LESS LAB

Ensuring Critical Properties of Test Oracles for Effective Bug Detection

Soneya Binta Hossain

University of Virginia

Motivation

- Software bugs can lead to security vulnerability, system outage, even potential loss of life
- Effective testing and bug detection mechanisms are crucial for ensuring software reliability
- **Test oracles** are key to effective software testing
- Developer-written test oracles are effective but expensive
- Automated test oracles are cost-effective; however, existing methods suffer from inadequate checks, high false positive rates, and poor bug detection effectiveness **[1]**

Definition and Example of Test Oracles

Test Oracle: Given an input for a system, a test oracle (or just oracle) is a procedure that distinguishes between the correct and incorrect behaviors of the System Under Test (SUT)

Assertion Oracle: Checks the program output against expected output

```
public void testPushAndPopStack() {

    Stack<Integer> stack = new Stack<>();
    stack.push(10);
    Integer val = stack.pop();
    assertEquals(Integer.valueOf(10), val);
}
```

Figure 1: Test With Assertion Oracle

Exception Oracle: Checks that the erroneous input states are detected by the SUT

```
public void testPopEmptyStack() {

    Stack<Integer> stack = new Stack<>();
    try {
        stack.pop();
        fail("");
    } catch (EmptyStackException e) {
        // Test passed
    }
}
```

Figure 2: Test With Exception Oracle

Assessing and Improving Oracle Checking

In this research, we identify gaps in the existing test suite and propose a recommender system to help mitigate these gaps [5]

Evaluation: 13 large-scale Java projects, 248K SLOC, 237K test SLOC, 16K test cases and 51.6K assertions

```
final class GJDayOfWeekDateTimeField extends PreciseDurationDateTimeField {

   GJDayOfWeekDateTimeField(BasicChronology chronology, DurationField days) {
      /* writes iUnitMillies field */
      super(DateTimeFieldType.dayOfWeek(), days);
      iChronology = chronology; }
}

Recommendation:
   org.Joda.time.field.PreciseDurationDateTimeField.getUnitMillis

/* method from super class that reads iUnitMillis */
public final long getUnitMillis() {
   return iUnitMillis;
}
```

Findings:

- **34%** of the executed code are in the gap
- Larger gaps correlate with lower fault detection effectiveness
- Recommender recommends **67%** of the focal methods from developer-written tests
- Adding recommended assertions improved bug detection by an average of **13pp**, up to **58pp**

Conclusion

- Inspired from the **PIE** fault model, my research identified three critical properties of test oracles, called the **CCS (check, correct and strong) property** required for effective bug detection.
- Through large-scale studies, my research identified that developer-written and automated test oracles suffer from insufficient checks, high false positive rates and strength issue, resulting is poor bug detection effectiveness.
- To mitigate these issues, my research proposed **OracleGuru,** a comprehensive framework that can identify gaps, i.e., code executed but unchecked by test oracle and recommend additional oracles to check more code.
- My research identified that SOTA oracle generation method has significant accuracy issues. To mitigate this, my research proposed **TOGLL**, an LLM-based method for test oracle generation which significantly outperformed previous SOTA.
- In summary, **OracleGuru** significantly contributes toward effective bug detection, ensuring software reliability.

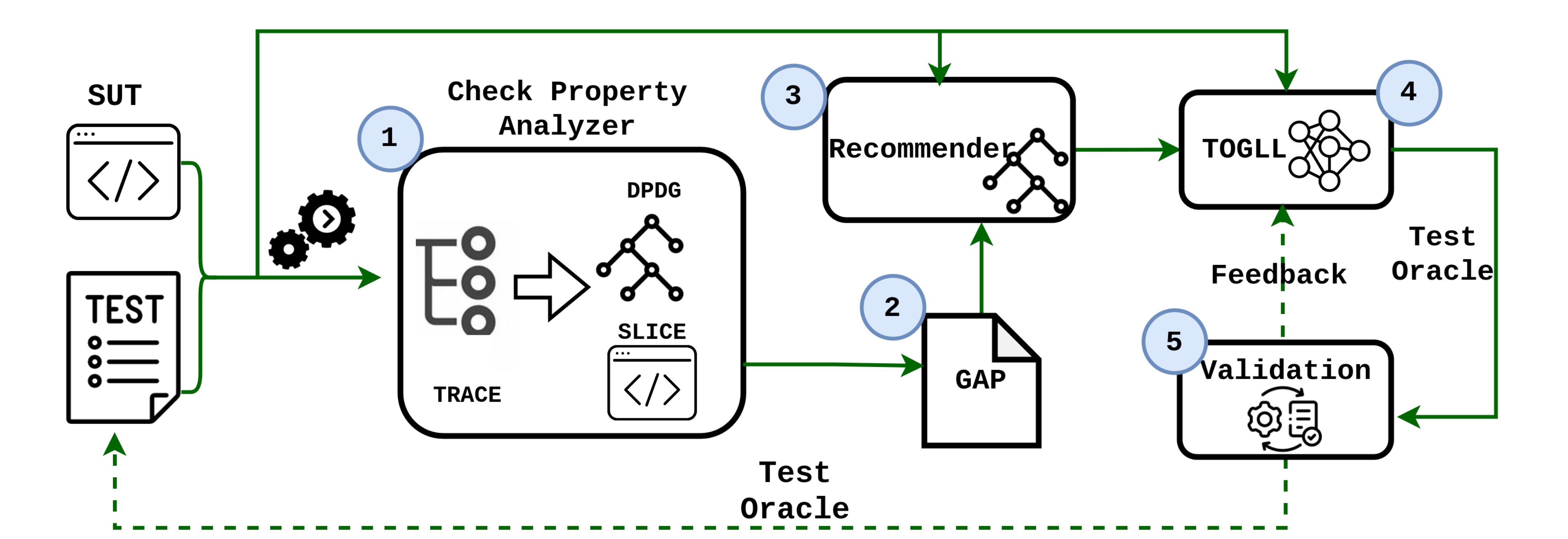
Limitation of Automated Test Oracles

```
public void testWithInsufficientCheck() {
  PFBuilder.Literal pFL0 = PFBuilder.Literal.EMPTY;
  Locale l0 = Locale.CHINA;
  Months m0 = Months.ZERO;
  int int0 = pFL0.calculatePrintedLength(m0, l0);
  // does not check int0
  assertEquals("", l0.getVariant());
}

public void testWithWeakAssertion() {
  Angle.Rad angle_Rad0 = Angle.Rad.PI;
  Angle.Rad angle_Rad1 = angle_Rad0.toRad();
  //check constant values
  assertEquals(1.57, Angle.PI_OVER_TWO, 0.01);
}

public void testWithIncorrectAssertion() {
  Stack<Integer> stack = new Stack<>();
  stack.push(10);
  Integer val = stack.pop() //buggy pop operation
  //incorrect regression oracle
  assertEquals(1, stack.size())
}
```

Contributions

- Investigating the extent to which test oracles check code and evaluating the influence of unchecked code on bug detection effectiveness
- Developing novel method for automatically enhancing test oracles so that they **check** more program behavior
- Large-scale evaluation of SOTA oracle generation methods to identify their limitations and areas for improvement
- Developing an LLM-based test oracle generation method **TOGLL**, bolstering the **correctness** and **strength** of test oracles
- Introducing the **OracleGuru** framework, comprising a suite of tools and methodologies geared towards ensuring the **CCS (check, correct, strong)** properties of test oracles.
- Publicly available datasets and artifacts [1,5]

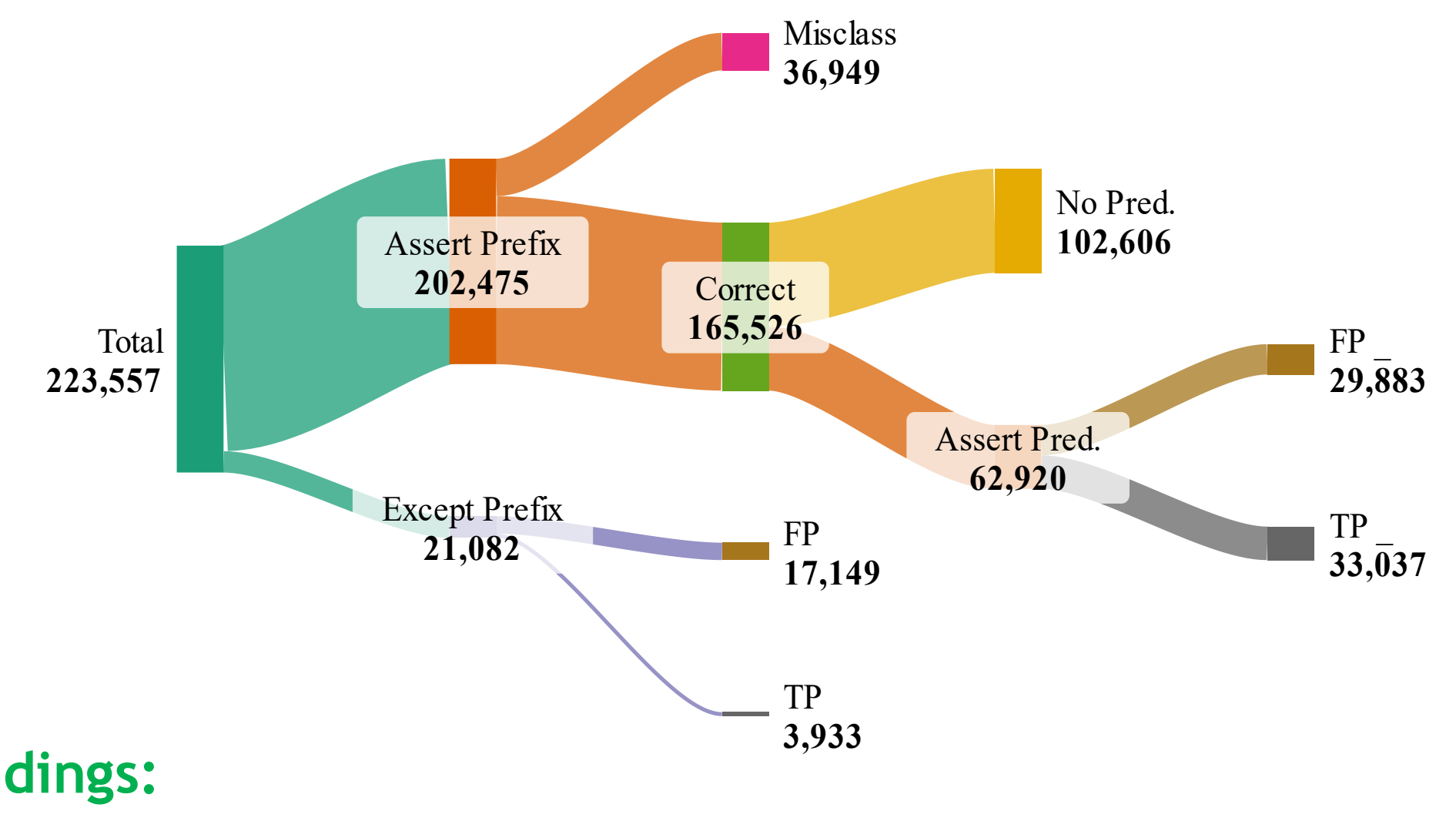
Overview of the OracleGuru Framework



Assessing Correctness and Strength

In this research [1], we evaluate the correctness and strength of the SOTA neural method for Test Oracle Generation (TOGA) [4]

Evaluation: 25 large-scale Java projects, 223,557 input samples, 51,385, EvoSuite [3], PIT



Findings:

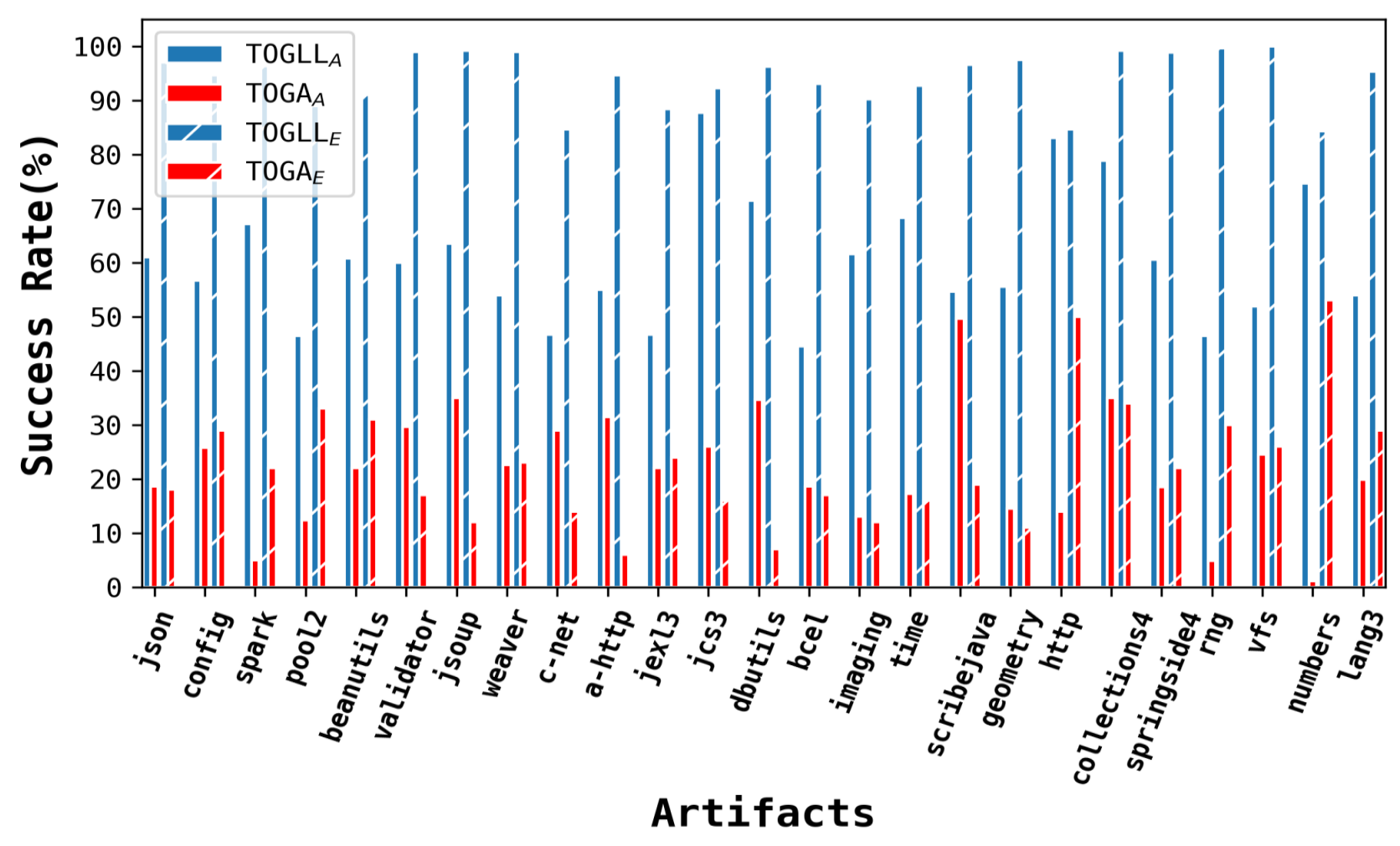
- SOTA neural method exhibits significant accuracy issues
- **81%** and **47%** incorrect exception and assertion oracle, **62%** no assertion generation rate
- Only **0.2%** additional unique bug detection w.r.t EvoSuite

Improving Correctness and Strength

In this research, we propose TOGLL, an LLM-based test oracle generation method

Evaluation:

- **Seven code LLMs :** CodeGPT, CodeParrot, CodeGen, PolyCoder, Phi-1, from 110M - 2.7B parameters
- **Two large datasets** (SF110, Apache Commons)
- **Six different prompts**



Findings:

- Fine-tuned LLMs can achieve up to **79%** accuracy in generating test oracles
- **TOGLL** generates significantly more correct test oracles than SOTA; **3.8x** and **4.9x** assertion oracles and exception oracles
- **TOGLL** generates assertions that are diverse with respect to both the assertion statement used and the variables and expressions targeted for observation in those assertions
- Detects **10x** times more unique bugs than previous SOTA

References

1. Hossain, Soneya Binta, et al. "Neural-Based Test Oracle Generation: A Large-Scale Evaluation and Lessons Learned." *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2023.
2. Barr, Earl T., et al. "The oracle problem in software testing: A survey." *IEEE transactions on software engineering* 41.5 (2014): 507-525.
3. Fraser, Gordon, and Andrea Arcuri. "EvoSuite: automatic test suite generation for object-oriented software." *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 2011.
4. Dinella, Elizabeth, et al. "Toga: A neural method for test oracle generation." *Proceedings of the 44th International Conference on Software Engineering*. 2022.
5. S. B. Hossain, M. B. Dwyer, S. Elbaum and A. Nguyen-Tuong, "Measuring and Mitigating Gaps in Structural Testing," *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, Melbourne, Australia, 2023, pp. 1712-1723, doi: 10.1109/ICSE48619.2023.00147.

Acknowledgement

Advised By: Dr. Matthew Dwyer

LOCKHEED MARTIN

DARPA

aws