1

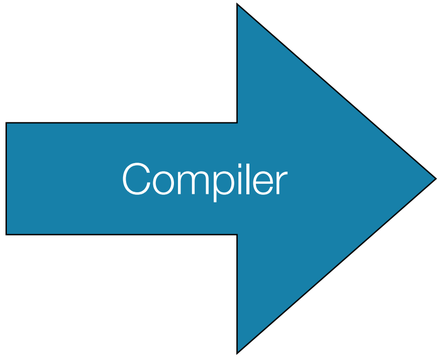
Verifiable Binary Lifting

Joe Hendrix, Andrew Kent and Simon Winwood
Galois, Inc
HCSS 2021

What is Decompilation?

- A **compiler** translates code written in a high-level language into a low level language for efficient execution.

```
uint64_t fib(uint64_t x) {
    if (x <= 1) {
        return x;
    } else {
        return fib(x-1)+fib(x-2);
    }
}
```




```
0000000000201000 fib:
  201000: 55                pushq %rbp
  201001: 4889e5            movq %rsp, %rbp
  201004: 4883ec20          subq $32, %rsp
  201008: 48897df0          movq %rdi, -16(%rbp)
  20100c: 48837df001        cmpq $1, -16(%rbp)
  201011: 0f870d000000      ja   13 <fib+0x24>
  201017: 488b45f0          movq -16(%rbp), %rax
  20101b: 488945f8          movq %rax, -8(%rbp)
  20101f: e934000000        jmp  52 <fib+0x58>
  201024: 488b45f0          movq -16(%rbp), %rax
  201028: 482d01000000      subq $1, %rax
  20102e: 4889c7            movq %rax, %rdi
  201031: e8caffffff        callq -54 <fib>
  201036: 488b4df0          movq -16(%rbp), %rcx
  20103a: 4881e902000000    subq $2, %rcx
  201041: 4889cf            movq %rcx, %rdi
  201044: 488945e8          movq %rax, -24(%rbp)
  201048: e8b3ffffff        callq        -77 <fib>
  20104d: 488b4de8          movq -24(%rbp), %rcx
  201051: 4801c1            addq %rax, %rcx
  201054: 48894df8          movq %rcx, -8(%rbp)
  201058: 488b45f8          movq -8(%rbp), %rax
  20105c: 4883c420          addq $32, %rsp
  201060: 5d                popq %rbp
  201061: c3                retq
```

- A **decompiler** reverses steps in this translation

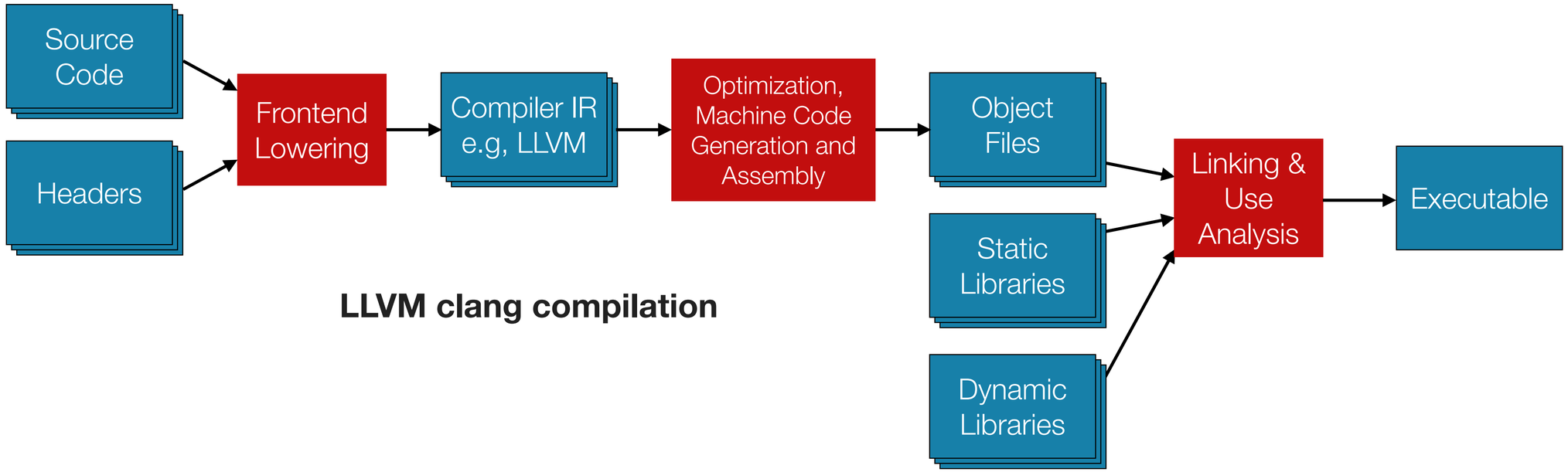
Who uses decompilers?

- Decompilers commonly used by **reverse engineers** to understand a program.
  - Decompile into a **language understandable by people**.
  - Engineer works with the decompiler to translate code into idiomatic code.
- Without hints or existing source to target, it is generally impossible to recover the original source.
  - Information lost includes all the structure within function bodies such as original control flow structure and local variables.
  - Much more information is lost when compiling with **optimization**.
- More recent programs are aimed at using decompilers for **program transformation** and **repair**.

Decompilation for Program Transformation

- Researchers are increasingly looking at using decompilers to transform programs.
  - Patch code with vulnerabilities.
  - Extract functionality from legacy code for use in new applications.
  - Apply new compiler optimizations or insert security checks into legacy applications.
  - Port a program from one platform to another, e.g. x86 to WASM.
- These new applications place greater emphasis on **program correctness** and may have less emphasis on **programmer understanding.**

Compilation Toolchain



LLVM clang compilation

- **Decompilation** needs to reverse these steps.

Decompilation/Binary Lifting Tools

- There are many such tools available:
  - Ghidra, IDA/Hex-Rays, Binary Ninja, McSema, RetDec, JEB, Grammatech, Phoenix, reopt
- The problem space is very large:
  - Several large and complex instruction set architectures:
    e.g., x86, x86_64, ARM 8A/7M, PPC, RISCV, …
  - Variety of operating systems and executable formats: PE, Elf, Macho
  - Language and toolchain specific features: e.g, GNU extensions, C++ vtables, eh_frame, …
  - Lack of specs for many features, and executables often out of spec (but still “work”).

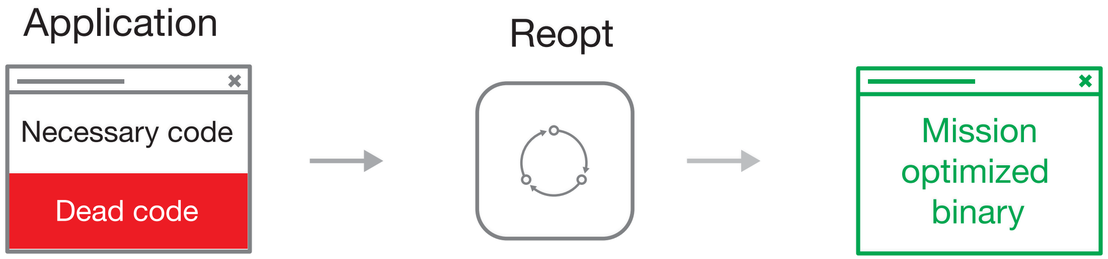
Related Work
Assurance of Decompilation

- **Scalable Validation of Binary Lifters**
  Sandeep Dasgupta, Sushant Dinesh, Deepan Venkatesh, Vikram S. Adve, Christopher W. Fletcher
  PLDI, 2020

- **Evolving Exact Decompilation**
  Eric Schulte, Jason Ruchti, Matt Noonan, David Ciarletta, Alexey Loginov
  Workshop on Binary Analysis Research, 2018

Program Recompilation

- We have implemented an end-to-end recompilation tool called **reopt**.



- Written in a modular fashion so components can be used on other use cases.
  - Core binary analysis component also used for verification of machine code.

Three Step Process

Three Step Process

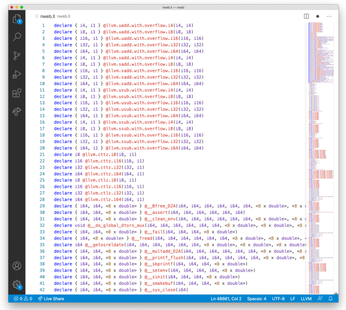
1. Decompilation
2. Recompilation
3. Relinking

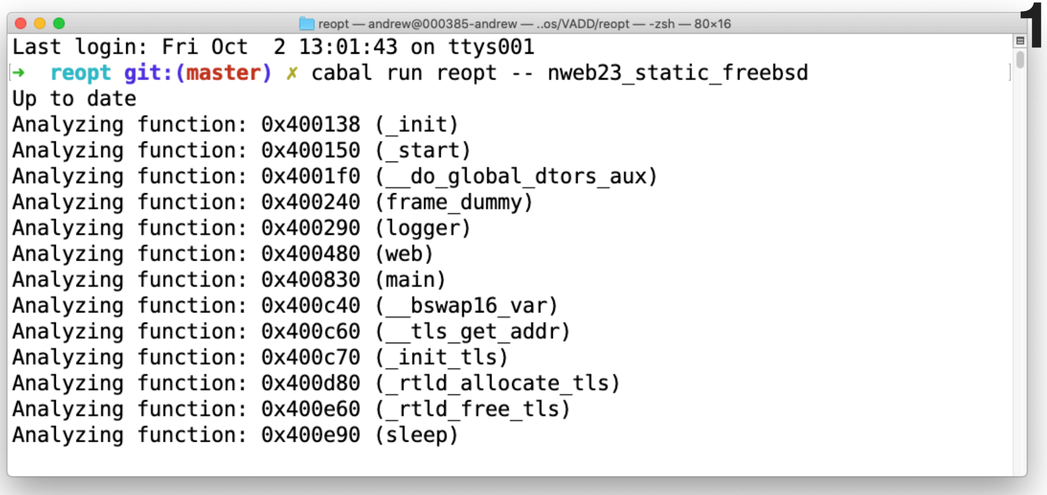
1.Decompilation

```
reopt — andrew@000385-andrew — ..os/VADD/reopt — -zsh — 80×16
Last login: Fri Oct  2 13:01:43 on ttys001
➜  reopt git:(master) ✗ cabal run reopt -- nweb23_static_freebsd
Up to date
Analyzing function: 0x400138 (_init)
Analyzing function: 0x400150 (_start)
Analyzing function: 0x4001f0 (__do_global_dtors_aux)
Analyzing function: 0x400240 (frame_dummy)
Analyzing function: 0x400290 (logger)
Analyzing function: 0x400480 (web)
Analyzing function: 0x400830 (main)
Analyzing function: 0x400c40 (__bswap16_var)
Analyzing function: 0x400c60 (__tls_get_addr)
Analyzing function: 0x400c70 (_init_tls)
Analyzing function: 0x400d80 (_rtld_allocate_tls)
Analyzing function: 0x400e60 (_rtld_free_tls)
Analyzing function: 0x400e90 (sleep)
```

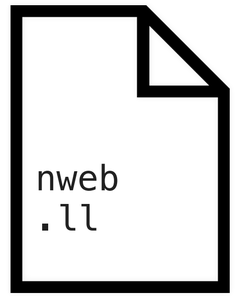
nweb
.exe

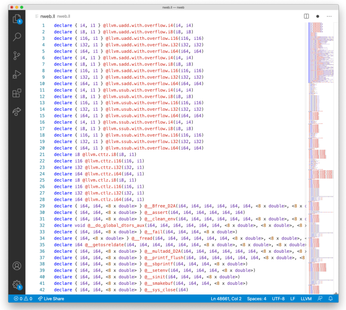
nweb
.ll



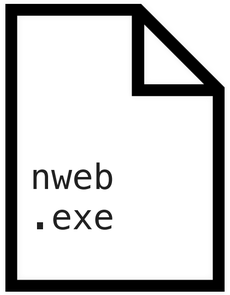


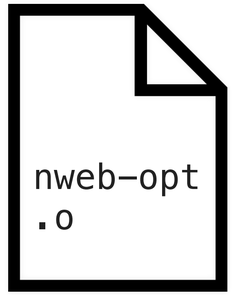
1.Decompilation

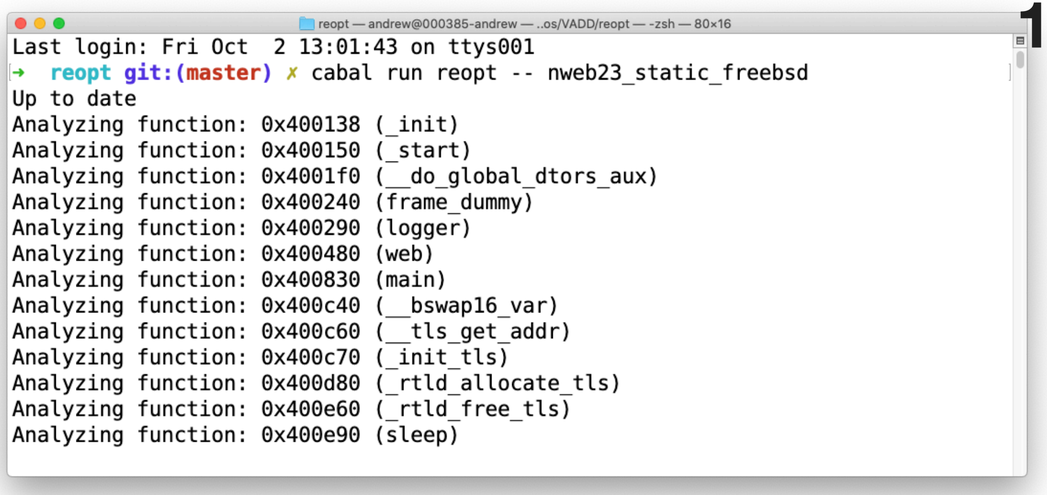




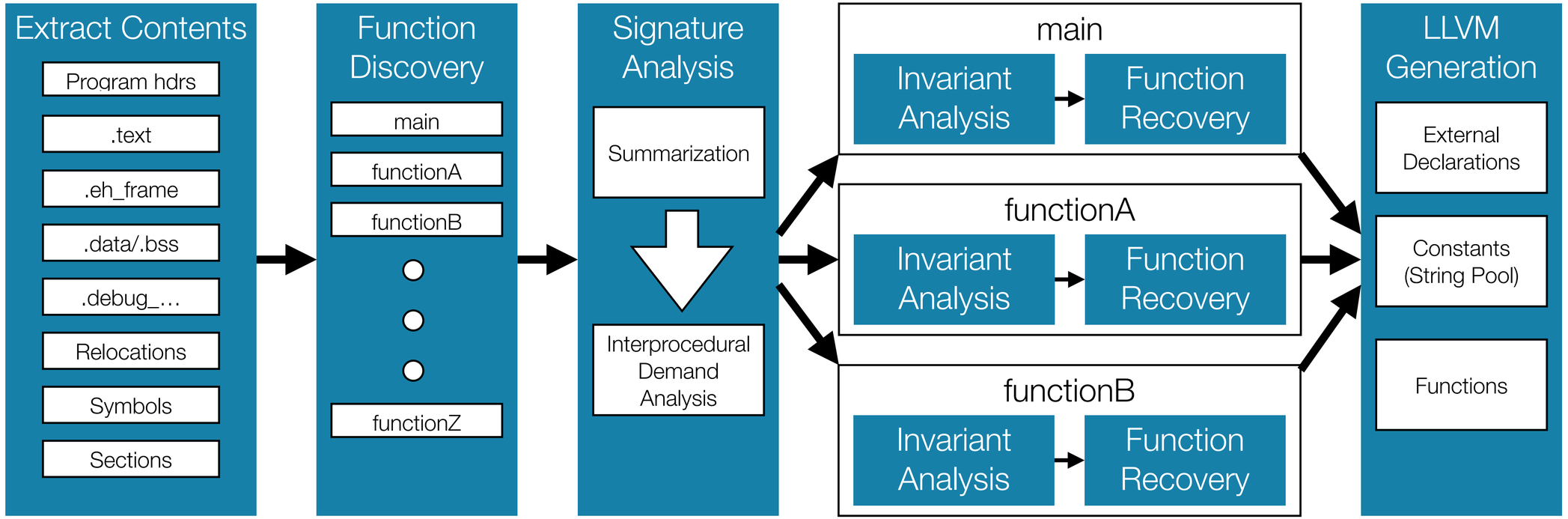
2. Recompilation







Decompilation Pipeline



Supported Features

- Analysis component has large coverage of x86_64, ARM and PowerPC ISAs
  - x86_64 includes significant SSE/AVX support, some FPU/MMX support.
- Invariant inference and LLVM generation limited to x86_64 ISA.
  - x86_64 coverage is a more limited.
- Static and dynamically linking executables.
- Extracting information from debug and .eh_frame data.

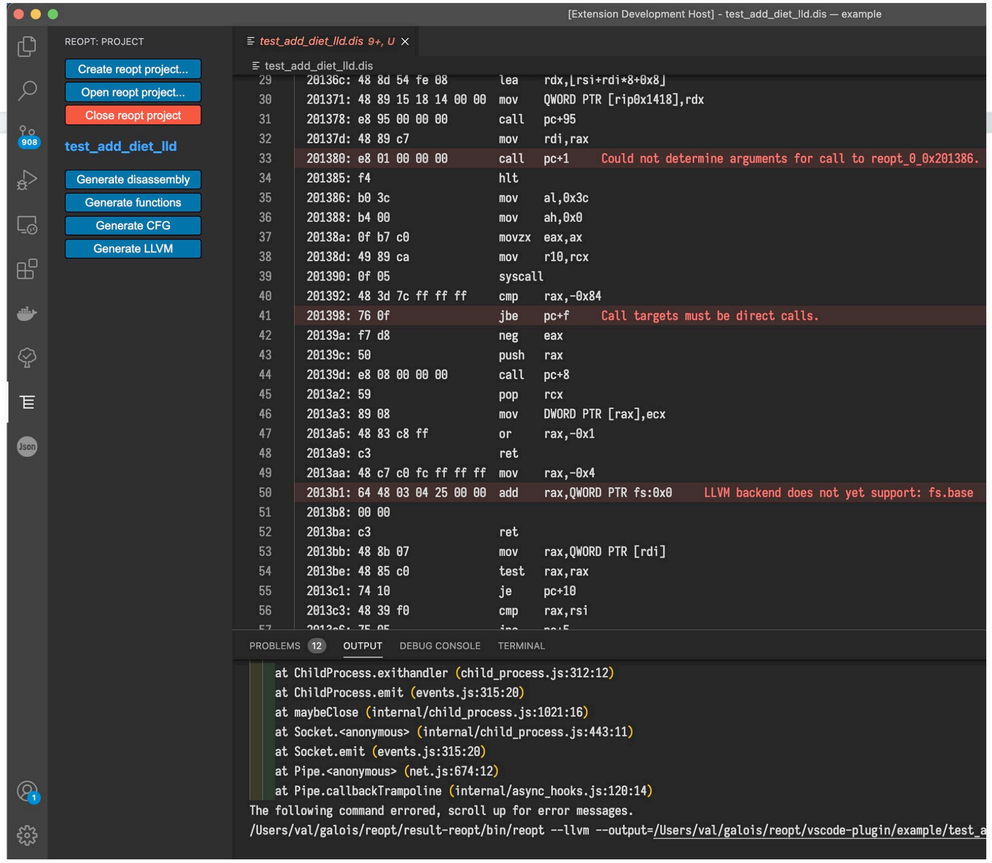
Handling Failures

Pipeline steps may fail, and reopt recovers as much as it can.

1. Elf files sometimes inconsistent or use extensions we do not support.
2. Discovery may fail to find all functions or spurious code.
   - Funcitions only indirectly referenced in structs primary failure to find.
   - Spurious code from no-return functions.
3. Function argument analysis is main limitation currently.
   - Lack mechanism for inferring arguments to externally linked functions.
4. Invariant analysis and function recovery do not support full instruction set.

Interoperability

- Can export intermediate results at each stage of pipeline.
- VSCode extension under development.
- Binary analysis plugins for Ghidra and Binary Ninja developed on previous projects.



Verification

Verification Properties

Recompilation Soundness

- Every observable execution in the LLVM should be possible in the machine code program.

$$t \in traces(P_{LLVM}) \Rightarrow \exists\ t' \in traces(P_{MC}),\ t \equiv t'$$

Verification Soundness

- If a property is true of the raised program, then it should be true of the machine code program.

Observational Equivalence

- Our current notion of equivalence is based on event traces.
- Required events include:
  - Writes to non-stack addresses.
  - Other operations that may raise signals (e.g., divide-by-zero).
  - System calls
- Internally, we make additional equivalence checks for compositional purposes.

Verification Approaches

1. Build a **verified decompiler** using interactive theorem proving.
2. Use **automation** to check program equivalence after generation.

Verification Approaches

1. Build a **verified decompiler** using interactive theorem proving.

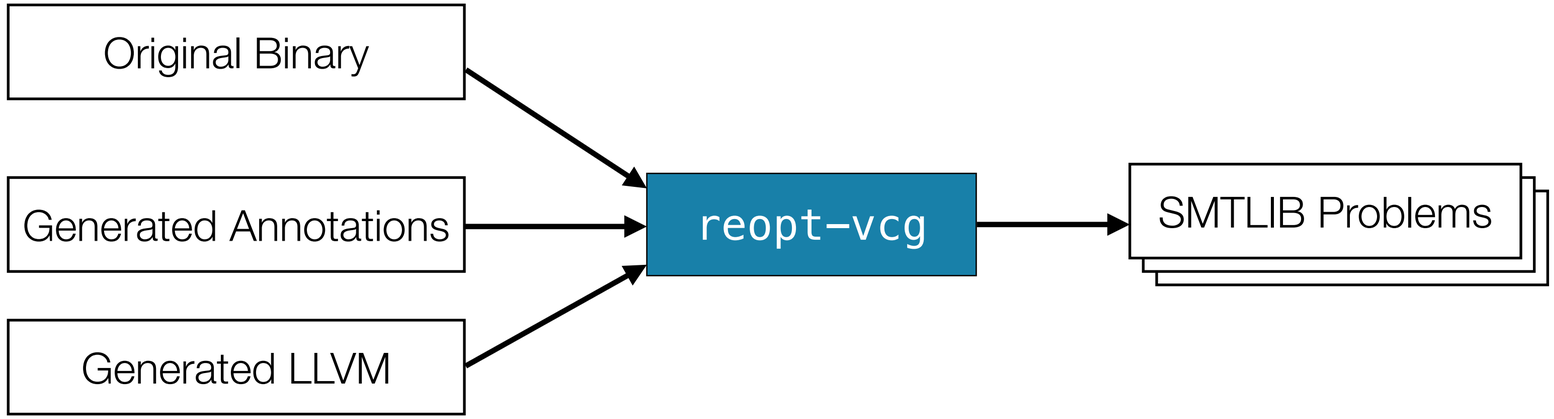
- Decompilation is an open-ended problem.
- Very complex to implement, and needs continued improvement.

2. Use **automation** to check program equivalence after generation.

- Program equivalence is ordinarily undecidable…
- However, the decompiler output is structurally similar to input binary.
- We have developed a **compositional approach** that checks equivalence of **basic blocks** using SMT solving.

Verification Approach

- We have implemented a verifier based on translation validation.



Correctness claim: If all SMTLIB SAT problems are unsat, then the generated LLVM refines the original binary

VCG Implementation

- The implementation of reopt-vcg is independent of reopt itself.

Reopt

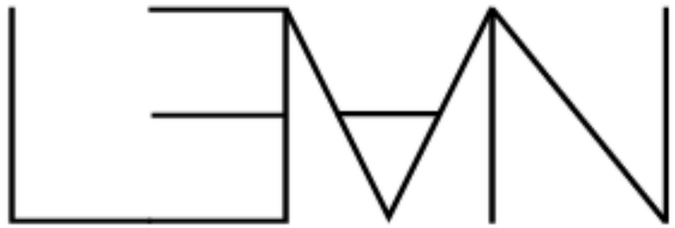
- Written in Haskell
- Disassembler based on udis86.
- Custom `x86_64` semantics.
- Haskell LLVM-pretty library used for generating LLVM.

reopt-vcg

- Written in Lean 4
  - Long-term goal is to verify reopt-vcg.
- Disassembler from LLVMMC.
- UIUC K `x86_64` semantics.
- libllvm used for parsing

Lean 4

- Lean 4 is a new programming language and theorem prover.
  - Publicly available but not officially released yet.
  - Functional language based on dependent type theory.
  - Compiles to efficient C code; No cyclic data-structures.
  - Self hosting - Largely written in Lean itself.
- Builds on Lean 3
  - Mathlib — a large community effort to formalize mathematics.
  - ~450kloc of definitions and theorems as of January 2021.

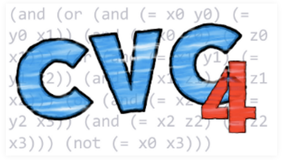


Satisfiability Modulo Theories (SMT) Solvers

- **SMT Solvers** can automatically prove theorems involving specific **theories**.
  - A theory has one or more types (called **sorts** in SMT) along with operations.
  - There are combination methods that allow **theory solvers** to work together.
  - reopt-vcg uses bitvectors, arrays (with a partial equivalence extension), and uninterpreted functions.
  - SMT solving is **NP-hard** (some theories are more difficult), but a lot of work has gone into making it tractable.

SMT-COMP 2021

https://smt-comp.github.io/



Compositional Proofs

- The key to making automation tractable is to decompose the overall equivalence of programs into many smaller proofs.
- Instead of asking:

  Is LLVM Program P equivalent to machine code program Q?

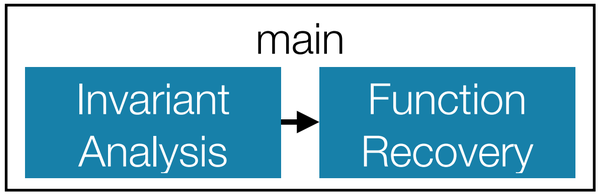
- We instead ask solvers to answer many questions of the form:

  Is this effect in a LLVM basic block B equivalent to this effect in the machine code?

- For a compositional strategy, we need
  - All the assumptions needed to make the statement true.
  - Check that the assumptions hold when jumping from one block to another.

Compositional Properties

- Reopt-VCG's compositional strategy enforces
  - Functions respect the ABI (how arguments are passed, callee-saved registers, etc)
  - The size of each stack frame is bounded to at most a page and all stack accesses are in bounds.
    - Needed to avoid accessing heap memory via stack pointers.
  - Callee saved information is saved and persisted and not modified by the function.



Getting Reopt

- reopt and reopt-vcg are publicly available under open source libraries.

```
https://github.com/GaloisInc/reopt
```

- You can try it out online through Gitpod, download a Docker image, or use prebuilt binaries.

Some Observations

- The binary analysis ecosystem has a rich variety of tools.
  - Made much easier by extensive documentation and open-source libraries.
  - Prefer libraries that provide semantics as data or a DSL rather than code.
- Large and complex instruction sets represent a significant but understood challenge.
- Operating system, debug and linker extensions challenging:
  - Lack of consistent documentation
  - Large variety of extensions
  - Implementations make different choices and change regularly.

Thank you

Acknowledgements:

- Guannan Wei of Purdue implemented an early prototype of reopt-vcg.
- Rob Dockins (Galois) wrote the Lean LLVM bindings.
- Leonardo de Moura and Sebastian Ulrich for help with Lean 4.
- The CVC4 team for the partial array equality extension to CVC4.
- Andrei Stefanescu (Galois) and the UIUC K Semantics group for their x86_64 semantics.

The project depicted is sponsored by the Office of Naval Research under Contract No. N68335-17-C-0558. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.