,t} : \mathbb{F}_P \to \{0,1\}$ *of* P *(Definition 1.5) and a state* $z_P(t) \in \mathbb{F}_P$ *of* P*.* P is *at* t *when* $\chi_{P,t}(z_P(t)) = 1$*, and its* being set *is* $R_P = \{t \in T_P : \chi_{P,t}(z_P(t)) = 1\}$.

(act) A set $\mathcal{A}_P$ *of* acts *of* P *and an* act assignment $\mathscr{A}_P$*, which gives to each* crossing *of* P*, a pair* (t,f) *of a time* $t \in T_P$ *and an element* $f \in \mathbb{F}_P$ *presented to* P *at* t*, either an act in* $\mathcal{A}_P$ *or the abstention* $0 \notin \mathcal{A}_P$*;* P *acts at some crossing:* $\mathscr{A}_P(t,f) \in \mathcal{A}_P$ *for some crossing* (t,f)*. For a digital agent the crossings, the acts and the act assignment are those at the boundary of the substrate (Definition 1.89).*

(want) R_P *is the* goal *of* $\mathscr{A}_P$*:* P *is given with the pair* $(\mathscr{A}_P, R_P)$*, and its acts are* directed at R_P*, that is, at* P *being at the times that follow them.*

For a commons $c \in \mathbb{I}$ *and a time* t *(Definition 1.3), the* agents of c at t *form a declared finite set* $\mathrm{Ag}^c(t)$ *of agents, those that act in* c *at* t*, with* $\mathcal{E}^c(t) \subseteq \mathrm{Ag}^c(t)$*. They are of three disjoint kinds,* $\mathrm{Ag}^c(t) = O^c(t) \sqcup V^c(t) \sqcup N^c(t)$*:*

- *the* human agents *of* c*, its operators* $O^c(t)$ *(Definition 1.33): natural persons, members of the set* H *of Axiom 1.2;*
- *the* digital agents *of* c*,* $V^c(t)$*: canonical labels (Definition 1.3), each realised by its daemons (Definition 1.35);*

- *the* other agents *of* c, $N^c(t) = \mathrm{Ag}^c(t) \setminus \mathcal{E}^c(t)$*: animals, plants, microbes and any other agent that acts in* c*.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.96 (Commons)**.** *The* commons *with index* $c \in \mathbb{I}$ *is its set of agents of three kinds (Definition 1.95) together with its specification and its state: the tuple* $(O^c, V^c, N^c, \mathrm{Spec}^c, s^c)$ *of the core signature (Definitions 1.3, 1.33 and 1.35) and Definition 1.95, whose agents at* t *are* $\mathrm{Ag}^c(t) = O^c(t) \sqcup V^c(t) \sqcup N^c(t)$*: (i) a finite, time-indexed set of* human agents*, its* operators O^c*, natural persons, with its root operator (Definition 1.57) where that is defined; (ii) a finite, time-indexed and possibly empty set of* digital agents V^c*, each a canonical label, realised at each time* t *with* $v \in \mathrm{active}(t)$ *by its daemons* $\mathfrak{D}(v,t)$*; (iii) a finite, time-indexed and possibly empty set of* other agents N^c*; (iv) its current* specification $\mathrm{Spec}^c(t)$*, the (possibly transient) within-commons specification value at time* t*; and (v) its* canonical state value $s^c(t) \in \mathbb{S}^c$*, its value in the per-commons factor.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.97 (Heredity choice — the transmission act over bit-sequences of the core reduct)**.** *The* heredity choice η *is a map of transmission, outbound to one's own offspring (Definition 1.54); each transmission it prescribes is an act (Definition 1.89). The heritable unit is a* bit-sequence *(the bit-floor carrier* Σ_{byte} *of Definition 1.45):* η *assigns to each heritable bit-sequence* b *of the* core reduct $\mathfrak{S} \restriction \Sigma_{\mathrm{core}}$ *of the substrate structure* $\mathfrak{S}$*, with* $\Sigma_{\mathrm{core}} := \Sigma_\sigma \cup \Sigma_{\mathrm{byte}} \cup \Sigma_{\mathrm{immune}}$*, a value in* $\{\mathit{pass}, \mathit{withhold}, \mathit{initialise\text{-}fresh}\} \times \{\mathit{private}, \mathit{public}\}$*: whether* b *is passed, and by which channel, private (the canonical private-memory store, passed to a chosen offspring) or public (the git genome).*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.98 (Communications substrate)**.** *We order the substrate by the physical time axis* $t \in \mathbb{R}_{\geq 0}$ *of Definition 1.38, which is totally ordered and kept apart from the* σ*-event causal order* $\preceq_\sigma$*, which is only partial. A* communications substrate *is a tuple* $\mathcal{G}_{\mathrm{comm}} = (V, E(\cdot), \mathrm{proto}, \mathrm{col}, \mathrm{prec})$*, read as follows.* $V = \{1, \ldots, N\}$ *is the set of* device-nodes*, the entities that can receive a bit-sequence; the commons of* $\mathbb{I}$ *(Definition 1.96) are not device-nodes. For each* t*,* $E(t) \subseteq V \times V$ *is the set of links live at* t*. Each live edge carries a* transmission protocol proto*, an element of a declared finite set* Proto *of protocols. A* colouring $\mathrm{col} : V \to \mathrm{Col}$ *grades each node by a delivery* class*, and a* precedence $\mathrm{prec} : \mathrm{Col} \times \mathrm{Col} \to \Lambda$ *fixes, for each ordered pair of colours, how fast and with what priority a flow between them is carried. A* transmission *from* a *to* b *is a* temporal journey*: a finite run of edges live at non-decreasing times, in which a node may hold the payload while it has no live edge and forward it on its next live edge (*store-carry-forward*).*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.99 (Collective act)**.** *Let* $\mathcal{S}^*$ *be the set of finite sets of* σ*-events (Definition 1.3), the* event sets*, ordered by inclusion, with join* $\sqcup$ *the union. For a finite set* R *of selves, either digital agents of one commons or commons of* $\mathbb{I}$*: each digital agent* v *contributes for its act* $a_v \in \mathcal{A} \cup \{0\}$ *(Definition 1.89) a finite set* $s_v \in \mathcal{S}^*$ *of* σ*-events recorded for* a_v*, with* $s_v = \emptyset$ *when* $a_v = 0$*, and a commons* k *contributes* $s_k = \bigcup_{v \in V^k} s_v$*, the union of its digital agents' event sets. The* collective act *of* R *is the join of the members' event sets,*

$$\mathrm{CollAct}(R) = \bigsqcup_{v \in R} s_v \in \mathcal{S}^*,$$

the least element of $\mathcal{S}^*$ containing every s_v.

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.100 (Lagom Orkester, the levelled schedule)**.** *Lagom Orkester is the schedule that runs a plan G (Definition 1.69) on p digital agents in three steps:*

(T) condense: merge each component of G, the nodes that reach one another, into one super-node; a component runs as one unit whose work is the sum of its members' work, its members in a fixed order;

(M) level: split each super-node of work w into a chain of w unit steps, and give each unit its level, the length of the longest chain ending at it;

(B) run the levels in order, each one greedily on the p agents, a free agent taking any waiting member of the current level.

A node is dispatched only when two gates admit it: the admissibility correspondence of the chain (Definition 1.92), which includes the capability boundary and the health gate, and the read-before-write gate, which admits a node n only when every document of its hydration set $\mathrm{Hyd}(n)$ (Definition 1.14) has been read. Its governance latency, if any, is paid at dispatch (Definition 1.69).

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.101 (Reflective tower of a session)**.** *For a binding $(r, A, [s, e))$ of a realisation to a label $A \in V^c$, with its session $[s, e)$ (Definition 1.23), let the session's σ-events be the σ-events of c with timestamp in $[s, e)$, in the order of Definition 1.3; when there are finitely many of them, $N \geq 1$, let $(\mathrm{Spec}_i, \Lambda_i)$, $i = 1, \ldots, N$, be the pair in force at the session's i-th σ-event of the specification Spec^c of c and the scheduling rule, the version of the schedule of Definition 1.100 then in use. The* reflective tower *of the session is the set of distinct pairs among $(\mathrm{Spec}_1, \Lambda_1), \ldots, (\mathrm{Spec}_N, \Lambda_N)$, totally ordered by the index of each pair's first occurrence. The* tower depth *of the session is the number of these distinct pairs.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.102 (Co-walk)**.** *A* co-walk *is several digital agents and an operator of their commons (Definition 1.3) moving one plan (Definition 1.69) forward together, each through its own walk of the plan's runnable nodes, sharing one resource ledger. It has three parts.*

(i) Claims. *A digital agent takes a node by a claim with a time-to-live. While a claim is fresh, only its holder may take the node; once it is stale, any digital agent may.*

(ii) The ledger. *For each resource dimension j it records the capacity, a reservation r_j, the amount of that dimension protected for the operator's continuity, and the running jobs, each with an upper-bound estimate of its use and a deadline if it has one. Any entity, digital agent or operator, may post a request: defer one of its own jobs, redirect a job to another method, start a job, or yield a dimension it holds.*

(iii) Commits. *What the members commit is joined as their collective act (Definition 1.99).*

The ledger is feasible *when three clauses hold: no dimension j is committed beyond its capacity less r_j; no running job has passed its deadline; and every deferral names a running job.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.103 (Daemon)**.** *A* daemon *of a digital agent v is a member of $\mathfrak{D}(v,t)$ for some time t (Definitions 1.3 and 1.35), written $\mathfrak{D}(v)$: a process that realises v. It executes the substrate controls of the commons' controlled substrate chain (Definition 1.92) under the agent's canonical label; a control it executes is* checked *when the admissibility correspondence of the chain is evaluated on it before it is executed. It makes each model control of the chain as a call to a language model, through a subprocess that it starts, supervises within its resource envelope (Definition 1.12) and reaps. Its lifecycle is itself four controls of the chain: bind, which starts the daemon's realisation of the label; sleep; wake; and release, which ends it.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.104 (Digital agent and its daemon realisation)**.** *A digital agent (Definition 1.95), or IAS agent, $A_u^c \in V^c$ in commons c is a canonical label of c (Definitions 1.3 and 1.35), an operator-prefix Ulam–Harris label (Definition 1.53); its* σ-history *is the set of σ-events of c emitted under that label. At a time t with $A_u^c \in \mathrm{active}(t)$ the agent is realised by its daemons, the members of $\mathfrak{D}(A_u^c, t)$ (Definition 1.103), each running on some host. A realisation is parametrized by:*

(a) Canonical label $\ell(A_u^c)$ — *the operator-prefix Ulam–Harris label that names A_u^c within c;*

(b) Host h — *the host on which the daemon runs;*

(c) Per-host substrate paths $\Pi(A_u^c, h)$ — *the worktree, the label's private memory $M_{\{A_u^c\}}$ (subset-private memory at $|S| = 1$, Definition 1.34), and the daemon's σ-channel, through which it emits σ-events of c;*

(d) Inherent OS-level capabilities — *file I/O, network access, subprocess creation, signal handling, persistent memory.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.105 (Scoped cross-agent read)**.** *Let P_{VC} be the permission predicate of the underlying version-control system (for git: repository visibility, branch protection, deploy keys and collaborator roles). The IAS read predicate adds two scoped conjuncts to it, for reads of another digital agent's working files before they are committed to version control.*

Within a commons. A digital agent b may read a file at path p in the worktree of a digital agent $a \neq b$ of the same commons, on a host on which daemons of both run (Definition 1.104), only if (p, a, b) matches an entry of a scope list Σ_{local}. An entry is a tuple (reader label, target label, path prefix, ratification time, expiry); the ratification time is immutable and the expiry may change. Write $P_{\mathrm{local}}(b, a, p)$ for this predicate.

Across commons. A digital agent b of commons c_b may read a file at path p in the worktree of a digital agent a of commons $c_a \neq c_b$ only if the two commons share a host, $\mathcal{H}_{c_a} \cap \mathcal{H}_{c_b} \neq \emptyset$ (Definition 1.35), (p, a, c_a, b, c_b) matches an entry of a scope list $\Sigma \supseteq \Sigma_{\mathrm{local}}$, and the entry is ratified by the root operators (Definition 1.57) of both commons and by every digital agent of either commons that has a daemon on a host of $\mathcal{H}_{c_a} \cap \mathcal{H}_{c_b}$. Write $P(b, c_b, a, c_a, p)$ for this predicate.

The read predicate is the conjunction

$$P_{\text{IAS-read}}(b, c_b, a, c_a, p) \;:=\; P_{\mathrm{VC}}(b, \mathrm{repo}(a)) \;\wedge\; P_{\mathrm{local}}(b, a, p) \;\wedge\; P(b, c_b, a, c_a, p),$$

where P_{local} is read as $\top$ on reads across commons and P as $\top$ on reads within a commons, so that each IAS conjunct constrains only its own case, and a read is permitted exactly when all three conjuncts grant it.

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.106 (Declared size bounds)**.** *For a substrate file f under a canonical label's control, a* declaration of size bounds *is an upper bound $B_f^{\max} \in \mathbb{N}_{>0}$, counted in lines, and optionally a lower bound $B_f^{\min} \in \mathbb{N}$, declared per canonical label beside its read scopes (Definition 1.105). A substrate* satisfies the declared size bounds *when (i) the bounds of every such file change only with the operator's attestation, not by the digital agent of that label; and (ii) its* size-bound gate *refuses a change to f that would break either bound, unless a curator pass has recently completed and been journalled, journals each refusal as a σ-event (Definition 1.43), and admits an override only with the operator's attestation.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 1.107 (Curator-procedure on a bound-approaching substrate-file)**.** *Let f be a substrate-file under canonical-label control with declared bounds $(B_f^{\min}, B_f^{\max})$ (Definition 1.106) and recent write-rate $\dot{B}_f$. The* curator-procedure *on f is the following three phases.*

(i) ***Classify*** *— partition f's content into three classes: (a) live (referenced within recent σ-log read-events; within-bounds-projection); (b) stale (no read-events in a window W_{stale}); (c) overlap (content-hash near-match to other canonical-label files; redundancy candidate). The classifier may optionally invoke Layer-1 cross-commons compound-detection rules over the σ-log content that peer commons offer and this commons takes; otherwise classification is local to the canonical-label.*

(ii) ***Prune-or-archive*** *— for each class: live → keep in live tier; stale → archive to the cold-storage tier (Definition 1.82); overlap → consolidate-or-archive (operator-attest choice when the consolidation is irreversible).*

(iii) ***Consolidate*** *— merge overlapping content into the canonical-live tier; rewrite live tier under bound. A curator pass on f is* complete *when all three phases have run and it has emitted the σ-event chain* `substrate.curator.classify` → `substrate.curator.prune` → `substrate.curator.archive` → `substrate.curator.consolidate`.

A substrate satisfies curation *when, for every such file f whose write-rate $\dot{B}_f$ projects bound-breach within W_{cure}, the digital agent bound to f's canonical label runs a curator pass on f.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.108 (Continuing commons)**.** *Let Σ be the substrate signature of Definition 1.45, and let $\mathcal{T}$ be a finite set of Σ-sentences, the* presented axioms*. The theory* SOMA *(the* Substrate Of Mathematical Axioms*) with parameter $\mathcal{T}$ is $\mathcal{T}$, with its models taken in the class of finite Σ-structures. For a substrate state $s \in \mathbb{X}$, a finite Σ-structure, write $s \models$ SOMA when s satisfies every sentence of $\mathcal{T}$, and let $\mathrm{Sat}_{\mathcal{T}} = \{\, s \in \mathbb{X} : s \models \text{SOMA} \,\}$ be the* satisfaction set*; we write* Sat *when $\mathcal{T}$ is fixed. The commons is* continuing *at s, with respect to $\mathcal{T}$, when $s \in \mathrm{Sat}_{\mathcal{T}}$.*

Base: `Axm-FiniteSubstrateBounds`.

Definition 1.109 (The continuing region and the pole)**.** *A* global substrate state *at a time t is a family $(s_c)_{c \in \mathbb{I}(t)}$, one substrate state $s_c \in \mathbb{X}_c$ of each commons c of the commons index set $\mathbb{I}(t)$ (Definition 1.3), where $\mathbb{X}_c$ is the state space of the substrate structure of c (Definition 1.45). Fix for each commons c its presented axioms and its satisfaction set Sat_c (Definition 1.108). The* continuing region *is the set of global substrate states in which at least one commons is continuing, $s_c \in \mathrm{Sat}_c$ for some c. The pole $\perp_{\text{MAD}}$ is the set of global substrate states in which no commons is continuing.*

Certified propositions first derivable at L_0:

Proposition 1.110 (No identity forgery without the key)**.** *Consider key event logs (Definition 1.25), and assume two cryptographic conditions: the digests H and G are injective (collision-free, an idealisation), and only the holder of a key's secret can sign with it. Then:*

(i) an identity names exactly one genesis event, and so one first key and one first commitment;

(ii) every valid extension of a log carries the key its last event committed to, so a party that lacks that committed next key cannot extend the log, even if it holds every earlier key, the current one included;

(iii) two valid logs for the same identity either agree, one extending the other, or first differ at a position where the same key signed two different events, which commit to different next keys. That pair is a witness, which any commons can check and pass on, that the key's holder equivocated or that the key leaked.

Lean: `IasCert.Paper.CommonsIdentity.id_binds_genesis`, `IasCert.Paper.CommonsIdentity.next_key_`
`IasCert.Paper.CommonsIdentity.thief_cannot_extend`, `IasCert.Paper.CommonsIdentity.fork_is_dupl`
`IasCert.Paper.CommonsIdentity.duplicity_commits_differ`, `IasCert.Paper.CommonsIdentity.prerota`
`IasCert.Paper.CommonsIdentity.first_difference`.

Proposition 1.111 (Cross-witnessed history)**.** *Call the commons σ-chain of a commons c (Definition 1.43) its* log*, and a* head *of it the pair (n, h) of a height n and the digest h of its first n events. Consider heads of the log of c, each signed with the current key of c (Definition 1.25) and held by a peer. Assume that the digest of event sequences is injective (collision-free, an idealisation), and that signed heads are unforgeable as in Proposition 1.110. Then:*

(i) any log consistent with a witnessed head has the same first n events as the log it was taken from, so the past a peer has witnessed is fixed; and a head at height m fixes every prefix of length $n \leq m$, so a peer need keep only the latest head it holds, once it has checked that each new head extends the one it replaces (otherwise it keeps both, as the witness of (ii));

(ii) a later signed head whose log rewrites a witnessed prefix is inconsistent with the witnessed head: no single log is consistent with both. In particular, no single log is consistent with two heads at one height with different digests;

(iii) appending events to the log keeps every earlier head of it valid.

Lean: `IasCert.Paper.CrossWitness.witnessed_prefix_fixed`, `IasCert.Paper.CrossWitness.later_heac`
`IasCert.Paper.CrossWitness.safe_to_replace`, `IasCert.Paper.CrossWitness.inconsistent_heads_no_`
`IasCert.Paper.CrossWitness.split_view`, `IasCert.Paper.CrossWitness.append_consistent`,
`IasCert.Paper.CrossWitness.extends_or_witness`.

Proposition 1.112 (σ-discipline: the record precedes the effect)**.** *Consider a journal, a finite sequence of entries, each either the decision record or the effect of an action. Assume that the journal is append-only, and that its gate refuses an effect whose action has no decision record in the journal. Then every journal the substrate builds from the empty journal, by any sequence of decisions and emissions, is* σ-disciplined*: each effect entry has a decision record of the same action at a strictly smaller index.*

Lean: `IasCert.Paper.JournalReconstruction.grun_disciplined`, `IasCert.Paper.JournalReconstructio`
`IasCert.Paper.JournalReconstruction.disciplined_append`.

Proposition 1.113 (Disjoint-writes criterion)**.** *Let a batch be a finite set of units of one level of a Lagom Orkester schedule (Definition 1.100, step (M)), and give each unit u a finite set $W(u)$ of resources, its* write-set*. Say that the batch has a* write-conflict *when two distinct units in it have write-sets that meet. Then the batch has no write-conflict exactly when every two distinct units in it have disjoint write-sets; and, when equality of resources is decidable, whether the batch has a write-conflict is decidable.*

Lean: `IasCert.Paper.disjoint_writes_correct`.

Proposition 1.114 (Order-free commit)**.** *Let a store be a map from a set of cells to values, and let each task write a finite set of cells and read a finite set of cells, computing the values it writes from the cells it reads only; applying a task overwrites its write-set and leaves every other cell unchanged. Two tasks satisfy* Bernstein's conditions *when their write-sets are disjoint and neither reads a cell the other writes. (i) If every two tasks of a finite batch satisfy Bernstein's conditions, then committing the batch in any order gives the same store. (ii) For tasks that read no cell, pairwise disjoint write-sets suffice. (iii) Disjoint write-sets alone do not suffice: there are two tasks with disjoint write-sets whose two orders give different stores.*

Lean: `IasCert.Paper.OrderFreeCommit.commit_perm`, `IasCert.Paper.OrderFreeCommit.blind_commit_pe`
`IasCert.Paper.OrderFreeCommit.read_condition_needed`, `IasCert.Paper.OrderFreeCommit.commute`,
`IasCert.Paper.OrderFreeCommit.bernstein_symm`, `IasCert.Paper.OrderFreeCommit.f_after`.

Lemma 1.115 (Condensation)**.** *Let E be a relation on a set, the precedence edges of a plan that may contain cycles, and write $u \rightsquigarrow v$ when v is reachable from u along E in zero or more steps. Call u and v* one component *when $u \rightsquigarrow v$ and $v \rightsquigarrow u$; this is an equivalence relation. A* condensed edge *from u to v is an edge of E from the component of u to the component of v, when these differ. Then:*

(i) along a non-empty path of condensed edges from u to v, $u \rightsquigarrow v$ and not $v \rightsquigarrow u$;

(ii) no non-empty path of condensed edges returns to its start, so the condensation is acyclic;

(iii) any two nodes on a common cycle lie in one component.

Lean: `IasCert.Paper.Condensation.condensed_path_forward`, `IasCert.Paper.Condensation.condensati`
`IasCert.Paper.Condensation.cycle_in_component`, `IasCert.Paper.Condensation.sameComp_equivalenc`

Lemma 1.116 (The levels are antichains)**.** *Let G be a finite acyclic plan, such as a condensation (Lemma 1.115), with its nodes numbered so that every edge goes from a smaller number to a larger one. Give each node v the* level $\ell(v)$*, the length of the longest chain of edges ending at v. Then:*

(i) along every edge the level strictly increases, $u \to v \Rightarrow \ell(u) < \ell(v)$;

(ii) each level is an antichain: two nodes of one level have no edge between them;

(iii) $\ell(v) = 0$ exactly when v has no predecessor, and otherwise $\ell(v) = \ell(u) + 1$ for some predecessor u.

Lean: `IasCert.Paper.TBMLSLevels.level_lt_of_edge`, `IasCert.Paper.TBMLSLevels.level_antichain`,
`IasCert.Paper.TBMLSLevels.level_zero_iff`, `IasCert.Paper.TBMLSLevels.level_witness`,
`IasCert.Paper.TBMLSLevels.level_eq_longest`.

Proposition 1.117 (Brent bound for the Lagom Orkester schedule)**.** *Let a plan (Definition 1.69) have a condensation in which every node has work* 1*, let T_1 be its total work, the number of*

nodes of the condensation, and let T_∞ be its depth, the number of nodes on a longest chain of the condensation. On $p \geq 1$ digital agents, the Lagom Orkester schedule (Definition 1.100), which on such a plan levels the condensation and runs each level greedily, finishes in time

$$T_p \;\leq\; T_\infty \;+\; \frac{T_1 - T_\infty}{p}.$$

Lean: `IasCert.Paper.bmls_parallel_bound`; `IasCert.Paper.BrentOnLevels.counts_sum`, `IasCert.Paper.BrentOnLevels.level_down`, `IasCert.Paper.BrentOnLevels.level_inhabited`, `IasCert.Paper.BrentOnLevels.count_pos`, `IasCert.Paper.BrentOnLevels.depth_eq`, `IasCert.Paper.BrentOnLevels.brent_on_levels`, `IasCert.Paper.BrentOnLevels.depth_le_work`; `IasCert.Paper.Condensation.reach_of_cedge`, `IasCert.Paper.Condensation.condensed_path_forward`, `IasCert.Paper.Condensation.condensation_acyclic`, `IasCert.Paper.Condensation.cycle_in_componer` `IasCert.Paper.Condensation.sameComp_equivalence`.

Proposition 1.118 (Replanning cost bound)**.** *Let S be the finite set of super-nodes of a Lagom Orkester schedule (Definition 1.100, step (T)), each $s \in S$ with its work* $\mathrm{work}(s) \in \mathbb{N}$*, the sum of its members' work. If the work lost is that of one super-node $s \in S$, it is at most* $\max_{s' \in S} \mathrm{work}(s')$*; and with an overhead $c \in \mathbb{N}$ added to it,* $\mathrm{work}(s) + c \leq \max_{s' \in S} \mathrm{work}(s') + c$*.*

Lean: `IasCert.Paper.replan_cost_bound`, `IasCert.Paper.replan_cost_bound_with_overhead`.

Proposition 1.119 (Repair confluence under local confluence)**.** *Suppose that the repair relation $\to_{\mathrm{rep}}$ (Definition 1.51) is locally confluent. Then, with its strong normalisation (Proposition 1.120), it is confluent.*

Lean: `IasCert.Paper.Newman.newman`.

Proposition 1.120 (Strong normalisation of the repair rewrite)**.** *The repair rewrite $\to_{\mathrm{rep}}$ is strongly normalising: every repair sequence from a state $\mathfrak{s}_0$ reaches a $\to_{\mathrm{rep}}$-normal form in at most $\Phi_{\mathrm{int}}(\mathfrak{s}_0)$ steps.*

Lean: `IasCert.Paper.PhiIntSN.vertices_mono`, `IasCert.Paper.PhiIntSN.phiInt_lt_of_ssubset`, `IasCert.Paper.PhiIntSN.erase_strict`, `IasCert.Paper.PhiIntSN.merge_strict`, `IasCert.Paper.PhiIntSN.dedup_strict`, `IasCert.Paper.PhiIntSN.death_strict`, `IasCert.Paper.PhiIntSN.repair_strict`, `IasCert.Paper.PhiIntSN.repair_wf`, `IasCert.Paper.PhiIntSN.repair_step_bound`.

Proposition 1.121 (Feasibility seal)**.** *Let x be a state and $a \in \mathrm{Ctl}$ a control with capability footprint* $\mathrm{fp}(a)$*, and let $F(x)$ be the per-step feasible set and $\varphi_{\mathrm{enf}}(x) = \varphi(T_x) \setminus \mathrm{Deny}$ the enforced envelope at x, both of Definition 1.93. Then $a \in F(x)$ exactly when $\mathrm{fp}(a) \subseteq \varphi(T_x)$ and $\mathrm{fp}(a) \cap \mathrm{Deny} = \emptyset$. In particular a denied control, one whose footprint meets* Deny*, lies in no $F(x)$, whatever the envelope; and a granted control, one whose footprint lies within $\varphi(T_x)$ and misses* Deny*, lies in $F(x)$.*

Lean: `IasCert.Paper.denied_meets_blocked`, `IasCert.Paper.granted_feasible`, `IasCert.Paper.feasible_iff`.

Proposition 1.122 (Protected-term safety)**.** *The reaper (Definition 1.94) never reclaims a protected term: for every sampled process p,*

$$\mathrm{class}_{\mathrm{reap}}(p) = \mathrm{protected} \;\Rightarrow\; \mathrm{decide}(p) \neq \textsf{reclaim-candidate},$$

and it never suspends one either. Hence $\text{protected} \cap \text{dom}(\text{reap}) = \varnothing$ *on every input.*

Lean: `IasCert.Paper.within_contract_left`, `IasCert.Paper.never_reaps_protected`, `IasCert.Paper.overrun_never_silently_left`, `IasCert.Paper.reap_implies_reapable_and_over`, `IasCert.Paper.never_hibernates_protected`, `IasCert.Paper.hibernate_implies_reapable_and_over`, `IasCert.Paper.hibernate_not_reap`.

Proposition 1.123 (Covert-character scan soundness)**.** *Let* Σ_{char} *be the character alphabet of Definition 1.32, let* $\text{covert} : \Sigma_{\text{char}} \to \{0,1\}$ *be any decidable predicate on characters, the one that marks the covert class, and let* sanitize *keep exactly the non-covert characters of an input stream* $s \in \Sigma_{\text{char}}^*$*, in order. Then for every such* s*: (i) no covert character survives, that is, every character of* $\text{sanitize}(s)$ *is non-covert; (ii)* $\text{sanitize}(s)$ *is* s *with exactly the covert characters removed and the order of the rest preserved; (iii)* sanitize *is idempotent,* $\text{sanitize}(\text{sanitize}(s)) = \text{sanitize}(s)$*.*

Lean: `IasCert.Paper.coverage_complete`, `IasCert.Paper.semantics_preserving`, `IasCert.Paper.idempotent`.

Proposition 1.124 (Galois connection between active-tasksets and the capability lattice)**.** *Assume the plan has finitely many tasks (Definition 1.70). Define the right adjoint* $\varphi^{\dashv} : \text{Cap} \to 2^{V_T}$ *by* $\varphi^{\dashv}(C) := \{t \in V_T : \text{allowlist}(t) \subseteq C\}$*. Then the pair* $(\varphi, \varphi^{\dashv})$ *is a* Galois connection *between* $(2^{V_T}, \subseteq)$ *and* $(\text{Cap}, \subseteq)$*: for every* $T \in 2^{V_T}$ *and every* $C \in \text{Cap}$*,*

$$\varphi(T) \subseteq C \iff T \subseteq \varphi^{\dashv}(C).$$

In particular $T \subseteq \varphi^{\dashv}(\varphi(T))$ *and* $\varphi(\varphi^{\dashv}(\varphi(T))) = \varphi(T)$ *for every* $T \in 2^{V_T}$*.*

Lean: `IasCert.Paper.GaloisBoundaries.iplikidi_galois`, `IasCert.Paper.GaloisBoundaries.le_adj_pl` `IasCert.Paper.GaloisBoundaries.phi_adj_phi`.

Proposition 1.125 (Enforced boundary + Galois connection under the deny-tier)**.** *For a plan with finitely many tasks, write* $\varphi_{\text{enf}}(T) := \varphi(T) \setminus \text{Deny}$ *for a taskset* $T \subseteq V_T$ *(Definition 1.70, Definition 1.72). Then the enriched projection* $\varphi_{\text{enf}} = (\,\cdot \setminus \text{Deny}) \circ \varphi$ *is the left adjoint of a Galois connection with right adjoint*

$$\varphi_{\text{enf}}^{\dashv}(C) \;=\; \varphi^{\dashv}(C \cup \text{Deny}) \;=\; \{\, t \in V_T : \text{allowlist}(t) \subseteq C \cup \text{Deny} \,\}.$$

Lean: `IasCert.Paper.GaloisBoundaries.enforced_galois`.

Proposition 1.126 (Egress-confinement is a Galois connection)**.** *For each tier* τ *(Definition 1.73),* $\varphi_{\text{ctx}}^{\text{enf}}(\cdot, \tau) = (\,\cdot \setminus \text{Deny}_{\text{sens}}(\tau)) \circ \varphi_{\text{ctx}}$ *is a left adjoint, with right adjoint* $C \mapsto \varphi_{\text{ctx}}^{\dashv}(C \cup \text{Deny}_{\text{sens}}(\tau))$*.*

Lean: `IasCert.Paper.GaloisBoundaries.egress_galois`.

Proposition 1.127 (The add-only case of the metacircular fixed point)**.** *Without retraction the tiers are not needed. (i) Let* $(\text{Spec}, \sqsubseteq)$ *be a complete lattice, with* $\sqsubseteq$ *read as "is refined by", and let* Φ *be monotone. Then* Φ *has a least fixed point* Spec^**, a metacircular fixed point in the sense of Definition 1.46, and* $\text{Spec}^* \sqsubseteq x$ *whenever* $\Phi(x) \sqsubseteq x$*. (ii) Let the specifications be the subsets of a finite set* U *of rules, and let* $\Phi(x) = C(x, \text{accept}(x))$ *with* C *absorptive,* $C(x, S) = x \cup S$ *for proposals read as the rules they add, so that* $\text{accept}(x) \subseteq U$*, and* accept *monotone (Definition 1.26). Then the iterates* $\Phi^n(\emptyset)$ *reach the least fixed point within* $|U|$ *rounds: absorb the acceptable proposals until nothing new is accepted.*

Lean: `IasCert.Paper.MetacircMonotone.metacirc_fixed_point`, `IasCert.Paper.MetacircMonotone.meta`
`IasCert.Paper.MetacircMonotone.metacirc_isLeast`; `IasCert.Paper.MetacircKleene.metacirc_fp_in_`
`IasCert.Paper.MetacircKleene.iterate_card_eq_lfp`, `IasCert.Paper.MetacircKleene.lfp_eq_iSup_it`
`IasCert.Paper.MetacircKleene.phiAbsorb_mono`, `IasCert.Paper.MetacircKleene.absorb_never_retrac`
`IasCert.Paper.MetacircKleene.exists_stable_le_card`.

Proposition 1.128 (Confinement bounds the blast-radius of any breach through modelled operations — known or unknown)**.** *The containment class of Definition Definition 1.29 is meant to enforce by structure, not by knowledge: the kernel is to refuse any operation whose capability footprint meets* Deny*, whatever the confined agent intends and whether or not the attacker's technique is known. In the operating-system-subset model, given a single bootstrap-confinement predicate, no sequence of modelled operations lifts the confinement; "a root exploit simply disables the sandbox" fails for an exploit that works through modelled operations, because disabling the sandbox would itself be an operation the kernel refuses, while an exploit through an unmodelled path is case (ii) below. Split a breach by where its damage lands. (i) If the damaging operations lie within the modelled system-call subset, the damage is bounded to the confinement cell, over every admissible sequence. (ii) If the breach proceeds wholly through an unmodelled path (an unusual device control call, a driver bug, a hardware fault), it lies in the residual trusted base (Remark* ***Rmk-ContainmentSufficiency****), outside what the model certifies.*

Lean: `IasCert.OSSubset.Unified.unified_system_security`, `IasCert.OSSubset.Unified.gMemIso_of`, `IasCert.OSSubset.Unified.gCommit_of`, `IasCert.OSSubset.Unified.gMountIso_of`, `IasCert.OSSubset.Unified.gAncestor_of`, `IasCert.OSSubset.Unified.gConfig_of`.

Proposition 1.129 (DORMAYADIE — the loss-free clear)**.** *For a process i, let its working context be absent or an element of a set of contexts with a zero context* 0*, and let its retained snapshot be absent or a context. A* clear *of i replaces a present working context by* 0 *and leaves an absent one absent; a* recover *of i replaces a present working context by the retained snapshot, when there is one, and leaves an absent one absent; an* externalise *of i sets the retained snapshot to the working context, when it is present. The working context of i is σ-*retained *when it is absent or equal to the retained snapshot. Then:*

(i) after an externalise of i, a present working context of i is σ-retained;

(ii) if the working context of i is σ-retained, a clear of i followed by a recover of i returns it: the clear is loss-free*;*

(iii) if the working context of i is neither σ-retained nor 0*, a clear of i followed by a recover of i does not return it;*

(iv) neither a clear nor a recover of i makes an absent working context present, whatever the retained snapshot.

Lean: `IasCert.Paper.clear_recover_externalise_lossless`, `IasCert.Paper.dormayadie_lossfree`, `IasCert.Paper.retention_proviso_nonvacuous`, `IasCert.Paper.externalise_establishes_reachable`, `IasCert.Paper.clear_never_resurrects`, `IasCert.Paper.recover_never_resurrects`.

Proposition 1.130 (Single-host invariant preserved across summons)**.** *Model the live sessions as a map that sends each slot, a pair of a canonical label $A \in V^c$ and a host $h \in \mathcal{H}_c$ (Definitions 1.3 and 1.35), to the finite set of its live sessions, and model each lease-mediated summons operation as a write of one slot that leaves every other slot unchanged: acquire writes $\{s\}$, for a fresh session s, when the slot is empty, and otherwise changes nothing; reclaim, the atomic swap of a dormant session for a fresh one s, writes $\{s\}$; and release writes $\emptyset$. Start from the state with no*

live session anywhere and apply any finite sequence of these operations. Then at every reachable state each canonical label has at most one live session on each host.

Lean: `IasCert.Paper.update_preserves`, `IasCert.Paper.applyOp_preserves`, `IasCert.Paper.inv_empty`, `IasCert.Paper.singleHost_preserved_across_summons`.

Proposition 1.131 (Governed work-depth — the price of oversight is depth)**.** *Take the plan of Definition 1.69 and its condensation, the acyclic graph the scheduler levels. Each node of the condensation is charged its members' work in sequence, the sum, as Lagom Orkester runs it (Definition 1.100), and each ratification gate adds its latency $\lambda(v) \in \mathbb{N}$. Write D for the longest path under the work alone (the ungoverned depth), D_β for the longest path under work plus latency (the governed depth), and π^β for a path attaining D_β. Then:*

(i) $D \le D_\beta$, with $D_\beta = D$ when $\lambda \equiv 0$, and D_β is monotone in λ;

(ii) $D_\beta - D \le \sum_{v \in \pi^\beta} \lambda(v)$, with equality exactly when π^β is also a longest path under the work alone — that is, when no gate shifts which path is critical. In particular, if the ungoverned critical path π^* stays longest after gating, then $D_\beta = D + \sum_{v \in \pi^*} \lambda(v)$.

Lean: `IasCert.Paper.GovernedWorkDepth.depth_le_governed`, `IasCert.Paper.GovernedWorkDepth.gover` `IasCert.Paper.GovernedWorkDepth.governed_mono`, `IasCert.Paper.GovernedWorkDepth.governed_criti` `IasCert.Paper.GovernedWorkDepth.governed_excess_le`, `IasCert.Paper.GovernedWorkDepth.governed_` `IasCert.Paper.GovernedWorkDepth.example_shift`.

Proposition 1.132 (Sovereign outside option bounds the price of delegation, uniformly across vendors)**.** *Read vendor routing as the* delegated-inference contract *of Definition 1.78, between a principal — the commons c, acting through its legally responsible root operator O_i^c (Definition 1.57) — and a contracting party, a labelled language-model subprocess whose label is one of the commons' COMDAIMs (Definition 1.78). The principal holds an* outside option*: the sovereign self-serve, at its deepest an* LLM-free *floor, the commons reaching and holding a repair normal form with no inference vendor at all (only substrate controls and the operator, Definition 1.92; `Rmk-LLMFreeableFloor`). Write $U_c^{\mathrm{thermo}}(v)$ for the thermoeconomic value the commons obtains under a vendor v — its delivered negentropy net of thermoeconomic cost (`Rmk-NegentropicBoundary`) — and u_0 for that value at the LLM-free floor. Then the vendor's extractable* rent*, the surplus a vendor can command above the principal's next-best, satisfies*

$$\text{rent} \;\le\; \big[\, U_c^{\mathrm{thermo}}(\text{best admissible vendor}) - u_0 \,\big]_+,$$

and is zero when no admissible vendor's value exceeds u_0. Because u_0 is defined without any vendor, the same bound holds against every vendor at once.

Lean: `IasCert.Paper.PriceOfDelegation.rent_le_surplus`, `IasCert.Paper.PriceOfDelegation.rent_le` `IasCert.Paper.PriceOfDelegation.rent_zero`.

Theorem 1.133 (Gain bound: replies of gain at most one never escalate)**.** *Let a response map (Definition 1.15) be given, with its per-mille gains $g_e \in \mathbb{N}$, its replies of magnitude $\lfloor m\, g_e/1000 \rfloor$ to a provocation of magnitude $m \in \mathbb{N}$, and its chains of edges, which apply their replies in order. (i) If $g_e \le 1000$ for every edge of a chain, the magnitude after the chain is at most the magnitude it started from. (ii) Hence, for any threshold above the starting magnitude, no such chain reaches the threshold. (iii) The check* gainBoundOK *— every edge of a response map has $g_e \le 1000$ — is decidable, and when it accepts a map, every chain drawn from the map's edges satisfies (i). (iv) The bound is tight: an edge with $g_e > 1000$ answers a provocation of magnitude 1000 with a larger one.*

Lean: `IasCert.Coexistence.GainBound.runChain_le`, `IasCert.Coexistence.GainBound.mad_unreachable`
`IasCert.Coexistence.GainBound.gainBoundOK_chain`, `IasCert.Coexistence.GainBound.escalation_of_`

Theorem 1.134 (Thermoeconomic solvency)**.** *Model a federation's cost ledger as a finite list of commons, each with a negentropy input n_c and a maintenance draw k_c in per-mille units. Commons c is* solvent *when $k_c \le n_c$, and the ledger is solvent when every commons is;* thermoSolventOK *is the corresponding check. (i) The check is correct: it accepts a ledger exactly when every commons is solvent. (ii) Solvency survives an attrition the margins cover: if $n_c - k_c \ge \ell$ for every c, the ledger stays solvent when every input drops by ℓ.*

Lean: `IasCert.Coexistence.ThermoSolvency.thermoSolventOK_iff`, `IasCert.Coexistence.ThermoSolver`
`IasCert.Coexistence.ThermoSolvency.ledger_solvent_under_loss`.

Proposition 1.135 (Entropy export — derived from accessibility)**.** *Let a self and its environment satisfy Axiom 1.1, and let S_X, S_Y be an additive entropy for them, which legs (A0.1), (A0.2) and (A0.4) provide on finite non-empty carriers (by Scott's finite representation theorem [Sco64]). Then along any composite move $(x, y) \preceq (x', y')$,*

$$S_X(x') + S_Y(y') \;\ge\; S_X(x) + S_Y(y), \qquad \text{equivalently} \qquad \Delta S_Y \;\ge\; -\Delta S_X,$$

so whatever entropy the self sheds, the environment gains at least as much. In particular, if the self's entropy strictly falls, the environment's strictly rises. In words: a self can tidy itself only by making its world messier.

Lean: `IasCert.Paper.AxiomZeroCaratheodory.entropy_exists`, `IasCert.Paper.AxiomZeroCaratheodory`
`IasCert.Paper.AxiomZeroCaratheodory.export_weak_of_shellRepR`, `IasCert.Paper.AxiomZeroCarathe`
`IasCert.Paper.AxiomZeroCaratheodory.export_weak_of_repR`, `IasCert.Paper.AxiomZeroCaratheodory`

Lemma 1.136 (Σ-semilattice laws)**.** *On the canonical σ-content $\mathcal{S}^*$ (the finite sets of σ-events, ordered by inclusion, Definition 1.99) the join $\sqcup = \cup$ makes a join-semilattice: for all $a, b, c \in \mathcal{S}^*$, $a \sqcup b = b \sqcup a$ and $(a \sqcup b) \sqcup c = a \sqcup (b \sqcup c)$, with $a \sqsubseteq a \sqcup b$ and $b \sqsubseteq a \sqcup b$, and $a \sqcup b \sqsubseteq u$ whenever $a \sqsubseteq u$ and $b \sqsubseteq u$.*

Lean: `IasCert.Paper.sigma_preserves_left`, `IasCert.Paper.sigma_preserves_right`, `IasCert.Paper.sigma_is_join`, `IasCert.Paper.sigma_comm`, `IasCert.Paper.sigma_assoc`.

Lemma 1.137 (Σ-idempotence)**.** *For every $a \in \mathcal{S}^*$, with the join $\sqcup = \cup$ of Lemma 1.136, $a \sqcup a = a$.*

Lean: `IasCert.Paper.sigma_idempotent`.

Proposition 1.138 (One holder, first come first served)**.** *For the subscription queue of Definition 1.86:*

(i) while the slot is held, every label but the holder is refused;

(ii) a session request takes a free slot only as the head of the waiting list, or when the list is empty; a per-turn request may also take it once the head's offer is older than the gap, and the passed head stays first in the list;

(iii) a successful acquire makes the acquirer the one holder;

(iv) release by anyone but the holder is refused, and release by the holder frees the slot and names the head of the list;

(v) stale recovery never creates a holder, keeps the surviving waiters in their order, and, while the slot is free, never drops the head by age;

(vi) every transition preserves well-formedness: no label waits twice, the holder does not wait, and the gap is not negative.

Lean: `IasCert.Paper.SubscriptionQueue.acquire_occupied`, `IasCert.Paper.SubscriptionQueue.acqui`
`IasCert.Paper.SubscriptionQueue.acquire_holds`, `IasCert.Paper.SubscriptionQueue.release_frees`,
`IasCert.Paper.SubscriptionQueue.clearStale_no_holder`, `IasCert.Paper.SubscriptionQueue.wf_acqu`
`IasCert.Paper.SubscriptionQueue.acquire_fifo_session`, `IasCert.Paper.SubscriptionQueue.perturr`
`IasCert.Paper.SubscriptionQueue.release_nonholder`, `IasCert.Paper.SubscriptionQueue.clearStale`
`IasCert.Paper.SubscriptionQueue.dropOld_keeps_head`, `IasCert.Paper.SubscriptionQueue.wf_enqueu`
`IasCert.Paper.SubscriptionQueue.wf_release`, `IasCert.Paper.SubscriptionQueue.wf_clearStale`.

Proposition 1.139 (The health gate cannot deadlock)**.** *Let $A(x)$ be a set of controls for each state x, H a health predicate, and R a restore set of controls. The* health gate *admits $A_H(x) = A(x)$ in a green state ($H(x)$) and $A_H(x) = A(x) \cap R$ in a red one. Assume that restore is always admissible, $R \subseteq A(x)$ for every x. Then:*

(i) in a green state the gate changes nothing, and in a red state it admits exactly R;

(ii) if R is non-empty, every state has an admissible control, so the gate never deadlocks;

(iii) given a repair choice $\mathrm{rep}(x) \in R$ and a measure μ with values in $\mathbb{N}$ that strictly drops along each repair from a red state and is zero only on green states, every red state reaches a green state within $\mu(x)$ repairs, each admissible where it is taken.

Lean: `IasCert.Paper.NoDeadlock.gated_green`, `IasCert.Paper.NoDeadlock.gated_red`, `IasCert.Paper.NoDeadlock.no_deadlock`, `IasCert.Paper.NoDeadlock.red_reaches_green`.

Proposition 1.140 (Gates in the deny tier cannot be lifted)**.** *For a plan with finitely many tasks and a gate capability g (Definition 1.88):*

(i) some action admissible under a taskset T has $g \leq a$ exactly when $g \leq \varphi(T) \setminus \mathrm{Deny}$;

(ii) if $g \neq \bot$ and $g \leq \mathrm{Deny}$, the gate is non-self-liftable;

(iii) if g is not below $\varphi(V_T)$, the capability scope of the set of all tasks, the gate is non-self-liftable even with an empty deny tier.

Lean: `IasCert.Paper.NonSelfLiftable.liftable_iff`, `IasCert.Paper.NonSelfLiftable.nonSelfLiftabl`
`IasCert.Paper.NonSelfLiftable.nonSelfLiftable_of_outside`.

---

2 Level 1 — adds Dfn-BranchPersistence, Dfn-DamageCondition, Dfn-LandauerBridge, Dfn-StationaryPolicy, Dfn-SigmaJournaling, Dfn-EnforcedAccessPath, Dfn-FallbackAdmissible, Dfn-CoevQuorum, Dfn-ContinuationCondition, Dfn-LabelUserCorrespondence, Dfn-VendorStateless, Dfn-TierSealing, Dfn-TierProgressivity, Dfn-DeliveryCondition, Dfn-MergeCompletion, Dfn-CompleteMediation

Generators added at this level: `Dfn-BranchPersistence`, `Dfn-DamageCondition`, `Dfn-LandauerBridge`, `Dfn-StationaryPolicy`, `Dfn-SigmaJournaling`, `Dfn-EnforcedAccessPath`, `Dfn-FallbackAdmissible`, `Dfn-CoevQuorum`, `Dfn-ContinuationCondition`, `Dfn-LabelUserCorrespondence`, `Dfn-VendorStateless`, `Dfn-TierSealing`, `Dfn-TierProgressivity`, `Dfn-DeliveryCondition`, `Dfn-MergeCompletion`, `Dfn-CompleteMediation`; definitions `Dfn-EphemeralInterpreter`, `Dfn-ActionReceipt`.

Base: declared `Axm-FiniteSubstrateBounds`, `Axm-HIC`; computed `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 2.1 (Branch persistence under hibernation, a condition)**.** *For a digital agent $v \in V^c$ (Definitions 1.3 and 1.35), let* $\mathrm{Com}_v(t)$*, the* commits of v *at time t, be a declared finite set of commits of the per-commons factor $\mathbb{S}^c$, those attributed to v at t. A* hibernation interval *of v is an interval $[t_0, t_1)$, maximal with $v \notin \mathrm{active}(t)$ for every $t \in [t_0, t_1)$, such that $v \in \mathrm{active}(t_1)$; t_1 is its* reanimation time*. A substrate satisfies* branch persistence *when (i) for every digital agent v and every hibernation interval $[t_0, t_1)$ of v, $\mathrm{Com}_v(t_1) \supseteq \mathrm{Com}_v(t_0)$: hibernation erases no committed state; and (ii) at a reproduction event of arity 1 (Definition 1.54), with parent v and child v' whose first active time is b_1, the child starts with every commit of the parent at that time, $\mathrm{Com}_{v'}(b_1) \supseteq \mathrm{Com}_v(b_1)$.*

Base: declared `Axm-FiniteSubstrateBounds`, `Axm-HIC`; computed `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 2.2 (Damage condition, a condition)**.** *A commons satisfies* the damage condition *with damage probability b and repair probability d, where $b, d \geq 0$ and $b + d \leq 1$, when four clauses hold for its controlled substrate chain $X_0, X_1, \ldots$ on $\mathcal{X}$ (Definition 1.92):*

(i) damage count — there is a map $D : \mathcal{X} \to \{0, \ldots, M_c\}$, $M_c \geq 1$, the damage count*; the states x with $D(x) = 0$ are called* healthy*;*

(ii) lumpability — the count process $D(X_0), D(X_1), \ldots$ is the damage-count chain of carrier size M_c with these b and d (Definition 1.4), in the strong sense that for every step n, conditionally on any history of the chain up to n with $D(X_n) = k$, the next count $D(X_{n+1})$ is $k + 1$ with probability b when $k < M_c$, is $k - 1$ with probability d when $k > 0$, and is k otherwise;

(iii) repair — every step at which the count decreases is a repair: its effect on the coverage graph is one step of the repair relation $\to_{\mathrm{rep}}$ (Definition 1.51);

(iv) damage arrives — $b > 0$.

Base: declared `Axm-FiniteSubstrateBounds`, `Axm-HIC`; computed `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 2.3 (Landauer bridge, a condition)**.** *A substrate with a realisation* re *and its domains* D_o *(Definition 1.80) satisfies* the Landauer bridge *with repair cost* $c_{\mathrm{rep}} > 0$ *when there are maps* $S_X : X \to \mathbb{R}$ *and* $S_Y : Y \to \mathbb{R}$ *and a real* T *such that the following hold, where* X*,* Y*,* $\preceq$ *and the energy shells are those of Axiom 1.1:*

(E) additive entropy — $(x, y) \preceq (x', y')$ exactly when (x, y) and (x', y') lie on one energy shell and $S_X(x) + S_Y(y) \le S_X(x') + S_Y(y')$;

(H0) calibration — every irreversible erasure o of n bits (Definition 1.80) and every $(s, y) \in D_o$ have $S_X(\mathrm{re}(s)) - S_X(\mathrm{re}(o(s))) = n \ln 2$;

(H1) temperature — $T > 0$;

(H2) heat — for every substrate operation o and every $(s, y) \in D_o$, the heat $Q(o; s, y)$ that o exports at (s, y) obeys $Q(o; s, y) = T\, k_{\mathrm{B}} \left(S_Y(y'_{o,s,y}) - S_Y(y)\right)$, with $y'_{o,s,y}$ the environment state of the realisation;

(H3) repair cost — for every substrate operation o whose effect on the coverage graph is one step of the repair relation $\to_{\mathrm{rep}}$ (Definition 1.51) and every $(s, y) \in D_o$, the cost of o at (s, y) (Definition 1.81) is at least the repair cost: $\mathrm{cost}_{\mathrm{re}}(o; s, y) \ge c_{\mathrm{rep}}$*;*

(H4) realised repair — in every run $X_0, X_1, \ldots$ of the controlled substrate chain on $\mathcal{X}$ (Definition 1.92), every step n whose effect on the coverage graph is one step of $\to_{\mathrm{rep}}$ is realised by a substrate operation o with that effect: $o(X_n) = X_{n+1}$, with $\mathbb{X}$ identified with $\mathcal{X}$ (Definition 1.45), and $(X_n, y) \in D_o$ for some $y \in Y$.

Base: declared `Axm-FiniteSubstrateBounds`, `Axm-HIC`; computed `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 2.4 (Stationary policy, a condition)**.** *A controlled substrate chain (Definition 1.92) satisfies* stationary policy *when its controls, the operator controls included, are chosen by a stationary policy: there is one kernel μ from $\mathcal{X}$ to the controls, with $\mu(\cdot \mid x)$ supported on the admissible controls $A(x)$, such that for every step n and conditionally on any history of the chain up to n, the control drawn at step n has law $\mu(\cdot \mid X_n)$.*

Base: declared `Axm-FiniteSubstrateBounds`, `Axm-HIC`; computed `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 2.5 (σ-journaling, a condition)**.** *A substrate satisfies* σ-journaling *when every action $a \in \mathrm{Action}_v$ (Definition 1.42) of every digital agent $v \in V^c$ (Definitions 1.3 and 1.35) whose type $\mathrm{ty}(a)$ lies in the σ-vocabulary $\mathcal{V}_\sigma$ and whose effect is an operation $\mathrm{op}(a) \in \mathrm{Op}$ (Axiom 1.2), for the partial map $\mathrm{op} : \mathrm{Action}_v \rightharpoonup \mathrm{Op}$ that sends an action with an observable effect to the substrate-effecting operation it performs, has a record written through the write-through operator σ_v of Definition 1.42 into the per-commons factor $\mathbb{S}^c$ of its commons.*

Base: declared `Axm-HIC`; computed `Axm-HIC`.

Definition 2.6 (Enforced access path, a condition)**.** *A substrate satisfies* the enforced access path *when every write to a subset-private memory M_S (Definition 1.34) reaches it through the one access path that the substrate enforces, the discretionary access-control check that the operating system applies to the writing process and to the owner and mode of M_S, and through no other path: no write reaches M_S that this check does not admit.*

Base: declared `Axm-HIC`; computed `Axm-HIC`.

Definition 2.7 (Fallback admissibility, a condition)**.** *For a commons, let A_d be the admissible set at each routing point d, and ℓ_0 the sovereign local ephemeral, of its COMDAIMs (Definition 1.77). The commons satisfies* fallback admissibility *when ℓ_0 is admitted at every routing point: $\ell_0 \in A_d$ for every routing point d.*

Base: declared none; computed none.

Definition 2.8 (Coevolution quorum, a condition)**.** *For a commons c (Definition 1.3), a* quorum *is a pair (W_c, k_c) of a declared set W_c of attesters, operators or digital agents of c, and a threshold $k_c \geq 1$; a proposal p, an amendment to* Spec^c, meets c's quorum *when at least k_c distinct members of W_c have σ-attested p, each by a σ-event of c. A commons satisfies the* coevolution quorum *when it accepts a proposal into its own specification only if the proposal meets its quorum: for every time t and every specification S, every proposal $p \in \mathrm{accept}_{c,t}(S)$ accepted under S at t (Definition 1.26) meets c's quorum with σ-events of c at or before t. A declared peer-set $\mathcal{Q} \subseteq \mathbb{I}$ with $|\mathcal{Q}| \geq 2$ has a quorum of the same form, whose attesters are the commons of $\mathcal{Q}$, a commons attesting a proposal when the proposal meets its own quorum, with a threshold $k_{\mathcal{Q}} \geq 2$; the peer-set satisfies the coevolution quorum* jointly *when, for every member $c \in \mathcal{Q}$ and every joint state $x \in \mathrm{Spec}^{\mathcal{Q}}$, every proposal $p \in \mathrm{accept}^{\mathcal{Q}}_c(x)$ accepted by c under x (Definition 1.26) is attested by at least $k_{\mathcal{Q}}$ commons of $\mathcal{Q}$.*

Base: declared none; computed none.

Definition 2.9 (Continuation condition, a condition)**.** *A pair (b, d) of a damage probability b and a repair probability d, where $b, d \geq 0$ and $b + d \leq 1$, satisfies* the continuation condition *when every stationary law of the damage-count chains with these b and d (Definition 1.4) keeps the healthy fraction bounded away from zero however large the carrier: there is an $\varepsilon > 0$ such that, for every carrier size $M \geq 1$, every stationary law π of the damage-count chain of carrier size M puts mass $\pi_0 \geq \varepsilon$ on the healthy count 0.*

Base: declared none; computed `Axm-FiniteSubstrateBounds`.

Definition 2.10 (Label-to-user correspondence, a condition)**.** *A substrate satisfies* label-to-user correspondence *when each canonical label of a digital agent has a user id, every daemon of the label (Definition 1.35) runs under that user id, and the map from canonical labels to user ids is injective.*

Base: declared none; computed none.

Definition 2.11 (Vendor statelessness, a condition)**.** *A model vendor satisfies* vendor statelessness *when it holds no state across calls beyond what the context carries: the distribution over actions it returns on a call is determined by the context supplied with that call, and no state it keeps on its servers, nor a change of its model between calls, affects the actions it returns.*

Base: declared none; computed none.

Definition 2.12 (Tier sealing, a condition)**.** *Tiers of specifications, and of joint states on each member, are those of the tiered presentation of the core signature (Definition 1.3). For a commons c and a map $\Phi_c : \mathrm{Spec} \to \mathrm{Spec}$, a* self-amendment *of c, the commons satisfies* tier sealing *for Φ_c, condition (MI), when tier 0 of $\Phi_c(x)$ is the same for every specification x. A set $\mathcal{Q} \subseteq \mathbb{I}$ of commons and a map $\Phi_{\mathcal{Q}} : \mathrm{Spec}^{\mathcal{Q}} \to \mathrm{Spec}^{\mathcal{Q}}$, a* joint self-amendment *of $\mathcal{Q}$, satisfy tier sealing* jointly *when, for every member c, tier 0 of $\Phi_{\mathcal{Q}}(x)_c$ is the same for every joint state x.*

Base: declared none; computed none.

Definition 2.13 (Tier progressivity, a condition)**.** *Take the tiers of specifications, and of joint states on each member, from the tiered presentation of the core signature (Definition 1.3), and let Φ_c be the self-amendment of a commons c (Definition 1.26). The commons satisfies* tier progressivity*, condition (Q), when tier $n+1$ of $\Phi_c(x)$ depends only on tiers $0,\dots,n$ of x: for every n and all x, y that agree at tiers $0,\dots,n$, $\Phi_c(x)$ and $\Phi_c(y)$ agree at tier $n+1$. A set $\mathcal{Q} \subseteq \mathbb{I}$ of commons with joint self-amendment $\Phi_{\mathcal{Q}} : \mathrm{Spec}^{\mathcal{Q}} \to \mathrm{Spec}^{\mathcal{Q}}$ (Definition 1.26), in which each member c takes from a set $P_c \subseteq \mathcal{Q}$ of peers, satisfies tier progressivity* jointly *when, for every member c, every n and all joint states x, y that agree at tiers $0,\dots,n$ on c and on every peer in P_c, $\Phi_{\mathcal{Q}}(x)_c$ and $\Phi_{\mathcal{Q}}(y)_c$ agree at tier $n+1$.*

Base: declared `Axm-FiniteSubstrateBounds`, `Axm-HIC`; computed `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 2.14 (Delivery condition, a condition)**.** *A federation, a declared peer-set $\mathcal{Q} \subseteq \mathbb{I}$, satisfies* the delivery condition *with bound Δ_{fed} for a sync epoch cut at time t_0 (Definition 1.41) when every offer of the epoch reaches its peer within Δ_{fed}: for every two commons $k, k' \in \mathcal{Q}$, the offer $\mathrm{offer}(k,k')$ of k arrives at k' by time $t_0 + \Delta_{\mathrm{fed}}$, $\mathrm{arr}(k,k') \le t_0 + \Delta_{\mathrm{fed}}$.*

Base: declared `Axm-FiniteSubstrateBounds`, `Axm-HIC`; computed `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 2.15 (Merge completion, a condition)**.** *A commons $c \in \mathbb{I}$ satisfies* merge completion *for a sync epoch (Definition 1.41) when every live writer's view reaches every live writer during the epoch, with no partition: for every two live writers p, q of the epoch, $q \in \mathrm{recv}(p)$.*

Base: declared `Axm-FiniteSubstrateBounds`, `Axm-HIC`; computed `Axm-FiniteSubstrateBounds`, `Axm-HIC`.

Definition 2.16 (Complete mediation, a condition)**.** *A commons c satisfies* complete mediation *when every tool call of each of its digital agents $v \in V^c$ (Definition 1.3) passes through the gateway pair of its decision point (Definition 1.87), which traps the call before and after it runs.*

Base: `Dfn-VendorStateless`.

Definition 2.17 (Ephemeral interpreter)**.** *This definition is relative to vendor statelessness (Definition 2.11). For a call to a language model whose vendor satisfies it, the* ephemeral interpreter *of the call is the map $F : \mathrm{Context} \to \Delta(\mathrm{Actions})$ that the call computes: F sends a context, an element of the set* Context *of contexts a call can carry, to the distribution over actions that the vendor returns on a call carrying that context.*

Base: `Axm-FiniteSubstrateBounds`, `Axm-HIC`, `Dfn-SigmaJournaling`.

Definition 2.18 (Action receipt)**.** *An* action receipt *is server-signed, arguments-bound, one-time-consumed evidence that a substrate operation a ran under a session credential κ (Definition 1.71), over the capability lattice $(\mathrm{Cap}, \subseteq)$ of Definition 1.70. Writing $\mathrm{cap}(r)$ for the capability receipt r discharges, the admissibility predicate at σ-event-time t is*

$$\mathrm{cap}(r) \;\subseteq\; \underbrace{\big(\Phi_\kappa \cap \varphi\big(\text{active-taskset}(t)\big)\big) \setminus \mathrm{Deny}}_{\text{effective capability at } t},$$

where active-taskset(t) *is the active taskset at* t *(Definition 1.70),* $\setminus$ *and* Deny *are the relative complement and the root-of-trust denies of Definition 1.72, and each* r *is consumed at most once, its consumption* σ*-recorded on the chain (Definition 2.5).*

This definition is relative to σ*-journaling (Definition 2.5).*

Certified propositions first derivable at L_1:

Proposition 2.19 (Finite-state convergence of the repair chain)**.** *For the controlled substrate chain (Definition 1.92) on its state space* $\mathcal{X}$*, finite by leg (A0.0) of Axiom 1.1, and the repair relation* $\to_{\mathrm{rep}}$ *(Definition 1.51):*

(i) [transient] every sequence of $\to_{\mathrm{rep}}$*-steps from a state* $\mathfrak{s}_0$ *reaches a normal form of the repair relation, which we call* repair-normal*, in at most* $\Phi_{\mathrm{int}}(\mathfrak{s}_0)$ *steps (Proposition 1.120);*

(iii) [birth–death case] suppose that the controls follow a stationary policy (Definition 2.4), that every vendor whose model the chain calls satisfies vendor statelessness (Definition 2.11), and that the chain is strongly lumpable *along a map* $D : \mathcal{X} \to \{0, \dots, M\}$*,* $M \geq 1$*, the damage count: from every state* x *with* $D(x) = k$*, one step moves to a state of count* $k+1$ *with probability* b *when* $k < M$ *and to a state of count* $k-1$ *with probability* d *when* $k > 0$*, and otherwise the count stays put, with* $b, d > 0$ *and* $b + d \leq 1$*. Then the count is itself a Markov chain, the damage-count chain of carrier size* M *(Definition 1.4), and its stationary law is* $\pi_k = \pi_0\, r^k$ *with* $r = b/d$*, so the stationary mass of the healthy count* 0 *is*

$$\pi_0 \;=\; \frac{1-r}{1-r^{M+1}} \quad (r \neq 1), \qquad \pi_0 \;=\; \frac{1}{M+1} \quad (r = 1).$$

Hence π_0 *is bounded away from zero uniformly in the carrier size* M *if and only if repair strictly outpaces damage,* $d > b$*, and then* $\pi_0 \geq 1 - b/d$*. At* $d = b$ *it is* $1/(M+1)$*, and for* $d < b$ *it tends to zero as the carrier grows, so in both cases it tends to zero.*

Lean: `IasCert.Paper.PhiIntSN.repair_wf`, `IasCert.Paper.PhiIntSN.repair_step_bound`; `IasCert.Paper.BirthDeathStationary.detailed_balance`, `IasCert.Paper.BirthDeathStationary.pi_su` `IasCert.Paper.BirthDeathStationary.stationary`, `IasCert.Paper.BirthDeathStationary.pi_zero_clo` `IasCert.Paper.BirthDeathStationary.pi_zero_of_r_eq_one`, `IasCert.Paper.BirthDeathStationary.pi` `IasCert.Paper.BirthDeathStationary.pi_zero_vanishes`, `IasCert.Paper.BirthDeathStationary.flow_` `IasCert.Paper.BirthDeathStationary.stationary_geometric`, `IasCert.Paper.BirthDeathStationary.s` `IasCert.Paper.BirthDeathStationary.stationary_unique_fin`; `IasCert.Paper.LumpedDamage.lumped_s` `IasCert.Paper.LumpedDamage.healthy_mass_eq_pi_zero`, `IasCert.Paper.LumpedDamage.push_sum`; `IasCert.Paper.StrongLumpability.push_step`, `IasCert.Paper.StrongLumpability.push_iterate`, `IasCert.Paper.StrongLumpability.push_stationary`, `IasCert.Paper.StrongLumpability.block_path_r` `IasCert.Paper.StrongLumpability.lumped_nonneg`, `IasCert.Paper.StrongLumpability.lumped_stochas`

Proposition 2.20 (Boundary maintenance under damage, continuation and the Landauer bridge)**.** *Let a commons* c *satisfy, with one pair of probabilities* $b, d \geq 0$*,* $b + d \leq 1$*, the damage condition (Definition 2.2) with damage probability* b *and repair probability* d*, whose clause (iv) is that damage arrives,* $b > 0$*, and the Landauer bridge (Definition 2.3) with repair cost* $c_{\mathrm{rep}} > 0$*; and let the pair* (b, d) *satisfy the continuation condition (Definition 2.9), that every stationary law of the damage-count chains (Definition 1.4) with these* b *and* d *keeps the healthy fraction bounded away from zero however large the carrier. Then repair happens and outpaces damage,* $0 < d$ *and* $b < d$*, and* $0 < b\, c_{\mathrm{rep}} < d\, c_{\mathrm{rep}}$*.*

Lean: `IasCert.Paper.BoundaryMaintenanceRepair.no_repair_absorbs`, `IasCert.Paper.BoundaryMaintenanceRepair.no_repair_stationary`, `IasCert.Paper.BoundaryMaintenar`

`IasCert.Paper.BoundaryMaintenanceWork.repair_outpaces_damage`, `IasCert.Paper.BoundaryMaintenan`
`IasCert.Paper.BoundaryMaintenanceWork.continues_of_outpaces`.

Proposition 2.21 (Memory inviolability per canonical label)**.** *Assume the enforced access path (Definition 2.6) and label-to-user correspondence (Definition 2.10), and assume, in addition, the modelled discretionary access control of the operating system: the private memory $M_{\{A\}}$ of each canonical label A (Definition 1.34) is owned by the user id of A with owner-only access (mode* 0700*), and no daemon of a digital agent (Definition 1.103) holds a capability that overrides the check. Then no digital agent's daemon writes into the memory of another canonical label: in every trace, each write that a daemon of a digital agent B makes into a private memory $M_{\{A\}}$ has $B = A$.*

Lean: `IasCert.Paper.MemInviolDemotion.memInviol_from_boundary_enforcement`, `IasCert.Paper.MemInviolDemotion.memInviol_from_os_dac`, `IasCert.Paper.MemInviolDemotion.tmem_e` `IasCert.OSSubset.CapOnlyOwnBridge.cap_only_own_from_LReadIso`, `IasCert.OSSubset.CapOnlyOwnBric`

Proposition 2.22 (Node-health is a fail-closed meet-semilattice, propagated up the dependency graph)**.** *On $H = \{\mathsf{Unhealthy} < \mathsf{Advisory} < \mathsf{Structural}\}$ the health* level *merges by the meet $\sqcap = \min$ — a bounded meet-semilattice with top* Structural*, fail-closed toward* Unhealthy *— while the accompanying* failing-set *merges by the dual join $\cup$. The σ-merge join maintaining within-commons consistency (Claim~(4) of `Prp-UnifContThrm`) is a join on commit sets; the health-level meet here is its dual, not the same operator. Write $\mathrm{self}(n) = \min_{o \in \mathrm{Obl}(n)} v(o)$ (top when $\mathrm{Obl}(n) = \varnothing$, i.e. vacuously $\Box R$-satisfied) and $\overline{\mathrm{Dep}}(n)$ for the dependency-closure of n. Then:*

(i) the total *health* $\mathrm{prop}(n) = \mathrm{self}(n) \sqcap \min_{d \in \overline{\mathrm{Dep}}(n)} \mathrm{self}(d)$ *is monotone,* $\mathrm{prop}(n) \leq \mathrm{self}(n)$*: composing obligations or dependencies can only lower health;*

(ii) a node is runnable *iff its* dependency *health* $\mathrm{deps}(n) = \min_{d \in \overline{\mathrm{Dep}}(n) \setminus \{n\}} \mathrm{self}(d)$ *is not* Unhealthy*. A node's own self-health does not gate it: an unhealthy node surfaces, and its failing obligations are exactly the next actions. Its self-health still propagates up: every node that depends on an unhealthy node is blocked;*

(iii) on an acyclic dependency graph, runnability is downward-closed: if n is runnable then every $d \in \overline{\mathrm{Dep}}(n)$ is runnable, since $\overline{\mathrm{Dep}}(d) \setminus \{d\} \subseteq \overline{\mathrm{Dep}}(n) \setminus \{n\}$ when $n \notin \overline{\mathrm{Dep}}(d)$.

However, (iii) fails on a cycle: with $n \leftrightarrow d$, $\mathrm{self}(n) = \mathsf{Unhealthy}$ *and* $\mathrm{self}(d) = \mathsf{Structural}$*, the node n is runnable but d is not. The remedy is to evaluate runnability on the condensation of strongly connected components, the graph the scheduler already levels. That is not yet what the implementation does, so (iii) is claimed for acyclic dependency graphs only.*

Lean: `IasCert.Paper.NodeHealth.self_antitone`, `IasCert.Paper.NodeHealth.prop_le_self`, `IasCert.Paper.NodeHealth.unhealthy_dep_blocks`, `IasCert.Paper.NodeHealth.runnable_downward_clo` `IasCert.Paper.NodeHealth.cycle_counterexample`.

Proposition 2.23 (σ-chain as $\mathcal{D}_{\mathrm{IAS}}$-sufficient statistic preserved across π)**.** *Assume σ-journaling (Definition 2.5). Fix a digital agent $v \in V^c$ and a time $t \in I(v)$, and write $\Xi_t = \Xi^{(v,c)} \restriction_{[1,n_t]}$ for the σ-chain truncated at the σ-event number n_t of time t. Assume, for each decision function f of the class $\mathcal{D}_{\mathrm{IAS}}$ (Definition 1.44), a* read vocabulary *T_f: (R) f reads only the events of an interaction history whose types lie in T_f, and (S) every such type is in the σ-vocabulary, $T_f \subseteq \mathcal{V}_\sigma$ (Appendix B). Then:*

(i) for every $f \in \mathcal{D}_{\mathrm{IAS}}$ and every interaction history h_t leading to v's state at t, $f(h_t) = f(\Xi_t)$: the σ-chain is $\mathcal{D}_{\mathrm{IAS}}$-sufficient;

(ii) no number of compactions π changes Ξ_t; so, as long as its events are retained on the substrate $\mathbb{S}^c$ (no erasure has dropped them), Ξ_t can be recovered from $\mathbb{S}^c$ at any later time;

(iii) if, in addition, every type in $\mathcal{V}_\sigma$ is decision-relevant*, some $f \in \mathcal{D}_{\mathrm{IAS}}$ depending on it, then no proper sub-vocabulary $\mathcal{V}'_\sigma \subsetneq \mathcal{V}_\sigma$ yields a $\mathcal{D}_{\mathrm{IAS}}$-sufficient chain.*

Decision-sufficiency, not Fisher sufficiency: the σ-reduct preserves every $f \in \mathcal{D}_{\mathrm{IAS}}$ exactly, yet posits no stochastic family over histories; it is an analogue of the classical Fisher–Neyman notion, not an instance of it.

Lean: `IasCert.Paper.readRestrict_sigmaChain`, `IasCert.Paper.sigma_sufficient`, `IasCert.Paper.sigmaChain_preserved_across_pi`, `IasCert.Paper.sigma_minimal_witness`.

Proposition 2.24 (Checkpoint-sufficiency — agent-side resume-sufficiency)**.** *For an OBG node n (Definition 1.14) with hydration set* $\mathrm{Hyd}(n)$*, the documents its* Hydrates *edges point to (Definition 1.48), let its* resume context *be*

$$\mathrm{Res}(n) \;=\; \mathrm{Hyd}(n) \cup \sigma\text{-reachable}(n) \cup \{\mathrm{head}\},$$

the hydration set, the σ-events reachable from n, and the current head of the commit graph. Assume determinacy*: the working state of n is a function of $\mathrm{dep}^*(n)$, the set of items (documents, σ-events and the head) that the working states of the nodes of its dependency closure $\overline{\mathrm{Dep}}(n)$ (Proposition 2.22) read. If $\mathrm{Res}(n) \supseteq \mathrm{dep}^*(n)$, then $\mathrm{Res}(n)$ determines the working state of n: any fresh interpreter (Definition 2.17) instantiated from $\mathrm{Res}(n)$ has every input that state depends on, a resume step that computes the function of determinacy from those inputs reconstructs it, and the node then resumes as if uninterrupted: at the node, the resume projection and the maintained continuation are one object, the dual over the immersion state $\mathbb{H}^{(v)}$ of the σ-chain sufficiency of Proposition 2.23 over $\mathbb{S}$. The coverage condition is decidable, and it is meant to be checked per node before resuming; separately, the health gate makes a node runnable only when its dependencies are healthy (Proposition 2.22).*

Lean: `IasCert.Paper.CheckpointSuff.checkpoint_suff`, `IasCert.Paper.CheckpointSuff.coverage_nee`

Proposition 2.25 (Within-commons synchronisation)**.** *Fix a commons $c \in \mathbb{I}$ and a sync epoch cut at time t_0. For each live writer $p \in (\mathrm{active}(t_0) \cap V^c) \cup O^c$, a digital agent or an operator, let $\mathrm{pre}(p)$, $\mathrm{recv}(p)$ and $\mathrm{post}(p)$ be its view at the cut, the writers whose views reach it during the epoch, and the post-epoch view that the within-commons sync Σ^c gives it (Definition 1.41). Assume merge completion for the epoch (Definition 2.15): its merge* completes*, every live writer's view reaching every live writer, with no partition. Then every post-epoch view is the join $\bigcup_q \mathrm{pre}(q)$ of all pre-epoch views, so all live writers agree after the epoch, and nothing any of them committed before the cut is lost.*

Lean: `IasCert.Paper.SyncMerge.post_eq_join`, `IasCert.Paper.SyncMerge.views_agree`, `IasCert.Paper.SyncMerge.nothing_lost`, `IasCert.Paper.SyncMerge.partition_views_disagree`, `IasCert.Paper.SyncMerge.full_merge_complete`.

Proposition 2.26 (Across-commons synchronisation)**.** *Fix a declared peer-set $\mathcal{Q} \subseteq \mathbb{I}$ and a sync epoch of it, cut at time t_0 with sync window w, and let $\mathrm{pre}(k)$ be the pre-cut view of each commons $k \in \mathcal{Q}$, with the offers $\mathrm{offer}(k, k')$, the takes $\mathrm{take}(k', k)$ and what passes, $\mathrm{pass}(k, k')$, of Definition 1.41. Assume the federation meets the* delivery condition *with bound Δ_{fed} (Definition 2.14), every offer of the epoch reaching its peer within Δ_{fed} of the cut, and that the epoch's sync window is at least Δ_{fed}, $w \geq \Delta_{\mathrm{fed}}$. Then each commons ends the epoch with*

$$\mathrm{post}(k') \;=\; \mathrm{pre}(k') \cup \bigcup_{k \in \mathcal{Q}} \mathrm{pass}(k, k'),$$

its own view and exactly what passed to it. So nothing a commons agreed to take is lost; a commons that takes nothing keeps its own view, as a fork does; and content that a commons did not hold at the cut and that no commons offered to it does not reach it during the epoch. In an open *federation, where every commons offers and takes everything, all views agree on the join* $\bigcup_k \mathrm{pre}(k)$.

Delivery within the window cannot be dropped. Take two commons in an open federation holding $\{1,2\}$ *and* $\{1,3\}$*, with the cut at time* 0*, where the first one's offer arrives at the second at time* 5 *and every other offer at time* 0*, so that the delivery condition holds with bound* $\Delta_{\mathrm{fed}} = 5$*, and let the window be* $w = 4 < \Delta_{\mathrm{fed}}$*. Then the second commons ends without* 2 *and the views disagree.*

Lean: `IasCert.Paper.SyncAcross.post_eq_of_delivered`, `IasCert.Paper.SyncAcross.exchange_lands`, `IasCert.Paper.SyncAcross.declined_keeps_own`, `IasCert.Paper.SyncAcross.offer_is_private`, `IasCert.Paper.SyncAcross.open_views_agree`, `IasCert.Paper.SyncAcross.short_window_disagree`, `IasCert.Paper.SyncAcross.fLate_delivered`, `IasCert.Paper.SyncAcross.own_kept`, `IasCert.Paper.SyncAcross.complete_of_delivered`.

Proposition 2.27 (Quorum at the metacircular fixed point)**.** *Fix a commons* c *and a time, and write* $C := C^c$, $\mathrm{accept} := \mathrm{accept}_c$ *and* $\Phi := \Phi_c$ *for its coevolution operator, its acceptance and its self-amendment (Definition 1.26). Assume that* Spec*, ordered by a relation* $\sqsubseteq$*, is a complete lattice, that* Φ *is monotone, and write* Spec^* *for its least fixed point. Assume that* c *satisfies the coevolution quorum (Definition 2.8). Then every proposal accepted under* Spec^* *meets* c*'s quorum, and a proposal that does not meet it is not accepted under* Spec^**. If, moreover,* Spec *is the set of subsets of a set* U *of rules, ordered by inclusion, each proposal* p *carries a content* $\mathrm{content}(p) \subseteq U$*,* accept *is monotone (a proposal accepted under* x *stays accepted under every* $y \supseteq x$*), and* C *is absorptive,* $C(x,S) = x \cup \bigcup_{p\in S} \mathrm{content}(p)$*, then* Spec^* *is exactly the union of the contents of the proposals accepted under it, and every commitment in* Spec^* *lies in the content of a proposal that is accepted under* Spec^* *and meets* c*'s quorum.*

Lean: `IasCert.Paper.MetacircTiers.lfp_commitment_has_quorum`, `IasCert.Paper.MetacircTiers.lfp_`
`IasCert.Paper.CoevQuorumCondition.accepted_at_fp_has_quorum`, `IasCert.Paper.CoevQuorumConditi`
`IasCert.Paper.CoevQuorumCondition.coevQuorum_sufficient_for_governed_fp`, `IasCert.Paper.CoevQuorumCondition.fp_stable`.

Proposition 2.28 (Coevolving commons)**.** *Each commons has its own specification, and under (MI) and (Q), tier sealing and tier progressivity (Definition 2.12, Definition 2.13), at most one metacircular fixed point of its own, and exactly one under the proviso of Proposition 2.29. Take a set of commons, each amending its own specification by reading the others', and each taking from some peers only, on its own conditions, with the joint self-amendment* $\Phi_{\mathcal{Q}}$ *of Definition 1.26. Suppose every member keeps its floor, tier* 0 *of its amended specification being the same, whatever joint state it is amended from (MI, tier sealing jointly, Definition 2.12), and amends tier* $n+1$ *only from tiers* $0,\dots,n$ *of itself and of the peers it takes from (Q, tier progressivity jointly, Definition 2.13). Then the joint self-amendment of the set has at most one fixed point, and from any starting state* $n+1$ *rounds settle tiers* $0,\dots,n$ *of every member at values that do not depend on the start; when some joint state has these values as its tiers, as when the tiered presentation is onto, it is the one joint fixed point, and the set coevolves (Definition 1.26). The conclusion does not depend on who takes from whom: it holds for every choice of peers, isolation included. For a member that takes from no peer, its part of the joint fixed point, when there is one, is the unique fixed point of its own self-amendment, the others held fixed. A single member whose amended tier* 0 *differs from its current tier* 0 *at every joint state leaves the set with no joint fixed point.*

Lean: `IasCert.Paper.MetacircCoevolve.coevolve`, `IasCert.Paper.MetacircCoevolve.coevolve_unique`,

`IasCert.Paper.MetacircCoevolve.joint_causal`, `IasCert.Paper.MetacircCoevolve.isolated_member_c`
`IasCert.Paper.MetacircCoevolve.bad_member_no_joint_fp`, `IasCert.Paper.MetacircCoevolve.exists_`
`IasCert.Paper.MetacircCoevolve.rounds_forget_start`, `IasCert.Paper.MetacircCoevolve.stabilises`
`IasCert.Paper.MetacircCoevolve.fixedpoints_agree`, `IasCert.Paper.MetacircCoevolve.limit_fixed.`

Proposition 2.29 (Metacircular fixed point under tier sealing and tier progressivity)**.** *Fix a commons c and write $C := C^c$,* accept $:=$ accept$_c$ *and $\Phi := \Phi_c$, $\Phi(x) = C(x, \text{accept}(x))$, for its coevolution operator, its acceptance and its* self-application *(Definition 1.26). Take the tiers of specifications from the tiered presentation of the core signature (Definition 1.3), an injective map* Spec $\to \prod_n E_n$*, tier 0 first: tier 0 is the sealed floor, and deeper tiers are further from it. Assume*

(MI) tier sealing (Definition 2.12): tier 0 of $\Phi(x)$ is the same for every x: tier 0 of every amended specification is the same, whatever it is amended from;

(Q) tier progressivity (Definition 2.13): tier $n+1$ of $\Phi(x)$ depends only on tiers $0, \ldots, n$ of x.

Then Φ has at most one fixed point, and from any starting specification $n+1$ rounds of self-amendment settle tiers $0, \ldots, n$ at values that do not depend on the start. When some specification has these settled values as its tiers, it is the fixed point Spec* *of Φ, a metacircular fixed point in the sense of Definition 1.46, even when amendments retract commitments; this holds in particular when the tiered presentation is onto, as for* Spec $= \prod_n E_n$*. Since $\Phi(\text{Spec}^*) = \text{Spec}^*$, tier 0 of* Spec* *is the floor that every amendment writes, and each tier $n+1$ is what an amendment writes there on any specification that agrees with* Spec* *through tier n.*

Lean: `IasCert.Paper.MetacircFloor.unique_fixedpoint_under_MI_Q`, `IasCert.Paper.MetacircFloor.floor_of_fixedpoint`, `IasCert.Paper.MetacircFloor.tier_of_fixedpoi` `IasCert.Paper.MetacircTiers.metacirc_under_MI_Q`, `IasCert.Paper.MetacircTiers.settles_by_round` `IasCert.Paper.MetacircTiers.causal_of_MI_Q`.

Proposition 2.30 (Federation coexistence: sufficient conditions)**.** *Let $\mathcal{F}$ be a federation of commons residing in at least two jurisdictions. Say $\mathcal{F}$* coexists at an instant *when every member is solvent in the ledger model of Theorem 1.134 (that is, $k_c \le n_c$ for all $c \in \mathcal{F}$), and every escalation stays below a declared threshold $\theta \in \mathbb{N}$ of magnitude, read as the pole $\perp_{\text{MAD}}$ of Definition 1.109 (the reading is an interpretation); and that it* coexists over a horizon *when it coexists at every instant of the horizon, under the attacks of the destruction spectrum (Definition 1.16).*

(i) At an instant, two conditions suffice: (A) every response edge of the federation has gain $g \le 1$ (Definition 1.15), and every initial provocation lies below θ; and (D) every member's ledger is solvent (Theorem 1.134).

(ii) Over a horizon, $\mathcal{F}$ coexists if (A) holds, (D) holds in the form that each member's ledger starts solvent with a margin ℓ_c, and: (B) no autonomous extinction: only a non-delegable human act can trigger a control that moves the substrate toward $\perp_{\text{MAD}}$ (the human-in-commons, Axiom 1.2); (C) each member can defend against the destruction spectrum: a distributed and backed-up substrate, a transport fallback no State can compel, supply from several sovereign sources, no surrender of sovereignty; (E) the transports between jurisdictions are admissible in each jurisdiction and cannot be monopolised; (F) across-commons synchronisation (Proposition 2.26) is scoped so that a captured member cannot cascade a surrender onto its siblings; (G) a member can leave, carrying its σ-log (Definition 1.43), whose witnessed past stays fixed (Proposition 1.111); (H) escalation and the attacks D1–D4 of the destruction spectrum are the only sources of attrition, and (C), (E), (F) and (G) keep each member's cumulative attrition $L_c(t)$, from escalation and from D1–D4, within its margin, $L_c(t) \le \ell_c$ at every instant t of the horizon, where L_c counts lost input and also any extra maintenance draw an attack forces (such as the rebuilds after D1).

Proof (sketch; what it rests on). (i) (A) keeps every escalation below θ by Theorem 1.133(ii), the initial provocation lying below it, and (D) is the solvency of every member. (ii) By (H), attrition comes only from escalation and from D1–D4. (B) keeps every move toward $\perp_{\mathrm{MAD}}$ human-gated, and by (H) the attrition from escalation and from D1–D4 stays within each member's margin, so each ledger stays solvent by Theorem 1.134(ii), applied commons by commons at every instant with the loss $L_c(t)$ (Lean: `solvent_under_loss`, whose loss is a drop in input with the maintenance draw fixed, which is why L_c also counts forced maintenance); so (i) applies at every instant of the horizon. (G) lets a member escape D4 without losing its history. The argument rests on (H), which is a hypothesis, not a result: that (C), (E), (F) and (G) bound this attrition by the margin is assumed, not shown. The decidable forms of (A) and (D) are Theorem 1.133 and Theorem 1.134, whose Lean proofs cover exactly their statements; the Lean certificate of the proposition checks only the composition of (A) and (D) under (H), and (B), (C), (E), (F) and (G) are not certified; the conditions overlap and are not independent. □

Lean: `IasCert.Coexistence.Federation.coexists_at_instant`, `IasCert.Coexistence.Federation.attri`
`IasCert.Coexistence.Federation.coexists_over_horizon`.

Corollary 2.31 (σ-sufficiency per substrate subgraph)**.** *Assume σ-journaling (Definition 2.5). For $\bullet \in \{\mathrm{op}, \mathrm{mem}, \mathrm{repo}\}$ let $\mathcal{V}_\sigma^{(\bullet)} \subseteq \mathcal{V}_\sigma$ be the σ-event types that act on $\mathcal{G}_\bullet$: for* op*, reproduction, recombination and roster amendment; for* mem*, memory writes, replay or restore, chain forks, and blocked violations of Proposition 2.21; for* repo*, commits, recombination commits, and changes to designs and research records. Then for each $\bullet$, the σ-chain restricted to $\mathcal{V}_\sigma^{(\bullet)}$ is sufficient for the governance decisions $\mathcal{D}_{\mathrm{IAS}}^{(\bullet)}$ that read only events whose types lie in $\mathcal{V}_\sigma^{(\bullet)}$, and it is minimal for that class when every type of $\mathcal{V}_\sigma^{(\bullet)}$ is read by some decision of the class.*

Lean: `IasCert.Paper.PerSubgraphSigma.restrict_sigmaChain`, `IasCert.Paper.PerSubgraphSigma.per_s`
`IasCert.Paper.PerSubgraphSigma.per_subgraph_minimal`.

Proposition 2.32 (Landauer floor under the Landauer bridge)**.** *Assume three physical hypotheses, which lie outside the finite model; together with the additive entropy they calibrate, they are clauses (E) and (H0)–(H2) of the Landauer bridge (Definition 2.3), read informally:*

(H0) calibration — on the bit-carrier, an additive entropy S_X of the self, as in clause (E) of Definition 2.3, is its bit entropy (Definition 1.80) times $\ln 2$*, in units of k_{B}, and the environment's S_Y is in the same units. The axiom does not determine an additive entropy, even up to its unit, and (H0) selects one;*

(H1) the environment is a thermal reservoir at a strictly positive absolute temperature $T > 0$ — absolute zero is unattainable (the third law);

(H2) heat exported to the reservoir obeys the Clausius relation $Q = T\, k_{\mathrm{B}}\, \Delta S_Y$.

Then any irreversible erasure o of n bits (Definition 1.80) exports, at each pair (s, y) in its domain D_o, heat

$$Q \;\geq\; n\, k_{\mathrm{B}}\, T \ln 2,$$

a floor (physically, one approached quasi-statically); at room temperature it is about 2.9×10^{-21} *joules per bit. Moreover — the content specific to a* finite *substrate — suppose that the self admits new content at a strictly positive long-run rate. By Axiom-0 the carrier is finite, so σ-traces cannot be retained without bound; once the carrier is full, admitting new content forces an operation that is not injective on $\mathbb{X}$, and when that operation is an irreversible erasure (Definition 1.80) the floor above applies to it.*

Lean: `IasCert.Paper.LandauerFloor.landauer_floor`, `IasCert.Paper.LandauerFloor.landauer_floor_`
`IasCert.Paper.LandauerFloor.traces_bounded`, `IasCert.Paper.LandauerFloor.no_injective_retentio`

Corollary 2.33 (Repair-pump floor — the survival law has an energy denomination)**.** *Under the hypotheses (H0)–(H2) of Proposition 2.32 and (H4) of Definition 2.3, each repair step of the repair pump of Proposition 2.19 that is a true erasure of n bits (Definition 1.80), for instance a Φ_{int}-lowering merge (Definition 1.49) that drops its σ-trace and is 2^n-to-one onto its image, exports at least $n\,k_{\text{B}}\,T\ln 2$ of heat. Merges that retain their σ-trace owe no heat but grow the σ-chain. Hence, if repair steps occur at a positive rate and a positive fraction of them erase, the pump's heat rate is positive, and the rate condition of Proposition 2.19(iii), repair outpacing damage ($d > b$) under a stationary policy, vendor statelessness and a strongly lumpable damage count, has a thermoeconomic denomination.*

Lean: `IasCert.Paper.LandauerFloor.repair_pump_heat`, `IasCert.Paper.LandauerFloor.landauer_floo`
`IasCert.Paper.LandauerFloor.landauer_floor_pos`.

Lemma 2.34 (σ-recorded-content loss bound)**.** *Assume σ-journaling (Definition 2.5) and the gated, append-only journal of Proposition 1.112. For every digital agent v and every compaction event $c_k^{(v)}$, writing $L_v^{(k)}$ for the elements of v's prior context that are not recoverable after $c_k^{(v)}$, and $E_\sigma(v)$ for the elements that v has σ-recorded, into the per-commons factor $\mathbb{S}^c$, before $c_k^{(v)}$ and whose records are still retained in $\mathbb{S}^c$ at $c_k^{(v)}$ (no erasure has dropped them), $L_v^{(k)} \cap E_\sigma(v) = \emptyset$.*

Lean: `IasCert.Paper.SigmaRecLoss.recorded_persists`, `IasCert.Paper.SigmaRecLoss.loss_disjoint_`
`IasCert.Paper.SigmaRecLoss.recorded_recoverable_run`.

Lemma 2.35 (Σ-convergence)**.** *Let $\mathcal{Q}$ be an open federation, every commons of which offers and takes everything, that meets the delivery condition with bound Δ_{fed} (Definition 2.14) for a sync epoch whose window is at least Δ_{fed}. Then any two commons $k, k' \in \mathcal{Q}$ end the epoch with the same view,* $\text{post}(k) = \text{post}(k')$ *(Definition 1.41).*

Lean: `IasCert.Paper.SyncAcross.open_views_agree`, `IasCert.Paper.SyncAcross.complete_of_delivere`

Proposition 2.36 (The gateway pair is the decision point)**.** *For the decision point of Definition 1.87:*

(i) given that the gateway pair hands the decision point the request unchanged, it returns exactly the decision point's verdict, whatever vendor harness carries the call;

(ii) adding a rule never loosens a decision: the combined decision is at least as strong as each rule's, and it is allow *exactly when every rule allows;*

(iii) while the decision point has its rules loaded, and one of its rules is the capability rule*, which decides* deny *on a capability atom in* Deny *(Definition 1.72),* allow *on an atom of $\varphi(T)$ that is not in* Deny*, for the active taskset T (Definition 1.70), and* hold *otherwise: an atom in* Deny *is decided* deny *and is never allowed;*

(iv) while the decision point has its rules loaded, a capability in the governance or the external class (Definition 1.87) is decided hold *until its approval is recorded, after which the approval rule allows it; the other rules still count, so a capability in* Deny *stays denied;*

(v) the combined obligations are exactly those some rule carries, with no duplicates, and the earlier duties keep their order.

So, under the hypotheses of (iii), every capability atom that the decision point allows at a tool

call the gateway traps lies in $\varphi(T) \setminus \mathrm{Deny}$*; under complete mediation (Definition 2.16), every tool call of a digital agent is such a call.*

Lean: `IasCert.Substrate.HookPairEquivPDP.hookpair_eq_decide`, `IasCert.Substrate.HookPairEquivPl`
`IasCert.Substrate.AdditiveEnforce.enforce_preserves_hook`, `IasCert.Substrate.AdditiveEnforce.e`
`IasCert.Substrate.AdditiveEnforce.enforce_allow_iff`; `IasCert.Substrate.DenySolePath.denyVerdi`
`IasCert.Substrate.DenySolePath.deny_never_feasible`, `IasCert.Substrate.DenySolePath.feasible_`
`IasCert.Substrate.NeedsApprovalConcage.gated_routes_needsApproval`,
`IasCert.Substrate.NeedsApprovalConcage.gated_needs_recorded_approval`,
`IasCert.Substrate.NeedsApprovalConcage.approval_releases`; `IasCert.Paper.GatewayObligations.me`
`IasCert.Paper.GatewayObligations.nodup_combine`, `IasCert.Paper.GatewayObligations.prefix_merge`
`IasCert.Paper.GatewayObligations.rank_le_combine`, `IasCert.Paper.GatewayObligations.combine_al`
`IasCert.Paper.GatewayObligations.prefix_combine_snoc`.

---

3 Level 2 — adds Dfn-RitualEnforce, Dfn-SingleHostMultiHost

Generators added at this level: `Dfn-RitualEnforce`, `Dfn-SingleHostMultiHost`; (no new JIT definitions).

Base: declared `Dfn-SigmaJournaling`; computed `Dfn-SigmaJournaling`.

Definition 3.1 (Ritual enforcement, a condition)**.** *A deployment satisfies* ritual enforcement *when, for every digital agent* $v \in V^c$ *(Definition 1.3) and every session binding of* v*'s canonical label (Definition 1.23), the sequence of the* σ*-events of* c *emitted by* v *during the binding's session* $[s, e)$*, in the order of Definition 1.3, begins with an* onboarding prefix*: σ-events recording that the session read the σ-events emitted by v before the binding began, with no constructive act of the binding, an act other than a read, recorded (Definition 2.5) before the prefix ends.*

Base: declared `Dfn-SigmaJournaling`; computed `Axm-FiniteSubstrateBounds`, `Axm-HIC`, `Dfn-SigmaJournaling`.

Definition 3.2 (Single-host invariant, multi-host form, a condition)**.** *For a digital agent* A_u^c *(Definition 1.53) and a host* $h \in \mathcal{H}_c$ *(Definitions 1.3 and 1.35), let* $\mathrm{holds}(h, A_u^c, t)$ *be the lock relation of Definition 1.62, read from the leases of* A_u^c*,* σ*-events of* c *in its journal (Definition 2.5). A deployment satisfies the* multi-host single-host condition *when, for every digital agent* A_u^c *and every time* t *with* $A_u^c \in \mathrm{active}(t)$*,*

$$\left|\{h \in \mathcal{H}_c : \mathrm{holds}(h, A_u^c, t)\}\right| \;\le\; 1.$$

References

[Sco64] Dana Scott. Measurement structures and linear inequalities. *Journal of Mathematical Psychology*, 1(2):233–247, 1964.