BASIL Decompiler User Interface

Jesse Graf | s4641125 | Supervisors: Kirsten Winter, Alistair Michael

The University Of Queensland AUSTRALIA

What is BASIL? – Binary Analysis for Secure Information-Flow Logics

BASIL decompiles binary code into an Intermediate Representation (IR), applies transform passes (static analysis passes that simplify and instrument the IR), then transpiles the result to Boogie (an intermediate verification language developed by Microsoft) to formally verify that private data is not leaked (see Figure 1).

AIMS

- Support developers during static-analysis transform passes with a user-friendly interface.
- Visualise IR transformations across decompilation (IR → IR' → … → IR''), emphasising changes.
- Design an extensible architecture to support future extensions.

Features

- Three comparison modes: IR–IR, CFG–CFG, and IR–CFG
- Expandable CFG nodes → straight-line code
- Multi-epoch comparison (e.g., IR → IR'')
- Light / Dark mode; collapsible sidebar
- Specific colour coding to display differences
- Colour-coded diffs + code syntax theme

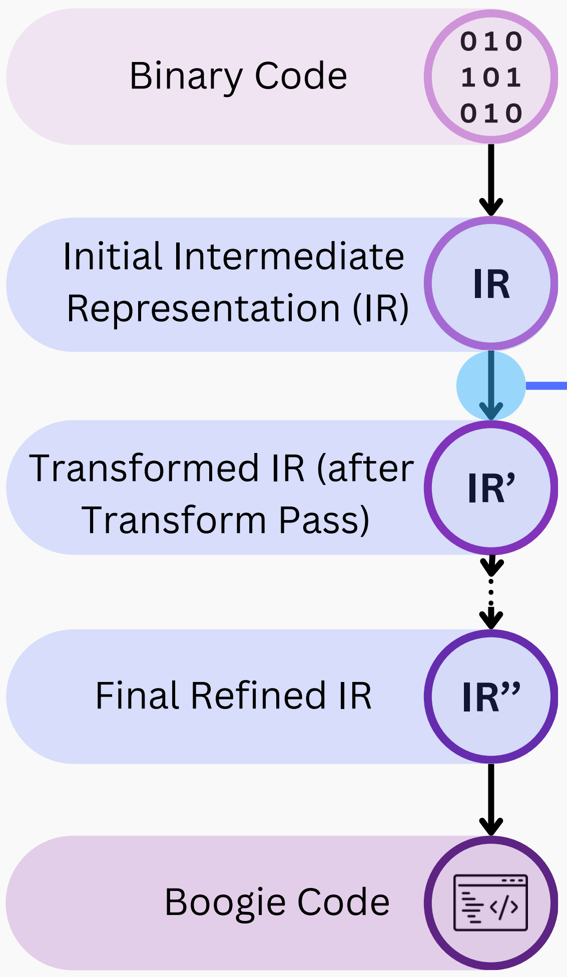
Binary Code
010
101
010
Initial Intermediate Representation (IR)
IR
Transformed IR (after Transform Pass)
IR'
Final Refined IR
IR''
Boogie Code

Figure 1: Simplified BASIL toolchain

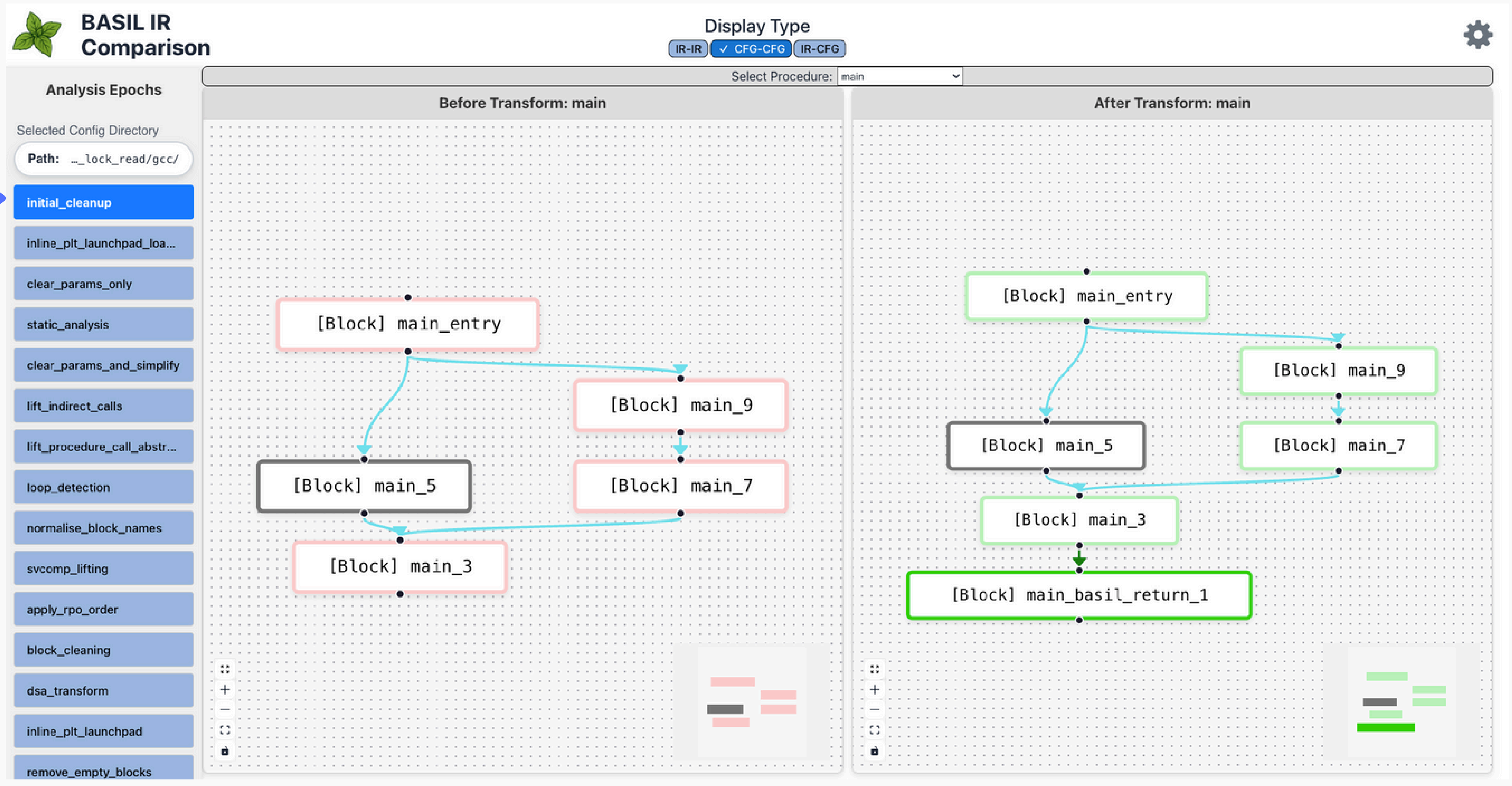

Figure 2: BASIL interface showing the Control Flow Graph (CFG) and CFG comparison view for the initial_cleanup transform pass

Key Libraries

- **CFG display:** ReactFlow + ELK.js + Graphviz
- **Code comparison:** Diff2html + Diff
- **Code colour scheme:** Prism.js
- **Backend:** Http4s + Ember

Development Environment

- **Frontend:** React + TypeScript, built with Vite for fast development and hot reloading
- **Backend:** Scala, the BASIL backend language, provides core logic for IR transformations and exposes a RESTful API to communicate with the frontend
- **Deployment:** Offline web application, accessible across multiple operating systems

Frontend Architecture

Smart Hook / Dumb Component (React version of Container / Presentation pattern)

- Dumb components: Purely UI rendering
- Smart Hooks: Manage state, business logic, and side effects

Orchestrator Pattern

- Central coordinator for sequential workflow and maintaining data consistency
- Ensures that each step of a complex process is executed in the correct order

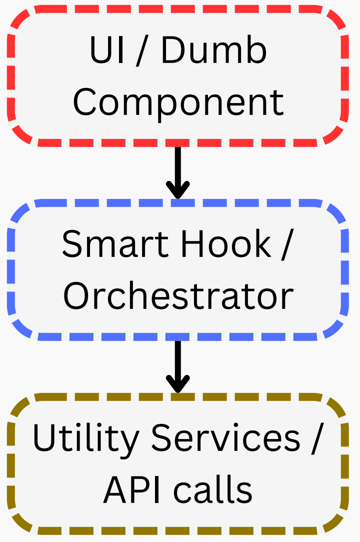

Figure 3: Frontend patterns