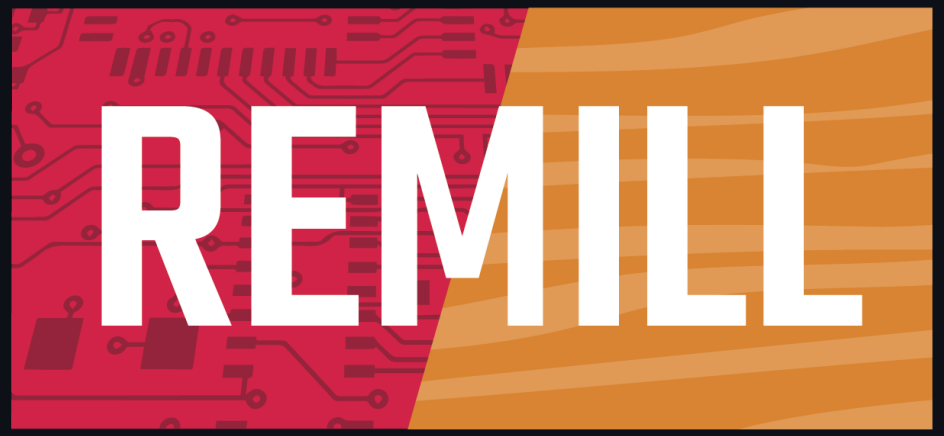
Workshop: LLVM Lifting Basics



Duncan Ogilvie

Setup: remill.ogilvie.pl

- Install & Start Docker Desktop
- Install Visual Studio Code
- Clone and open the repository
- Click 'Reopen in Container'

Introduction

- Meant for absolute (LLVM) beginners
- Format: hands-on workshop
- Interactive

Outline

- LLVM Theory
- LLVM Command Line
- LLVM C++ API
- Remill Theory
- Remill Command Line

What is LLVM?

- Low Level Virtual Machine
- Meant for compiler development
- LLVM IR: cross-platform representation
  - Loops
  - Switch tables
  - Functions
- Reusable optimizations

LLVM IR Module

- Functions
- Basic Blocks
  - Instructions
- Globals
- Metadata

LLVM IR Concepts

- Identifiers: `@global` / `%local`
- Well-formedness
  - Type checking
  - Terminator instructions
  - Entry no predecessor
  - CFG integrity
  - phi/alloca at the start
- `llvm::verifyModule` + `LLVM_ENABLE_ASSERTIONS`

Other Concepts

- LLVM Language Reference
- Opaque Pointers
- `getelementptr`
    - Godbolt Example

Time for a hands-on session!

Instructions: `exercises/1_llvmir/README.md`

LLVM C++ API

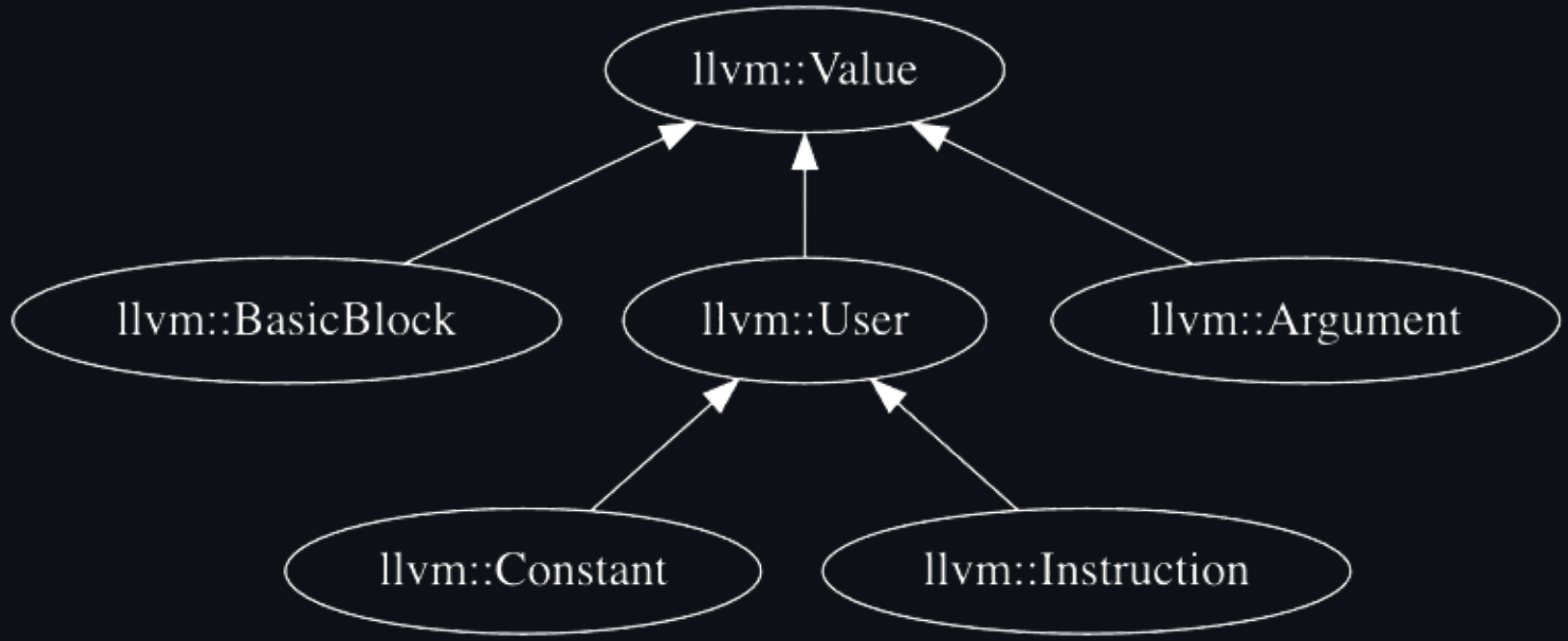
- 😭 Difficult to navigate
- ✅ Annoying to set up
- 🤖 Use Google/ChatGPT liberally

LLVM Hello World

Show: `src/process-module.cpp`

LLVM Memory Model

- `llvm::Value *` (same pointer -> same value)
- `llvm::isa<T>` / `llvm::dyn_cast<T>`



Time for a hands-on session!

Instructions: `exercises/2_api/README.md`

What is Remill?

- Trail of Bits
- 'Lifts' native instructions to LLVM IR
  - Applications: binary analysis/instrumentation/emulation
- Mild abuse of the IR, requires some tricks

Remill Insights

- Semantics in C++
  - Easier to maintain
  - Compiled to LLVM IR
- `State*` structure -> CPU Registers
- `Memory*` pointer -> Memory Manager
  - Total ordering to preserve semantics
- 'Massaging' required

Instruction Semantics

```
template <typename D, typename S>
DEF_SEM(MOV, D dst, const S src) {
  WriteZExt(dst, Read(src));
  return memory;
}

DEF_ISEL(MOV_GPRv_MEMv_32) = MOV<R32W, M32>;
```

Lifting Basic Blocks

Basic Block Definition:

```
Memory *__remill_basic_block(State &state, Ptr block_addr, Memory *memory);
```

- Calls to the semantics are inserted here.
- State is fully symbolic
- Requires additional work to restore the calling convention

High level example

```
Memory *__remill_basic_block(State &state, Ptr block_addr, Memory *memory) {
    // mov rax, rdi
    state.rax = state.rdi;
    state.rip += 3;
    // ret
    state.rip = *(Ptr*)state.rsp;
    state.rsp += sizeof(Ptr);
    return __remill_function_return(state, state.rip, memory);
}
```

Time for a hands-on session!

Instructions: `exercises/3_lifting/README.md`

Thanks

Matteo Favaro - mentoring and slides