



UnitCon: Synthesizing Targeted Unit Tests for Java Runtime Exceptions

Sujin Jang, Yeonhee Ryou, Heewon Lee, Kihong Heo

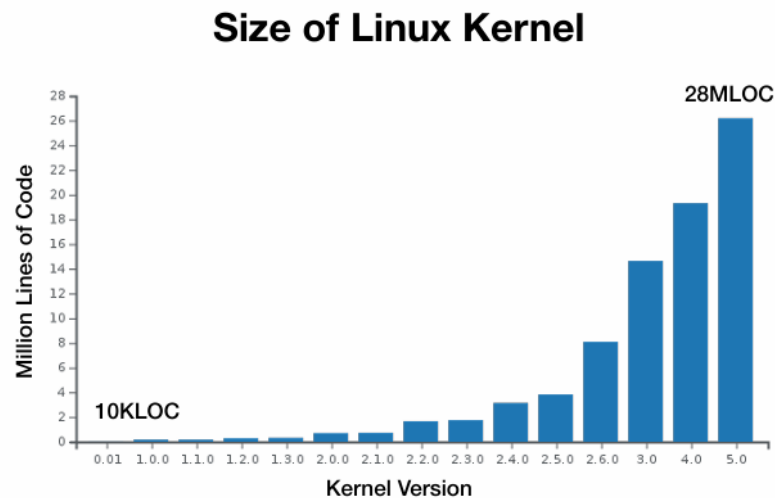
KAIST



Programming
Systems Laboratory

Motivation

- Software is getting larger, with frequent changes.
- Increasing need for **targeted unit testing**.
 - Aims to reveal a bug at a given specific location.
 - e.g., continuous integration, static analysis alarm inspection.
- Conventional test case generation techniques are not effective for targeted testing.
 - Aim to generate regression tests by maximizing code coverage.



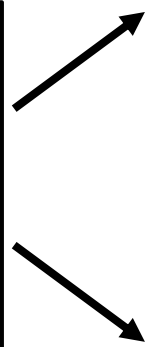
Program	Commits (2024.3)	Active Developers (2024.3)
Linux Kernel	864	59
LLVM	3,525	56
V8	800	31
OpenSSL	80	17
Elastic Search	514	49

Problem

- **Exponential growth of partial test cases** makes simple synthesis ineffective.
 - Only **one** step can generate thousands of partial test cases.
 - Other tools (Randoop, EvoSuite) found the bug in **only 0 to 1** out of 10 runs.

* JSqlParser (13K LOC)

```
public void test() {  
    Adapter recv = new Adapter();  
    recv.M(ID);  
    Select select = new Select();  
    recv.visit(select);  
}
```



```
public void test() {  
    Adapter recv = new Adapter();  
    recv.accept(ID);  
    Select select = new Select();  
    recv.visit(select);  
}
```

...

```
public void test() { // Goal  
    Adapter recv = new Adapter();  
    recv.setVisitor(ID);  
    Select select = new Select();  
    recv.visit(select);  
}
```

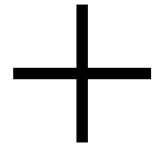


2000 !!

Our Solution



**Program with
a specific location**



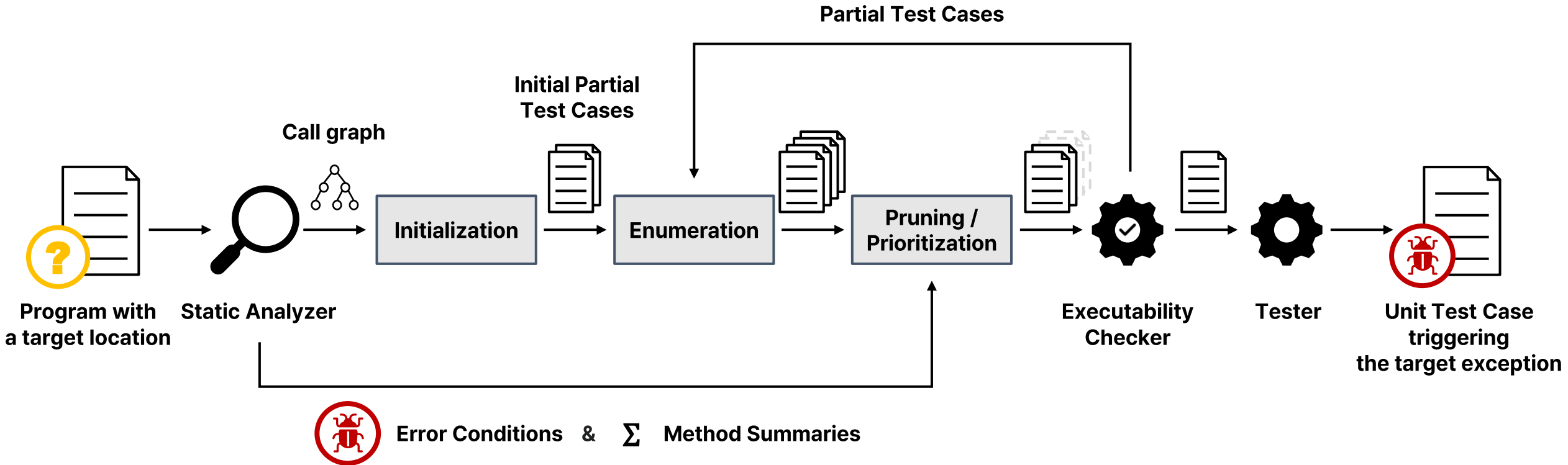
**Program Synthesis
Guided by Static Analyzer**



```
public void test() {  
    Adapter recv = new Adapter();  
    Visitor visitor = new Visitor();  
    recv.setVisitor(visitor);  
    Select select = new Select();  
    recv.visit(select); // NPE  
}
```

Targeted Unit Test

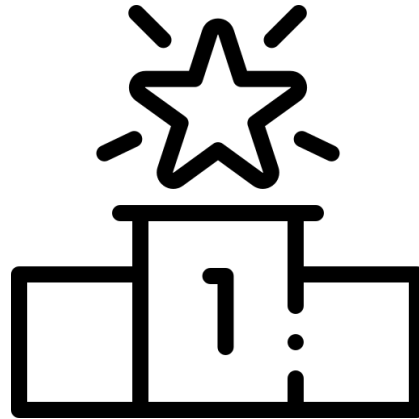
UnitCon System



Contributions



First targeted
unit test synthesizer

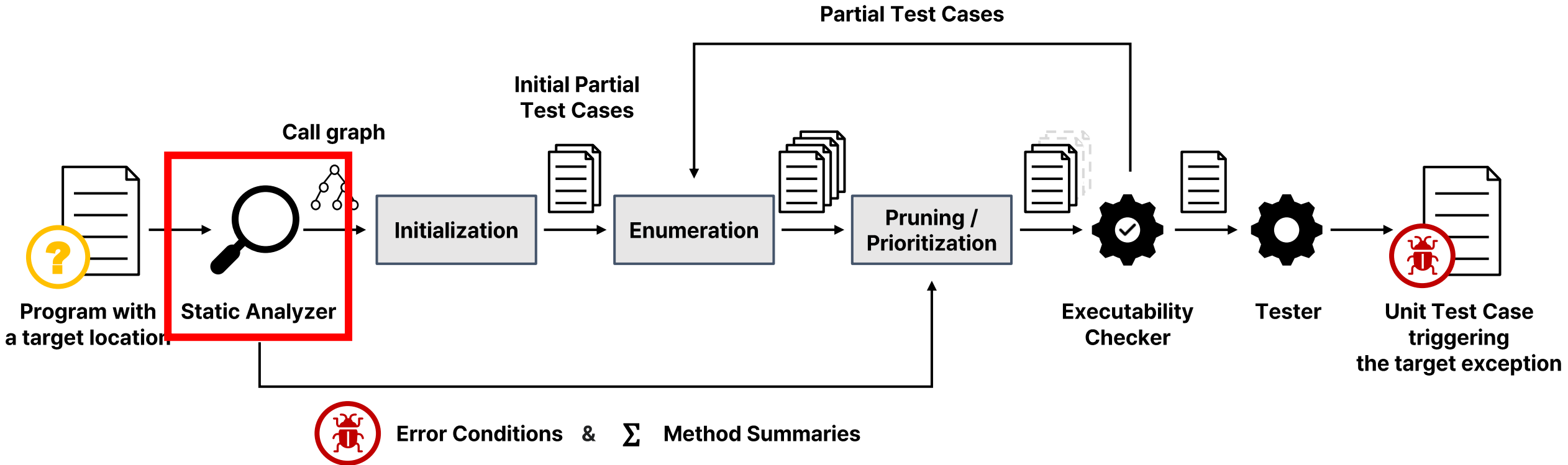


Up to **3.6 x** better performance
compared to baselines



Found **21** new bugs
in open-source programs

UnitCon System



Example

```
1 public class Adapter {
2     private Visitor visitor;
3
4     public void setVisitor(Visitor visitor) {
5         this.visitor = visitor;
6     }
7
8     public void visit(Select select) {
9         if (visitor != null) {
10            ItemsList itemsList = select.getItemsList();
11            for (Item item: itemsList) { // Target location
12                ...
13            }
14        }
15
16        public class Select {
17            private List <Item> itemsList;
18
19            public List <Item> getItemsList() { return itemsList; }
20        }
21
22        public class Merge {
23            private Select usingSelect;
24
25            public Select getUsingSelect() { return usingSelect; }
26        }
```



Error Method

Adapter.visit(Select)

Example

```
1 public class Adapter {
2     private Visitor visitor;
3
4     public void setVisitor(Visitor visitor) {
5         this.visitor = visitor;
6     }
7
8     public void visit(Select select) {
9         if (visitor != null) {
10            ItemsList itemsList = select.getItemsList();
11            for (Item item: itemsList) { // Target location
12                ...
13            }
14        }
15    }
16    public class Select {
17        private List <Item> itemsList;
18
19        public List <Item> getItemsList() { return itemsList; }
20    }
21    field itemsList of select == null
22    public class UsingSelect {
23        private Select usingSelect;
24
25        public Select getUsingSelect() { return usingSelect; }
26    }
```

Error Method

Adapter.visit(Select)

Error Conditions

Object	Condition
	select.itemsList == null

Example

```
1 public class Adapter {
2     private Visitor visitor;
3
4     public void setVisitor(Visitor visitor) {
5         this.visitor = visitor;
6     }
7
8     public void visit(Select select) {
9         if (visitor != null) {
10            ItemsList itemsList = select.getItemsList();
11            for (Item item: itemsList) { // Target location
12                ...
13            }
14        }
15
16        public class Select {
17            private List <Item> itemsList;
18
19            public List <Item> getItemsList() { return itemsList; }
20        }
21
22        public class Merge {
23            private Select usingSelect;
24
25            public Select getUsingSelect() { return usingSelect; }
26        }
```

Error Method

```
Adapter.visit(Select)
```

Error Conditions

Object	Condition
select.itemsList	== null
this.visitor	!= null
select	!= null

Example

```
1 public class Adapter {
2     private Visitor visitor;
3
4     public void setVisitor(Visitor visitor) {
5         this.visitor = visitor;
6     }
7
8     public void visit(Select select) {
9         if (visitor != null) {
10            ItemsList itemsList = select.getItemsList();
11            for (Item item: itemsList) { // Target location
12                ...
13            }
14        }
15
16        public class Select {
17            private List <Item> itemsList;
18
19            public List <Item> getItemsList() { return itemsList; }
20        }
21
22        public class Merge {
23            private Select usingSelect;
24
25            public Select getUsingSelect() { return usingSelect; }
26        }
```

Error Method

```
Adapter.visit(Select)
```

Error Conditions

Object	Condition
select.itemsList	== null
this.visitor	!= null
select	!= null

Method Summaries

Method	Memory
Merge	{ usingSelect ↦ null }
getUsingSelect	{ ret ↦ usingSelect }
...	...

Example

```
1 public class Adapter {
2     private Visitor visitor;
3
4     public void setVisitor(Visitor visitor) {
5         this.visitor = visitor;
6     }
7
8     public void visit(Select select) {
9         if (visitor != null) {
10             ItemsList itemsList = select.getItemsList();
11             for (Item item: itemsList) { // Target location
12                 ...
13             }
14         }
15
16     public class Select {
17         private List <Item> itemsList;
18
19         public List <Item> getItemsList() { return itemsList; }
20     }
21
22     public class Merge {
23         private Select usingSelect;
24
25         public Select getUsingSelect() { return usingSelect; }
26     }
```

Error Method

```
Adapter.visit(Select)
```

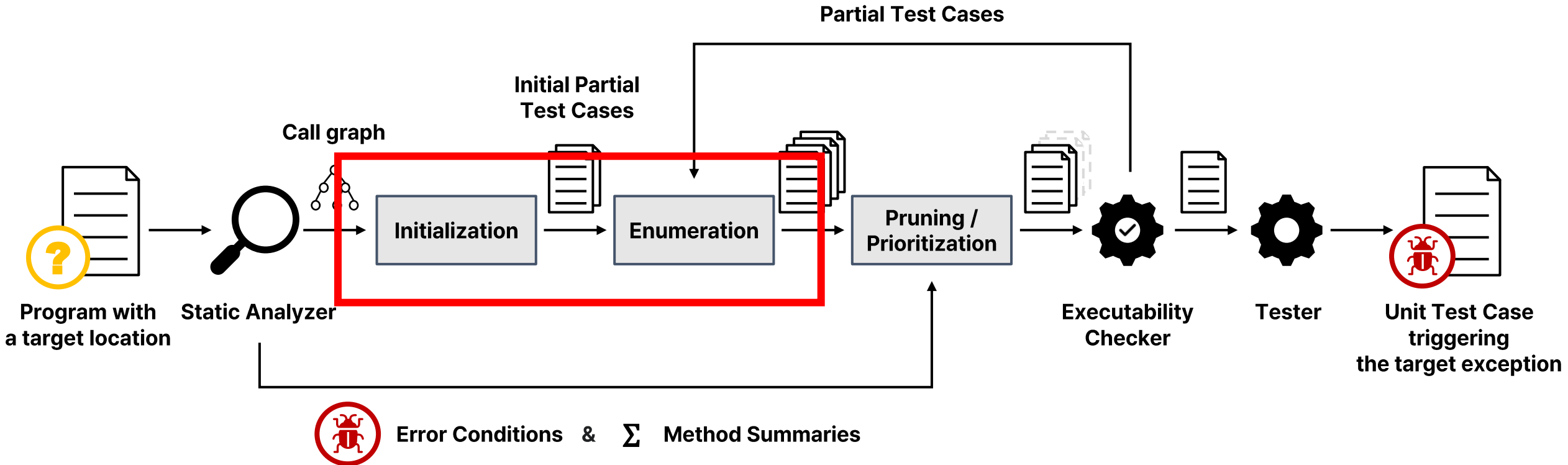
Error Conditions

Object	Condition
select.itemsList	== null
this.visitor	!= null
select	!= null

Method Summaries

Method	Memory
Merge	{ usingSelect ↦ null }
getUsingSelect	{ ret ↦ usingSelect }
...	...

UnitCon System



Top-down enumerative search

- Expand test cases top-down based on defined rules.

Rule form: **Before** → **After**

Error Method

```
Adapter.visit(Select)
```



```
ID.visit(ID);
```

$ID.f(ID) \rightarrow x = ID.M(ID); x.f(ID)$

$ID.f(ID) \rightarrow x = EXP; x.f(ID)$

```
Adapter recv = ID.M(ID);  
recv.visit(ID);
```

```
Adapter recv = EXP;  
recv.visit(ID);
```

$x0.f(ID) \rightarrow x1 = ID.M(ID); x0.f(x1)$

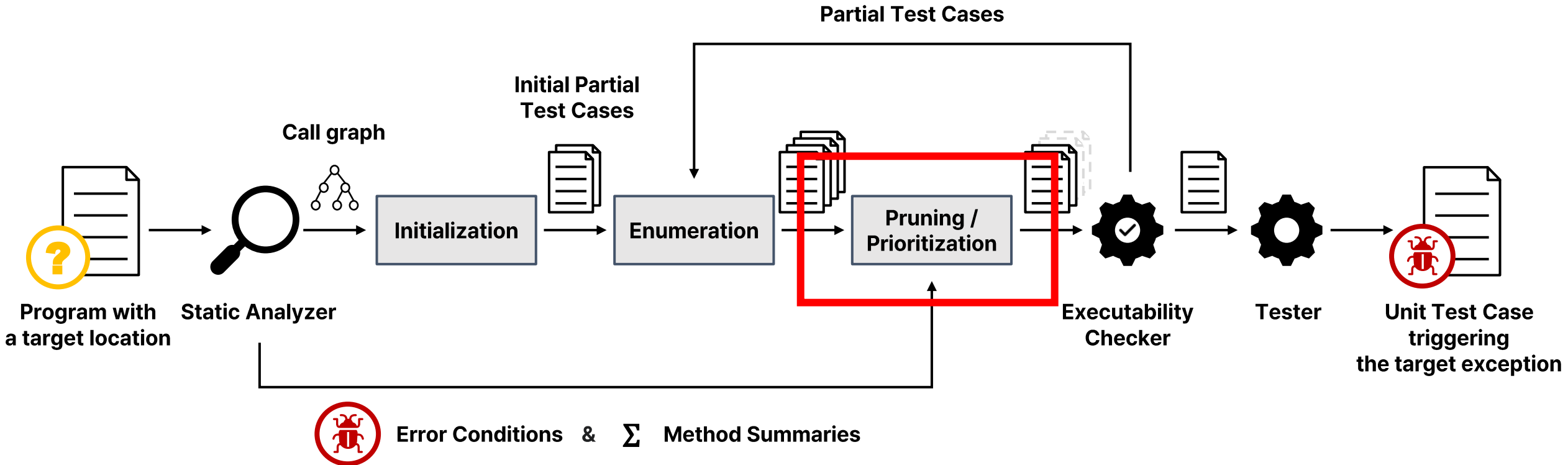
$x0.f(ID) \rightarrow x1 = EXP; x0.f(x1)$

```
Select select = ID.M(ID);  
Adapter recv = ID.M(ID);  
recv.visit(select);
```

```
Select select = EXP;  
Adapter recv = ID.M(ID);  
recv.visit(select);
```

...

UnitCon System



Pruning

- Discard partial test cases with identical semantics.

Current partial test case

```
Adapter recv = ID.M(ID);  
  
Select select =   
recv.visit(select);
```

Method Summaries

Method	Memory
Merge	{ usingSelect ↦ null }
getUsingSelect	{ ret ↦ usingSelect }
...	...

```
Adapter recv = ID.M(ID);  
Select select = null;  
recv.visit(select);
```



null

```
Adapter recv = ID.M(ID);  
Merge merge = new Merge();  
Select select = merge.getUsingSelect();  
recv.visit(select);
```



null

```
Adapter recv = ID.M(ID);  
Select select = new Select();  
recv.visit(select);
```



!= null

...

...

Prioritization

- Prioritize test cases that are more likely to satisfy the error conditions.

```
Adapter recv = ID.M(ID);  
Select select = null;  
recv.visit(select);
```



Error Conditions

Object	Condition
select.itemsList	== null
this.visitor	!= null
select	!= null

```
Adapter recv = ID.M(ID);  
Select select = new Select();  
recv.visit(select);
```



...

Example

```
1 public class Adapter {
2     private Visitor visitor;
3
4     public void setVisitor(Visitor visitor) {
5         this.visitor = visitor;
6     }
7
8     public void visit(Select select) {
9         if (visitor != null) {
10            ItemsList itemsList = select.getItemsList();
11            for (Item item: itemsList) { // Target location
12                ...
13            }
14        }
15
16    public class Select {
17        private List <Item> itemsList;
18
19        public List <Item> getItemsList() { return itemsList; }
20    }
21
22    public class Merge {
23        private Select usingSelect;
24
25        public Select getUsingSelect() { return usingSelect; }
26    }
```



```
public void test() {
    Adapter recv = new Adapter();
    Visitor visitor = new Visitor();
    recv.setVisitor(visitor);
    Select select = new Select();
    recv.visit(select);
}
```

Exception reproduced in **39 seconds**.

Evaluation: Known Bug Reproduction

Benchmarks

- 198 Java programs (1 target exception per program).
 - 40K – 100K LOC
 - Defects4J (ISSTA'14), Bears (SANER'19), GENESIS (ESEC/FSE'17), NPEX (ICSE'22), VFIX (ICSE'19)

Baselines

- EvoSuite (ESEC/FSE'11), EvoFuzz (SBFT'24), NPETest (ASE'24), UTBot (SBFT'23), Randoop (OOPSLA'07)
- Repeat the 10 runs for tools with randomness.

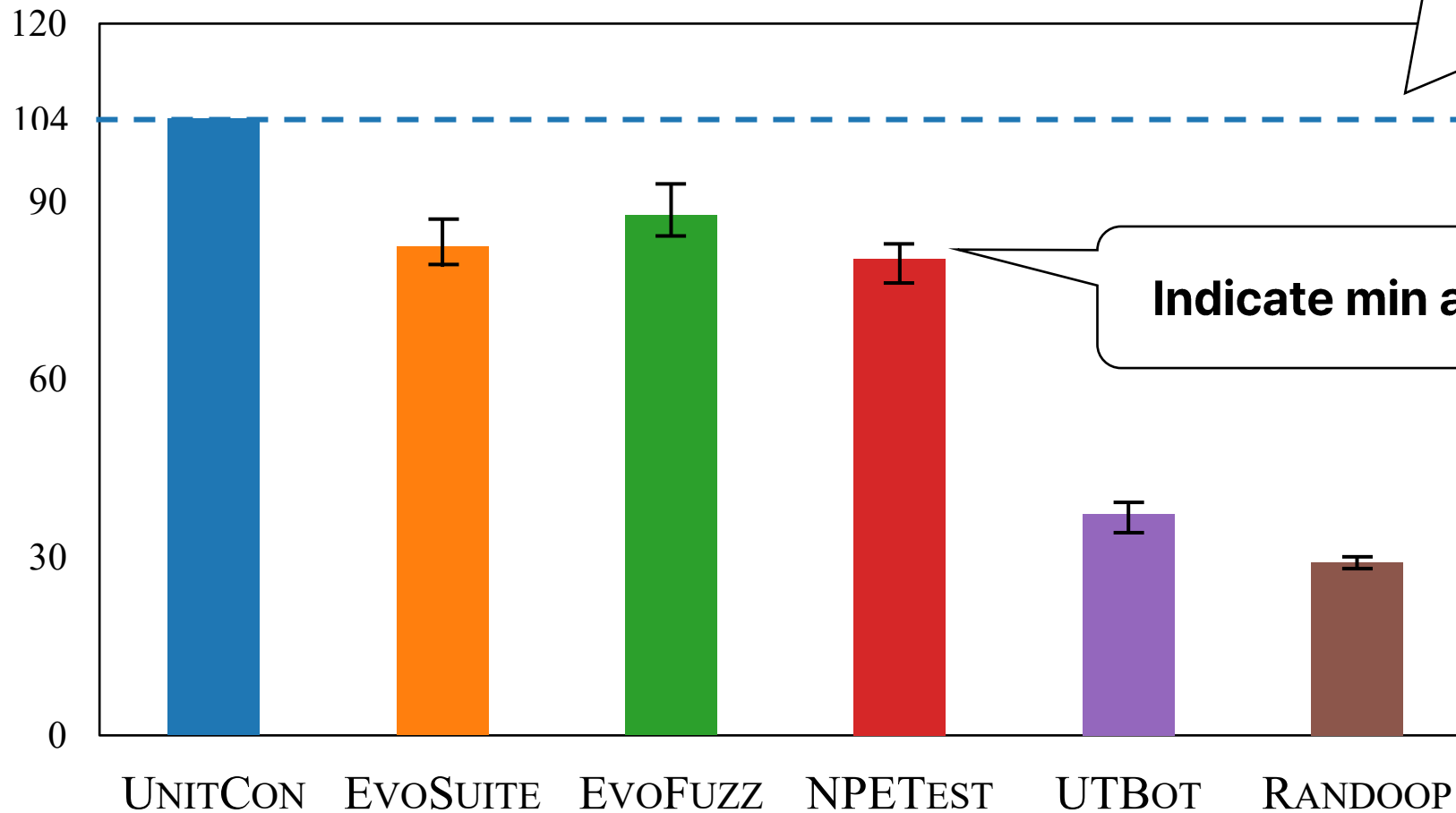
Task

- 10 minutes time limit per 1 target exception.

Comparison to Baselines

- **1.2 – 3.6 x** more success than baselines.

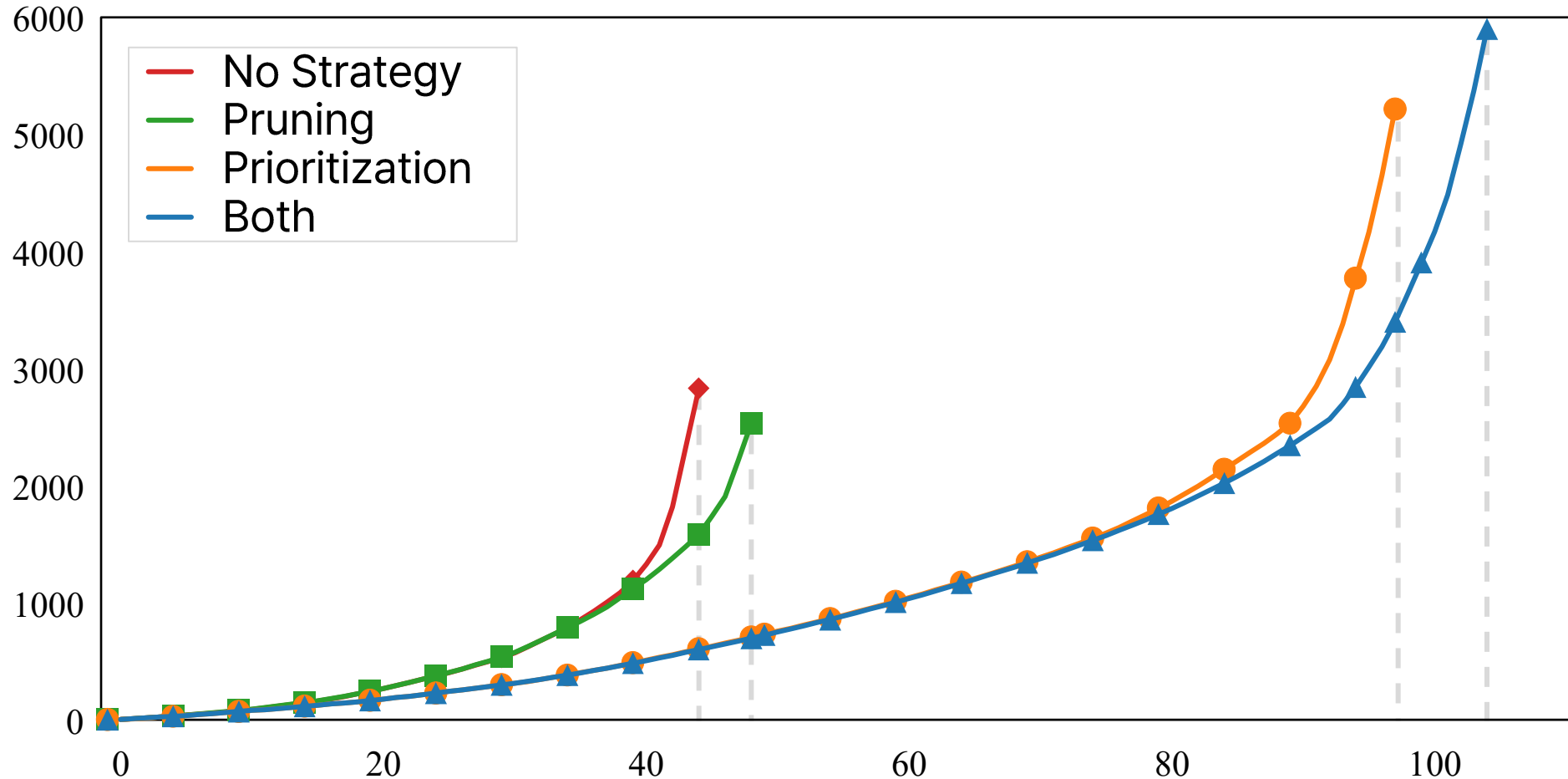
Deterministically outperform baselines.



Indicate min and max

Impact of Each Strategy

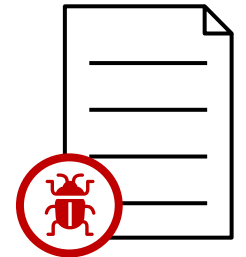
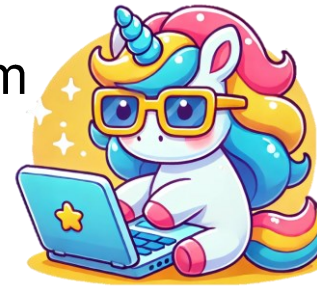
- Using **both strategies** yields the best performance.



Find New Bugs



5m / alarm



51 Java programs
(e.g., Apache, Kubernetes)

Static Analysis
(Facebook Infer)

4,290 NPE alarms

UnitCon

21 new bugs found

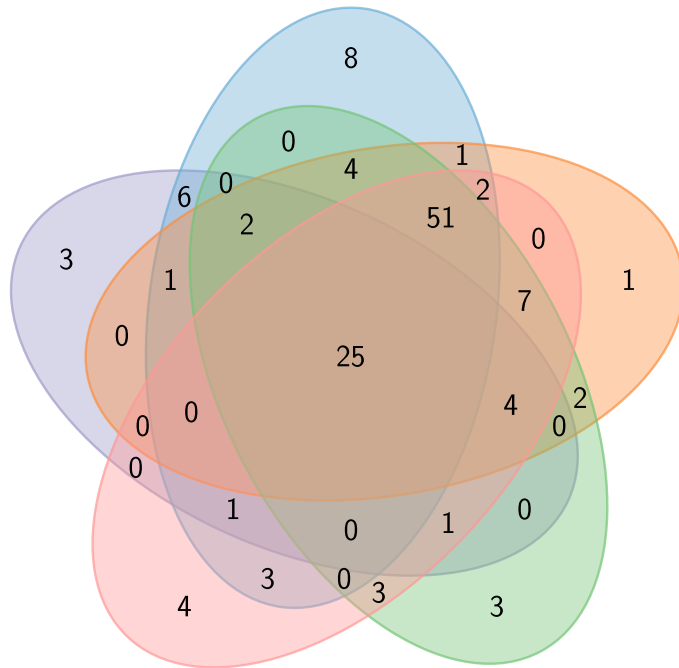
Summary



Webpage & Artifact

- **UnitCon: Synthesizing targeted unit tests for Java runtime exceptions.**
- **Key Idea: Guided Search via Abstract Semantics**
 - Discard partial test cases with identical semantics.
 - Prioritize test cases that are more likely to satisfy the error conditions.
- **Performance**
 - **Deterministically** reproduces up to **3.6 X** more target errors than baselines.
 - Found **21** new bugs.

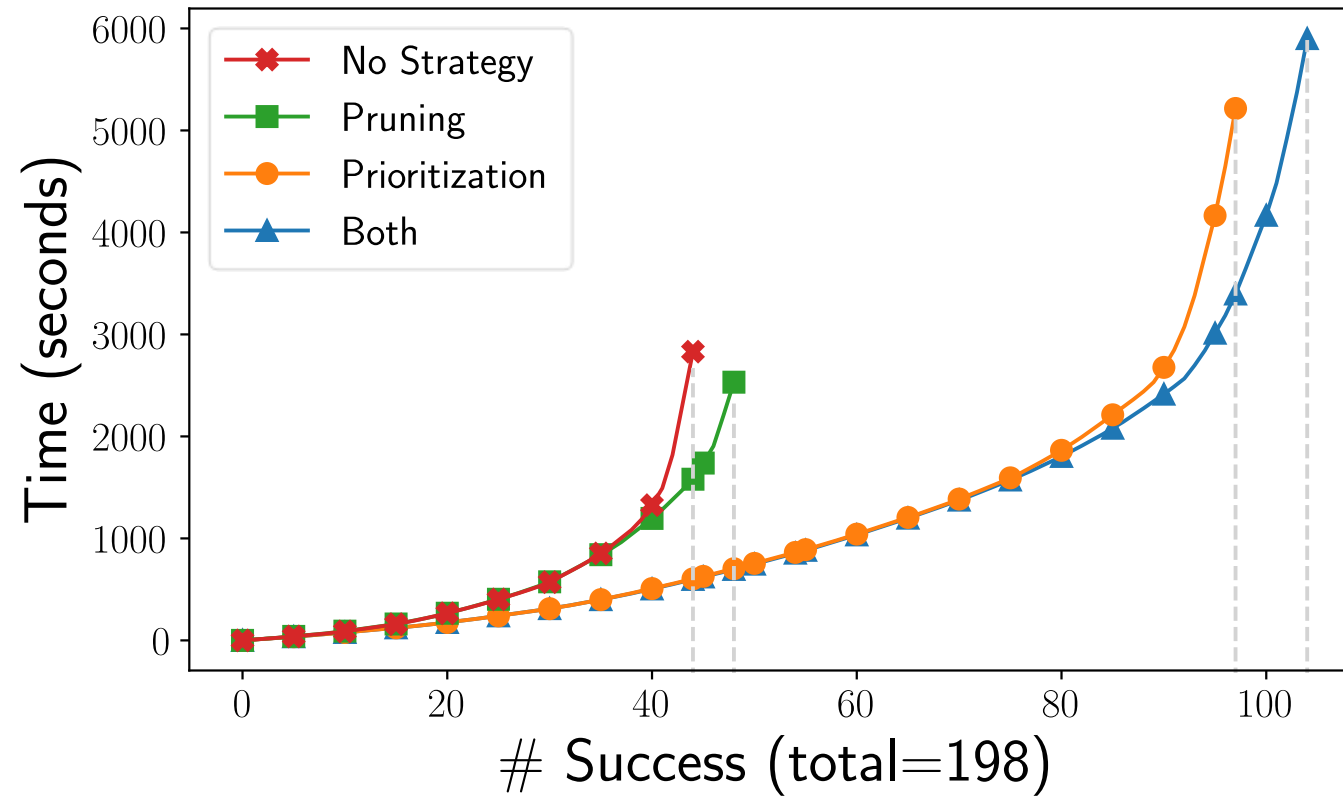
Relationship of Successful Cases Between Tools



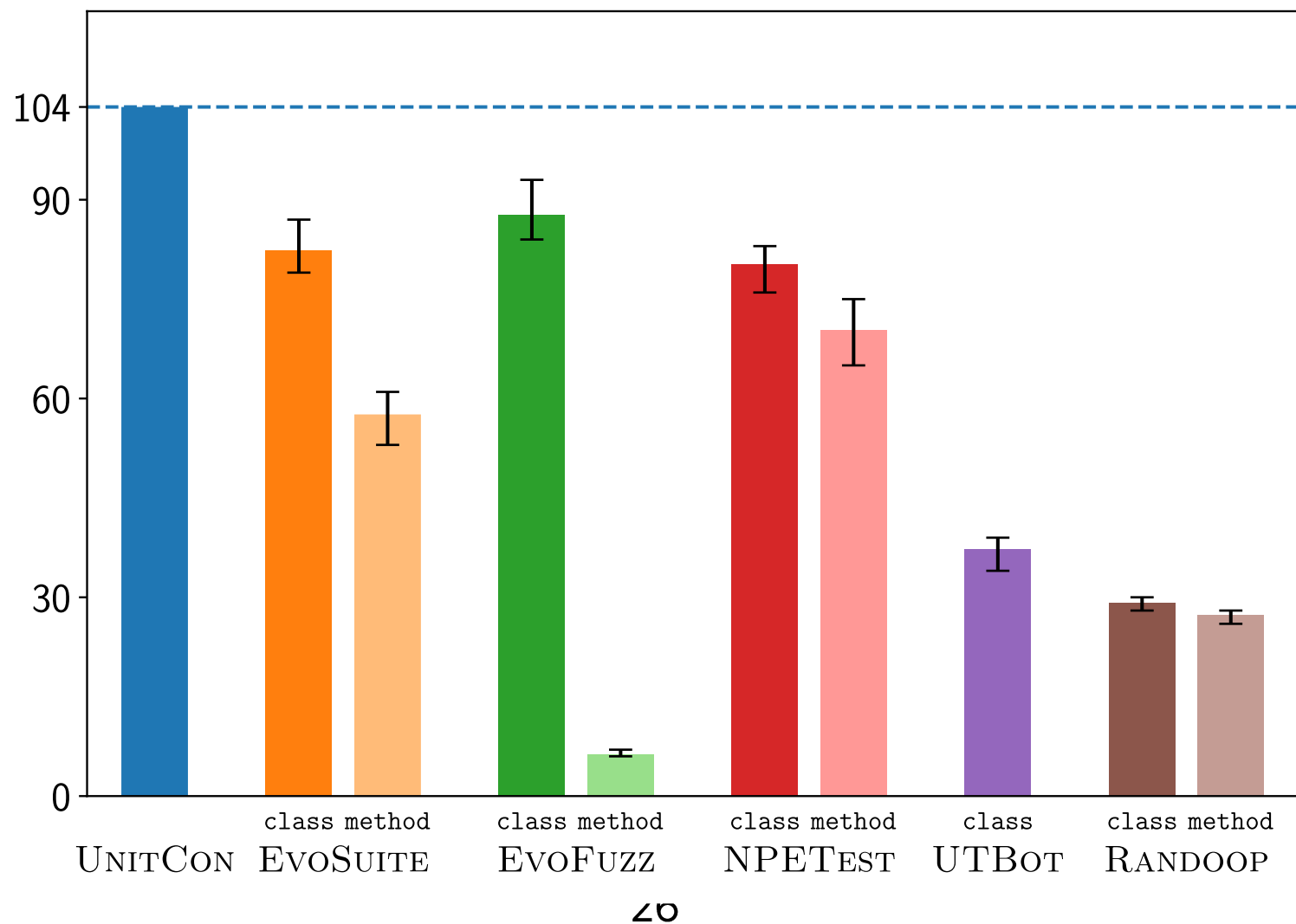
- Each tool has unique strengths.
- UnitCon uniquely reproduced the most errors.

Impact of UnitCon's Strategies

- Pruning conservatively reduces the search space.
- Make time-consuming explorations feasible.



Options of baselines



Characteristics of Infer



- Under-approximate.
- Path-sensitive analyzer.
 - handle up to K (e.g., 5) distinct paths per method.
- Indirect calls are handled imprecisely.
 - e.g., overriding, abstract class

Domain-Specific Language

$Stmt$	\rightarrow	$ID := Exp$	assignment
		$ID := ID.M(ID)$	non-void method call
		$ID.M(ID)$	void method call
		$Stmt; Stmt$	sequence
		$Skip$	no-op
M	\rightarrow	f	methods
Exp	\rightarrow	$n \mid null$	primitive values
		g	global constants
ID	\rightarrow	x	variables
		C	class names

- Define a DSL to support Java at the source-code level in a simple yet powerful way.

Limitation of UnitCon

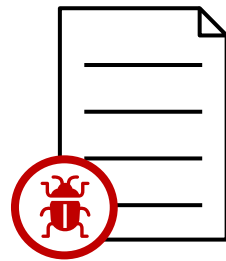
- No special limitations tied to specific types of bugs.
- However, it struggles with cases that require **strings** or **numbers** not already present in the program.
 - UnitCon's goal: **deterministically** synthesizing the test cases.

Reliance on Static Analysis



9 NPE alarms

- Found the new bugs that the static analyzer was unable to find.
- Program analysis vs. Program synthesis



21 new bugs found