CS @ ILLINOIS

PURE
Promoting Undergraduate Research in Engineering

Evaluating the Effectiveness of Pre-Condition Generation Tools

Sejal K. Parmar (sejalkp2@illinois.edu), Project Mentor: Angello Astorga, PURE Mentor: Wei Yang

Motivation

Motivation: Writing specifications(preconditions, postconditions) for programs provides a means for describing behavioral properties of the program. However, the task of writing them is error prone and time consuming. Given the importance of this task, there exists a large body of research work addressing this problem.

Goal: Evaluate the effectiveness of automated precondition inference tools.

Commutativity of Methods:

$\varphi(\vec{x},\vec{y}) \Rightarrow M1(\vec{x})M2(\vec{y}) == M2(\vec{y})M1(\vec{x})$

The order in which methods M1 and M2 occur does not matter.

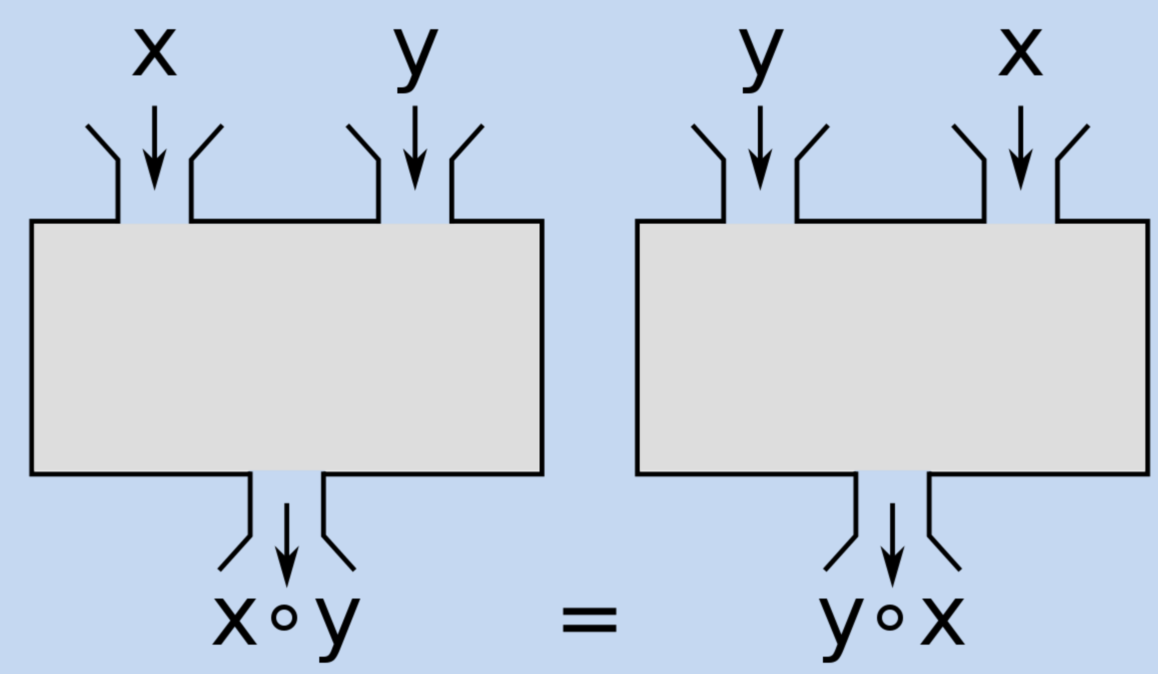
Application: *Parallel Computing:* Compilers cannot parallelize a wide range of computations unless they recognize and exploit commuting operations

Idempotency:

$\varphi(\vec{x}) \Rightarrow M1(\vec{x}) == M1(\vec{x})M1(\vec{x})$

If a call to M1 is made once or multiple times, the results will still be the same.

Application: *Fault-Tolerance:* When an online shopper pays, clicking submit multiple times should only make one transaction

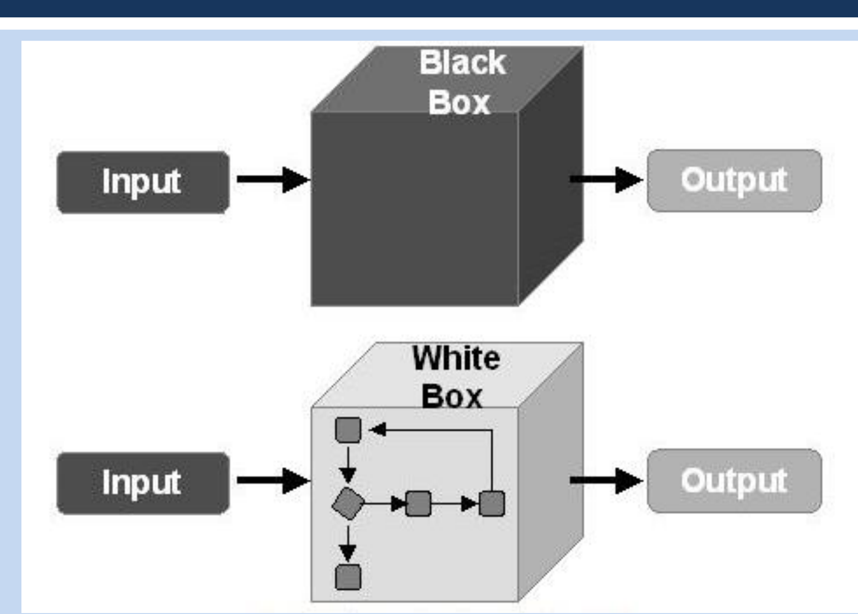


HTTP Method	Safe	Idempotent
GET	✓	✓
POST	✗	✗
PUT	✗	✓
DELETE	✗	✓
OPTIONS	✓	✓
HEAD	✓	✓

Black Box vs. White Box

Black Box:

Typically, these approaches are data-driven. They extract tests(as data points) from random sampling or test generation tools and feed these data points to a machine learning algorithm.



White Box:

This approach leverages inner structural and semantic properties. These types of approaches can also use an input test suite and generalize from the observed behavior in test runs.

By manual inspection of results of both white box and black box approaches, we can analyze the outputs to see which one produces better results.

My Work

- Wrote specifications for commutativity and idempotency of methods using Parameterized Unit Tests
- Evaluated two tools for precondition inference: one based on testing + machine learning (black-box)(submitted for publishing) and the other based on testing + dynamic analysis (white-box)
- Performed a manual inspection to assess the goodness of preconditions based on correctness and complexity

```
[PexMethod]
public void PUT_CommutativityPushPeekComm([PexAssumeUnderTest]DataStructures.Stack<int> s1, int x) {
        /* Peek throws exception when stack is empty*/
        Stack<int> s2 = (Stack<int>)s1.Clone();
        StackEqualityComparer eq = new StackEqualityComparer();
        PexAssume.IsTrue(x > -101 && x < 101);

        AssumePrecondition.IsTrue( (( s1.Contains(x) ) && ((( 0*s1.Count + -1*x + 1*s1.Peek() <= 0 ) && ((( 0*s1.Count
        + -1*x + 1*s1.Peek() <= -1 ) && (false)) ||  ((!( 0*s1.Count + -1*x + 1*s1.Peek() <= -1 )) && (true)))) ||
        ((!( 0*s1.Count + -1*x + 1*s1.Peek() <= 0 )) && (false)))) ||  ((!( s1.Contains(x) )) && (false)) );

        int p1 = -1, p2 = -2;

        s1.Push(x); // First test
        p1 = s1.Peek();

        p2 = s2.Peek(); // Second test
        s2.Push(x);

        PexAssert.IsTrue(p1 == p2 && eq.Equals(s1, s2));
    }
```

Black Box Precondition:

$(\mathrm{contains}(x)\ \mathrm{and}\ ((-x + \mathrm{peek}()) <= 0)\ \&\&\ (\mathrm{not}\ ((-x + \mathrm{Peek}()) <= -1)))$

Which simplifies to:

Contains(x) && Peek() == x

White Box Precondition:

$1 == (\mathrm{Stack}<\mathrm{int}>)\mathrm{null}\ ||\ \mathrm{methodof}(\mathrm{s1.Clone})\ != \mathrm{methodof}(\mathrm{Stack}<\mathrm{int}>.\mathrm{Clone})\ ||\ -1 >= \mathrm{s1.\_size}\ ||\ -1 >= \mathrm{s0}\ ||\ \mathrm{s4} < \mathrm{s0}\ ||\ \mathrm{s1.\_size} < \mathrm{s0}\ \ldots$

Which simplifies to:

For all Elements in stack must equal zero