



Technische
Universität
Braunschweig



Institute of Operating Systems
and Computer Networks

Reliable System Software

Towards a Vertically Integrated Compiler Infrastructure to Improve Solver Time during Symbolic Execution

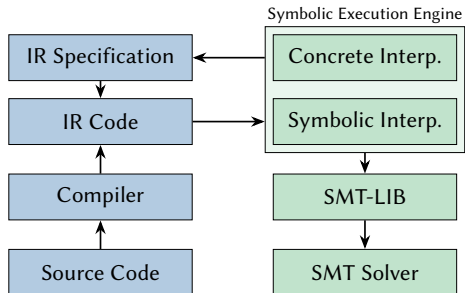
Sören Tempel, Christian Dietrich

tempel@ibr.cs.tu-bs.de

Challenge: Established SE engines build upon off-the-shelf compilers

↳ These compilers optimize for fast execution on physical hardware

- Do not optimize for fast formal reasoning!
- SE generates SMT-LIB from the compiler's IR
- Representation in IR affects emitted SMT-LIB

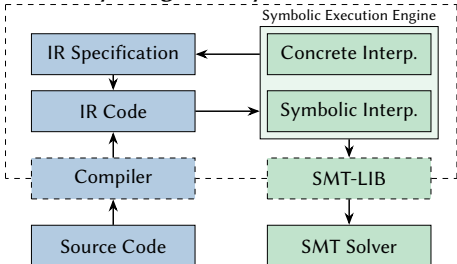


Idea: Closer integration of symbolic execution and compilers

↳ Tailor IR and compiler optimization passes to SE!

- Contrib: Qute, a vertically integrated SE stack
- Qute is build around QBE, an extensible IR
- Provides a symbolic executor for the QBE IR
- And facilities to build QBE compiler passes

Vertically Integrated Symbolic Execution



```
char getsymbol(uint8_t code) {  
    if (code <= CAPITAL_BOUND) {  
        return (code + 'A');  
    } else if (code <= SMALL_BOUND) {  
        return (code + ('z' - SMALL_BOUND));  
    } else if (code <= NUMBER_BOUND) {  
        return (code + ('9' - NUMBER_BOUND));  
    }  
  
    return (char)NOT_DEFINED;  
}
```

```
char getsymbol(uint8_t code) {
    if (code <= CAPITAL_BOUND) {
        return (code + 'A');
    } else if (code <= SMALL_BOUND) {
        return (code + ('z' - SMALL_BOUND));
    } else if (code <= NUMBER_BOUND) {
        return (code + ('9' - NUMBER_BOUND));
    }

    return (char)NOT_DEFINED;
}
```

Desired
SMT-LIB



```
(ite (bvule code CAPITAL_BOUND)
    (bvadd code ASCII_A)
    (ite (bvule code SMALL_BOUND)
        (bvadd
            code
            (bvsub ASCII_z SMALL_BOUND))
        (ite (bvule code NUMBER_BOUND)
            (bvadd
                code
                (bvsub ASCII_9 NUMBER_BOUND))
            NOT_DEFINED)))
```

Problem: KLEE does not generate this program representation

↳ As LLVM represents the program using multiple branches

```
char getsymbol(uint8_t code) {
    if (code <= CAPITAL_BOUND) {
        return (code + 'A');
    } else if (code <= SMALL_BOUND) {
        return (code + ('z' - SMALL_BOUND));
    } else if (code <= NUMBER_BOUND) {
        return (code + ('9' - NUMBER_BOUND));
    }

    return (char)NOT_DEFINED;
}
```

Desired
SMT-LIB

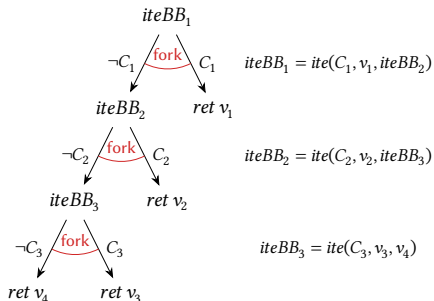


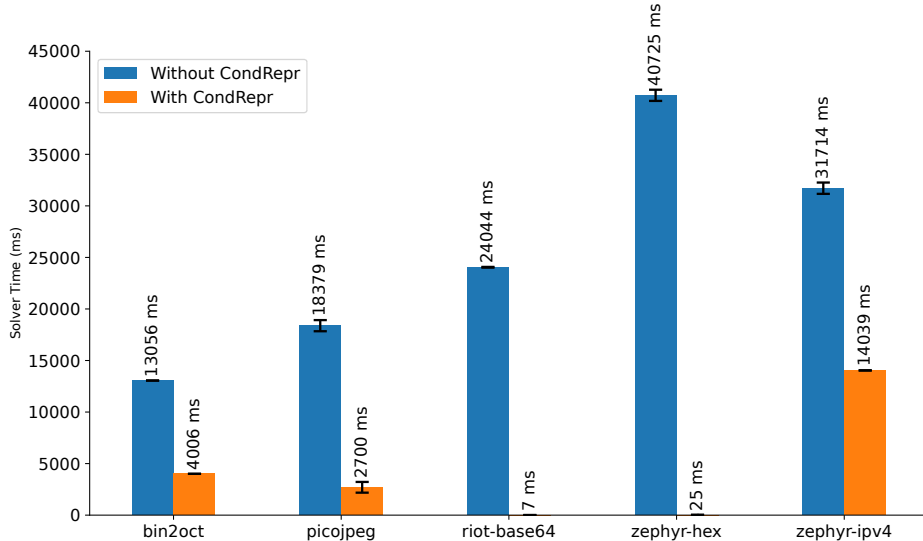
```
(ite (bvule code CAPITAL_BOUND)
    (bvadd code ASCII_A)
    (ite (bvule code SMALL_BOUND)
        (bvadd
            code
            (bvsub ASCII_z SMALL_BOUND))
        (ite (bvule code NUMBER_BOUND)
            (bvadd
                code
                (bvsub ASCII_9 NUMBER_BOUND))
            NOT_DEFINED)))
```

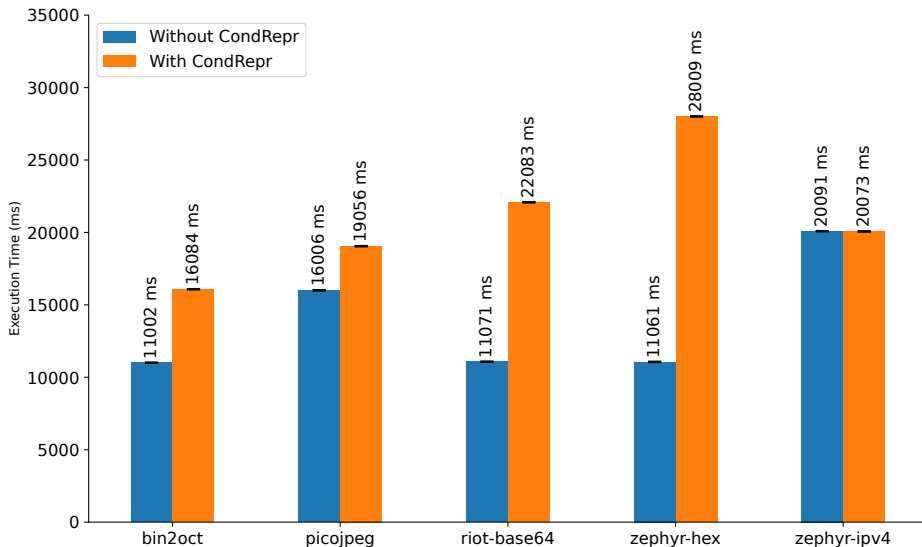
Approach: Add a custom `iteBB` exit point instruction to our IR

↳ Evaluates two basic blocks, “selects” the result of one of them

- `iteBB` is an alternative to conditional jumps
- Denotes that branches have no side effects
- Can be mapped directly to SMT-LIB’s `ite`







Key Insights:

1. Co-Design of IR and SE has potential for solver time savings
2. Conflict wrt. optimization goals of concrete/symbolic execution

Key Insights:

1. Co-Design of IR and SE has potential for solver time savings
2. Conflict wrt. optimization goals of concrete/symbolic execution

Future Work:

1. Experiment with additional IR enhancements
2. Mitigate conflicting concrete/symbolic optimization goals



Technische
Universität
Braunschweig



Institute of Operating Systems
and Computer Networks
Reliable System Software

Towards a Vertically Integrated Compiler Infrastructure to Improve Solver Time during Symbolic Execution

Sören Tempel, Christian Dietrich

tempel@ibr.cs.tu-bs.de