Practical Automation of any Bug Avoidance Strategy in the Engineering of Safety Critical Software

Oliver Schneider

Hubert B. Keller

Veit Hagenmeyer

Karlsruhe Institute of Technology

Overview

- Software Engineering Best Practices
- Actual Practices
- Road Blocks
- Incremental Introduction

Best Practices

- Tooling
- Standards and Guidelines
- Documentation
- Requirements tracing

Actual Practices

- Instruction Lists
- Ad-hoc non-uniform Guidelines
- Only sporadic Documentation
- No static analyses
- Little automated testing

Road Blocks

- Social
  - Resistance
  - Training
- Technological
  - Expensive
  - Transition difficulties

Incremental Introduction

- Improve by doing small steps
- Project developers also write analyses
- Only add analyses with immediate gains

Analysis API

- Many compilers expose APIs
  - clang (C++)
  - gcc (C++)
  - rustc (Rust)
  - Ada (ASIS, libadalang)
  - ...

Building a compiler from scratch

```
pub fn main() {
    let args: Vec<_> = std::env::args().collect();
    rustc_driver::run(move || {

        Your own code goes here

        rustc_driver::run_compiler(&args, Box::new(compiler), None, None)
    });
}
```

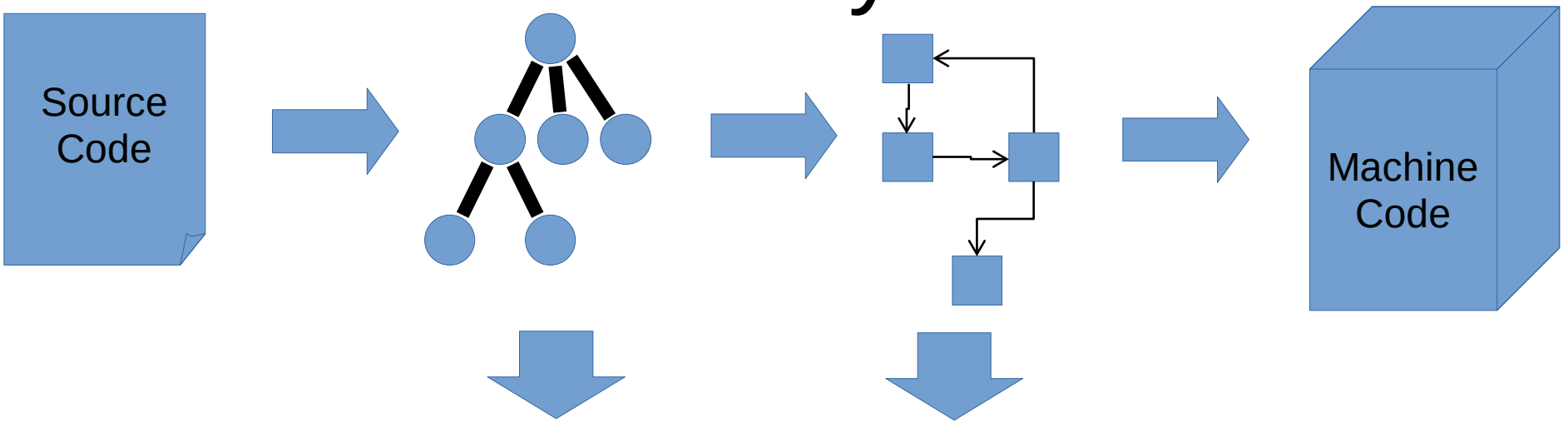
Building a compiler from scratch

```
pub fn main() {
    let args: Vec<_> = std::env::args().collect();
    rustc_driver::run(move || {
        let mut compiler = driver::CompileController::basic();
        compiler.after_parse.callback = Box::new(move |state| {
            let mut ls = state.session.lint store.borrow mut();
            Your own analyses go here
        });
        rustc_driver::run_compiler(&args, Box::new(compiler), None, None)
    });
}
```

Building a compiler from scratch

```
pub fn main() {
    let args: Vec<_> = std::env::args().collect();
    rustc_driver::run(move || {
        let mut compiler = driver::CompileController::basic();
        compiler.after_parse.callback = Box::new(move |state| {
            let mut ls = state.session.lint_store.borrow_mut();
            ls.register_early_pass(None, false, box NoTransmute);
            ls.register_late_pass(None, false, box Pass);
        });
        rustc_driver::run_compiler(&args, Box::new(compiler), None, None)
    });
}
```

How do analyses work?



Memory Safety
Modularity
Maintainability
Style

Dataflow
Information Flow
Privacy
Security

Simple example

- MISRA forbids `a && (b && c)`
- AND
  - A
  - PARENTHESES
    - AND
      - B
      - C

Simple example

- MISRA forbids `a && (b && c)`
- AND
  - A
  - PARENTHESES
    - AND
      - B
      - C

Simple Example

```
if let ExprKind::Parens(ref inner) = expr.node;
if let ExprKind::ExprBinary(ref op, ref left, ref right) = inner.node;
if BinOpKind::And == op.node;
```

The above code is the only code unique for this analysis
All other code is the same for any analysis and thus abstracted away

Simple Example

- Writing simple analyses is time consuming but not hard
- Perfect candidate for automation

```
#[clippy::author]
if a && (b && c) {
    // some code here
}
```

Simple Example - Autogenerated

```
if let ExprKind::If(ref cond, ref then, None) = expr.node;
if let ExprKind::Binary(ref op, ref left, ref right) = cond.node;
if BinOpKind::And == op.node;
if let ExprKind::Path(ref path) = left.node;
if match_qpath(path, &["a"]);
if let ExprKind::Parens(ref inner) = right.node;
if let ExprKind::Binary(ref op1, ref left1, ref right1) = inner.node;
if BinOpKind::And == op1.node;
if let ExprKind::Path(ref path1) = left1.node;
if match_qpath(path1, &["b"]);
if let ExprKind::Path(ref path2) = right1.node;
if match_qpath(path2, &["c"]);
if let ExprKind::Block(ref block1) = then.node;
```

Critique

- Developers have enough to do already
- Writing compiler extensions is hard
- Compiler APIs change and break analyses
- “Not good enough”

Critique – busy developers

Developers are frequently busy with

- Fixing nearly identical issues
- Teaching interns, new hires and trainees
- Looking up guideline/standard rules

Proposal: automate these tasks

Critique – compilers are hard

- Menial tasks are already automated
- Configuring style/guideline/standard checkers has a similar difficulty level
- Anecdotal evidence suggests otherwise
  - Beginners at compilers and Rust write analyses within a day

Critique – (lack of) API stability

- Ask yourself
  - How frequently do you update compilers?
  - How often does a compiler update break your code?
- Rust and Go compilers automatically update your code

Critique - insufficient

- This approach does not provide any “proof” of correctness
- But
  - Immediately applicable
  - Incremental!
  - Provide a sane platform for proofs
    - Proofs usually require code to be in a specific format

Conclusion

- Applicable **now**
- Possible in **many languages**
- Forward compatible to proving correctness
- **Incrementally** move towards proofs
- For hobbyist beginners and professionals alike