IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Quarkslab
Securing every bit of your data

Trace-based approach to compiler debugging

GT Debugging - GDR-GPL 2023

Bruno MATEU

June, 8th

Introduction

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Quarkslab
Securing every bit of your data

Introduction
Obfuscations

```
; Function Attrs: mustprogress nofree norecurse nosync nounwind ↩
    readnone sspstrong willreturn uwtable
define dso_local i32 @foo(i32 noundef %0, i32 noundef %1, i32 noundef ↩
    %2) local_unnamed_addr #0 {
  %4 = add nsw i32 %1, %0
  ret i32 %4
}
```

```
; Function Attrs: mustprogress nofree norecurse nosync nounwind ↩
    readnone sspstrong willreturn uwtable
define dso_local i32 @foo(i32 noundef %0, i32 noundef %1, i32 noundef ↩
    %2) local_unnamed_addr #0 {
  %4 = xor i32 %1, %0
  %5 = and i32 %1, %0
  %6 = shl i32 %5, 1
  %7 = add i32 %6, %4
  ret i32 %7
}
```

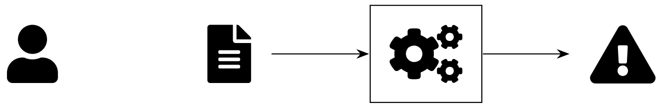
1 • Context

Quarkslab
Securing every bit of your data

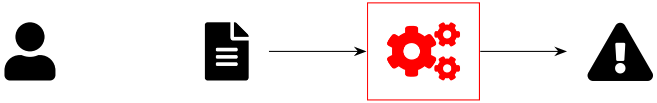
IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Context

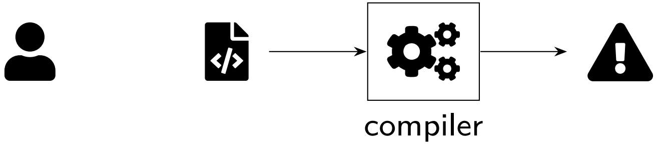
Finding bugs in compilers



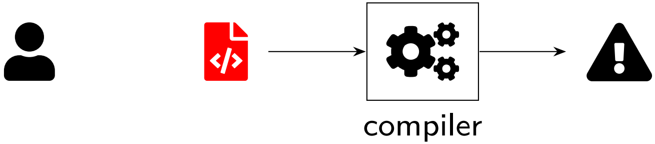
Context
Finding bugs in compilers



Context
Finding bugs in compilers

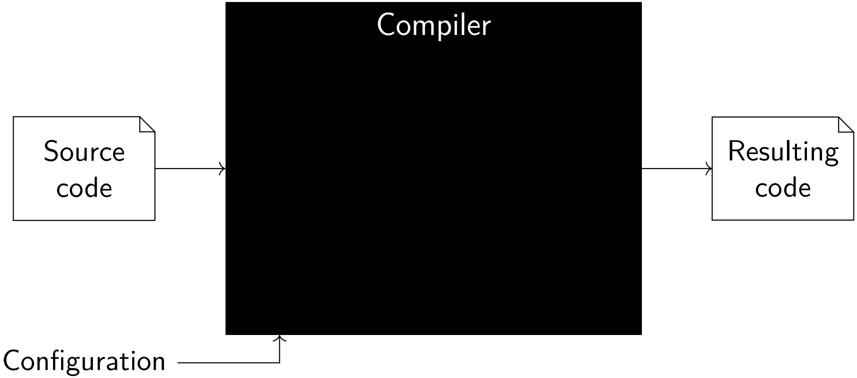


Context
Finding bugs in compilers



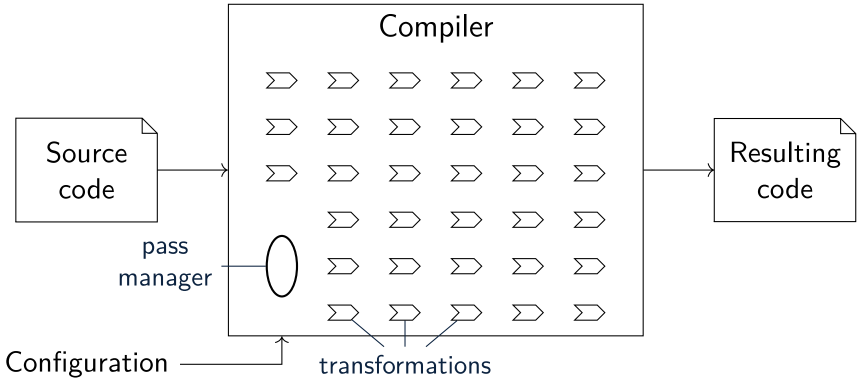
Context

Compiler overview

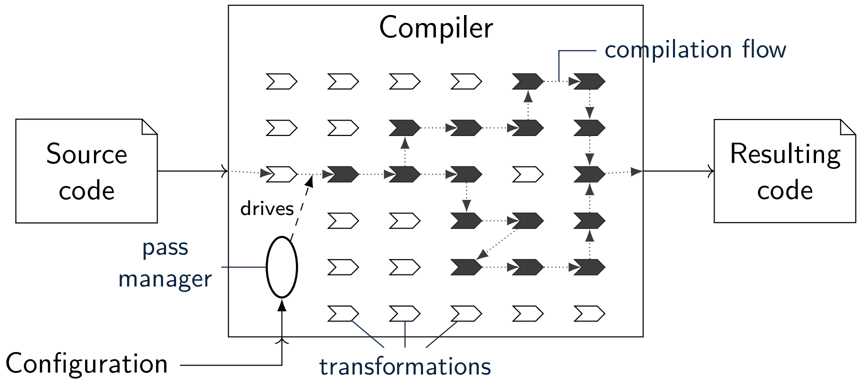


Context

Compiler insides



Context
Compiler insides



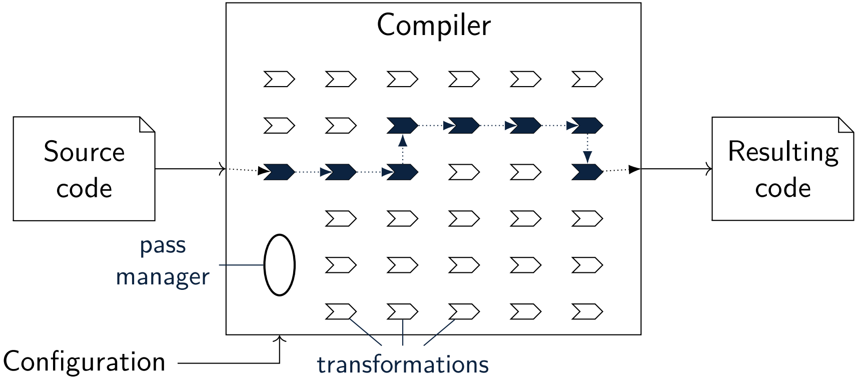
2 • Motivations

Quarkslab
Securing every bit of your data

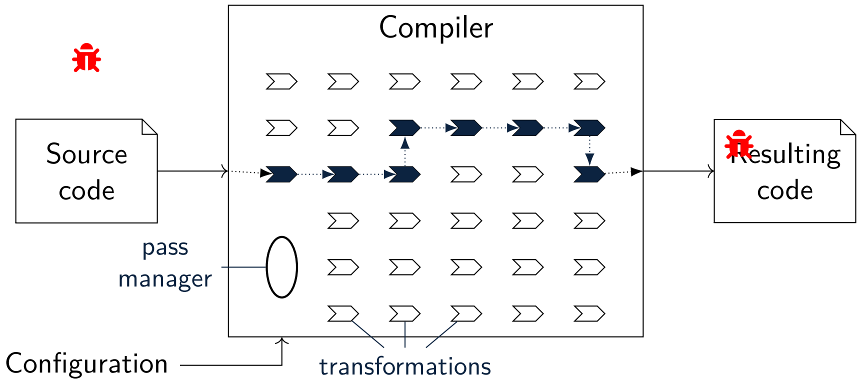
IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Motivations

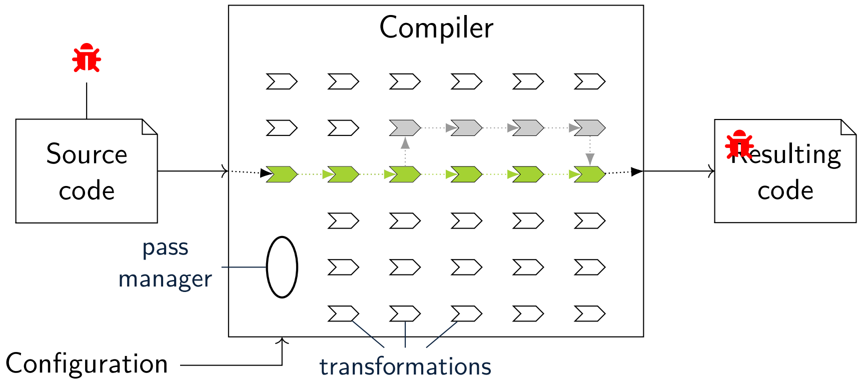
Finding bugs in compilers



Motivations
Finding bugs in compilers

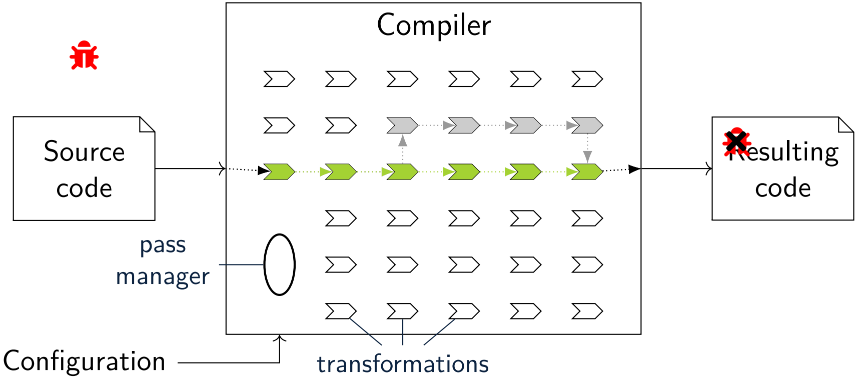


Motivations
Finding bugs in compilers



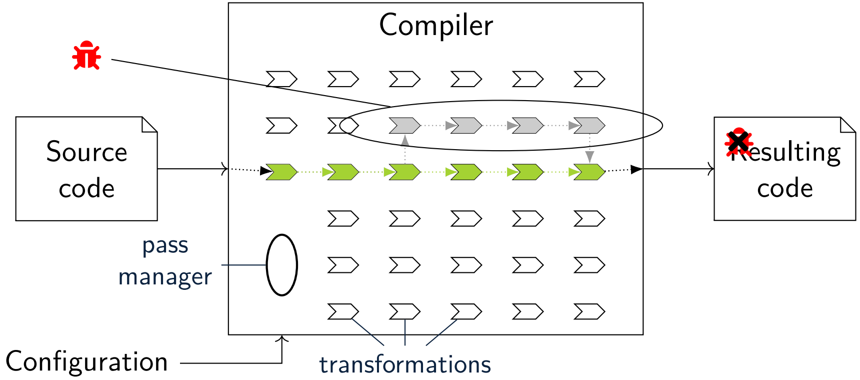
Motivations

Finding bugs in compilers

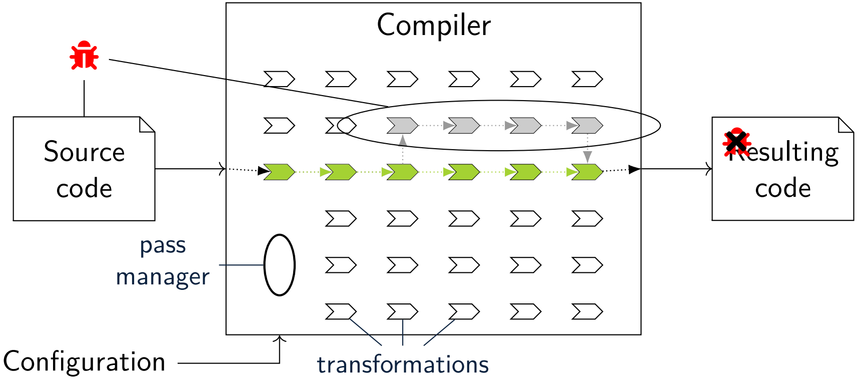


Motivations

Finding bugs in compilers



Motivations
Finding bugs in compilers

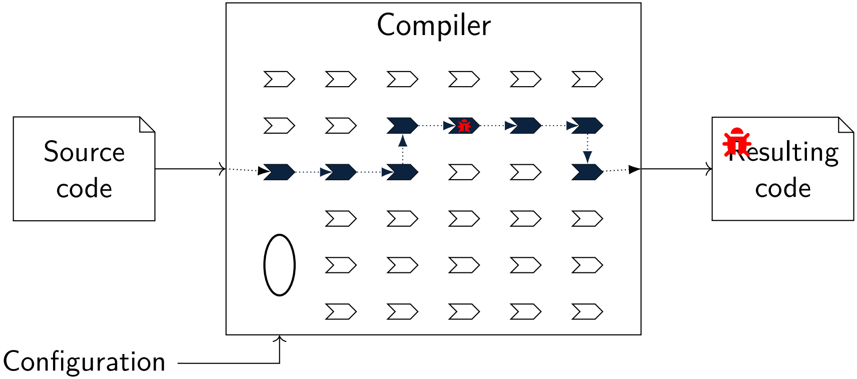


Insufficient debugging tools

1st use-case

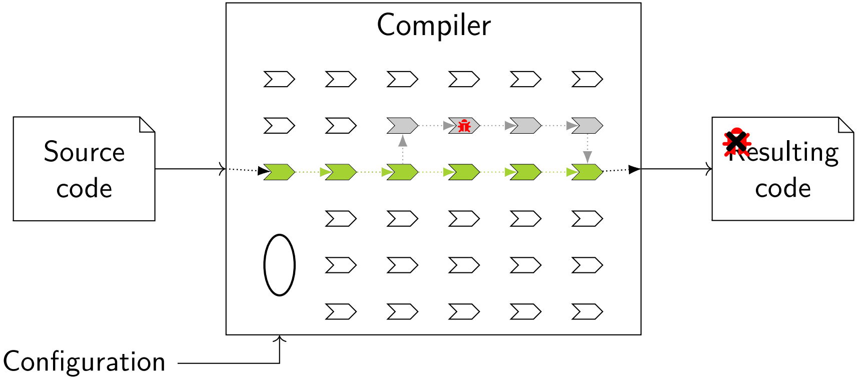
Insufficient debugging tools

1st use-case



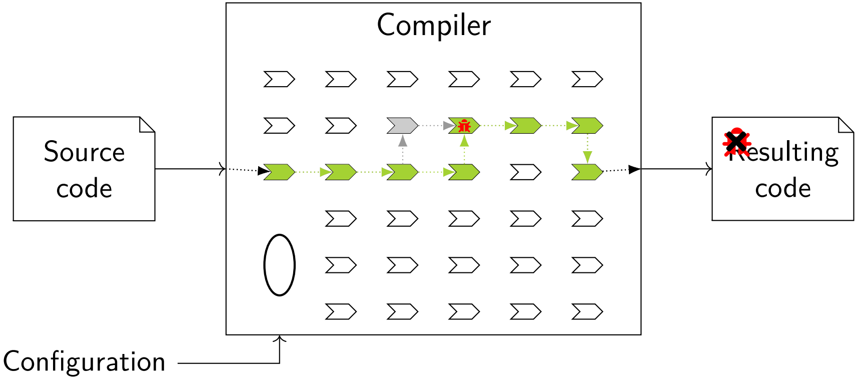
Insufficient debugging tools

1st use-case



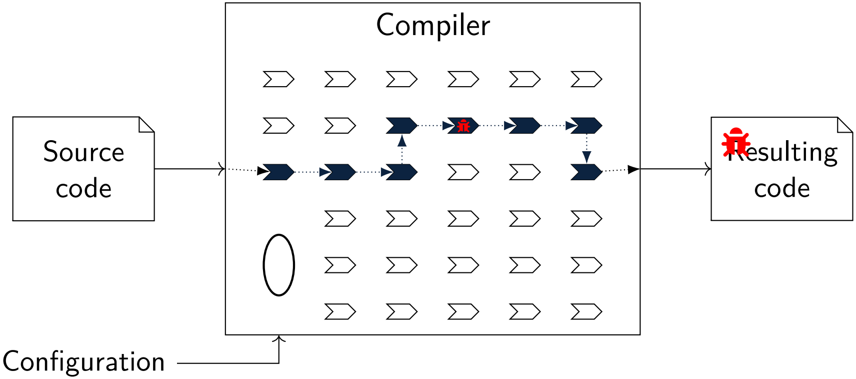
Insufficient debugging tools

1st use-case



Insufficient debugging tools

1st use-case

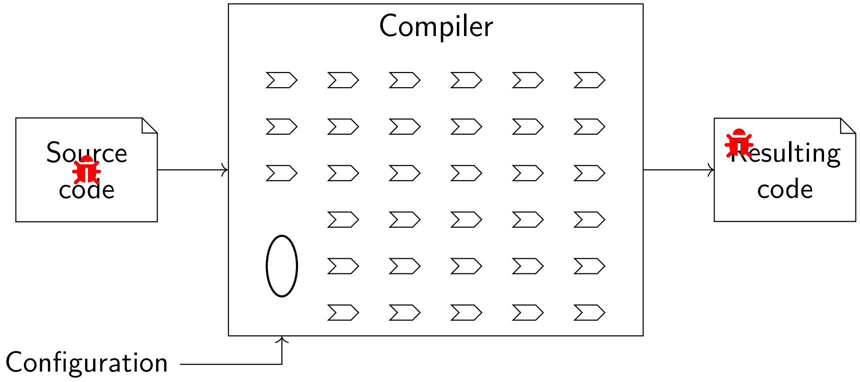


Debugging tools

- Debug information (DWARF)
- Debug outputs produced during transformations

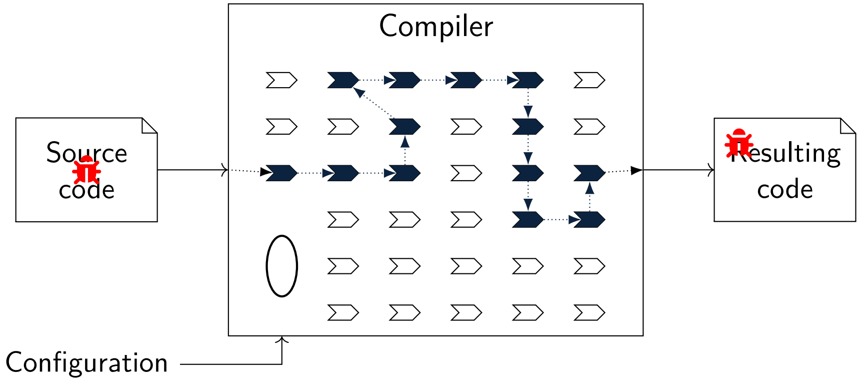
Insufficient debugging tools

2nd use-case



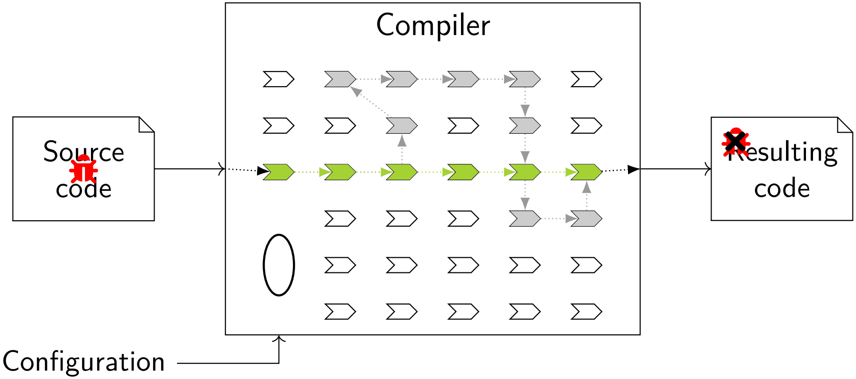
Insufficient debugging tools

2nd use-case



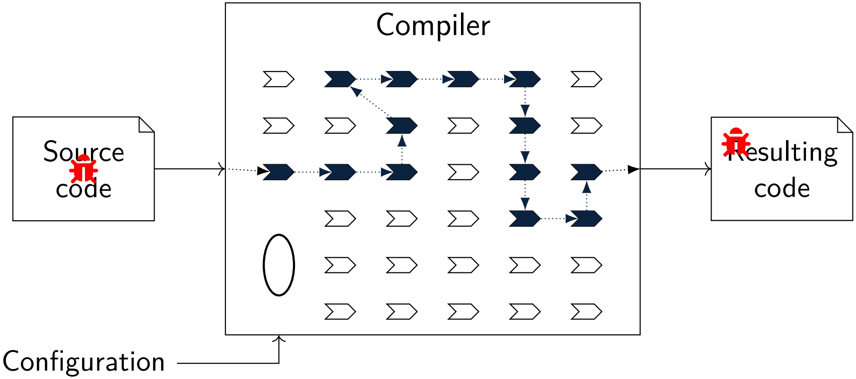
Insufficient debugging tools

2nd use-case



Insufficient debugging tools

2nd use-case



Debugging tools

- Debug information (DWARF)
- Common debugging techniques, e.g. bissection

Insufficient debugging tools
Summary

Generic debugging tools

- Help compiler users to debug their code
- Hard to use for compilers

Insufficient debugging tools
Summary

Generic debugging tools

- Help compiler users to debug their code
- Hard to use for compilers

Towards specialized compiler debugging tools

- Offer higher-level, filtered information
- Aware of the compiler process
- **Understand the transformations that the compiler is doing**

Work in progress

Goal: Tracing high-level transformations

- This multiplication has been replaced by this optimized version that use bitwise shift
- This function has been inlined at these three places

Work in progress

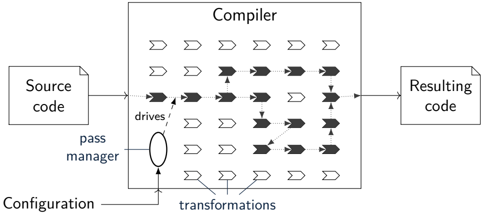
Goal: Tracing high-level transformations

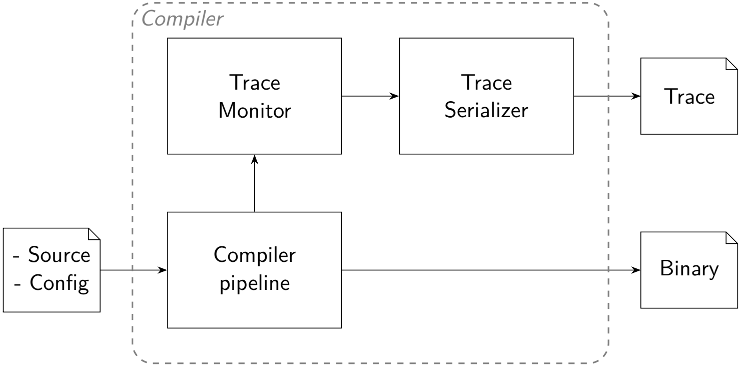
- This multiplication has been replaced by this optimized version that use bitwise shift
- This function has been inlined at these three places

Current status: Tracing atomic transformations

- The 'Instruction' `add nsw i32 %201 %202` has been created with identifier 203
- All occurences of 'Instruction' 203 have been replaced by the 'Instruction' 210

Trace monitor
Current state of the trace API





3 • Conclusions

Quarkslab
Securing every bit of your data

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Conclusions

Debugging compilers is hard

- Compiler-specific issues
- Benefits of specialized tools
- Poorly studied

Compiler traces

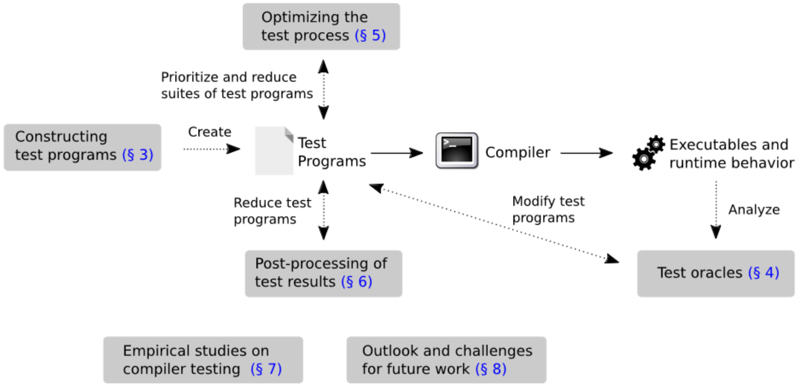
- Proposition to help compiler developers to understand a compilation process
- Could help for debugging...
- ...but also maybe useful in other situations, like certification

Bibliography

- Chen, Junjie et al. "A Survey of Compiler Testing". In: *ACM Computing Surveys* 53.1 (Feb. 2020), pp. 1–36. DOI: 10.1145/3363562.
- IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. 1990. DOI: 10.1109/ieeestd.1990.101064.
- Gotel, Orlena et al. "Traceability Fundamentals". In: *Software and Systems Traceability*. Springer London, Oct. 2011, pp. 3–22. DOI: 10.1007/978-1-4471-2239-5_1.

Context

Classification of compiler testing studies


Optimizing the test process (§ 5)
Prioritize and reduce suites of test programs
Constructing test programs (§ 3)
Create
Test Programs
Compiler
Executables and runtime behavior
Reduce test programs
Modify test programs
Analyze
Post-processing of test results (§ 6)
Test oracles (§ 4)
Empirical studies on compiler testing (§ 7)
Outlook and challenges for future work (§ 8)

Chen et al., “A Survey of Compiler Testing”

Goals
Traceability

Traceability within compilers

- Create a traceability framework
- Not dedicated to a specific usage
- Implemented in LLVM, but designed with a global approach

Goals
Traceability

Traceability within compilers

- Create a traceability framework
- Not dedicated to a specific usage
- Implemented in LLVM, but designed with a global approach

Trace is optional

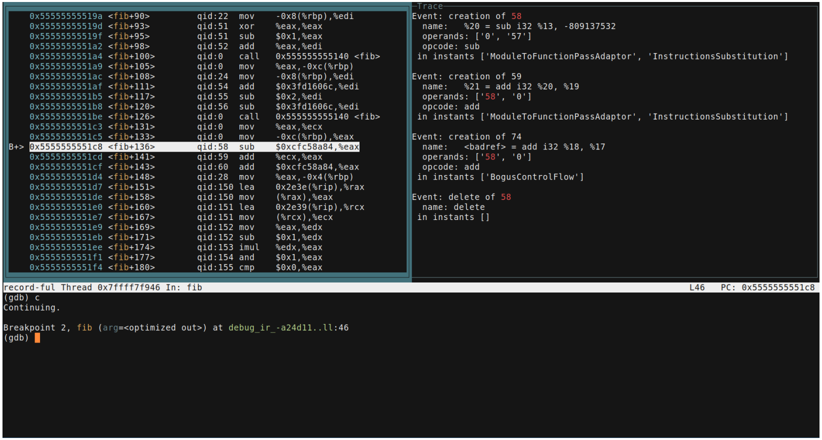
- Enable and disable it on demand
- Partial traces must be useful
- No need to implement trace features in every compiler pass to produce useful data

5 • Existing work about traceability

Quarkslab
Securing every bit of your data

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Using the trace



```
    0x55555555519a <fib+90>          qid:22  mov    -0x8(%rbp),%edi
    0x55555555519d <fib+93>          qid:51  xor    %eax,%eax
    0x55555555519f <fib+95>          qid:51  sub    $0x1,%eax
    0x5555555551a2 <fib+98>          qid:52  add    %eax,%edi
    0x5555555551a4 <fib+100>         qid:0   call   0x555555555140 <fib>
    0x5555555551a9 <fib+105>         qid:0   mov    %eax,-0xc(%rbp)
    0x5555555551ac <fib+108>         qid:24  mov    -0x8(%rbp),%edi
    0x5555555551af <fib+111>         qid:54  add    $0x3fd1606c,%edi
    0x5555555551b5 <fib+117>         qid:55  sub    $0x2,%edi
    0x5555555551b8 <fib+120>         qid:56  sub    $0x3fd1606c,%edi
    0x5555555551be <fib+126>         qid:0   call   0x555555555140 <fib>
    0x5555555551c3 <fib+131>         qid:0   mov    %eax,%ecx
    0x5555555551c5 <fib+133>         qid:0   mov    -0xc(%rbp),%eax
B+> 0x5555555551c8 <fib+136>         qid:58  sub    $0xcfc58a84,%eax
    0x5555555551cd <fib+141>         qid:59  add    %ecx,%eax
    0x5555555551cf <fib+143>         qid:60  add    $0xcfc58a84,%eax
    0x5555555551d4 <fib+148>         qid:28  mov    %eax,-0x4(%rbp)
    0x5555555551d7 <fib+151>         qid:150 lea    0x2e3e(%rip),%rax
    0x5555555551de <fib+158>         qid:150 mov    (%rax),%eax
    0x5555555551e0 <fib+160>         qid:151 lea    0x2e39(%rip),%rcx
    0x5555555551e7 <fib+167>         qid:151 mov    (%rcx),%ecx
    0x5555555551e9 <fib+169>         qid:152 mov    %eax,%edx
    0x5555555551eb <fib+171>         qid:152 sub    $0x1,%edx
    0x5555555551ee <fib+174>         qid:153 imul   %edx,%eax
    0x5555555551f1 <fib+177>         qid:154 and    $0x1,%eax
    0x5555555551f4 <fib+180>         qid:155 cmp    $0x0,%eax
```

```
-Trace-
Event: creation of 58
  name:   %20 = sub i32 %13, -809137532
  operands: ['0', '57']
  opcode: sub
 in instants ['ModuleToFunctionPassAdaptor', 'InstructionsSubstitution']

Event: creation of 59
  name:   %21 = add i32 %20, %19
  operands: ['58', '0']
  opcode: add
 in instants ['ModuleToFunctionPassAdaptor', 'InstructionsSubstitution']

Event: creation of 74
  name:   <badref> = add i32 %18, %17
  operands: ['58', '0']
  opcode: add
 in instants ['BogusControlFlow']

Event: delete of 58
  name: delete
 in instants []
```

```
record-ful Thread 0x7ffff7f946 In: fib                                                    L46   PC: 0x5555555551c8
(gdb) c
Continuing.

Breakpoint 2, fib (arg=<optimized out>) at debug_ir_-a24d11..ll:46
(gdb) 
```

Definitions of Trace

Debugging traces

"A trace is a record of the execution of a computer program, showing the sequence of instructions executed, the names and values of variables, or both."[1]

[1] IEEE, *IEEE Standard Glossary of Software Engineering Terminology*.
[2] Gotel et al., "Traceability Fundamentals".

Definitions of Trace

Debugging traces

"A trace is a record of the execution of a computer program, showing the sequence of instructions executed, the names and values of variables, or both."[1]

More generally

"Traceablity is the potential to relate data 2 can be separate file or included in binary that is stored within artifacts of some kind, along with the ability to examine this relationship"[2]

[1] IEEE, *IEEE Standard Glossary of Software Engineering Terminology*.

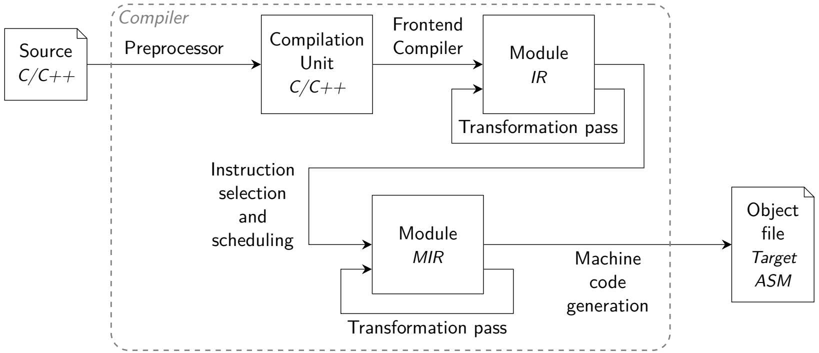
[2] Gotel et al., "Traceability Fundamentals".

Costs

Lua		88 kB source, 300 kB compiled			
Options		-O0	-O1	-O2	-O3
time (s)	Clang-14	1.16	3.81	4.24	4.42
	Clang-15 patched	17.00	22.36	23.44	23.70
Trace size (MB)		5.80	43.51	48.53	50.04

keepassxc		9.3 MB source, 6.9 MB compiled			
Options		-O0	-O1	-O2	-O3
time (s)	Clang-14	327.55	371.96	377.75	384.157
	Clang-15 patched	6302.95		7790.97	7721.80
Trace size (MB)		38		250	

LLVM Metamodel



Trace specification

Existing concepts

- Artifacts: The IR at a given stage of the compilation process
- Trace links: Called events in my case

Trace specification

Existing concepts

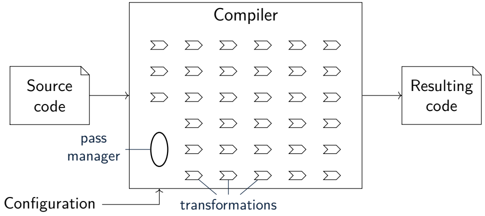
- Artifacts: The IR at a given stage of the compilation process
- Trace links: Called events in my case

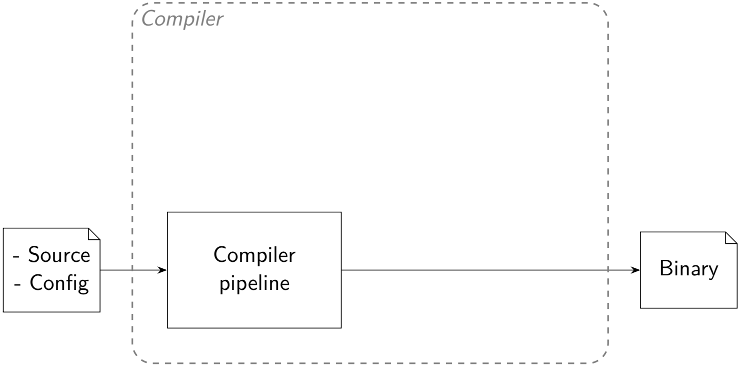
Instant: Timeline information

- Has a start and an end
- Describes a time window of the compilation process
- Can be nested

Trace monitor

Current state of the trace API

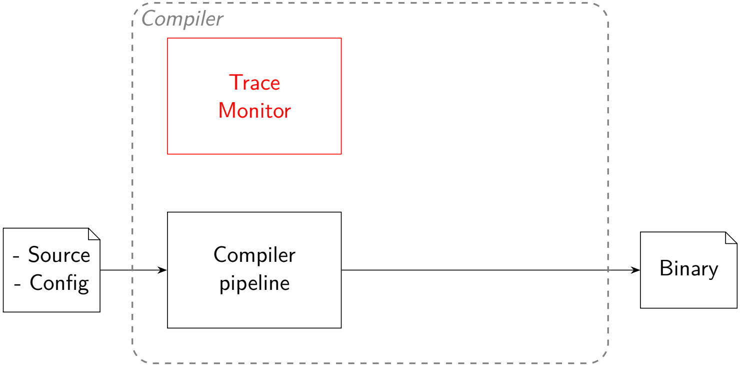




Trace monitor
Current state of the trace API

Trace Monitor

- Inside LLVM core
- API to register *Instants* and *Events*
- Accessible from anywhere

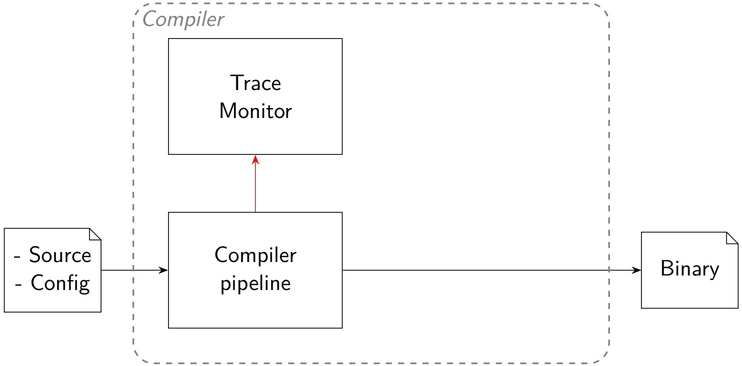


Trace monitor

Current state of the trace API

Integration with LLVM codebase

- Modifications to LLVM to use trace *events* and *instants*
- New *events* and *instants* types can be created to enrich the trace

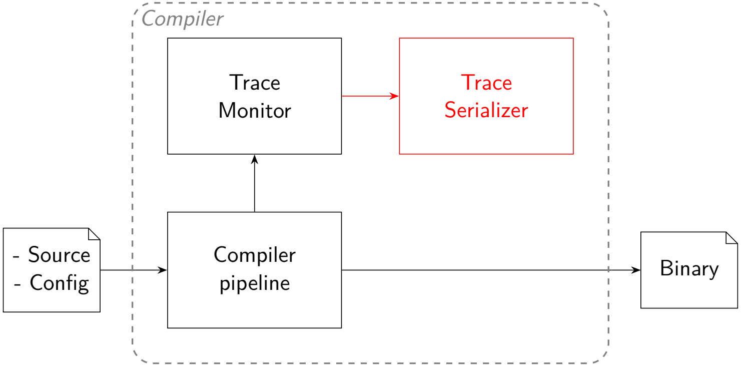


Trace monitor

Current state of the trace API

Serializer

- No pre-analysis is done by the serializer
- Easily parseable by external tools

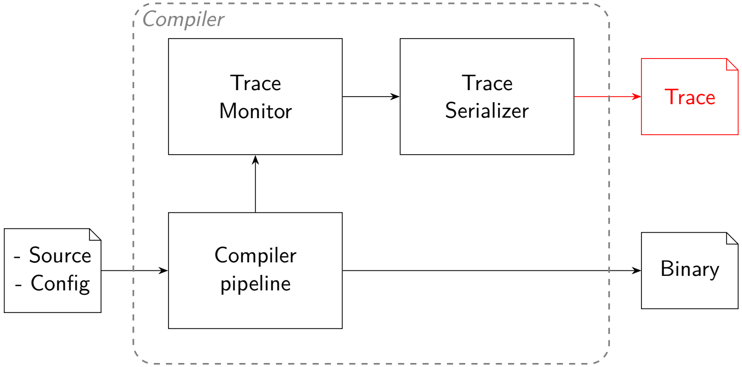


Trace monitor

Current state of the trace API

Serializer

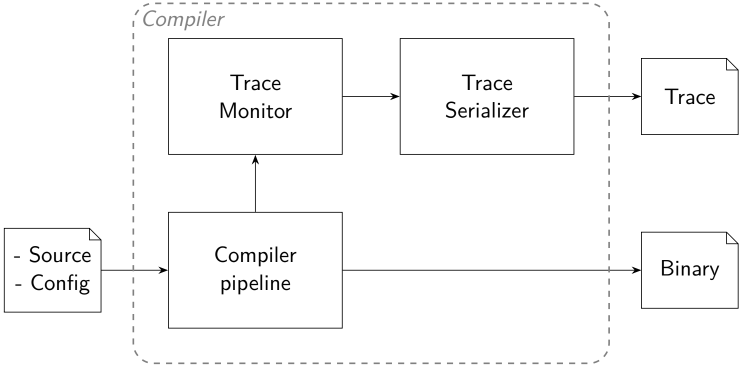
- No pre-analysis is done by the serializer
- Easily parseable by external tools



Trace monitor
Current state of the trace API

Link with the binary

- Binary and Trace are separate artifacts
- Each *Value* is uniquely identified in the trace


Compiler
Trace Monitor
Trace Serializer
Trace
- Source
- Config
Compiler pipeline
Binary