Test Oracles

Outline

- AST vs bytecode Transformations
- Oracles
  - Safety, functional, regression
- Mutation Testing

Bytecode vs AST transformations

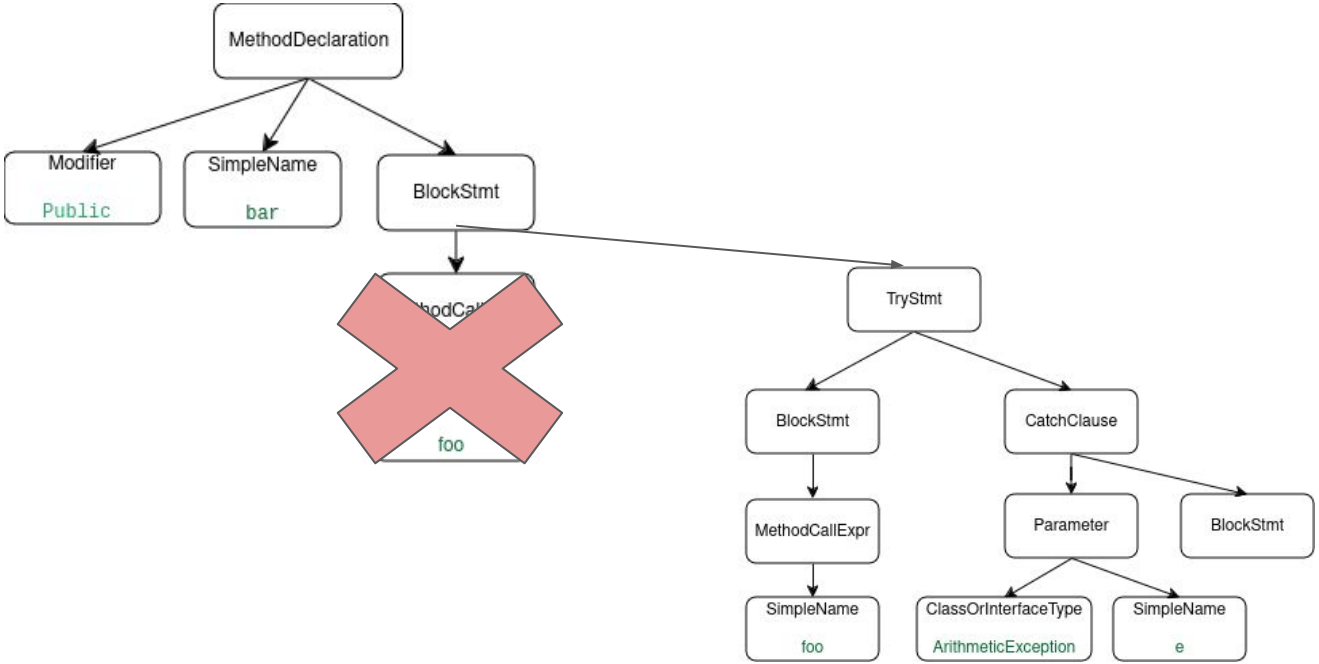
Transformation: Wrap all calls to a method `foo` in a try-catch block as precisely as possible.

Consider performing this transformation at both the **bytecode** and **AST** level:

- Come up with an example source and draw its representation
- Work through it by hand
- Consider error cases
- Which transformation do you prefer?

AST try-catch Transformation

```
public void bar(int x) {
  foo();
}
```

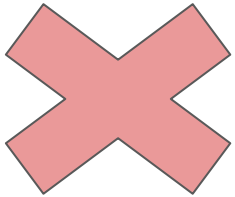


AST try-catch Transformation

Failure cases:

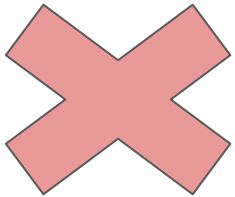
```
public void bar(int x) {
    int z = foo();
    z = z +1;
}
```

```
public void bar(int x) {
   try {
    int z = foo();
   } catch (Exception e) {

   }
   z = z +1;
}
```



```
public void bar(int x) {
    if (foo()) {
            ...
    }
}
```

```
public void bar(int x) {
try {
    if (foo()) {
            ...
    }
}
}
```

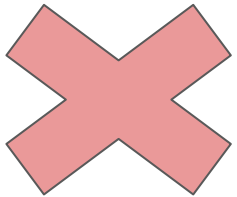


AST try-catch Transformation

Failure cases:

```
public void bar(int x) {
   for (int i=0; i<foo(); i++)
          ..
}
```

```
public void bar(int x) {
   try {
          for (int i=0; i<foo(); i++)          ..

   } catch (Exception e) {

   }
}
```



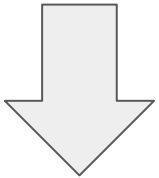
```
public void bar(int x) {
     baz(foo());
}
```

...

Bytecode try-catch Transformation

1. Add statements after invokevirtual:
   a. goto x
   b. astore_x //store exception in LVT
2. Add entry to exception table
3. modify labels on following statements

```
public void bar();
  Code:
    0: aload_0
    1: invokevirtual #2                  // Method foo:()V
    4: return
```



```
public void bar();
  Code:
    0: aload_0
    1: invokevirtual #2                  // Method foo:()V
    4: goto          8
    7: astore_2
    8: return
```

Bytecode try-catch Transformation

Failure cases of transformation on an AST:

- Output of `foo` saved in a variable
- `foo` is the condition in an if-statement
- `foo` is the stopping condition in a for-loop

Would these be problematic in bytecode transformations?

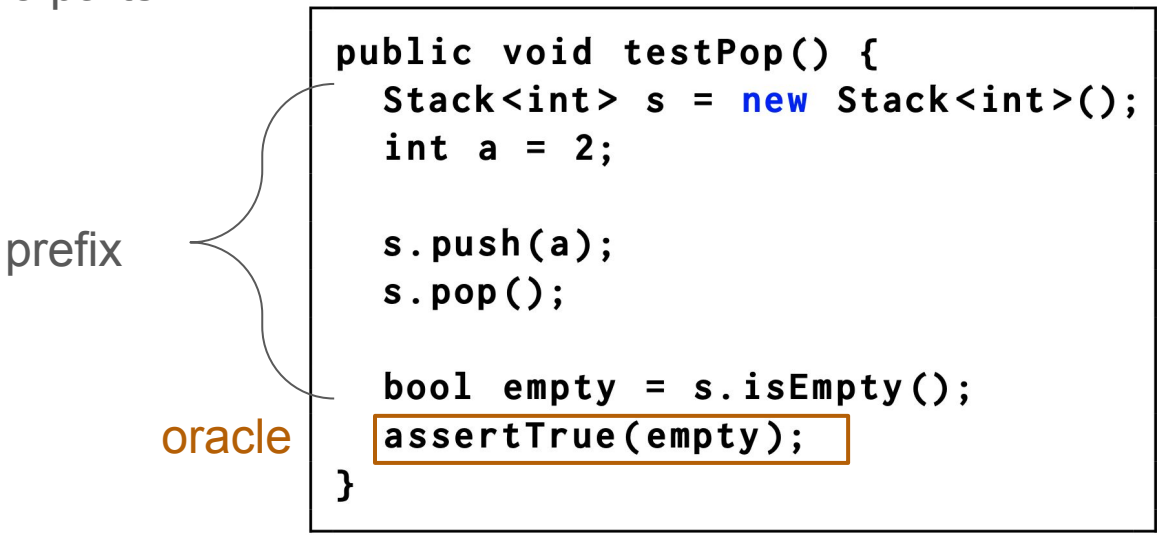
Test Oracles

Test Oracles

A unit test consists of two parts:

1. Test Prefix
2. Test Oracle

There are many different types of oracles



Safety Oracles

- Properties which should always be true *across all programs* in a particular language

- Language Specific

Java Safety Oracles

- NPE should not occur
- DivByZero should not occur
- Class Cast should only occur on objects that can be casted to that type
- An object should be equal to itself
- ...

Many of these are checked by the JVM

JavaScript Safety Oracles

- A program should never read from an absent variable
- ...

Safety Oracles

- It is very difficult to define a property which should always be true *across all programs*
- Sometimes a NPE is expected!

```
// no null inputs allowed
// throws NPE on null input
void reset(int[] values) {
    for (int i=0; i < values.length; i++) {
        values[i] = 0;
    }
}
```

Safety Oracles

- Safety oracles are a *heuristic*
  - problem-solving approach that uses practical methods or shortcuts to produce solutions that may not be perfect but are good enough for immediate goals

- Specification can violate these heuristics

Functional Oracles

- Properties which specify behavior for a *particular* program

- add(1,2) should return 3
- parseInteger(“-99”) should return -99
- A blog post() function should only execute if the user is logged in

Regression Oracle

Regression Testing aims to detect whether modifications made to a new version of the system disrupted existing functionality

- Regression oracles capture the **current behavior** of the code
- Can be safety or functional
- The oracle is *derived* from existing behavior

Generating Test Oracles

Given an input for a system, the challenge of distinguishing the corresponding desired, correct behaviour from potentially incorrect behavior is called the “**test oracle problem**”

Approaches to Generate Test Oracles

1. Specification mining techniques
   a. Creates oracles from JavaDoc comments
2. IR approach
   a. *Retrieves* a similar test and uses its assertion
3. Neural Techniques

Approaches to Generate Test Oracles

1. **Specification mining techniques**
    a. Creates oracles from JavaDoc comments
2. IR approach
    a. *Retrieves* a similar test and uses its assertion
3. Neural Techniques

Generating Oracles from the Javadoc Comment

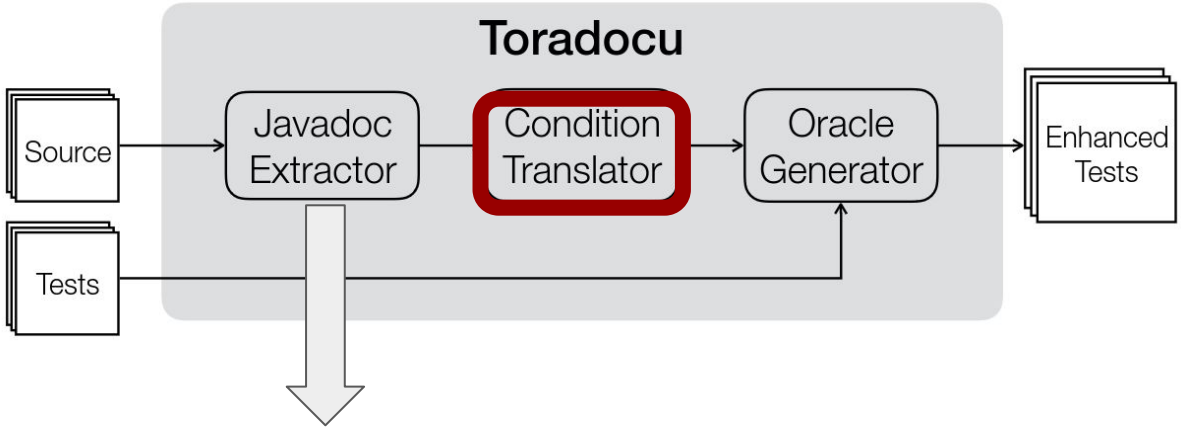
Toradocu:

- Parses documentation into grammatical relations between words in a sentence
- Matches parsed subjects and predicates to source code elements
- Converts these to assertions

```
/**
 * @throws NullPointerException if either
 *  the iterator or predicate are null
 */
public Object next() {…}
```

This work only targets exceptional behavior!

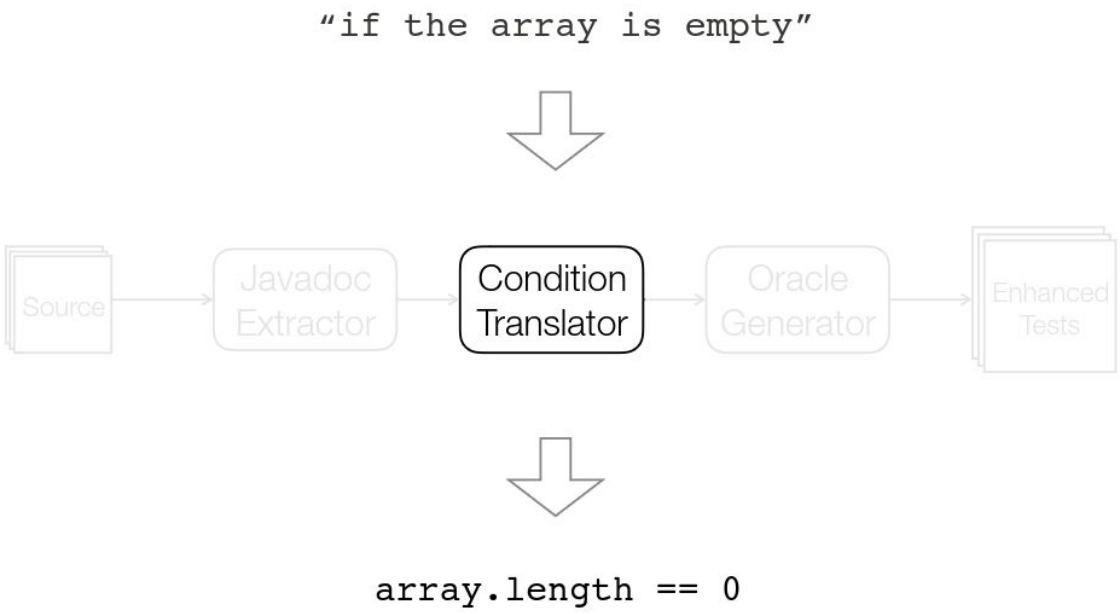
```
/**
 * @throws NullPointerException if either
 *  the iterator or predicate are null
 */
public Object next() {…}
```



"if either the iterator or predicate are null"

getIterator() == null || getPredicate() == null

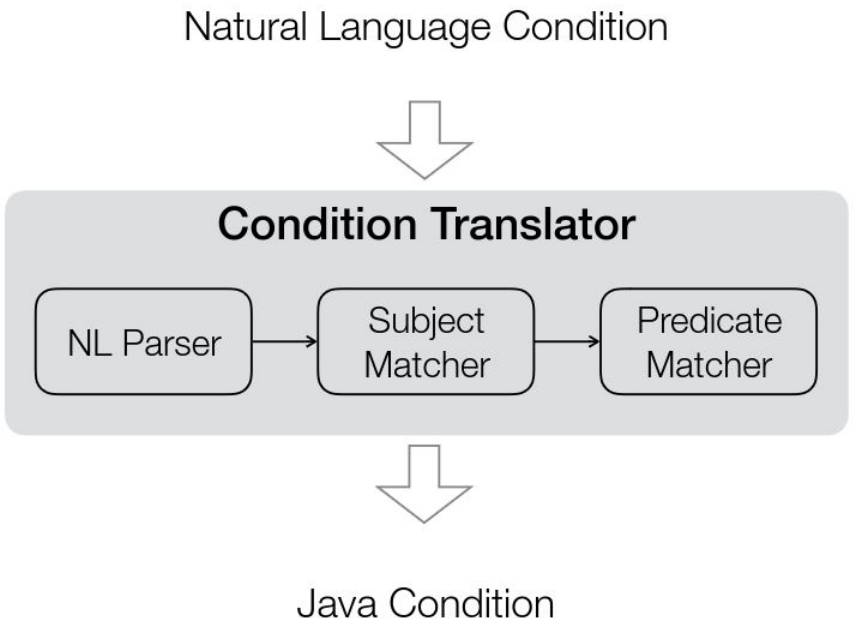
More on the Condition Translator



More on the Condition Translator

Subject: the who or what the sentence is about. Typically contains a noun

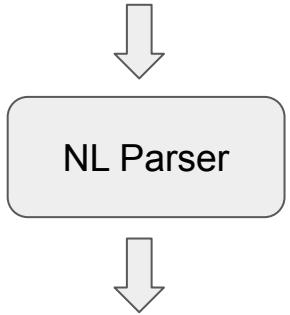
Predicate: tells us what the subject does or what happens to the subject. It includes the verb



More on the Condition Translator

```
/**
 * @throws NullPointerException if either
 *  the iterator or predicate are null
 */
public Object next() {…}
```

“If either the iterator or predicate are null”

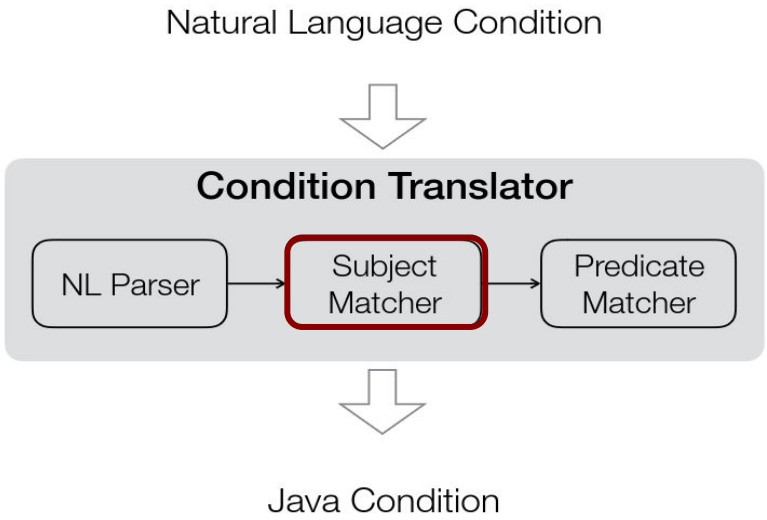


Proposition	Subject	Predicate
1	“iterator”	“are null”
2	“predicate”	“are null”

```
/**
 * @throws NullPointerException if either
 *  the iterator or predicate are null
 */
public Object next() {…}
```

Subject Matcher matches each subject with a program element based on string similarity

Proposition	Subject	Predicate
1	"iterator"	"are null"
2	"predicate"	"are null"



```
/**
 * @throws NullPointerException if either
 *  the iterator or predicate are null
 */
public Object next() {…}
```

Natural Language Condition

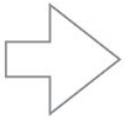
Condition Translator

NL Parser → Subject Matcher → Predicate Matcher

Java Condition

Proposition	Subject	Predicate
1	"iterator"	"are null"
2	"predicate"	"are null"

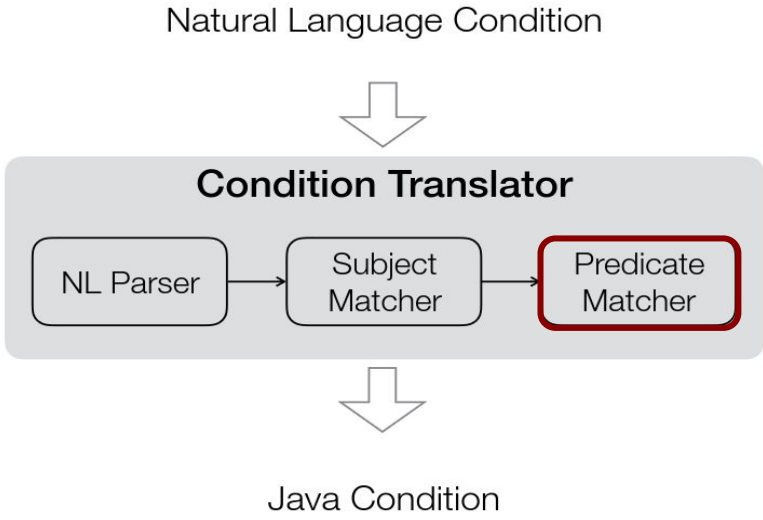
Candidates
Formal Parameters
Class Name
Non-void Nullary Methods
Fields



Candidate	Distance
`FilterIterator`	6
`getIterator`	**3**
`getPredicate`	9
`next`	6
…	

```
/**
 * @throws NullPointerException if either
 *  the iterator or predicate are null
 */
public Object next() {…}
```

Predicate Matcher matches each predicate with a program element based on string similarity



Proposition	Subject	Predicate
1	`FilterIterator.getIterator()`	"are null"
2	`FilterIterator.getPredicate()`	"are null"

NL to Spec Limitations

1. Only as good as the docstring
    a. Vaguely worded or missing docstrings are problematic

2. Only works on exceptional oracles

Approaches to Generate Test Oracles

1. Specification mining techniques
   a. Creates oracles from JavaDoc comments
2. **IR approach**
   a. *Retrieves* a similar test and uses its assertion
3. Neural Techniques

Information Retrieval

- Information retrieval (IR) is the process of obtaining relevant information from a large collection of documents.
- Used in search engines
- Fetch the object that *best matches* a given query from a database / corpus
- Best Match is defined with a *Similarity Metric*

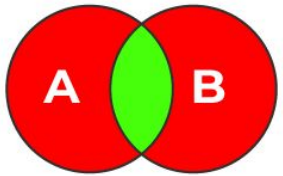
Information Retrieval for Assertion Generation

Given a Test Prefix and a corpus of Tests, find the most similar prefix and retrieve its assertion.

Similarity Metric - Jaccard

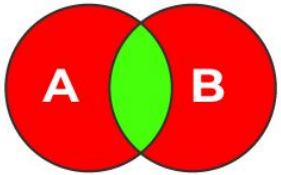
- Measures similarity between two sets
- Values range from o (no similarity) to 1 (identical sets)

$$J(X, Y) = |X \cap Y| / |X \cup Y|$$



$$\text{Jaccard} = \frac{\text{Intersection (A, B)}}{\text{Union (A, B)}}$$

Jaccard Coefficient



$$\text{Jaccard} = \frac{\text{Intersection (A, B)}}{\text{Union (A, B)}}$$

D1:

"Information Retrieval is useful"

D2:

"Retrieval of information is important"

{ information, retrieval, is useful }

{ retrieval, of, information, is, important }

$(A \cap B) = 3$

$(A \cup B) = 6$

$J(D1, D2) = 3 / 6 = 0.5$

Issues with Jaccard

1. It doesn’t consider *frequency*
2. Often times, rare words are more informative than frequent words. Jaccard doesn’t consider this

Jaccard over Test Cases

D1:

```
public void testKeyedValues() {
    KeyedValues kv;
    kv = new KeyedValues();
    Short short0 = new Short(2);
    kv.insertValue(0, short0, 2);
    kv.removeValue(0);
}
```

```
{public, void, testKeyedValues,
KeyedValues, kv, new, Short,
short0, insertValue, removeValue}
```

D2:

```
public void testMultipleKeyedValues() {
    KeyedValues kv = new KeyedValues();
    kv.insertValue(1, "First", 10);
    kv.insertValue(2, "Second", 20);

    int value2 = kv.getValue(2);

    assertEquals(20, value2);
}
```

```
{public, void, testMultipleKeyedValues,
KeyedValues, kv, insertValue, getValue,
assertEquals}
```

Jaccard over Test Cases

D1:

```
{public, void,
testKeyedValues,
KeyedValues, kv, new,
Short, short0, insertValue,
removeValue}
```

D2:

```
{public, void,
testMultipleKeyedValues,
KeyedValues, kv, insertValue,
getValue, assertEquals}
```

$(A \cap B) = 5$
$(A \cup B) = 12$

$J(D1, D2) = 5 / 12 = .4167$

IR with Jaccard

Given a test case and a corpus of tests, retrieve a test from the corpus with the highest jaccard similarity and inspect the assertion

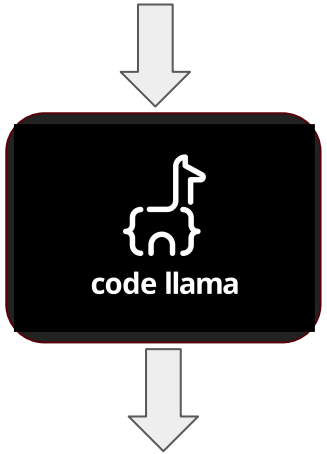
```
public void testKeyedValues() {
    KeyedValues kv;
    kv = new KeyedValues();
    Short short0 = new Short(2);
    kv.insertValue(0, short0, 2);
    kv.removeValue(0);
}
```

```
public void testRemove() {
    KeyedValues values;
    values = new KeyedValues();
    values.insertValue(0, 7, 2);
    values.removeValue(0);

   assertEquals(values.size(), 0);
}
```

IR with Jaccard - small neural edit to the retrieved assertion

```
public void testKeyedValues() {
    KeyedValues kv;
    kv = new KeyedValues();
    Short short0 = new Short(2);
    kv.insertValue(0, short0, 2);
    kv.removeValue(0);
}
```

assertEquals(values.size(), 0);



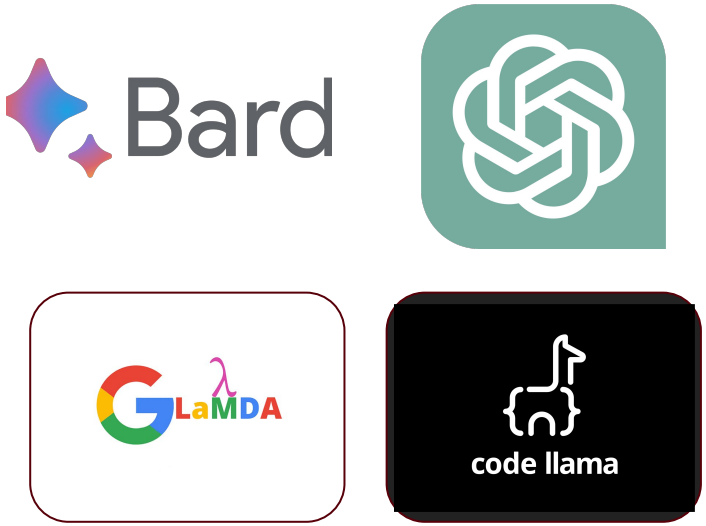
assertEquals(kv.size(), 0);

Approaches to Generate Test Oracles

1. Specification mining techniques
   a. Creates oracles from JavaDoc comments
2. IR approach
   a. *Retrieves* a similar test and uses its assertion
3. **Neural Techniques**

Neural Oracle Generation

Given a **test** and **focal method**, generate an assertion token by token



Neural Oracle Generation

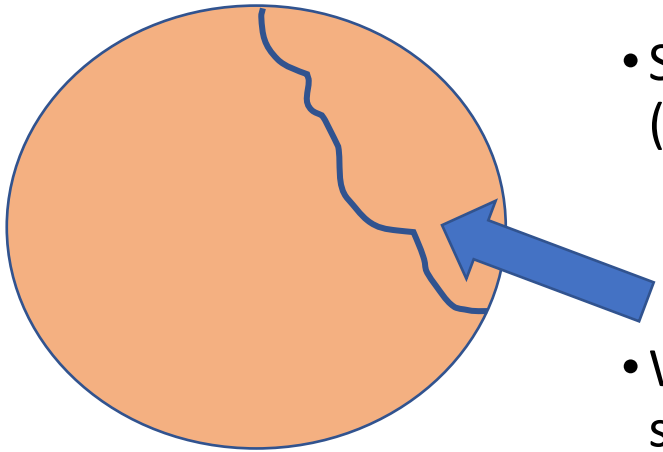
```
public void testPop() {
  Stack<int> s = new Stack<int>();
  int a = 2;

  s.push(a);
  s.pop();

  bool empty = s.isEmpty();
  <AssertPlaceHolder>
}
```

```
public void pop () {
  // NO-OP
}
```

assertTrue(s.length == 0);

NO GUARANTEES THAT THE MODEL OUTPUT COMPILES, TYPE CHECKS, OR IS CORRECT IN ANY WAY!

Leveraging Observed Oracle Structures



Set of all possible oracles

- Set of all possible oracles is massive… (maybe even infinite)
- We observed a small common set of oracle structures **in tests**

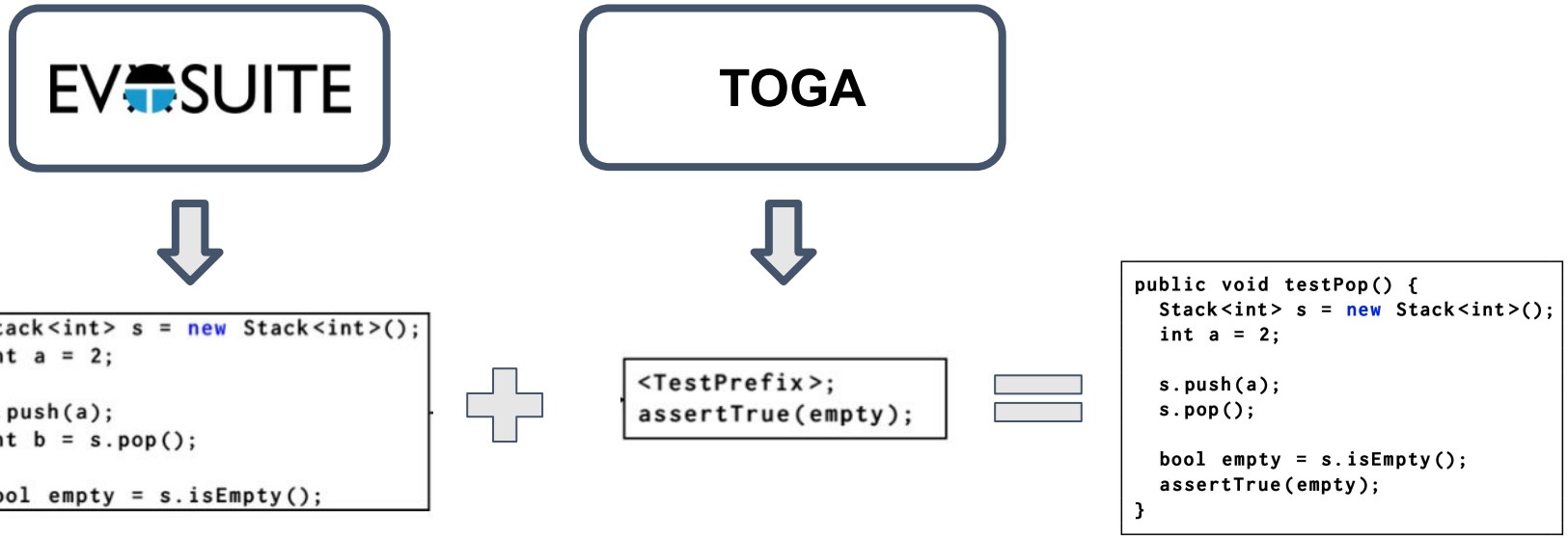
Grammatical Structure of Oracles

```
Test              T    := O(P)
Prefix            P    := statement | P; P
Oracle         O(P)    := E(P) | R(P)
Except Oracle  E(P)    := try{P; fail();} catch(Exception e){}
Return Oracle  R(P)    := P; A
Assertion         A    := assertEquals(const|var,expr) |
                          assertTrue(expr) | assertFalse(expr) |
                          assertNull(expr) | assertNotNull(expr)
```

Deliberately Restricted, but 82% tests fit grammar when evaluated on ATLAS corpus

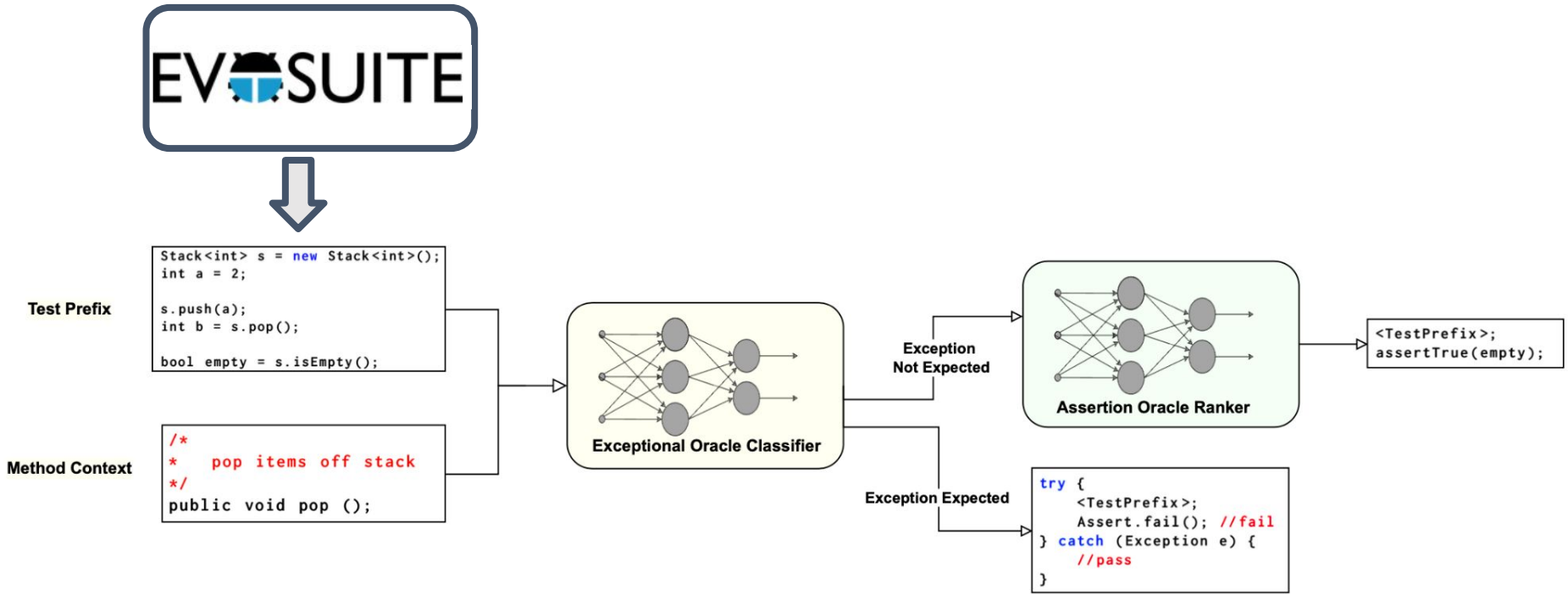
System Overview

generated by automated testing tool



System Overview

generated by automated testing tool



Assertion Inference

Method Docstring +

```
/*
*    pop items off stack
*/
public void pop ();
```

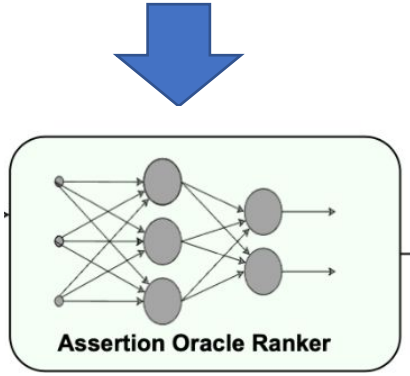
+

Assertion

```
assertEquals(2, b)
assertEquals(a, b)

assertEquals(1, b)
assertEquals(0, b)
assertEquals(100, b)
```

Test Prefix

```
public void testPop() {
  Stack<int> s = new Stack<int>();
  int a = 2;

  s.push(a);
  int b = s.pop();
  <AssertPlaceHolder
  >
}
```




```
assertEquals(2, b)
```

Summary

- Safety oracles
  - properties which should be true for *any* program
- Functional Oracles
  - Properties of a particular program
- Regression Oracle
  - Based on current behavior
- Generating Oracles
  - IR, neural, and natural language approaches

Summary

- HW1 (due next Wednesday Feb 12)
  - AST and Bytecode transformations

- Lab2 due Sunday