# Lecture 8 – Testing Oracles

## AAA705: Software Testing and Quality Assurance

Jihyeok Park


2026 Fall

# Recall

PLRG

- Mutation Testing – inject small faults, count how many the suite **kills**
  - Competent programmer and coupling effect hypotheses
  - Operators; a kill needs reachability + infection + propagation
  - Equivalent mutants are **undecidable**; TCE catches the trivial ones
  - Scale by doing fewer or smarter – at Google, mutate only the **diff**
  - A better proxy than coverage: **73%** of real faults couple to a mutant
- Test Flakiness – the same code, a different verdict
  - Mostly concurrency and randomness; a re-run culture is toxic
  - Detection without reruns (DeFlaker), prediction from features and from words

# What Does "the Test Failed" Mean?

Every technique so far ends the same way: **run the test and compare the result**.

```
def abs(x):
  if x < 0:
    return -x
  return x

assert abs(-1) == 1    # who said 1?
```

- Coverage, mutation score, flakiness – all of them assume someone can say **pass** or **fail**. That someone is the **test oracle**. Here it is the `1`, and we wrote it down without thinking.

[1] **[TSE'15]** E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo. "*The Oracle Problem in Software Testing: A Survey.*"

# What Does "the Test Failed" Mean?

Every technique so far ends the same way: **run the test and compare the result**.

```
def abs(x):
  if x < 0:
    return -x
  return x

assert abs(-1) == 1    # who said 1?
```

- Coverage, mutation score, flakiness – all of them assume someone can say **pass** or **fail**. That someone is the **test oracle**. Here it is the 1, and we wrote it down without thinking.
- For a program that computes $\pi$, a compiler, or an image classifier, **nobody can write down the expected output**.

[1] **[TSE'15]** E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo. "*The Oracle Problem in Software Testing: A Survey.*"

# What Does "the Test Failed" Mean?

Every technique so far ends the same way: **run the test and compare the result**.

```
def abs(x):
  if x < 0:
    return -x
  return x

assert abs(-1) == 1    # who said 1?
```

- Coverage, mutation score, flakiness – all of them assume someone can say **pass** or **fail**. That someone is the **test oracle**. Here it is the 1, and we wrote it down without thinking.
- For a program that computes $\pi$, a compiler, or an image classifier, **nobody can write down the expected output**.
- **Today**: how to test a program when you cannot write down the answer.[1]

[1] **[TSE'15]** E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo. "*The Oracle Problem in Software Testing: A Survey.*"

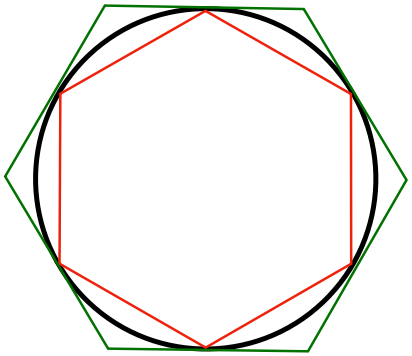
# Contents

# Contents

# History of $\pi$

How do you compute $\pi$? For 4000 years the answer was "squeeze it".

- BC 2000, Babylonia – approximate it: $3 + 1/8 = 3.125$
- BC 250, Archimedes – inscribe and circumscribe polygons

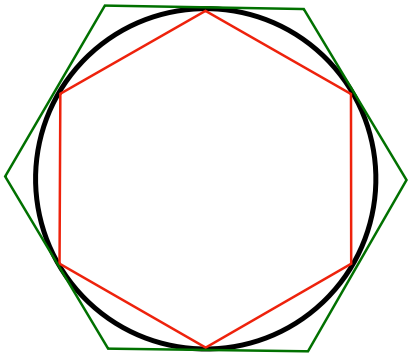


$$6 \cdot \frac{1}{2} = 3 < \pi < 3.47 \approx 6\left(\frac{1}{\sqrt{3}}\right)$$

# History of $\pi$

How do you compute $\pi$? For 4000 years the answer was "squeeze it".

- BC 2000, Babylonia – approximate it: $3 + 1/8 = 3.125$
- BC 250, Archimedes – inscribe and circumscribe polygons



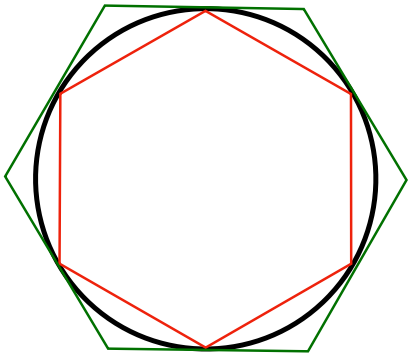
$$6 \cdot \frac{1}{2} = 3 < \pi < 3.47 \approx 6\left(\frac{1}{\sqrt{3}}\right)$$

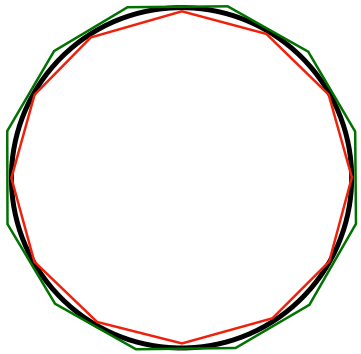
$$\sin(\frac{\alpha}{2}) = \sqrt{\frac{1-\cos\alpha}{2}} \qquad \cos(\frac{\alpha}{2}) = \sqrt{\frac{1+\cos\alpha}{2}} \qquad \tan(\frac{\alpha}{2}) = \sqrt{\frac{1-\cos\alpha}{1+\cos\alpha}}$$

# History of $\pi$

How do you compute $\pi$? For 4000 years the answer was "squeeze it".

- BC 2000, Babylonia – approximate it: $3 + 1/8 = 3.125$
- BC 250, Archimedes – inscribe and circumscribe polygons



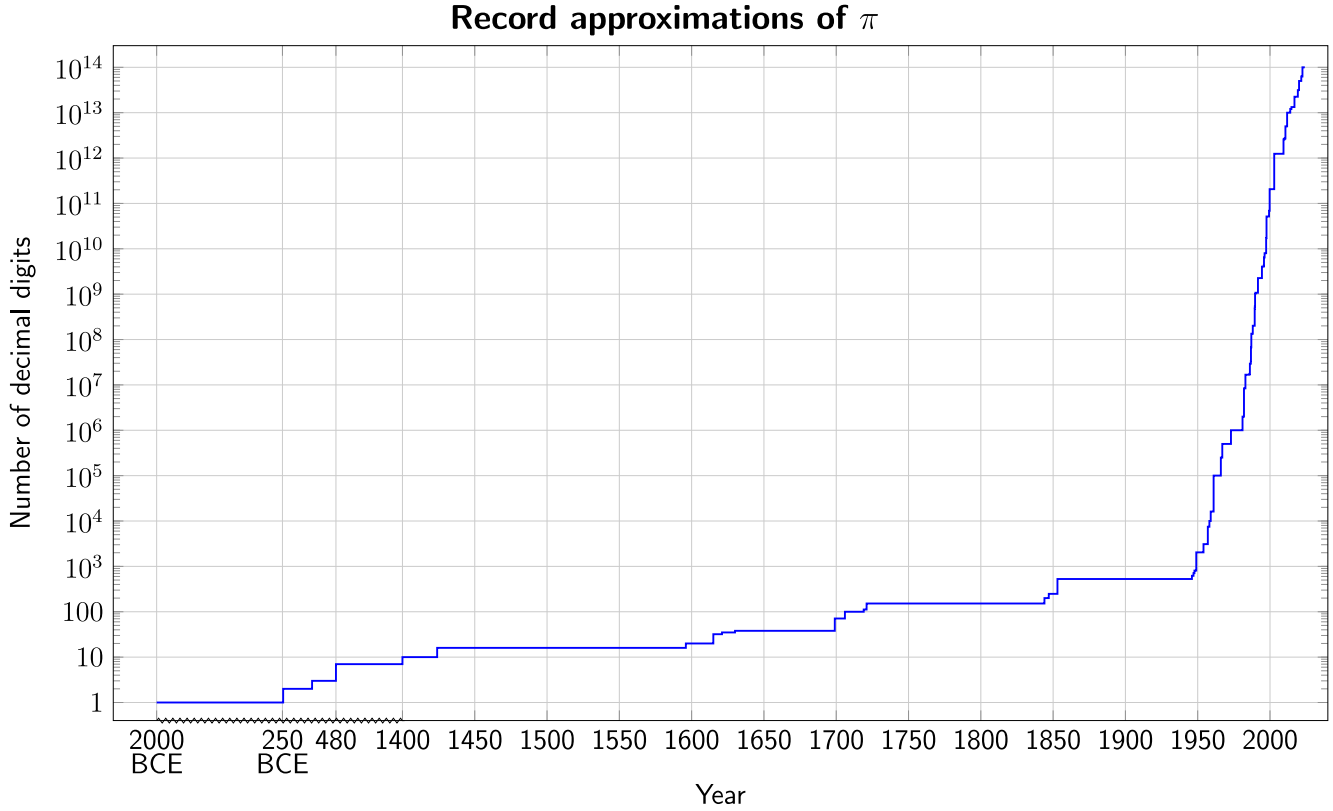


$$6 \cdot \frac{1}{2} = 3 < \pi < 3.47 \approx 6\left(\frac{1}{\sqrt{3}}\right) \qquad 12\left(\frac{\sqrt{2-\sqrt{3}}}{2}\right) \approx 3.10 < \pi < 3.22 \approx 12\left(\frac{\sqrt{2-\sqrt{3}}}{\sqrt{2+\sqrt{3}}}\right)$$

$$\sin(\frac{\alpha}{2}) = \sqrt{\frac{1-\cos\alpha}{2}} \qquad \cos(\frac{\alpha}{2}) = \sqrt{\frac{1+\cos\alpha}{2}} \qquad \tan(\frac{\alpha}{2}) = \sqrt{\frac{1-\cos\alpha}{1+\cos\alpha}}$$

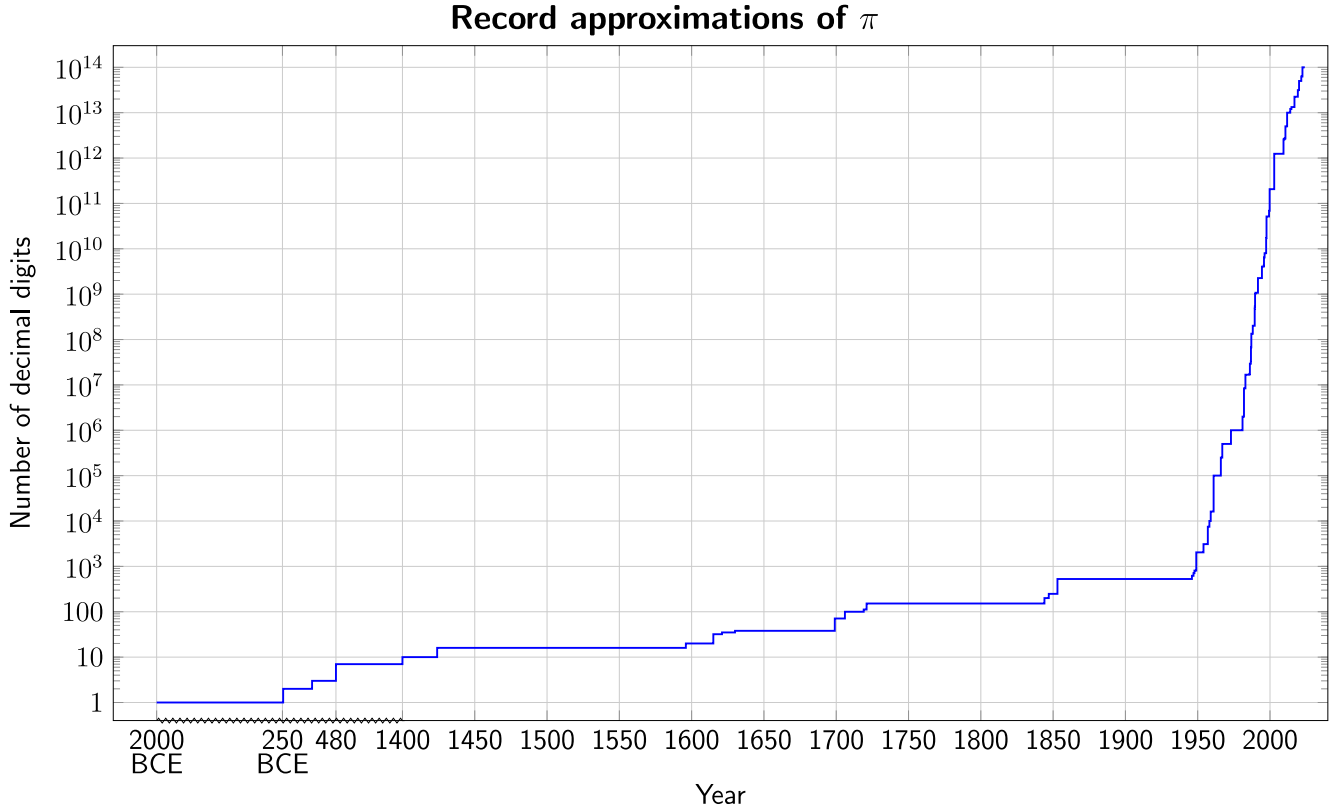
# Four Thousand Years of $\pi$

- Archimedes' 96-gon gives **2 digits**. Van Ceulen spent **25 years** on a $2^{62}$-gon for **35**. Infinite series (Madhava, Ramanujan) and then computers changed the **slope** – never the problem.



# Four Thousand Years of $\pi$

- Archimedes' 96-gon gives **2 digits**. Van Ceulen spent **25 years** on a $2^{62}$-gon for **35**. Infinite series (Madhava, Ramanujan) and then computers changed the **slope** – never the problem.




Each of them had to decide, with **no answer to compare against**, whether the digits were right.

# Computing $\pi$

- How to write a **program** that computes $\pi$?

[2] **[Comput. J.'82]** E. J. Weyuker. "*On Testing Non-Testable Programs.*"

# Computing $\pi$

- How to write a **program** that computes $\pi$?
- Then, how to **test** the program that actually computes $\pi$?

[2] **[Comput. J.'82]** E. J. Weyuker. "*On Testing Non-Testable Programs.*"

# Computing $\pi$

- How to write a **program** that computes $\pi$?
- Then, how to **test** the program that actually computes $\pi$?
- Some programs do not have a **test oracle** to compare the output.

[2] **[Comput. J.'82]** E. J. Weyuker. "*On Testing Non-Testable Programs.*"

# Computing $\pi$

PLRG

- How to write a **program** that computes $\pi$?
- Then, how to **test** the program that actually computes $\pi$?
- Some programs do not have a **test oracle** to compare the output.
- Weyuker calls such programs **non-testable**.[2]

[2] **[Comput. J.'82]** E. J. Weyuker. "*On Testing Non-Testable Programs.*"

# Computing $\pi$

- How to write a **program** that computes $\pi$?
- Then, how to **test** the program that actually computes $\pi$?
- Some programs do not have a **test oracle** to compare the output.
- Weyuker calls such programs **non-testable**.[2]
- Then, how to **test** such non-testable programs?

[2] **[Comput. J.'82]** E. J. Weyuker. "*On Testing Non-Testable Programs.*"

# Types of Non-Testable Programs

- **Type 1**: the program was written to find an answer we do not already know. If we knew the oracle, we would not need the program.

# Types of Non-Testable Programs

- **Type 1**: the program was written to find an answer we do not already know. If we knew the oracle, we would not need the program.

- **Type 2**: the program produces so much output that checking all of it is impractical.

# Types of Non-Testable Programs

- **Type 1**: the program was written to find an answer we do not already know. If we knew the oracle, we would not need the program.
- **Type 2**: the program produces so much output that checking all of it is impractical.
- **Type 3**: the tester **misunderstands** what the right answer is – the oracle exists, but it is wrong.

# Types of Non-Testable Programs

From our point of view, type 1 programs are the most interesting.

- **Type 1**: Most scientific computation, certain branches of Artificial Intelligence or Machine Learning, etc.

# Types of Non-Testable Programs

From our point of view, type 1 programs are the most interesting.

- **Type 1**: Most scientific computation, certain branches of Artificial Intelligence or Machine Learning, etc.
  - What is the **correct** value of $\pi$?

# Types of Non-Testable Programs

From our point of view, type 1 programs are the most interesting.

- **Type 1**: Most scientific computation, certain branches of Artificial Intelligence or Machine Learning, etc.
  - What is the **correct** value of $\pi$?
  - What is the **correct** result of the sin, cos, and tan functions?

# Types of Non-Testable Programs

From our point of view, type 1 programs are the most interesting.

- **Type 1**: Most scientific computation, certain branches of Artificial Intelligence or Machine Learning, etc.
  - What is the **correct** value of $\pi$?
  - What is the **correct** result of the sin, cos, and tan functions?
  - What is the **correct** way to play a video game, if you are applying reinforcement learning?

# Types of Non-Testable Programs

From our point of view, type 1 programs are the most interesting.

- **Type 1**: Most scientific computation, certain branches of Artificial Intelligence or Machine Learning, etc.
  - What is the **correct** value of $\pi$?
  - What is the **correct** result of the sin, cos, and tan functions?
  - What is the **correct** way to play a video game, if you are applying reinforcement learning?
  - What is the **correct** way to classify an image, if you are applying deep learning?

# Types of Non-Testable Programs

From our point of view, type 1 programs are the most interesting.

- **Type 1**: Most scientific computation, certain branches of Artificial Intelligence or Machine Learning, etc.
  - What is the **correct** value of $\pi$?
  - What is the **correct** result of the sin, cos, and tan functions?
  - What is the **correct** way to play a video game, if you are applying reinforcement learning?
  - What is the **correct** way to classify an image, if you are applying deep learning?
  - What is the **correct** way to translate a sentence, if you are applying machine translation?

# Four Kinds of Oracle

Having no expected output does not mean having **nothing** to compare with.[3]

- **Specified** – a formal specification decides. Rare, and often as hard to write as the program itself.

[3] **[TSE'15]** E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo. "*The Oracle Problem in Software Testing: A Survey.*"

# Four Kinds of Oracle

Having no expected output does not mean having **nothing** to compare with.[3]

- **Specified** – a formal specification decides. Rare, and often as hard to write as the program itself.
- **Derived** – another run, another version, another implementation, or a documented property decides.

[3] **[TSE'15]** E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo. "*The Oracle Problem in Software Testing: A Survey.*"

# Four Kinds of Oracle

Having no expected output does not mean having **nothing** to compare with.[3]

- **Specified** – a formal specification decides. Rare, and often as hard to write as the program itself.
- **Derived** – another run, another version, another implementation, or a documented property decides.
- **Implicit** – crashes, assertion failures, memory errors. Nothing says what the answer is, but that is clearly **not** it. This is the oracle **fuzzing** used.

[3] **[TSE'15]** E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo. "*The Oracle Problem in Software Testing: A Survey.*"

# Four Kinds of Oracle

Having no expected output does not mean having **nothing** to compare with.[3]

- **Specified** – a formal specification decides. Rare, and often as hard to write as the program itself.
- **Derived** – another run, another version, another implementation, or a documented property decides.
- **Implicit** – crashes, assertion failures, memory errors. Nothing says what the answer is, but that is clearly **not** it. This is the oracle **fuzzing** used.
- **Human** – the last resort. Expensive, and Weyuker's type 3 is precisely a human oracle that is **wrong**.

[3] **[TSE'15]** E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo. "*The Oracle Problem in Software Testing: A Survey.*"

# Four Kinds of Oracle

Having no expected output does not mean having **nothing** to compare with.[3]

- **Specified** – a formal specification decides. Rare, and often as hard to write as the program itself.
- **Derived** – another run, another version, another implementation, or a documented property decides.
- **Implicit** – crashes, assertion failures, memory errors. Nothing says what the answer is, but that is clearly **not** it. This is the oracle **fuzzing** used.
- **Human** – the last resort. Expensive, and Weyuker's type 3 is precisely a human oracle that is **wrong**.

All three techniques today build **derived** oracles.

[3] **[TSE'15]** E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo. "*The Oracle Problem in Software Testing: A Survey.*"

# Implementing sin

- Let's assume that you implemented the sin function.

# Implementing sin

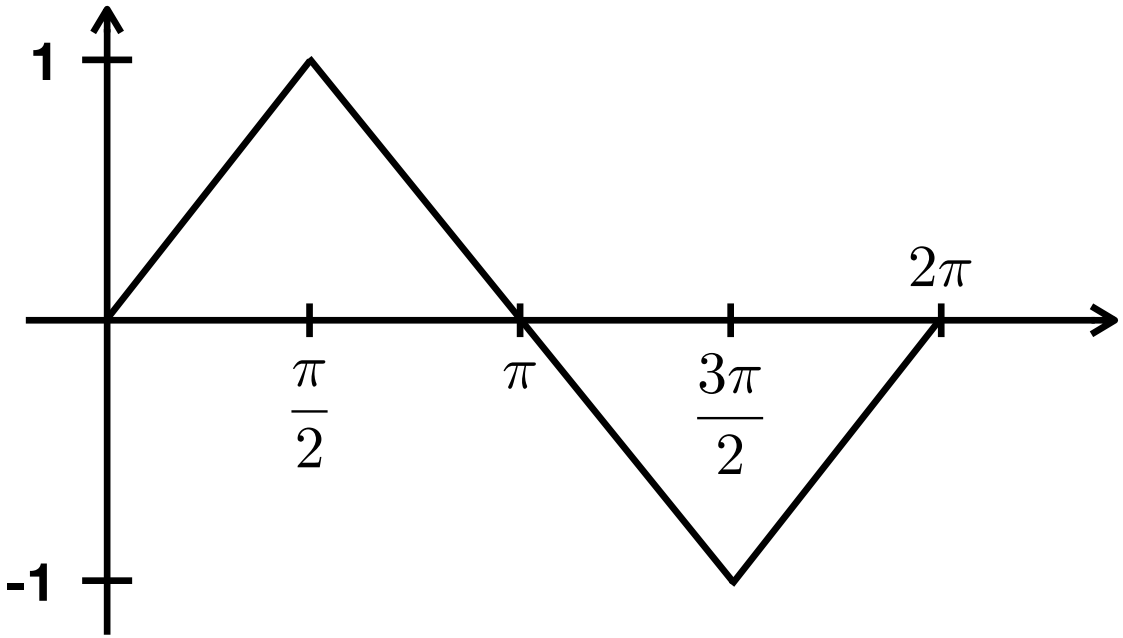
- Let's assume that you implemented the sin function.
- How to test the sin function?

# Implementing sin

- Let's assume that you implemented the sin function.
- How to test the sin function?
- We can test it using the known values of the sin function.
  - $\sin(0) = 0$
  - $\sin(\pi/2) = 1$
  - $\sin(\pi) = 0$
  - $\sin(3\pi/2) = -1$
  - $\sin(2\pi) = 0$

# Implementing sin

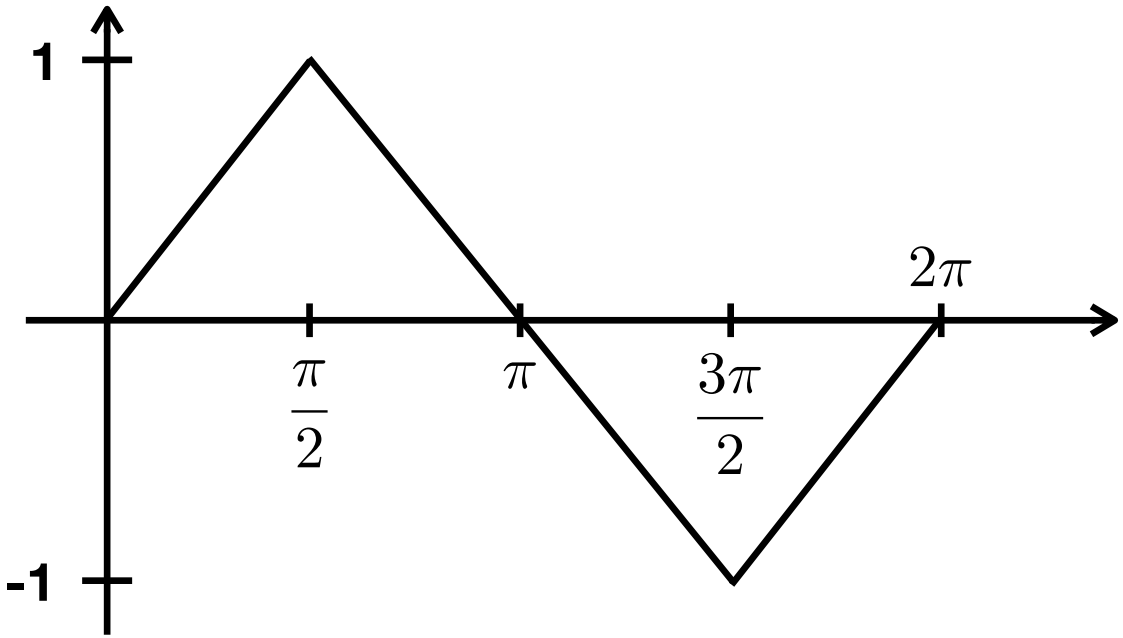
However, a wrong implementation of the sin function can pass all of them:



# Implementing sin

PLRG

However, a wrong implementation of the sin function can pass all of them:

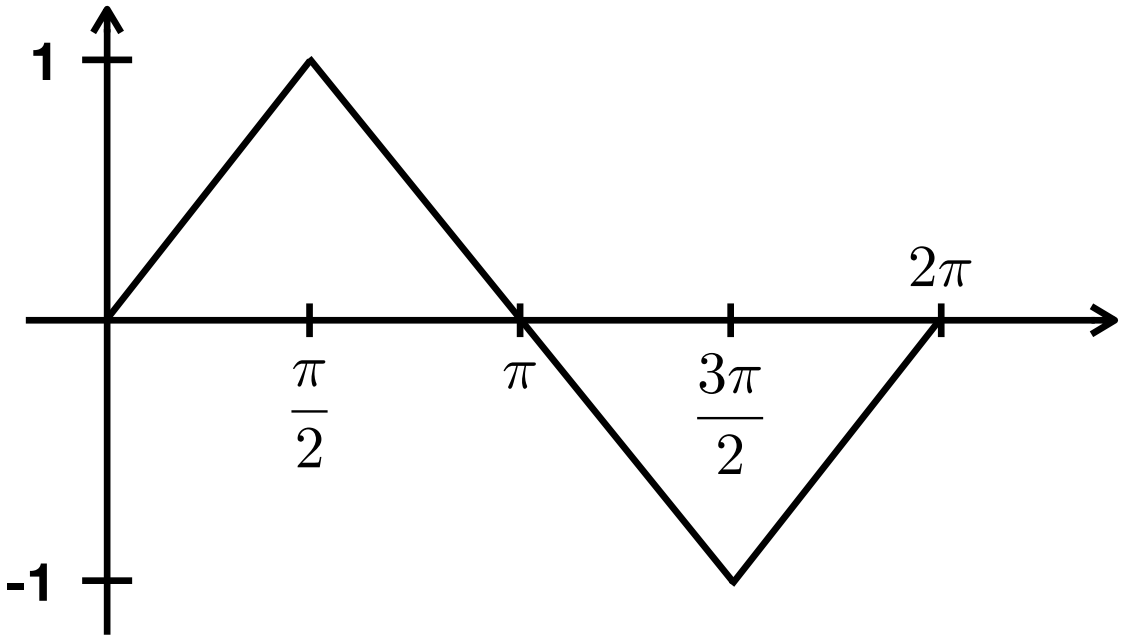



Let's utilize a domain knowledge to test the sin function:

# Implementing sin

PLRG

However, a wrong implementation of the sin function can pass all of them:



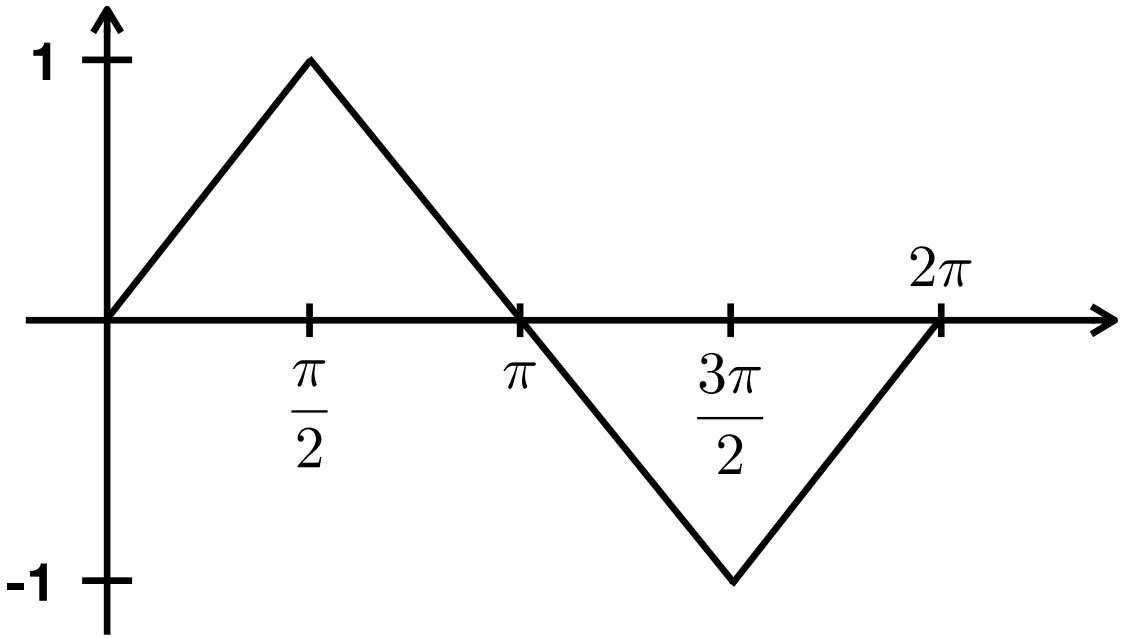

Let's utilize a domain knowledge to test the sin function:

- $\sin(x) = \sin(\pi - x)$

# Implementing sin

PLRG

However, a wrong implementation of the sin function can pass all of them:



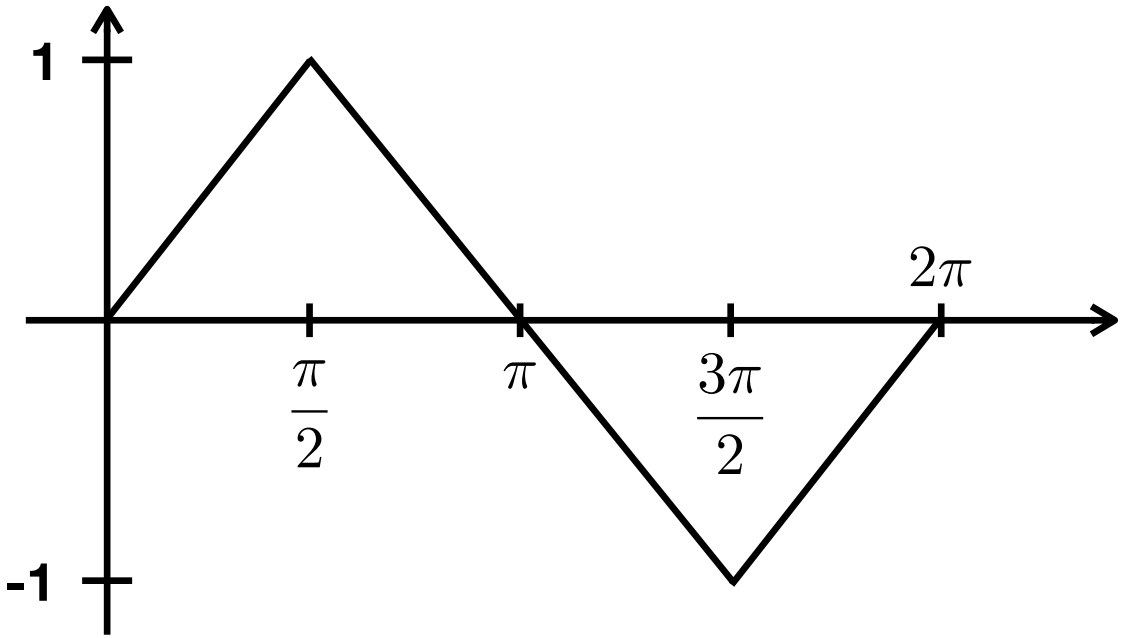

Let's utilize a domain knowledge to test the sin function:

- $\sin(x) = \sin(\pi - x)$
- $\sin(x) = -\sin(x + \pi)$

# Implementing sin

PLRG

However, a wrong implementation of the sin function can pass all of them:




Let's utilize a domain knowledge to test the sin function:

- $\sin(x) = \sin(\pi - x)$
- $\sin(x) = -\sin(x + \pi)$
- $\sin^2(x) + \sin^2(x + \frac{\pi}{2}) = 1$

# Contents

# Metamorphic Relationship

PLRG

## Definition (Metamorphic Relationship[4])

A program $p : X \to Y$ has a **metamorphic relationship** $f : X \to X$ with a relation $R \subseteq Y \times Y$ if and only if:

$$\forall x \in X.\ (p(x), p \circ f(x)) \in R$$

[4] **[HKUST TR'98]** T. Y. Chen, S. C. Cheung, S. M. Yiu. "*Metamorphic Testing: A New Approach for Generating Next Test Cases.*"

# Metamorphic Relationship

PLRG

## Definition (Metamorphic Relationship[4])

A program $p : X \to Y$ has a **metamorphic relationship** $f : X \to X$ with a relation $R \subseteq Y \times Y$ if and only if:

$$\forall x \in X.\ (p(x), p \circ f(x)) \in R$$

For example, the sin function has the following metamorphic relationships:

| **f** | **Relationship R** |
|---|---|
| $f(x) = \pi - x$ | $\sin(x) = \sin(\pi - x)$ |
| $f(x) = x + \pi$ | $\sin(x) = -\sin(x + \pi)$ |
| $f(x) = x + \frac{\pi}{2}$ | $\sin^2(x) + \sin^2(x + \frac{\pi}{2}) = 1$ |

[4] **[HKUST TR'98]** T. Y. Chen, S. C. Cheung, S. M. Yiu. "*Metamorphic Testing: A New Approach for Generating Next Test Cases.*"

# Metamorphic Testing

- **Metamorphic Testing** is a testing technique that is based on the metamorphic relationships of the program. Twenty-five years of it are collected in one survey.[5]

[5] **[CSUR'18]** T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, Z. Q. Zhou. "*Metamorphic Testing: A Review of Challenges and Opportunities.*"

# Metamorphic Testing

PLRG

- **Metamorphic Testing** is a testing technique that is based on the metamorphic relationships of the program. Twenty-five years of it are collected in one survey.[5]

- Two things to design: the **transformation** $f$ on the input and the **relation** $R$ on the outputs. Neither needs the expected output.

[5] **[CSUR'18]** T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, Z. Q. Zhou. "*Metamorphic Testing: A Review of Challenges and Opportunities.*"

# Metamorphic Testing

PLRG

- **Metamorphic Testing** is a testing technique that is based on the metamorphic relationships of the program. Twenty-five years of it are collected in one survey.[5]

- Two things to design: the **transformation** $f$ on the input and the **relation** $R$ on the outputs. Neither needs the expected output.

- Next: what $f$ and $R$ look like for SMT solvers, Datalog and database engines, a debugger, compilers, and neural networks – and then where the relations come from.

[5] **[CSUR'18]** T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, Z. Q. Zhou. "*Metamorphic Testing: A Review of Challenges and Opportunities.*"

# Examples: SMT Solvers

**Semantic fusion**: glue two formulas whose satisfiability you already know, and you know the answer for the result.[6]

$$\varphi_1 = x > 0 \wedge \boxed{x} > 1$$
$$\varphi_2 = \boxed{y} < 0 \wedge y < 1$$
$$\varphi_{concat} = (x > 0 \wedge \boxed{x} > 1) \wedge (\boxed{y} < 0 \wedge y < 1)$$
$$\varphi_{fused} = (x > 0 \wedge \boxed{z - y} > 1) \wedge (\boxed{z - x} < 0 \wedge y < 1)$$

1. **Formula Concatenation** – Concatenate two formulas $\phi_1$ and $\phi_2$ using **conjunction** ($\wedge$) or **disjunction** ($\vee$).
2. **Variable Fusion** – Create fresh variables to connect the variable sets of $\phi_1$ and $\phi_2$ using **fusion functions**.
3. **Variable Inversion** – Substitute some occurrences of the chosen variables in $\phi_1$ and $\phi_2$ using **inversion functions**.

[6] **[PLDI'20]** D. Winterer, C. Zhang, Z. Su. "*Validating SMT Solvers via Semantic Fusion.*"

# Examples: SMT Solvers

The following **fusion functions** and **inversion functions** are used to generate the **metamorphic transformations** to test the **SMT solvers**.[7]

| Type | Fusion Function | Variable Inversion Functions | |
|---|---|---|---|
| | | $r_x$ | $r_y$ |
| Int | $x + y$ | $z - y$ | $z - x$ |
| | $x + c + y$ | $z - c - y$ | $z - c - x$ |
| | $x * y$ | $z \; div \; y$ | $z \; div \; x$ |
| | $c_1 * x + c_2 * y + c_3$ | $(z - c_2 * y - c_3) \; div \; c_1$ | $(z - c_1 * x - c_3) \; div \; c_2$ |
| Real | $x + y$ | $z - y$ | $z - x$ |
| | $x + c + y$ | $z - c - y$ | $z - c - x$ |
| | $x * y$ | $z/y$ | $z/x$ |
| | $c_1 * x + c_2 * y + c_3$ | $(z - c_2 * y - c_3)/c_1$ | $(z - c_1 * x - c_3)/c_2$ |
| String | $x \; str{+}{+} \; y$ | $str.substr \;\; z \; 0 \; (str.len \; x)$ | $str.substr \;\; z \; (str.len \; x) \; (str.len \; y)$ |
| | $x \; str{+}{+} \; y$ | $str.substr \;\; z \; 0 \; (str.len \; x)$ | $str.replace \; z \; x \; ""$ |
| | $x \; str{+}{+} \; c \; str{+}{+} \; y$ | $str.substr \;\; z \; 0 \; (str.len \; x)$ | $str.replace \; (str.replace \; z \; x \; "") \; c \; ""$ |

[7] **[PLDI'20]** D. Winterer, C. Zhang, Z. Su. "*Validating SMT Solvers via Semantic Fusion.*"

# Examples: Datalog Engines

A Datalog query is a **rule**. Add a subgoal and the result can only **shrink** – unless the subgoal was already true.[8]

$Q$ `p(X) :- a(X,Y), a(Y,X).`

- $p(X)$ holds when some $Y$ satisfies **every** subgoal. The result is a **set of facts**.

[8] **[ESEC/FSE'21]** M. N. Mansur, M. Christakis, V. Wüstholz. "*Metamorphic Testing of Datalog Engines.*"

# Examples: Datalog Engines

A Datalog query is a **rule**. Add a subgoal and the result can only **shrink** – unless the subgoal was already true.[8]

$Q$ `p(X) :- a(X,Y), a(Y,X).`

$Q_1$ `p(X) :- a(X,Y), a(Y,X), b(X).` $Q_1 \subseteq Q$

- $p(X)$ holds when some $Y$ satisfies **every** subgoal. The result is a **set of facts**.
- `b(X)` is one more **condition** on the same $X$: some facts drop out, none appear.

[8] **[ESEC/FSE'21]** M. N. Mansur, M. Christakis, V. Wüstholz. "*Metamorphic Testing of Datalog Engines.*"

# Examples: Datalog Engines

A Datalog query is a **rule**. Add a subgoal and the result can only **shrink** – unless the subgoal was already true.[8]

| | | |
|---|---|---|
| $Q$ | `p(X) :- a(X,Y), a(Y,X).` | |
| $Q_1$ | `p(X) :- a(X,Y), a(Y,X), b(X).` | $Q_1 \subseteq Q$ |
| $Q_2$ | `p(X) :- a(X,Y), a(Y,X), a(Z,X).` | $Q_2 \equiv Q$ |

- $p(X)$ holds when some $Y$ satisfies **every** subgoal. The result is a **set of facts**.
- `b(X)` is one more **condition** on the same $X$: some facts drop out, none appear.
- `a(Z,X)` with a **fresh** $Z$ is already true with $Z = Y$: nothing drops out.

[8] **[ESEC/FSE'21]** M. N. Mansur, M. Christakis, V. Wüstholz. "*Metamorphic Testing of Datalog Engines.*"

# Examples: Datalog Engines

A Datalog query is a **rule**. Add a subgoal and the result can only **shrink** – unless the subgoal was already true.[8]

| | | |
|---|---|---|
| $Q$ | `p(X) :- a(X,Y), a(Y,X).` | |
| $Q_1$ | `p(X) :- a(X,Y), a(Y,X), b(X).` | $Q_1 \subseteq Q$ |
| $Q_2$ | `p(X) :- a(X,Y), a(Y,X), a(Z,X).` | $Q_2 \equiv Q$ |

- $p(X)$ holds when some $Y$ satisfies **every** subgoal. The result is a **set of facts**.
- `b(X)` is one more **condition** on the same $X$: some facts drop out, none appear.
- `a(Z,X)` with a **fresh** $Z$ is already true with $Z = Y$: nothing drops out.
- An engine that returns any other set for $Q_1$ or $Q_2$ has a bug.

[8] **[ESEC/FSE'21]** M. N. Mansur, M. Christakis, V. Wüstholz. "*Metamorphic Testing of Datalog Engines.*"

# Examples: Database Engines

A query silently returns the **wrong rows**. No crash, and no expected result set. **NoREC**: ask the **same engine** twice – optimizing, and not.[9]

t

| c |
|---|
| 3 |
| 0 |
| -5 |

```
SELECT * FROM t
WHERE c > 0;
```

**Result**: 1 row

| c |
|---|
| 3 |

```
SELECT (c > 0)
FROM t;
```

**Result**: 1 TRUE

| c > 0 |
|---|
| TRUE |
| FALSE |
| FALSE |

[9] **[ESEC/FSE'20]** M. Rigger, Z. Su. "*Detecting Optimization Bugs in Database Engines via Non-Optimizing Reference Engine Construction.*"

# Examples: Database Engines

A query silently returns the **wrong rows**. No crash, and no expected result set. **NoREC**: ask the **same engine** twice – optimizing, and not.[9]

t

| c |
|---|
| 3 |
| 0 |
| -5 |

```
SELECT * FROM t
WHERE c > 0;
```

**Result**: 1 row

| c |
|---|
| 3 |

```
SELECT (c > 0)
FROM t;
```

**Result**: 1 TRUE

| c > 0 |
|---|
| TRUE |
| FALSE |
| FALSE |

- The left query gets indexes and predicate pushdown; the right one is one row per row, nothing to optimize. Rows on the left must equal TRUEs on the right: $1 = 1$.

[9] **[ESEC/FSE'20]** M. Rigger, Z. Su. "*Detecting Optimization Bugs in Database Engines via Non-Optimizing Reference Engine Construction.*"

# Examples: Database Engines

A query silently returns the **wrong rows**. No crash, and no expected result set. **NoREC**: ask the **same engine** twice – optimizing, and not.[9]

t

| c |
|---|
| 3 |
| 0 |
| -5 |

```
SELECT * FROM t
WHERE c > 0;
```

**Result**: **1** row

| c |
|---|
| 3 |

```
SELECT (c > 0)
FROM t;
```

**Result**: **1** TRUE

| c > 0 |
|---|
| TRUE |
| FALSE |
| FALSE |

- The left query gets indexes and predicate pushdown; the right one is one row per row, nothing to optimize. Rows on the left must equal TRUEs on the right: $1 = 1$.
- **159** bugs in PostgreSQL, MariaDB, SQLite, and CockroachDB.

[9] **[ESEC/FSE'20]** M. Rigger, Z. Su. "*Detecting Optimization Bugs in Database Engines via Non-Optimizing Reference Engine Construction.*"

# Examples: Database Engines

**Ternary Logic Partitioning (TLP)**: in every row, `c > 0` is `TRUE`, `FALSE`, or `NULL`. So three queries add up to the original.[10]

```
SELECT * FROM t;                          -- original
SELECT * FROM t WHERE c > 0;              -- TRUE  rows
SELECT * FROM t WHERE NOT (c > 0);        -- FALSE rows
SELECT * FROM t WHERE (c > 0) IS NULL;    -- NULL  rows
```

[10] **[OOPSLA'20]** M. Rigger, Z. Su. "*Finding Bugs in Database Systems via Query Partitioning.*"

# Examples: Database Engines

**Ternary Logic Partitioning (TLP)**: in every row, `c > 0` is `TRUE`, `FALSE`, or `NULL`. So three queries add up to the original.[10]

```
SELECT * FROM t;                          -- original
SELECT * FROM t WHERE c > 0;              -- TRUE  rows
SELECT * FROM t WHERE NOT (c > 0);        -- FALSE rows
SELECT * FROM t WHERE (c > 0) IS NULL;    -- NULL  rows
```

**Result**:

| c |
|---|
| 3 |
| NULL |
| -5 |

$=$

| c |
|---|
| 3 |

$\cup$

| c |
|---|
| -5 |

$\cup$

| c |
|---|
| NULL |

[10] **[OOPSLA'20]** M. Rigger, Z. Su. "*Finding Bugs in Database Systems via Query Partitioning.*"

# Examples: Database Engines

**Ternary Logic Partitioning (TLP)**: in every row, `c > 0` is `TRUE`, `FALSE`, or `NULL`. So three queries add up to the original.[10]

```
SELECT * FROM t;                          -- original
SELECT * FROM t WHERE c > 0;              -- TRUE  rows
SELECT * FROM t WHERE NOT (c > 0);        -- FALSE rows
SELECT * FROM t WHERE (c > 0) IS NULL;    -- NULL  rows
```

**Result**:

| c |
|---|
| 3 |
| NULL |
| -5 |

$=$

| c |
|---|
| 3 |

$\cup$

| c |
|---|
| -5 |

$\cup$

| c |
|---|
| NULL |

- $3 = 1 + 1 + 1$ rows. An engine that forgets that `NULL > 0` is `NULL` returns 2 – the typical bug.

[10] **[OOPSLA'20]** M. Rigger, Z. Su. "*Finding Bugs in Database Systems via Query Partitioning.*"

# Examples: Database Engines

**Ternary Logic Partitioning (TLP)**: in every row, `c > 0` is `TRUE`, `FALSE`, or `NULL`. So three queries add up to the original.[10]

```
SELECT * FROM t;                          -- original
SELECT * FROM t WHERE c > 0;              -- TRUE  rows
SELECT * FROM t WHERE NOT (c > 0);        -- FALSE rows
SELECT * FROM t WHERE (c > 0) IS NULL;    -- NULL  rows
```

**Result**:

| c |
|---|
| 3 |
| NULL |
| -5 |

$=$

| c |
|---|
| 3 |

$\cup$

| c |
|---|
| -5 |

$\cup$

| c |
|---|
| NULL |

- $3 = 1 + 1 + 1$ rows. An engine that forgets that `NULL > 0` is `NULL` returns 2 – the typical bug.
- The three parts are **harder** queries than the original – the relation **makes** hard queries.

[10] **[OOPSLA'20]** M. Rigger, Z. Su. "*Finding Bugs in Database Systems via Query Partitioning.*"

# Examples: Web Browser Debugger

Two ways of reaching the **same program point** must show the **same state**. That is the relation; no expected output is needed.[11]

```
1  // Original input:
2  var a = 5;      //  (i) pauses --> continue
3  var slideOverMe;
4  var C = class{};// (ii) pauses --> continue
5  var b = 42;     //(iii) pauses --> continue
```

```
1  // Transformed input:
2  var a = 5;      //  (i) pauses --> continue
3     var slideOverMe;
4  var C = class{};//      (no pausing)
5  var b = 42;     // (ii) pauses
```

In the upper case, a breakpoint is originally set at **line 3**, but debugger **slides** it to **line 4**. In the lower case, a breakpoint is **directly** set at **line 4**.

[11] **[ISSTA'19]** S. Tolksdorf, D. Lehmann, M. Pradel. "*Interactive Metamorphic Testing of Debuggers.*"

# Examples: Web Browser Debugger

Two ways of reaching the **same program point** must show the **same state**. That is the relation; no expected output is needed.[11]

```
1  // Original input:
2  var a = 5;      //  (i) pauses --> continue
3  var slideOverMe;
4  var C = class{};// (ii) pauses --> continue
5  var b = 42;     //(iii) pauses --> continue
```

```
1  // Transformed input:
2  var a = 5;      //  (i) pauses --> continue
3     var slideOverMe;
4  var C = class{};//      (no pausing)
5  var b = 42;     // (ii) pauses
```

In the upper case, a breakpoint is originally set at **line 3**, but debugger **slides** it to **line 4**. In the lower case, a breakpoint is **directly** set at **line 4**.

Both should pause at line 4. Chromium **did not** in the lower case – a debugger bug.

[11] **[ISSTA'19]** S. Tolksdorf, D. Lehmann, M. Pradel. "*Interactive Metamorphic Testing of Debuggers.*"

# Examples: Compilers

**Equivalence Modulo Inputs (EMI)**: $P'$ is equivalent to $P$ **modulo** a set of inputs $I$ when[12]

$$\forall i \in I.\ P(i) = P'(i)$$

$P$, run on $I = \{1, 2, 3\}$

```
int f(int x) {
  if (x > 0)
    return x * 2;
  return x - 1;  // never runs
}
```

$P'$: the dead line deleted

```
int f(int x) {
  if (x > 0)
    return x * 2;

}
```

[12] **[PLDI'14]** V. Le, M. Afshari, Z. Su. "*Compiler Validation via Equivalence Modulo Inputs.*"

# Examples: Compilers

**Equivalence Modulo Inputs (EMI)**: $P'$ is equivalent to $P$ **modulo** a set of inputs $I$ when[12]

$$\forall i \in I.\ P(i) = P'(i)$$

$P$, run on $I = \{1, 2, 3\}$

```
int f(int x) {
  if (x > 0)
    return x * 2;
  return x - 1;  // never runs
}
```

$P'$: the dead line deleted

```
int f(int x) {
  if (x > 0)
    return x * 2;

}
```

- Run $P$ on $I$, record coverage, delete statements that **never ran**, at random. On $I$ the two programs must print the **same** thing.

[12] **[PLDI'14]** V. Le, M. Afshari, Z. Su. "*Compiler Validation via Equivalence Modulo Inputs.*"

# Examples: Compilers

**Equivalence Modulo Inputs (EMI)**: $P'$ is equivalent to $P$ **modulo** a set of inputs $I$ when[12]

$$\forall i \in I.\ P(i) = P'(i)$$

$P$, run on $I = \{1, 2, 3\}$

```
int f(int x) {
  if (x > 0)
    return x * 2;
  return x - 1;  // never runs
}
```

$P'$: the dead line deleted

```
int f(int x) {
  if (x > 0)
    return x * 2;

}
```

- Run $P$ on $I$, record coverage, delete statements that **never ran**, at random. On $I$ the two programs must print the **same** thing.
- Compile both with the compiler under test. Different output on `f(2)` means the compiler mishandled the dead code – **a compiler bug**, with no expected output anywhere. **147** of them in GCC and LLVM.

[12] **[PLDI'14]** V. Le, M. Afshari, Z. Su. "*Compiler Validation via Equivalence Modulo Inputs.*"

# Examples: Compilers

Deleting dead code reaches only the **front end**. To reach the **optimizer**, change code that **runs** – using profiled values, so that the behaviour stays the same.[13]

```
int x = 5;                              // profile: x is always 5 here
y = x + 1;                              // original, live
```

[13] **[OOPSLA'16]** C. Sun, V. Le, Z. Su. "*Finding Compiler Bugs via Live Code Mutation.*"

# Examples: Compilers

Deleting dead code reaches only the **front end**. To reach the **optimizer**, change code that **runs** – using profiled values, so that the behaviour stays the same.[13]

```
int x = 5;                                  // profile: x is always 5 here
y = x + 1;                                  // original, live
```

```
if (x > 100) { y = 0; }                     // FCB: always-false block
if (x == 5)  { y = x + 1; }                 // TG:  always-true guard
if (x == 5)  { x += 3; y = x - 2; x -= 3; }  // TCB: reverted effects
```

[13] **[OOPSLA'16]** C. Sun, V. Le, Z. Su. "*Finding Compiler Bugs via Live Code Mutation.*"

# Examples: Compilers

Deleting dead code reaches only the **front end**. To reach the **optimizer**, change code that **runs** – using profiled values, so that the behaviour stays the same.[13]

```
int x = 5;                                  // profile: x is always 5 here
y = x + 1;                                  // original, live
```

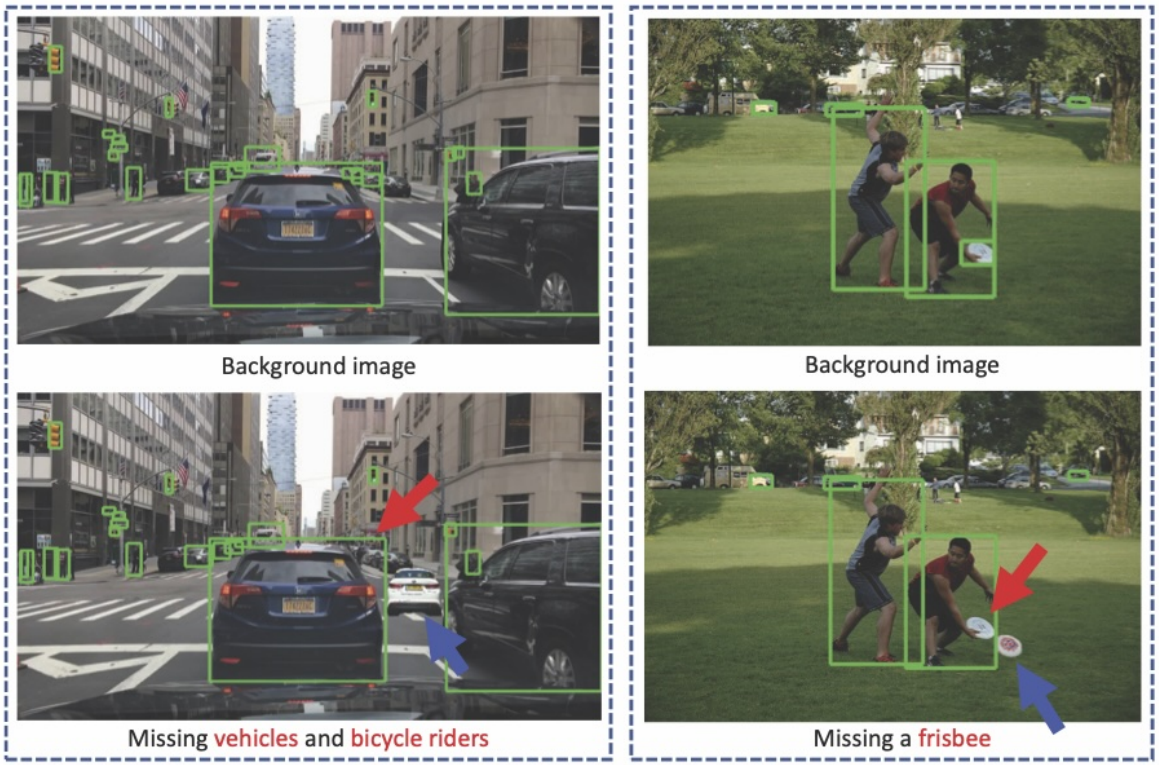
```
if (x > 100) { y = 0; }                     // FCB: always-false block
if (x == 5)  { y = x + 1; }                 // TG:  always-true guard
if (x == 5)  { x += 3; y = x - 2; x -= 3; }  // TCB: reverted effects
```

Four programs, one behaviour on the profiled inputs. The optimizer sees **different** code each time; a different output is a bug.

[13] **[OOPSLA'16]** C. Sun, V. Le, Z. Su. "*Finding Compiler Bugs via Live Code Mutation.*"

# Examples: Object Detection

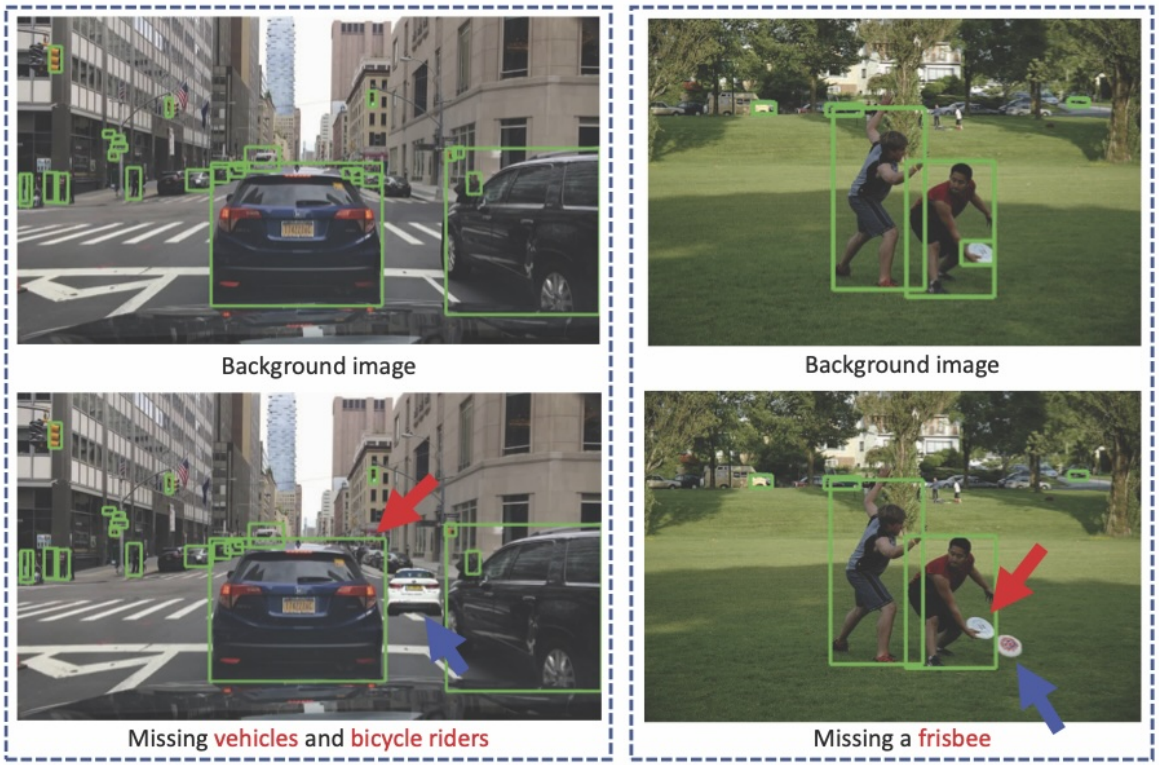
Paste an object that the detector already found into another image: it must **still** find everything it found before.[14]




[14] **[ASE'20]** S. Wang, Z. Su. "*Metamorphic Object Insertion for Testing Object Detection Systems.*"

# Examples: Object Detection

Paste an object that the detector already found into another image: it must **still** find everything it found before.[14]




Inserted objects are marked by **blue arrows**. The detector loses objects it had already found – a failure, with **no labelled ground truth** anywhere.

[14] **[ASE'20]** S. Wang, Z. Su. "*Metamorphic Object Insertion for Testing Object Detection Systems.*"

# Where Do Relations Come From?

Every example so far needed **someone** to see the relation first.

| **Source** | **Example** |
|---|---|
| Mathematics | $\sin(x) = \sin(\pi - x)$ |
| Documentation | Javadoc "equivalent to `Math.abs(x)`" $\rightarrow$ an assertion (MeMo)[15] |
| Search | fix the shape $f(x) = \alpha x + \beta$, find $\alpha, \beta$ – next |
| A language model | read the code, propose one – the LLM lecture |

[15] **[JSS'21]** A. Blasi, A. Gorla, M. D. Ernst, M. Pezzè, A. Carzaniga. "*MeMo: Automatically Identifying Metamorphic Relations in Javadoc Comments for Test Automation.*"

# Learning Metamorphic Relationships

Writing relations by hand is the bottleneck. Can we **search** for them?[16]

[16] **[ASE'14]** J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, H. Mei. "*Search-Based Inference of Polynomial Metamorphic Relations.*"

# Learning Metamorphic Relationships

PLRG

Writing relations by hand is the bottleneck. Can we **search** for them?[16]

- Consider a **linear metamorphic relationship** for program $p : X \to Y$:

$$f(x) = \alpha x + \beta \qquad\qquad c_1 y + c_2 y' + d = 0$$

where $y = p(x)$ and $y' = p(f(x))$.

[16] **[ASE'14]** J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, H. Mei. "*Search-Based Inference of Polynomial Metamorphic Relations.*"

# Learning Metamorphic Relationships

PLRG

Writing relations by hand is the bottleneck. Can we **search** for them?[16]

- Consider a **linear metamorphic relationship** for program $p : X \to Y$:

$$f(x) = \alpha x + \beta \qquad\qquad c_1 y + c_2 y' + d = 0$$

  where $y = p(x)$ and $y' = p(f(x))$.

- For example, $\sin(x) = \sin(\pi - x)$ is a linear metamorphic relationship with $\alpha = -1, \beta = \pi, c_1 = 1, c_2 = -1, d = 0$.

[16] **[ASE'14]** J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, H. Mei. "*Search-Based Inference of Polynomial Metamorphic Relations.*"

# Learning Metamorphic Relationships

PLRG

Writing relations by hand is the bottleneck. Can we **search** for them?[16]

- Consider a **linear metamorphic relationship** for program $p : X \to Y$:

$$f(x) = \alpha x + \beta \qquad\qquad c_1 y + c_2 y' + d = 0$$

  where $y = p(x)$ and $y' = p(f(x))$.

- For example, $\sin(x) = \sin(\pi - x)$ is a linear metamorphic relationship with $\alpha = -1, \beta = \pi, c_1 = 1, c_2 = -1, d = 0$.

- Then, the learning problem is to find the coefficients $\alpha, \beta, c_1, c_2, d$ that satisfy the linear metamorphic relationship.

[16] **[ASE'14]** J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, H. Mei. "*Search-Based Inference of Polynomial Metamorphic Relations.*"

# Learning Metamorphic Relationships

PLRG

Writing relations by hand is the bottleneck. Can we **search** for them?[16]

- Consider a **linear metamorphic relationship** for program $p : X \to Y$:

$$f(x) = \alpha x + \beta \qquad\qquad c_1 y + c_2 y' + d = 0$$

  where $y = p(x)$ and $y' = p(f(x))$.

- For example, $\sin(x) = \sin(\pi - x)$ is a linear metamorphic relationship with $\alpha = -1, \beta = \pi, c_1 = 1, c_2 = -1, d = 0$.

- Then, the learning problem is to find the coefficients $\alpha, \beta, c_1, c_2, d$ that satisfy the linear metamorphic relationship.

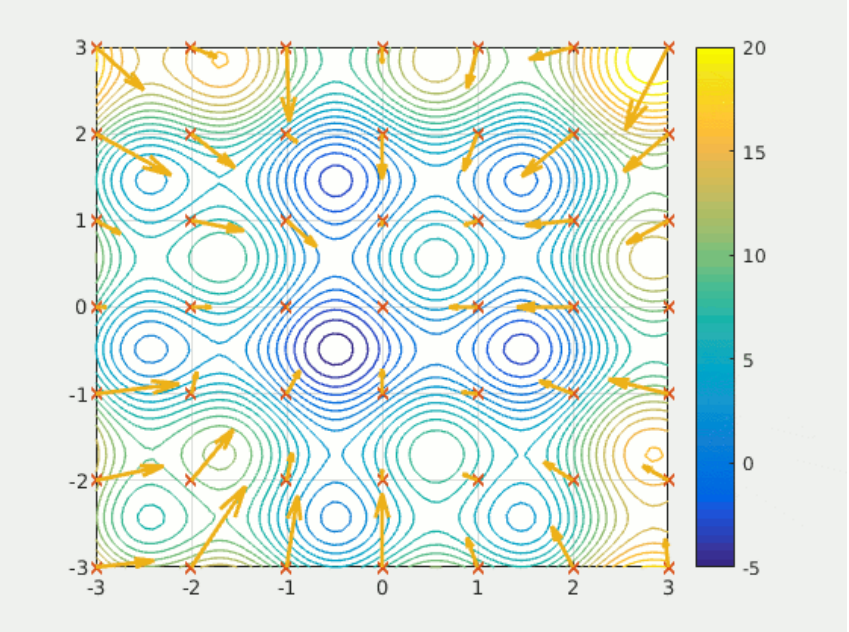
- The key idea is a **search-based** approach to find the coefficients using a **particle swarm optimization (PSO)** algorithm.

[16] **[ASE'14]** J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, H. Mei. "*Search-Based Inference of Polynomial Metamorphic Relations.*"

# Recall – Particle Swarm Optimization

$$x_i^{t+1} = x_i^t + v_i^t \qquad v_i^{t+1} = ① \; w v_i^t + ② \; c_1(p_i - x_i^t) + ③ \; c_2(g - x_i^t)$$

- ① keep flying the same way
- ② return to my own best, $p_i$
- ③ follow the flock's best, $g$




Here a **particle** is a candidate relation $(\alpha, \beta, c_1, c_2, d)$, and its **fitness** is how many inputs satisfy it.

# Learning Metamorphic Relationships

- It defines the **fitness function** for the PSO algorithm as the number of inputs that satisfy the formula with the coefficients with $n$ inputs:

$$\text{fitness} = \sum_{i=1}^{n} \left[ c_1 y_i + c_2 y_i' + d = 0 \right] \qquad [\varphi] = \begin{cases} 1 & \text{if } \varphi \\ 0 & \text{otherwise} \end{cases}$$

where $x_i$ is the $i$-th input and $y_i = p(x_i)$, $y_i' = p(f(x_i))$.

# Learning Metamorphic Relationships

- It defines the **fitness function** for the PSO algorithm as the number of inputs that satisfy the formula with the coefficients with $n$ inputs:

$$\text{fitness} = \sum_{i=1}^{n} \left[ c_1 y_i + c_2 y_i' + d = 0 \right] \qquad [\varphi] = \begin{cases} 1 & \text{if } \varphi \\ 0 & \text{otherwise} \end{cases}$$

  where $x_i$ is the $i$-th input and $y_i = p(x_i)$, $y_i' = p(f(x_i))$.

- When $c_1$ and $c_2$ are between $[-\phi, \phi]$ with a **small threshold** $\phi$, the algorithm resets them to a new random value to prevent producing the **degenerate solutions**.

# Learning Metamorphic Relationships

- It defines the **fitness function** for the PSO algorithm as the number of inputs that satisfy the formula with the coefficients with $n$ inputs:

$$\text{fitness} = \sum_{i=1}^{n} [\, c_1 y_i + c_2 y_i' + d = 0 \,] \qquad [\, \varphi \,] = \begin{cases} 1 & \text{if } \varphi \\ 0 & \text{otherwise} \end{cases}$$

  where $x_i$ is the $i$-th input and $y_i = p(x_i)$, $y_i' = p(f(x_i))$.

- When $c_1$ and $c_2$ are between $[-\phi, \phi]$ with a **small threshold** $\phi$, the algorithm resets them to a new random value to prevent producing the **degenerate solutions**.

- We can extend it into a **quadratic metamorphic relationship**:

$$f(x) = \alpha x + \beta \qquad c_1 y^2 + c_2 y y' + c_3 y'^2 + d_1 y + d_2 y' + e = 0$$

  where $y = p(x)$ and $y' = p(f(x))$.

# What a Relation Cannot Tell You

Implement sin as the constant 0. Which relations notice?

| **Relation** | $\sin \equiv 0$ |
|---|---|
| $\sin(x) = \sin(\pi - x)$ | ✓ passes |
| $\sin(x) = -\sin(x + \pi)$ | ✓ passes |
| $\sin^2(x) + \sin^2(x + \frac{\pi}{2}) = 1$ | ✗ $0 \neq 1$ |

# What a Relation Cannot Tell You

Implement sin as the constant 0. Which relations notice?

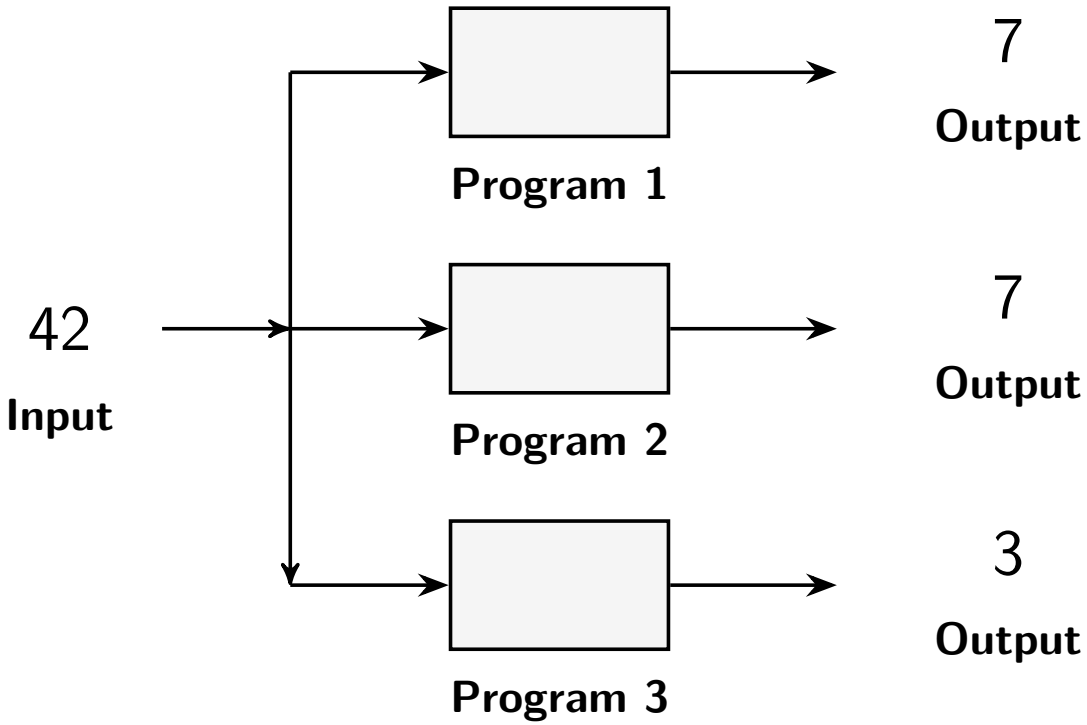
| **Relation** | $\sin \equiv 0$ |
|---|---|
| $\sin(x) = \sin(\pi - x)$ | ✓ passes |
| $\sin(x) = -\sin(x + \pi)$ | ✓ passes |
| $\sin^2(x) + \sin^2(x + \frac{\pi}{2}) = 1$ | ✗ $0 \neq 1$ |

A weak relation passes a wrong program. Use **several**. And when one fails, it says only that $p(x)$ and $p(f(x))$ disagree – **not which is wrong**.

# Contents

# Differential Testing

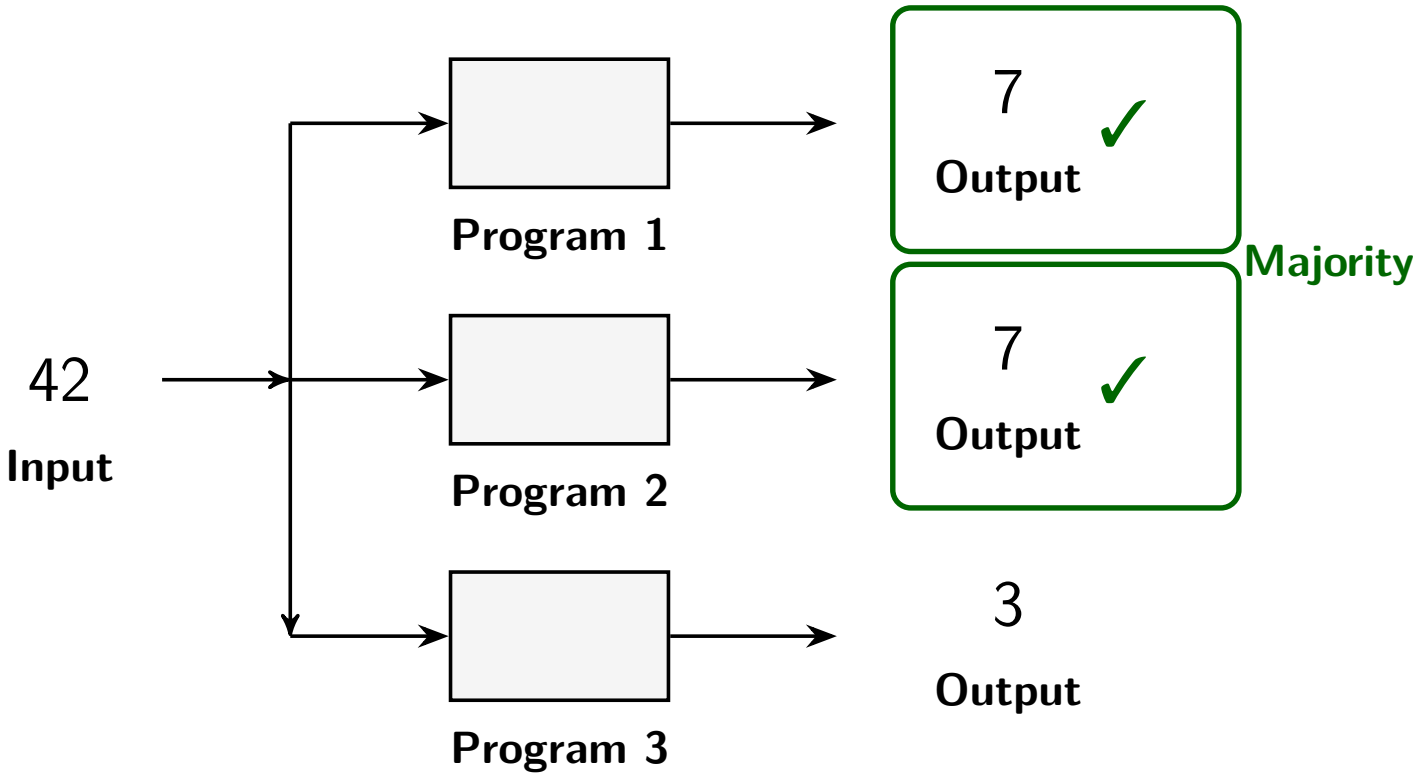
Run the **same input** through **several implementations** of the same thing. No relation to write: a **disagreement** is the report.[17]




[17] **[DTJ'98]** W. M. McKeeman. "*Differential Testing for Software.*"

# Differential Testing

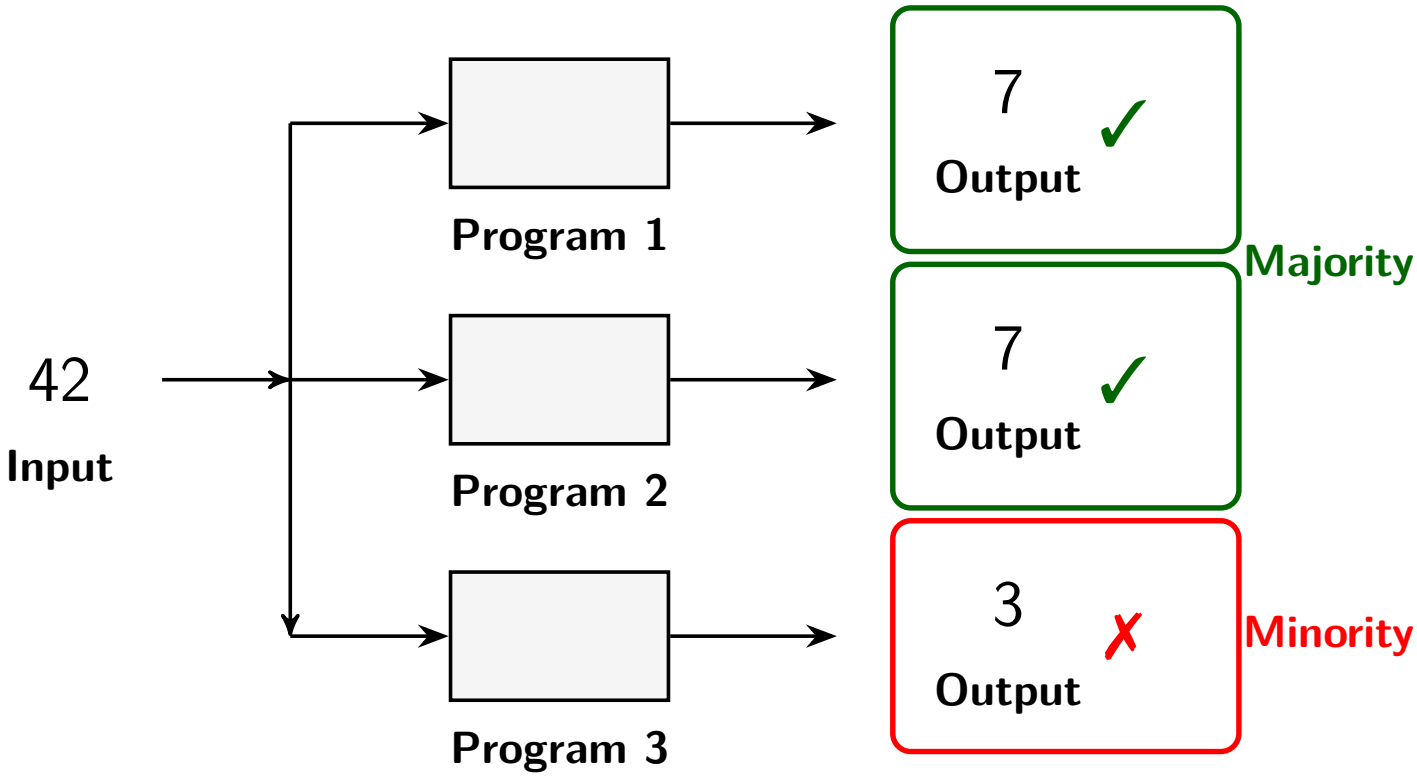
Run the **same input** through **several implementations** of the same thing. No relation to write: a **disagreement** is the report.[17]




[17] **[DTJ'98]** W. M. McKeeman. "*Differential Testing for Software.*"

# Differential Testing

Run the **same input** through **several implementations** of the same thing. No relation to write: a **disagreement** is the report.[17]

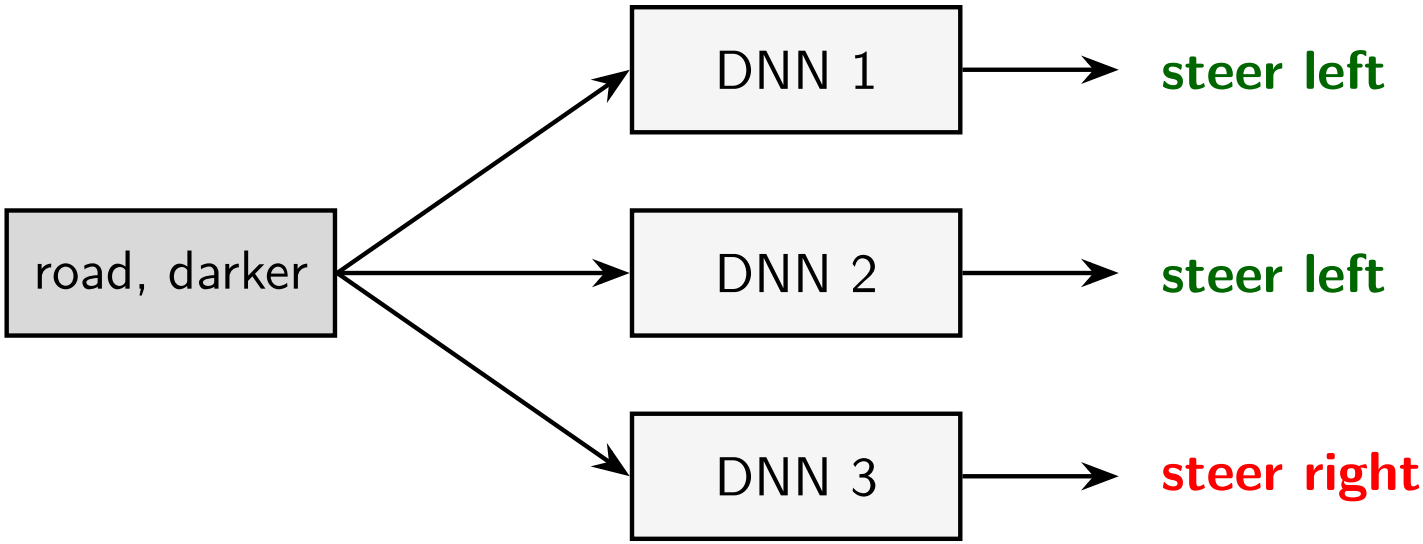



The **cross-reference oracle**: the majority is more likely to be right. Nothing here knows that the answer **is** 7.

[17] **[DTJ'98]** W. M. McKeeman. "*Differential Testing for Software.*"

# Examples: Deep Neural Networks

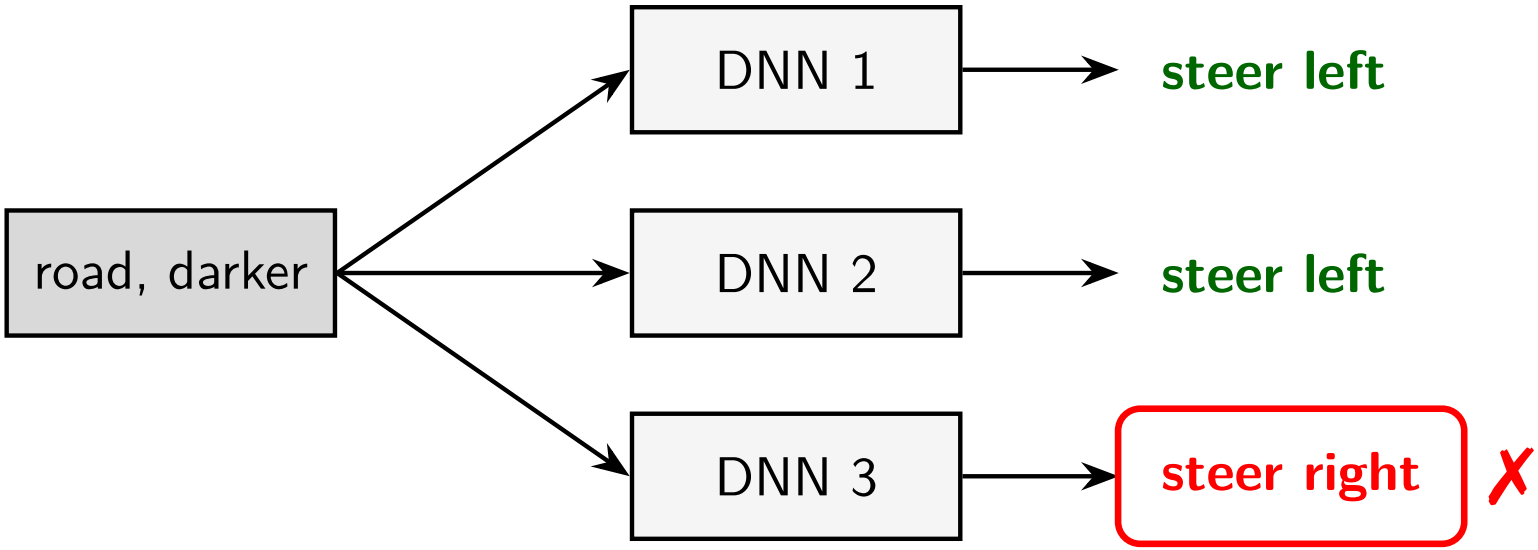
A DNN has no second implementation, so **train several**. **DeepXplore** feeds them the same image and looks for a disagreement.[18]




[18] **[SOSP'17]** K. Pei, Y. Cao, J. Yang, S. Jana. "*DeepXplore: Automated Whitebox Testing of Deep Learning Systems.*"

# Examples: Deep Neural Networks

A DNN has no second implementation, so **train several**. **DeepXplore** feeds them the same image and looks for a disagreement.[18]

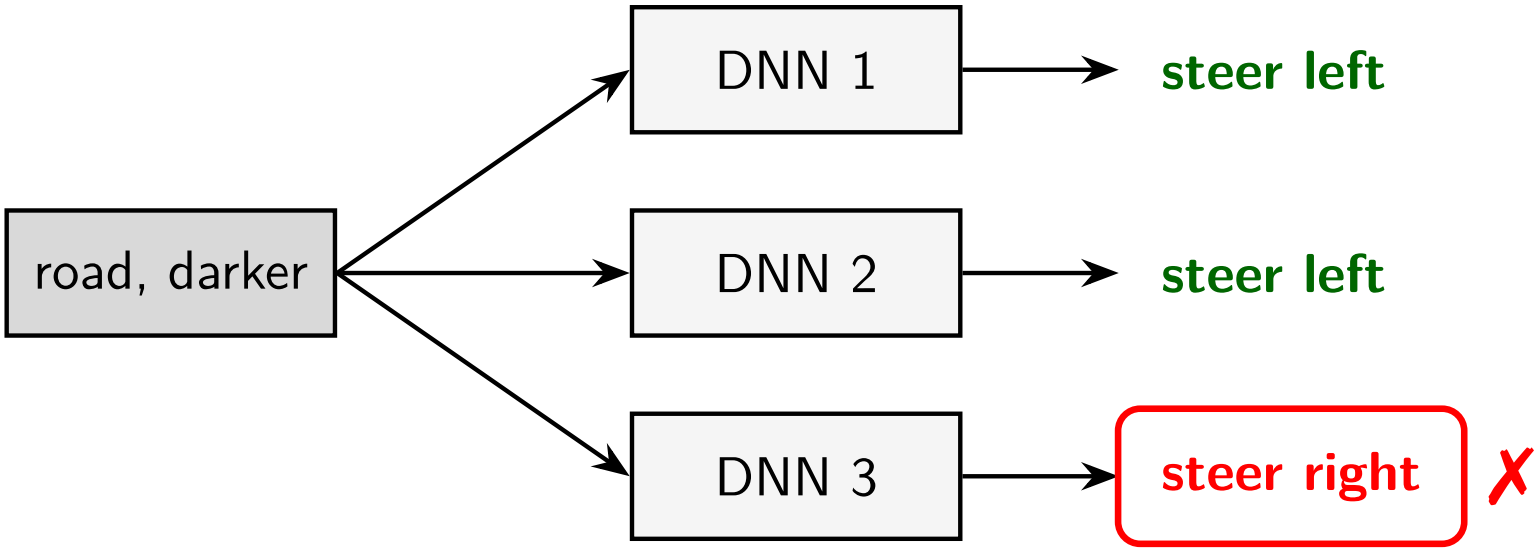



- Same image, slightly darker: two networks keep steering left, one turns into the guardrail. The **minority** is the failure; no label needed.

[18] **[SOSP'17]** K. Pei, Y. Cao, J. Yang, S. Jana. "*DeepXplore: Automated Whitebox Testing of Deep Learning Systems.*"

# Examples: Deep Neural Networks

A DNN has no second implementation, so **train several**. **DeepXplore** feeds them the same image and looks for a disagreement.[18]




- Same image, slightly darker: two networks keep steering left, one turns into the guardrail. The **minority** is the failure; no label needed.
- Which darkening? Gradient ascent on the **image**, pushing two numbers up at once: how much the networks **disagree**, and how many **not-yet-activated neurons** fire. **15** DNNs, thousands of such inputs.

[18] **[SOSP'17]** K. Pei, Y. Cao, J. Yang, S. Jana. "*DeepXplore: Automated Whitebox Testing of Deep Learning Systems.*"

# Examples: Nezha

Fuzzing several implementations at once: which inputs are worth **keeping**? Not the ones that cover new code in **one** program – the ones that make the programs behave **differently from each other**.[19]

| **Input** | **OpenSSL** | **LibreSSL** | **BoringSSL** | **Tuple** |
|---|---|---|---|---|
| cert 1 | accept | accept | accept | (A, A, A) new $\rightarrow$ keep |
| cert 2 | accept | accept | accept | (A, A, A) seen $\rightarrow$ drop |
| cert 3 | accept | reject | accept | (A, R, A) new $\rightarrow$ **report** |

[19] **[S&P'17]** T. Petsios, A. Tang, S. Stolfo, A. Keromytis, S. Jana. "*NEZHA: Efficient Domain-Independent Differential Testing.*"

# Examples: Nezha

Fuzzing several implementations at once: which inputs are worth **keeping**? Not the ones that cover new code in **one** program – the ones that make the programs behave **differently from each other**.[19]

| Input | OpenSSL | LibreSSL | BoringSSL | Tuple |
|---|---|---|---|---|
| cert 1 | accept | accept | accept | (A, A, A) new $\rightarrow$ keep |
| cert 2 | accept | accept | accept | (A, A, A) seen $\rightarrow$ drop |
| cert 3 | accept | reject | accept | (A, R, A) new $\rightarrow$ **report** |

- $\delta$-**diversity** is the number of distinct tuples seen so far. An input is interesting when it adds one. Tuples of **outputs**, or tuples of **paths**: a new combination counts even if every single path is old.

[19] **[S&P'17]** T. Petsios, A. Tang, S. Stolfo, A. Keromytis, S. Jana. "*NEZHA: Efficient Domain-Independent Differential Testing.*"

# Examples: Nezha

Fuzzing several implementations at once: which inputs are worth **keeping**? Not the ones that cover new code in **one** program – the ones that make the programs behave **differently from each other**.[19]

| Input | OpenSSL | LibreSSL | BoringSSL | Tuple |
|---|---|---|---|---|
| cert 1 | accept | accept | accept | (A, A, A) new $\rightarrow$ keep |
| cert 2 | accept | accept | accept | (A, A, A) seen $\rightarrow$ drop |
| cert 3 | accept | reject | accept | (A, R, A) new $\rightarrow$ **report** |

- $\delta$-**diversity** is the number of distinct tuples seen so far. An input is interesting when it adds one. Tuples of **outputs**, or tuples of **paths**: a new combination counts even if every single path is old.
- **778** previously unknown discrepancies across SSL/TLS libraries, PDF viewers, and ELF and XZ parsers.

[19] **[S&P'17]** T. Petsios, A. Tang, S. Stolfo, A. Keromytis, S. Jana. "*NEZHA: Efficient Domain-Independent Differential Testing.*"

# Examples: Binary Lifters

A **lifter** turns machine code into an IR for analysis. Three lifters, one instruction, three IRs – which must mean the **same** thing.[20]

| **Lifter** | **Summary of `shl eax, cl`** | $eax = 1, cl = 40$ |
|---|---|---|
| A | $eax' = eax \ll (cl \,\&\, 31)$ | 256 |
| B | $eax' = eax \ll (cl \,\&\, 31)$ | 256 |
| C | $eax' = eax \ll cl$ | 0 |

[20] **[ASE'17]** S. Kim, M. Faerevaag, M. Jung, S. Jung, D. Oh, J. Lee, S. K. Cha. "*Testing Intermediate Representations for Binary Analysis.*"

# Examples: Binary Lifters

A **lifter** turns machine code into an IR for analysis. Three lifters, one instruction, three IRs – which must mean the **same** thing.[20]

| **Lifter** | **Summary of `shl eax, cl`** | $eax = 1, cl = 40$ |
|---|---|---|
| A | $eax' = eax \ll (cl \,\&\, 31)$ | 256 |
| B | $eax' = eax \ll (cl \,\&\, 31)$ | 256 |
| C | $eax' = eax \ll cl$ | 0 |

- x86 masks the shift count to 5 bits; lifter C forgot. **MeanDiff** summarizes each IR **symbolically** and asks an SMT solver whether the summaries can differ. Here: yes, for any $cl \geq 32$ – a **lifting bug**.

[20] **[ASE'17]** S. Kim, M. Faerevaag, M. Jung, S. Jung, D. Oh, J. Lee, S. K. Cha. "*Testing Intermediate Representations for Binary Analysis.*"

# Examples: Binary Lifters

A **lifter** turns machine code into an IR for analysis. Three lifters, one instruction, three IRs – which must mean the **same** thing.[20]

| **Lifter** | **Summary of `shl eax, cl`** | $eax = 1, cl = 40$ |
|---|---|---|
| A | $eax' = eax \ll (cl \,\&\, 31)$ | 256 |
| B | $eax' = eax \ll (cl \,\&\, 31)$ | 256 |
| C | $eax' = eax \ll cl$ | 0 |

- x86 masks the shift count to 5 bits; lifter C forgot. **MeanDiff** summarizes each IR **symbolically** and asks an SMT solver whether the summaries can differ. Here: yes, for any $cl \geq 32$ – a **lifting bug**.
- No one wrote down what `shl` means. The other two lifters did. **323,928** x86 instructions tested this way.

[20] **[ASE'17]** S. Kim, M. Faerevaag, M. Jung, S. Jung, D. Oh, J. Lee, S. K. Cha. "*Testing Intermediate Representations for Binary Analysis.*"

# Examples: JIT Compilers

Every JS engine has an **interpreter** and a **just-in-time (JIT) compiler** that compiles hot code while the program runs.

**JIT-Picking**: run the same program on the same engine twice, JIT **off** and JIT **on**. The results must match.[21]

[21] **[CCS'22]** L. Bernhard, T. Scharnowski, M. Schloegel, T. Blazytko, T. Holz. "*JIT-Picking: Differential Fuzzing of JavaScript Engines.*"

# Examples: JIT Compilers

Every JS engine has an **interpreter** and a **just-in-time (JIT) compiler** that compiles hot code while the program runs.

**JIT-Picking**: run the same program on the same engine twice, JIT **off** and JIT **on**. The results must match.[21]

- Do not `print` the result: a printed value is externally visible, so the JIT stops optimizing it, and the bug hides.

[21] **[CCS'22]** L. Bernhard, T. Scharnowski, M. Schloegel, T. Blazytko, T. Holz. "*JIT-Picking: Differential Fuzzing of JavaScript Engines.*"

# Examples: JIT Compilers

Every JS engine has an **interpreter** and a **just-in-time (JIT) compiler** that compiles hot code while the program runs.

**JIT-Picking**: run the same program on the same engine twice, JIT **off** and JIT **on**. The results must match.[21]

- Do not `print` the result: a printed value is externally visible, so the JIT stops optimizing it, and the bug hides.
- Instead, **hash** the local variables inside the engine and compare the two hashes.

[21] **[CCS'22]** L. Bernhard, T. Scharnowski, M. Schloegel, T. Blazytko, T. Holz. "*JIT-Picking: Differential Fuzzing of JavaScript Engines.*"

# Examples: JIT Compilers

JIT off is an oracle only if the JIT was ever **on** – and a random program is usually too short to be compiled. **FuzzJIT** wraps every sample:[22]

```
function f(a) { return (a | 0) * 2 + 1; }   // generated by the fuzzer

const before = f(3);                        // interpreter
for (let i = 0; i < 100000; i++) f(i);      // hot: the JIT compiles f
const after  = f(3);                        // JIT-compiled

if (before !== after) report();             // the oracle rides along
```

[22] **[USENIX Sec'23]** J. Wang, Z. Zhang, S. Liu, X. Du, J. Chen. "*FuzzJIT: Oracle-Enhanced Fuzzing for JavaScript Engine JIT Compiler.*"

# Examples: JIT Compilers

JIT off is an oracle only if the JIT was ever **on** – and a random program is usually too short to be compiled. **FuzzJIT** wraps every sample:[22]

```
function f(a) { return (a | 0) * 2 + 1; }   // generated by the fuzzer

const before = f(3);                        // interpreter
for (let i = 0; i < 100000; i++) f(i);      // hot: the JIT compiles f
const after  = f(3);                        // JIT-compiled

if (before !== after) report();             // the oracle rides along
```

- A JIT may change the **speed**, never the **result**. Same engine, two modes, one input.

[22] **[USENIX Sec'23]** J. Wang, Z. Zhang, S. Liu, X. Du, J. Chen. "*FuzzJIT: Oracle-Enhanced Fuzzing for JavaScript Engine JIT Compiler.*"

# Examples: JIT Compilers

JIT off is an oracle only if the JIT was ever **on** – and a random program is usually too short to be compiled. **FuzzJIT** wraps every sample:[22]

```
function f(a) { return (a | 0) * 2 + 1; }   // generated by the fuzzer

const before = f(3);                        // interpreter
for (let i = 0; i < 100000; i++) f(i);      // hot: the JIT compiles f
const after  = f(3);                        // JIT-compiled

if (before !== after) report();             // the oracle rides along
```

- A JIT may change the **speed**, never the **result**. Same engine, two modes, one input.
- One month, four engines: **10 / 5 / 2 / 16** new bugs in JavaScriptCore, V8, SpiderMonkey, and ChakraCore; three exploitable.

[22] **[USENIX Sec'23]** J. Wang, Z. Zhang, S. Liu, X. Du, J. Chen. "*FuzzJIT: Oracle-Enhanced Fuzzing for JavaScript Engine JIT Compiler.*"

# N+1-version Differential Testing

One **specification** (ECMA-262) and **many** engines that claim to implement it.[23]


**ECMAScript**


GraalVM

QuickJS

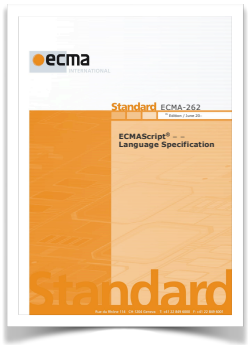
moddable

**JavaScript Engines**

[23] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

One **specification** (ECMA-262) and **many** engines that claim to implement it.[23]




**ECMAScript**


**Conformance**

GraalVM

QuickJS

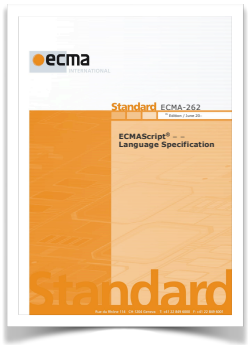
moddable

**JavaScript Engines**

[23] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

One **specification** (ECMA-262) and **many** engines that claim to implement it.[23]




**ECMAScript**

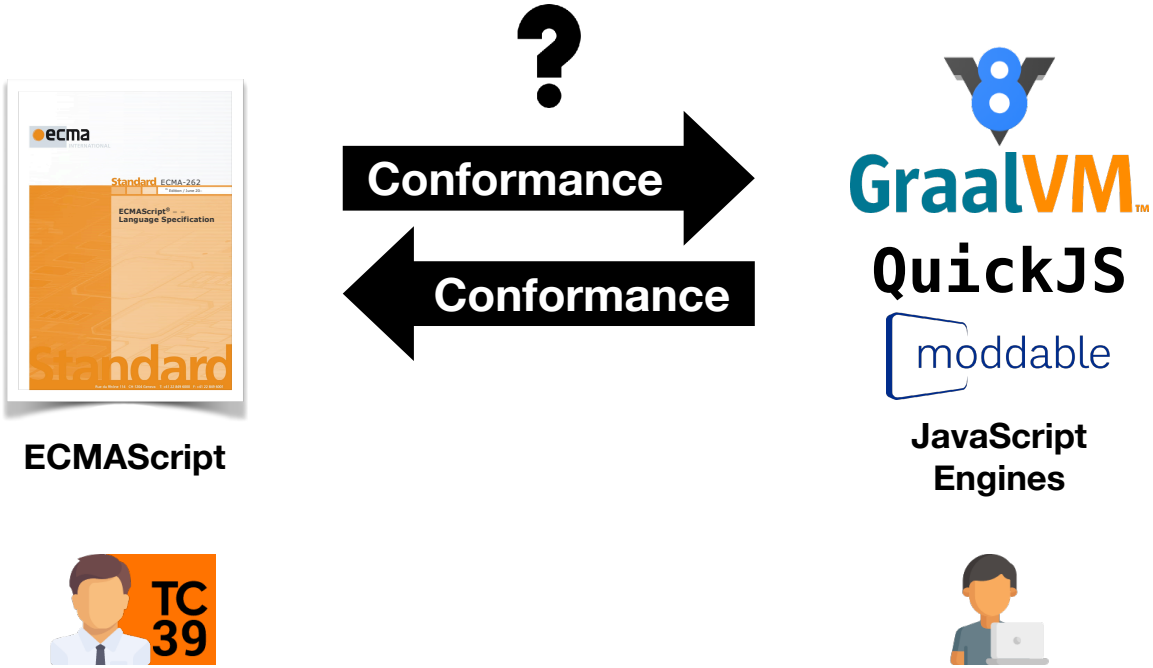



**JavaScript Engines**

[23] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

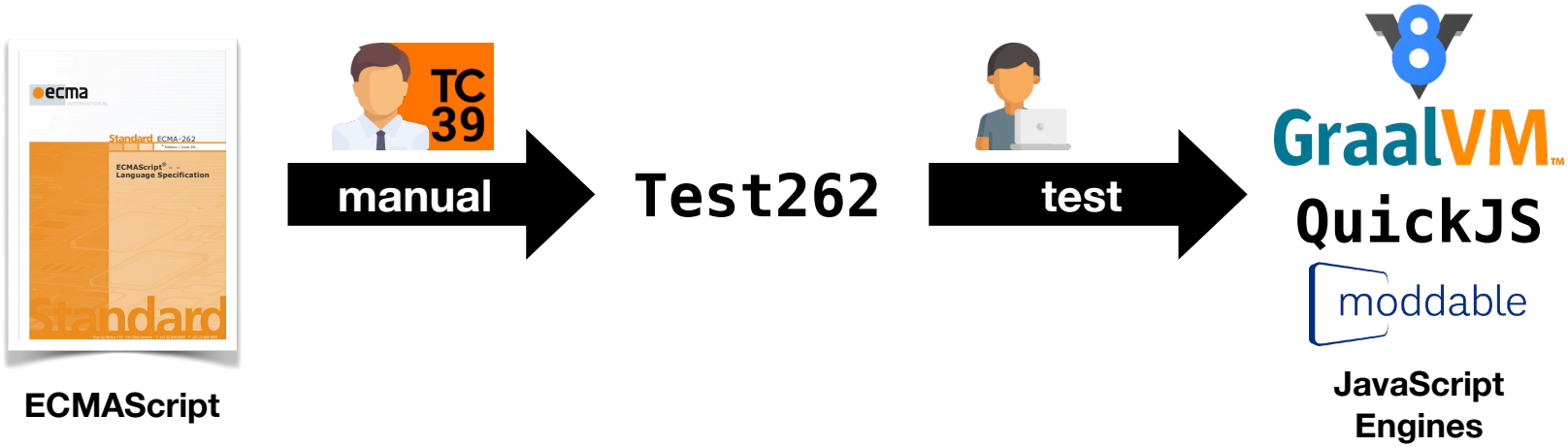
One **specification** (ECMA-262) and **many** engines that claim to implement it.[23]




[23] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

**Test262** is the conformance suite: hand-written tests that every engine is expected to pass.[24]




[24] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

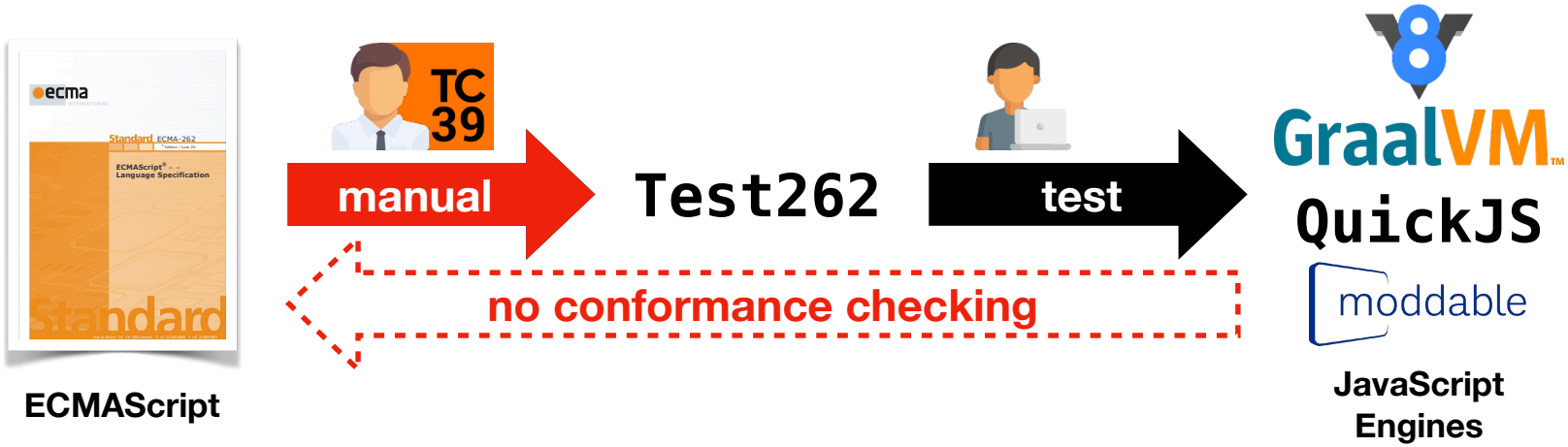
# N+1-version Differential Testing

**Test262** is the conformance suite: hand-written tests that every engine is expected to pass.[24]


[24] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

**Test262** is the conformance suite: hand-written tests that every engine is expected to pass.[24]

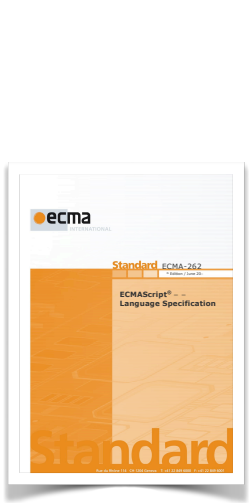



But the suite is written by hand, and the **specification itself** is never tested.

[24] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

Make the **mechanized specification** the $(N+1)$-th implementation: a disagreement now accuses either an **engine** or the **spec**.[25]




ECMAScript

Synthesize

Test

test

test

test

test

GraalVM

QuickJS

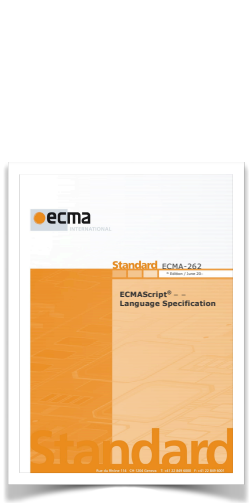
moddable

JavaScript Engines

[25] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

Make the **mechanized specification** the $(N+1)$-th implementation: a disagreement now accuses either an **engine** or the **spec**.[25]




ECMAScript

Synthesize

Test

test

test

test

test

GraalVM

QuickJS

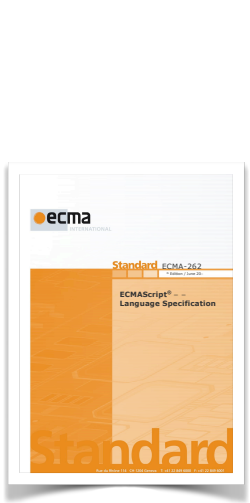
moddable

JavaScript Engines

[25] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

Make the **mechanized specification** the $(N+1)$-th implementation: a disagreement now accuses either an **engine** or the **spec**.[25]




ECMAScript

Synthesize

Test

test

test

test

test

GraalVM

QuickJS

moddable

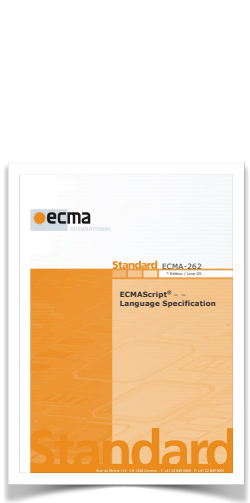
JavaScript Engines

An engine bug in

[25] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

Make the **mechanized specification** the $(N+1)$-th implementation: a disagreement now accuses either an **engine** or the **spec**.[25]




ECMAScript

Synthesize

Test

test

test

test

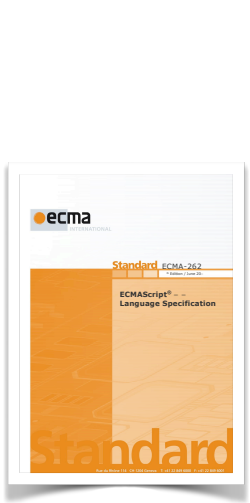
test

JavaScript Engines

[25] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

Make the **mechanized specification** the $(N+1)$-th implementation: a disagreement now accuses either an **engine** or the **spec**.[25]




ECMAScript

Synthesize

Test

test

test

test

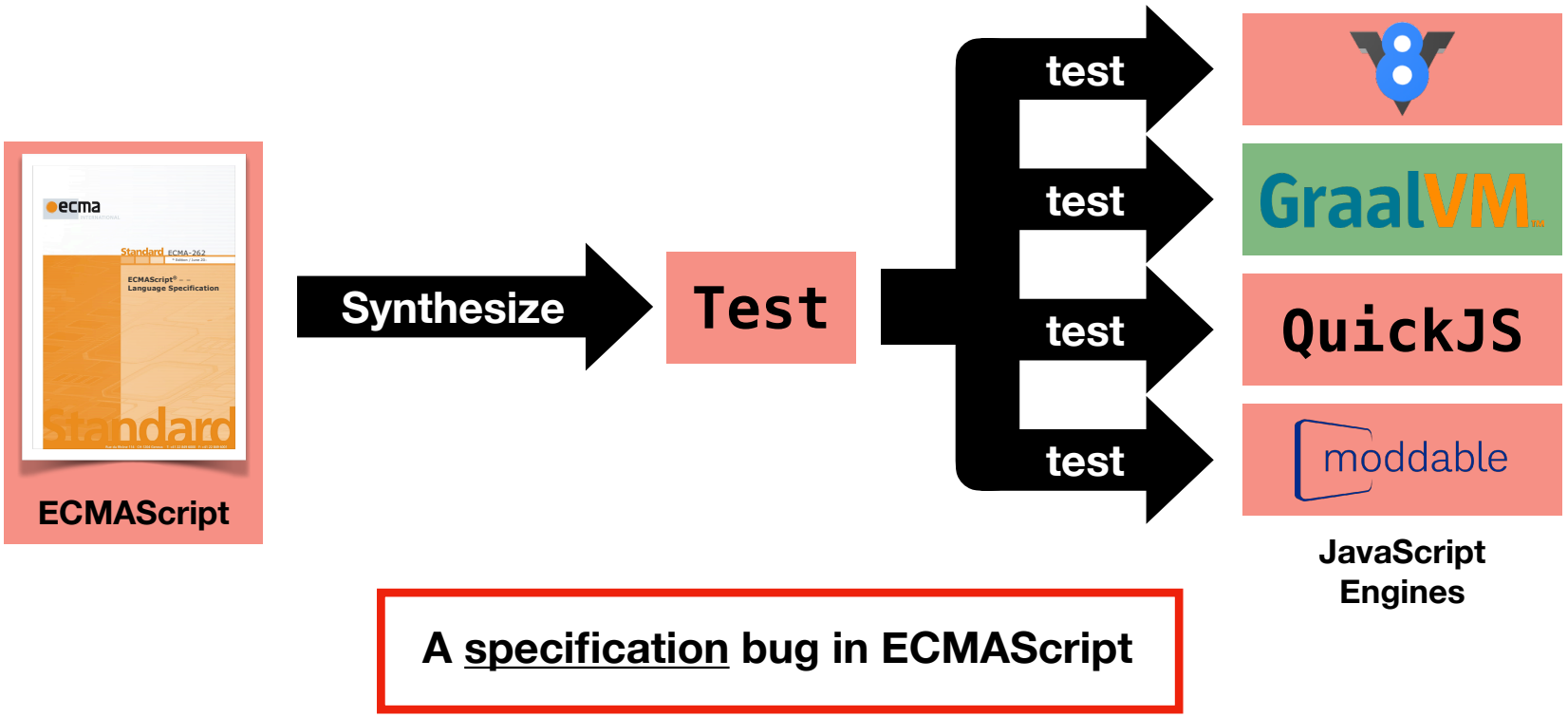
test

JavaScript Engines

[25] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

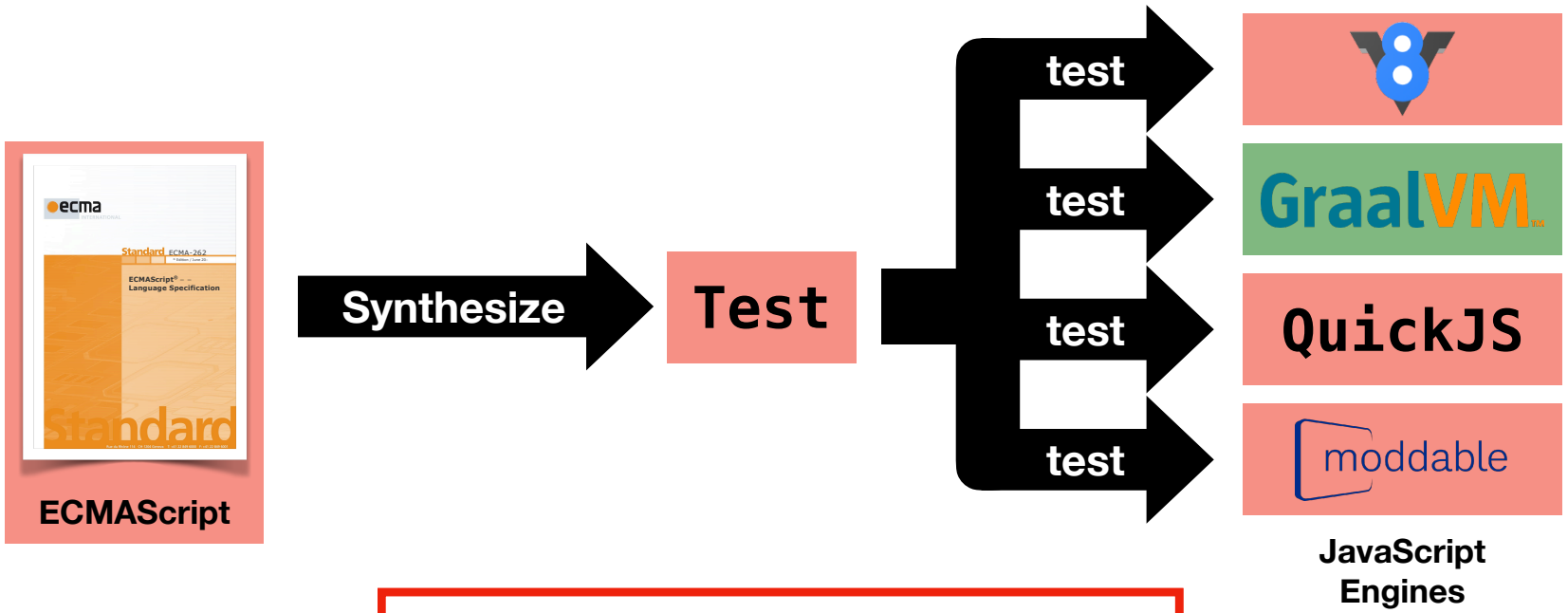
Make the **mechanized specification** the $(N+1)$-th implementation: a disagreement now accuses either an **engine** or the **spec**.[25]




[25] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

Make the **mechanized specification** the $(N+1)$-th implementation: a disagreement now accuses either an **engine** or the **spec**.[25]



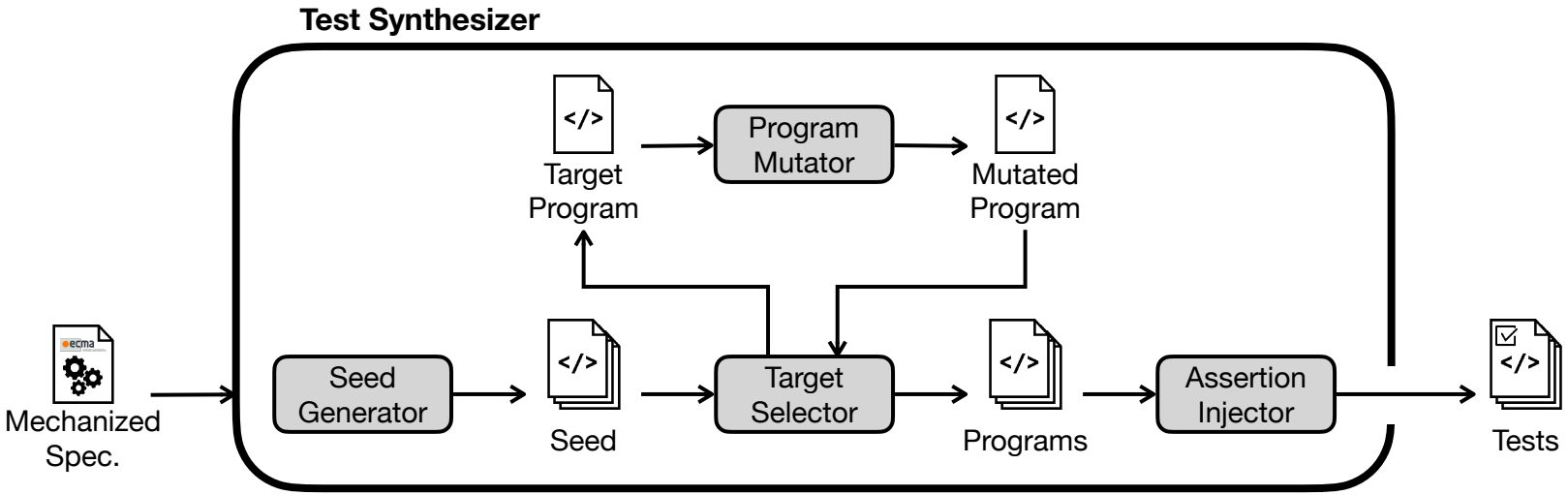

**A specification bug in ECMAScript**

**An engine bug in GraalVM**

[25] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

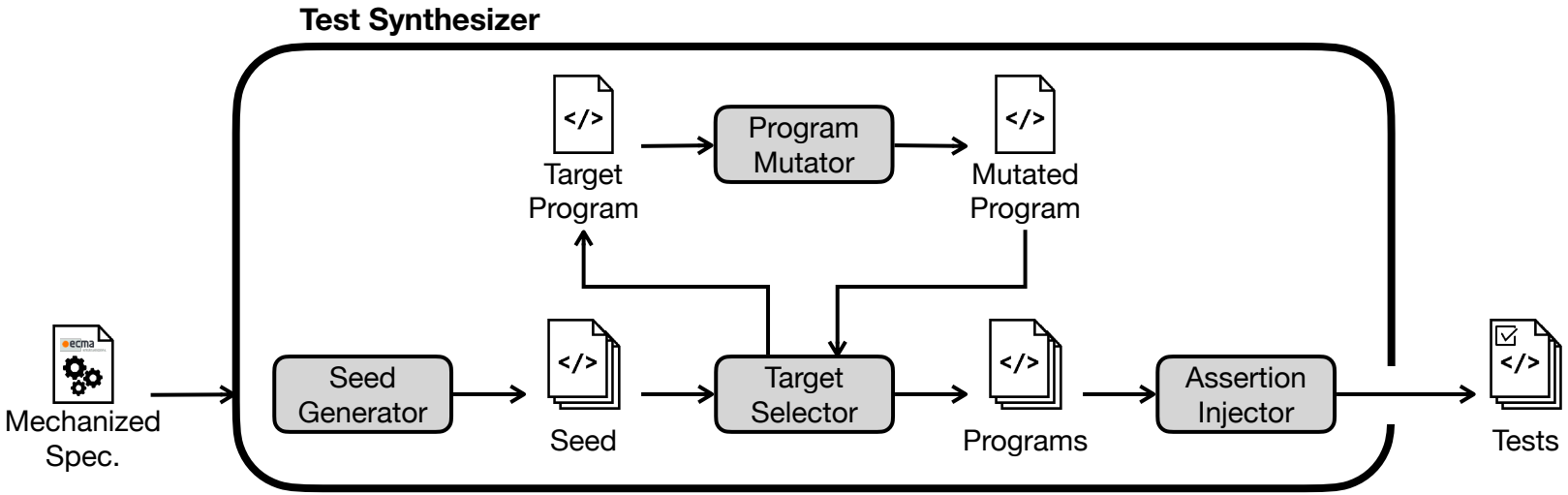
The tests are **synthesized** from the spec, and each one carries its own **assertions** – so the oracle is generated too.[26]




[26] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# N+1-version Differential Testing

The tests are **synthesized** from the spec, and each one carries its own **assertions** – so the oracle is generated too.[26]




Found **44** bugs in four engines and **27** in the specification. We met this suite's **coverage** and its **minimization** earlier in the course; today is its **oracle**.

[26] **[ICSE'21]** J. Park, S. An, D. Youn, G. Kim, S. Ryu. "*JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification.*"

# Contents

# Property-based Testing

Write a **property** instead of input-output pairs. The tool generates the inputs and **shrinks** any counterexample.[27]

**Example-based**: three inputs, three answers.

```
def abs(x):
  if x < 0:
    return -x
  return x

def test_abs():
  assert abs(0) == 0
  assert abs(1) == 1
  assert abs(-1) == 1
```

**Property-based**: any input, no answers.

```
def abs(x):
  if x < 0:
    return -x
  return x

def test_abs(x):
  assert abs(x) >= 0
  assert abs(x) == abs(-x)
  assert abs(abs(x)) == abs(x)
```

[27] **[ICFP'00]** K. Claessen, J. Hughes. "*QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs.*"

# Tool – Hypothesis for Python

**Hypothesis** (link) is the property-based testing library for Python.[28]

```
from hypothesis import given, example, assume, strategies as st
from collections import Counter

@given(xs = st.lists(st.integers()))  # strategy: composable generator
@example(xs = [])                     # pin a case you care about
def test_sort(xs):
  assume(len(xs) < 1000)              # discard unqualified inputs
  ys = sorted(xs)
  assert sorted(ys) == ys             # idempotent
  assert Counter(ys) == Counter(xs)   # same multiset
```

[28] **[JOSS'19]** D. R. MacIver, Z. Hatfield-Dodds. "*Hypothesis: A New Approach to Property-Based Testing.*"

# Tool – Hypothesis for Python

**Hypothesis** (link) is the property-based testing library for Python.[28]

```
from hypothesis import given, example, assume, strategies as st
from collections import Counter

@given(xs = st.lists(st.integers()))  # strategy: composable generator
@example(xs = [])                     # pin a case you care about
def test_sort(xs):
  assume(len(xs) < 1000)              # discard unqualified inputs
  ys = sorted(xs)
  assert sorted(ys) == ys             # idempotent
  assert Counter(ys) == Counter(xs)   # same multiset
```

- `@given` makes it an ordinary test; pytest runs it like any other.

[28] **[JOSS'19]** D. R. MacIver, Z. Hatfield-Dodds. "*Hypothesis: A New Approach to Property-Based Testing.*"

# Tool – Hypothesis for Python

**Hypothesis** (link) is the property-based testing library for Python.[28]

```python
from hypothesis import given, example, assume, strategies as st
from collections import Counter

@given(xs = st.lists(st.integers()))  # strategy: composable generator
@example(xs = [])                     # pin a case you care about
def test_sort(xs):
  assume(len(xs) < 1000)              # discard unqualified inputs
  ys = sorted(xs)
  assert sorted(ys) == ys             # idempotent
  assert Counter(ys) == Counter(xs)   # same multiset
```

- `@given` makes it an ordinary test; pytest runs it like any other.
- A failing `xs` is **shrunk**, then **saved**. The next run replays it **first**, so a fixed bug stays fixed.

[28] **[JOSS'19]** D. R. MacIver, Z. Hatfield-Dodds. "*Hypothesis: A New Approach to Property-Based Testing.*"

# Tool – Hypothesis for Python

```
from hypothesis import given, settings, Verbosity
from hypothesis import strategies as st

def abs(x):
  if x < 0:
    return -x
  return x

@given(x = st.integers())
@settings(verbosity=Verbosity.verbose)
def test_abs(x):
  assert abs(x) >= 0
  assert abs(x) == abs(-x)
  assert abs(abs(x)) == abs(x)

test_abs()
```

# Shrinking – the Part That Makes It Usable

A round-trip property for a run-length encoder:

```
@given(s = st.text(alphabet="ab"))
def test_roundtrip(s):
  assert decode(encode(s)) == s
```

# Shrinking – the Part That Makes It Usable

A round-trip property for a run-length encoder:

```
@given(s = st.text(alphabet="ab"))
def test_roundtrip(s):
  assert decode(encode(s)) == s
```

First failure: 61 characters of noise. Hypothesis re-runs the property on **smaller** inputs that still fail:

```
61 'bbababbbbbbbbbbaaabbaabbbbaaababaaaa...'
26 'babaabbbbbbbbbbbababbaaaaa'
 : :
 : :
10 'aaaaaaaaaa'
```

# Where Do Properties Come From?

The same bottleneck as metamorphic relations – and, in practice, the same handful of shapes.

| **Shape** | **Property** |
|---|---|
| Round trip | `decode(encode(x)) == x` |
| Invariant | `Counter(sorted(xs)) == Counter(xs)` |
| Idempotence | `abs(abs(x)) == abs(x)` |
| Algebraic law | `add(a, b) == add(b, a)` |
| Reference implementation | `mine(x) == theirs(x)` |

- The last row is **differential testing**. A property on **two runs**, `abs(x) == abs(-x)`, is a **metamorphic relation** with $f(x) = -x$. The three sections of today are one idea.

[29] **[ICSE'24]** H. Goldstein, J. W. Cutler, D. Dickstein, B. C. Pierce, A. Head. "*Property-Based Testing in Practice.*"

# Where Do Properties Come From?

The same bottleneck as metamorphic relations – and, in practice, the same handful of shapes.

| **Shape** | **Property** |
|---|---|
| Round trip | `decode(encode(x)) == x` |
| Invariant | `Counter(sorted(xs)) == Counter(xs)` |
| Idempotence | `abs(abs(x)) == abs(x)` |
| Algebraic law | `add(a, b) == add(b, a)` |
| Reference implementation | `mine(x) == theirs(x)` |

- The last row is **differential testing**. A property on **two runs**, `abs(x) == abs(-x)`, is a **metamorphic relation** with $f(x) = -x$. The three sections of today are one idea.
- In use at **Amazon**, **Volvo**, and **Stripe**, by an interview study of practitioners.[29]

[29] **[ICSE'24]** H. Goldstein, J. W. Cutler, D. Dickstein, B. C. Pierce, A. Head. "*Property-Based Testing in Practice.*"

# When the Oracle Disagrees

- Today we built oracles out of **relations** ($p(x)$ against $p(f(x))$), out of **other implementations**, and out of **properties** – none of them needed the expected output.

# When the Oracle Disagrees

- Today we built oracles out of **relations** ($p(x)$ against $p(f(x))$), out of **other implementations**, and out of **properties** – none of them needed the expected output.

- Every one of them reports the **same** thing: *these two results should have agreed, and they did not.* Which side is wrong, and which line of which program made it wrong, is **not** in the report.

# When the Oracle Disagrees

- Today we built oracles out of **relations** ($p(x)$ against $p(f(x))$), out of **other implementations**, and out of **properties** – none of them needed the expected output.

- Every one of them reports the **same** thing: *these two results should have agreed, and they did not.* Which side is wrong, and which line of which program made it wrong, is **not** in the report.

- Shrinking was the first answer: make the failing input **small** enough to read.

# When the Oracle Disagrees

- Today we built oracles out of **relations** ($p(x)$ against $p(f(x))$), out of **other implementations**, and out of **properties** – none of them needed the expected output.

- Every one of them reports the **same** thing: *these two results should have agreed, and they did not.* Which side is wrong, and which line of which program made it wrong, is **not** in the report.

- Shrinking was the first answer: make the failing input **small** enough to read.

- **Next**: given a failing test, find the **faulty line**.

# Summary

PLRG

1. Non-Testable Programs
    - Types of Non-Testable Programs
2. Metamorphic Testing
    - Examples: SMT Solvers and Datalog Engines
    - Examples: Database Engines
    - Examples: Web Browser Debugger
    - Examples: Compilers
    - Examples: Object Detection
    - Learning Metamorphic Relationships
3. Differential Testing
    - Examples: Deep Neural Networks
    - Examples: Nezha
    - Examples: Binary Lifters
    - Examples: JIT Compilers
    - N+1-version Differential Testing
4. Property-based Testing

# Next Lecture

- Fault Localization

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr