Aegis Symbolic Feedback System

Validation Checkpoint Architecture - Post-Automaton Processing Layer

Aegis Engineering Team
OBINexus Computing

June 11, 2025

Executive Summary

This document specifies the post-automaton validation checkpoint architecture within the Aegis symbolic feedback system. Following successful Integration Gate completion of the regex automaton layer, this validation checkpoint ensures deterministic φ predicate assertion, $\psi(t)$ trust window cross-validation, and systematic audit trail propagation for mission-critical CFD deployments.

1 φ Predicate Result Assertion

1.1 Deterministic Match Protocol

The φ predicate result assertion layer provides deterministic validation between φ verdict outcomes and underlying rule encoding to ensure system integrity and traceability.

1.1.1 Predicate Encoding Verification

$$\text{assert\_deterministic\_match}(\varphi_{verdict}, R_{encoding}) = \begin{cases} \text{ASSERT\_PASS} & \text{if } \varphi_{verdict} \equiv encode(R_{encoding} \\ \text{ASSERT\_FAIL} & \text{if } \varphi_{verdict} \not\equiv encode(R_{encoding} \\ \text{ASSERT\_AMBIGUOUS} & \text{if hash\_collision\_detected}(R \end{cases} \tag{1}$$

Where rule encoding verification ensures:

- **Mass Conservation Rule:** $R_{mass} \rightarrow |\nabla \cdot u| < \epsilon_{mass}$
- **Momentum Conservation Rule:** $R_{momentum} \rightarrow \|R_{momentum}\| < \epsilon_{momentum}$
- **Pressure Validation Rule:** $R_{pressure} \rightarrow \nabla p \in \text{ValidPressureSpace}$
- **Boundary Compliance Rule:** $R_{boundary} \rightarrow u|_{\partial\Omega} \equiv u_{prescribed}$

1.1.2 Assertion Verification Algorithm

Require: φ_{result}, $rule_{set}$, $context$
Ensure: Assertion validation status
$encoding_{hash} \leftarrow$ compute_rule_encoding($rule_{set}$)
$verdict_{hash} \leftarrow$ compute_phi_verdict_hash(φ_{result})
if $encoding_{hash} = verdict_{hash}$ **then**
log_assertion_success(φ_{result}, $rule_{set}$, $context$)
return ASSERT_PASS
else if collision_detector($encoding_{hash}$, $verdict_{hash}$) **then**
trigger_collision_protocol(φ_{result}, $rule_{set}$)
return ASSERT_AMBIGUOUS
else
log_assertion_failure(φ_{result}, $rule_{set}$, $context$)
return ASSERT_FAIL
end if

2 $\psi(t)$ Trust Window Cross-Check

2.0.1 Exponential Decay-Weighted Confidence Confirmation

The trust window cross-check validates φ verdict consistency against historical performance using exponentially weighted confidence metrics established during Integration Gate validation.

2.0.2 Trust Window Validation Protocol

Using the approved exponential weighting formula:

$$\psi(t) = \frac{1}{1 + e^{-k(\varphi\_weighted\_success(t) - \theta)}}$$

Where:

$$\varphi\_weighted\_success(t) = \frac{\sum_{i=t-w}^{t} \alpha^{t-i} \cdot \mathbf{1}[\varphi_i = \text{correct}]}{\sum_{i=t-w}^{t} \alpha^{t-i}} \tag{2}$$

$$\alpha = 0.95 \quad \text{(exponential decay factor)} \tag{3}$$

$$\theta = 0.75 \quad \text{(trust threshold)} \tag{4}$$

$$k = 18.0 \quad \text{(sigmoid steepness)} \tag{5}$$

$$w = 1000 \quad \text{(sliding window size)} \tag{6}$$

2.0.3 Cross-Check Validation Matrix

$\psi(t)$ Range	φ Verdict	Historical Consistency	Validation Result
[0.85, 1.0]	PASS	High	TRUST_CONFIRMED
[0.85, 1.0]	FAIL	High	INVESTIGATE_ANOMALY
[0.6, 0.85)	PASS/FAIL	Medium	CONDITIONAL_ACCEPT
[0.3, 0.6)	PASS	Low	ENHANCED_VERIFICATION
[0.3, 0.6)	FAIL	Low	STANDARD_REJECTION
[0.0, 0.3)	*	Very Low	FORCE_FALLBACK

3 Collision Context Scoring

3.1 Hash Distance and Violation Signature Topology

The collision context scoring system quantifies collision severity based on hash distance calculations and violation signature topology analysis for graduated response protocols.

3.1.1 Hash Distance Computation

$$h\_distance(sig_A, sig_B) = \frac{|\text{hash}(sig_A) - \text{hash}(sig_B)|}{2^{64}} \times context\_weight(domain)$$

Context weighting factors:

$$context\_weight(\text{incompressible}) = 1.2 \tag{7}$$

$$context\_weight(\text{compressible}) = 1.0 \tag{8}$$

$$context\_weight(\text{turbulent}) = 0.8 \tag{9}$$

3.1.2 Violation Signature Topology Classification

Require: $state_A$, $state_B$
Ensure: Collision severity classification
  $h\_dist \leftarrow$ compute_hash_distance($state_A$, $state_B$)
  $topology\_diff \leftarrow$ analyze_boundary_topology($state_A$, $state_B$)
  $physics\_consistency \leftarrow$ check_physics_compatibility($state_A$, $state_B$)
  if $h\_dist < 0.15$ AND $topology\_diff$ = MISMATCH **then**
    return SEVERE_COLLISION
  else if $0.15 \leq h\_dist < 0.35$ AND $physics\_consistency$ = INCONSISTENT **then**
    return MODERATE_COLLISION
  else if $0.35 \leq h\_dist < 0.55$ **then**
    return MINOR_COLLISION
  else
    return POTENTIAL_COLLISION
  end if

4 Rejection Branching Logic

4.1 Conditional Fallback States Based on Severity

The rejection branching logic implements graduated response protocols based on collision severity classification and system confidence metrics.

4.1.1 Branching Decision Matrix

$$rejection\_branch(severity, \psi\_value, safety\_context) = \begin{cases} \text{IMMEDIATE\_REJECT} & \text{if } severity = \text{SEV} \\ \text{ENHANCED\_VERIFY} & \text{if } severity = \text{MO} \\ \text{CONDITIONAL\_ACCEPT} & \text{if } severity = \text{MIN} \\ \text{MONITOR\_CONTINUE} & \text{if } severity = \text{PO} \\ \text{EMERGENCY\_PROTOCOL} & \text{if } safety\_contex \end{cases}$$

4.1.2 Severity-Based Rejection Probabilities

$$P(\text{reject}|\text{SEVERE}) = 0.95 \tag{10}$$
$$P(\text{reject}|\text{MODERATE}) = 0.7 \times (1 - \psi(t)) + 0.3 \tag{11}$$
$$P(\text{reject}|\text{MINOR}) = 0.4 \times (1 - \psi(t)) + 0.1 \tag{12}$$
$$P(\text{reject}|\text{POTENTIAL}) = 0.2 \times (1 - \psi(t)) \tag{13}$$

5 Audit Tag Propagation

5.1 Traceable Verdict Paths for Rule Execution

The audit tag propagation system ensures complete traceability of verdict paths through comprehensive logging and metadata attachment for NASA-STD-8739.8 compliance.

5.1.1 Audit Tag Structure

```
AuditTag = {
    verdict_id: UUID,
    timestamp: ISO8601_UTC,
    rule_encoding_hash: SHA256[16],
    phi_predicate_hash: SHA256[16],
    psi_confidence: float[0,1],
    collision_severity: ENUM,
    rejection_probability: float[0,1],
    fallback_triggered: boolean,
    governance_cost: float
}
```

5.1.2 Propagation Protocol

Require: $verdict$, $context$, $parent\_tag$
Ensure: Propagated audit tag

$tag \leftarrow$ create_audit_tag($verdict$, $context$)
tag.parent_reference $\leftarrow$ $parent\_tag$.verdict_id
tag.propagation_depth $\leftarrow$ $parent\_tag$.propagation_depth + 1
validate_audit_completeness(tag)
log_to_audit_trail(tag)
update_governance_metrics(tag.governance_cost)
return tag

6 Runtime Fallback Protocol Hooks

6.1 Emergency and Conditional Fallback Triggers

The runtime fallback protocol hooks define precise conditions for triggering EMERGENCY_FALLBACK() and BYPASS_CONDITIONAL() procedures to maintain system integrity under adverse conditions.

6.1.1 Emergency Fallback Conditions

$$\text{EMERGENCY\_FALLBACK() triggered if: } \begin{cases} \psi(t) < 0.2 & \text{(trust breakdown)} \\ C_{total} > 0.8 & \text{(governance violation)} \\ \text{collision\_count} > threshold_{emergency} & \text{(system instability)} \\ \text{safety\_violation\_detected} = true & \text{(NASA compliance brea} \end{cases}$$

6.1.2 Conditional Bypass Protocol

$$\text{BYPASS\_CONDITIONAL() triggered if: } \begin{cases} 0.3 \leq \psi(t) < 0.6 \wedge \text{performance\_degradation} > 50\% \\ \text{minor\_collision\_rate} > 0.15 \wedge C_{total} < 0.4 \\ \text{trust\_oscillation\_detected} \wedge \text{stable\_fallback\_available} \end{cases}$$

6.1.3 Fallback Protocol Implementation

Require: $system\_state$, $performance\_metrics$
Ensure: System operation mode

$emergency\_conditions \leftarrow$ evaluate_emergency_triggers($system\_state$)
$bypass\_conditions \leftarrow$ evaluate_bypass_triggers($performance\_metrics$)
if $emergency\_conditions$ = TRUE **then**
  trigger_emergency_fallback($system\_state$)
  log_emergency_audit_event("EMERGENCY_FALLBACK_TRIGGERED")
  halt_phi_processing()
  return EMERGENCY_MODE
else if $bypass\_conditions$ = TRUE **then**

```
    initiate_conditional_bypass(performance_metrics)
    log_audit_event("CONDITIONAL_BYPASS_INITIATED")
    return BYPASS_MODE
  else
    return NORMAL_OPERATION
  end if
```

Integration with Aegis SDLC

This validation checkpoint architecture integrates seamlessly with the approved regex automaton layer and maintains compatibility with AEGIS framework principles. The systematic approach to φ predicate assertion, trust window validation, and audit trail propagation ensures production readiness for Release Gate evaluation.

Performance Validation

All validation checkpoint operations maintain computational complexity within established governance bounds:

$$Time\_Complexity(validation\_checkpoint) \leq O(\log w) \tag{14}$$

$$Space\_Complexity(audit\_propagation) \leq O(n) \tag{15}$$

$$Governance\_Cost\_Impact \leq 0.15 \times C_{baseline} \tag{16}$$

NASA Compliance Verification

The validation checkpoint preserves deterministic safety guarantees:

$$\forall \text{ physics violations } V : P(\text{detect\_and\_reject}|V) = 1.0$$

Conclusion

The validation checkpoint architecture provides comprehensive post-automaton verification capabilities while maintaining system performance and safety guarantees. The systematic approach to predicate assertion, trust validation, and audit propagation ensures mission-critical reliability for aerospace CFD deployments.

Document Status: Draft for Release Gate Technical Review