

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

Sören Tempel <tempel@ibr.cs.tu-bs.de>, Christian Dietrich <dietrich@ibr.cs.tu-bs.de>

Motivation

- **Context:** Established symbolic execution engines (e.g., KLEE) build upon off-the-shelf compiler infrastructures, such as LLVM.
- **Challenge:** These existing compilers optimize for fast execution on real hardware, not for fast formal reasoning using a solver.
- **Approach:** We strive for a closer integration of these components to improve SMT-LIB encoding and thus reduce SMT solver time.

Contributions

1. **QUTE:** An extensible symbolic execution engine for a modular IR, which is thus designed specifically for a co-design methodology.
2. **Conditional Representation:** A novel IR representation for solver-efficient SMT encoding of pure functions w/o side effects.

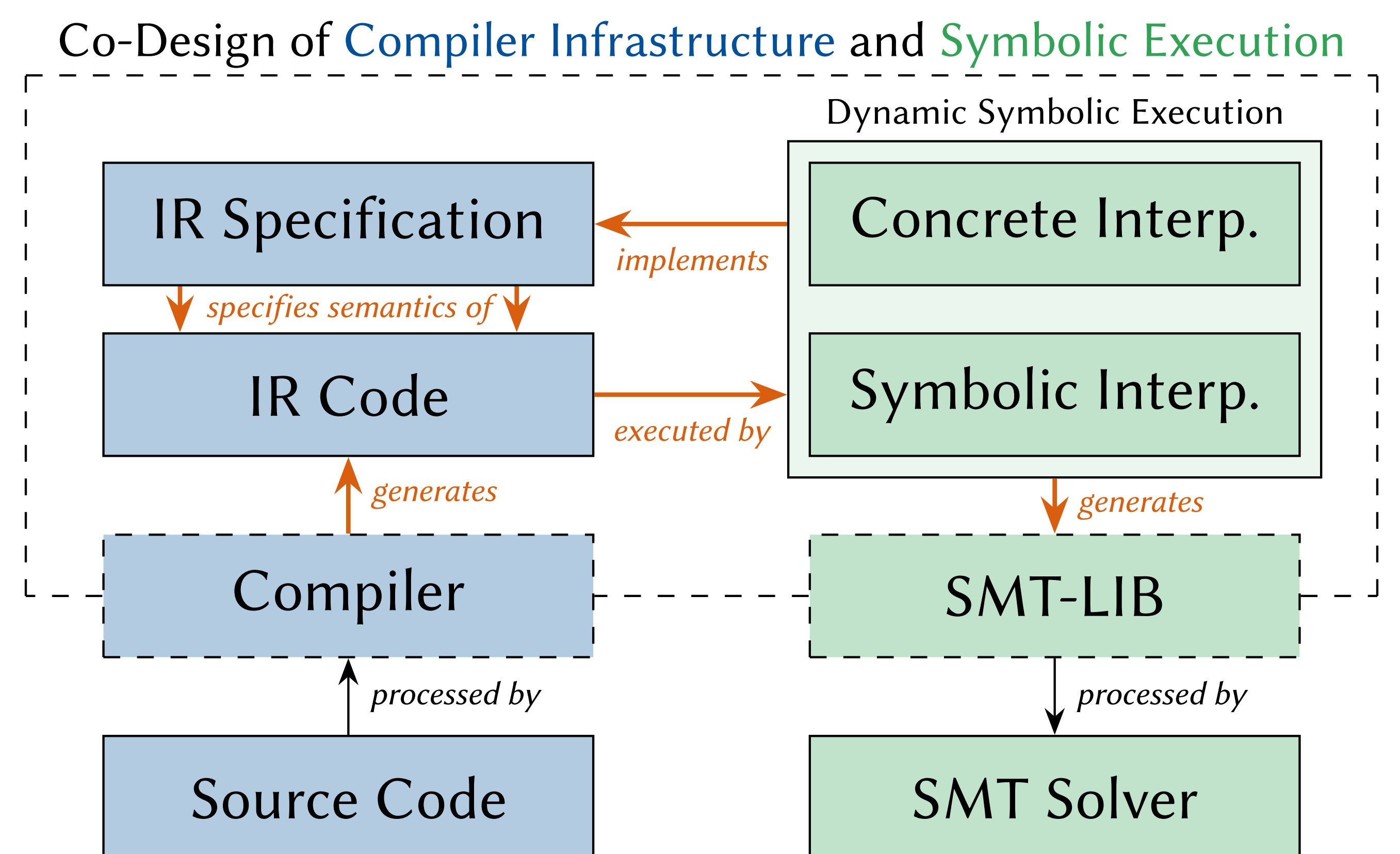


Figure 1: Proposed co-design approach.

Case Study: ITE Expressions

Problem Statement: Clang commonly represents if-statements as branches in LLVM IR.

→ KLEE generates SMT-LIB from LLVM IR, does thus not use ITE expressions in the SMT-LIB!

```
char getsymbol(uint8_t code) {
    if (code <= BASE64_CAPITAL_BOUND) { // C1
        return (code + 'A');
    }

    if (code <= BASE64_SMALL_BOUND) { // C2
        return (code + ('z' - BASE64_SMALL_BOUND));
    }

    if (code <= BASE64_NUMBER_BOUND) { // C3
        return (code + ('9' - BASE64_NUMBER_BOUND));
    }

    return (char)BASE64_NOT_DEFINED;
}
```

Listing 1: Pure C function w/o side effects.

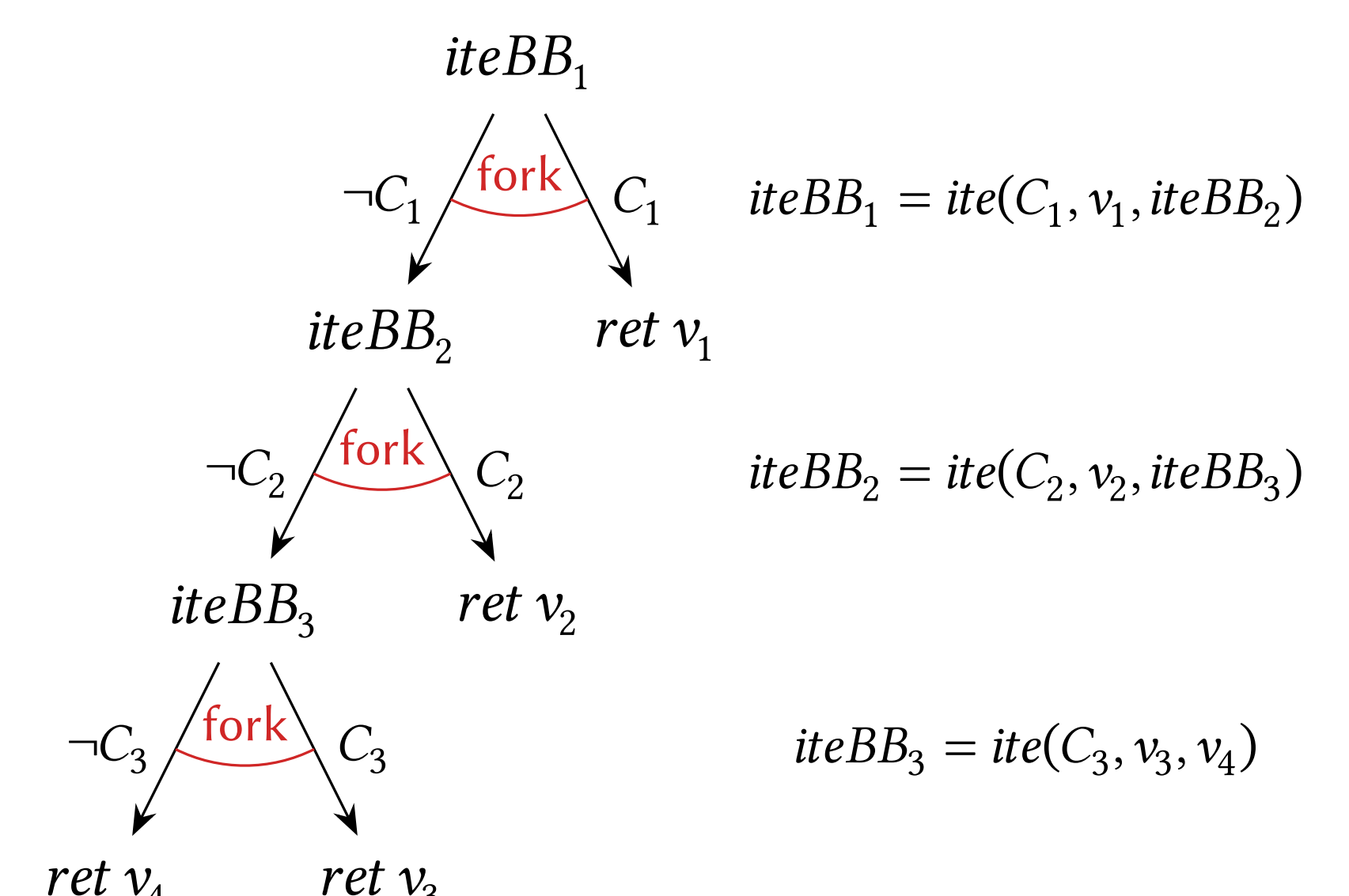
```
(define-fun getsymbol ((code (_ BitVec 8))) (_ BitVec 8)
  ;; C1: code <= BASE64_CAPITAL_BOUND
  (ite (bvule code BASE64_CAPITAL_BOUND)
    (bvadd code ASCII_A)
    ;; C2: code <= BASE64_SMALL_BOUND
    (ite (bvule code BASE64_SMALL_BOUND)
      (bvadd
        code
        (bvsub ASCII_z BASE64_SMALL_BOUND))
      ;; C3: code <= BASE64_NUMBER_BOUND
      (ite (bvule code BASE64_NUMBER_BOUND)
        (bvadd
          code
          (bvsub ASCII_9 BASE64_NUMBER_BOUND))
        BASE64_NOT_DEFINED))))
```

Listing 2: Desired SMT-LIB representation.

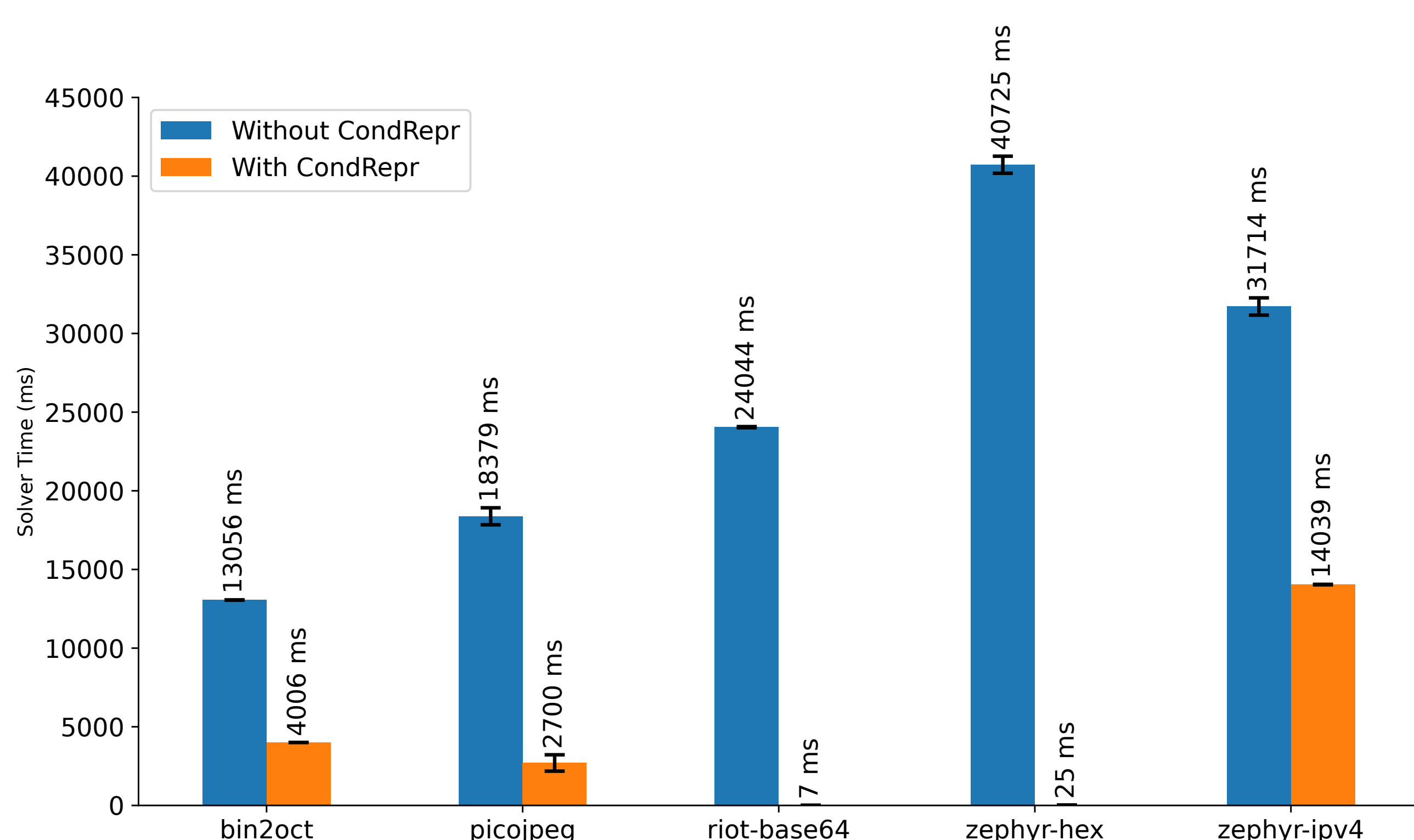
Proposal: CondRepr.

Adds custom `iteBB` exit point IR instruction:

- Eagerly evaluate both BB jump labels.
- Conditionally select the result of a BB.



Evaluation: Impact on Solver Time



Conclusion

Key Insights:

- Co-design of IR and SE enables potential solver-time savings.
- CondRepr reduces query amount but increases their complexity.

Limitations:

- **Concrete Execution:** While solver time is reduced by 80%, concrete execution is increased by a geometric mean of roughly 16%.
- **Compiler Transformation:** The `iteBB` compiler optimization only instruments entire functions that are pure and free of loops.

More Information:

- Further details are available in a publication that is to appear in the proceedings of the ASYDE'26 workshop; scan the QR code!
- Our OSS prototype and artifacts are referenced in the paper.

