变异测试

南京大学 软件学院 iSE实验室

目录

[1] Papadakis M, Kintis M, Zhang J, et al. Mutation testing advances: an analysis and survey[M]//Advances in Computers. Elsevier, 2019, 112: 275-378.

01

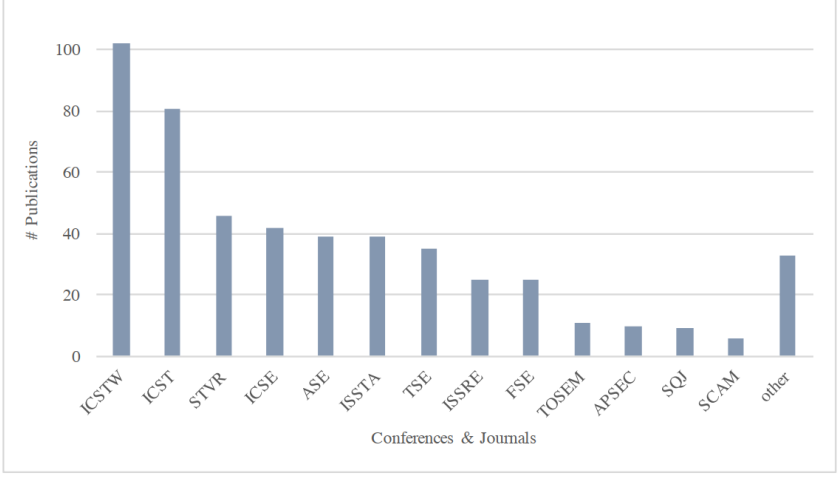
变异测试背景

变异测试的产生

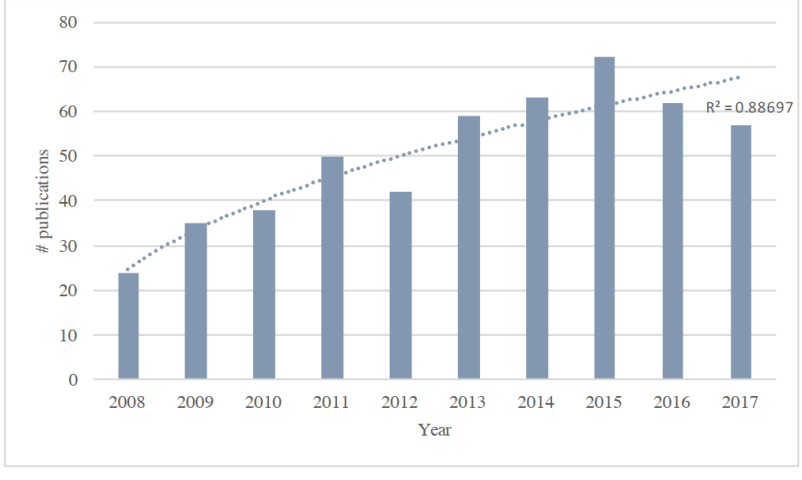
- 两个测试人员关心的问题：
  - 如何编写能够暴露缺陷的测试用例？ → 如何引导测试
  - 如何评估测试套件的质量（提升测试可信度）？ → 如何评估测试
- 变异测试的产生
  - **模拟**缺陷，**量化**缺陷检测能力，扮演测试有效性的指示器
  - 模拟：**变异**产生错误版本，模拟探测Bug的过程
  - 量化：**变异得分**（变异杀死率）

研究现状

- 变异测试
  - 从缺陷的角度评估测试用例/套件（Fault-based Testing）
  - 在学术界和工业界都得到了广泛关注



按出版物分布



按时间分布

变异分析与变异测试

- **变异分析**（Mutation Analysis）→ Research
  - 自动化“变异”源程序以得到源程序语义变体，并对其进行优化和分析的过程 → 自动化生成**人工缺陷**
  - 语义变体：语法上、语义上与源程序都不相同 → **变异体**
  - **“变异”**：程序变换规则 → **变异算子**
- **变异测试**（Mutation Testing）→ Practice
  - 将变异体视作测试目标[1]（Test Goal）
  - 利用变异分析的结果来支持、评估、引导软件测试的过程

[1] Kaufman S J, Featherman R, Alvin J, et al. Prioritizing mutants to guide mutation testing[C]//Proceedings of the 44th International Conference on Software Engineering. 2022: 1743-1754.

变异体

- **变异体**（Mutant）
  - 基于一定的语法（Syntax）变换规则，通过对源程序进行程序变换（Program Transformation）得到的一系列变体
  - 假设
    - 源程序不包含缺陷 → 现有的测试套件没有发现缺陷
    - 变异体表达了某种缺陷 → 人工缺陷（Artificial Defect）

```
public void foo(int x, int y) {
    if (x > y)
        return x;
    else
        return y;
}
```

源程序

变异

Replace Return

```
public void foo(int x, int y) {
    if (x > y)
        return x;
    else
        return x;  // A Bug!
}
```

变异体

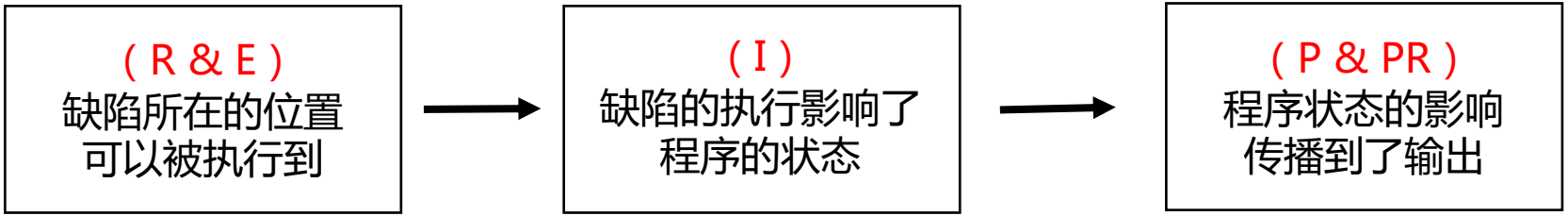
变异得分

- **变异得分**（Mutation Score）
  - 变异测试对测试套件错误检测能力的量化
  - **杀死与存活**：变异体是否导致某个测试用例运行失败；测试用例是否“检测”到某个变异体
    - 检测到：变体被杀死（Killed）
    - 未检测到：变体存活（Survived）
  - 计算公式

$$score = \frac{mut_k}{mut_s + mut_k} \times 100\%$$

变异杀死的条件

- **缺陷传播模型**：RIPR和PIE[1]
  - RIPR：**R**eachability, **I**nfection, **P**ropagation **R**evealability
  - PIE：**E**xecution，**I**nfection, **P**ropagation



[1] Voas J M. PIE: A dynamic failure-based technique[J]. IEEE Transactions on software Engineering, 1992, 18(8): 717.

变异杀死的条件

- **杀死条件**（Mutant Killing Condition）
  - 受程序行为（Program Behavior）的定义影响
  - 程序行为
    - （一般定义）可观测的程序输出：待测程序输出到标准输出的内容，或者被测试用例添加断言（Assert）的部分 ⟶ Propagation
    - 程序的中间状态：程序执行某些操作后处在的状态（Program State）⟶ Infection

变异杀死的条件

- **变异分类**
  - 根据杀死条件的不同，变异可以分为三种
    - Weak mutation：变异导致紧随变化位置的程序状态发生了改变（R & E）
    - Firm Mutation：变异导致远离变化位置的程序状态发生了改变（I）
    - Strong Mutation：变异导致程序的输出发生了变化（P & PR）
  - Weak Mutation → Firm Mutation → Strong Mutation
    - 变异的要求变高，变异体质量提升

变异体的分类

- **变异体的分类**
  - 有效变异体
  - 夭折变异体（Stillborn Mutant）
  - 冗余变异体（Redundant Mutant）
    - 等价变异体（Equivalent Mutant）
    - 重复变异体（Duplicated Mutant）
    - 蕴含变异体（Subsumed Mutant）

等价变异体

- **等价**
  - 对于待测程序P的变体mut，如果mut和P的语法不同、语义相同，则称mut是P的等价变异体
  - 语义相同：对于给定输入，两个程序总能给出相同的输出

等价变异体

```
public void foo(int a) {
    int abs = 0;
    if (a > 0) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

语义相同

```
public void foo(int a) {
    int abs = 0;
    if (a >= 1) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

```
public void testFoo() {
    assert foo(1) == 1;
}
```

PASS

```
public void testFoo() {
    assert foo(1) == 1;
}
```

SURVIVE

重复变异体

- **重复**
  - 对于待测程序P的两个变体mut_1和mut_2，如果mut_1和mut_2语义相同，则mut_1和mut_2互为重复

重复变异体

```
public void foo(int a) {
    int abs = 0;
    if (a > 0) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

```
public void testFoo() {
    assert foo(1) == 1;
}
```

PASS

```
public void foo(int a) {
    int abs = 0;
    if (a > 1) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

mut_1

语义相同

```
public void foo(int a) {
    int abs = 0;
    if (a >= 2) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

mut_2

```
public void testFoo() {
    assert foo(1) == 1;
}
```

KILL

蕴含变异体

- **蕴含**
  - 对于两个变体mut_1和mut_2，如果∀杀死mut_1的测试用例t都可以杀死mut_2，则称mut_1蕴含mut_2

蕴含变异体

```
public void foo(int a) {
    int abs = 0;
    if (a > 0) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

```
public void foo(int a) {
    int abs = 0;
    if (a >= 3) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

mut_1

蕴含

```
public void foo(int a) {
    int abs = 0;
    if (a >= 4) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

mut_2

```
public void testFoo() {
    assert foo(1) == 1;
}
```

```
public void testSym() {
    assert foo(x) == x;
}
```

Kill $mut_1 : x \in \{1, 2\}$

Kill $mut_2 : x \in \{1, 2, 3\}$

变异算子

- **变异算子**（Mutation Operator）
  - **一系列**语法变换规则（Syntactic Transformation Rule）
  - **变异**（Mutate）的依据，反映了测试人员关注的缺陷种类
  - 基本形式
    - 对程序的源代码进行变换（Source Code Level）
    - 对程序的编译结果（中间表示）进行变换，例如：针对Java Bytecode的程序变换算子
    - 元变异（Meta-mutation）

变异算子

Names	Description	Specific mutation operator
ABS	Absolute Value Insertion	$\{(e,0), (e,\texttt{abs}(e)), (e,\texttt{-abs}(e))\}$
AOR	Arithmetic Operator Replacement	$\{((\text{a } op \text{ b}), a), ((\text{a } op \text{ b}), b), (x,y) \mid x,y \in \{\texttt{+},\texttt{-},\texttt{*},\texttt{/},\texttt{\%}\} \wedge x \neq y\}$
LCR	Logical Connector Replacement	$\{((\text{a } op \text{ b}), a), ((\text{a } op \text{ b}), b), ((\text{a } op \text{ b}), \texttt{false}), ((\text{a } op \text{ b}), \texttt{true}), (x,y) \mid x,y \in \{\texttt{\&},\texttt{\|},\texttt{\^{}},\texttt{\&\&},\texttt{\|\|}\} \wedge x \neq y\}$
ROR	Relational Operator Replacement	$\{((\text{a } op \text{ b}), \texttt{false}), ((\text{a } op \text{ b}), \texttt{true}), (x,y) \mid x,y \in \{\texttt{>},\texttt{>=},\texttt{<},\texttt{<=},\texttt{==},\texttt{!=}\} \wedge x \neq y\}$
UOI	Unary Operator Insertion	$\{(cond, \texttt{!}cond), (v, \texttt{-}v), (v, \sim v), (v, \texttt{--}v), (v, v\texttt{--}), (v, \texttt{++}v), (v, v\texttt{++})\}$

五种经典变异算子[1]

[1] Offutt A J, Lee A, Rothermel G, et al. An experimental determination of sufficient mutant operators[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 1996, 5(2): 99-118.

变异算子

Mutators	"OLD_DEFAULTS" group	"DEFAULTS" group	"STRONGER" group	"ALL" group
Conditionals Boundary	yes	yes	yes	yes
Increments	yes	yes	yes	yes
Invert Negatives	yes	yes	yes	yes
Math	yes	yes	yes	yes
Negate Conditionals	yes	yes	yes	yes
Return Values	yes			yes
Void Method Calls	yes	yes	yes	yes
Empty returns		yes	yes	yes
False Returns		yes	yes	yes
True returns		yes	yes	yes

Java变异测试工具PIT提供的算子[2]

[1] Coles H, Laurent T, Henard C, et al. Pit: a practical mutation testing tool for java[C]//Proceedings of the 25th international symposium on software testing and analysis. 2016: 449-452.
[2] PITest变异算子：https://pitest.org/quickstart/mutators/

基础假设

- 变异测试为什么有效？
  - 假设1：缺陷是简单的、可模拟的
  - 假设2：缺陷可叠加
  - 假设3：缺陷检测有效性

基础假设

- 假设1：缺陷是简单的、可模拟的
  - 变异体能够模拟测试人员关注的缺陷类型，即：最常出现的缺陷类型；
  - 能够在变异测试中暴露这些人工缺陷的测试套件一定能检测出待测程序中潜在的同类缺陷
  - **老练程序员假设**：一个老练程序员编写的错误程序与正确程序相差不大

基础假设

- 假设2：缺陷可叠加
  - 复杂变异体可以通过耦合简单变异体得到
  - 能够杀死简单变异体的测试用例可以杀死复杂变异体

基础假设

```
public void foo(int a) {
    int abs = 0;
    if (a > 0) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

```
public void foo(int a) {
    int abs = 0;
    if (a >= -1) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

```
public void foo(int a) {
    int abs = 0;
    if (a > 0) {
        abs = a;
    } else {
        abs = a;
    }
    return abs;
}
```

```
public void testFoo1() {
    assert foo(-1) == 1;
}
```

```
public void testFoo2() {
    assert foo(-2) == 2;
}
```

KILL

KILL

基础假设

```
public void foo(int a) {
    int abs = 0;
    if (a >= -1) {
        abs = a;
    } else {
        abs = -a;
    }
    return abs;
}
```

+

```
public void foo(int a) {
    int abs = 0;
    if (a > 0) {
        abs = a;
    } else {
        abs = a;
    }
    return abs;
}
```

→

```
public void foo(int a) {
    int abs = 0;
    if (a >= -1) {
        abs = a;
    } else {
        abs = a;
    }
    return abs;
}
```

复杂变异体

```
public void testFoo1() {
    assert foo(-1) == 1;
}
```

KILL

```
public void testFoo2() {
    assert foo(-2) == 2;
}
```

KILL

基础假设

- 假设2：缺陷可叠加
  - 复杂变异体可以通过耦合简单变异体得到
  - 能够杀死简单变异体的测试用例可以杀死复杂变异体
  - **结合RIPR/PIE模型思考**：该假设是否总是正确？

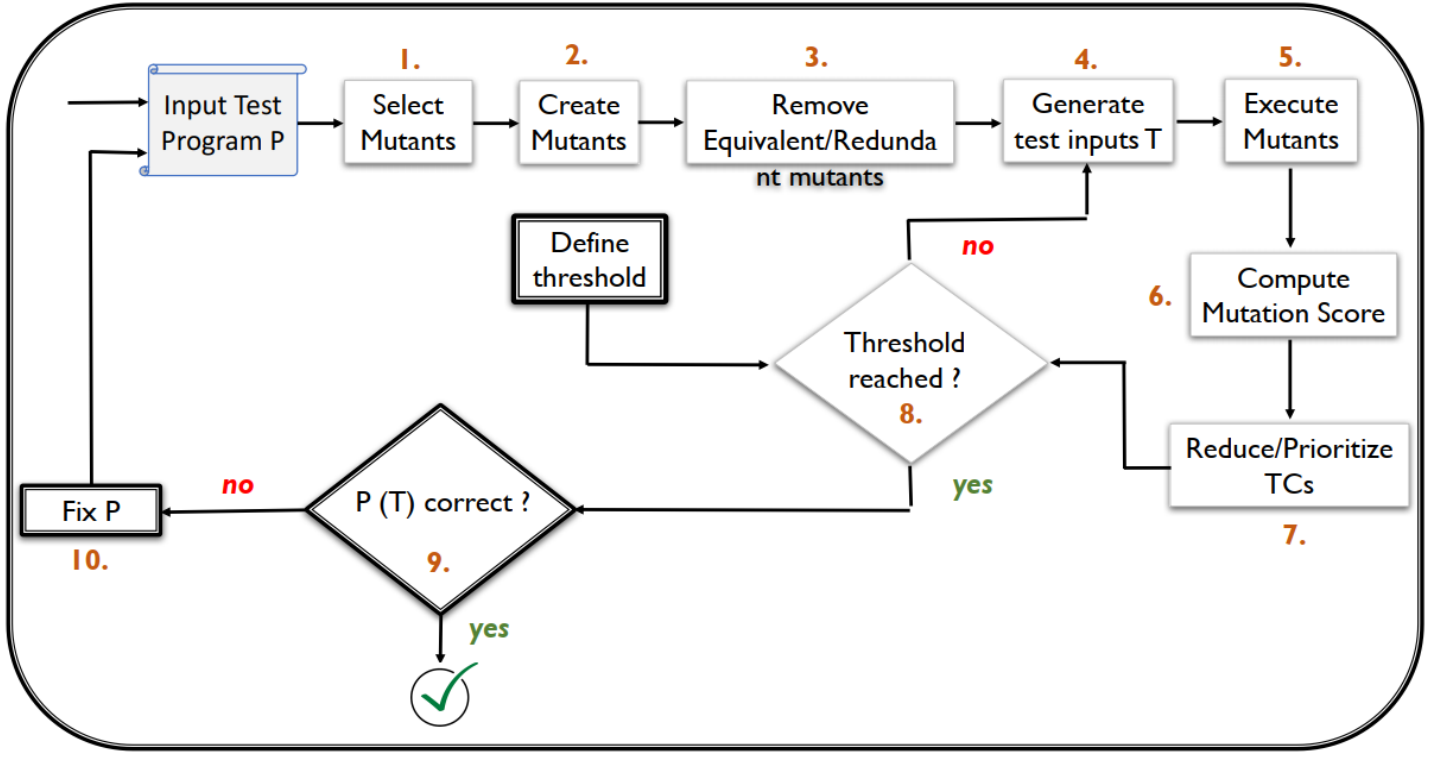
基础假设

- 假设3：缺陷检测有效性
  - **缺陷检测能力**是测试用例最重要的属性
  - 变异得分能够有效地反映测试用例/套件的错误检测能力
  - **传递性假设**：能够杀死变异体（暴露人工缺陷）的测试用例也能有效发现真实世界的缺陷

02

变异测试过程

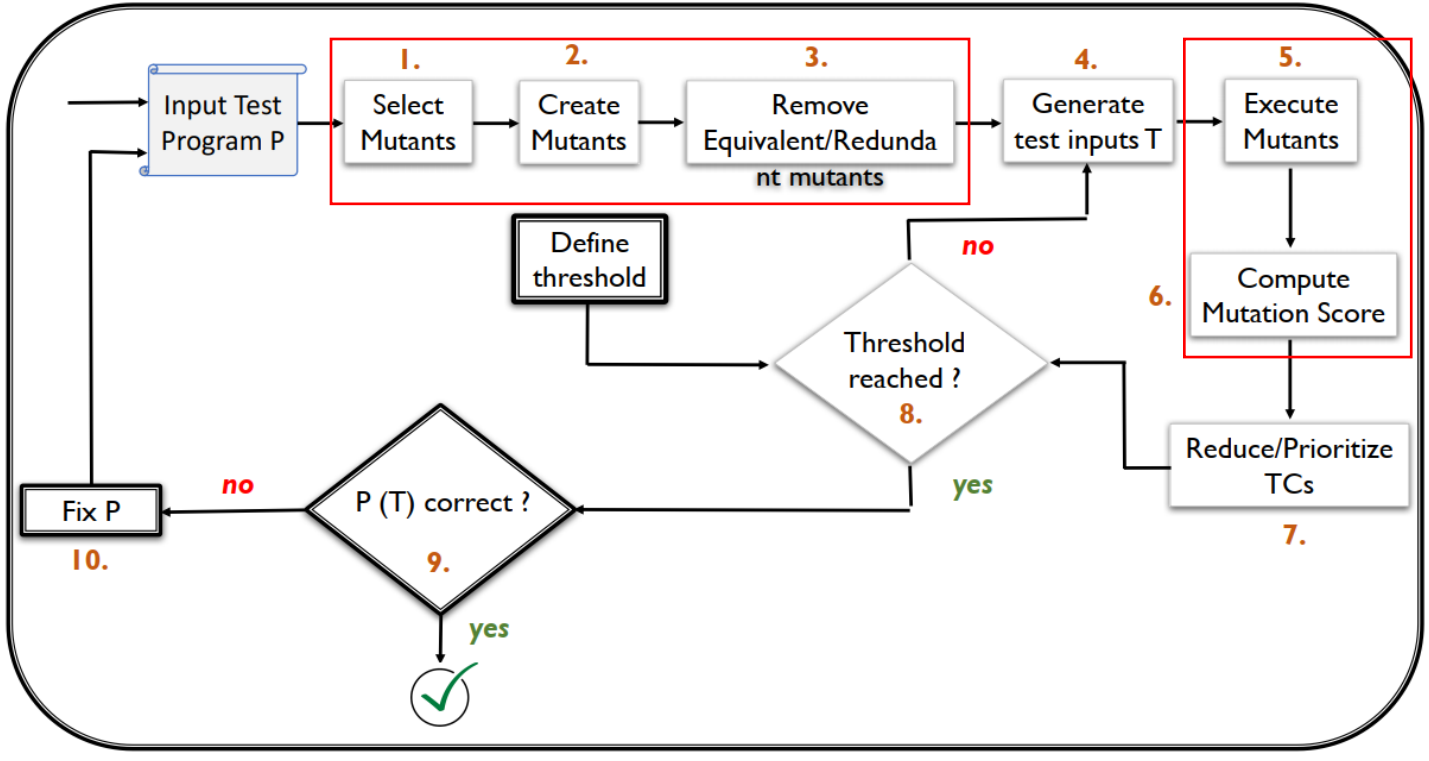
总览



变异测试流程总览[1]

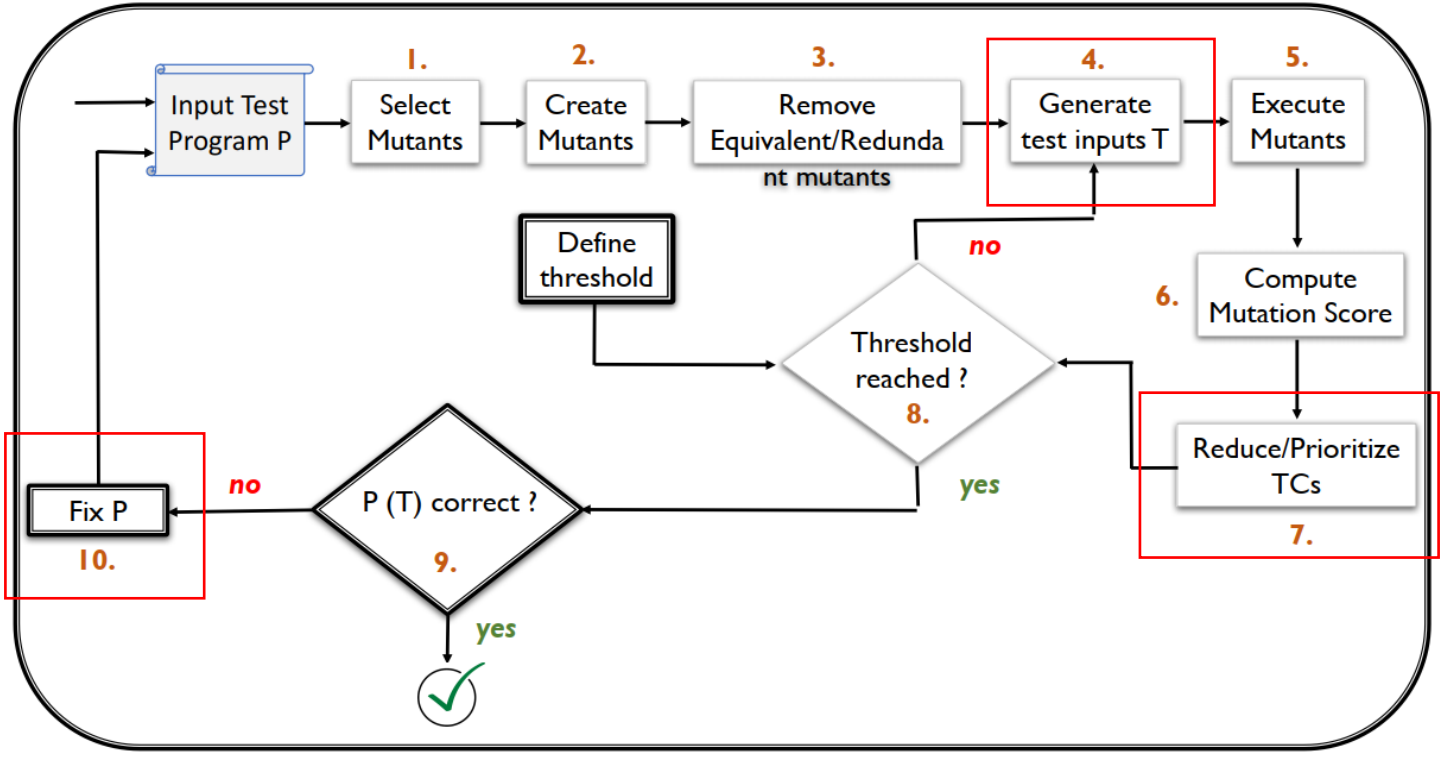
[1] Papadakis M, Kintis M, Zhang J, et al. Mutation testing advances: an analysis and survey[M]//Advances in Computers. Elsevier, 2019, 112: 275-378.

总览



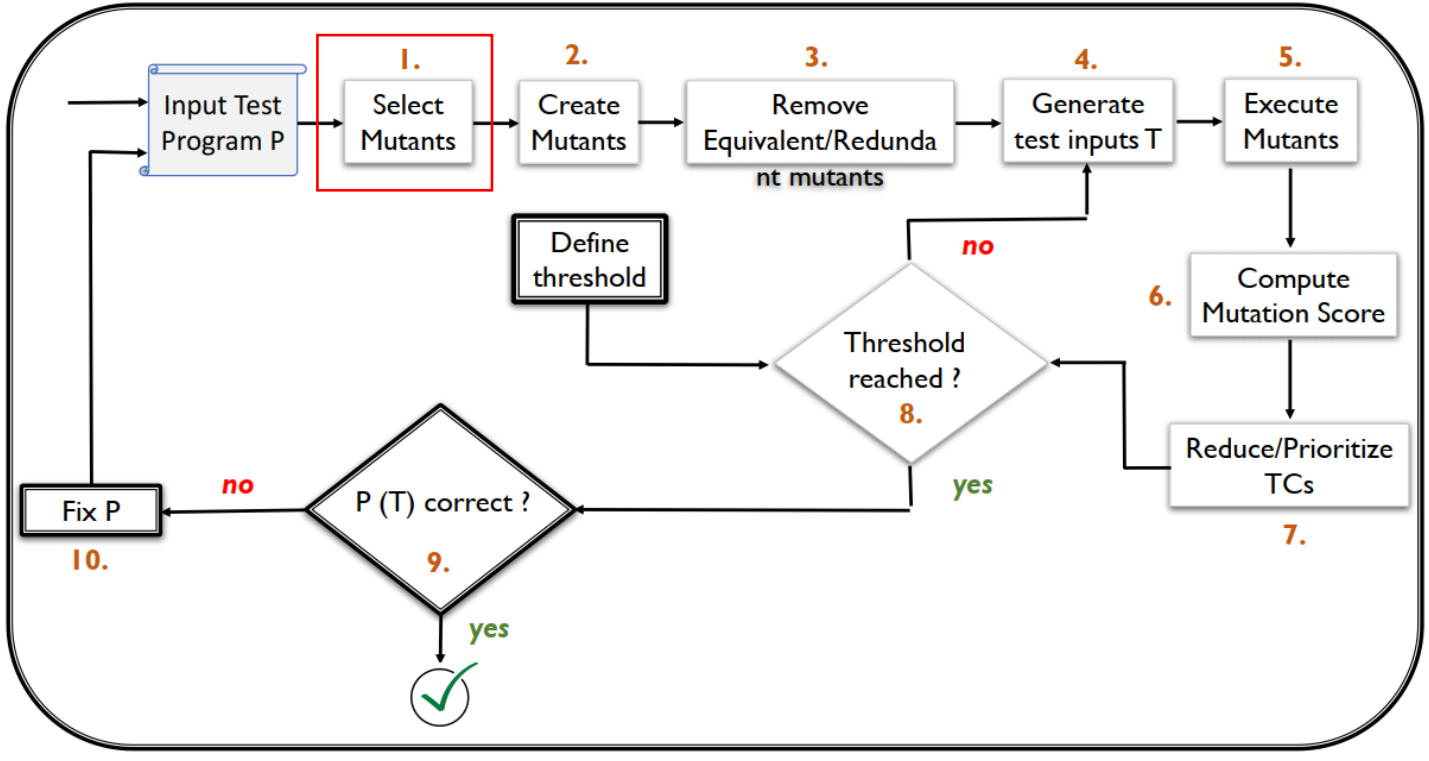
变异分析：1，2，3，5，6

总览


Input Test Program P
1.
Select Mutants
2.
Create Mutants
3.
Remove Equivalent/Redundant mutants
4.
Generate test inputs T
5.
Execute Mutants
6.
Compute Mutation Score
7.
Reduce/Prioritize TCs
8.
Threshold reached ?
no
yes
Define threshold
9.
P (T) correct ?
no
yes
10.
Fix P

变异应用：4，7，10

变异体筛选


Input Test Program P
1.
Select Mutants
2.
Create Mutants
3.
Remove Equivalent/Redundant mutants
4.
Generate test inputs T
5.
Execute Mutants
6.
Compute Mutation Score
7.
Reduce/Prioritize TCs
8.
Threshold reached ?
no
yes
Define threshold
9.
P (T) correct ?
no
yes
10.
Fix P

变异体筛选

- 变异筛选/约减（Reduction）
  - 对变异体进行筛选（Mutant Selection）：从生成的变体集合中选出一个子集
  - 通过变异算子筛选（Selective Mutation）：定义一系列变异算子，选取其中的一部分构成子集
- 研究内容
  - 变异算子的定义
  - 变异算子的选取/约减策略

变异算子定义

- 变异算子的定义
  - 算子是一组语法转换规则；算子反映了特定的缺陷类型
  - 研究的核心：如何设计**有效的**变异算子？
- 设计原则
  - 根据编程语言：Java，C#，C/C++，JS，AspectJ
  - 根据语言设计原则：OO，AOP
  - 根据应用场景：Web，Android，Ajax
  - 根据Bug类型：安全漏洞、内存缺陷、交互错误

变异算子定义

Name	Transformation	Example	Name	Transformation	Example
Cond. Bound.	Replaces one relational operator instance with another one (single replacement).	$< \rightsquigarrow \leq$	**Return Values**	Transforms the return value of a function (single replacement).	`return 0` ⇝ `return 1`
Negate Cond.	Negates one relational operator (single negation).	$== \rightsquigarrow !=$	**Void Meth. Call**	Deletes a call to a void method.	`void m()` ⇝
Remove Cond.	Replaces a cond. branch with true or false.	`if (...)` ⇝ `if (true)`	**Meth. Call**	Deletes a call to a non-void method.	`int m()` ⇝
Math	Replaces a numerical op. by another one (single replacement).	$+ \rightsquigarrow -$	**Con-structor Call**	Replaces a call to a constructor by null.	`new C()` ⇝ `null`
Incre-ments	Replace incr. with decr. and vice versa (single replacement).	$++ \rightsquigarrow --$	**Member Variable**	Replaces an assignment to a variable with the Java default values.	`a = 5` ⇝ `a`
Invert Neg.	Removes the negative from a variable.	$-a \rightsquigarrow a$	**Switch**	Replaces switch statement labels by the Java default ones.	
Inline Const.	Replaces a constant by another one or increments it.	$1 \rightsquigarrow 0, a \rightsquigarrow a+1$			

PIT基础算子[1,2]

[1] Coles H, Laurent T, Henard C, et al. Pit: a practical mutation testing tool for java[C]//Proceedings of the 25th international symposium on software testing and analysis. 2016: 449-452.
[2] PITest变异算子：https://pitest.org/quickstart/mutators/

变异算子定义

Name	Transformation	Example	Name	Transformation	Example
Cond. Bound.	Replaces one relational operator instance with another one (single replacement).	$< \rightsquigarrow \leq$	**Return Values**	Transforms the return value of a function (single replacement).	`return 0` ⇝ `return 1`
Negate Cond.	Negates one relational operator (single negation).	$== \rightsquigarrow !=$	**Void Meth. Call**	Deletes a call to a void method.	`void m()` ⇝
Remove Cond.	Replaces a cond. branch with true or false.	`if (...)` ⇝ `if (true)`	**Meth. Call**	Deletes a call to a non-void method.	`int m()` ⇝
Math	Replaces a numerical op. by another one (single replacement).	$+ \rightsquigarrow -$	**Con-structor Call**	Replaces a call to a constructor by null.	`new C()` ⇝ `null`
Incre-ments	Replace incr. with decr. and vice versa (single replacement).	$++ \rightsquigarrow --$	**Member Variable**	Replaces an assignment to a variable with the Java default values.	`a = 5` ⇝ `a`
Invert Neg.	Removes the negative from a variable.	$-a \rightsquigarrow a$	**Switch**	Replaces switch statement labels by the Java default ones.	
Inline Const.	Replaces a constant by another one or increments it.	$1 \rightsquigarrow 0, a \rightsquigarrow a+1$			

语法特征

PIT基础算子[1,2]

[1] Coles H, Laurent T, Henard C, et al. Pit: a practical mutation testing tool for java[C]//Proceedings of the 25th international symposium on software testing and analysis. 2016: 449-452.
[2] PITest变异算子：https://pitest.org/quickstart/mutators/

变异算子定义

Name	Transformation	Example	Name	Transformation	Example
Cond. Bound.	Replaces one relational operator instance with another one (single replacement).	$< \rightsquigarrow \leq$	**Return Values**	Transforms the return value of a function (single replacement).	`return 0` ⇝ `return 1`
Negate Cond.	Negates one relational operator (single negation).	$== \rightsquigarrow !=$	**Void Meth. Call**	Deletes a call to a void method.	`void m()` ⇝
Remove Cond.	Replaces a cond. branch with true or false.	`if (...)` ⇝ `if (true)`	**Meth. Call**	Deletes a call to a non-void method.	`int m()` ⇝
Math	Replaces a numerical op. by another one (single replacement).	$+ \rightsquigarrow -$	**Con-structor Call**	Replaces a call to a constructor by null.	`new C()` ⇝ `null`
Incre-ments	Replace incr. with decr. and vice versa (single replacement).	$++ \rightsquigarrow --$	**Member Variable**	Replaces an assignment to a variable with the Java default values.	`a = 5` ⇝ `a`
Invert Neg.	Removes the negative from a variable.	$-a \rightsquigarrow a$	**Switch**	Replaces switch statement labels by the Java default ones.	
Inline Const.	Replaces a constant by another one or increments it.	$1 \rightsquigarrow 0, a \rightsquigarrow a+1$			

OO特性

PIT基础算子[1,2]

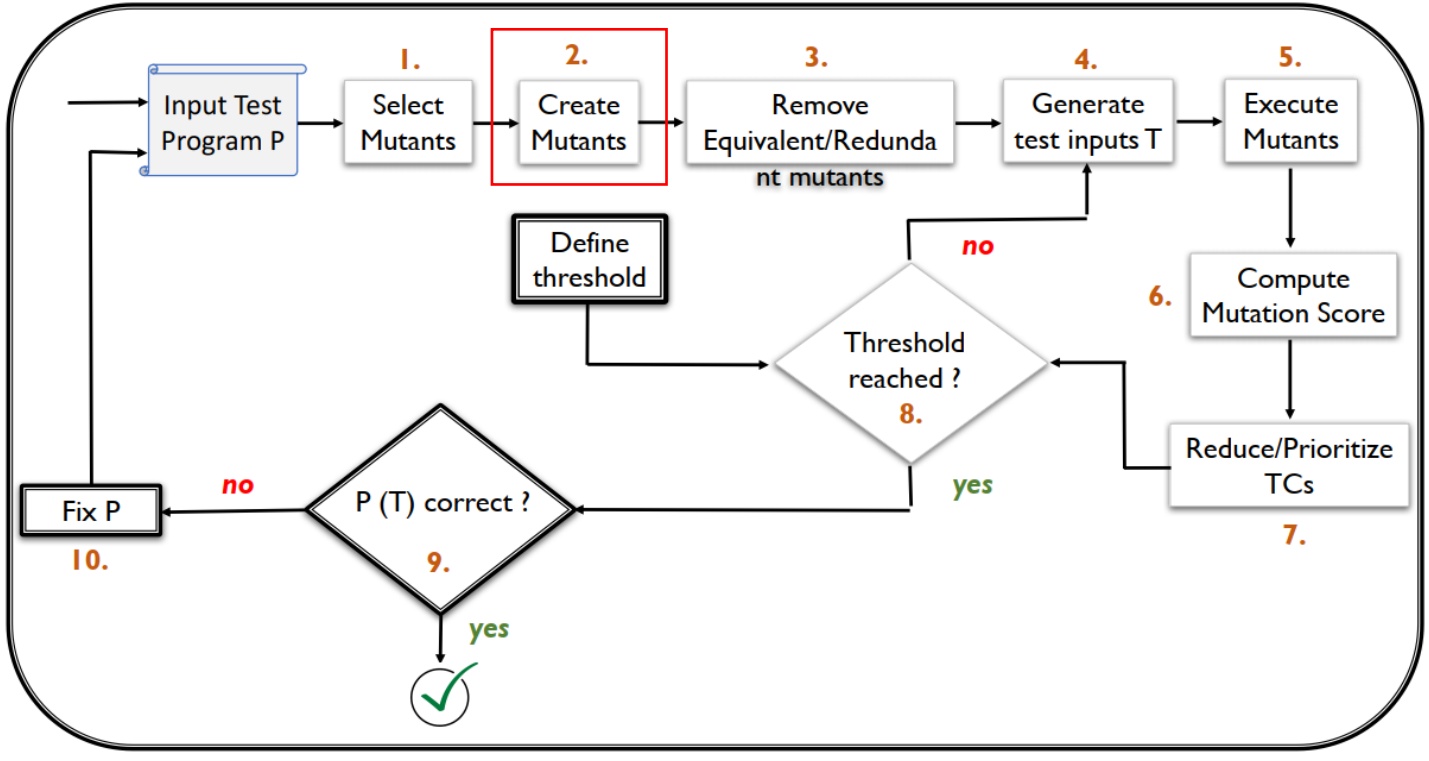
[1] Coles H, Laurent T, Henard C, et al. Pit: a practical mutation testing tool for java[C]//Proceedings of the 25th international symposium on software testing and analysis. 2016: 449-452.
[2] PITest变异算子：https://pitest.org/quickstart/mutators/

变异体约减策略

- 变异体约减（Mutant Reduction）
  - 变异体约减旨在从变异体全集中选出具有代表性变异体子集
  - 目的：减少变异测试所需的资源，提高可拓展性
- 约减策略
  - 随机选取[1]：选取总量的10%~60%，缺陷丢失率为6%~26%
  - 基于类型[2]：某些类型的变异体可能要更加重要
  - 基于分布[3]：利用AST选取更加分散的变异体

[1] Papadakis M, Malevris N. An empirical evaluation of the first and second order mutation testing strategies[C]//2010 Third International Conference on Software Testing, Verification, and Validation Workshops. IEEE, 2010: 90-99.
[2] Namin A S, Andrews J, Murdoch D. Sufficient mutation operators for measuring test effectiveness[C]//2008 ACM/IEEE 30th International Conference on Software Engineering. IEEE, 2008: 351-360.
[3]Just R, Kurtz B, Ammann P. Inferring mutant utility from program context[C]//Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2017: 284-294.

变异体生成


Input Test Program P
1.
Select Mutants
2.
Create Mutants
3.
Remove Equivalent/Redundant mutants
4.
Generate test inputs T
5.
Execute Mutants
6.
Compute Mutation Score
7.
Reduce/Prioritize TCs
8.
Threshold reached ?
no
yes
Define threshold
9.
P (T) correct ?
no
yes
10.
Fix P

变异体生成

- 变异体生成
  - 将选中的变异体（算子）实例化
  - 基本方式：为每个变体构建一个单独的源文件（Source File）
    - 缺陷：运行时间较长，会产生较大的额操作开销
- 研究内容：变异体生成技术
  - 元变异（Meta-mutation）
  - 基于字节码操作（Bytecode Manipulation）
  - 热替换（Hot-Swap, Just-In-Time）

元变异

- 元变异[1]
  - Meta-mutation / Mutation Schemata
  - 核心：程序模式（Program Schema）
    - 程序模板（Template） & 部分编译（Partially Interpreted）
    - 形式上类似于一般程序，但是包含自由标识符（Free Identifier），i.e. 用一些符号替换程序中的结构（变量、语句）
  - 目的：减少生成变异体时所需的编译次数；集中存储变异体

[1] Untch R H, Offutt A J, Harrold M J. Mutation analysis using mutant schemata[C]//Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis. 1993: 139-148.

元变异

- 元变异[1]

```
Original=> Delta := NewGuess - Sqrt;
--------- -----------------------------------
[Mutagens] / [Mutations]
----- -----------------------------------
[ABS] Delta := ABS(NewGuess) - Sqrt;
[ABS] Delta := NEGABS(NewGuess) - Sqrt;
[AOR] Delta := NewGuess + Sqrt;
[AOR] Delta := NewGuess * Sqrt;
[CSR] Delta := 0.001 - Sqrt;
[CSR] Delta := 2 - Sqrt;
[SVR] Number := NewGuess - Sqrt;
[SVR] Sqrt := NewGuess - Sqrt;
[UOI] Delta := -NewGuess - Sqrt;
[UOI] Delta := ++(NewGuess) - Sqrt;
```

部分元变异体

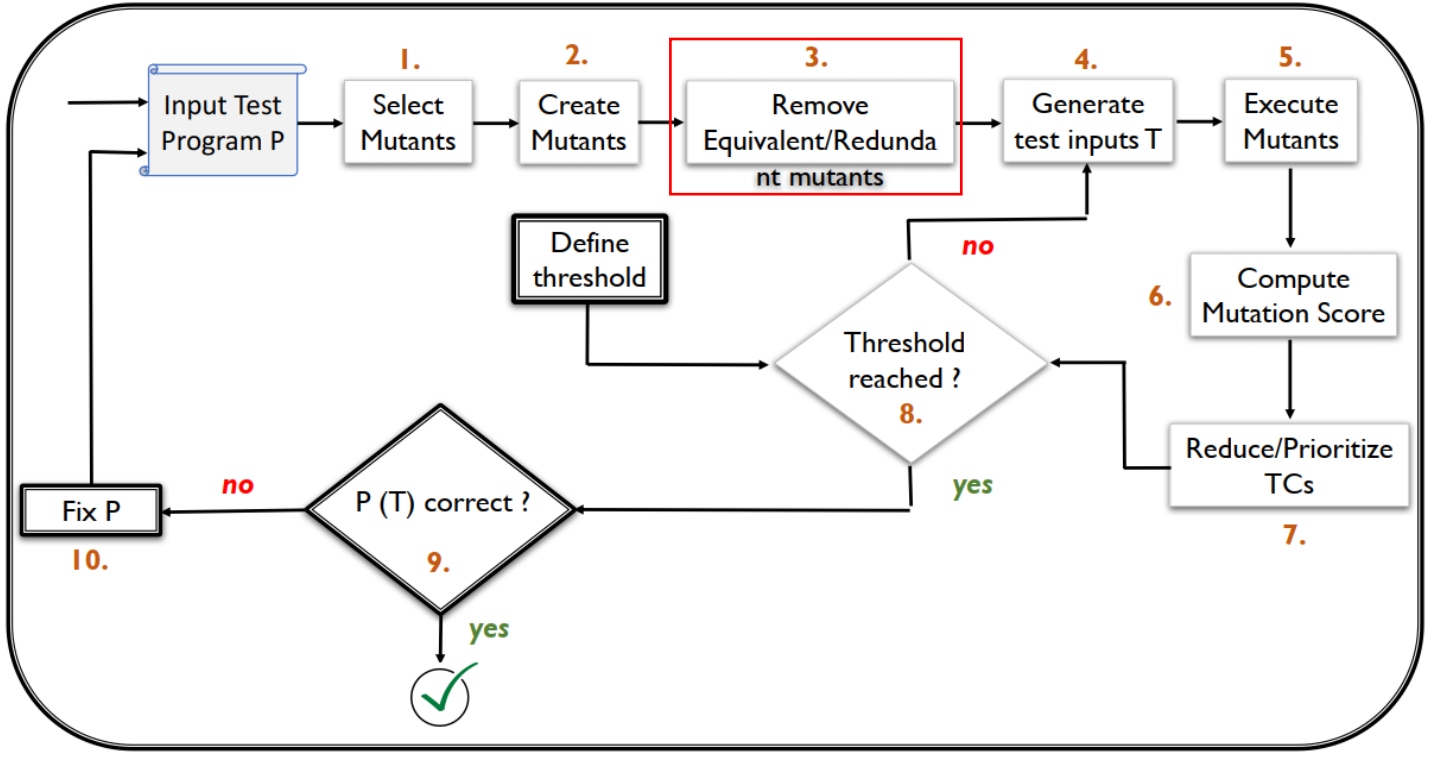
[1] Untch R H, Offutt A J, Harrold M J. Mutation analysis using mutant schemata[C]//Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis. 1993: 139-148.

基于字节码操作

- 基于字节码操作的变异体生成
  - 避免编译，从而减少时间开销
  - 操作目标：中间表示（Intermedia Interpretation），如：Java字节码、.NET和LLVM的通用中间表示
  - 代表工作：PIT（Java）[1]、Mull[2]（LLVM for C）

[1] Coles H, Laurent T, Henard C, et al. Pit: a practical mutation testing tool for java[C]//Proceedings of the 25th international symposium on software testing and analysis. 2016: 449-452.
[2] Denisov A, Pankevich S. Mull it over: mutation testing based on LLVM[C]//2018 IEEE international conference on software testing, verification and validation workshops (ICSTW). IEEE, 2018: 25-31.

变异体优化


Input Test Program P
1.
Select Mutants
2.
Create Mutants
3.
Remove Equivalent/Redundant mutants
4.
Generate test inputs T
5.
Execute Mutants
6.
Compute Mutation Score
7.
Reduce/Prioritize TCs
8.
Threshold reached ?
no
yes
Define threshold
9.
P (T) correct ?
no
yes
10.
Fix P

变异体优化

- 变异体优化
  - 内容：去除有等价和无效变异体
  - 目的
    - 减少后续执行变异体阶段的开销
    - 提高变异得分的可信度
- 基本形式
  - 通过**静态分析**的方式，**识别**并移除有问题的变异体

变异体优化

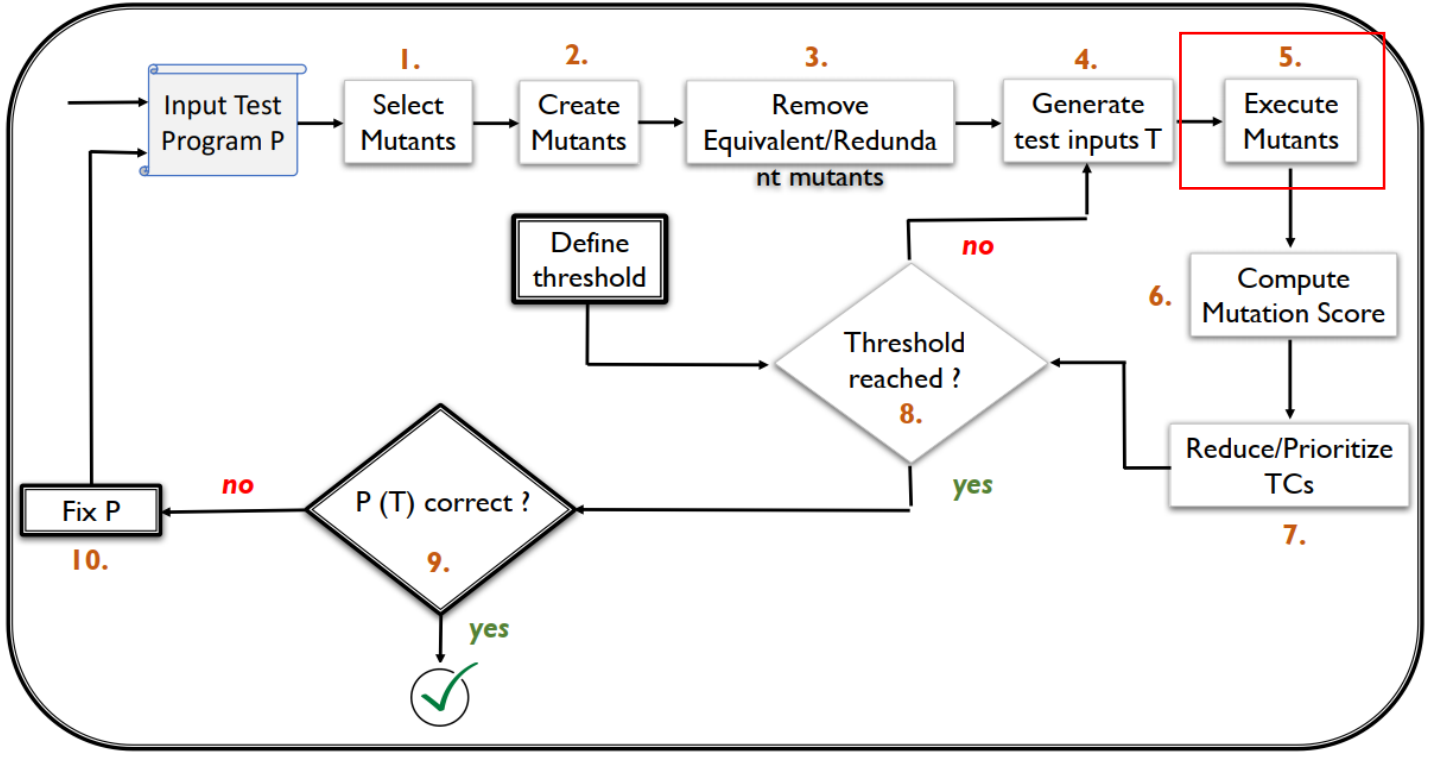
- 识别等价变异体
  - 特征：不可确定问题（Undecidable problem），无法建立识别所有对等变异体的算法，只能设计相应的启发式策略
  - 常用技术
    - 代码优化（Code optimization）。对源代码和变异体进行优化，移除优化后与源代码相同的变异体
    - 静态数据流分析，如值分析（Value Analysis）。等价变异体汇总存在特定的数据流模板，可以由此识别等价变异体

变异体优化

- 识别冗余变异体
  - 操作符的角度：如>, ==, <之间的有机组合[1]
  - 缺陷层级（Fault Hierarchy）角度[2]：变异体之间的从属关系
  - 程序分析角度：代码优化[3]、符号执行[4]

[1] Papadakis M, Malevris N. Mutation based test case generation via a path selection strategy[J]. Information and Software Technology, 2012, 54(9): 915-932.
[2] Hariri F, Shi A, Converse H, et al. Evaluating the effects of compiler optimizations on mutation testing at the compiler IR level[C]//2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2016: 105-115.[2] Namin A S, Andrews J, Murdoch D. Sufficient mutation operators for measuring test effectiveness[C]//2008 ACM/IEEE 30th International Conference on Software Engineering. IEEE, 2008: 351-360.
[3] Kintis M, Papadakis M, Jia Y, et al. Detecting trivial mutant equivalences via compiler optimisations[J]. IEEE Transactions on Software Engineering, 2017, 44(4): 308-333.
[4] Kurtz B, Ammann P, Offutt J. Static analysis of mutant subsumption[C]//2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2015: 1-10.

变异体执行


Input Test Program P
1.
Select Mutants
2.
Create Mutants
3.
Remove Equivalent/Redundant mutants
4.
Generate test inputs T
5.
Execute Mutants
6.
Compute Mutation Score
7.
Reduce/Prioritize TCs
8.
Threshold reached ?
Define threshold
no
yes
9.
P (T) correct ?
no
yes
10.
Fix P

变异体执行

- 变异体执行
  - 特征：变异测试过程中最昂贵的阶段
  - 基本形式：假设一个待测程序P有n个变异体和一个包含m个测试用例的测试套件，则执行变异体的最大次数为$n \times m$。
- 研究内容：针对执行优化的两大场景优化执行过程
  - 场景A：计算变异得分
  - 场景B：计算**变异体矩阵**（Mutant Matrix）

变异体执行

- 变异体矩阵
  - 记录每个测试用例对每个变异体的执行情况
  - $n \times m$维矩阵：n为变异体个数，m为测试用例个数

Tests	Mutants			
	m_1	m_2	m_3	m_4
t_1	✓			✓
t_2	✓			✓
t_3			✓	✓
t_4			✓	
t_5			✓	✓

4×5变异体矩阵

- 计算变异体矩阵所需执行次数： 20（5*4）
- 计算变异得分所需执行次数： **10** （1+5+3+1）

变异体执行

- 优化策略
  - 改变测试用例的顺序[1,2]（TCP）→ A

[1] Zhang L, Marinov D, Khurshid S. Faster mutation testing inspired by test prioritization and reduction[C]//Proceedings of the 2013 International Symposium on Software Testing and Analysis. 2013: 235-245.
[2] Just R, Kapfhammer G M, Schweiggert F. Using non-redundant mutation operators and test suite prioritization to achieve efficient and scalable mutation analysis[C]//2012 IEEE 23rd International Symposium on Software Reliability Engineering. IEEE, 2012: 11-20.
[3] Zhu Q, Panichella A, Zaidman A. Speeding-up mutation testing via data compression and state infection[C]//2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2017: 103-109.
[4] M. E. Delamaro, Proteum - A Mutation Analysis Based Testing Environmen, Masters thesis, University of S~ao Paulo, Sao Paulo, Brazil (1993).
[5] Mateo P R, Usaola M P. Mutant execution cost reduction: Through music (mutant schema improved with extra code)[C]//2012 IEEE Fifth International Conference on Software Testing, Verification and Validation. IEEE, 2012: 664-672.

变异体执行

- 优化策略
  - 改变测试用例的顺序[1,2]（TCP）→ A

Tests	Mutants			
	m_1	m_2	m_3	m_4
t_1	✓			✓
t_2	✓			✓
t_3			✓	✓
t_4			✓	
t_5			✓	✓

t1，t2，t3，t4，t5 → t1，**t3，t2**，t4，t5

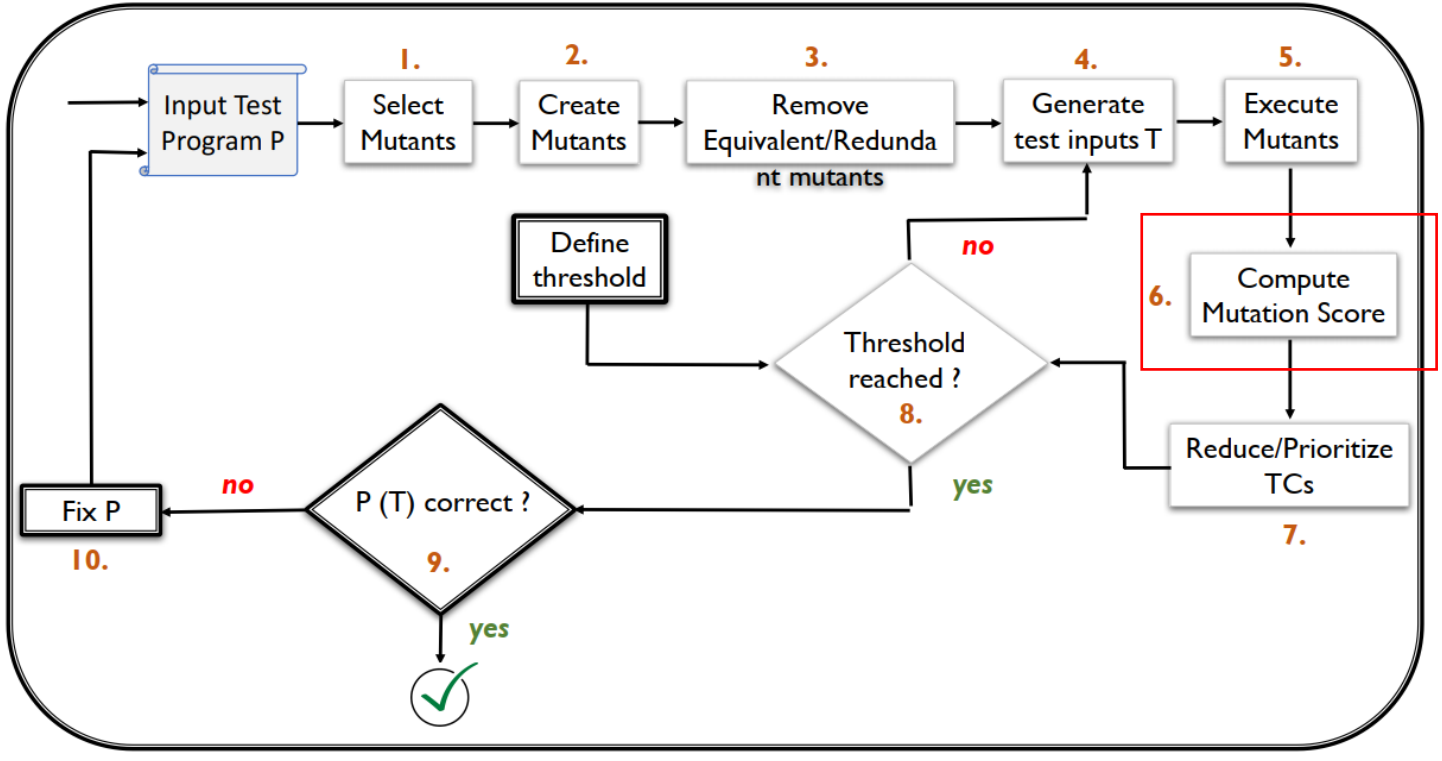
执行次数：10 → **9**（1+5+2+1）

变异体执行

- 优化策略
  - 改变测试用例的顺序[1,2]（TCP）→ A
  - 匹配测试用例与变异体[3] → A
  - 避免执行必定存活的变异体[4] → A，B
  - 限定变异体的执行时间[5] → A，B

[1] Zhang L, Marinov D, Khurshid S. Faster mutation testing inspired by test prioritization and reduction[C]//Proceedings of the 2013 International Symposium on Software Testing and Analysis. 2013: 235-245.
[2] Just R, Kapfhammer G M, Schweiggert F. Using non-redundant mutation operators and test suite prioritization to achieve efficient and scalable mutation analysis[C]//2012 IEEE 23rd International Symposium on Software Reliability Engineering. IEEE, 2012: 11-20.
[3] Zhu Q, Panichella A, Zaidman A. Speeding-up mutation testing via data compression and state infection[C]//2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2017: 103-109.
[4] M. E. Delamaro, Proteum - A Mutation Analysis Based Testing Environmen, Masters thesis, University of S~ao Paulo, Sao Paulo, Brazil (1993).
[5] Mateo P R, Usaola M P. Mutant execution cost reduction: Through music (mutant schema improved with extra code)[C]//2012 IEEE Fifth International Conference on Software Testing, Verification and Validation. IEEE, 2012: 664-672.

变异得分计算


Input Test Program P
1.
Select Mutants
2.
Create Mutants
3.
Remove Equivalent/Redunda nt mutants
4.
Generate test inputs T
5.
Execute Mutants
6.
Compute Mutation Score
7.
Reduce/Prioritize TCs
8.
Threshold reached ?
no
yes
Define threshold
9.
P (T) correct ?
no
yes
10.
Fix P

变异得分计算

- 变异得分计算
  - 计算被杀死的变异体数量，进而得到变异得分、量化测试的充分性
- 研究内容
  - 变异杀死的条件：确定一个变异体是否被杀死
  - 测试预言的生成：如何生成更多、更好的测试预言来杀死更多的变异体、提高变异得分

变异得分计算

- 变异杀死的条件
  - 核心：定义程序行为（Program behavior）
  - Weak，Firm，Strong Mutation
  - 代表性工作
    - 针对确定性系统：定义新的系统级行为[1]
    - 针对**非确定性**系统：定义、模拟程序规格[2]

[1] Mateo P R, Usaola M P, Alemán J L F. Validating second-order mutation at system level[J]. IEEE Transactions on Software Engineering, 2012, 39(4): 570-587.
[2] Patrick M, Craig A P, Cunniffe N J, et al. Testing stochastic software using pseudo-oracles[C]//Proceedings of the 25th International Symposium on Software Testing and Analysis. 2016: 235-246.

03

变异测试应用

应用总览

- 评估作用：利用变异得分度量测试的充分性
- 引导作用：利用变异测试/分析的结果来引导测试过程
- 传统应用：应用于确定性系统
  - 测试生成（Test Generation）
  - 预言生成（Oracle Problem）
  - 测试优化（排序 & 选择）
  - Debug引导（缺陷定位 & 自动修复）
- 变异 & AI：应用于非确定性系统（DNN）

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - 基于搜索的软件测试（SBST）+ 变异分析
  - SBST：将软件测试过程转化为搜索问题，利用启发式搜索算法自动化完成测试目标
    - 常见算法：遗传算法、登山法
    - 核心：Fitness function
  - 将变异分析得到的信息嵌入SBST框架，利用变异分析的结果引导单元测试以及测试预言的生成

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

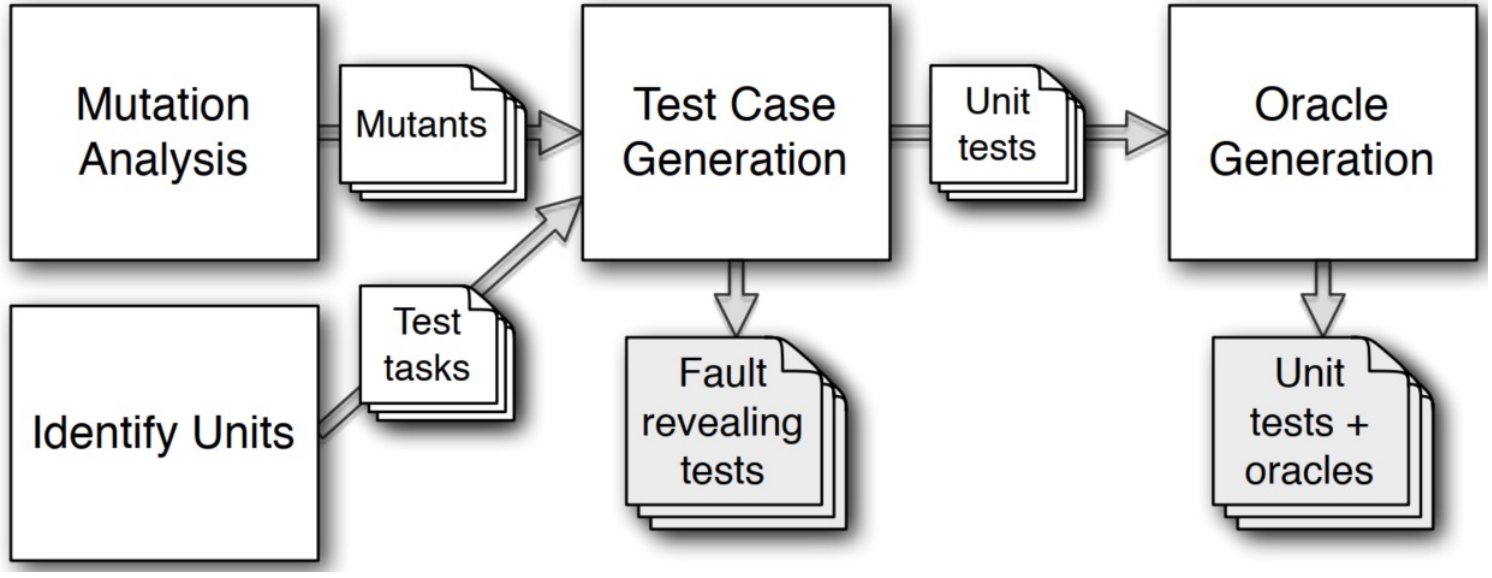
用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - μTest：遗传算法 + 变异分析
  - 用例生成：GA建模（Crossover、Mutate、Fitness Function）
  - 预言生成：程序行为的甄选（What is expected?）

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - μTest：遗传算法（Genetic Algorithm，GA） + 变异分析

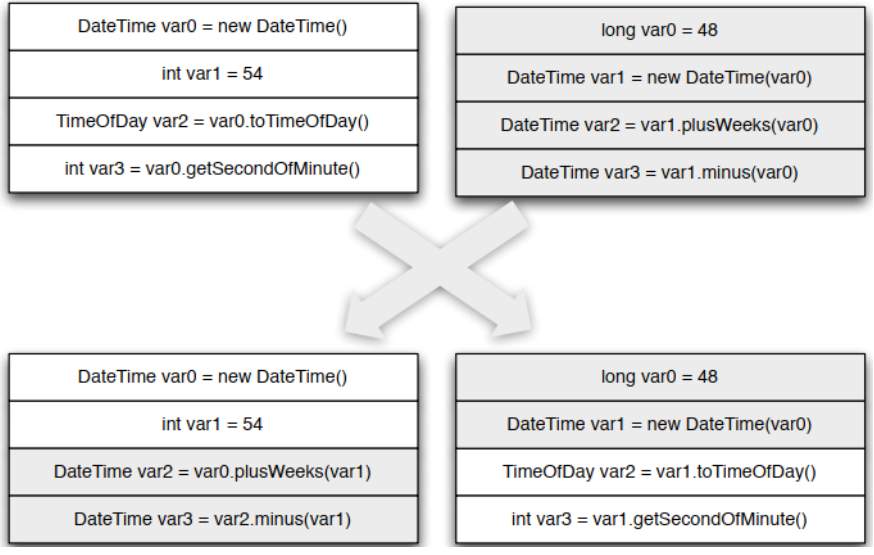


μTest整体流程

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Crossover（杂交）：交换两个体（测试用例）的部分语句



μTest杂交操作

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Mutate（突变）：删除、插入、修改

操作（针对单条语句）	名称	描述
删除（Delete)	Delete a statement	删除一条语句
插入（Insert)	Insert an object	创建一个已在测试用例中使用的类型的新对象
	Insert a method call	为测试用例的对象之一添加随机方法调用
修改（Modify)	Change callee	改变某条方法调用或域引用的源对象
	Change parameters	改变某条方法调用或构造方法的某个参数
	Change method	将某条方法调用替换为另一个具有相同返回值的方法调用
	Change constructor	将某个构造方法替换为其他同类构造方法
	Change field	将某个域引用替换为另一条在相同类上的同类型域引用
	Change primitive	替换某个基础类型的值

μTest突变操作

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - D_f：个体达成“执行变异体”的距离

问题：需要生成多少方法调用和参数才能执行（某些）变异体？

要求：D_f值越小越好

$$d(c,t) = 1 + \#params + hava\_callee$$

$$D_f(t) = \min\{d(c,t) | calls\ c\ related\ to\ mutation\}$$

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - D_f：个体达成“执行变异体”的距离

问题：需要生成多少方法调用和参数才能执行（某些）变异体？

要求：D_f值越小越好

$$d(c,t) = 1 + \#params + hava\_callee$$

$$D_f(t) = \min\{d(c,t) | calls\ c\ related\ to\ mutation\}$$

与突变相关的方法调用

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - D_f：个体达成“执行变异体”的距离

问题：需要生成多少方法调用和参数才能执行（某些）变异体？

要求：D_f值越小越好

$$d(c,t) = 1 + \#params + hava\_callee$$

$$D_f(t) = \min\{d(c,t) | calls\ c\ related\ to\ mutation\}$$

个体测试用例

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - D_f：个体达成“执行变异体”的距离

问题：需要生成多少方法调用和参数才能执行（某些）变异体？

要求：D_f值越小越好

满足条件所需的参数个数

$$d(c,t) = 1 + \#params + hava\_callee$$

$$D_f(t) = \min\{d(c,t) | calls\ c\ related\ to\ mutation\}$$

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - D_f：个体达成“执行变异体”的距离

问题：需要生成多少方法调用和参数才能执行（某些）变异体？

要求：D_f值越小越好

t是否包含被突变的方法

$$d(c,t) = 1 + \#params + hava\_callee$$

$$D_f(t) = \min\{d(c,t) | calls\ c\ related\ to\ mutation\}$$

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - D_m：个体达成“执行变异语句”的距离

问题：需要走过哪些分支才能到达变异位置？

要求：D_m值越小越好

⇒ $D_m(t) = Approach + Branch/Necessity\ Distance$

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - D_m：个体达成“执行变异语句”的距离

问题：需要走过哪些分支才能到达变异位置？

要求：D_m值越小越好

借用传统SBST常用的分支距离

$$D_m(t) = Approach + Branch/Necessity\ Distance$$

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - I_m：变异体本身造成的影响，i.e. 缺陷的感染与传播

问题：人工缺陷对输出部分的影响如何？

要求：I_m值越大越好

$$I_m(t) = \frac{c \times |C| + r \times |A|}{1 + |t|}$$

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - I_m：变异体本身造成的影响，i.e. 缺陷的感染与传播

问题：人工缺陷对输出部分的影响如何？

要求：I_m值越大越好

覆盖次数发生改变的调用语句

$$I_m(t) = \frac{c \times |C| + r \times |A|}{1 + |t|}$$

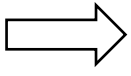
[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - I_m：变异体本身造成的影响，i.e. 缺陷的感染与传播

问题：人工缺陷对输出部分的影响如何？

要求：I_m值越大越好



可观测的行为变化

$$I_m(t) = \frac{c \times |C| + r \times |A|}{1 + |t|}$$

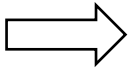
[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - I_m：变异体本身造成的影响，i.e. 缺陷的感染与传播

问题：人工缺陷对输出部分的影响如何？

要求：I_m值越大越好



可调节的参数

$$I_m(t) = \frac{c \times |C| + r \times |A|}{1 + |t|}$$

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

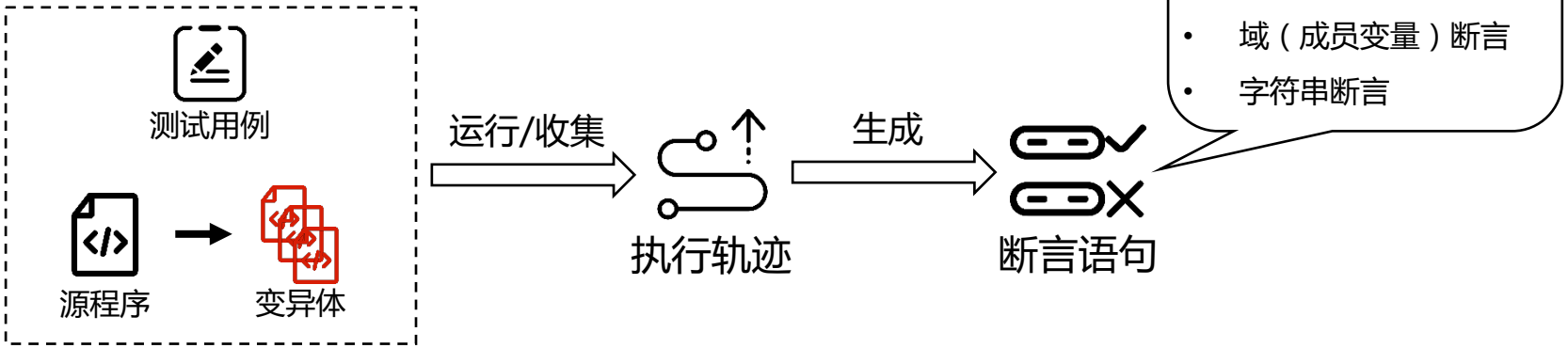
- 变异引导的单元测试及预言生成[1]
  - Fitness Function（适应性方程）：D_f、D_m、I_m
  - $fitness$：组合三个部分得到整体适应性方程

$$fitness\,(t) = \frac{1}{D_f(t) + D_m(t)} + I_m(t)$$

[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

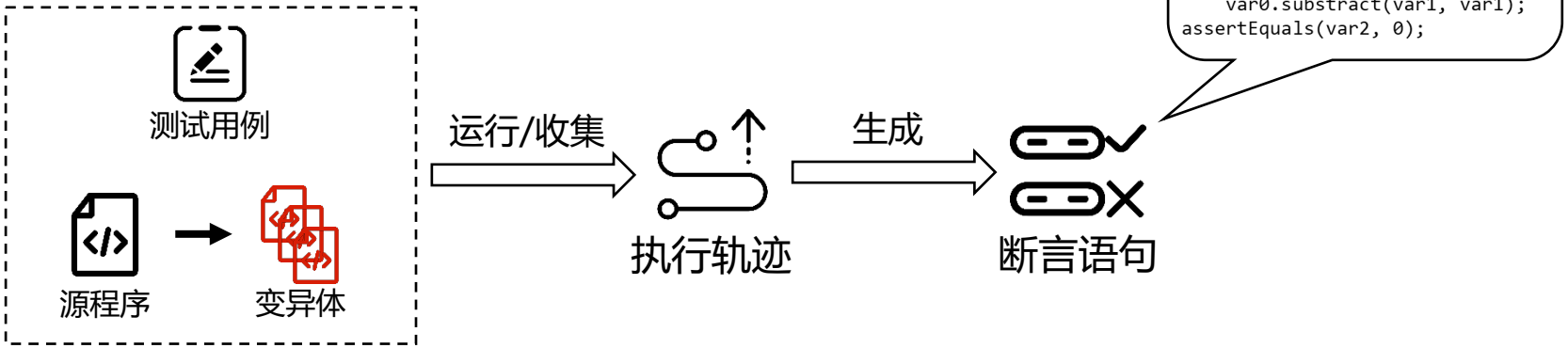
- 变异引导的单元测试及预言生成[1]
  - 预言生成：生成断言以杀死变异体
  - 基本思路：添加能够杀死变异体的断言语句 → 检测到变异体与源程序之间的不同行为
  - 变异引导的优势：揭示“Where”&“What”



[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

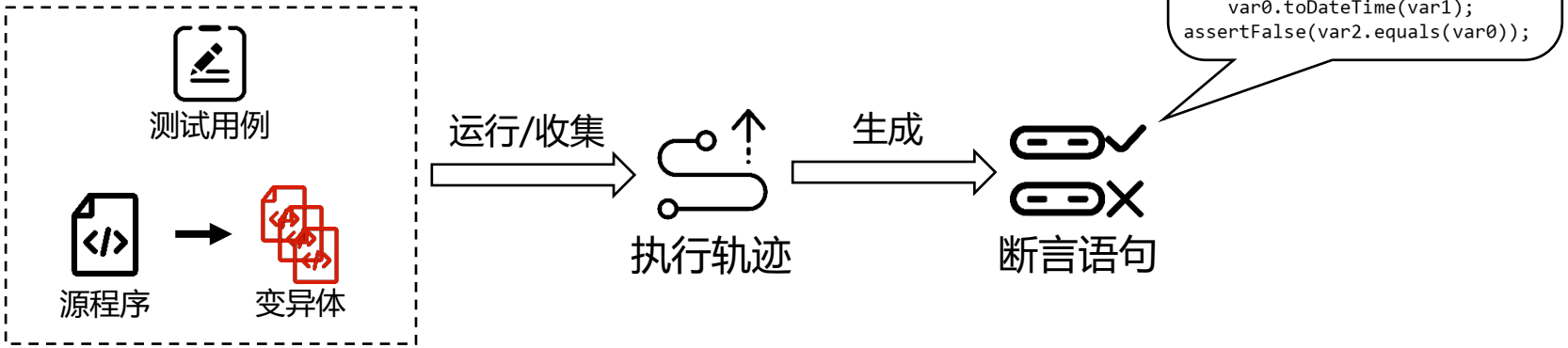
- 变异引导的单元测试及预言生成[1]
  - 预言生成：生成断言以杀死变异体
  - 基本思路：添加能够杀死变异体的断言语句 → 检测到变异体与源程序之间的不同行为
  - 变异引导的优势：揭示“Where”&“What”



[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

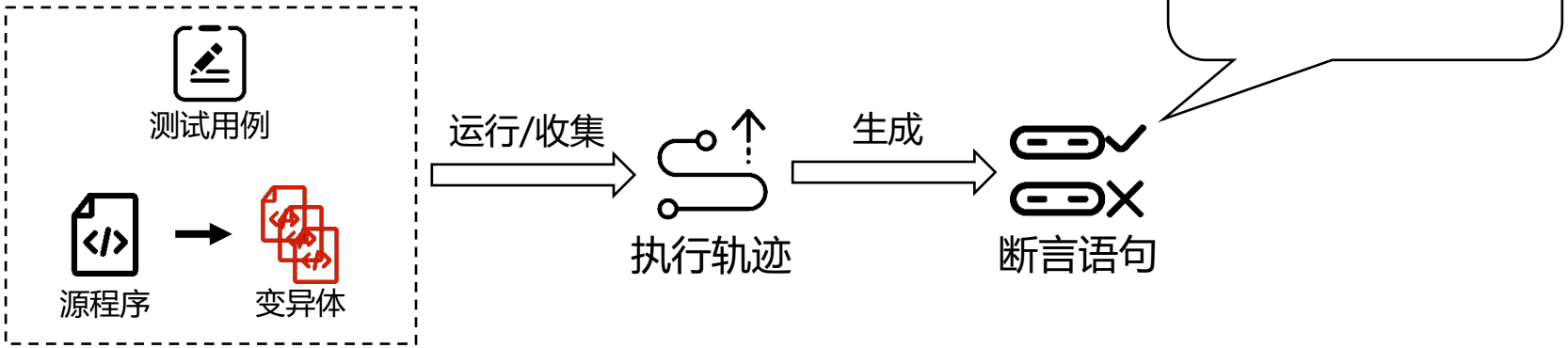
- 变异引导的单元测试及预言生成[1]
  - 预言生成：生成断言以杀死变异体
  - 基本思路：添加能够杀死变异体的断言语句 → 检测到变异体与源程序之间的不同行为
  - 变异引导的优势：揭示“Where”&“What”



[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

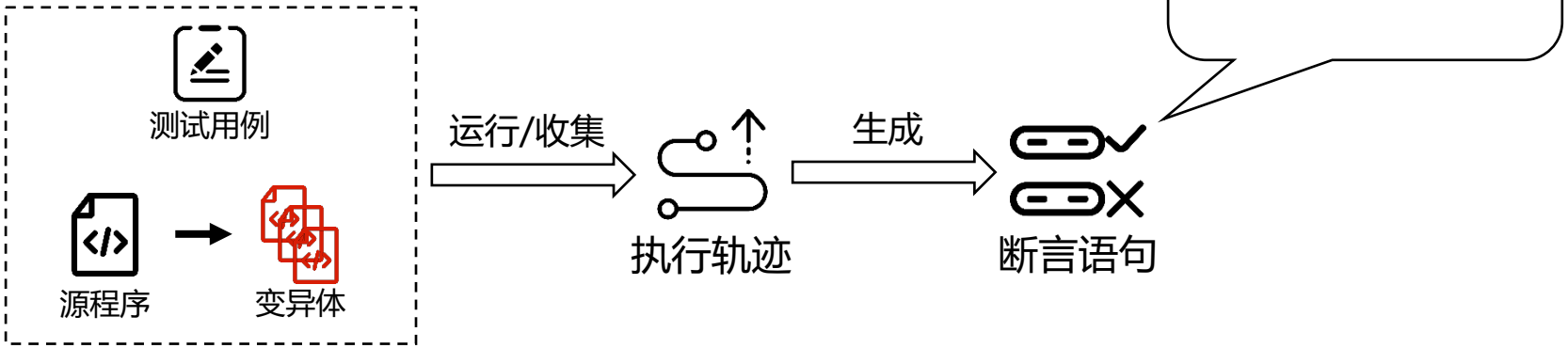
- 变异引导的单元测试及预言生成[1]
  - 预言生成：生成断言以杀死变异体
  - 基本思路：添加能够杀死变异体的断言语句 → 检测到变异体与源程序之间的不同行为
  - 变异引导的优势：揭示“Where”&“What”



[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

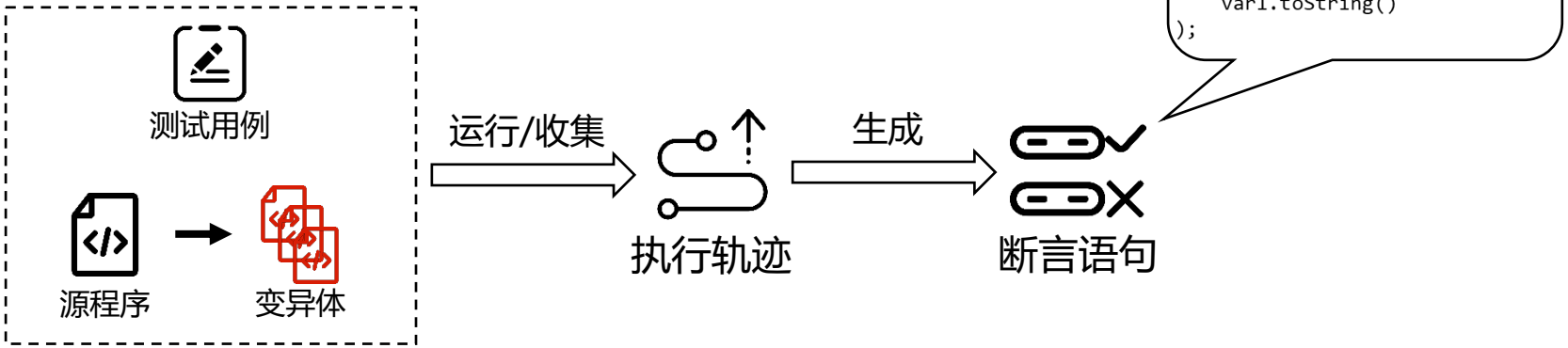
- 变异引导的单元测试及预言生成[1]
  - 预言生成：生成断言以杀死变异体
  - 基本思路：添加能够杀死变异体的断言语句 → 检测到变异体与源程序之间的不同行为
  - 变异引导的优势：揭示“Where”&“What”



[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

用例与预言生成

- 变异引导的单元测试及预言生成[1]
  - 预言生成：生成断言以杀死变异体
  - 基本思路：添加能够杀死变异体的断言语句 → 检测到变异体与源程序之间的不同行为
  - 变异引导的优势：揭示“Where”&“What”



[1] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles[J]. IEEE Transactions on Software Engineering, 2011, 38(2): 278-292.

变异辅助Debug

- 利用变异自动为有缺陷程序推荐补丁[1]
  - Debug的两个阶段：
    - 定位：判断缺陷所在位置、分析可疑的缺陷特征
    - 修复：着手修复缺陷本身
  - 自动Debug的挑战
    - 缺少缺陷版本作为试验评估的数据集
    - 变异分析的自身特性为自动Debug提供了数据支持

[1] Debroy V, Wong W E. Using mutation to automatically suggest fixes for faulty programs[C]//2010 Third International Conference on Software Testing, Verification and Validation. IEEE, 2010: 65-74.

变异辅助Debug

- 利用变异自动为有缺陷程序推荐补丁[1]
  - 缺陷定位技术
    - 抽象测试轨迹，并利用这些轨迹计算程序组件的可疑度（Suspiciousness）
    - 以可疑度作为指标，挑选出最有可能的缺陷组件（Faulty Component）
  - 自动修复技术
    - 根据一定的语法规则转换缺陷程序为正确版本
    - 结合变异：是否可以**逆向利用**变异过程进行自动修复？

[1] Debroy V, Wong W E. Using mutation to automatically suggest fixes for faulty programs[C]//2010 Third International Conference on Software Testing, Verification and Validation. IEEE, 2010: 65-74.

变异辅助Debug

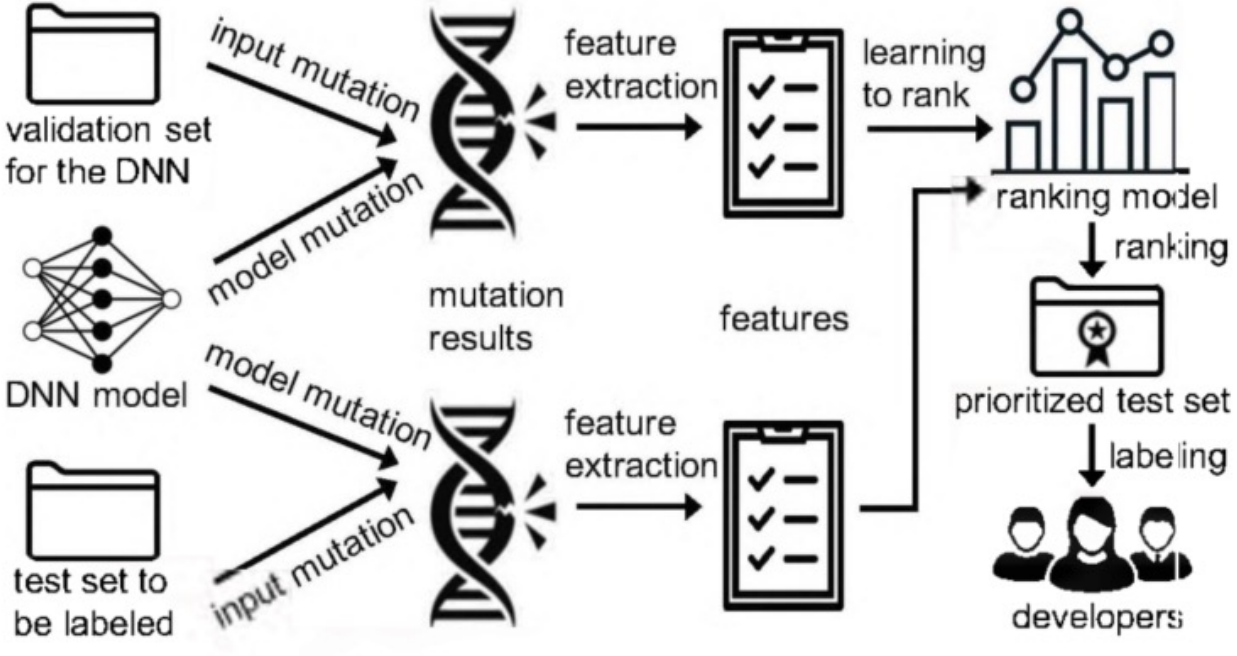
- 利用变异自动为有缺陷程序推荐补丁[1]
  - 技术挑战：全局变异分析会带来较大的开销，降低整个流程的效率
  - 解决方案
    - 利用缺陷定位技术进行可疑度排序
    - 高可疑组件即为最可能的缺陷位置
    - 按照可疑度从高到低的顺序对组件进行变异

[1] Debroy V, Wong W E. Using mutation to automatically suggest fixes for faulty programs[C]//2010 Third International Conference on Software Testing, Verification and Validation. IEEE, 2010: 65-74.

- 通过变异分析进行DNN测试输入排序[1]
  - 变异分析 + 测试排序 + AI测试
  - TCP for DNN：回答“谁应该最先被打标签？”
  - MA for DNN：模型的变异、输入的变异
  - 核心思路：能够杀死最多**变异模型**的，且**变异后**能够产生最多预测结果的**测试输入**最有可能暴露DNN模型中存在的缺陷

[1] Wang Z, You H, Chen J, et al. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 397-409.

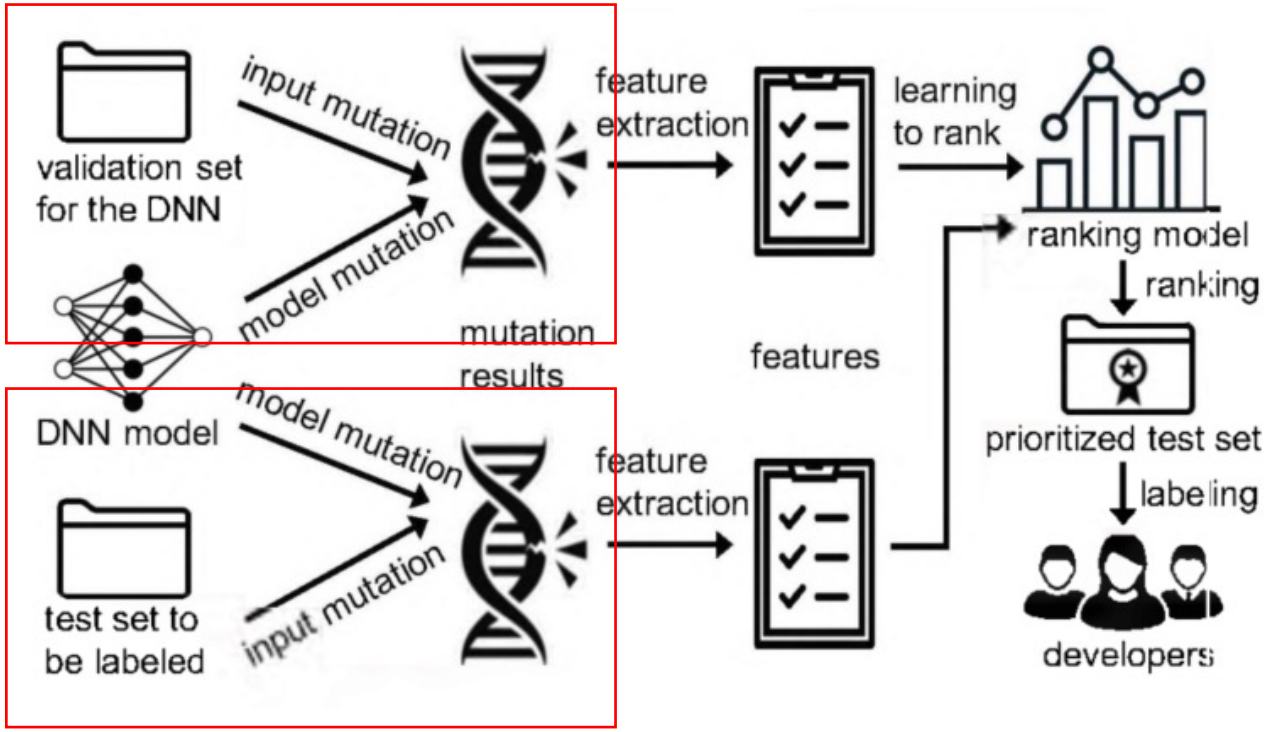
- 通过变异分析进行DNN测试输入排序[1]



[1] Wang Z, You H, Chen J, et al. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 397-409.

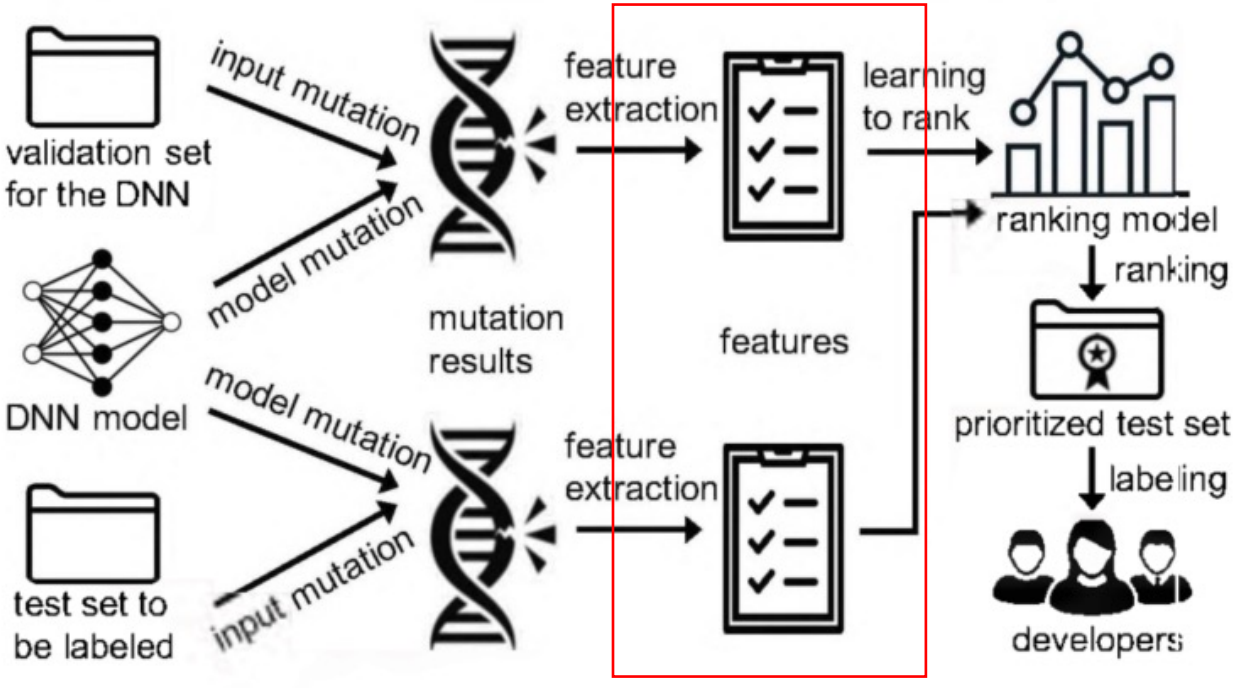
- 通过变异分析进行DNN测试输入排序[1]

两方面变异



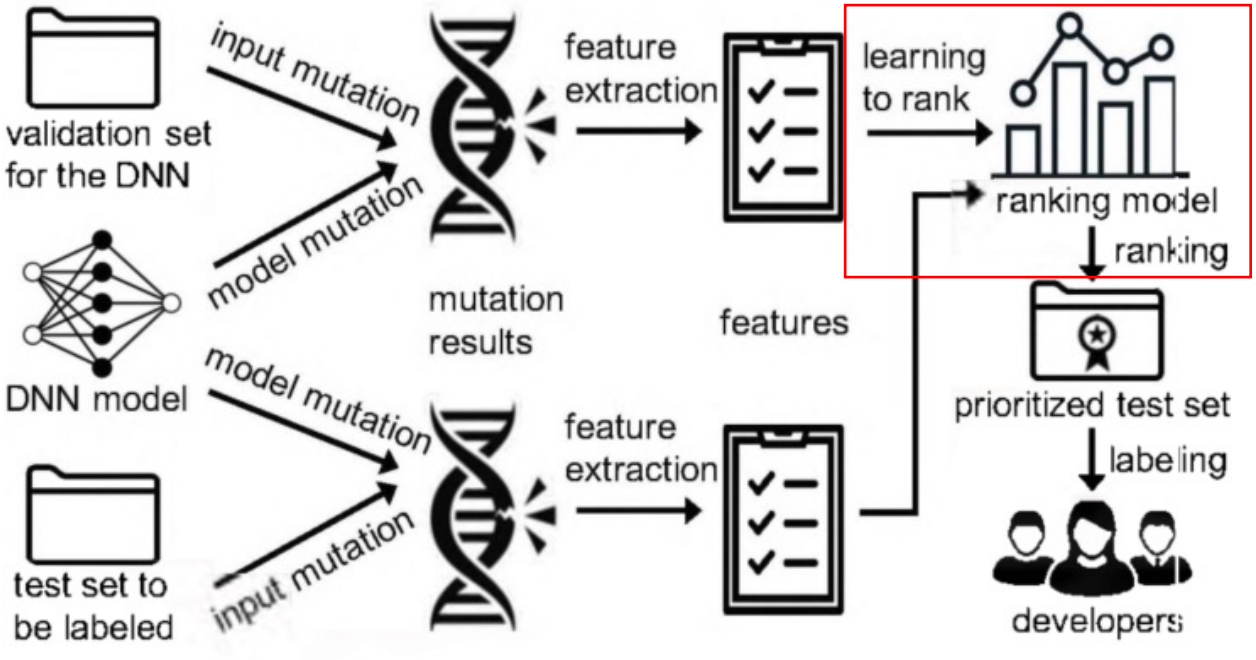
[1] Wang Z, You H, Chen J, et al. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 397-409.

- 通过变异分析进行DNN测试输入排序[1]



[1] Wang Z, You H, Chen J, et al. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 397-409.

- 通过变异分析进行DNN测试输入排序[1]



[1] Wang Z, You H, Chen J, et al. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 397-409.

- 通过变异分析进行DNN测试输入排序[1]
  - 定义变异规则/算子：模型变异

助记符	名称	描述
NAI	Neuron Activation Inverse	取反神经元输出，变换神经元激活状态
NEB	Neuron Effect Block	设置神经元输出为0，阻塞神经元权重传递
GF	Gauss Fuzzing	遵循高斯分布，为神经元权重添加噪声
WS	Weight Shuffling	重组上一层神经元的权重

[1] Wang Z, You H, Chen J, et al. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 397-409.

- 通过变异分析进行DNN测试输入排序[1]
  - 定义变异规则/算子：输入变异

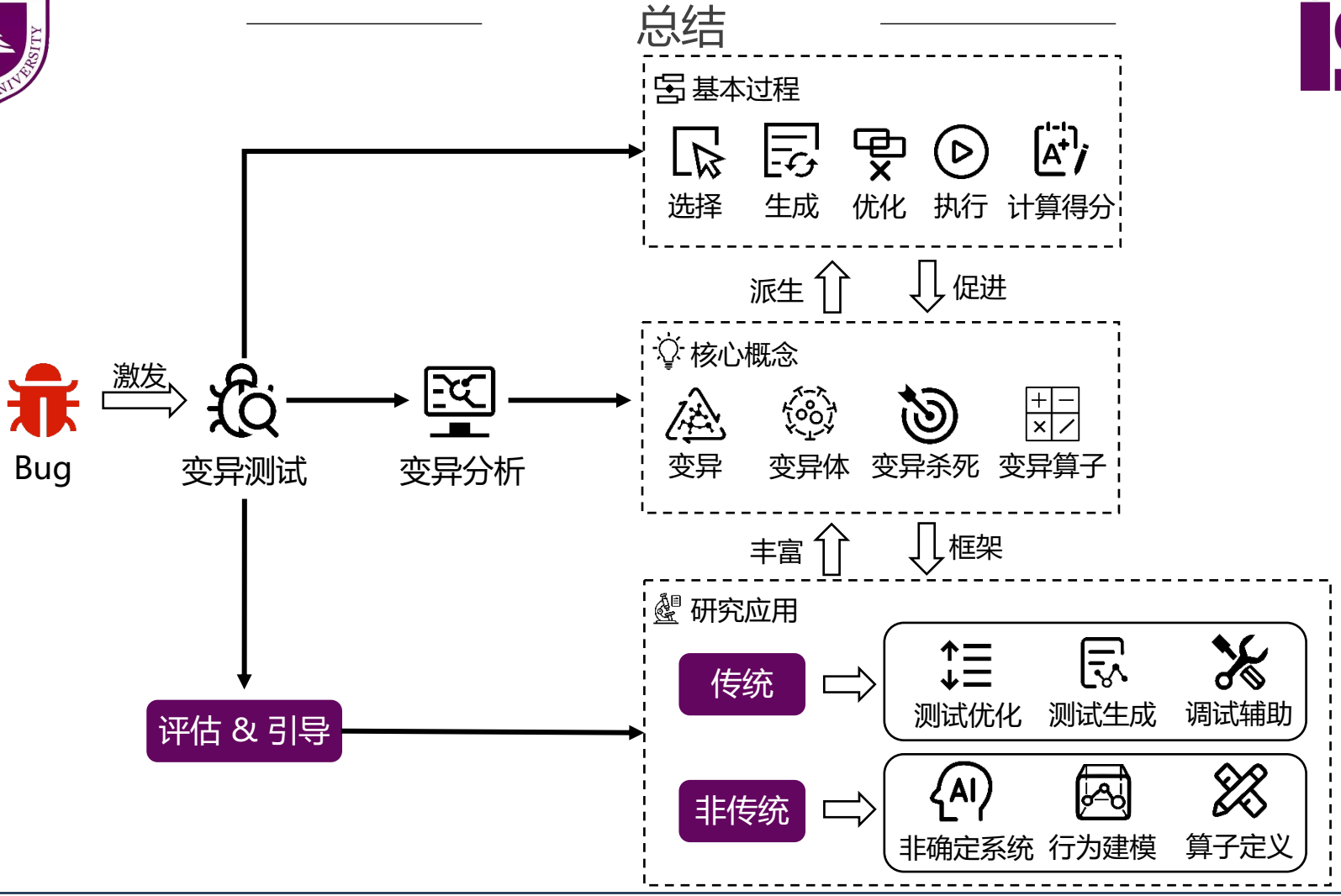
类型	助记符	名称	描述
图像	PGF	Pixel Gauss Fuzzing	遵循高斯分布，为选中的像素点添加噪声
	PS	Pixels Shuffling	重组选中的像素点
	CPW	Coloring Pixel White	将选中的像素点变为白色
	CPB	Coloring Pixel Black	将选中的像素点变为黑色
	PCR	Pixel Color Reverse	转置选中像素点的颜色
文本	CS	Character Shuffling	重组选中的字符
	CRL	Character Replacement	将选中的字符随机替换成字符全集中的其他字符
	CRE	Character Repetition	重复选中的字符
预设特征	DVR	Discrete Value Replacement	将选中的特征值随机替换成离散值全集中的其他值
	CVM	Continuous Value Modification	随机增加或减少选中值的e%

[1] Wang Z, You H, Chen J, et al. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 397-409.

04

总结

总结


基本过程
选择
生成
优化
执行
计算得分
派生
促进
核心概念
变异
变异体
变异杀死
变异算子
丰富
框架
研究应用
传统
测试优化
测试生成
调试辅助
非传统
非确定系统
行为建模
算子定义
Bug
激发
变异测试
变异分析
评估 & 引导

zychen@nju.edu.cn
fangchunrong@nju.edu.cn

Thank you!