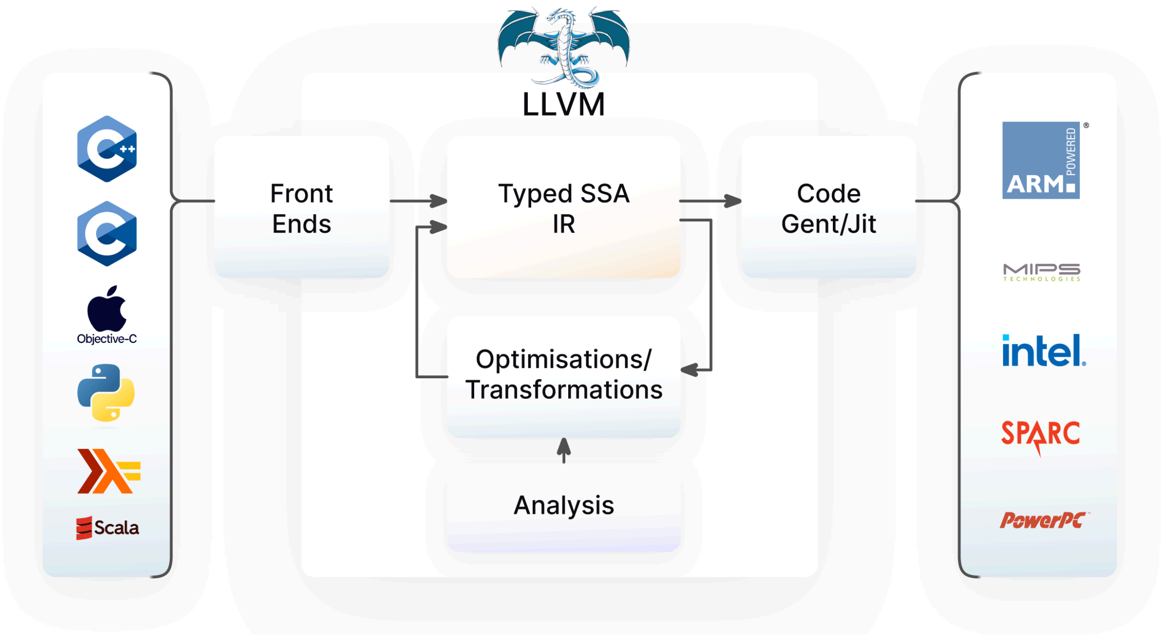
Assignment

Introduction

- LLVM Core, is a target-independent optimizer and code generator.
- It can be used to develop a frontend for any programming language and a backend for any instruction set architecture.
- LLVM Compilation pipeline looks something like the following (Source):



- The sub-projects of LLVM are:
  - LLVM Core
  - Clang
    - Clang Static Analyzer
    - clang-tidy
  - LLDB

- ...
- ...

Motivation

Go through the paper: "TROJAN SOURCE: INVISIBLE VULNERABILITIES" by *Nicholas Boucher & Ross Anderson*. As you learn from the paper, due to the encoding, a compiler sees source code differently than human which can lead to vulnerabilities in code. Can we automatically detect them before the code is actually compiled and produce warnings?

You may use some parts of the LLVM project in your own code. For example, you may use the `clang` compiler for your C++ code. `clang-tidy` , which is part of the llvm project is one such tool that can perform checks on code. The entire list of checks it can perform on source code is present HERE. For example, there are checks for the vulnerabilities listed in the paper:

- MISC-MISLEADING-BIDIRECTIONAL
- MISC-MISLEADING-IDENTIFIER

Additionally, there are checks for code which are often confusing for developers or frequently leads to bugs. For example, MISC-MISPLACED-CONST & BUGPRONE-MACRO-PARENTHESES.

Using *clang-tidy*

Note, you need to install `clang` and `clang-tidy` in your system for this work. One option is to BUILD it from source and use the binary. Let's look at how we can use `clang-tidy` to perform checks. In particular, we are interested in checking **only** BUGPRONE-MACRO-PARENTHESES. You might have seen macro pitfalls similar to the following code.

```
#include <iostream>
#define CUBE(X) (X) * (X) * (X)

int main()
{
int i = 3;
int a = 81 / CUBE(i);
std::cout << "a = " << a;
return 0;
}
```

We may use `clang-tidy` to check for this pitfalls which are error prone. The following command produces warning about this. Assume the code above is saved as *testing.cpp*

`clang-tidy -checks='bugprone-macro-parentheses' testing.cpp -- -std=c++17`

Produces the following output:

```
11630 warnings generated.
testing.cpp:2:27: warning: macro replacement list should be enclosed in
parentheses [bugprone-macro-parentheses]
    2 | #define CUBE(X) (X) * (X) * (X)
      |                           ^
      |                 (              )
Suppressed 11629 warnings (11629 in non-user code).
Use -header-filter=.* to display errors from all non-system headers. Use -
system-headers to display errors from system headers as well.
```

Problem

- Your task is to write such a check and integrate into `clang-tidy`.
- This PAGE lists a wide range of recommendations for developers. You task is to integrate the following recommendations:
  - ARR02-C
  - PRE06-C
  - EXP00-C.
- The first step is to find violations of the recommendations.
- The next step is to automatically fix such violations by rewriting the code.
- Once implemented, run your checks on top-25 C++ projects in GitHub sorted by the number of stars and report the findings.

Resources

The following are some of the resources you might find useful. Feel free to search for more on your own.

1. Download and build `clang-llvm` by following this TUTORIAL.
2. Official tutorial on writing clang-tidy CHECKS.
3. Clang transformer TUTORIAL.
4. AST Matcher REFERENCE.
5. Blog posts:
   1. EXPLORING CLANG TOOLING PART 1: EXTENDING CLANG-TIDY
   2. EXPLORING CLANG TOOLING PART 2: EXAMINING THE CLANG AST WITH CLANG-QUERY
   3. EXPLORING CLANG TOOLING PART 3: REWRITING CODE WITH CLANG-TIDY
   4. LOCATION
   5. AST-MATCHING REFACTORING