测试预言问题

南京大学 软件学院 iSE实验室

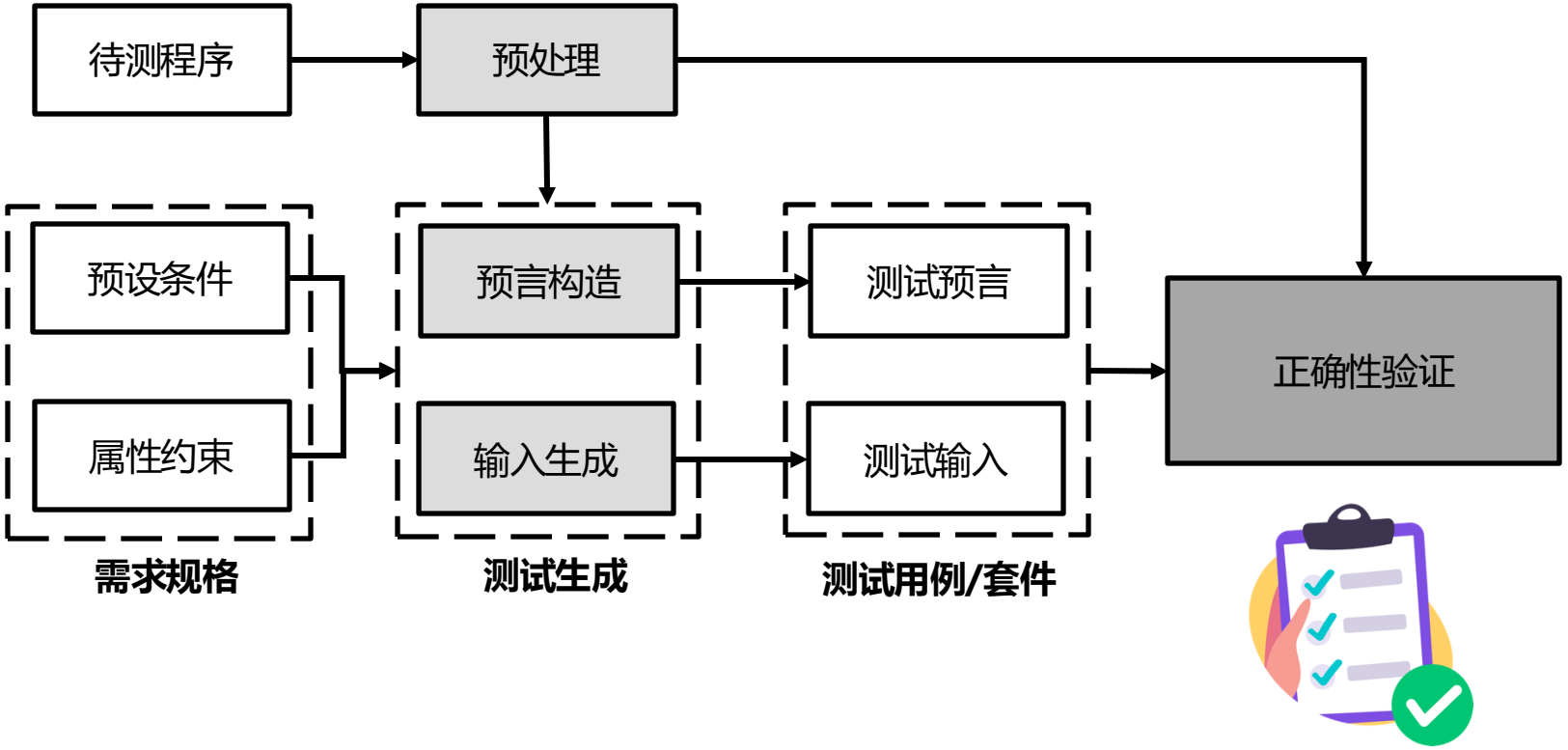
目录

01

预言问题

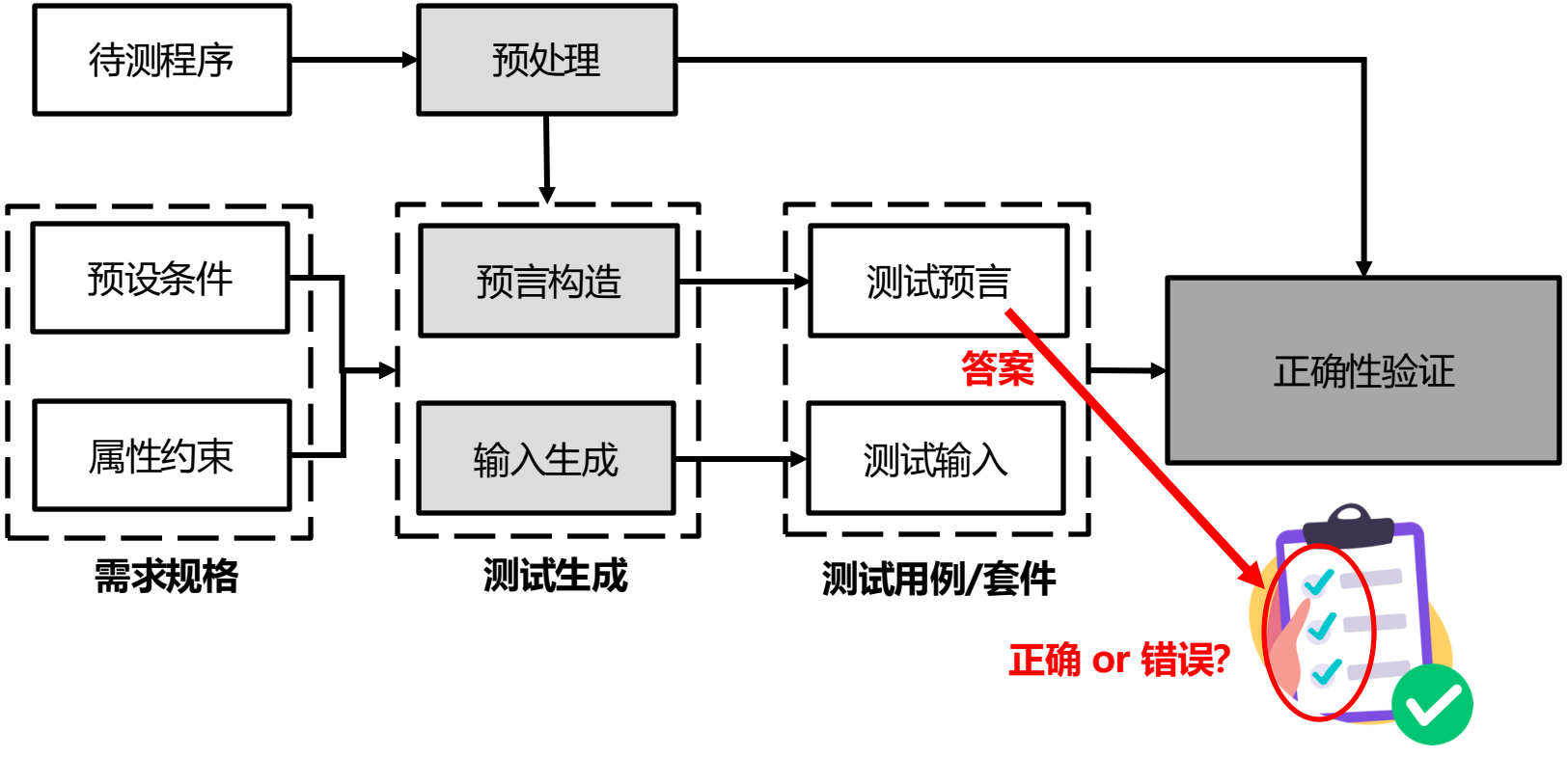
测试预言

- 测试预言是（自动化）软件测试中不可或缺的一部分



测试预言

- 测试预言是（自动化）软件测试中不可或缺的一部分



- **什么是测试预言（Test Oracle）？**
  - 是文档：体现被测单元的预期功能[1]（Intended Functionality）
    - 被测单元：模块、类、方法、函数、语句
  - 是机制：验证待测程序的行为是否符合预期[2]
  - 是程序：判定程序的执行是否违反了某种正确性政策[3]
  - 是约束：要求$P(i) = o_{exp}$为$true$
  - 是映射：$\omega: \mathbb{I} \times \mathbb{O} \rightarrow \{success, fail\}$

[1] Dinella E, Ryan G, Mytkowicz T, et al. Toga: A neural method for test oracle generation[C]//Proceedings of the 44th International Conference on Software Engineering. 2022: 2130-2141.
[2] Chen J, Bai Y, Hao D, et al. Supporting oracle construction via static analysis[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016: 178-189.
[3] Manès V J M, Han H S, Han C, et al. The art, science, and engineering of fuzzing: A survey[J]. IEEE Transactions on Software Engineering, 2019, 47(11): 2312-2331.

- **测试预言举例**

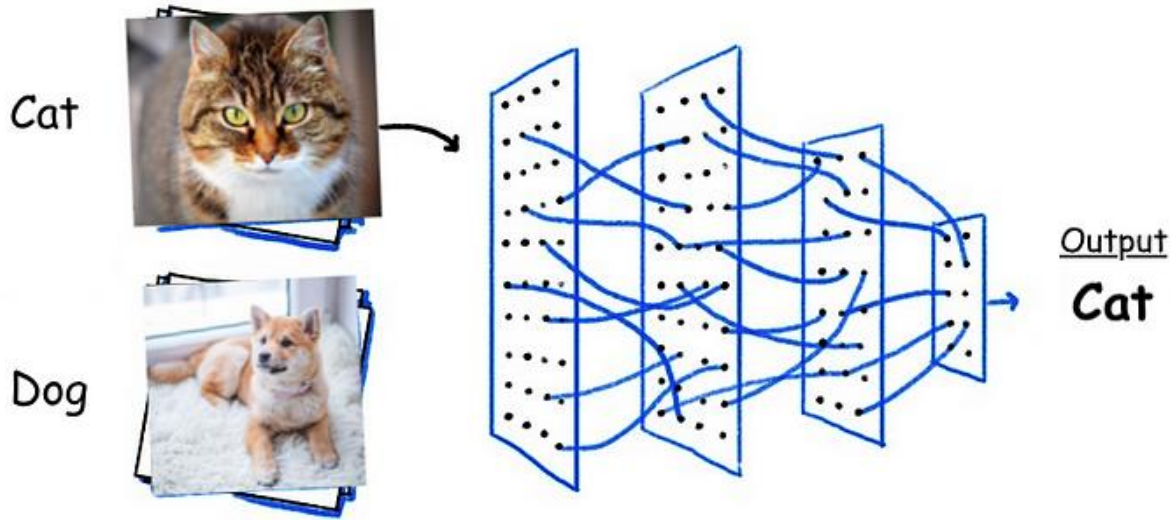
```
public void testPop() {
  Stack<int> s = new Stack<int>();
  int a = 2;

  s.push(a);
  s.pop();

  bool empty = s.isEmpty();
  assertTrue(empty);
}
```

```
public void testPop() {
 try {
  Stack<int> s = new Stack<int>();
  s.pop();

  Assert.fail(); //fail
 } catch (Exception e) {
  //pass
 }
}
```

单元测试中的测试断言[1]

[1] Dinella E, Ryan G, Mytkowicz T, et al. Toga: A neural method for test oracle generation[C]//Proceedings of the 44th International Conference on Software Engineering. 2022: 2130-2141.

- **测试预言举例**



DL模型测试中的Ground Truth

- **测试预言举例**

初始购物车

添加商品到购物车

商品添加成功

移动应用测试中的功能性预言

- 测试预言形式化定义
  - 给定一次测试$T = \{P, I\}$，其中P和I分别表示本次测试的待测程序和测试输入。程序P对应输入I的输出表示为$O = P(I)$；
  - 给定预期输出O^*，则**测试预言**可以表示为约束$\tau: O \cong O^*$，其中$\cong$表示**符合关系**；
  - 当τ满足时，表明输出O符合预期输出O^*，本次测试通过；反之则表明O不符合O^*，本次测试未通过。

- 测试预言分类：隐式（Implicit）、显式（Explicit）[1]
  - **隐式预言**：用于检测**显著缺陷**（Obvious Bug）的测试预言。例如，预期外的程序崩溃（Crash）就是一种典型的显著缺陷。
  - **显式预言**：根据软件的预期功能提取得到的预言，一般用于检测功能性缺陷（Functional Bug）。

[1] Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: A survey[J]. IEEE transactions on software engineering, 2014, 41(5): 507-525.

- 测试预言分类：隐式（Implicit）、显式（Explicit）[1]

```
public void testPop() {
  Stack<int> s = new Stack<int>();
  int a = 2;

  s.push(a);
  s.pop();

  bool empty = s.isEmpty();
  assertTrue(empty);
}
```

```
public void testPop() {
 try {
  Stack<int> s = new Stack<int>();
  s.pop();

  Assert.fail(); //fail
 } catch (Exception e) {
  //pass
 }
}
```

单元测试中的显式预言

[1] Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: A survey[J]. IEEE transactions on software engineering, 2014, 41(5): 507-525.

- 测试预言分类：隐式（Implicit）、显式（Explicit）[1]

```
1  #include <string.h>
2  int main(int argc, char** argv) {
3      if (argc > 1 && argv[1][0] == 'H')
4          if (argc > 1 && argv[1][1] == 'I') {
5              char buffer[100];
6              strcpy(buffer, argv[1]); // 模糊目标
7          }
8      return 0;
9  }
```

可能触发能够导致程序崩溃的缓冲区溢缺陷

模糊测试中的隐式预言

[1] Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: A survey[J]. IEEE transactions on software engineering, 2014, 41(5): 507-525.

测试预言

- 测试预言分类：隐式（Implicit）、显式（Explicit）[1]

初始购物车

添加商品到购物车

商品添加成功

移动应用测试中的显式预言

移动应用测试中的隐式预言

[1] Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: A survey[J]. IEEE transactions on software engineering, 2014, 41(5): 507-525.

- **什么是预言问题（Oracle Problem）？**
  - **预言问题**：给定系统的输入，如何找到能够正确辨别出符合期望的正确行为与发现潜在的不正确行为测试预言的挑战性难题。[1]
  - **预言问题具有挑战性的原因**[2]
    - **缺少清晰的规格**：文档通常用自然语言编写，而自然语言存在二义性
    - **软件的动态特征**：网络问题导致的问题、线程调度产生的并发问题
    - **优先的测试资源**：测试预言空间和测试输入空间一样无法穷尽

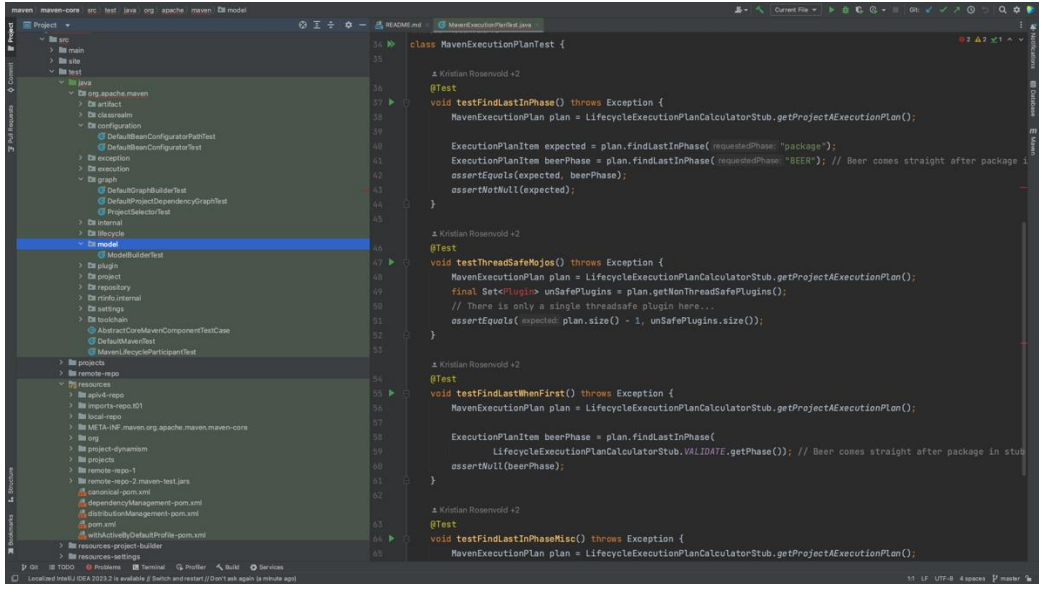
[1] Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: A survey[J]. IEEE transactions on software engineering, 2014, 41(5): 507-525.
[2] Yu B, Mang Q, Guo Q, et al. Retromorphic Testing: A New Approach to the Test Oracle Problem[J]. arXiv preprint arXiv:2310.06433, 2023.

预言问题

- **预言问题现状：**复杂的预言难以构建，容易构建的预言效果差

预言问题

- **预言问题现状：**复杂的预言难以构建，容易构建的预言效果差



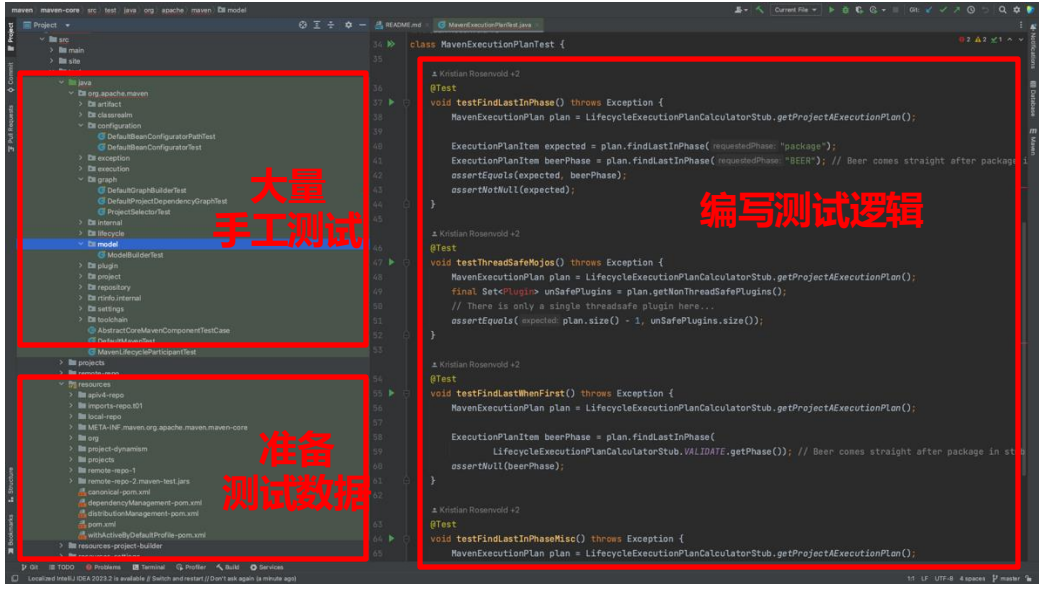
Apache Maven项目下的手工单元测试用例

```java
import ...

@RunWith(JQF.class)
public class ModelReaderTest {

    @Fuzz
    public void testWithInputStream(InputStream in) {
        ModelReader reader = new DefaultModelReader();
        try {
            Model model = reader.read(in,  map: null);
            Assert.assertNotNull(model);
        } catch (IOException e) {
            Assume.assumeNoException(e);
        }
    }

    @Fuzz
    public void testWithGenerator(@From(XmlDocumentGenerator.class)
                                  @Size(min = 0, max = 10)
                                  @Dictionary("dictionaries/maven-model.dict") Document dom) {
        testWithInputStream(XMLDocumentUtils.documentToInputStream(dom));
    }

    @Fuzz
    public void debugWithGenerator(@From(XmlDocumentGenerator.class)
                                   @Size(min = 0, max = 10)
                                   @Dictionary("dictionaries/maven-model.dict") Document dom) {
        System.out.println(XMLDocumentUtils.documentToString(dom));
        testWithGenerator(dom);
    }
```

JQF中提供的Maven模糊驱动

预言问题

- **预言问题现状：**复杂的预言难以构建，容易构建的预言效果差



Apache Maven项目下的手工单元测试用例

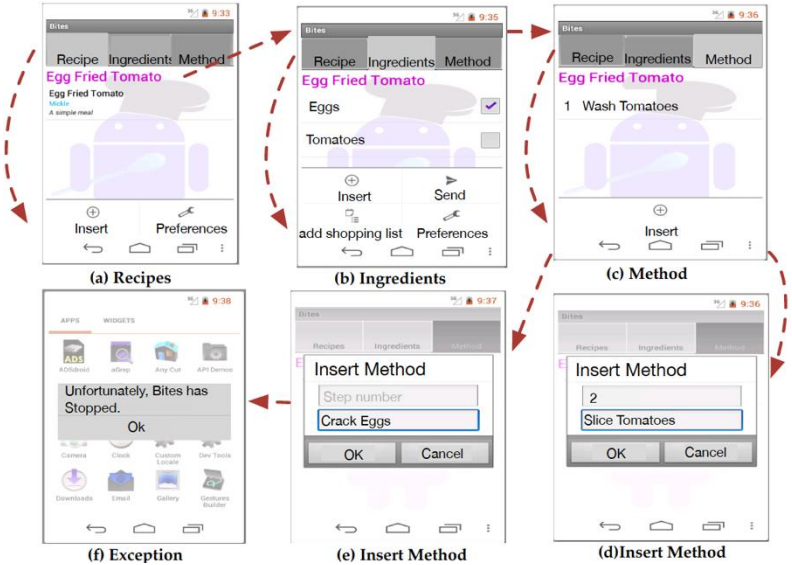
```java
import ...

@RunWith(JQF.class)
public class ModelReaderTest {

    @Fuzz
    public void testWithInputStream(InputStream in) {
        ModelReader reader = new DefaultModelReader();
        try {
            Model model = reader.read(in, map: null);
            Assert.assertNotNull(model);
        } catch (IOException e) {
            Assume.assumeNoException(e);
        }
    }

    @Fuzz
    public void testWithGenerator(@From(XmlDocumentGenerator.class)
                                  @Size(min = 0, max = 10)
                                  @Dictionary("dictionaries/maven-model.dict") Document dom) {
        testWithInputStream(XMLDocumentUtils.documentToInputStream(dom));
    }

    @Fuzz
    public void debugWithGenerator(@From(XmlDocumentGenerator.class)
                                   @Size(min = 0, max = 10)
                                   @Dictionary("dictionaries/maven-model.dict") Document dom) {
        System.out.println(XMLDocumentUtils.documentToString(dom));
        testWithGenerator(dom);
    }
```

以read方法
不抛异常作为预言

JQF中提供的Maven模糊驱动

预言问题

- **预言问题现状：**复杂的预言难以构建，容易构建的预言效果差



复杂预言：长操作序列导致的意外崩溃[1]

简单预言：APP不应随意崩溃

[1] Su T, Meng G, Chen Y, et al. Guided, stochastic model-based GUI testing of Android apps[C]//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017: 245-256.

- **预言问题现状：**复杂的预言难以构建，容易构建的预言效果差

初始分类为熊猫， 置信度60%

+

添加对抗扰动

=

将对抗样本错误地分类为长臂猿， 置信度99%

- **预言问题解决方案**
  - 利用静态程序分析手段自动化构建测试预言[1]
  - 从文本文档/注释中自动提取[2]（Specification Mining）
  - 设计形式化的文档语言[3]：体系结构设计语言（ADL）
  - **使用不完全/有偏测试预言（Partial Test Oracle）**
    - 蜕变测试（Metamorphic Testing）
    - 差分测试（Differential Testing）

[1] Chen J, Bai Y, Hao D, et al. Supporting oracle construction via static analysis[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016: 178-189.
[2] Le T D B, Lo D. Deep specification mining[C]//Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2018: 106-117.
[3] Medvidovic N, Taylor R N. A classification and comparison framework for software architecture description languages[J]. IEEE Transactions on software engineering, 2000, 26(1): 70-93.

02

蜕变测试

蜕变测试

- **蜕变测试**：一种测试用例生成的新思路[1]
  - 1998年，由Tsong Yueh Chen提出[1]
  - **动机**：应对（自动化）软件测试中的两大难题[2]
    - **可靠测试集问题**（Reliable Test Set Problem）
    - **预言问题**（Oracle Problem）

Tsong Yueh Chen

Professor of Software Engineering, Swinburne University of Technology
Verified email at swin.edu.au

Software Testing Software Analysis Debugging

[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

蜕变测试

- **蜕变测试**：一种测试用例生成的新思路[1]
  - 1998年，由Tsong Yueh Chen提出[1]
  - **动机**：应对（自动化）软件测试中的两大难题[2]
    - **可靠测试集问题**（Reliable Test Set Problem）→ 测试生成/扩增
    - **预言问题**（Oracle Problem）→ 正确性验证

Tsong Yueh Chen

Professor of Software Engineering, Swinburne University of Technology
Verified email at swin.edu.au

Software Testing Software Analysis Debugging

[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- **蜕变测试**：一种测试用例生成的新思路[1]
  - **两类测试用例**
    - 成功测试用例：测试执行符合预期
    - 失败测试用例：测试执行违反预期，发现了某种缺陷
  - **基本假设**[2]：虽然不能通过成功测试用例排除程序存在缺陷的可能，但仍然可以利用多个相关的成功测试用例所展现出的测试输入和预期输出之间的关系来协助后续测试。

[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- **蜕变测试**：一种测试用例生成的新思路[1]
  - **两类测试用例**
    - 成功测试用例：测试执行符合预期
    - 失败测试用例：测试执行违反预期，发现了某种缺陷
  - **基本假设**[2]：①虽然不能通过成功测试用例排除程序存在缺陷的可能，但仍然可以利用多个相关的成功测试用例所展现出的测试输入和预期输出之间的关系来协助后续测试。

[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

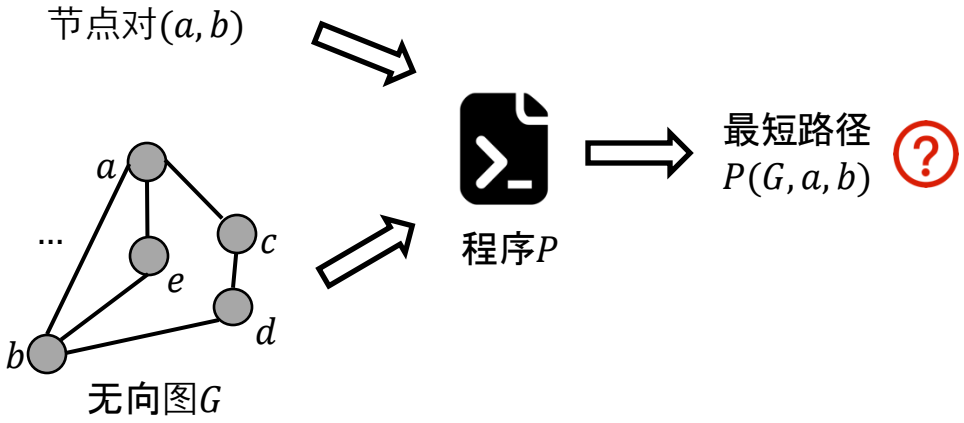
- **蜕变测试**：一种测试用例生成的新思路[1]
  - **两类测试用例**
    - 成功测试用例：测试执行符合预期
    - 失败测试用例：测试执行违反预期，发现了某种缺陷
  - **基本假设**[2]：虽然不能通过成功测试用例排除程序存在缺陷的可能，②但仍然可以利用多个相关的成功测试用例所展现出的测试输入和预期输出之间的关系来协助后续测试。

[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- **蜕变测试**：一种测试用例生成的新思路[1]
  - **蜕变测试本质**：充分利用成功测试用例
    - 对**成功测试用例**表现出的**必要属性**[2]（Necessary Properties）的复用；
    - **必要属性**：从相关成功测试用例的集合和程序规格中提取出的、待测软件必须遵守的属性；
    - **属性的必要性**：违反必要属性的测试一定发现了某种软件缺陷。

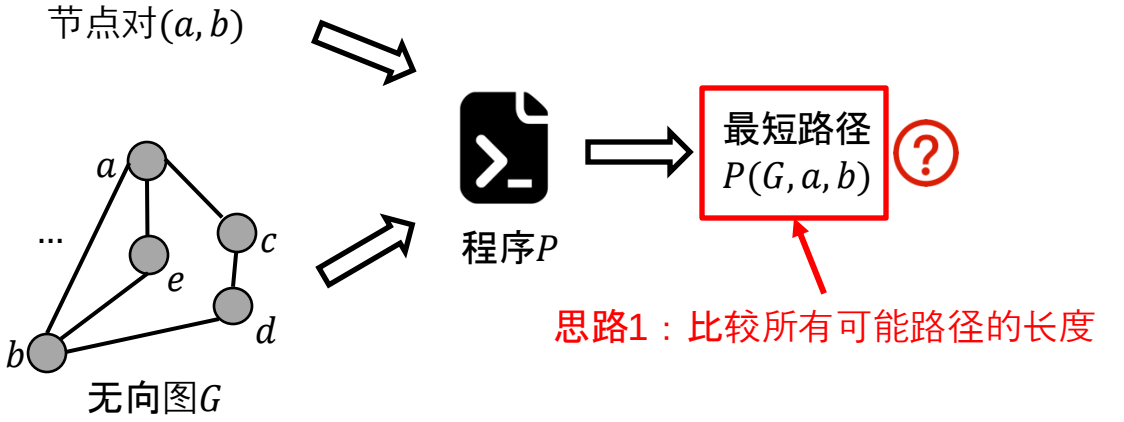
[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- **蜕变测试**：一种测试用例生成的新思路[1]
  - **举例：**假设f是一个用于计算无向图中最短路径的算法，它的输入是一个无向图G和两个节点a和b。现在有一个实现了f的程序P，如何有效验证程序P的正确性？



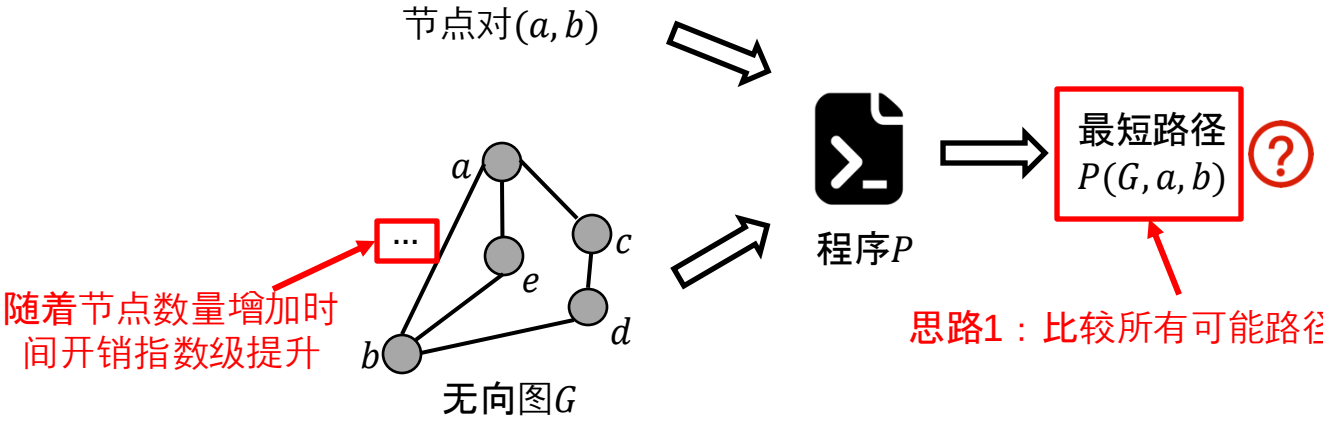
[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- **蜕变测试**：一种测试用例生成的新思路[1]
  - **举例**：假设f是一个用于计算无向图中最短路径的算法，它的输入是一个无向图G和两个节点a和b。现在有一个实现了f的程序P，如何有效验证程序P的正确性？



[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

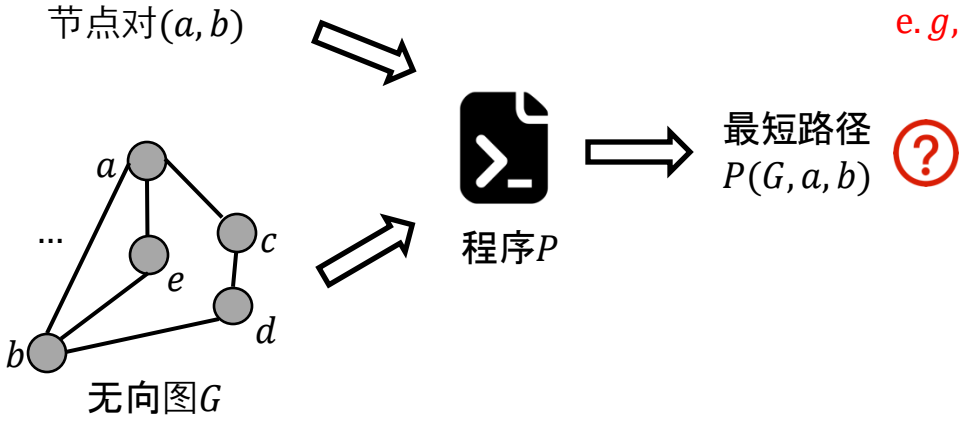
- **蜕变测试**：一种测试用例生成的新思路[1]
  - **举例**：假设f是一个用于计算无向图中最短路径的算法，它的输入是一个无向图G和两个节点a和b。现在有一个实现了f的程序P，如何有效验证程序P的正确性？



[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- **蜕变测试**：一种测试用例生成的新思路[1]
  - **举例**：假设f是一个用于计算无向图中最短路径的算法，它的输入是一个无向图G和两个节点a和b。现在有一个实现了f的程序P，如何有效验证程序P的正确性？

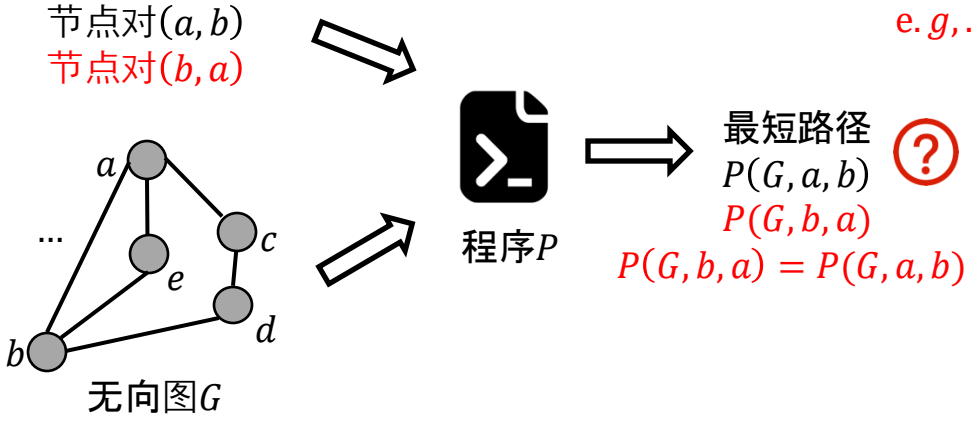
思路2：利用f的必要性质
e.g,. $|f(G,b,a)| = |f(G,a,b)|$



[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- **蜕变测试**：一种测试用例生成的新思路[1]
  - **举例**：假设f是一个用于计算无向图中最短路径的算法，它的输入是一个无向图G和两个节点a和b。现在有一个实现了f的程序P，如何有效验证程序P的正确性？

思路2：利用f的必要性质
e.g,. $|f(G,b,a)| = |f(G,a,b)|$



[1] Chen T Y, Cheungx S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases [J]. 1998
[2] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- 三要素[1]：**蜕变关系**、蜕变集合、蜕变测试过程
  - **蜕变关系（MR, Metamorphic Relation）**：一组待测算法/功能的必要属性，蜕变测试的核心
  - **蜕变集合（Metamorphic Group）**：由表达了蜕变关系的一组测试输入组成的集合
  - **蜕变测试过程**：应用蜕变关系和蜕变集合进行测试的一般流程

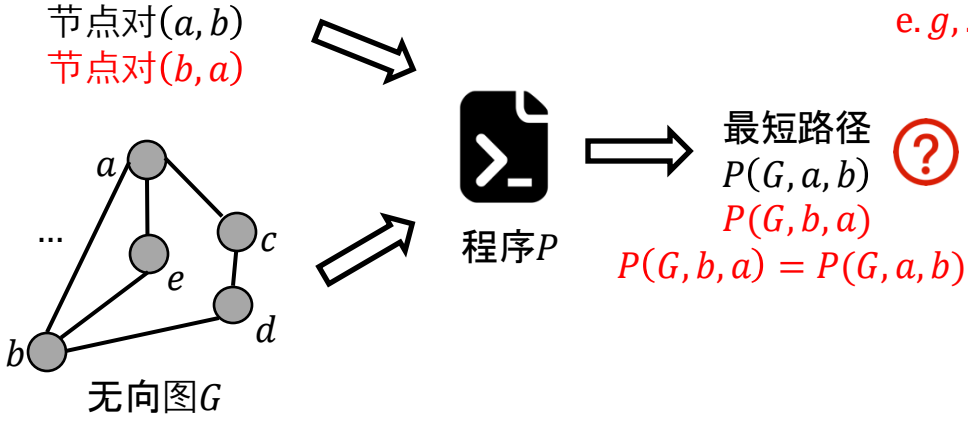
[1] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- 定义1：**蜕变关系**
  - 假设f是待测功能/算法*，给定一个长度为$n(n \geq 2)$的测试输入的序列$\{x_1, x_2, \ldots, x_n\}$和其对应的输出$\{f(x_1), f(x_2), \ldots, f(x_n)\}$，一个蜕变关系$\mathcal{R}$为这对输入输出序列上一个必要属性，记作$\mathcal{R} \subseteq X^n \times Y^n$。其中$X^n$和$Y^n$表示n维输入和输出空间的的笛卡尔积。
  - 简单起见，可以用$\mathcal{R}(x_1, x_2, \ldots, x_n, f(x_1), f(x_2), \ldots, f(x_n))$表示$< x_1, x_2, \ldots, x_n, f(x_1), f(x_2), \ldots, f(x_n) > \in \mathcal{R}$

* 本课件的后续部分将简称“待测功能/算法”为待测算法

- 定义1：**蜕变关系**
  - **思路2中的蜕变关系**：当起始节点和目标节点a和b交换后，算法f得出的最短路径的长度应该不变，即$|f(G,b,a)| = |f(G,a,b)|$。其中，输入$x = (G,a,b)$，对应输出$f(G,a,b)$

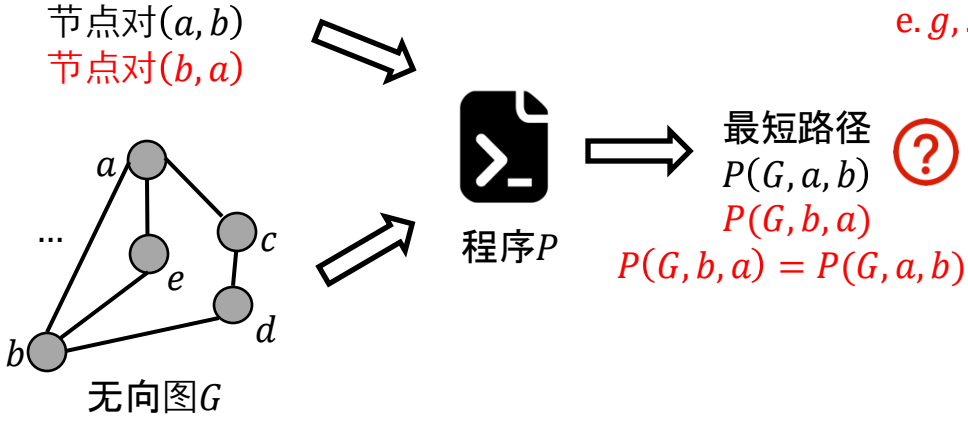
思路2：利用f的必要性质
e.g,. $|f(G,b,a)| = |f(G,a,b)|$



- 定义2：**源输入与后续输入**
  - 给定蜕变关系$\mathcal{R}(x_1, x_2, \dots, x_n, f(x_1), f(x_2), \dots, f(x_n))$，假设从第$j$个测试输入开始，后续输入$x_j(j = k+1, k+2, \dots, n)$是在前$k$个输入输出序列$\{x_1, x_2, \dots, x_k, f(x_1), f(x_2), \dots, f(x_k)\}$的基础上、根据蜕变关系$\mathcal{R}$构建的，则对于任意$i = 1,2, \dots, k$，我们称$x_i$为**源输入（Source Input）**；对于任意$j = k+1, k+2, \dots, n$，我们称x_j为**后续输入（Follow-up Input）**。
  - **延伸**：对于任意蜕变关系$\mathcal{R}$，在前k个源测试输入给定的情况下，我们就可以根据蜕变关系$\mathcal{R}$构建后续输入 → 可用作测试生成的依据

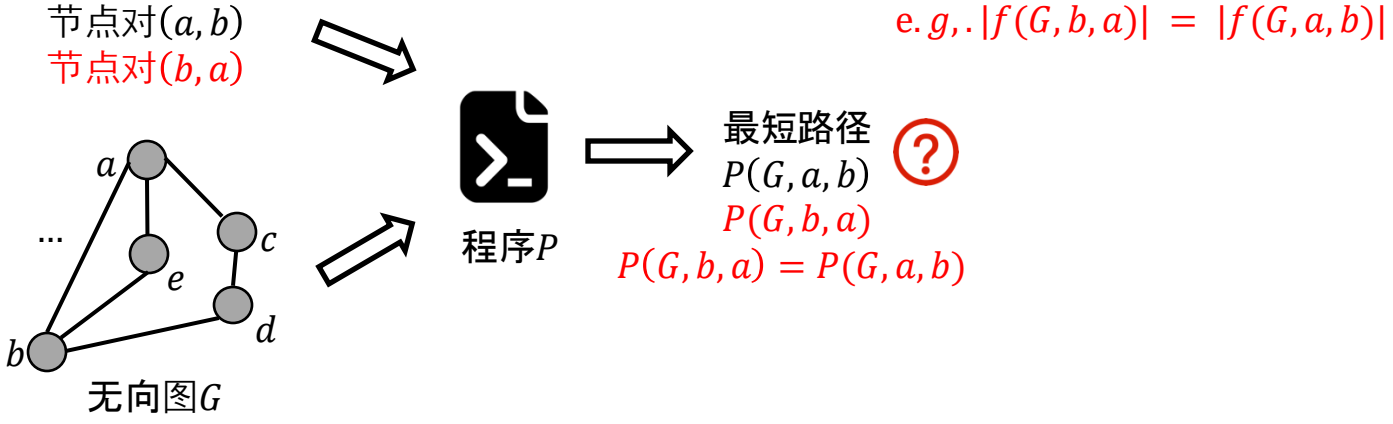
- 定义2：**源输入与后续输入**
  - **思路2中的源输入与后续输入**：(G, a, b) 是在测试目标中给定的源输入，(G, b, a)则是根据蜕变关系$|f(G,b,a)| = |f(G,a,b)|$和源输入生成的后续输入

思路2：利用f的必要性质
e.g,.$|f(G,b,a)| = |f(G,a,b)|$



- 定义3：**蜕变集合**
  - 蜕变集合又称蜕变输入集合，是由表达了蜕变关系的测试输入构成的集合。给定蜕变关系$\mathcal{R}(x_1, x_2, \dots, x_n, f(x_1), f(x_2), \dots, f(x_n))$，则表达了$\mathcal{R}$的输入序列$\{x_1, x_2, \dots, x_n\}$就构成了$\mathcal{R}$的蜕变集合。
  - 结合定义2，可以看出蜕变集合是由关于蜕变关系$\mathcal{R}$的源输入序列$\{x_1, x_2, \dots, x_k\}$和后续输入序列$\{x_{k+1}, x_{k+2}, \dots, x_n\}$一同构成的集合

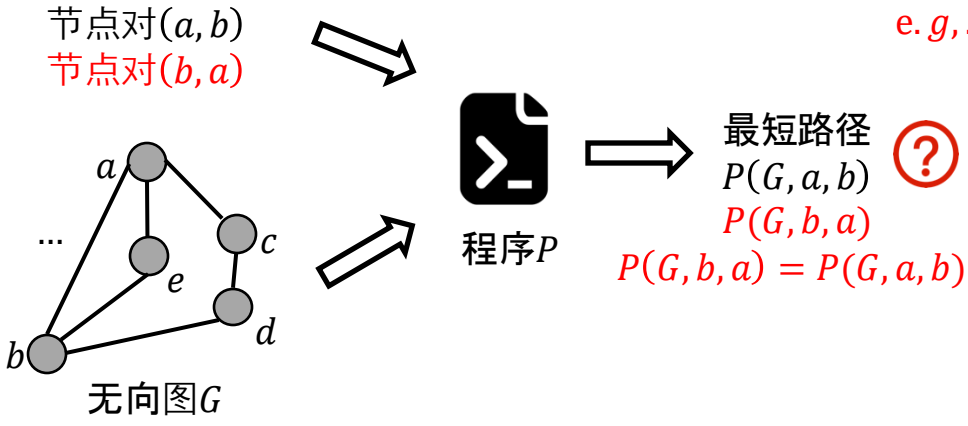
- 定义3：**蜕变集合**
  - **思路2中的蜕变集合**：$\{(G,a,b),(G,b,a)\}$



- 定义4：**蜕变测试过程**
  - 同时也是**蜕变测试**的定义。假设程序P是待测算法f的一个实现，给定蜕变关系$\mathcal{R}(x_1, x_2, \dots, x_n, f(x_1), f(x_2), \dots, f(x_n))$，依托$\mathcal{R}$的蜕变测试流程可以归纳为以下几步：
    - **构造待验证的蜕变关系$\mathcal{R}'$**：将$\mathcal{R}$中的待测算法f替换为程序P
    - **构造蜕变集合**：在程序P上执行给定源测试输入序列$\{x_1, x_2, \dots, x_k\}$以获得源输出序列$\{P(x_1), P(x_2), \dots, P(x_k)\}$；根据$\mathcal{R}'$构造并执行后续输入序列$\{x_{k+1}, x_{k+2}, \dots, x_n\}$以获得后续输出序列$\{P(x_{k+1}), P(x_{k+2}), \dots, (x_n)\}$
    - **检查执行结果是否满足$\mathcal{R}'$**：如果$\mathcal{R}'$没有被满足，则说明程序P存在缺陷

- 定义4：**蜕变测试过程**
  - **思路2中描述的蜕变测试**：构造$\mathcal{R}'$: $|P(G,b,a) = P(G,a,b)|$ → 构造蜕变集合$\{(G,a,b),(G,b,a)\}$ → 验证是否满足$|P(G,b,a) = P(G,a,b)|$

思路2：利用f的必要性质
e.g,. $|f(G,b,a)| = |f(G,a,b)|$



课堂练习

- 尝试为以下场景设计蜕变测试过程
  - **场景一**：待测算法f用于**计算正弦值**，程序P是f的实现
  - **场景二**：待测算法f用于**图像分类**，模型P是f的实现
  - **场景三**：待测算法f用于**编译优化**，程序P是f的实现
    - **编译优化**：在不影响程序运行结果为前提下提升程序的某种性能

- **场景一**：待测算法f用于**计算正弦值**，程序P是f的实现
  - **确定输入输出**：输入为角度值x，输出为计算得到的正弦值
  - **确定蜕变关系$\mathcal{R}$**：正弦函数为奇函数，$-f(x) = f(-x)$
  - **构建待验证的蜕变关系$\mathcal{R}'$**：$-P(x) = P(-x)$
  - **构造蜕变集合**：$\{30,60,90,-30,-60,-90\}$，其中，源输出序列为$\{P(30),P(60),P(90)\}$，后续输出序列为$\{P(-30),P(-60),P(-90)\}$
  - **检查执行结果是否满足$\mathcal{R}'$**：依次检验$-P(30) = P(-30)$，$-P(60) = P(-60)$，和$-P(90) = P(-90)$

- **场景二**：待测算法f用于**图像分类**，模型P是f的实现
  - **确定输入输出**：输入为二元组$x = (p, r)$，其中p为待分类图片，r为图片的旋转角度；输出为图像分类结果
  - **确定蜕变关系$\mathcal{R}$**：将图片旋转90度后的分类结果不变，$f(p, r) = f(p, r + 90)$
  - **构建待验证的蜕变关系$\mathcal{R}'$**：$P(p, r) = P(p, r + 90)$
  - **构造蜕变集合**：$\{(p_1, 0), (p_2, 0), (p_1, 90), (p_2, 90)\}$
  - **检查执行结果是否满足$\mathcal{R}'$**：依次检验$P(p_1, 0) = P(p_1, 90)$和$P(p_2, 0) = P(p_2, 90)$

- **场景三**：待测算法f用于**编译优化**，程序P是f的实现
  - **确定输入输出**：输入为二元组$x = (p, o)$，其中p表示待编译的程序，o表示是否优化，优化为1，不优化为0；输出为编译结果
  - **确定蜕变关系$\mathcal{R}$**：编译优化不影响待编译程序的执行结果，$f(p, false) = f(p, true)$
  - **构建待验证的蜕变关系$\mathcal{R}'$**： $P(p, false) = P(p, true)$
  - **构造蜕变集合**：$\{(p_1, false), (p_2, false), (p_1, true), (p_2, true)\}$
  - **检查执行结果是否满足$\mathcal{R}'$**：依次检验$P(p_1, false) = P(p_1, true)$和$P(p_2, false) = P(p_2, true)$

- **场景三**：待测算法f用于**编译优化**，程序P是f的实现
  - PLDI'14-Orion[1]：去除未执行的代码不会影响编译结果，即$\mathcal{R}$: $f(p,i) = f(p^-,i)$，其中p为待编译程序，p^-为移除未执行代码后的p，i为程序p或p^-的输入。
  - Orion框架：两个阶段，准备阶段和验证阶段
    - **准备阶段**：生成大量测试输入I执行待编译程序p，识别程序中未执行的代码片段，移除未执行的片段得到p^-
    - **验证阶段**：对于所有$i \in I$，检查是否满足$\mathcal{R}'$: $P(p,i) = P(p^-,i)$。

[1] Le V, Afshari M, Su Z. Compiler validation via equivalence modulo inputs[C]//Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2014: 216-226.

- 蜕变测试中的四大误解[1]
  - **误解一**：所有的必要属性都是蜕变关系
  - **误解二**：所有的蜕变关系都能够划分成**输入端**（Input-only）和**输出端**（Output-only）的两个子关系
  - **误解三**：所有的蜕变关系都是等式关系
  - **误解四**：蜕变测试只能应用在测试预言缺失的场景下

[1] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities[J]. ACM Computing Surveys (CSUR), 2018, 51(1): 1-27.

- **误解一**：所有的必要属性都是蜕变关系
  - **误解修正**：并非所有的必要属性都是蜕变关系。蜕变关系是和待测算法的**多个输入（Multiple Inputs）**以及这些输入对应的**期望输出**相关的**必要属性** → 蜕变关系应当和多个输入实例相关
  - **举例**：$-1 \leq \sin(x) \leq 1$是一个必要属性，但不适合作为蜕变关系，因为该属性仅涉及**一个**输入实例 → $-1 \leq \sin(x_1) \leq 1$
  - **使用非蜕变关系的必要属性充当测试预言**：断言检查[1]、多版本编程[2]、差分测试[3]

[1] Rosenblum D S. A practical approach to programming with assertions[J]. IEEE transactions on Software Engineering, 1995, 21(1): 19-31.
[2] Manolache L I, Kourie D G. Software testing using model programs[J]. Software: Practice and Experience, 2001, 31(13): 1211-1236.
[3] Elbaum S, Chin H N, Dwyer M B, et al. Carving differential unit test cases from system test cases[C]//Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. 2006: 253-264.

- **误解二**：所有的蜕变关系能够划分成输出端和输入端两个子关系
  - **误解修正**：蜕变关系不一定能够划分成输出端和输入端两个子关系
  - **能够划分成输入和输出端子关系的蜕变关系举例**
    - $\mathcal{R}$: 求无向图中两点a和b间最短路径的算法，交换开始和结束节点不影响算法的结果
    - 输入端子关系$\mathcal{R}_{in}$:交换开始节点a和结束节点b
    - 输出端子关系$\mathcal{R}_{out}$:输出的结果不变，即$|P(G,b,a) = P(G,a,b)|$

- **误解二**：所有的蜕变关系能够划分成输出端和输入端两个子关系
  - **误解修正**：蜕变关系不一定能够划分成输出端和输入端两个子关系
  - **不能划分成输入和输出端子关系的蜕变关系举例**
    - $\mathcal{R}'$: $|P(G,a,c)| + |P(G,c,b)| = |P(G,a,b)|$，其中c是存在于a和b最短路径上的一点
    - 构建蜕变集合：给定源输入(G,a,b)能够构建后续测试输入(G,a,c)和(G,c,b)，蜕变集合为$\{(G,a,b),(G,a,c),(G,c,b)\}$ → 缺少输出端关系

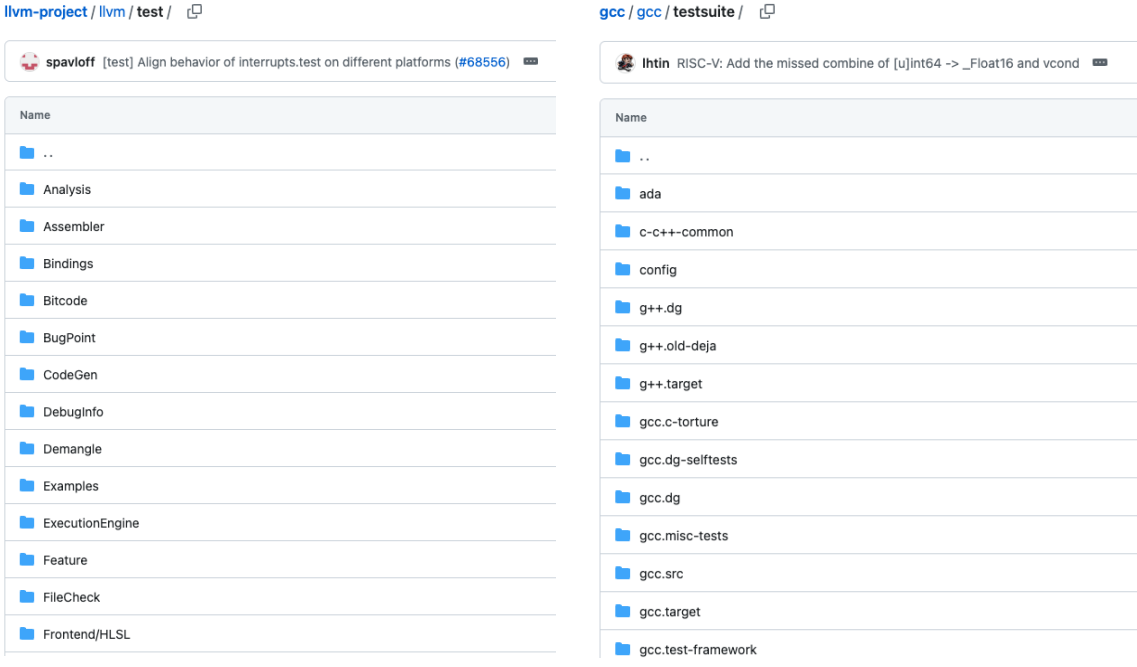
- **误解三**：所有的蜕变关系都是等式关系
  - **误解修正**：蜕变关系可以是非等式关系
  - **非等式蜕变关系举例**：假设q是一个数据库访问指令，能够根据条件表达式$c_1 \vee c_2 \vee \ldots \vee c_n$从数据库中提取数据。根据$q$的描述不难发现，如果将条件表达式中的某个谓词$c_i (1 \leq i \leq n)$移除后再进行数据查询，则提取出的数据应当是移除前数据的子集，即满足下列关系

$$q(c_1 \vee c_{2 \vee} \quad \ldots \vee c_{i-1} \vee c_{i+1} \vee \ldots \vee c_n) \subseteq q(c_1 \vee c_2 \vee \ldots \vee c_n)$$

- **误解四**：蜕变测试只能应用在测试预言缺失的场景下
  - **误解修正**：蜕变测试在测试预言存在和不存在的情况下都可以应用
  - **主要场景**：利用蜕变测试进行测试生成 → 蜕变关系作为生成的依据
    - 深度模型的测试：机器翻译、图像分类
    - 编译器测试：LLVM、GCC、JVM

蜕变测试

- **误解四**：蜕变测试只能应用在测试预言缺失的场景下



蜕变测试从已经被充分测试的编译器项目中检测出大量未被发现的缺陷[1]

[1] Le V, Afshari M, Su Z. Compiler validation via equivalence modulo inputs[C]//Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2014: 216-226.

03

差分测试

差分测试

- **差分测试**：利用相似/竞品软件系统进行测试
  - **定义**：差分测试（Differential Testing）也称为差分模糊测试，是一种常用的软件测试技术，通过向一系列类似的应用程序 （或同一应用程序的不同实现）提供相同的输入，根据这些相似程序执行结果是否存在差异来判定是否检测到缺陷[1,2]。
  - 1998年，由William M. McKeeman提出[2]

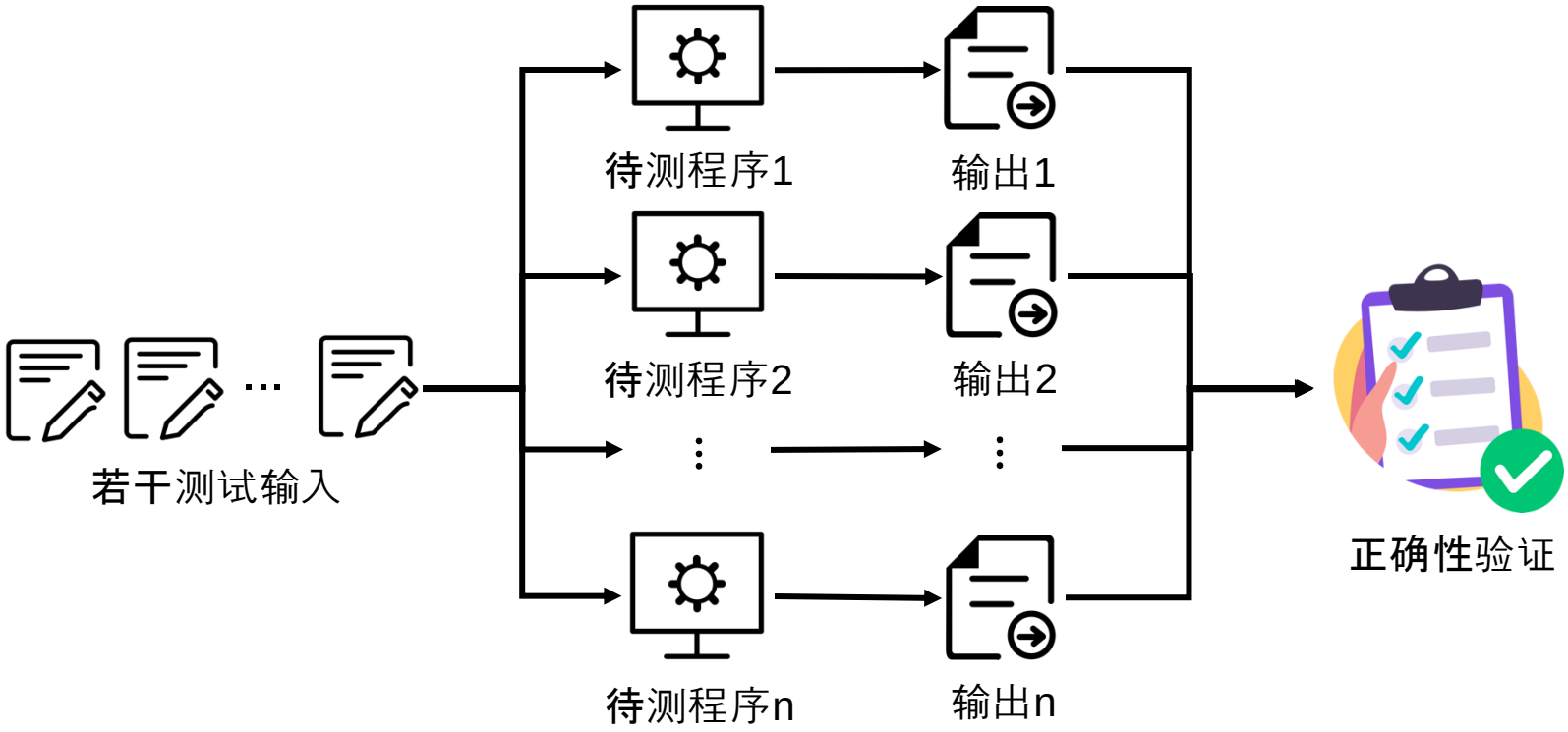
[1] 维基百科（差分测试）：https://en.wikipedia.org/wiki/Differential_testing#cite_note-1
[2] McKeeman W M. Differential testing for software[J]. Digital Technical Journal, 1998, 10(1): 100-107

差分测试流程

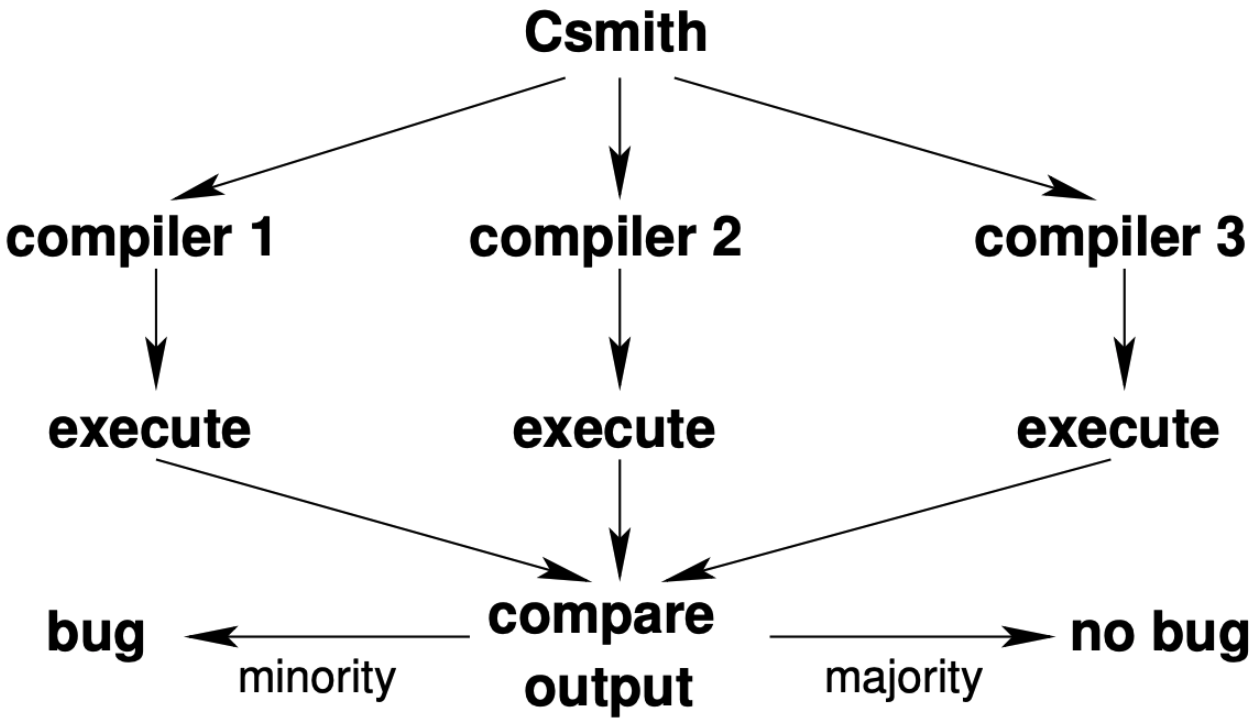
- **差分测试**：利用相似/竞品软件系统进行测试
  - **确定一组相似的待测程序**$\mathbb{P} = \{P_1, P_2, \dots, P_n\}(\mathrm{n} \geq 2)$。其中，对于任意$P_i, P_j \in \mathbb{P}(1 \leq i, j \leq n \wedge i \neq j)$，都满足$P_i \approx P_j$，$\approx$表示程序近似关系。
  - **生成测试输入和输出**。生成一组测试输入$\mathbb{I} = \{I_1, I_2, \dots, I_m\}(m \geq 1)$，在所有的待测程序上执行后得到测试输入集合$\mathcal{O} = \{\mathbb{O}_1, \mathbb{O}_2, \dots, \mathbb{O}_n\}$，其中$\mathbb{O}_k(1 \leq k \leq n)$表示所有相似程序执行第后得到的输出的集合，满足$\mathbb{O}_k = \{P_1(I_k), P_2(I_k), \dots, P_n(I_k)\}$
  - **正确性验证**。对于任意i, j, k，检查$\mathcal{D}$: $P_i(I_k) = P_j(I_k)$是否成立；若存在i, j, k不满足$\mathcal{D}$，则说明P_i和P_j**至少有一个程序**存在缺陷。

差分测试流程

- **差分测试**：利用相似/竞品软件系统进行测试



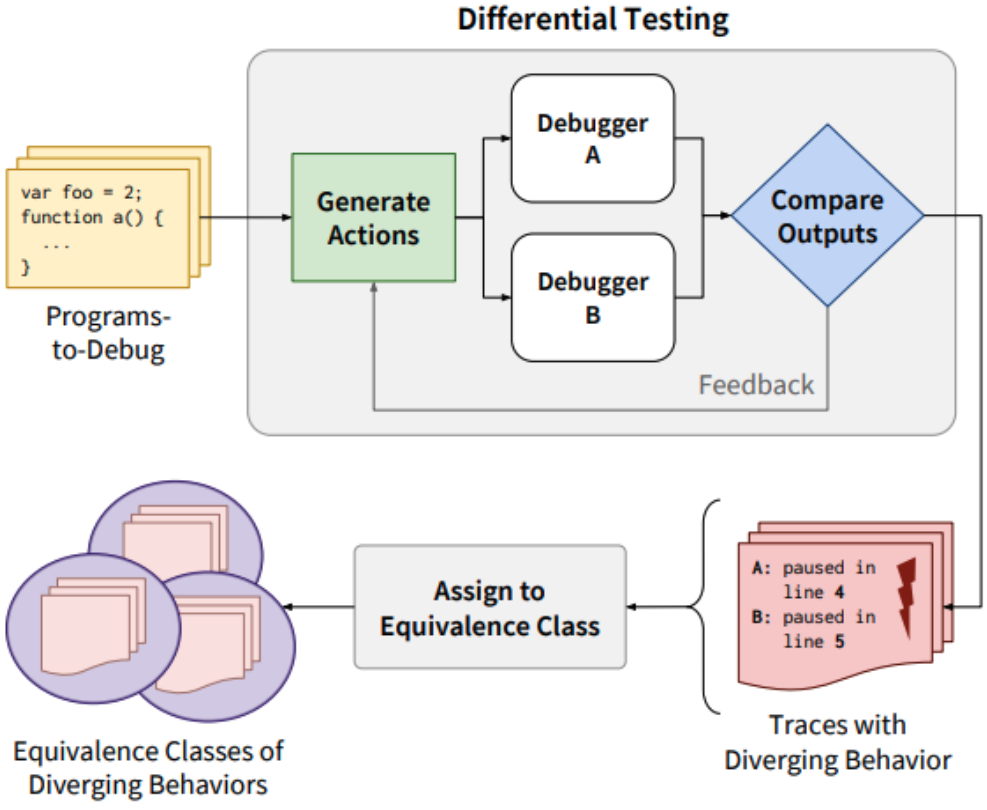
差分测试应用



针对C编译器的差分测试[1]

[1] Yang X, Chen Y, Eide E, et al. Finding and understanding bugs in C compilers[C]//Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation. 2011: 283-294.

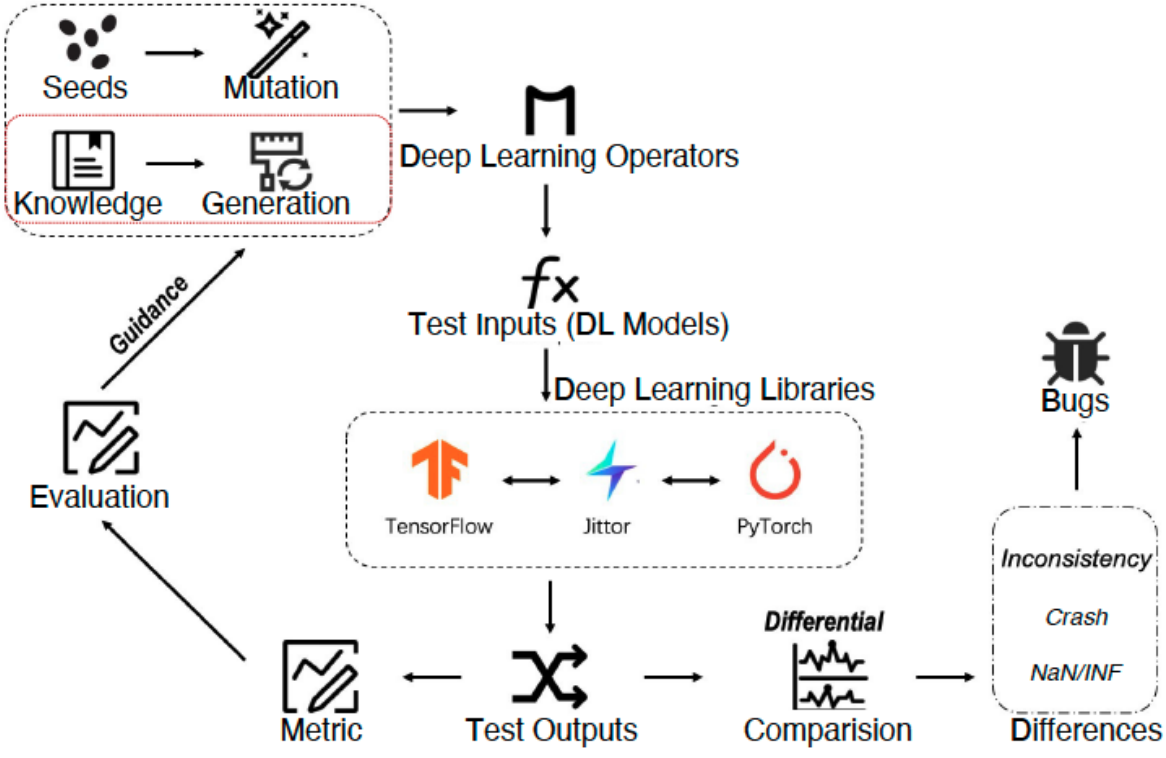
差分测试应用



针对调试器（Debugger）的差分测试[1]

[1] Lehmann D, Pradel M. Feedback-directed differential testing of interactive debuggers[C]//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018: 610-620.

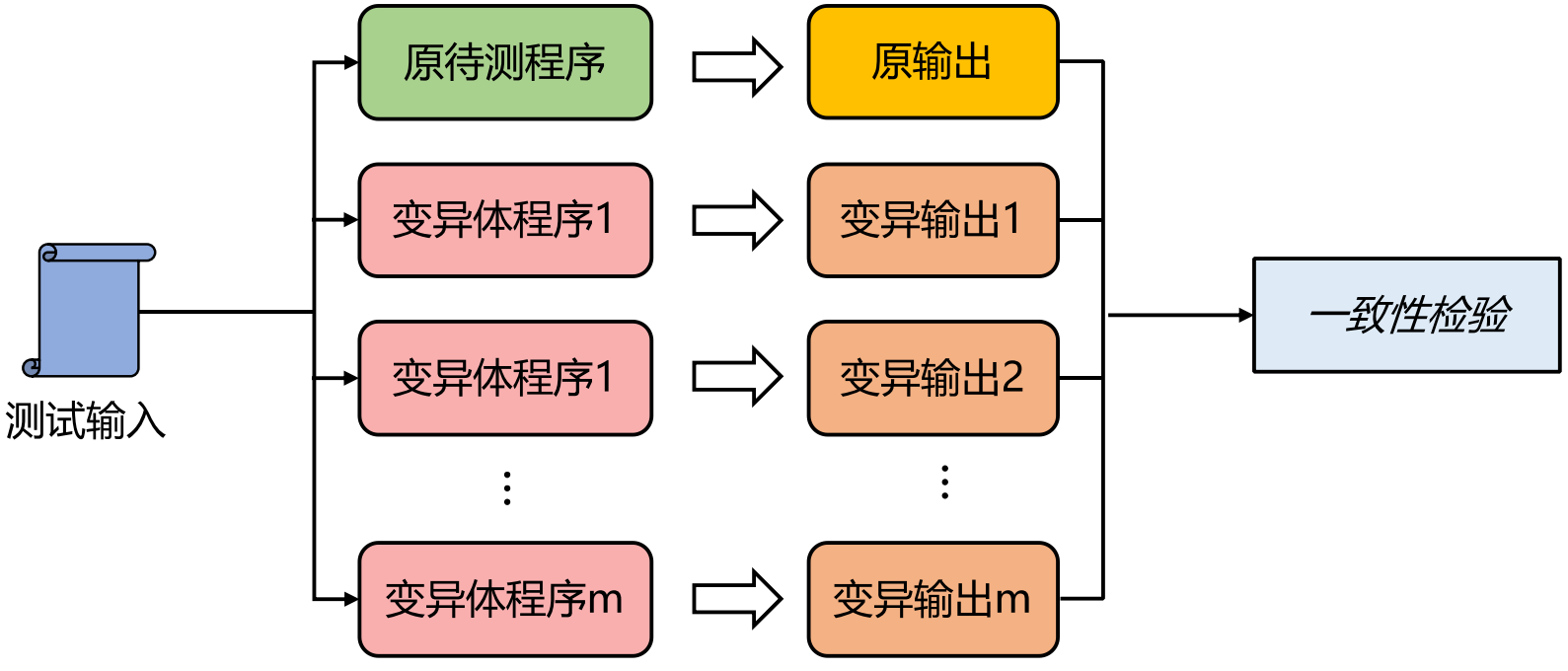
差分测试应用



针对深度学习框架的差分测试[1]

[1] Liu J, Huang Y, Wang Z, et al. Generation-Based Differential Fuzzing for Deep Learning Libraries[J]. ACM Transactions on Software Engineering and Methodology, 2023.

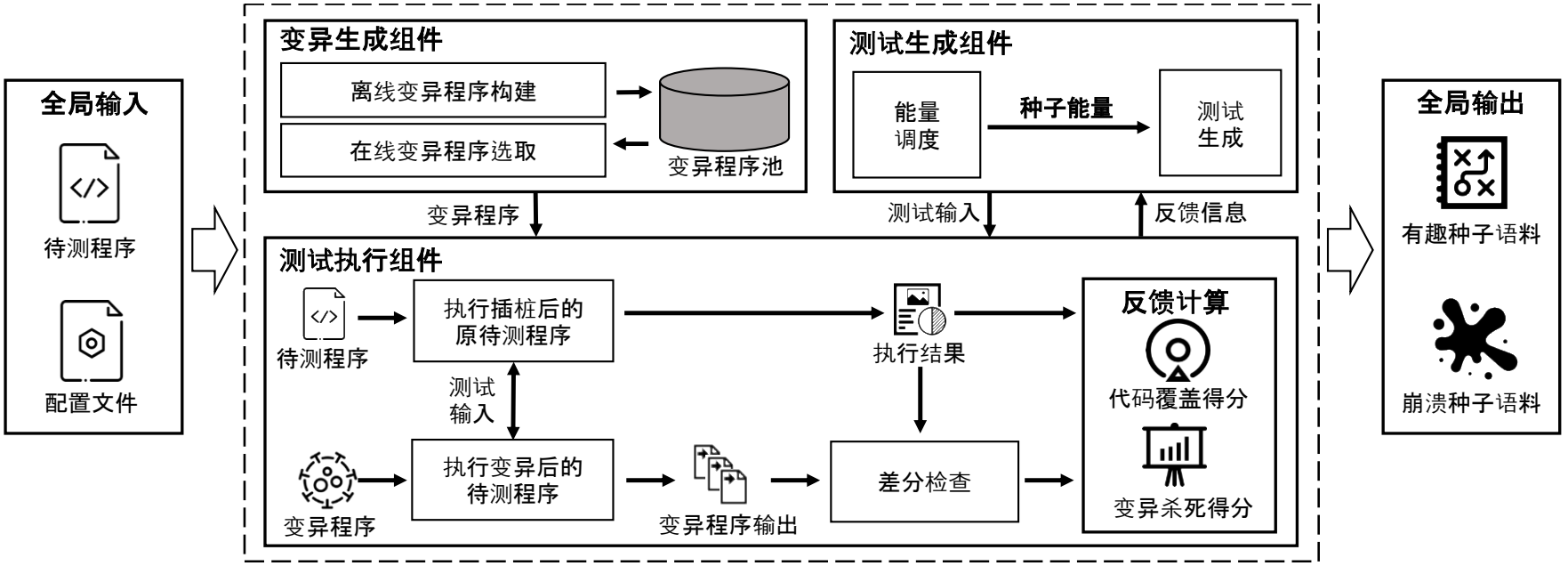
差分测试应用



MuFuzz: 模糊测试+变异差分

[1] Qian R, Zhang Q, Fang C, et al. Investigating coverage guided fuzzing with mutation testing[C]//Proceedings of the 13th Asia-Pacific Symposium on Internetware. 2022: 272-281.

差分测试应用



MuFuzz: 模糊测试+变异差分

[1] Qian R, Zhang Q, Fang C, et al. Investigating coverage guided fuzzing with mutation testing[C]//Proceedings of the 13th Asia-Pacific Symposium on Internetware. 2022: 272-281.

04

总结

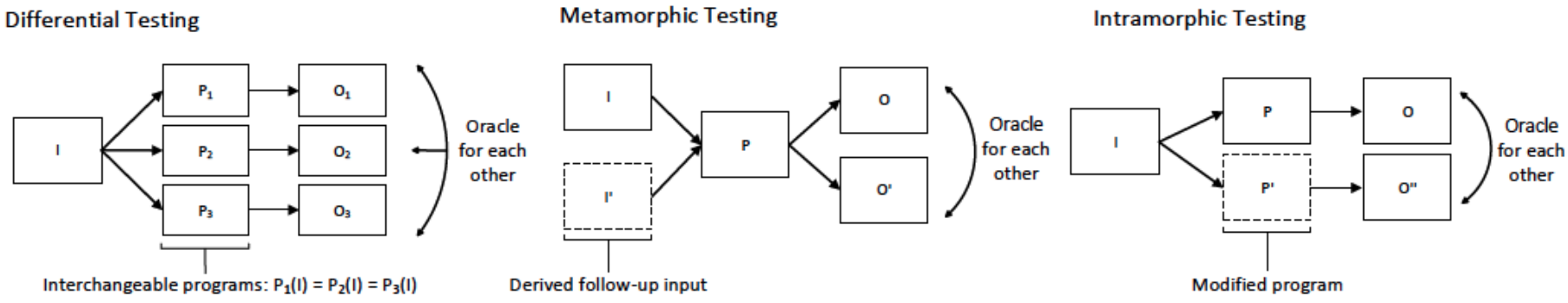
- **蜕变和差分测试的辨析**
  - **相同点**：都围绕待测程序中/间的某种必要属性（Necessary Property）展开；都是黑盒测试技术
  - **不同点**
    - **必要属性的类型不同**：蜕变测试利用了单一待测程序的必要属性；差分测试则利用了多个相似待测程序间的必要属性；
    - **正确性验证的准则不同**：蜕变测试的正确性验证准则是可变的，由蜕变关系决定；差分测试的正确性准则是固定的，即不同相似待测程序在同一测试输入上的输出结果应该相同。

- **蜕变和差分测试相结合**
  - Intramorphic Testing[1]：引入白盒思想，构造待测程序的修改版本来提供测试预言。例如，将排序算法f中的≥替换成<得到修改版本f'，则“两个算法的排序结果相反”可以作为正确性验证的准则。
  - Retromorphic Testing[2]：提出了双向程序验证结构，利用**反程序**提供测试预言。例如待测程序为$f(x) = \sin(x)$，那么这个待测程序的反程序就是$f^{-1}(x) = arc\sin(x)$，两者之间的关系$f^{-1}(f(x)) = x, -\frac{\pi}{2} \leq x \leq \frac{\pi}{2}$，就可以作为正确性验证的准则

[1] Rigger M, Su Z. Intramorphic testing: A new approach to the test oracle problem[C]//Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. 2022: 128-136.
[2] Yu B, Mang Q, Guo Q, et al. Retromorphic Testing: A New Approach to the Test Oracle Problem[J]. arXiv preprint arXiv:2310.06433, 2023.

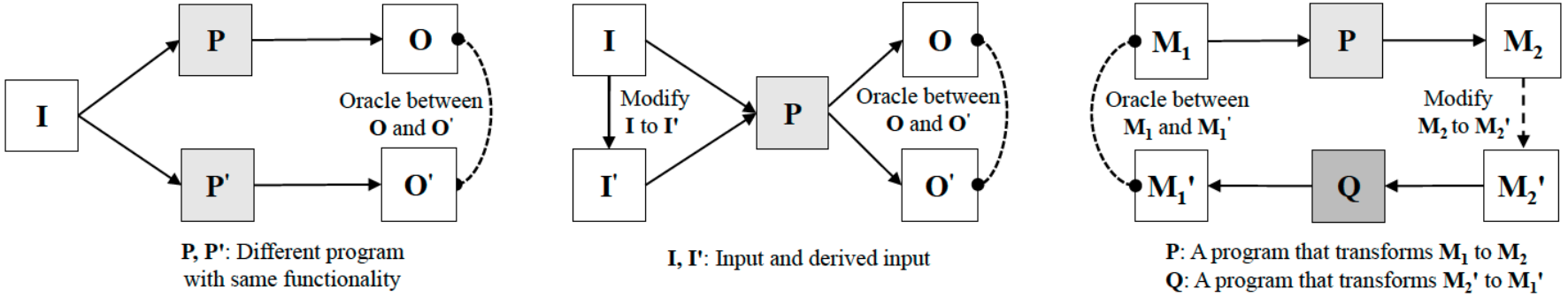
蜕变与差分

- Intramorphic Testing框架[1]



[1] Rigger M, Su Z. Intramorphic testing: A new approach to the test oracle problem[C]//Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. 2022: 128-136.
[2] Yu B, Mang Q, Guo Q, et al. Retromorphic Testing: A New Approach to the Test Oracle Problem[J]. arXiv preprint arXiv:2310.06433, 2023.

- Retromorphic Testing框架[2]



[1] Rigger M, Su Z. Intramorphic testing: A new approach to the test oracle problem[C]//Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. 2022: 128-136.
[2] Yu B, Mang Q, Guo Q, et al. Retromorphic Testing: A New Approach to the Test Oracle Problem[J]. arXiv preprint arXiv:2310.06433, 2023.

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

Thank you!