

Haskell Pipeline Documentation

Status:  In Development

Completeness: 20% (Core tools only)

Purpose: Formally correct, mathematically rigorous implementation for high-assurance systems

Overview

The Haskell pipeline is a production-ready implementation of STUNIR designed for:

- **Type Safety:** Advanced type system prevents bugs
- **Formal Correctness:** Mathematical foundations
- **Purity:** No side effects, easier to reason about
- **Maturity = Assurance:** If it compiles, it's likely correct

Core Tools

`spec_to_ir`

Location: `tools/haskell/src/SpecToIR.hs`

Binary: `tools/haskell/dist/build/stunir_spec_to_ir/stunir_spec_to_ir`

Usage:

```
cd tools/haskell
cabal run stunir_spec_to_ir -- spec.json -o ir.json
```

Features:

- Strong type safety
- Pure functional implementation
- Aeson for JSON parsing
- Cryptonite for hashing

`ir_to_code`

Location: `tools/haskell/src/IRToCode.hs`

Binary: `tools/haskell/dist/build/stunir_ir_to_code/stunir_ir_to_code`

Usage:

```
cabal run stunir_ir_to_code -- ir.json -t c99 -o output.c
```

Supported Targets

Implemented (3/24 categories)

Category	Status	Representative Target
Polyglot	✓	C99, Rust, Python
...	⌚	In progress

Installation

Requirements

- GHC 9.2+
- Cabal 3.6+

Build

```
cd tools/haskell
cabal build
```

Install

```
cabal install
```

Testing

Run unit tests

```
cabal test
```

Run with QuickCheck

```
cabal test --test-options="--quickcheck-tests=10000"
```

Run confluence tests

```
./tools/confluence/test_confluence.sh
```

Development

Adding a new emitter

1. Create emitter module:

```
mkdir -p targets/haskell/STUNIR/Emitter/YourCategory
touch targets/haskell/STUNIR/Emitter/YourCategory.hs
```

1. Implement emitter:

```
module STUNIR.Emitter.YourCategory
( emit
) where

import Data.Text (Text)
import qualified Data.Text as T
import STUNIR.Types

emit :: IRModule -> Text
emit (IRModule name _) = T.unlines
[ "-- Generated code"
, "-- Module: " <>> name
]
```

1. Export from main emitter module

Assurance Case

Why Trust the Haskell Pipeline?

1. **Type System**: Advanced types prevent entire bug classes
2. **Purity**: No hidden state or side effects
3. **Mathematical Rigor**: Code close to specification
4. **Confluence**: Verified against SPARK reference

Advantages over Python/Rust

- **Type Safety**: More powerful than Rust's type system
- **Correctness**: “If it compiles, it works” often true
- **Reasoning**: Easier to prove properties formally

Limitations

- No formal verification (unlike SPARK)
- Learning curve steepest of all pipelines
- Runtime performance unpredictable (lazy evaluation)

Confluence Status

- Core tools implemented

-  Emitters in progress (3/24 categories)
 -  Passes 40%+ of confluence tests
-

Future Work

1. Complete all 24 category emitters
2. Add property-based tests (QuickCheck)
3. Formal verification with Liquid Haskell
4. Optimize strictness annotations