RIFT-SP111 Grammar Traversal System

Model-Agnostic Semantic Processing for RIFT-000 → RIFT-SP111 Pipeline

Executive Summary

This specification defines the mathematical foundation for minimal confidence parsing in the OBINexus RIFT compiler pipeline, bridging tokenized output from RIFT-000 substages to structured semantic parsing in RIFT-SP111. The system operates on concrete symbol matching with stage-bound execution and process-bound semantic resolution, maintaining full model-agnosticism while supporting WYSIWYM principles.

---

1. Pipeline Architecture Overview

1.1 Stage-Process-Phase Nomenclature

RIFT-000 (Tokenization Pipeline):

- **RIFT-000.0**: Lexeme Scanner (raw input → lexical atoms)
- **RIFT-000.1**: Tokenizer Rules (lexemes → preliminary tokens)
- **RIFT-000.2**: Token Type Resolution (typing, tag augmentation, typo fixups)
- **RIFT-000.3**: Token Triplet Builder (final: type, mem_ptr, value)

RIFT-SP111 (Semantic Processing Pipeline):

- **Stage 1**: First semantic processing layer
- **Process 1**: Intent extraction and resolution
- **Phase 1**: Initial grammar traversal and AST construction

1.2 Integration Protocol

```c
```

```c
// RIFT-000 TokenTriplet Output Format
typedef struct TokenTriplet {
    TokenType type;       // From RIFT-000.2 type resolution
    uint32_t mem_ptr;     // Memory position reference
    uint32_t value;       // Semantic value encoding
} TokenTriplet;

// RIFT-SP111 Semantic Token Input Format
typedef struct SP111Token {
    TokenTriplet source;      // Original RIFT-000 triplet
    double confidence;        // Computed ψ(s,r,c) value
    uint32_t row;             // Matrix row position
    uint32_t column;          // Matrix column position
    char* lexeme;             // Raw symbol representation
    void* semantic_hint;      // Process-bound intent annotation
    uint32_t stage_id;        // Stage binding (1)
    uint32_t process_id;      // Process binding (1)
    uint32_t phase_id;        // Phase binding (1)
} SP111Token;
```

---

2. Mathematical Foundation - Updated for SP111

2.1 Symbol Algebra with Process Semantics

Let **Σ** be our alphabet of symbols, partitioned into process-bound semantic classes:

```
Σ = Σ_term ∪ Σ_struct ∪ Σ_query ∪ Σ_close ∪ Σ_process
```

Where:

- **Σ_term** = Terminal symbols (identifiers, literals, operators)
- **Σ_struct** = Structural delimiters (parentheses, brackets, braces)
- **Σ_query** = Semantic query symbols (`?`, conditional expressions)
- **Σ_close** = Statement closure symbols (`.`, `;`, line terminators)
- **Σ_process** = Process-bound control symbols (stage transitions, phase markers)

2.2 Enhanced Confidence Metric Function

For any symbol **s** ∈ **Σ** at position **(r,c)** with stage-process binding **sp**:

```
ψ(s, r, c, sp) = α·κ(s) + β·ρ(r,c) + γ·τ(s) + δ·σ(sp)
```

Where:

- **κ(s)** = Symbol lexical confidence [0,1] (from RIFT-000 pipeline)
- **ρ(r,c)** = Positional context confidence [0,1]
- **τ(s)** = Type consistency confidence [0,1]
- **σ(sp)** = Stage-process binding confidence [0,1] (new for SP111)
- **α, β, γ, δ** = Weighting coefficients (α + β + γ + δ = 1)

2.3 Process-Bound Matrix Representation

Input stream organized as semantic matrix **M[R×C×P]** where **P** represents process depth:

```
M[R×C×P] = [
  Process 1: [
    [s_111, s_121, ..., s_1C_1],  ← Row 1, Process 1
    [s_211, s_221, ..., s_2C_1],  ← Row 2, Process 1
    [⋮    , ⋮    , ⋱  , ⋮    ]
  ],
  Process P: [
    [s_11p, s_12p, ..., s_1C_p],  ← Row 1, Process P
    [s_21p, s_22p, ..., s_2C_p],  ← Row 2, Process P
    [⋮    , ⋮    , ⋱  , ⋮    ]
  ]
]
```

Enhanced Semantic Interpretation:

- **Rows (r)**: Linear statement flow, temporal sequence
- **Columns (c)**: Structural depth, nesting level
- **Process (p)**: Stage-bound execution context, semantic boundaries

---

3. SP111 Traversal Algorithm Specification

3.1 Stage-Bound Core Traversal Function

```
traverse_sp111_matrix(M, θ_min, stage_config) → AST_forest
  Input: Process matrix M[R×C×P], confidence threshold θ_min,
         stage configuration from gov.riftrc.111.xml
  Output: List of validated AST nodes with stage binding
```

Algorithm Steps:

1. **Stage Initialization**: Load configuration from `gov.riftrc.111.xml`
   - Parse stage-specific confidence thresholds

- Initialize process-bound semantic gates
- Set up AEGIS framework compliance metrics

2. **Process-wise Primary Scan**: `for p = 1 to P:`
   - Compute process confidence: $\Psi_p = \sum_{r,c} \psi(M[r,c,p], r, c, sp) / (R \times C)$
   - If $\Psi_p < \theta\_min$: Flag process layer for secondary analysis
3. **Row-Column-Process Traversal**: `for each (r,c,p):`
   - Apply enhanced confidence function with stage binding
   - Execute semantic intent resolution with process context
4. **Stage-Bound AST Construction**:
   - Build AST nodes with SP111 metadata
   - Apply isomorphic reduction via Myhill-Nerode equivalence
   - Generate serialization-ready formats (.rift.ast.json, .rift.astb)

3.2 Process-Bound Semantic Intent Resolution

```c
// Enhanced intent resolution for SP111
typedef enum SemanticIntent {
    SP111_INTENT_DECLARE = 0x0100,   // Stage 1 declaration intent
    SP111_INTENT_ASSIGN = 0x0101,    // Stage 1 assignment intent
    SP111_INTENT_CONTROL = 0x0102,   // Stage 1 control flow intent
    SP111_INTENT_INVOKE = 0x0103,    // Stage 1 invocation intent
    SP111_INTENT_QUERY = 0x0104,     // Stage 1 conditional intent
    SP111_INTENT_TERMINATE = 0x0105, // Stage 1 termination intent
    SP111_INTENT_PROCESS = 0x0106    // Process-bound transition intent
} SP111SemanticIntent;
```

Intent Resolution Algorithm:

```
resolve_sp111_intent(token, process_context) → SP111SemanticIntent:
  1. Extract stage-process binding from token.stage_id, token.process_id
  2. Apply process-bound semantic gates defined in gov.riftrc.111.xml
  3. Resolve intent using enhanced confidence metrics
  4. Validate against stage-specific production rules
  5. Return stage-bound semantic intent with process metadata
```

---

4. Integration with OBINexus Toolchain

4.1 AEGIS Framework Compliance - SP111

Configuration Management:

```xml
<!-- gov.riftrc.111.xml example structure -->
<riftconfig stage="1" process="1" phase="1">
  <confidence_thresholds>
    <alpha>0.4</alpha>
    <beta>0.3</beta>
    <gamma>0.2</gamma>
    <delta>0.1</delta>  <!-- Stage-process binding weight -->
  </confidence_thresholds>
  <semantic_gates>
    <intent_class name="DECLARE" threshold="0.85"/>
    <intent_class name="ASSIGN" threshold="0.80"/>
    <intent_class name="QUERY" threshold="0.75"/>
  </semantic_gates>
  <process_bindings>
    <process id="1" description="Intent extraction and resolution"/>
  </process_bindings>
</riftconfig>
```

4.2 Pipeline Bridge Protocol - RIFT-000 → RIFT-SP111

```c
```

```c
typedef struct SP111BridgeContext {
    TokenTriplet* input_triplets;     // From RIFT-000.3
    size_t triplet_count;
    SP111Token* semantic_tokens;      // Converted for SP111 processing
    size_t token_count;
    uint32_t stage_id;                // Always 1 for SP111
    uint32_t process_id;              // Process binding identifier
    uint32_t phase_id;                // Phase execution context
    void* aegis_context;              // AEGIS framework integration
} SP111BridgeContext;

// Bridge function implementation
SP111BridgeContext* bridge_000_to_sp111(
    TokenTriplet* triplets,
    size_t count,
    const char* config_path  // Path to gov.riftrc.111.xml
) {
    SP111BridgeContext* ctx = calloc(1, sizeof(SP111BridgeContext));
    ctx->input_triplets = triplets;
    ctx->triplet_count = count;
    ctx->stage_id = 1;
    ctx->process_id = 1;
    ctx->phase_id = 1;

    // Convert triplets to SP111 semantic tokens
    ctx->semantic_tokens = convert_triplets_to_sp111(triplets, count);
    ctx->token_count = count;

    // Load stage configuration
    load_sp111_configuration(ctx, config_path);

    return ctx;
}
```

4.3 AST Output Specification - SP111 Format

```
c
```

```c
typedef struct SP111ASTNode {
    NodeType type;                    // TERMINAL, NONTERMINAL, PROCESS_BOUND
    SP111SemanticIntent intent;       // Process-bound resolved intent
    double aggregate_confidence;      // Combined ψ for subtree with stage binding
    struct SP111ASTNode** children;   // Child node array
    SP111Token* source_token;         // Original token reference
    uint32_t stage_id;                // Stage binding (1)
    uint32_t process_id;              // Process binding (1)
    uint32_t phase_id;                // Phase binding (1)
    void* aegis_metadata;             // AEGIS compliance tracking
} SP111ASTNode;
```

---

5. Performance Characteristics & Complexity

5.1 Enhanced Time Complexity with Process Binding

- **Matrix Traversal**: O(R × C × P) where R = rows, C = columns, P = processes
- **Confidence Computation**: O(|Σ| × |SP|) per symbol where SP = stage-process combinations
- **Semantic Resolution**: O(log|I|) where I = SP111-specific intent classes
- **Overall Pipeline**: O(R × C × P × log|I|)

5.2 Stage-Process Memory Overhead

- **Token Matrix**: O(R × C × P)
- **AST Forest with SP111 Metadata**: O(N × M) where N = nodes, M = metadata size
- **Configuration Cache**: O(|Config| × |Stages|)

---

6. Testing & Validation Framework

6.1 Stage-Process Coverage Testing

```python
```

```python
def test_sp111_confidence_thresholds():
    """Test SP111-specific confidence threshold performance"""
    test_cases = [
        # (α, β, γ, δ, expected_precision)
        (0.4, 0.3, 0.2, 0.1, 0.95),  # Balanced weighting
        (0.5, 0.2, 0.2, 0.1, 0.93),  # Lexical emphasis
        (0.3, 0.4, 0.2, 0.1, 0.97),  # Positional emphasis
        (0.3, 0.3, 0.3, 0.1, 0.94),  # Type consistency emphasis
    ]

    for alpha, beta, gamma, delta, expected in test_cases:
        precision = measure_sp111_parsing_precision(alpha, beta, gamma, delta)
        assert precision >= expected, f"SP111 precision below threshold: {precision}"

def test_stage_process_binding():
    """Validate stage-process-phase binding correctness"""
    tokens = generate_test_sp111_tokens()

    for token in tokens:
        assert token.stage_id == 1, "Invalid stage binding"
        assert token.process_id == 1, "Invalid process binding"
        assert token.phase_id == 1, "Invalid phase binding"

    # Test intent resolution with process context
    intents = resolve_all_sp111_intents(tokens)
    for intent in intents:
        assert (intent & 0xFF00) == 0x0100, "Invalid SP111 intent classification"
```

6.2 Integration Testing with RIFT-000

```python
```

```python
def test_000_to_sp111_bridge():
    """Test complete pipeline bridge functionality"""
    # Generate RIFT-000 token triplets
    triplets = simulate_rift_000_output([
        "int sum = x + 42;",
        "if (condition) { return true; }",
        "function process() { ... }"
    ])

    # Bridge to SP111
    bridge_ctx = bridge_000_to_sp111(triplets, "test_configs/gov.riftrc.111.xml")

    # Validate transformation
    assert bridge_ctx.semantic_tokens != None, "SP111 token conversion failed"
    assert bridge_ctx.stage_id == 1, "Incorrect stage binding"
    assert bridge_ctx.process_id == 1, "Incorrect process binding"

    # Execute SP111 traversal
    ast_forest = traverse_sp111_matrix(
        bridge_ctx.semantic_tokens,
        threshold=0.8,
        config=bridge_ctx.config
    )

    # Validate AST output
    validate_sp111_ast_forest(ast_forest)
```

---

7. Migration Path from Legacy Architecture

7.1 Systematic Migration Protocol

Phase 1: Nomenclature Updates

- Update all RIFT-0 references to RIFT-000
- Update all RIFT-1 references to RIFT-SP111
- Migrate configuration files to gov.riftrc.111.xml format

Phase 2: Enhanced Data Structures

- Extend TokenTriplet → SP111Token conversion
- Implement stage-process-phase binding metadata
- Add enhanced confidence metrics with δ parameter

Phase 3: Integration Testing

- Validate RIFT-000 → RIFT-SP111 bridge functionality
- Ensure AEGIS framework compliance
- Verify performance characteristics within acceptable bounds

7.2 Backward Compatibility Strategy

```c
// Legacy compatibility layer
#define RIFT_0_TOKEN TokenTriplet
#define RIFT_1_TOKEN SP111Token
#define rift_0_to_1_bridge bridge_000_to_sp111

// Configuration migration utility
void migrate_legacy_config(const char* old_path, const char* new_path) {
    // Convert legacy gov.riftrc.0.xml to gov.riftrc.111.xml format
    // Preserve existing confidence parameters
    // Add default stage-process-phase bindings
}
```

---

8. Future Enhancements - Multi-Stage Architecture

8.1 Extended Pipeline Support

- **RIFT-SP112**: Stage 1, Process 1, Phase 2 (advanced semantic analysis)
- **RIFT-SP211**: Stage 2, Process 1, Phase 1 (optimization stage)
- **RIFT-SP221**: Stage 2, Process 2, Phase 1 (code generation)

8.2 Dynamic Stage-Process Binding

```c
typedef struct DynamicSPBinding {
    uint32_t stage_range[2];    // [min_stage, max_stage]
    uint32_t process_range[2];  // [min_process, max_process]
    uint32_t phase_range[2];    // [min_phase, max_phase]
    double binding_confidence;  // Confidence in SP binding
} DynamicSPBinding;
```

This updated specification provides the technical foundation for RIFT-SP111 implementation, maintaining mathematical rigor while incorporating the enhanced stage-process-phase architecture. The specification is ready for deployment to the github.com/obinexus/rift-SP111 repository.