# Test Oracles

# Announcements

- HW1 deadline extended to Friday at Midnight
- Lab3 due Monday
- HW2 will be released Monday
  - You can work with a partner

# Outline

- Grammars

- Generating Regression Oracles

# Formal Grammars

A *grammar* describes which strings are valid according to a language's syntax

Defined as a set of *production rules*

- Containing non-terminal and terminal symbols
- Non-terminal symbols can be replaced by terminal symbols

# Formal Grammars

**S → a S b**
**S → b a**

The language of the grammar is the infinite set $\{a^n bab^n \mid n \geq 0\} = \{ba, abab, aababb, aaababbb, \ldots\}$
where $a^k$ is a repeated k times

# A Calculator Language

**S → E**
**S → T + T | T**
**T → num**
**num → 0 | 1**

**Exercise:** Exhaustively enumerate every possible expression in this grammar

# A grammar of Test Oracles

Are all possible java oracles captured by this grammar?

```
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)
```

# **Review:** Neural Approaches for Test Oracle Generation

Given the following test prefix, what is the set of valid assertions? Are they all captured by the previous grammar?

```
@Test
public void testDayOf() {
        int basePrice = 150;
        int daysUntilEvent = 0;
        int customerAge = 31;
        boolean isMember = true;

        double price = BuyTicket.calculatePrice(basePrice, daysUntilEvent, customerAge, isMember);
}
```

```
assertEquals(price, 135);
assertEquals(135, price);
assertTrue(price == 135);
assertTrue(isMember);
assertTrue(price > 0);
assertTrue(true);
....
```

# **Review:** Neural Approaches for Test Oracle Generation

An LLM's outputs are **very flexible**. It could output any of these, an assertion on a variable that doesn't exist, "potato" etc.

```
assertEquals(price, 135);
assertEquals(135, price);
assertTrue(price == 135);
assertTrue(isMember);
assertTrue(price > 0);
assertTrue(true);
....
```

**Idea**: instead of allows the LLM to generate anything, exhaustively enumerate all possible assertions from our grammar and ask a neural model to RANK THEM.

# Regression Oracles

# Generating Regression Oracles

First a bunch of mutants of the original program are generated (at they bytecode level)

Then, execute each test case against the mutant and record "*traces*" with necessary information to derive the assertion

After execution, the traces are analyzed for difference between runs. An assertion is added for each difference.

# Generating Regression Oracles

Assertions are generated based on the output type of the Method Under Test

**Primitive Assertions:** `assertEquals(var, value);`

Where `var` is the output of MUT and `value` is the observed value on the non-mutated trace

**Comparison Assertions:** `assertTrue/False(var2.equals(var));`

Where `var` is the output of MUT and `var2` is an observed value of the same type on the non-mutated trace.

**These assertions would fail if executed on a mutant**

# Generating Regression Oracles

Assertions are generated based on the output type of the Method Under Test

**Inspector Assertions:** `assertEquals(var2, value);`

Where `var2` is the output of **an observer method** and `value` is the observed value on the non-mutated trace

**Field Assertions:** `assertEquals(obj.VAR, value);`

Where `VAR` is a field on the object under test and `value` is an observed value of VAR on the non-mutated trace.

**These assertions would fail if executed on a mutant**

# Generating Regression Oracles

Assertions are generated based on the output type of the Method Under Test

**String Assertions:** `assertEquals(var.toString(), value);`

Where `var` is the output of the Method Under Test and `value` is the observed value on the non-mutated trace

Not always useful as not all classes implement `toString`

# Generating Regression Oracles

Given a unit under test `P`,

```
Execute P and record values of program variables

M = genMutants(P)

For each mutant in M:

    Execute mutant and record values of program variables

    Add assertion which kills mutant based on type of MUT
```

# Which approaches could generate oracles which find the bugs?

```
public static double calculatePrice(int basePrice, int daysUntilEvent, int customerAge, boolean isMember) {
        double maxDiscount = basePrice - 100;

        double discountRate = daysUntilEvent / maxDiscount; // BUG 1: potential division by zero

        if (isMember) discountRate += .10;

        if (customerAge >= 65 || customerAge < 18) {
            discountRate += (customerAge - 65) / maxDiscount; //BUG 2: customerAge < 18 gives negative discount
        }

        return basePrice - (basePrice * discountRate);
}
```

| | Bug 1 | Bug 2 |
|---|---|---|
| Toradocu | ☐ | ☐ |
| IR-based approach | ☐ | ☐ |
| Neural approach | ☐ | ☐ |
| Regression oracle | ☐ | ☐ |

# Summary

- Grammars describe a language by defining rules by which valid expressions are constructed
- Neural techniques for test oracle generation leverage grammars to restrict the output space
- Regression oracles reflect the current program behavior
- Regression oracles are generated by mutating P, executing, and observing the differences in program variables