STUNIR v0.9.0 Gap Analysis Report

Current Version: v0.8.5
Target Version: v0.9.0 (“Everything-but-Haskell Working”)
Analysis Date: February 1, 2026
Analyst: STUNIR Team

Executive Summary

This document provides a comprehensive gap analysis for achieving STUNIR v0.9.0 milestone, defined as “everything-but-Haskell working”. The analysis covers all three production pipelines (Python, Rust, SPARK) and identifies gaps across IR features, code generation, testing, documentation, and tooling.

Key Findings

- **Current Completion**: ~75% towards v0.9.0 goal
- **Estimated Effort**: 8-10 weeks (40-50 person-days)
- **Critical Gaps**: Test coverage (Rust/SPARK), advanced IR features, optimization passes
- **Risk Level**: **MEDIUM** - Core functionality complete, but testing and edge cases need work

---

1. Current State Analysis (v0.8.5)

1.1 Pipeline Status

Pipeline	spec_to_ir	ir_to_code	Control Flow	Function Bodies	Multi-File	Status
Python	✅ Complete	✅ Complete	✅ Complete	✅ Complete	✅ Complete	**100%**
Rust	✅ Complete	✅ Complete	✅ Complete	✅ Complete	✅ Complete	**100%**
SPARK	✅ Complete	✅ Complete	✅ Complete	✅ Complete	✅ Complete	**100%**

Features Implemented (v0.8.5):
- ✅ Basic control flow (if/while/for)
- ✅ Break/continue statements
- ✅ Switch/case statements
- ✅ Function bodies with type inference
- ✅ Multi-file specification support
- ✅ Local variable tracking
- ✅ Basic expressions and operations

1.2 Emitter Coverage

Category	Python	Rust	SPARK	Status
Total Emitters	41	32	82	✅
Core Emitters	5	5	5	✅
Assembly	2	2	2	✅
Polyglot (C89/C99/Rust)	3	3	3	✅
Lisp Family	8	7	8	⚠️
Prolog Family	8	2	9	⚠️
Specialized	15	13	55	⚠️

Analysis: SPARK has the most comprehensive emitter coverage (82 files) due to DO-178C Level A formal verification requirements. Python and Rust emitters are functional but may lack formal verification.

1.3 Test Coverage

Pipeline	Unit Tests	Integration Tests	Coverage %	Status
Python	97	~10	8.53%	❌ Insufficient
Rust	0	0	0%	❌ Critical Gap
SPARK	0 (manual)	6	N/A	⚠️ Manual Only
Overall	97	~16	8.53%	❌ Insufficient

Analysis: Test coverage is critically low. While core functionality works, extensive testing is needed for production readiness.

1.4 Documentation Status

Category	Files	Completeness	Status
Total Docs	200	~80%	⚠️
API Reference	15	~90%	✅
User Guides	8	~60%	⚠️
Developer Guides	12	~70%	⚠️
Tutorials	2	~40%	❌
Architecture Docs	10	~85%	✅

Analysis: Core documentation exists but lacks comprehensive tutorials and practical examples.

---

2. Gap Identification for v0.9.0

2.1 CRITICAL GAPS (Must-Fix for v0.9.0)

GAP-001: Test Coverage - Rust Pipeline [CRITICAL]

- **Description**: Rust pipeline has 0 tests despite full feature implementation
- **Impact**: High risk of regressions, cannot verify correctness
- **Priority**: P0 (Critical)
- **Effort**: 2 weeks (10 person-days)
- **Dependencies**: None
- **Acceptance Criteria**:
- [ ] 50+ unit tests for Rust spec_to_ir
- [ ] 50+ unit tests for Rust ir_to_code
- [ ] 10+ integration tests for full pipeline
- [ ] Achieve >60% code coverage
- [ ] All tests passing in CI/CD

GAP-002: Test Coverage - SPARK Pipeline [CRITICAL]

- **Description**: SPARK pipeline has only manual tests, no automated test suite
- **Impact**: Cannot verify formal verification claims, manual testing is error-prone
- **Priority**: P0 (Critical)
- **Effort**: 2 weeks (10 person-days)
- **Dependencies**: None
- **Acceptance Criteria**:
- [ ] 50+ automated tests for SPARK tools
- [ ] Integration with GNATprove verification
- [ ] Automated proof checking in CI/CD
- [ ] Regression test suite

- [ ] Performance baseline tests

GAP-003: Exception Handling IR Support [CRITICAL]

- **Description**: No IR schema or implementation for try/catch/finally
- **Impact**: Cannot generate code for error handling patterns
- **Priority**: P0 (Critical)
- **Effort**: 3 weeks (15 person-days)
- **Dependencies**: IR schema update, all 3 pipelines
- **Acceptance Criteria**:
- [ ] IR schema extended with exception operations
- [ ] Python implementation complete
- [ ] Rust implementation complete
- [ ] SPARK implementation complete
- [ ] Test coverage >60%
- [ ] Documentation updated

GAP-004: Integration Testing Framework [CRITICAL]

- **Description**: No comprehensive cross-pipeline validation
- **Impact**: Cannot verify IR determinism across pipelines
- **Priority**: P0 (Critical)
- **Effort**: 1.5 weeks (7 person-days)
- **Dependencies**: GAP-001, GAP-002
- **Acceptance Criteria**:
- [ ] Cross-pipeline IR comparison tests
- [ ] Determinism validation suite
- [ ] Hash verification tests
- [ ] Performance comparison framework
- [ ] Automated regression detection

2.2 HIGH PRIORITY GAPS (Should-Fix for v0.9.0)

GAP-005: Advanced Data Structures [HIGH]

- **Description**: Limited support for arrays, maps, sets, and custom structures
- **Impact**: Cannot generate code for complex data patterns
- **Priority**: P1 (High)
- **Effort**: 2 weeks (10 person-days)
- **Dependencies**: IR schema update
- **Acceptance Criteria**:
- [ ] Dynamic arrays (vector/list)
- [ ] Hash maps (dictionary/HashMap)
- [ ] Sets (HashSet)
- [ ] Nested structures
- [ ] Struct initialization
- [ ] Collection operations (add, remove, iterate)

GAP-006: Generic/Template Support [HIGH]

- **Description**: Limited support for generics and templates
- **Impact**: Cannot generate type-parameterized code

- **Priority**: P1 (High)
- **Effort**: 2 weeks (10 person-days)
- **Dependencies**: Type system enhancement
- **Acceptance Criteria**:
- [ ] Generic function definitions
- [ ] Template instantiation
- [ ] Type parameter constraints
- [ ] Monomorphization for C targets
- [ ] Rust-style trait bounds

GAP-007: Optimization Pass Framework [HIGH]

- **Description**: No optimization infrastructure for IR or code generation
- **Impact**: Generated code is unoptimized, performance poor
- **Priority**: P1 (High)
- **Effort**: 2 weeks (10 person-days)
- **Dependencies**: IR infrastructure
- **Acceptance Criteria**:
- [ ] Constant folding
- [ ] Dead code elimination
- [ ] Common subexpression elimination
- [ ] Loop optimization (unrolling, invariant hoisting)
- [ ] Inlining small functions
- [ ] Register allocation hints

GAP-008: Debug Information Generation [HIGH]

- **Description**: No debug info (DWARF, PDB) generation for targets
- **Impact**: Generated code is hard to debug
- **Priority**: P1 (High)
- **Effort**: 1.5 weeks (7 person-days)
- **Dependencies**: Code generation refactor
- **Acceptance Criteria**:
- [ ] DWARF debug info for C/C++/Rust targets
- [ ] Source line mapping
- [ ] Variable name preservation
- [ ] Type information in debug symbols
- [ ] Debugger integration validated

GAP-009: Comprehensive User Guide [HIGH]

- **Description**: User guide is incomplete, lacks practical examples
- **Impact**: Hard to onboard new users
- **Priority**: P1 (High)
- **Effort**: 1 week (5 person-days)
- **Dependencies**: None
- **Acceptance Criteria**:
- [ ] Getting started tutorial
- [ ] 10+ practical examples

- [ ] Common patterns guide
- [ ] Troubleshooting guide
- [ ] FAQ section
- [ ] Video tutorials (optional)

2.3 MEDIUM PRIORITY GAPS (Nice-to-Have for v0.9.0)

GAP-010: Function Pointers and Callbacks [MEDIUM]

- **Description**: Basic implementation exists but lacks advanced features
- **Impact**: Cannot generate code for event-driven patterns
- **Priority**: P2 (Medium)
- **Effort**: 1 week (5 person-days)
- **Dependencies**: Type system enhancement
- **Acceptance Criteria**:
- [ ] Function pointer type definitions
- [ ] Callback registration
- [ ] Higher-order functions
- [ ] Closure support (where applicable)

GAP-011: Async/Await Patterns [MEDIUM]

- **Description**: No support for asynchronous programming patterns
- **Impact**: Cannot generate modern async code
- **Priority**: P2 (Medium)
- **Effort**: 2 weeks (10 person-days)
- **Dependencies**: IR schema major update
- **Acceptance Criteria**:
- [ ] Async function definitions
- [ ] Await expressions
- [ ] Future/Promise types
- [ ] Async runtime integration (Tokio, async-std)
- [ ] Error handling in async context

GAP-012: Target-Specific Optimizations [MEDIUM]

- **Description**: No architecture-specific code generation
- **Impact**: Suboptimal code for specific targets
- **Priority**: P2 (Medium)
- **Effort**: 1.5 weeks (7 person-days)
- **Dependencies**: GAP-007
- **Acceptance Criteria**:
- [ ] SIMD instruction generation
- [ ] Cache-aware code layout
- [ ] Platform-specific intrinsics
- [ ] ARM NEON support
- [ ] x86 AVX support

GAP-013: CI/CD Pipeline Enhancement [MEDIUM]

- **Description**: Basic CI/CD exists but lacks comprehensive automation
- **Impact**: Manual steps still required for releases

- **Priority**: P2 (Medium)
- **Effort**: 1 week (5 person-days)
- **Dependencies**: GAP-001, GAP-002
- **Acceptance Criteria**:
- [ ] Automated testing on commit
- [ ] Multi-platform builds (Linux, macOS, Windows)
- [ ] Automated release creation
- [ ] Benchmark regression detection
- [ ] Security scanning integration

GAP-014: Performance Benchmarking Suite [MEDIUM]

- **Description**: No systematic performance testing
- **Impact**: Cannot track performance regressions
- **Priority**: P2 (Medium)
- **Effort**: 1 week (5 person-days)
- **Dependencies**: None
- **Acceptance Criteria**:
- [ ] Benchmark suite for spec_to_ir
- [ ] Benchmark suite for ir_to_code
- [ ] Cross-pipeline performance comparison
- [ ] Memory usage tracking
- [ ] Compilation time tracking
- [ ] CI/CD integration

2.4 LOW PRIORITY GAPS (Future Work)

GAP-015: IDE Integration [LOW]

- **Description**: No IDE plugins for STUNIR development
- **Impact**: Developer experience could be better
- **Priority**: P3 (Low)
- **Effort**: 2 weeks (10 person-days)
- **Dependencies**: Language server protocol
- **Deferred to**: v1.0.0

GAP-016: Package Manager Integration [LOW]

- **Description**: Limited integration with package managers
- **Impact**: Distribution could be easier
- **Priority**: P3 (Low)
- **Effort**: 1 week (5 person-days)
- **Dependencies**: None
- **Deferred to**: v1.0.0

GAP-017: Cross-Language Interop Testing [LOW]

- **Description**: No tests for calling code across language boundaries
- **Impact**: Cannot verify FFI correctness
- **Priority**: P3 (Low)
- **Effort**: 1.5 weeks (7 person-days)
- **Dependencies**: GAP-004

- **Deferred to**: v1.0.0

---

3. Known Issues and Bugs

3.1 Active Bugs

ID	Description	Impact	Priority	Status
BUG-001	Python variable redeclaration in nested blocks	Low	P2	Open
BUG-002	SPARK stack overflow with >5 nesting levels	Medium	P1	Workaround exists
BUG-003	Limited error messages in SPARK tools	Low	P2	Open
BUG-004	Type inference fails for complex expressions	Medium	P1	Open

3.2 Performance Issues

ID	Description	Impact	Priority	Status
PERF-001	Python pipeline 3x slower than SPARK	Medium	P2	Expected
PERF-002	Large IR files cause memory issues	Low	P3	Open
PERF-003	No incremental compilation	Medium	P2	Open

---

4. Effort Estimation Summary

4.1 By Priority

Priority	Gaps	Total Effort	Person-Weeks
P0 (Critical)	4	35 days	7 weeks
P1 (High)	5	44 days	8.8 weeks
P2 (Medium)	5	33 days	6.6 weeks
P3 (Low)	3	22 days	4.4 weeks
Total	17	134 days	26.8 weeks

4.2 Realistic Path to v0.9.0

Focused Scope (P0 + P1 only):
- **Total Effort**: 79 days (15.8 weeks)
- **With 2 developers**: ~8 weeks
- **With 3 developers**: ~5.3 weeks

Recommended: Focus on P0 and P1 gaps for v0.9.0, defer P2/P3 to v0.9.x or v1.0.0.

---

5. Dependencies and Risk Analysis

5.1 Critical Path

```
GAP-001 (Rust Tests) ──┐
GAP-002 (SPARK Tests) ─┼──→ GAP-004 (Integration Testing) ──→ v0.9.0 Release
GAP-003 (Exceptions) ───┘
```

5.2 Risk Assessment

Risk	Probability	Impact	Mitigation
Test infrastructure delays	Medium	High	Start early, dedicate resources
Exception handling complexity	Medium	High	Prototype in Python first
SPARK formal verific-ation challenges	Medium	Medium	Leverage existing patterns
Scope creep	High	High	Strict prioritization, defer P2/P3
Resource constraints	Medium	High	Focus on P0/P1 only

6. Recommendations

6.1 Immediate Actions (Week 1-2)

1. **Start Rust test suite** (GAP-001) - Critical blocker
2. **Start SPARK test suite** (GAP-002) - Critical blocker
3. **Design exception handling IR schema** (GAP-003) - Required for completeness
4. **Set up integration testing framework** (GAP-004) - Foundation for quality

6.2 Medium-Term Actions (Week 3-6)

1. **Implement exception handling** across all pipelines (GAP-003)
2. **Add advanced data structures** (GAP-005)
3. **Implement generic/template support** (GAP-006)
4. **Build optimization framework** (GAP-007)
5. **Add debug info generation** (GAP-008)

6.3 Final Polish (Week 7-8)

1. **Complete user guide** (GAP-009)
2. **Fix all P0/P1 bugs**
3. **Run comprehensive testing**
4. **Performance validation**
5. **Documentation review**

6.4 Defer to Future Releases

- **P2 gaps**: Consider for v0.9.1-0.9.5
- **P3 gaps**: Defer to v1.0.0
- **Async/await**: Complex feature, defer to v1.0.0
- **IDE integration**: Not critical for core functionality

7. Success Criteria for v0.9.0

7.1 Functional Criteria

- [ ] All 3 pipelines (Python, Rust, SPARK) achieve 100% feature parity
- [ ] Exception handling working in all pipelines
- [ ] Advanced data structures support complete
- [ ] Generic/template support working
- [ ] All P0 and P1 gaps resolved

7.2 Quality Criteria

- [ ] Test coverage >60% for all pipelines
- [ ] All unit tests passing
- [ ] All integration tests passing
- [ ] Cross-pipeline determinism validated
- [ ] No P0 or P1 bugs outstanding

7.3 Documentation Criteria

- [ ] User guide complete with 10+ examples
- [ ] API reference updated
- [ ] Migration guide for v0.8.x → v0.9.0
- [ ] Troubleshooting guide complete

7.4 Performance Criteria

- [ ] Benchmark suite established
- [ ] Performance baseline documented
- [ ] No major performance regressions

---

8. Version Roadmap

Incremental Releases to v0.9.0

Based on the gap analysis, here’s the proposed incremental roadmap:

v0.8.6 (Week 1-2) - “Test Infrastructure”

Focus: Establish comprehensive testing
- GAP-001: Rust test suite (50+ tests)
- GAP-002: SPARK test suite (50+ tests)
- GAP-004: Integration testing framework
- **Deliverable**: Test coverage >30%

v0.8.7 (Week 3-4) - “Exception Handling”

Focus: Complete exception support
- GAP-003: Exception handling IR and implementation
- Exception tests for all 3 pipelines
- Documentation updates
- **Deliverable**: Exception handling working

v0.8.8 (Week 5) - “Advanced Data Structures”

Focus: Rich data structure support
- GAP-005: Arrays, maps, sets implementation
- Collection operation tests
- Performance validation
- **Deliverable**: Advanced data structures working

v0.8.9 (Week 6-7) - “Generics and Optimization”

Focus: Type system and performance
- GAP-006: Generic/template support
- GAP-007: Optimization pass framework
- GAP-008: Debug info generation
- **Deliverable**: Optimized code generation

v0.9.0-rc1 (Week 8) - “Release Candidate”

Focus: Final polish and validation
- GAP-009: Complete user guide
- Bug fixes (BUG-001, BUG-002, BUG-004)
- Comprehensive testing
- Performance benchmarking
- **Deliverable**: Release candidate

v0.9.0 (End of Week 8) - “Everything-but-Haskell Working”

Focus: Production release
- All P0/P1 gaps resolved
- All success criteria met
- Documentation complete
- Release notes published
- **Deliverable**: Production-ready v0.9.0

---

9. Conclusion

STUNIR v0.8.5 has achieved significant milestones with all 3 pipelines supporting control flow and function bodies. The path to v0.9.0 requires focused effort on:

1. **Testing** - Comprehensive test coverage for Rust and SPARK
2. **Exception Handling** - Essential for real-world code generation
3. **Advanced Features** - Data structures, generics, optimization
4. **Documentation** - Complete user guides and examples

Estimated Timeline: 8 weeks with focused effort on P0/P1 gaps

Risk Level: MEDIUM - Core functionality proven, quality and edge cases need work

Recommendation: Proceed with incremental releases (v0.8.6 → v0.8.9 → v0.9.0) to validate progress and manage risk.

Report Version: 1.0
Next Review: After v0.8.6 completion
Contact: STUNIR Core Team