Using LLVM Compiler’s Analysis Infrastructure in ROSE Compiler

Sriram Aananthakrishnan (University of Utah)  Daniel Quinlan

**Motivation**

ROSE and LLVM Compilers support program analysis at different levels of Intermediate Representation (IR). Integrating ROSE with LLVM permits low level program analysis information from LLVM available to source level program analysis in ROSE.

**ROSE**

- Source to Source Compiler Infrastructure
- Generates Abstract Syntax Tree (AST) that could be unparsed to generate source code
- Fast AST traversals with attribute evaluation for complex program analysis and transformations
- Provides rich interface for users to write their own analysis and transformations

**LLVM**

- Uses simple low-level IR with strictly defined semantics
- LLVM IR is RISC like infinite register SSA form with low level control flow constructs
- Supports complex program analysis and optimizing transformations at low level
- Easy retargettable code generation system
- Provides Simple Alias Analysis Query Interface

**Goal**

Use LLVM’s Alias Analysis Infrastructure in ROSE and annotate ROSE AST with Alias Set information

**Workflow**

- Parse input C programs with ROSE
- Transform ROSE AST to LLVM IR using ROSE2LLVM Interface
- Invoke LLVM’s Alias Analysis Passes on LLVM IR
- Gather Alias Sets and annotate ROSE AST with results

**Alias Analysis**

- Determines if separate memory references point to same memory location
- Answers may or must alias between any pair of memory references
- Required for Memory Dependence and Data Flow analysis
- Aggressive Optimizations can be performed based on alias information
- Can be Flow sensitive or Flow insensitive and Context sensitive or Context insensitive

**Alias Analysis : Example**

\[ \text{AliasSet} = \{a, b, c, *p, *q\}; \quad \text{AliasSet Type : May} \]