SPARK Implementation Roadmap for v0.9.0 Features

Date: 2026-02-01
Target Version: v0.9.1
Features: break, continue, switch/case statements
Estimated Effort: 2-4 hours

Overview

This document provides a detailed roadmap for implementing v0.9.0 control flow features (break, continue, switch/case) in the SPARK pipeline. The implementation must maintain SPARK's formal verification guarantees and bounded type safety.

Prerequisites

Required Knowledge

- Ada SPARK language and SPARK Mode annotations
- Bounded types and static array sizes
- SPARK proof contracts (Pre/Post conditions)
- GNAT compiler toolchain
- JSON parsing in Ada with `STUNIR_JSON_Utils`

Code Locations

- **Spec to IR:** `tools/spark/src/stunir_spec_to_ir.adb`
- **IR to Code:** `tools/spark/src/stunir_ir_to_code.adb`
- **Type Definitions:** `tools/spark/src/semantic_ir/semantic_ir.ads`
- **Build Config:** `tools/spark/stunir_tools.gpr`

Implementation Plan

Phase 1: Type System Extensions (30-45 minutes)

1.1 Update IR Step Enumeration

File: `tools/spark/src/semantic_ir/semantic_ir.ads`

Current (estimated):

```
type IR_Step_Op is (
    Op_Assign,
    Op_Call,
    Op_Return,
    Op_If,
    Op_While,
    Op_For,
    Op_Nop
);
```

Add:

```
type IR_Step_Op is (
    Op_Assign,
    Op_Call,
    Op_Return,
    Op_If,
    Op_While,
    Op_For,
    Op_Break,      -- v0.9.0
    Op_Continue,   -- v0.9.0
    Op_Switch,     -- v0.9.0
    Op_Nop
);
```

1.2 Define Switch Case Structure

Add to semantic_ir.ads:

```ada
-- Maximum number of cases in a switch statement
Max_Switch_Cases : constant := 64;

type IR_Case_Value is record
    Is_Integer : Boolean := True;
    Int_Value  : Integer := 0;
    Str_Value  : Bounded_String := Null_Bounded_String;
end record;

type IR_Step_Array is array (Positive range <>) of IR_Step;
type IR_Steps_Bounded is new IR_Step_Array (1 .. Max_Nested_Steps);

type IR_Switch_Case is record
    Value       : IR_Case_Value;
    Body_Steps  : IR_Steps_Bounded;
    Body_Count  : Natural := 0;
end record;

type IR_Switch_Cases is array (1 .. Max_Switch_Cases) of IR_Switch_Case;

-- Extend IR_Step record
type IR_Step is record
    Op          : IR_Step_Op := Op_Nop;
    Target      : Bounded_String := Null_Bounded_String;
    Value       : Bounded_String := Null_Bounded_String;

    -- Control flow fields (existing)
    Condition   : Bounded_String := Null_Bounded_String;
    Then_Block  : IR_Steps_Bounded;
    Else_Block  : IR_Steps_Bounded;
    Body        : IR_Steps_Bounded;
    Init        : Bounded_String := Null_Bounded_String;
    Increment   : Bounded_String := Null_Bounded_String;

    -- v0.9.0: Switch/case fields
    Switch_Expr : Bounded_String := Null_Bounded_String;
    Cases       : IR_Switch_Cases;
    Cases_Count : Natural := 0;
    Default     : IR_Steps_Bounded;
    Has_Default : Boolean := False;
end record;
```

Note: Exact field names may differ based on existing SPARK implementation.

Phase 2: Spec to IR Parsing (45-60 minutes)

2.1 Add Break Statement Parsing

File: `tools/spark/src/stunir_spec_to_ir.adb`

Location: In the statement parsing procedure (likely in `Parse_Function_Body` or similar)

Add:

```
elsif Stmt_Type = "break" then
    -- v0.9.0: Break statement
    declare
        Step : IR_Step;
    begin
        Step.Op := Op_Break;
        -- Add to steps array
        Steps (Step_Count + 1) := Step;
        Step_Count := Step_Count + 1;
    end;
```

2.2 Add Continue Statement Parsing

Add:

```
elsif Stmt_Type = "continue" then
    -- v0.9.0: Continue statement
    declare
        Step : IR_Step;
    begin
        Step.Op := Op_Continue;
        -- Add to steps array
        Steps (Step_Count + 1) := Step;
        Step_Count := Step_Count + 1;
    end;
```

2.3 Add Switch Statement Parsing

Add (more complex):

```
elsif Stmt_Type = "switch" then
    -- v0.9.0: Switch/case statement
    declare
        Step : IR_Step;
        Switch_Expr : constant String := Get_JSON_String (Stmt_Obj, "expr");
        Cases_Array : JSON_Value := Get (Stmt_Obj, "cases");
        Case_Count  : Natural := 0;
    begin
        Step.Op := Op_Switch;
        Step.Switch_Expr := To_Bounded_String (Switch_Expr);

        -- Parse cases
        if not Is_Empty (Cases_Array) then
            for I in 1 .. Length (Cases_Array) loop
                exit when Case_Count >= Max_Switch_Cases;

                declare
                    Case_Obj  : constant JSON_Value := Get (Cases_Array, I);
                    Case_Val  : constant JSON_Value := Get (Case_Obj, "value");
                    Case_Body : constant JSON_Value := Get (Case_Obj, "body");
                    IR_Case   : IR_Switch_Case;
                begin
                    -- Parse case value
                    if Kind (Case_Val) = JSON_Int_Type then
                        IR_Case.Value.Is_Integer := True;
                        IR_Case.Value.Int_Value := Get (Case_Val);
                    else
                        IR_Case.Value.Is_Integer := False;
                        IR_Case.Value.Str_Value := To_Bounded_String (Get (Case_Val));
                    end if;

                    -- Parse case body (recursive)
                    Parse_Statements (Case_Body, IR_Case.Body_Steps,
IR_Case.Body_Count);

                    Case_Count := Case_Count + 1;
                    Step.Cases (Case_Count) := IR_Case;
                end;
            end loop;
        end if;

        Step.Cases_Count := Case_Count;

        -- Parse default case if present
        if Has_Field (Stmt_Obj, "default") then
            declare
                Default_Body : constant JSON_Value := Get (Stmt_Obj, "default");
                Default_Count : Natural;
            begin
                Parse_Statements (Default_Body, Step.Default, Default_Count);
                Step.Has_Default := True;
            end;
        end if;

        -- Add to steps array
        Steps (Step_Count + 1) := Step;
        Step_Count := Step_Count + 1;
    end;
```

Note: Function names like `Get_JSON_String`, `Has_Field`, etc. should match existing SPARK JSON utilities.

Phase 3: IR to Code Generation (45-60 minutes)

3.1 Add Break/Continue Code Generation

File: `tools/spark/src/stunir_ir_to_code.adb`

Location: In the step translation procedure (likely `Translate_Steps_To_C`)

Add:

```
when Op_Break =>
    -- v0.9.0: Break statement
    Append (Builder, Get_Indent (Indent) & "break;");

when Op_Continue =>
    -- v0.9.0: Continue statement
    Append (Builder, Get_Indent (Indent) & "continue;");
```

3.2 Add Switch Code Generation

Add:

```
when Op_Switch =>
    -- v0.9.0: Switch/case statement
    declare
        Switch_Expr : constant String := To_String (Step.Switch_Expr);
    begin
        Append (Builder, Get_Indent (Indent) & "switch (" & Switch_Expr & ") {");

        -- Generate case labels
        for I in 1 .. Step.Cases_Count loop
            declare
                Case_Entry : constant IR_Switch_Case := Step.Cases (I);
                Case_Value : constant String :=
                    (if Case_Entry.Value.Is_Integer
                     then Integer'Image (Case_Entry.Value.Int_Value)
                     else To_String (Case_Entry.Value.Str_Value));
            begin
                Append (Builder, Get_Indent (Indent) & "  case " & Case_Value & ":");

                -- Check recursion depth
                if Depth < Max_Recursion_Depth then
                    Translate_Steps (Case_Entry.Body_Steps,
                                     Case_Entry.Body_Count,
                                     Indent + 2,
                                     Depth + 1);
                else
                    Append (Builder, Get_Indent (Indent + 2) & "/* Recursion limit
*/");
                end if;
            end;
        end loop;

        -- Generate default case if present
        if Step.Has_Default then
            Append (Builder, Get_Indent (Indent) & "  default:");
            if Depth < Max_Recursion_Depth then
                -- Count default steps
                declare
                    Default_Count : Natural := 0;
                begin
                    for I in Step.Default'Range loop
                        exit when Step.Default (I).Op = Op_Nop
                                  and then To_String (Step.Default (I).Value) = "";
                        Default_Count := Default_Count + 1;
                    end loop;

                    Translate_Steps (Step.Default,
                                     Default_Count,
                                     Indent + 2,
                                     Depth + 1);
                end;
            end if;
        end if;

        Append (Builder, Get_Indent (Indent) & "}");
    end;
```

Phase 4: Build and Test (30-45 minutes)

4.1 Update Build Configuration

File: `tools/spark/stunir_tools.gpr`

- Verify Ada 2022 support is enabled (`-gnat2022`)
- Add any new source files to `Source_Dirs` if needed

4.2 Compile SPARK Binaries

```
cd tools/spark
gprbuild -P stunir_tools.gpr -p
```

Expected Output:
- `bin/stunir_spec_to_ir_main`
- `bin/stunir_ir_to_code_main`

4.3 Run Tests

```
# Test break_while
bin/stunir_spec_to_ir_main \
    --spec-root ../../test_specs/v0.9.0 \
    --out /tmp/test_spark_ir.json

bin/stunir_ir_to_code_main \
    /tmp/test_spark_ir.json \
    --target c \
    --out /tmp/test_spark.c

# Verify compilation
gcc -c /tmp/test_spark.c -o /tmp/test_spark.o
```

4.4 Cross-Validate with Python/Rust

Compare IR output and C code output with Python and Rust pipelines for all 6 test specs.

Phase 5: Proof Contracts (Optional, 30-60 minutes)

Add SPARK proof annotations for formal verification:

```
procedure Parse_Switch_Statement
    (Stmt : JSON_Value; Step : out IR_Step)
with
    Pre => Is_Valid_JSON_Object (Stmt)
           and then Has_Field (Stmt, "type")
           and then Get_String (Stmt, "type") = "switch",
    Post => Step.Op = Op_Switch
            and then Step.Cases_Count <= Max_Switch_Cases;
```

Testing Strategy

Unit Tests

1. **Break Statement**
   - Break in while loop
   - Break in for loop
   - Break in nested loop (inner only)

2. **Continue Statement**
   - Continue in while loop
   - Continue in for loop
   - Continue with condition

3. **Switch Statement**
   - Simple switch with 2-3 cases
   - Switch with default
   - Switch with fall-through behavior
   - Switch with no default

Integration Tests

- Run all 6 v0.9.0 test specs
- Compare output with Python and Rust
- Verify C code compilation
- Verify functional equivalence

SPARK Proof Tests

- Run `gnatprove` to verify contracts
- Ensure no proof failures
- Verify bounded array access safety

Risk Assessment

Risk	Likelihood	Impact	Mitigation
Bounded array overflow	Medium	High	Use explicit checks, Max_Switch_Cases limit
Recursion depth issues	Low	Medium	Existing Max_Recursion_Depth handling
JSON parsing errors	Low	Medium	Robust error handling in parse routines
Proof failures	Medium	Low	Start without contracts, add incrementally
Type mismatches	Low	Medium	Follow existing patterns for IR_Step

Success Criteria

- [ ] All 6 v0.9.0 test specs pass in SPARK pipeline
- [ ] Generated C code compiles without warnings
- [ ] IR output matches Python/Rust structurally
- [ ] C code output is functionally equivalent to Python/Rust
- [ ] SPARK proof succeeds (if contracts added)
- [ ] No compiler warnings with `-gnatwae` (all warnings as errors)
- [ ] Performance within 2x of Rust implementation

Timeline Estimate

Phase	Time	Cumulative
Type System Extensions	30-45 min	45 min
Spec to IR Parsing	45-60 min	105 min
IR to Code Generation	45-60 min	165 min
Build and Test	30-45 min	210 min
Proof Contracts (Optional)	30-60 min	270 min
Total	**3-4.5 hours**	

Recommended Schedule:
- Break into 2-3 sessions of 1.5-2 hours each
- Allow time for debugging and iteration
- Test incrementally (break/continue first, then switch)

References

Existing Implementation Examples

- **Control Flow (v0.6.1):** See `Op_If`, `Op_While`, `Op_For` in existing code
- **JSON Parsing:** See existing statement parsing in `stunir_spec_to_ir.adb`
- **Code Generation:** See existing C emission in `stunir_ir_to_code.adb`

Documentation

- Ada SPARK Reference: https://docs.adacore.com/spark2014-docs/
- GNAT User Guide: https://docs.adacore.com/gnat_ugn-docs/
- STUNIR IR Schema: `docs/schemas/stunir_ir_v1.md`

Next Steps

1. **Review this roadmap** with SPARK implementation maintainer
2. **Allocate time** for implementation (3-4 hours)
3. **Set up test environment** (GNAT compiler, test specs)
4. **Implement incrementally:** break/continue first, then switch
5. **Test thoroughly** with all 6 v0.9.0 specs
6. **Cross-validate** with Python and Rust
7. **Update documentation** (this roadmap + implementation status)

Document Status: Ready for Implementation
Next Review: After SPARK implementation completion
Owner: SPARK Pipeline Maintainer