The EGON Policy Compiler

A Deterministic Compilation from Failure Semantics to Policy Algebra

Extended Edition (Version 1)

Adrian Diamond

January 2026

https://aadsystems.com/egon/compiler

Contents

Abstract

We introduce the EGON Policy Compiler, a deterministic compiler that maps a finite semantic algebra of failure classes into a structured algebra of policy bundles. Building on the AEGON Algebra, which defines a total classifier from normalized observations to a closed ontology of failure semantics, the EGON compiler realizes a canonical compilation map from semantic regimes to policy artifacts. We define the target policy algebra, establish determinism, neutrality, and totality of compilation, and show that non-action is a first-class compiled result. The compiler admits a minimal policy language and a canonical serialization, while remaining independent of execution, inference, learning, and control. This extended edition presents the full mathematical framework, correctness arguments, and illustrative examples.

1 Semantic Domains and Policy Objects

We begin by recalling the semantic framework introduced in the AEGON Algebra and extending it with a formal policy target suitable for compilation.

Definition 1 (Failure-Class Ontology)**.** *Let*

$$F = \{F_0, F_1, \ldots, F_n\}$$

be a finite, closed set of failure classes, where F_0 denotes the neutral semantic regime corresponding to the absence of violated structural invariants. The set F is equipped with a partial order $\leq$ representing semantic dominance, as defined in the AEGON Algebra.

Definition 2 (Observation Space)**.** *Let O denote the space of normalized system observations. Elements of O are structured representations of system state derived from raw telemetry, signals, or invariant checks. The internal structure of O is not relevant to the present work.*

Definition 3 (Deterministic Classifier)**.** *A deterministic classifier is a total function*

$$C : O \to F$$

such that for all $o \in O$, repeated evaluation of $C(o)$ yields the same element of F.

The classifier C is assumed fixed throughout this work and is not modified by the policy compiler.

Definition 4 (Policy Action Symbols)**.** *Let A be a finite set of policy action symbols. Each action symbol $a \in A$ may be parameterized by a finite key–value map. Action symbols represent declarative intent and do not imply execution.*

Definition 5 (Capability Gates)**.** *Let G denote the set of capability gates. A capability gate is a tuple*

$$g = (\tau, \kappa)$$

where τ is a tier identifier and κ is a finite set of required capabilities.

Definition 6 (Policy Bundle)**.** *A policy bundle is a tuple*

$$p = (f, g, \alpha, e, \nu)$$

where:

- *$f \in F$ is the triggering failure class,*

- $g \in G$ *is a capability gate,*
- $\alpha = (a_1, \ldots, a_k)$ *is an ordered sequence of actions from* A*,*
- e *is an evidence specification defining required explanatory artifacts,*
- ν *is a finite annotation set intended for human interpretation.*

Let P *denote the set of all policy bundles.*

2 The Policy Algebra and Neutral Element

Definition 7 (Policy Monoid)**.** *Let* $(P, \oplus, p_0)$ *be a monoid, where:*

- P *is the set of policy bundles,*
- $\oplus$ *denotes policy composition under compatibility,*
- p_0 *is the neutral policy bundle.*

Definition 8 (Neutral Policy)**.** *The neutral policy bundle* p_0 *is defined by*

$$p_0 = (F_0, g, (), e, \nu),$$

where the action sequence is empty or contains a designated no-op action.

Proposition 1 (Existence and Uniqueness of Neutral Policy)**.** *There exists a unique neutral policy bundle* $p_0 \in P$ *such that*

$$p_0 \oplus p = p \oplus p_0 = p$$

for all $p \in P$ *whenever composition is defined.*

Remark 1. *The explicit representation of neutrality ensures that non-action is a first-class semantic outcome rather than an implicit default.*

3 The EGON Policy Compiler

Definition 9 (Policy Compiler)**.** *The EGON Policy Compiler is a deterministic map*

$$\Pi : F \to P$$

that assigns to each failure class $f \in F$ *a canonical policy bundle* $\Pi(f)$*.*

The composed map

$$\Pi \circ C : O \to P$$

is therefore deterministic by construction.

Theorem 1 (Determinism)**.** *For all* $f \in F$*, repeated evaluation of* $\Pi(f)$ *yields identical policy bundles.*

Theorem 2 (Semantic Neutrality)**.**

$$\Pi(F_0) = p_0,$$

where p_0 *is the neutral policy bundle.*

Corollary 1 (Totality)**.** *The policy compiler* Π *is total over* F*.*

Remark 2. *The compiler produces declarative policy artifacts only. Execution, inference, learning, and enforcement are explicitly excluded from the compiler's semantic domain.*

4 A Minimal Policy Language and Canonical Syntax

Although the policy compiler is defined abstractly as a map

$$\Pi : F \to P,$$

it is often convenient to specify policy intent using a finite syntactic language. This section introduces a minimal policy language whose sole purpose is to provide a canonical syntax for constructing elements of P. The language is not part of the semantic theory itself; it is a presentation layer whose semantics are fully determined by compilation.

Definition 10 (Policy Language Alphabet)**.** *Let Σ be a finite alphabet consisting of:*

- *failure-class symbols corresponding bijectively to elements of F,*
- *action symbols corresponding to elements of A,*
- *parameter symbols drawn from a finite key–value domain,*
- *control keywords specifying tiering, mode, and guards.*

Definition 11 (Policy Expressions)**.** *A policy expression is a finite syntactic object specifying a tier identifier, a failure-class symbol, and an ordered list of action expressions. Let L denote the set of all such policy expressions.*

Remark 3. *The language L admits no recursion, loops, conditionals, or variable binding. All expressiveness arises from the choice of failure class and the ordered action sequence.*

Definition 12 (Parsing Function)**.** *A deterministic parsing function*

$$\mathcal{P} : L \to F \times G \times A^*$$

maps a policy expression to its semantic components.

Definition 13 (Policy Interpretation)**.** *The semantic interpretation of a policy expression $\ell \in L$ is defined by*

$$\ell = \Pi(f),$$

where $(f, g, \alpha) = \mathcal{P}(\ell)$.

Proposition 2 (Semantic Irrelevance of Syntax)**.** *If two policy expressions parse to the same tuple (f, g, α), then they denote the same policy bundle.*

Theorem 3 (Canonicalization)**.** *For every policy expression $\ell \in L$, the compiler emits a unique canonical policy bundle.*

Corollary 2 (Idempotence)**.** *Recompiling a canonical policy bundle yields an identical bundle.*

5 Compilation Correctness and Structure Preservation

Definition 14 (Semantic Correctness)**.** *The policy compiler* Π *is semantically correct if it satisfies:*

- *trigger fidelity,*
- *preservation of semantic neutrality,*
- *absence of spurious action,*
- *non-execution of policy.*

Theorem 4 (Compiler Correctness)**.** *The EGON Policy Compiler is semantically correct.*

Definition 15 (Semantic Dominance)**.** *Let* $(F, \leq)$ *be the partially ordered set of failure classes. If* $f_1 \leq f_2$*, then* f_2 *semantically dominates* f_1*.*

Definition 16 (Structure Preservation)**.** *The compiler preserves semantic structure if domination in* F *is not weakened by compilation.*

Proposition 3 (Monotonicity)**.** *The EGON Policy Compiler is monotone with respect to semantic dominance.*

Definition 17 (Exhaustive Compilation)**.** *A compiler is exhaustive if every element of its domain maps to a defined output.*

Theorem 5 (Exhaustiveness)**.** *The EGON Policy Compiler is exhaustive over* F*.*

Definition 18 (Canonical Serialization)**.** *A serialization function*

$$S : P \to \mathcal{J}$$

maps policy bundles to canonical JSON representations.

Proposition 4 (Serialization Determinism)**.** *For all* $p \in P$*, repeated evaluation of* $S(p)$ *yields identical output.*

6 Examples of Policy Compilation

Neutral Regime. For $f = F_0$, the compiler emits the neutral policy bundle p_0, containing no effective actions.

Recognized but Unmapped Failure. If $f \in F \setminus \{F_0\}$ has no associated policy mapping, the compiler emits an explicit no-op policy annotated to reflect semantic recognition.

Control Plane Saturation. For a failure class corresponding to control plane saturation, the compiler emits a policy bundle containing an ordered sequence of declarative actions such as freezing control-plane mutation and notifying operators.

Guarded Actions. Actions may be annotated with guards imposing approval, rate limits, or execution constraints. Guards do not alter semantic classification.

Deterministic Recompilation. Repeated compilation of any failure class yields bit-for-bit identical serialized output.

7 Discussion and Scope

The EGON Policy Compiler enforces a strict separation between semantic interpretation and operational control. It consumes failure-class semantics and emits declarative policy artifacts without performing execution, inference, optimization, or learning.

This separation enables auditability, reproducibility, and governance. Downstream systems may interpret policy bundles in various operational contexts, but the compiler itself remains agnostic to enforcement.

Non-goals of the compiler include autonomous remediation, feedback-driven modification, and probabilistic reasoning. These exclusions preserve semantic stability and interpretability.

8 Conclusion

The EGON Policy Compiler completes the semantic pipeline introduced by the AEGON Algebra. By compiling finite failure semantics into canonical policy artifacts, the compiler enables deterministic reasoning about system response and supports controlled operational enforcement. Policy is treated as a compiled object rather than an inferred behavior, ensuring transparency, reproducibility, and governance.

A Correspondence Between Algebraic Objects and the EGON Policy IR

Failure classes correspond to the `FailureClass` enumeration. Policy bundles correspond to `PolicyBundle` records. Actions, guards, capability gates, and evidence specifications map directly to their respective IR components. Canonical JSON serialization implements the external representation of policy bundles.

B Proof Sketches and Extensions

Determinism

Determinism follows from finiteness of F and the absence of stateful or probabilistic operations.

Neutrality

Neutrality is preserved by explicit construction of $\Pi(F_0) = p_0$.

Extensions

Possible extensions include richer action algebras, typed evidence specifications, and formal policy composition semantics, all without altering the core compilation map.

References

- A. Diamond, *The AEGON Algebra: A Finite Semantic Framework for Failure Classification*, 2026.
- A. Diamond, *The eCASM Compiler and Algebra*, 2025.
- B. Pierce, *Types and Programming Languages*, MIT Press, 2002.
- S. Mac Lane, *Categories for the Working Mathematician*, Springer, 1971.