

Sanitizer & LLVM Pass

SWPP

Apr. 2nd

Juneyoung Lee

Assignment 3

- Two goals:
 1. Find bug in a C program using Clang Sanitizer & fix it
 2. Write a pass that detects unreachable basic blocks
- Deadline: **Next Sunday, Apr 12th**, 11:59 pm
- I'll announce the full document by tomorrow (Fri.)

Clang Sanitizer

- A tool that helps you detect undefined behaviors at runtime
- `clang -fsanitize=XXX a.c`
 - `undefined`: detects UBs from arithmetic operations
 - `memory`: detects reading uninitialized memory
 - `address`: detects use-after-free, etc
 - They are all undefined behaviors in C!

Running Example

```
#include<stdio.h>

int main() {
    printf("Type two (small, large) positives to calculate average: ");
    int a, b;
    scanf("%d %d", &a, &b);

    int average = (a + b) / 2;
    printf("Average: %d\n", average);

    return 0;
}
```

Solution

```
#include<stdio.h>

int main() {
    printf("Type two (small, large) positives to calculate average: ");
    int a, b;
    scanf("%d %d", &a, &b);

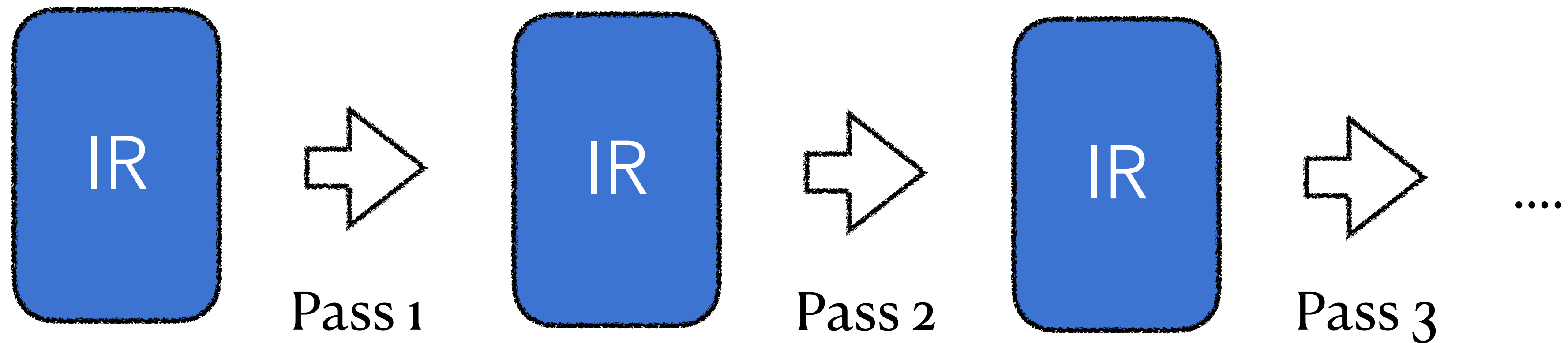
    int average = a + (b - a) / 2;
    printf("Average: %d\n", average);

    return 0;
}
```

Running Sanitizer Example

- People who had a problem with compiler-rt wouldn't have sanitizer enabled.. :(
 - Please download prebuilt release 9.0.0 & use it.
- Go to `swpp202001/practice/3.materials`
- `run-ubsan.sh my-llvm-release/bin`

LLVM IR Pass



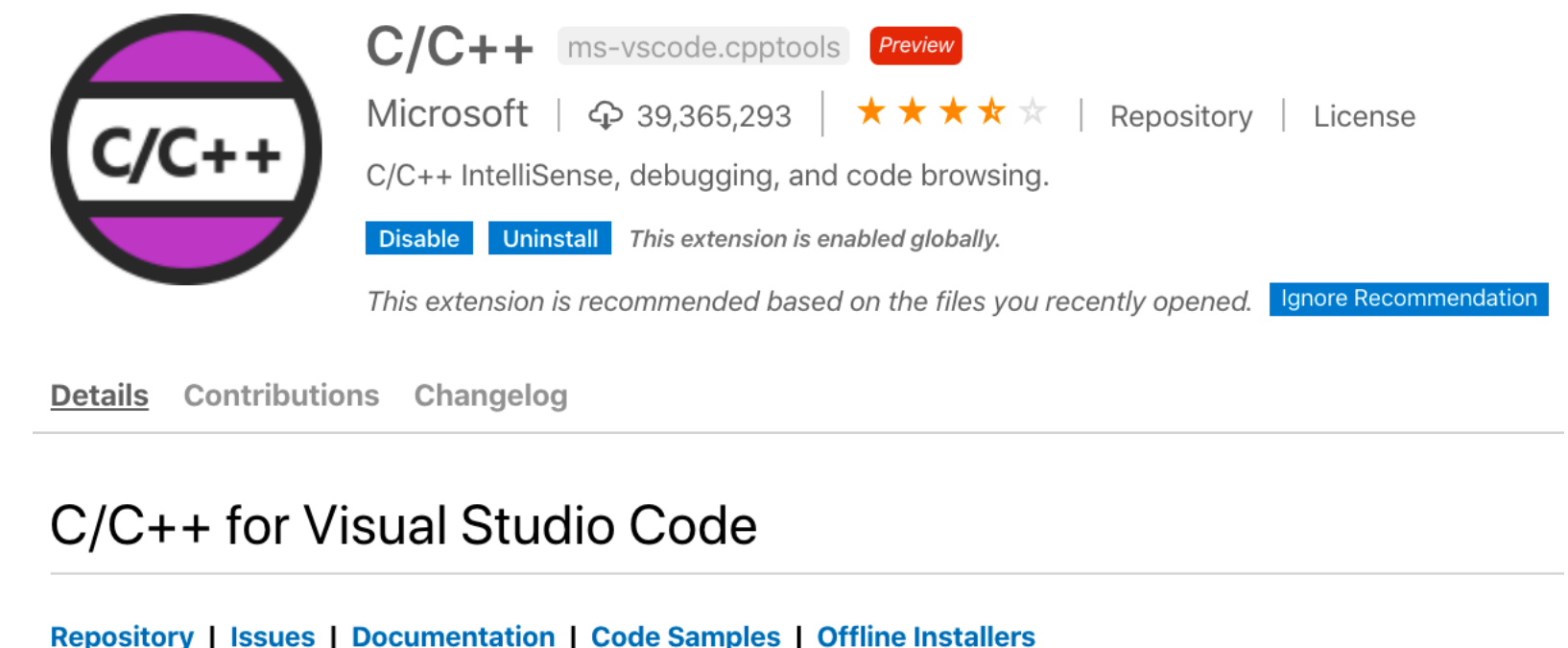
- Each transformation (or optimization) is called pass.
- -O1, -O2, -O3: a set of (more) passes.

Prerequisites

- Prebuilt release 9.0.0 is okay
- LLVM from master branch (the one built using llvmscript) is also okay

C++ IDE

- You will need C++ IDE, otherwise things will be pretty tough :/
- Recommended: Visual Studio Code! <https://code.visualstudio.com>
 - Interactive & fast (e.g. when you want to locate a class)
 - On Linux/Mac: Download it & execute
 - On Windows Subsystem for Linux: run “code .”
 - Install C/C++ extension
- For faster browsing, you’ll need to update include directory (#include will be underlined as red; please click it)



1. HelloPass

- Full code: hello.cpp

```
class HelloPass : public PassInfoMixin<HelloPass> {
public:
    PreservedAnalyses run(Function &F, FunctionAnalysisManager &FAM) {
       StringRef funcName = F.getName();
        outs() << "Hello, " << funcName << "!\n";
        return PreservedAnalyses::all();
    }
};

extern "C" ::llvm::PassPluginLibraryInfo
llvmGetPassPluginInfo() {
    // Registration of HelloPass: omitted for brevity
}
```

Hierarchy

- `llvm::Module` class
- `llvm::Function` class
- `llvm::BasicBlock` class
- `llvm::Instruction` class

To see class hierarchy & their methods..

Search Google / Read code / See Autocompletions



inherits (is-a)

`llvm::LoadInst`

`llvm::ICmplInst`

`llvm::BinaryOperator`

`llvm::StoreInst`

`llvm::BranchInst`

`llvm::PHINode...`

How To Run HelloPass?

- Compile: it is slightly complex.. Please use `./run-passes.sh`
 - If you like to challenge, have a look at the script! It isn't very hard
- Run: `./run-passes.sh` also works, but you can try:
 - `opt -disable-output -load-pass-plugin=libHello.so -passes="hello" foo.ll`

2. DumpPass

```
class DumpPass : public PassInfoMixin<DumpPass> {
public:
    PreservedAnalyses run(Function &F, FunctionAnalysisManager &FAM) {
        outs() << "<<" << F.getName() << ">>\n";

        for (BasicBlock &BB : F) {
            outs() << "BasicBlock: " << BB.getName() << "\n";

            for (Instruction &I : BB)
                outs() << "\t" << I << "\n";
        }
        return PreservedAnalyses::all();
    }
};
```

Print Successors

```
class DumpPass : public PassInfoMixin<DumpPass> {
public:
    PreservedAnalyses run(Function &F, FunctionAnalysisManager &FAM) {
        for (BasicBlock &BB : F) {
            outs() << "BasicBlock: " << BB.getName() << "\n";

            unsigned successorCnt = BB.getTerminator()->getNumSuccessors();

            for (unsigned i = 0; i < successorCnt; ++i) {
                BasicBlock *NextBB = BB.getTerminator()->getSuccessor(i);
                outs() << NextBB->getName() << "\n";
            }
        }
        return PreservedAnalyses::all();
    }
};
```