模糊测试

南京大学 软件学院 iSE实验室

目录

01

起源与发展

模糊测试的诞生

- Barton P. Miller，模糊测试之父
  - 1988年，Barton在自己的操作系统课上首次提出模糊测试
  - 1990年，*An Empirical Study of the Reliability of UNIX Utilities*[1]

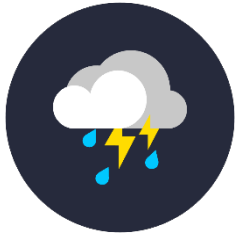
Barton P. Miller

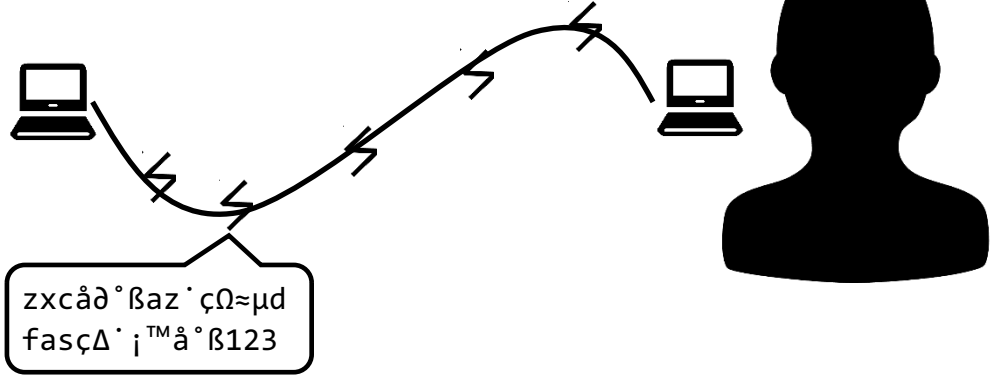
Vilas Distinguished Achievement Professor
Amar & Belinder Sohi Professor in Computer Sciences

[1] Miller B P, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM, 1990, 33(12): 32-44.

模糊测试的诞生

常见的字符乱码能够导致系统崩溃？





Crash !

模糊测试的诞生

- 出发点：提升UNIX操作系统的可靠性
- 技术构想[1]
  - **核心组件**：一组用于产生随机字符的程序
  - **中心思想**：以随机字符串作为输入，运行操作系统组件（Utilities），观察是否崩溃
  - **最终结果**：保留能够产生崩溃的字符串输入，分析崩溃的类型，对崩溃进行分类
- 结果：在22个（总90个）组件程序（Utility Program）上触发崩溃

[1] Miller B P, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM, 1990, 33(12): 32-44.

近期发展

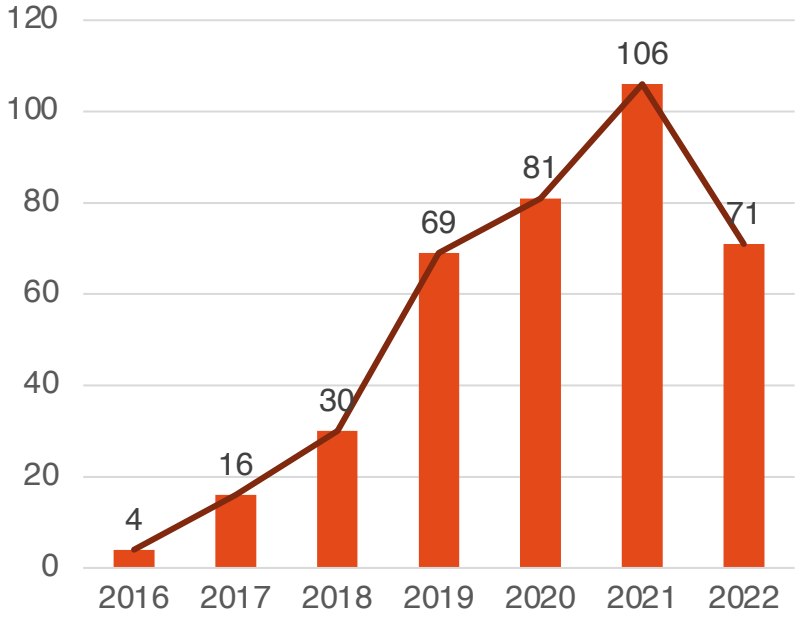
Year	Fuzzer	Solution(Process)	Fitness By	Target App/Bug	Input		Runtime Info.
					Muta. -based	Gene. -based	
2006	Sidewinder [62]	MC(seed.) + GA(rete.)	block transition	general	✓		◐
2007	RANDOOP [141]	GA(rete.)	legality	object-oriented	✓		●
2013	FuzzSim [192]	WCCP(seed.)	#bugs	general	✓		●
2014	COVERSET [158]	MSCP(set.)	code coverage	general		✓	◐
		ILP(seed.)	#bugs		✓		●+◐+○
2015	Joeri *et al.* [55]	GA(rete.)	state machine	protocol	✓		●
2016	AFLFast [23]	MC(seed.) + GA(rete.)	path transition	general	✓		◐
2016	classfuzz [42]	MH(mutation.) + GA(rete.)	code coverage	JVM	✓		◐
2017	VUzzer [157]	MC(seed.) + GA(rete.)	block transition	general	✓		◐
2017	AFLGo [22]	SA(seed.) + GA(rete.)	path transition	general	✓		◐
2017	NEZHA [151]	GA(rete.)	asymmetry	semantic bugs	✓		●+◐
2017	DeepXplore [148]	GA(rete.)	neuron coverage	deep learning	✓		◐
2018	STADS [17]	Species(seed.)*	state discovery	general	✓		◐
2018	CollAFL [66]	GA(rete.)	△code coverage	general	✓		◐
2018	Angora [37]	GD(byte.) + GA(rete.)	△code coverage	general	✓		◐
2019	DigFuzz [210]	MC(seed.) + GA(rete.)	block transition	general	✓		○
2019	MOPT [116]	PSO(mutation.) + GA(rete.)	code coverage	general	✓		◐
2019	NEUZZ [172]	NN(byte.) + GA(rete.)	branch behavior	general	✓		◐
2019	Cerebro [105]	MOO(seed.) + GA(rete.)	△code coverage	general	✓		◐
2019	DifFuzz [138]	GA(rete.)	asymmetry	side-channel	✓		●
2020	AFLNET [154]	GA(rete.)	state machine	protocol	✓		◐
2020	EcoFuzz [203]	VAMAB(seed.) + GA(rete.)	path transition	general	✓		◐
2020	Entropic [21]	Shannon(seed.) + GA(rete.)	state discovery	general	✓		◐
2020	MTFuzz [171]	MTNN(byte.) + GA(rete.)	△branch behavior	general	✓		◐
2020	Ankou [118]	GA(rete.)	△code coverage	general	✓		◐
2020	FIFUZZ [87]	GA(rete.)	△code coverage	error-handling	✓		◐
2020	IJON [6]	GA(rete.)	△code coverage	general	✓		◐
2020	Krace [194]	GA(rete.)	alias coverage	data race	✓		◐
2021	AFL-HIER [88]	UCB1(seed.) + GA(rete.)	△path transition	general	✓		◐
2021	PGFUZZ [82]	GA(rete.)	safety policy	robotic vehicle	✓		●
2021	Yousra *et al.* [1]	GA(rete.)	validation log	SmartTV	✓		●
2021	AFLChurn [214]	SA(seed.) + ACO(byte.) + GA(rete.)	path transition + commit history	general	✓		◐

2006~2021年间发表的主要模糊测试技术一览[1]

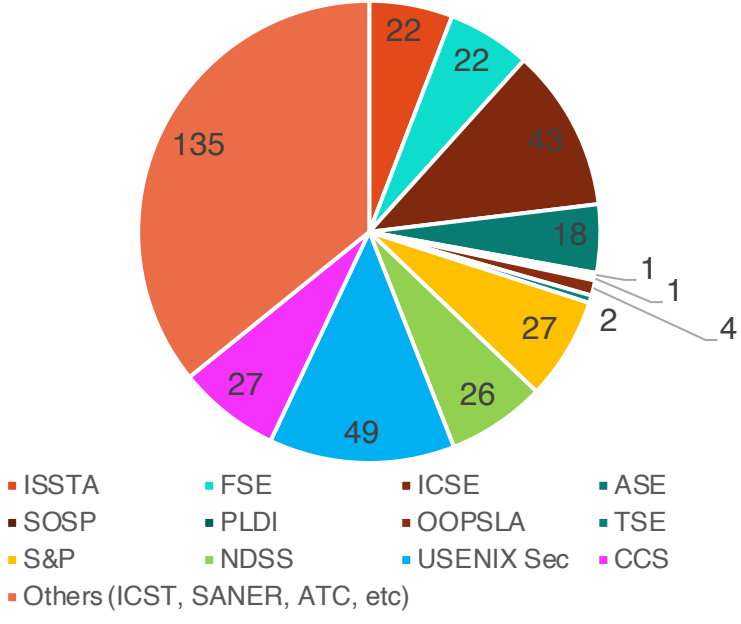
[1] Zhu X, Wen S, Camtepe S, et al. Fuzzing: a survey for roadmap[J]. ACM Computing Surveys (CSUR), 2022.

近期发展

简单的框架，多样的拓展



2016~2022年发表刊物数量趋势（不完全统计）



2016~2022年发表刊物分布（不完全统计）

[1] 数据来源：https://github.com/wcventure/FuzzingPaper/blob/master/README.md

02

概念与框架

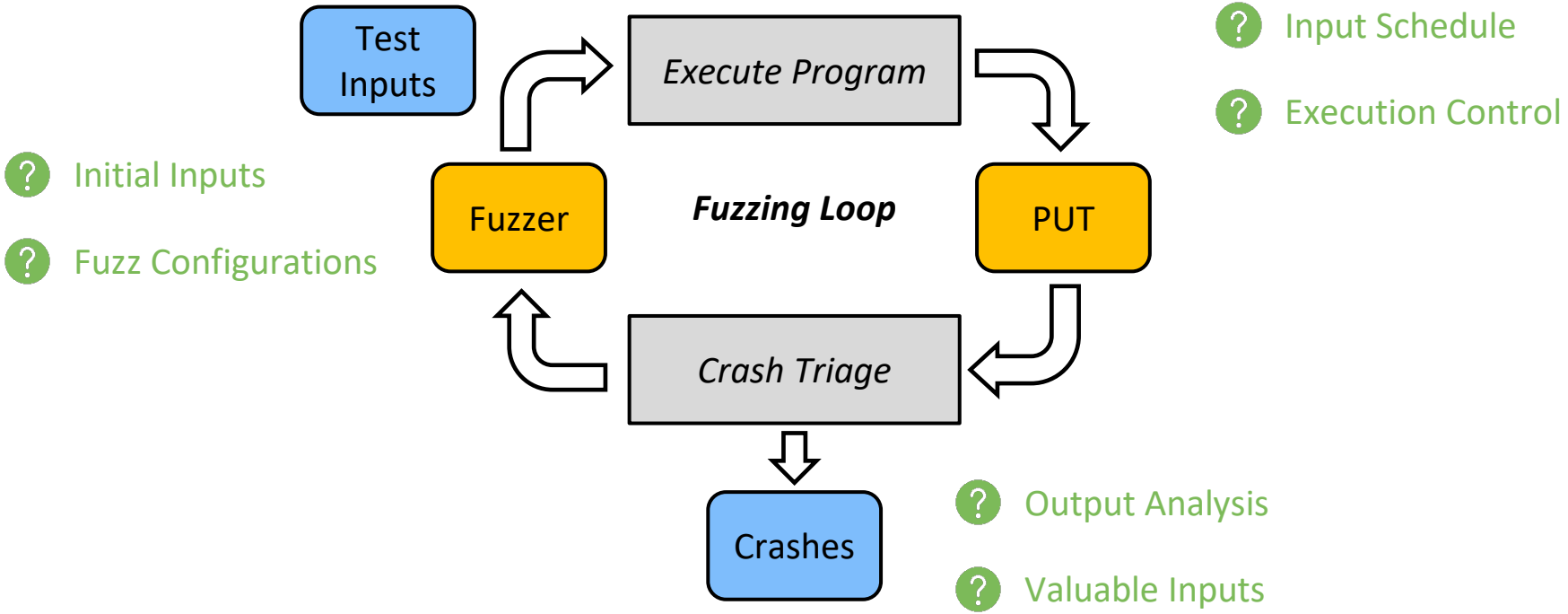
初始构想

- 三要素[1]：一个（套）**工具**、一个**目标**、一个**循环**
  - **工具**：模糊器（Fuzzer）
  - **目标**：待测程序（PUT）
  - **循环**：执行程序 ⇄ 崩溃分派（Crash Triage）

[1] Miller B P, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM, 1990, 33(12): 32-44.

初始构想

- 三要素[1]：一个（套）**工具**、一个**目标**、一个**循环**



[1] Miller B P, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM, 1990, 33(12): 32-44.

相关术语

- 模糊（Fuzzing）与模糊测试（Fuzz Testing）
- 模糊器（Fuzzer）
- 模糊运动（Fuzzing Campaign）
- 缺陷预言（Bug Oracle）
- 模糊配置（Fuzz Configuration）
- 测试输入（Test Input）、测试用例（Test Case）与种子输入（Seed Input）

[1] 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.
[2] Fraser G, Arcuri A. Evosuite: automatic test suite generation for object-oriented software[C]//Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. 2011: 416-419.
[3] 测试用例定义，维基百科：https://zh.wikipedia.org/zh-cn/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B

相关术语

- 模糊（Fuzzing）与模糊测试（Fuzz Testing）
  - **模糊**是指使用从输入空间（模糊输入空间）采样得到的输入来执行待测程序（PUT，Program Under Test）的过程。该模糊输入空间代表着测试人员针对待测程序定义的预期输入。
  - **模糊测试**是一种应用模糊（过程）来验证待测程序是否违反正确性策略（Correctness Policy）的测试技术。
  - 在文献中一般可互换

[1] 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.
[2] Fraser G, Arcuri A. Evosuite: automatic test suite generation for object-oriented software[C]//Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. 2011: 416-419.
[3] 测试用例定义，维基百科：https://zh.wikipedia.org/zh-cn/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B

相关术语

- 模糊器（Fuzzer）
  - **模糊器**是一个（组）用于实现模糊测试的程序。→ 核心组件
- 模糊运动（Fuzz Campaign）
  - **模糊运动**是指一个**模糊器**按照一组特定的**正确性政策**在一个给定**待测程序**上的一次具体的**执行**。→ $< fuzzer,\ program >$

[1] 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.
[2] Fraser G, Arcuri A. Evosuite: automatic test suite generation for object-oriented software[C]//Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. 2011: 416-419.
[3] 测试用例定义，维基百科：https://zh.wikipedia.org/zh-cn/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B

相关术语

- 缺陷预言（Bug Oracle）
  - 一个用于确定一次给定**执行**是否违反具体**正确性策略**的程序，通常作为**模糊器**的一部分出现。
- 模糊配置（Fuzz Configuration）
  - 一组控制和描述模糊（测试）算法的数据和约束

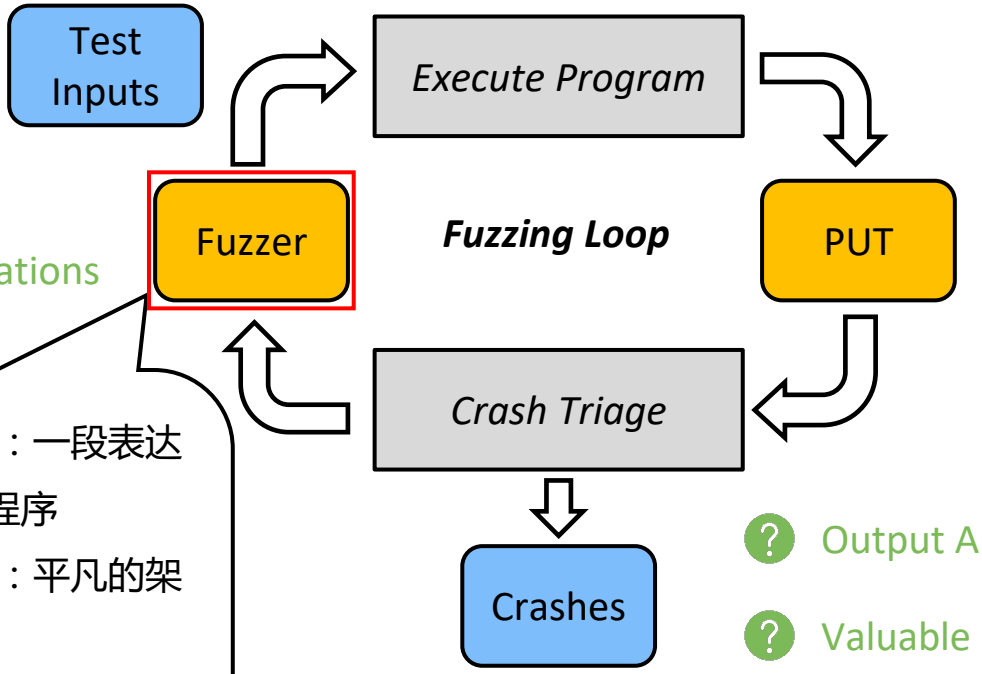
[1] Manès V J M, Han H S, Han C, et al. The art, science, and engineering of fuzzing: A survey[J]. IEEE Transactions on Software Engineering, 2019, 47(11): 2312-2331.
[2] Fraser G, Arcuri A. Evosuite: automatic test suite generation for object-oriented software[C]//Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. 2011: 416-419.
[3] 测试用例定义，维基百科，https://zh.wikipedia.org/zh-cn/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B

相关术语

- 测试输入（Test Input）、测试用例（Test Case）与种子输入（Seed Input）
  - **测试输入**是一组用于驱动待测程序执行的数据
  - **测试用例**是一组用于确定应用软件或软件系统是否能够正确工作的条件或变量[3] → 输入 + 逻辑（调用序列） + 预言
  - **种子输入**是一个或一组在模糊测试过程中为输入生成（Input Generation）提供基准的测试输入，简称**种子**（Seed）。

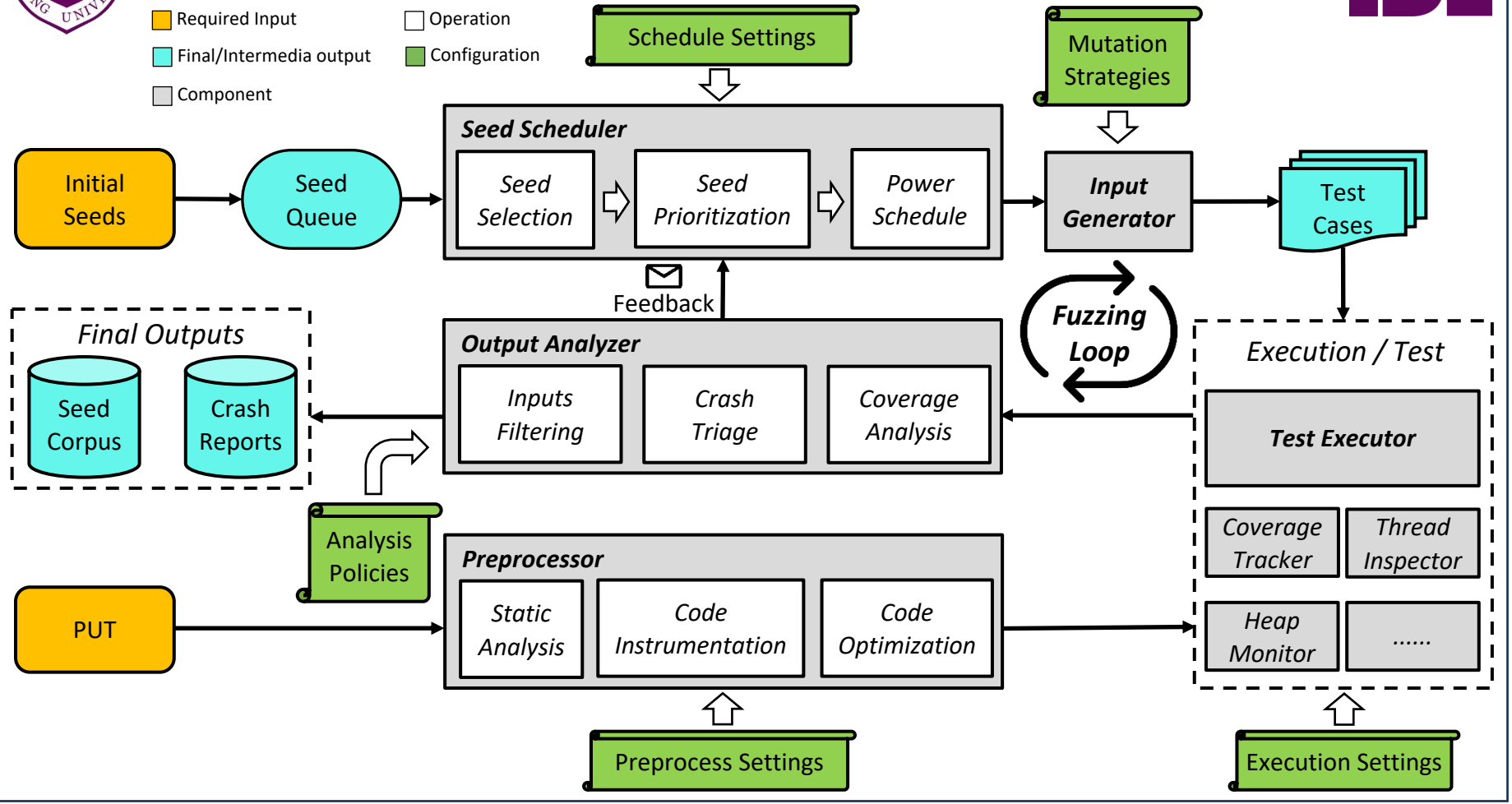
[1] Manès V J M, Han H S, Han C, et al. The art, science, and engineering of fuzzing: A survey[J]. IEEE Transactions on Software Engineering, 2019, 47(11): 2312-2331.
[2] Fraser G, Arcuri A. Evosuite: automatic test suite generation for object-oriented software[C]//Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. 2011: 416-419.
[3] 测试用例定义，维基百科：https://zh.wikipedia.org/zh-cn/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B

模糊测试框架



- **模糊测试的核心**：一段表达Fuzzing循环的程序
- **蓬勃发展的关键**：平凡的架构，多样的拓展
- **主要构成**：输入生成、测试执行、输出分析

模糊测试框架


Required Input
Final/Intermedia output
Component
Operation
Configuration
Schedule Settings
Mutation Strategies
Seed Scheduler
Seed Selection
Seed Prioritization
Power Schedule
Initial Seeds
Seed Queue
Input Generator
Test Cases
Feedback
Fuzzing Loop
Final Outputs
Seed Corpus
Crash Reports
Output Analyzer
Inputs Filtering
Crash Triage
Coverage Analysis
Execution / Test
Test Executor
Coverage Tracker
Thread Inspector
Heap Monitor
.....
Analysis Policies
Preprocessor
Static Analysis
Code Instrumentation
Code Optimization
PUT
Preprocess Settings
Execution Settings

03

家族与分类

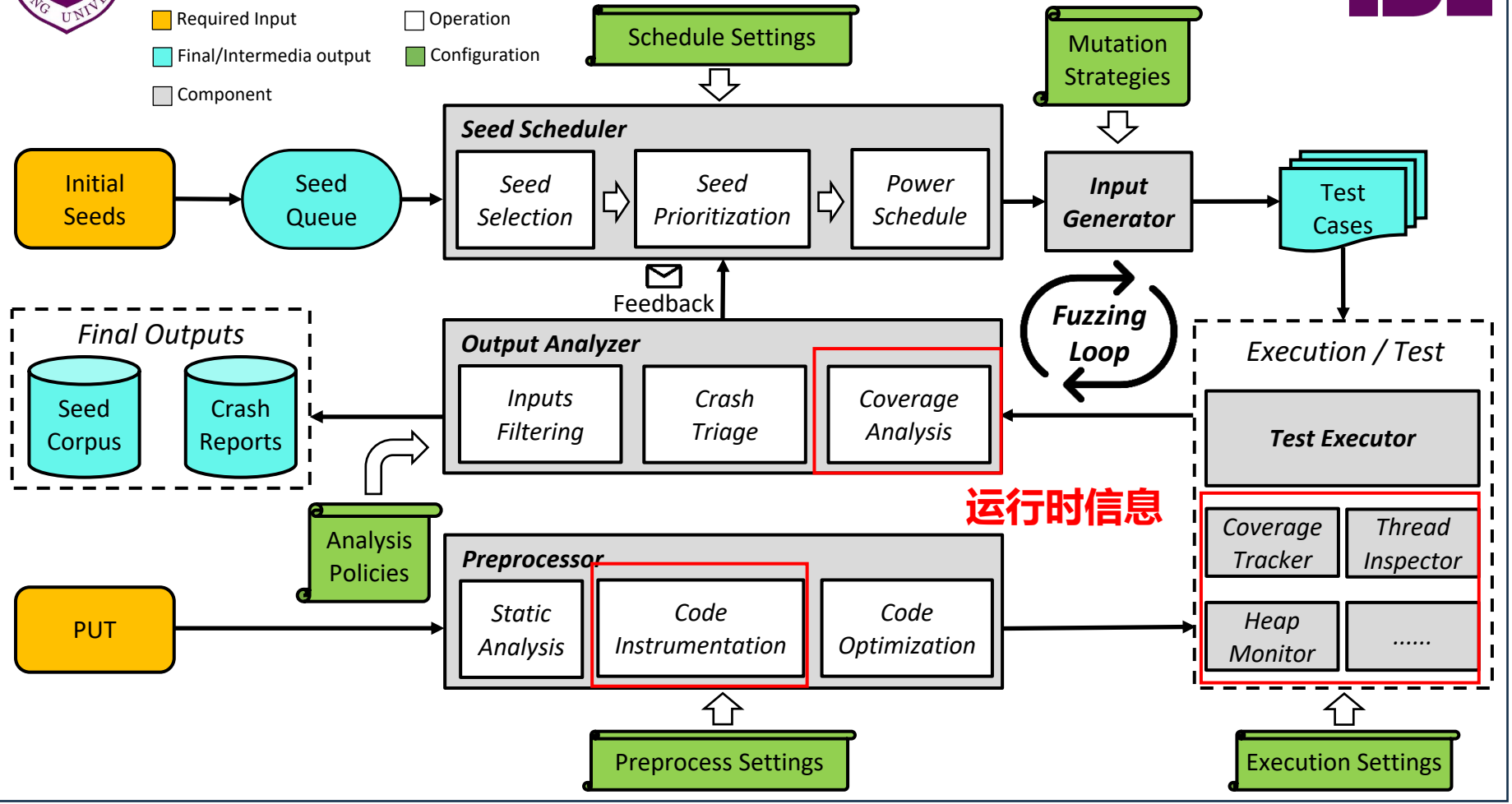
模糊测试家族

- 根据基础Fuzzer划分家族
  - AFL家族（C/C++）：AFL、AFLFast、AFLSmart、AFLNet、AFLGo、AFLIoT、FairFuzz、Mopt.、Neuzz
  - LibFuzzer家族（C/C++）：LibFuzzer、Entropic
  - JQF家族（Java）：JQF、BeDivFuzz、CONFETTI
  - 其他（Rust、Python等）：Angora、DeepXplore

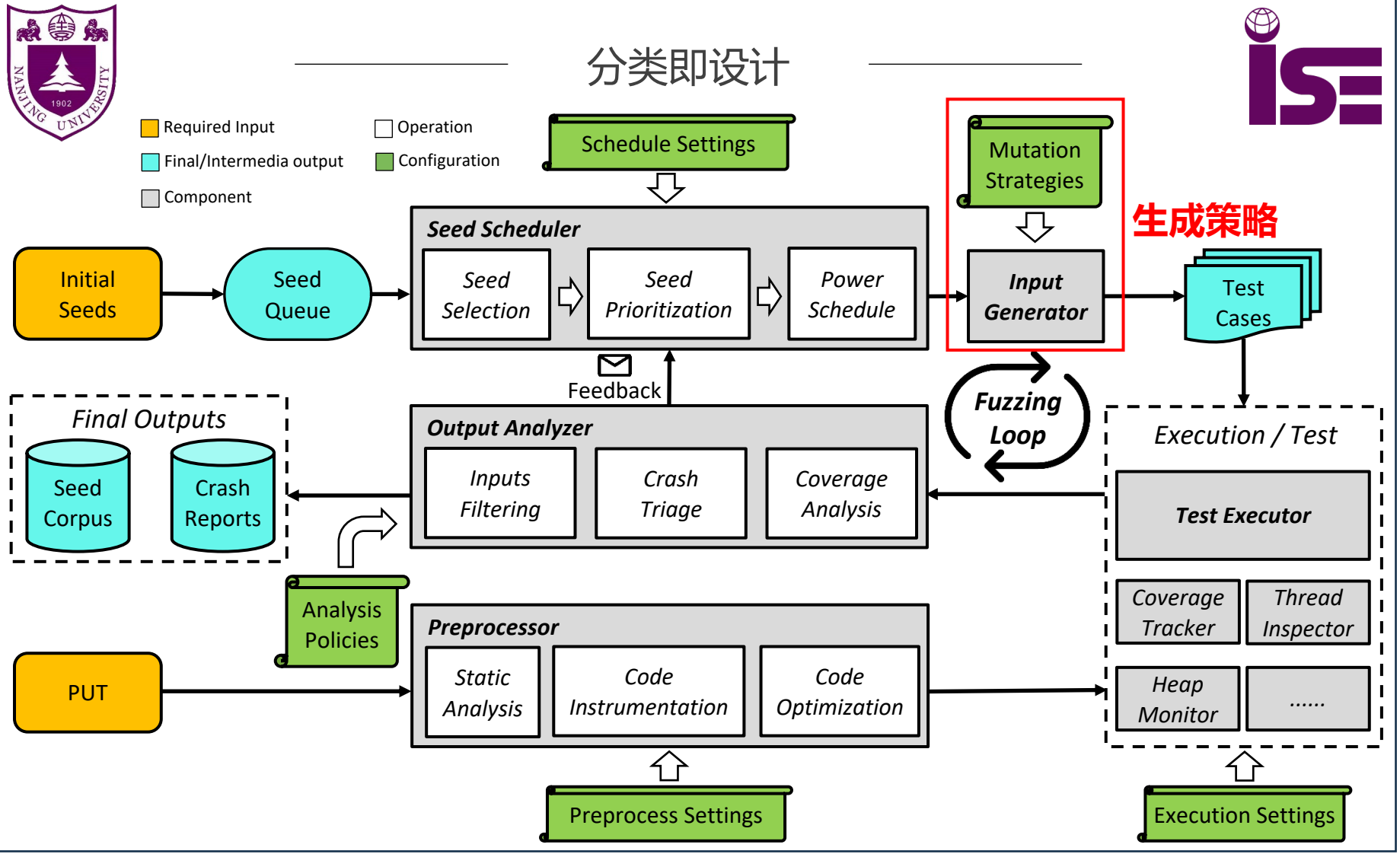
模糊测试分类

- 根据组件核心或技术贡献进行分类
  - 按照**采用的运行时信息**：黑盒、灰盒、白盒
  - 按照**输入生成的策略**：Mutation-based, Generation-based
  - 按照**引导过程**：Search-based（一些启发式算法），Gradient-based（梯度下降）
  - 按照**测试的目的**：定向、非定向、某一类缺陷
  - 按照**应用领域**：网络协议、Compiler、DNN、IoT、内核
  - 按照**优化角度**：种子调度、变异策略、能量调度、过程建模

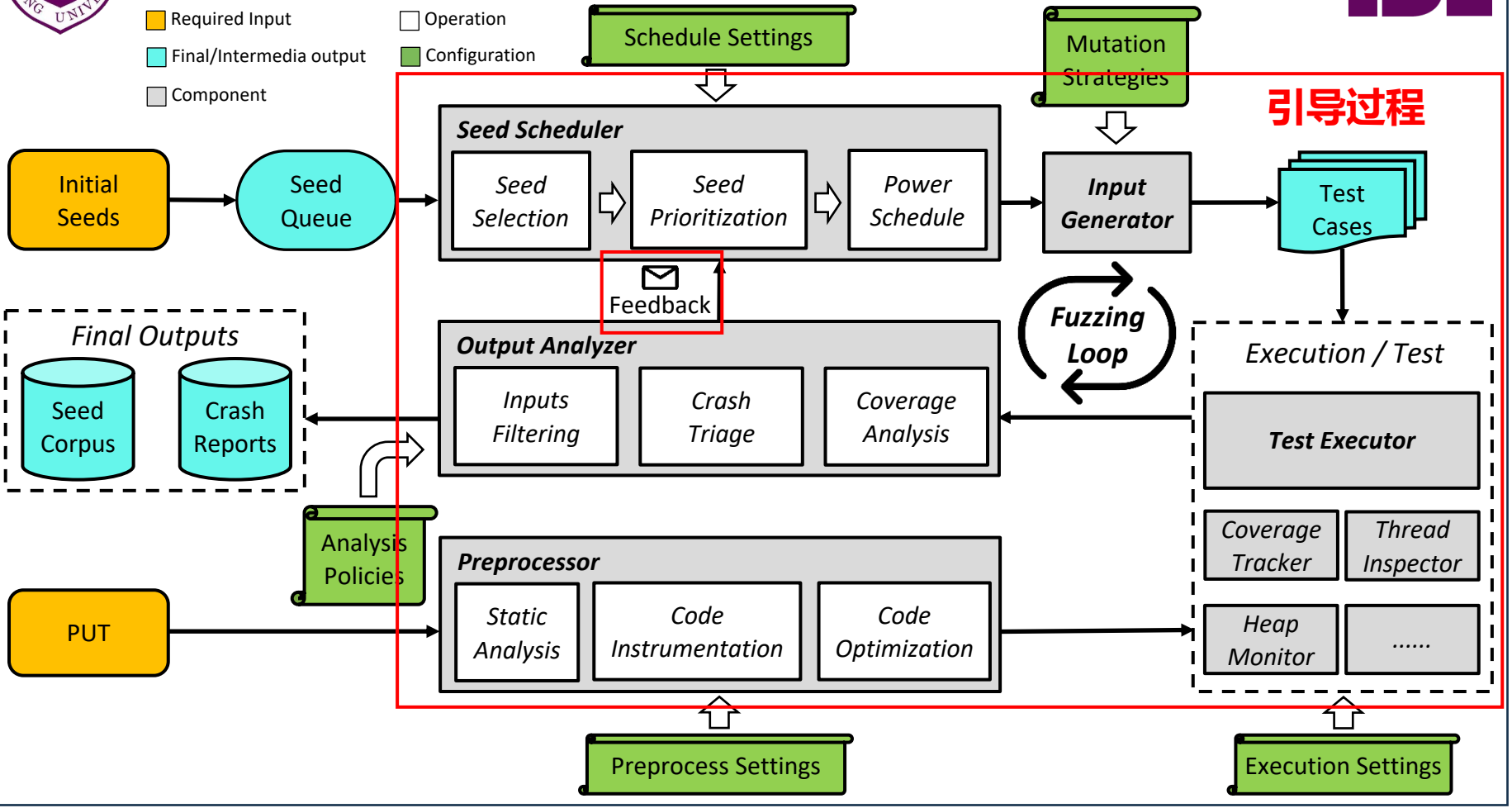
分类即设计


Required Input
Operation
Final/Intermedia output
Configuration
Component
Schedule Settings
Mutation Strategies
Seed Scheduler
Seed Selection
Seed Prioritization
Power Schedule
Initial Seeds
Seed Queue
Input Generator
Test Cases
Feedback
Fuzzing Loop
Final Outputs
Seed Corpus
Crash Reports
Output Analyzer
Inputs Filtering
Crash Triage
Coverage Analysis
Execution / Test
Test Executor
运行时信息
Coverage Tracker
Thread Inspector
Heap Monitor
.....
Analysis Policies
Preprocessor
Static Analysis
Code Instrumentation
Code Optimization
PUT
Preprocess Settings
Execution Settings

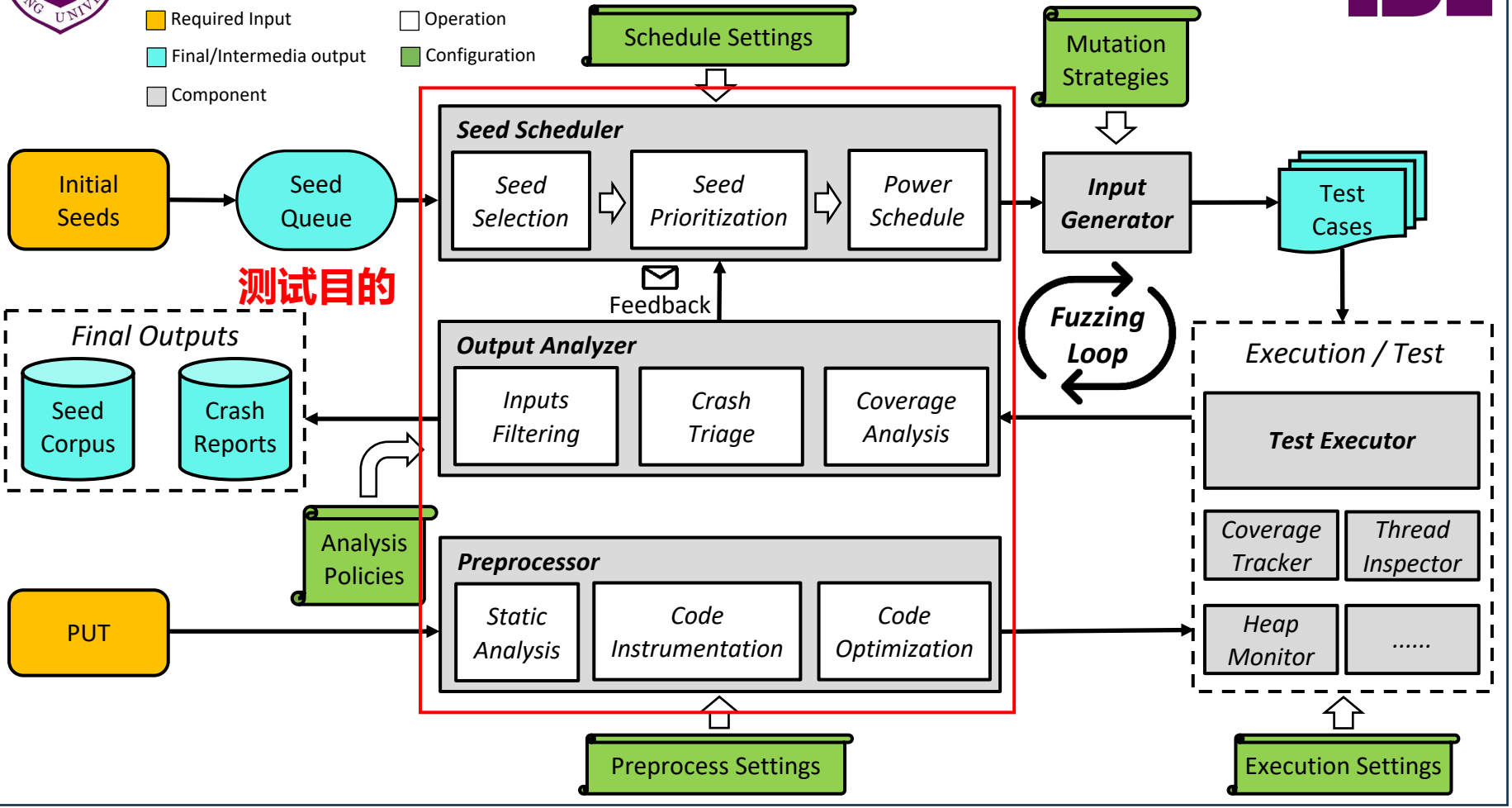
分类即设计


Required Input
Operation
Final/Intermedia output
Configuration
Component
Schedule Settings
Mutation Strategies
生成策略
Seed Scheduler
Seed Selection
Seed Prioritization
Power Schedule
Initial Seeds
Seed Queue
Input Generator
Test Cases
Feedback
Fuzzing Loop
Final Outputs
Seed Corpus
Crash Reports
Output Analyzer
Inputs Filtering
Crash Triage
Coverage Analysis
Execution / Test
Test Executor
Coverage Tracker
Thread Inspector
Heap Monitor
.....
Analysis Policies
Preprocessor
Static Analysis
Code Instrumentation
Code Optimization
PUT
Preprocess Settings
Execution Settings

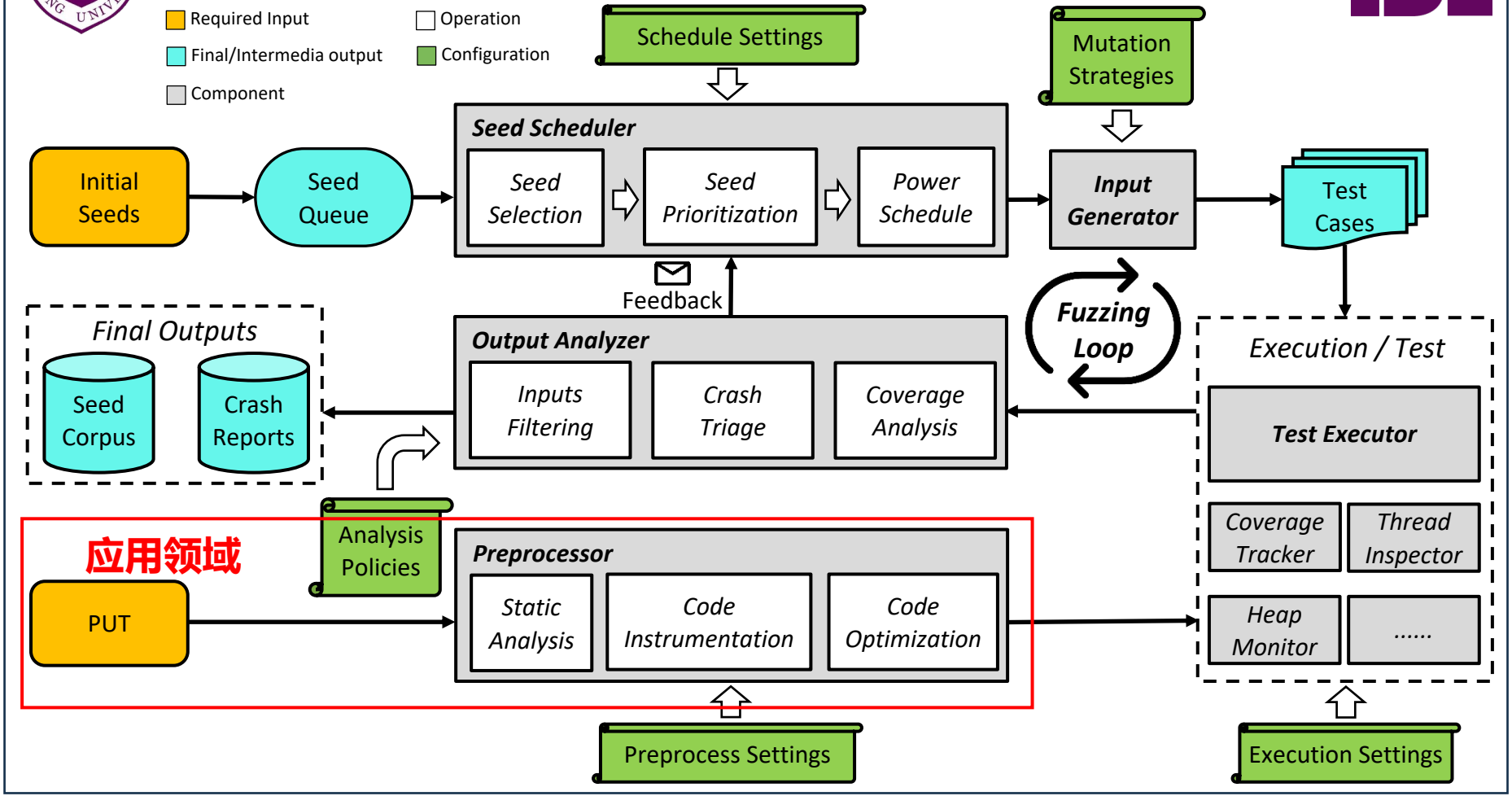
分类即设计


Required Input
Final/Intermedia output
Component
Operation
Configuration
Schedule Settings
Mutation Strategies
引导过程
Seed Scheduler
Seed Selection
Seed Prioritization
Power Schedule
Initial Seeds
Seed Queue
Input Generator
Test Cases
Feedback
Fuzzing Loop
Final Outputs
Seed Corpus
Crash Reports
Output Analyzer
Inputs Filtering
Crash Triage
Coverage Analysis
Execution / Test
Test Executor
Coverage Tracker
Thread Inspector
Heap Monitor
.....
Analysis Policies
Preprocessor
Static Analysis
Code Instrumentation
Code Optimization
PUT
Preprocess Settings
Execution Settings

分类即设计


Required Input
Operation
Final/Intermedia output
Configuration
Component
Schedule Settings
Mutation Strategies
Seed Scheduler
Seed Selection
Seed Prioritization
Power Schedule
Initial Seeds
Seed Queue
Input Generator
Test Cases
测试目的
Feedback
Fuzzing Loop
Final Outputs
Seed Corpus
Crash Reports
Output Analyzer
Inputs Filtering
Crash Triage
Coverage Analysis
Execution / Test
Test Executor
Coverage Tracker
Thread Inspector
Heap Monitor
.....
Analysis Policies
Preprocessor
Static Analysis
Code Instrumentation
Code Optimization
PUT
Preprocess Settings
Execution Settings

分类即设计


Required Input
Operation
Final/Intermedia output
Configuration
Component
Schedule Settings
Mutation Strategies
Seed Scheduler
Seed Selection
Seed Prioritization
Power Schedule
Initial Seeds
Seed Queue
Input Generator
Test Cases
Feedback
Fuzzing Loop
Final Outputs
Seed Corpus
Crash Reports
Output Analyzer
Inputs Filtering
Crash Triage
Coverage Analysis
Execution / Test
Test Executor
Coverage Tracker
Thread Inspector
Heap Monitor
.....
应用领域
Analysis Policies
PUT
Preprocessor
Static Analysis
Code Instrumentation
Code Optimization
Preprocess Settings
Execution Settings

按照运行时信息分类

- 按照**采用的运行时信息**[1]
  - 黑盒（Blackbox）模糊测试
    - 特点：不监控执行过程，也不使用执行过程中产生的任何信息，仅从输入和输出端入手优化模糊测试
    - 引导方式：利用输入格式或输出状态引导测试执行
    - 优缺点：效率高，但引导的有效性上面有所欠缺
    - 代表性工作：KIF[2]、IoTFuzzer[3]、CodeAlchemist[4]

[1] Zhu X, Wen S, Camtepe S, et al. Fuzzing: a survey for roadmap[J]. ACM Computing Surveys (CSUR), 2022.
[2] Abdelnur H J, State R, Festor O. KiF: a stateful SIP fuzzer[C]//Proceedings of the 1st international Conference on Principles, Systems and Applications of IP Telecommunications. 2007: 47-56.
[3] Chen J, Diao W, Zhao Q, et al. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing[C]//NDSS. 2018.
[4] Han H S, Oh D H, Cha S K. CodeAlchemist: Semantics-Aware Code Generation to Find Vulnerabilities in JavaScript Engines[C]//NDSS. 2019.

按照运行时信息分类

- 按照**采用的运行时信息**[1]
  - 白盒（Whitebox）模糊测试
    - 特点：使用混合执行、污点分析（Taint Analysis）等比较昂贵的白盒分析技术优化模糊测试过程
    - 引导方式：利用详细的程序分析结果引导测试执行
    - 优缺点：反馈更加有效，但是效率不高、适配性较差
    - 代表性工作：Driller[2]、QSYM[3]、CONFETTI[4]

[1] Zhu X, Wen S, Camtepe S, et al. Fuzzing: a survey for roadmap[J]. ACM Computing Surveys (CSUR), 2022.
[2] Stephens N, Grosen J, Salls C, et al. Driller: Augmenting fuzzing through selective symbolic execution[C]//NDSS. 2016, 16(2016): 1-16.
[3] Yun I, Lee S, Xu M, et al. QSYM: A practical concolic execution engine tailored for hybrid fuzzing[C]//27th USENIX Security Symposium (USENIX Security 18). 2018: 745-761.
[4] Kukucka J, Pina L, Ammann P, et al. CONFETTI: Amplifying Concolic Guidance for Fuzzers[C]//44th IEEE/ACM International Conference on Software Engineering, ser. ICSE. 2022, 22.

按照运行时信息分类

- 按照**采用的运行时信息**[1]
  - **灰盒**（Greybox）模糊测试
    - Coverage-based Greybox Fuzzing, CGF
    - 特点：采用轻量级插装对程序进行监控，在执行过程中收集各类信息，如分支覆盖、线程执行、堆栈状态等
    - 引导方式：利用收集到的执行信息（内部状态）引导测试执行
    - 代表性工作：AFL、AFLGo、EcoFuzz、Zest、BeDivFuzz

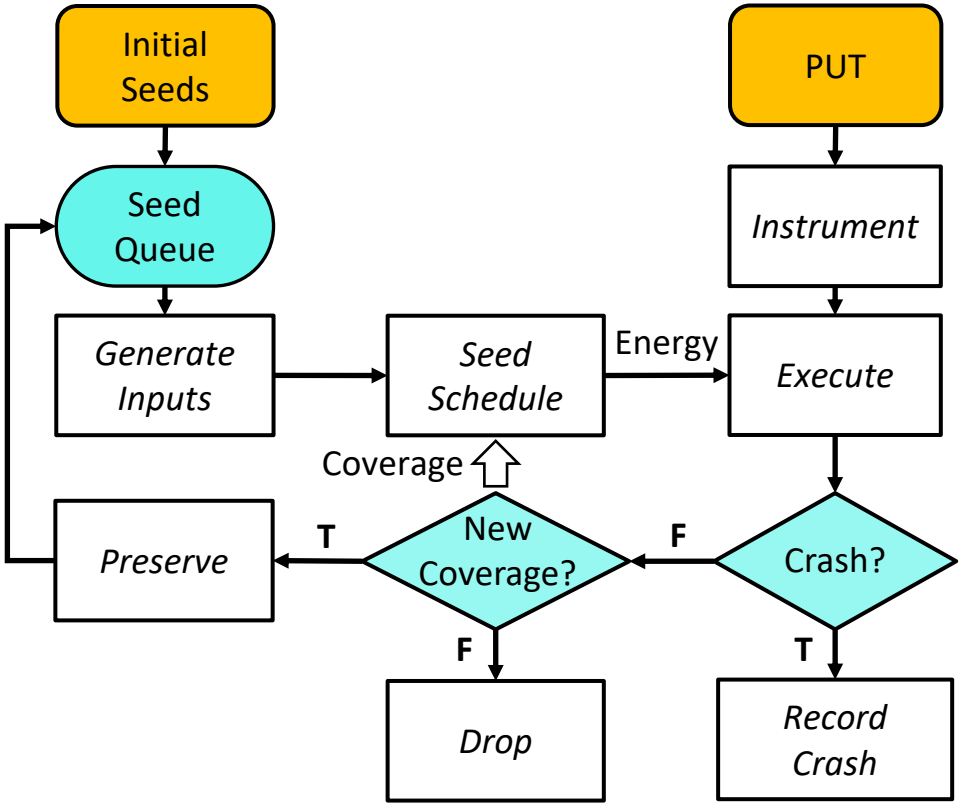
灰盒模糊测试框架

Algorithm 1 Greybox Fuzzing

Input: Initial Seeds S, Instrumented PUT p
Output: Seed Corpus S'

```
1:  S' ← S
2:  S_crash ← ∅
3:  repeat                                          ▷ Main fuzzing loop
4:      s ← CHOOSENEXT(S')
5:      e ← ASSIGNENERGY(s)                         ▷ Power scheduling
6:      for i from 1 to e do
7:          s' ← MUTATESEED(s)
8:          res ← EXECUTE(p, s')
9:          if ISCRASH(s', res) then
10:             add s' to S_crash
11:         else if ISINTERESTING(s', res) then
12:             add s' to S'
13:         end if
14:     end for
15: until resources exceeds or abort-signal comes
16: return S', S_crash
```

CGF伪代码



CGF一般流程

按照生成策略分类

- 按照**输入生成的策略**
  - Input Generator、Mutation Strategies
  - Mutation-based：基于随机或启发式变异策略
  - Generation-based：基于一定的文法规则/结构信息

基于变异的模糊测试

- Mutation-based：基于随机变异或启发式变异策略
  - **本质**：将种子输入转换为比特串（Bits），对比特串进行变换
  - **优点**：可拓展性强，易于泛化，理论上可用于任意输入（图片、文本、音视频）
  - **缺点**：容易破坏输入的结构、产生无效无效输入（Invalid Input）；生成的大部分输入都无法通过语义检查（Semantic Checking），很难探测深层次的程序状态和程序元素（Program Elements）

基于变异的模糊测试

- Mutation-based：AFL[1]
  - AFL变异算子
    - bitflip L/S ：以S为增量，每次翻转L位
    - arith L/8：加/减长度为L的小整数
    - interest L/8：翻转“有趣”字节位
    - **havoc**：对输入进行大肆破坏
    - splice：随机拼接两个种子输入

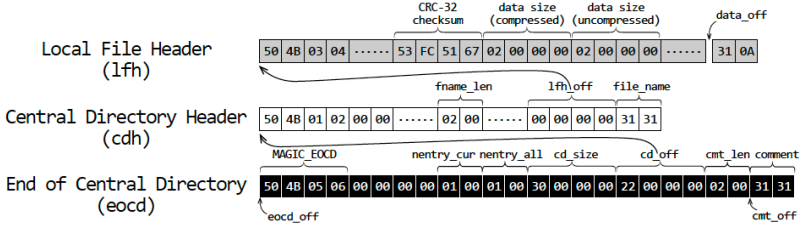
Fuzzing a Picture

[1] AFL博客：https://afl-1.readthedocs.io/en/latest/index.html

基于变异的模糊测试

- Mutation-based：SLF（ICSE'19）[1]
  - 思想：分析程序中的语义检查、识别比特串中与影响语义检查的域（Field）、根据两者之间的关系制定变异策略

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000h	50	4B	03	04	00	00	00	00	00	00	00	00	00	00	53	FC
00000010h	51	67	02	00	00	00	02	00	00	00	02	00	00	00	31	31
00000020h	31	0A	50	4B	01	02	00	00	00	00	00	00	00	00	00	00
00000030h	00	00	53	FC	51	67	02	00	00	00	02	00	00	00	02	00
00000040h	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000050h	31	31	50	4B	05	06	00	00	00	00	01	00	01	00	30	00
00000060h	00	00	22	00	00	00	02	00	31	31						

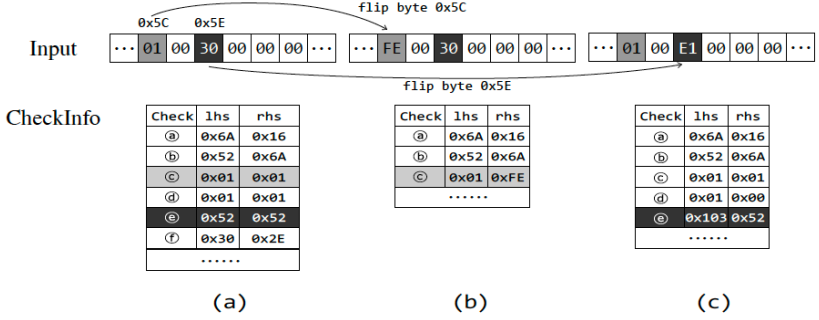


一个合法Zip文件的比特串

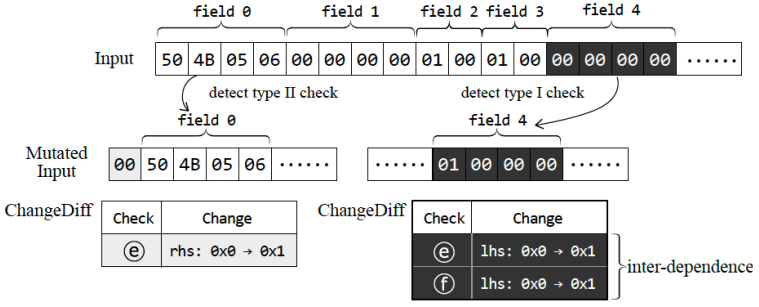
[1] You W, Liu X, Ma S, et al. SLF: Fuzzing without valid seed inputs[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 712-723.

基于变异的模糊测试

- Mutation-based：SLF（ICSE'19）[1]
  - 思想：分析程序中的语义检查、识别比特串中与影响语义检查的域（Field）、根据两者之间的关系制定变异策略



字节分组

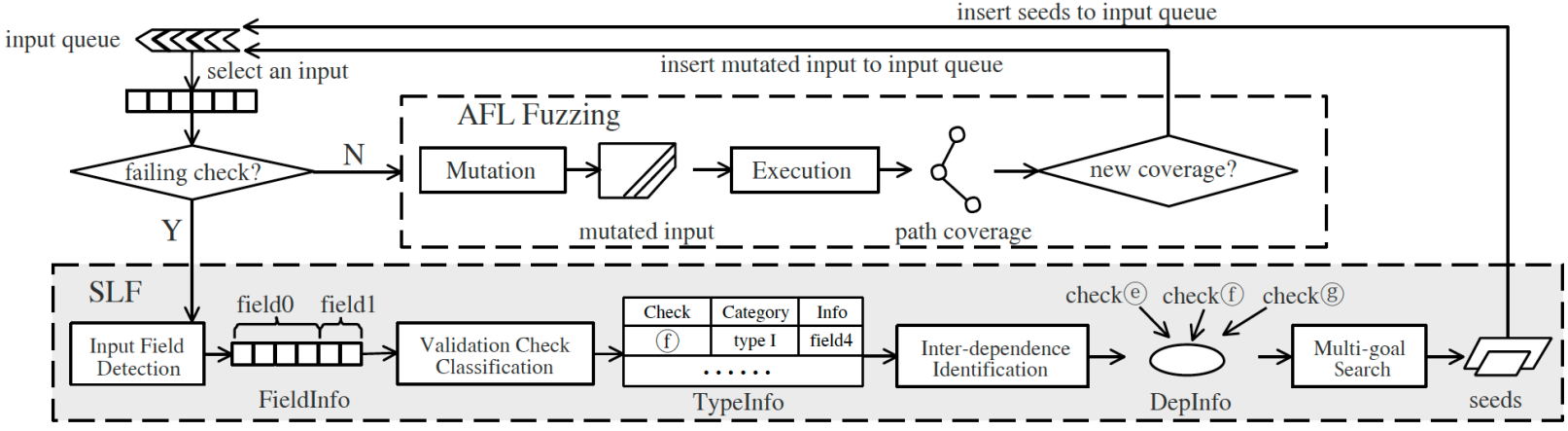


类型推断与关系建立

[1] You W, Liu X, Ma S, et al. SLF: Fuzzing without valid seed inputs[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 712-723.

基于变异的模糊测试

- Mutation-based：SLF（ICSE'19）[1]
  - 思想：分析程序中的语义检查、识别比特串中与影响语义检查的域（Field）、根据两者之间的关系制定变异策略



SLF技术流程概览

[1] You W, Liu X, Ma S, et al. SLF: Fuzzing without valid seed inputs[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 712-723.

基于生成的模糊测试

- Generation-based：基于一定的文法规则/结构信息
  - 本质：利用给定的、或者挖掘/学习得到的文法规则，来构建能够通过（语法）检查的结构化输入
  - **优点**：容易生成合法、有效输入（Valid Inputs），适用于对输入结构性要求较高的场景，如针对解析器（Parser）、解释器（Interpreter）以及编译器（Compiler）的测试
  - **缺点**：需要人工赋予一定的领域知识。在没有人工指定文法的情况下，很难得到完整的文法规则

基于生成的模糊测试

- Generation-based：Inputs from Hell (TSE'19)[1]
  - 思想：挖掘已有的测试输入，得到现有的测试输入分布。根据该分布进行输入生成以得到预期的测试输入
    - 表示输入的分布：概率文法（Probabilistic Grammar）
    - 预期输入
      - 与概率文法相同：生成与文法表示的输入**相似**的测试输入
      - 与概率文法相反：生成与文法表示的输入**互补**的测试输入

[1] Soremekun E, Pavese E, Havrikov N, et al. Inputs from hell learning input distributions for grammar-based test generation[J]. IEEE Transactions on Software Engineering, 2020.

基于生成的模糊测试

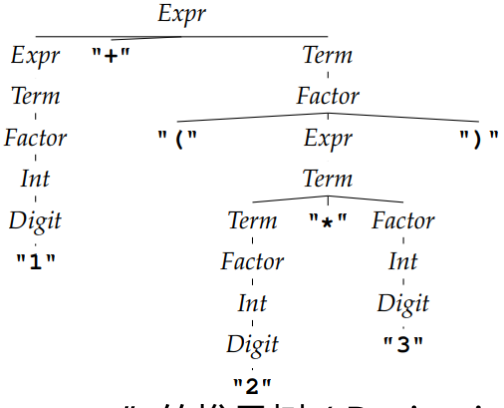
- Generation-based：Inputs from Hell (TSE'19)[1]
  - 思想：挖掘已有的测试输入，得到现有的测试输入分布。根据该分布进行输入生成以得到预期的测试输入

```
Expr → Term | Expr "+" Term | Expr "-" Term;
Term → Factor | Term "*" Factor
     | Term "/" Factor;
Factor → Int | "+" Factor
     | "-" Factor | "(" Expr ")";
Int → Digit Int | Digit;
Digit → "0" | "1" | "2" | "3" | ... | "9";
```

算数表达式的文法G

$I = 1 * (2 + 3)$

A Buggy Input

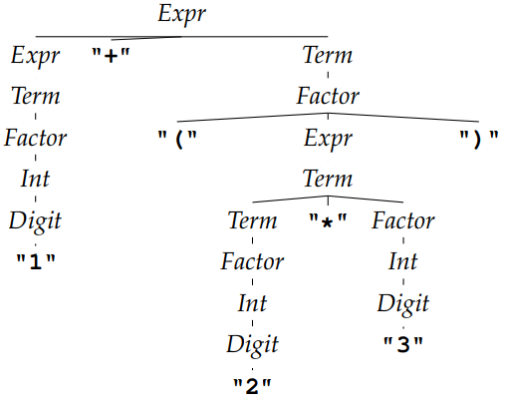


“$I = 1 + (2 * 3)$”的推导树（Derivation Tree）

[1] Soremekun E, Pavese E, Havrikov N, et al. Inputs from hell learning input distributions for grammar-based test generation[J]. IEEE Transactions on Software Engineering, 2020.

基于生成的模糊测试

- Generation-based：Inputs from Hell (TSE'19)[1]
  - 思想：挖掘已有的测试输入，得到现有的测试输入分布。根据该分布进行输入生成以得到预期的测试输入



"$I = 1 + (2 * 3)$" 的推导树（Derivation Tree）

```
Expr → 66.7% Term | 33.3% Expr "+" Term
     | 0% Expr "-" Term;
Term → 75% Factor | 25% Term "*" Factor
     | 0% Term "/" Factor;
Factor → 75% Int | 0% "+" Factor
     | 0% "-" Factor | 25% "(" Expr ")";
Int → 0% Digit Int | 100% Digit;
Digit → 0% "0" | 33.3% "1" | 33.3% "2"
     | 33.3% "3" | 0% "4" | 0% "5"
     | 0% "6" | 0% "7" | 0% "8" | 0% "9";
```

用于表示I产生的输入分布的概率文法G_p

[1] Soremekun E, Pavese E, Havrikov N, et al. Inputs from hell learning input distributions for grammar-based test generation[J]. IEEE Transactions on Software Engineering, 2020.

基于生成的模糊测试

- Generation-based：Inputs from Hell (TSE'19)[1]
  - 思想：挖掘已有的测试输入，得到现有的测试输入分布。根据该分布进行输入生成以得到预期的测试输入

```
Expr → 66.7% Term | 33.3% Expr "+" Term
     | 0% Expr "-" Term;
Term → 75% Factor | 25% Term "*" Factor
     | 0% Term "/" Factor;
Factor → 75% Int | 0% "+" Factor
     | 0% "-" Factor | 25% "(" Expr ")";
Int → 0% Digit Int | 100% Digit;
Digit → 0% "0" | 33.3% "1" | 33.3% "2"
     | 33.3% "3" | 0% "4" | 0% "5"
     | 0% "6" | 0% "7" | 0% "8" | 0% "9";
```

Input Generation

```
(2 * 3)
2 + 2 + 1 * (1) + 2
((3 * 3))
3 * (((3 + 3 + 3) * (2 * 3 + 3))) * (3)
3 * 1 * 3
((3) + 2 + 2 * 1) * (1)
1
((2)) + 3
```

用于表示I产生的输入分布的概率文法G_p

概率文法G_p产生的相似输入

[1] Soremekun E, Pavese E, Havrikov N, et al. Inputs from hell learning input distributions for grammar-based test generation[J]. IEEE Transactions on Software Engineering, 2020.

基于生成的模糊测试

- Generation-based：Inputs from Hell (TSE'19)[1]
  - 思想：挖掘已有的测试输入，得到现有的测试输入分布。根据该分布进行输入生成以得到预期的测试输入

```
Expr   → 66.7% Term | 33.3% Expr "+" Term
       | 0% Expr "-" Term;
Term   → 75% Factor | 25% Term "*" Factor
       | 0% Term "/" Factor;
Factor → 75% Int | 0% "+" Factor
       | 0% "-" Factor | 25% "(" Expr ")";
Int    → 0% Digit Int | 100% Digit;
Digit  → 0% "0" | 33.3% "1" | 33.3% "2"
       | 33.3% "3" | 0% "4" | 0% "5"
       | 0% "6" | 0% "7" | 0% "8" | 0% "9";
```

Grammar Inversion

```
Expr   → 0% Term | 0% Expr "+" Term
       | 100% Expr "-" Term;
Term   → 0% Factor | 0% Term "*" Factor
       | 100% Term "/" Factor;
Factor → 0% Int | 50% "+" Factor
       | 50% "-" Factor | 0% "(" Expr ")";
Int    → 100% Digit Int | 0% Digit;
Digit  → 14.3% "0" | 0% "1" | 0% "2" | 0% "3"
       | 14.3% "4" | 14.3% "5" | 14.3% "6"
       | 14.3% "7" | 14.3% "8" | 14.3% "9";
```

用于表示I产生的输入分布的概率文法G_p

表示与I相反分布的互补文法G_p^{-1}

[1] Soremekun E, Pavese E, Havrikov N, et al. Inputs from hell learning input distributions for grammar-based test generation[J]. IEEE Transactions on Software Engineering, 2020.

基于生成的模糊测试

- Generation-based：Inputs from Hell (TSE'19)[1]
  - 思想：挖掘已有的测试输入，得到现有的测试输入分布。根据该分布进行输入生成以得到预期的测试输入

```
Expr → 0% Term | 0% Expr "+" Term
     | 100% Expr "−" Term;
Term → 0% Factor | 0% Term "*" Factor
     | 100% Term "/" Factor;
Factor → 0% Int | 50% "+" Factor
     | 50% "−" Factor | 0% "(" Expr ")";
Int → 100% Digit Int | 0% Digit;
Digit → 14.3% "0" | 0% "1" | 0% "2" | 0% "3"
     | 14.3% "4" | 14.3% "5" | 14.3% "6"
     | 14.3% "7" | 14.3% "8" | 14.3% "9";
```

Input Generation

```
+5 / −5 / 7 − +0 / 6 / 6 − 6 / 8 − 5 − 4
−4 / +7 / 5 − 4 / 7 / 4 − 6 / 0 − 5 − 0
+5 / ++4 / 4 − 8 / 8 − 4 / 8 / 7 − 8 − 9
−6 / 9 / 5 / 8 − +7 / −9 / 6 − 4 − 4 − 6
+8 / ++8 / 5 / 4 / 0 − 5 − 4 / 8 − 8 − 8
−9 / −5 / 9 / 4 − −9 / 0 / 5 − 8 / 4 − 6
++7 / 9 / 5 − +8 / +9 / 7 / 7 − 6 − 8 − 4
−+6 / −8 / 9 / 6 − 5 / 0 − 5 − 8 − 0 − 5
```

表示与I相反分布的互补文法G_p^{-1}

互补文法G_p^{-1}产生的相反输入

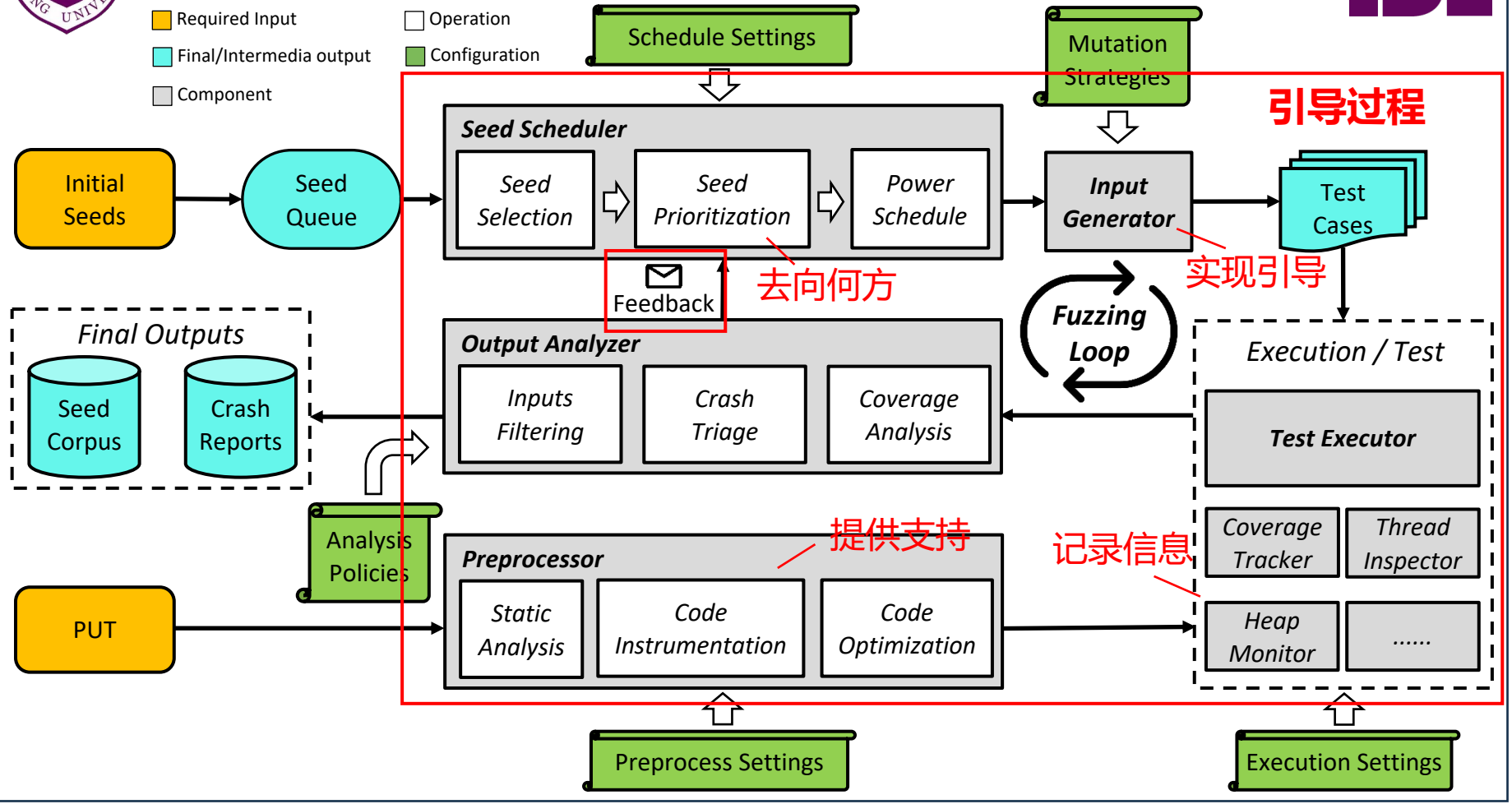
[1] Soremekun E, Pavese E, Havrikov N, et al. Inputs from hell learning input distributions for grammar-based test generation[J]. IEEE Transactions on Software Engineering, 2020.

按照引导方式分类

- 按照**引导方式**
  - Test Execution、Output Analysis
  - Search-based：将测试转化为**搜索问题**，以代码覆盖率作为指示器、以启发式算法（类遗传算法）为核心，将测试导向更高覆盖的方向 → CGF
  - Gradient-based：将测试转化为**优化问题**，以最大化缺陷发掘输入为目标构建目标函数，迭代求最优解 → 退而求覆盖率
  - 模糊测试中的**马太效应**（Matthew effect）[1]：好的种子输入能够演化产生品质良好的子代测试输入

[1] Chen H, Guo S, Xue Y, et al. MUZZ: Thread-aware grey-box fuzzing for effective bug hunting in multithreaded programs[C]//29th USENIX Security Symposium (USENIX Security 20). 2020: 2325-2342.

按照引导方式分类

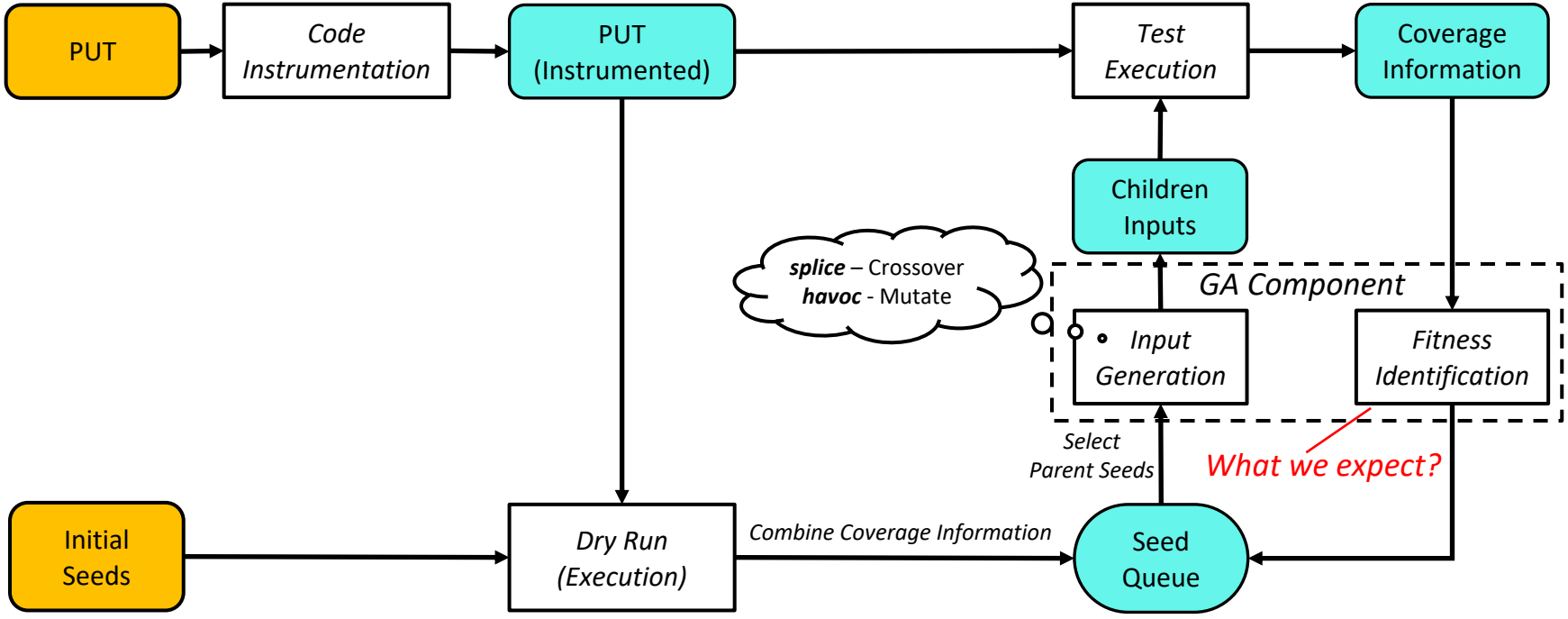

Required Input
Operation
Final/Intermedia output
Configuration
Component
Schedule Settings
Mutation Strategies
引导过程
Seed Scheduler
Initial Seeds
Seed Queue
Seed Selection
Seed Prioritization
Power Schedule
Input Generator
Test Cases
Feedback
去向何方
实现引导
Fuzzing Loop
Final Outputs
Seed Corpus
Crash Reports
Output Analyzer
Inputs Filtering
Crash Triage
Coverage Analysis
Execution / Test
Test Executor
Coverage Tracker
Thread Inspector
Heap Monitor
.....
Analysis Policies
提供支持
记录信息
Preprocessor
PUT
Static Analysis
Code Instrumentation
Code Optimization
Preprocess Settings
Execution Settings

基于搜索的模糊测试

- Search-based：Fuzzing as Search Problem
  - 核心：将模糊测试过程建模为搜索问题，根据模型构造启发式算法（Heuristic）解决问题
  - 启发式：遗传算法-AFL[1]、马尔科夫链-AFLFast[2]、信息熵-Entropic[3]、多臂老虎机问题-EcoFuzz[4]

[1] AFL README：https://github.com/google/AFL/blob/master/README.md
[2] Böhme M, Pham V T, Roychoudhury A. Coverage-based greybox fuzzing as markov chain[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016: 1032-1043.
[3] Böhme M, Manès V J M, Cha S K. Boosting fuzzer efficiency: An information theoretic perspective[C]//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020: 678-689.
[4] Yue T, Wang P, Tang Y, et al. {EcoFuzz}: Adaptive {Energy-Saving} Greybox Fuzzing as a Variant of the Adversarial {Multi-Armed} Bandit[C]//29th USENIX Security Symposium (USENIX Security 20). 2020: 2307-2324.

- Search-based：AFL，基于插装的遗传算法



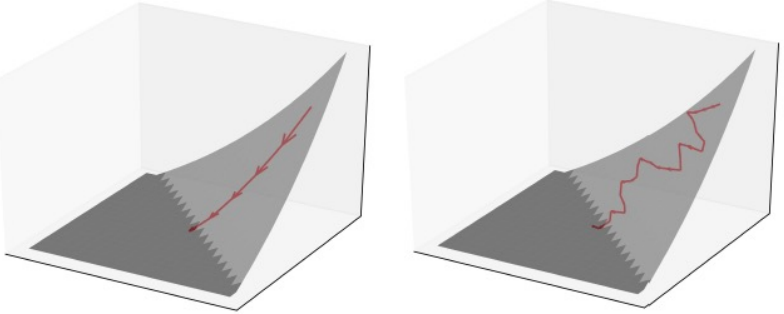
基于梯度的模糊测试

- Gradient-based：Fuzzing as Optimization Problem
  - 核心：将模糊测试过程建模为优化问题，问题的目标是最大化缺陷发掘数量，利用梯度下降算法持续求解最优解
  - 目标退阶：缺陷离散分布且无法预知 → 替换为代码覆盖
  - 应用DL技术-Neuzz[1] & MTFuzz[2]；利用梯度下降取代符号执行的约束求解过程-Angora[3]

[1] She D, Pei K, Epstein D, et al. Neuzz: Efficient fuzzing with neural program smoothing[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 803-817.
[2] She D, Krishna R, Yan L, et al. MTFuzz: fuzzing with a multi-task neural network[C]//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020: 737-749.
[3] Chen P, Chen H. Angora: Efficient fuzzing by principled search[C]//2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018: 711-725.

基于梯度的模糊测试

- Gradient-based：Neuzz[1]
  - 核心思路：将模糊测试建模为无约束优化问题，利用梯度下降算法优化模糊测试过程
  - 动机：Gradient Descend > Evolutionary Algorithm



(a) gradient descent　　(b) evolutionary algorithm

梯度下降和演进算法在高阶优化问题上面的表现

[1] She D, Pei K, Epstein D, et al. Neuzz: Efficient fuzzing with neural program smoothing[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 803-817.

基于梯度的模糊测试

- Gradient-based：Neuzz[1]
  - 挑战
    - 如何计算反馈 → 如何计算梯度
    - 优化目标（缺陷发现数目或覆盖率）是**不连续的**，无法直接适配梯度下降算法 → 引入程序平滑（Program Smoothing）技术

[1] She D, Pei K, Epstein D, et al. Neuzz: Efficient fuzzing with neural program smoothing[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 803-817.

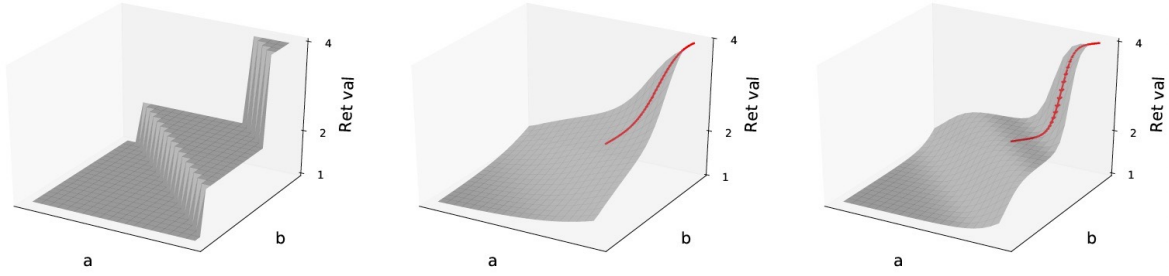
基于梯度的模糊测试

- Gradient-based：Neuzz[1]
  - 程序平滑（Neural Program Smoothing）：消除目标函数中的不连续性
    - 黑盒程序平滑：简单易用，但是会产生较大的近似错误
    - 白盒程序平滑：依赖符号执行（Symbolic Execution）和抽象解释技术（Abstract），会产生较大的额外开销

[1] She D, Pei K, Epstein D, et al. Neuzz: Efficient fuzzing with neural program smoothing[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 803-817.

基于梯度的模糊测试

- Gradient-based：Neuzz[1]
  - 程序平滑（Neural Program Smoothing）：消除目标函数中的不连续性
    - **神经程序平滑**（Neural Program Smoothing）：利用NN模型模拟程序行为



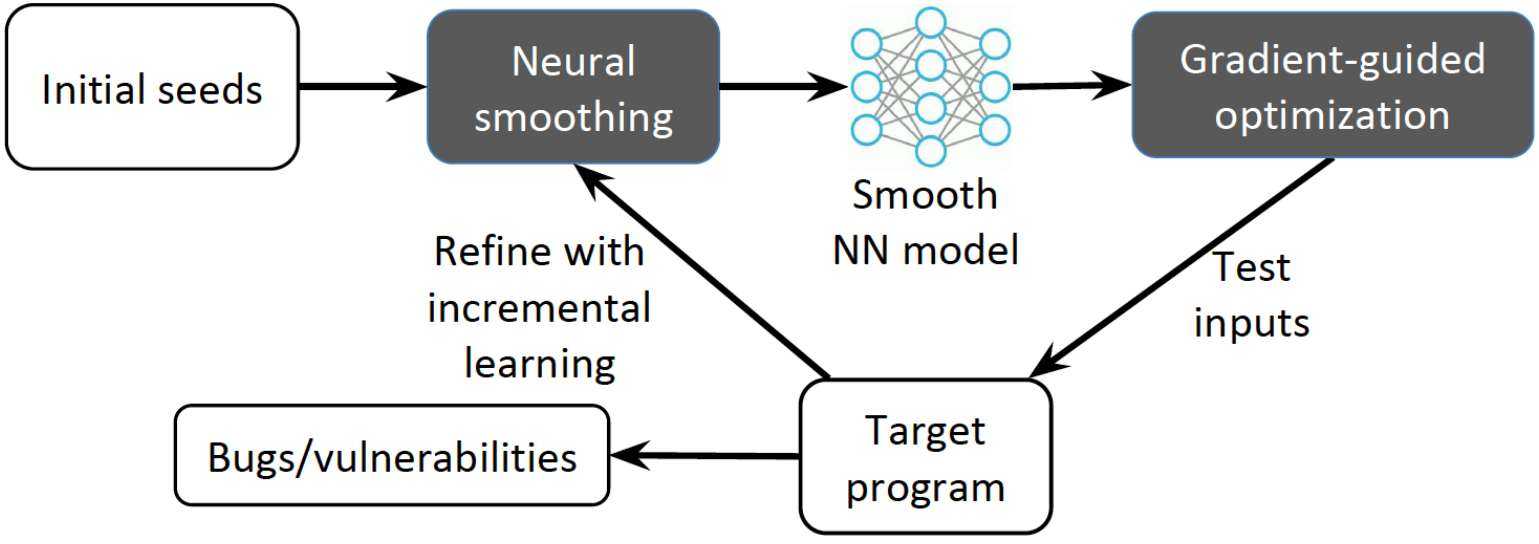
(a) Original (b) NN smoothing (c) NN smoothing + refining

Neuzz神经程序平滑

[1] She D, Pei K, Epstein D, et al. Neuzz: Efficient fuzzing with neural program smoothing[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 803-817.

基于梯度的模糊测试

- Gradient-based：Neuzz[1]



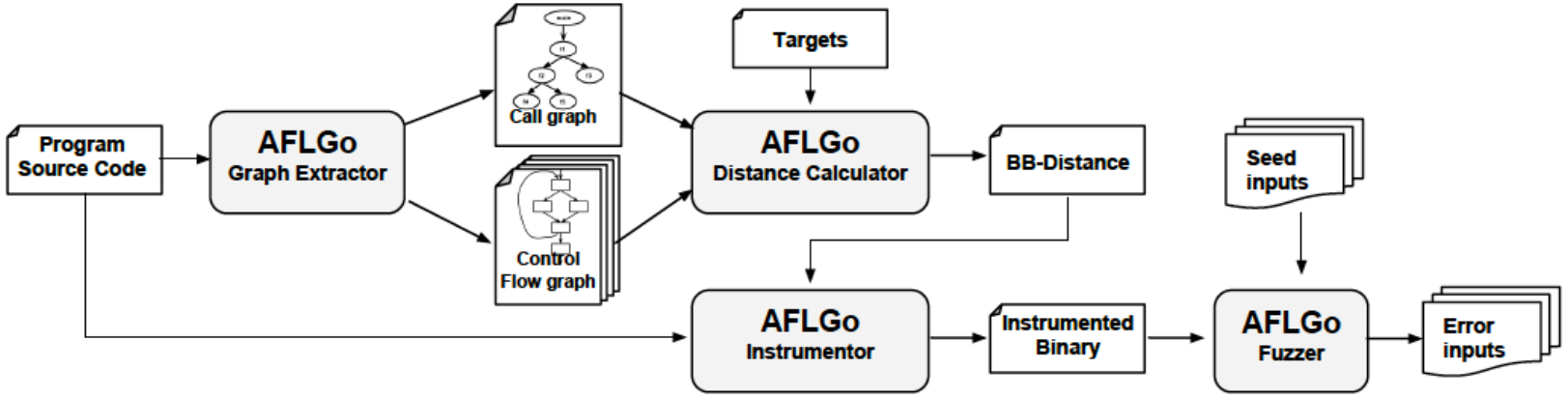
Neuzz工作流程概览

[1] She D, Pei K, Epstein D, et al. Neuzz: Efficient fuzzing with neural program smoothing[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 803-817.

按照测试目的分类

- 按照**测试的目的**
  - 非定向模糊测试：**Wider and Deeper**
    - 目标：验证程序的正确性，检测程序中潜在的缺陷
  - 定向模糊测试：**Directed and Targeted**
    - 目标：针对程序中的某个目标位置进行快而有效的测试
    - 场景：缺陷复现、补丁检验、静态分析报告验证
    - 分类：白盒、**灰盒**

- AFLGo[1]：定向灰盒模糊测试
  - 基本思路：Distance + Annealing-based Scheduling，为更靠近目标位置的种子分配更多的能量



AFLGo整体流程

[1] Böhme M, Pham V T, Nguyen M D, et al. Directed greybox fuzzing[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2329-2344.

定向灰盒模糊测试

- AFLGo[1]：定向灰盒模糊测试
  - 基本思路：Distance + Annealing-based Scheduling，为更靠近目标位置的种子分配更多的能量

$$d_b(m, T_b) = \begin{cases} 0 & \text{if } m \in T_b \\ c \cdot \min\limits_{n \in N(m)} (d_f(n, T_f)) & \text{if } m \in T \\ \left[\sum\limits_{t \in T} (d_b(m,t) + d_b(t, T_b))^{-1} \right]^{-1} & \text{otherwise} \end{cases}$$

$$d(s, T_b) = \frac{\sum_{m \in \xi(s)} d_b(m, T_b)}{|\xi(s)|} \qquad \tilde{d}(s, T_b) = \frac{d(s, T_b) - \text{minD}}{\text{maxD} - \text{minD}}$$

AFLGo距离计算

$$T_{\text{exp}} = T_0 \cdot \alpha^k \tag{7}$$

$$0.05 = \alpha^{k_x} \quad \text{for } T_{\text{exp}} = 0.05; k = k_x \text{ in Eq. (7)} \tag{8}$$

$$k_x = \log(0.05)/\log(\alpha) \quad \text{solving for } k_x \text{ in Eq. (8)} \tag{9}$$

$$T_{\text{exp}} = \alpha^{\frac{t}{t_x}\frac{\log(0.05)}{\log(\alpha)}} \quad \text{for } k = \frac{t}{t_x} k_x \text{ in Eq. (7)} \tag{10}$$

$$= 20^{-\frac{t}{t_x}} \quad \text{simplifying Eq. (10)} \tag{11}$$

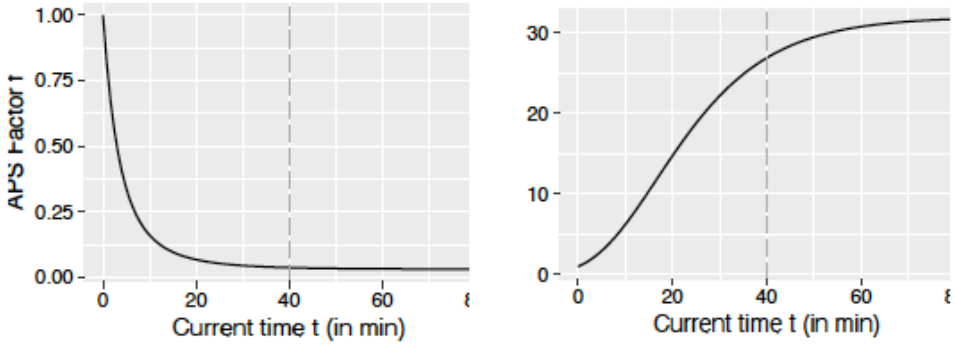
$$p(s, T_b) = (1 - \tilde{d}(s, T_b)) \cdot (1 - T_{\text{exp}}) + 0.5 T_{\text{exp}} \tag{12}$$

AFLGo模拟退火调度

[1] Böhme M, Pham V T, Nguyen M D, et al. Directed greybox fuzzing[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2329-2344.

定向灰盒模糊测试

- AFLGo[1]：定向灰盒模糊测试
  - 基本思路：Distance + Annealing-based Scheduling，为更靠近目标位置的种子分配更多的能量



(a) Distance $\tilde{d}(s, T_b) = 1$ (b) Distance $\tilde{d}(s, T_b) = 0$

能量分配趋势

[1] Böhme M, Pham V T, Nguyen M D, et al. Directed greybox fuzzing[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2329-2344.

04

iSE模糊测试

- Predoo：基于模糊测试的深度学习算子精度测试，ISSTA'21
- Duo：基于差分测试的深度学习算子精度测试，TR'21
- Gandalf：基于生成的深度学习框架模糊测试，TBD
- QATest：问答系统模糊测试框架，ASE'22
- **探索如何利用变异测试优化模糊测试**，Internetware'22
- **AFLFun：利用函数重要度优化灰盒模糊测试**，TBD

- **变异测试+模糊测试**：Investigating Coverage Guided Fuzzing with Mutation Testing， Internetware'22
- **动机1：现有CGF技术存在不足**
  - 现有的（灰盒）覆盖率引导的模糊测试往往只关注如何提升代码覆盖率
  - 覆盖更多的代码并不意味着能触发更多的Bug → PIE模型
  - 仅关注分支覆盖可能使得Fuzzer错过能够发现缺陷的输入，从而会损害技术的有效性

- **变异测试+模糊测试**：Investigating Coverage Guided Fuzzing with Mutation Testing， Internetware'22

```
1  int foo(int x, int y) {
2      if (x > y)
3          return x;
4      else
5          return x; // Should return y.   Bug
6  }
```

代码片段示例

先生成：保留

$i_1 = < 1, 1 >$ $expect = 1, o_1 = 1$

$i_2 = < 1, 2 >$ $expect = 2, o_2 = 1$

后生成: 抛弃

- **变异测试+模糊测试**：Investigating Coverage Guided Fuzzing with Mutation Testing， Internetware'22
- **动机2：变异测试的优势**
  - 变异测试/分析技术能够从缺陷的角度出发对测试输入进行评估，能够帮助Fuzzer识别具有缺陷暴露（Fault-revealing）能力的测试输入
  - 借助**马太效应**我们可以在模糊测试过程中演化生成具有更强缺陷暴露能力的测试输入

- **挑战**：变异测试和模糊测试存在天然的冲突
  - 问题1：如何保证执行的效率？

CID	Fuzz Case	#Mutants	LoC of Bench.
C01	SortingFuzz	69	88
C02	MatrixInverseFuzz	75	65
C03	SuffixArrayFuzz	215	219
C04	SimpleRegressionFuzz	49783	208891
C05	DivFuzz	577	30396

小型测试目标产生的大量测试用例

- **挑战**：变异测试和模糊测试存在天然的冲突
  - 问题1：如何保证执行的效率？
  - 问题2：如何判断变异杀死？

```
@Test(timeout = 4000)
public void test69() throws Throwable {
    IntegerBloomFilter integerBloomFilter0 = new IntegerBloomFilter(2242.643, (-421));
    integerBloomFilter0.clear();
    integerBloomFilter0.clear();
    int int0 = integerBloomFilter0.getFilterSize();
    assertEquals(6736, int0);
    double double0 = integerBloomFilter0.getExpectedFalsePositiveProbability();
    assertEquals((-16), integerBloomFilter0.getBitsPerElement());
    assertEquals(Double.POSITIVE_INFINITY, integerBloomFilter0.getCurrentFalsePositiveProbability(), 0.01);
    assertEquals(2180.0231231218227, double0, 0.01);
}
```

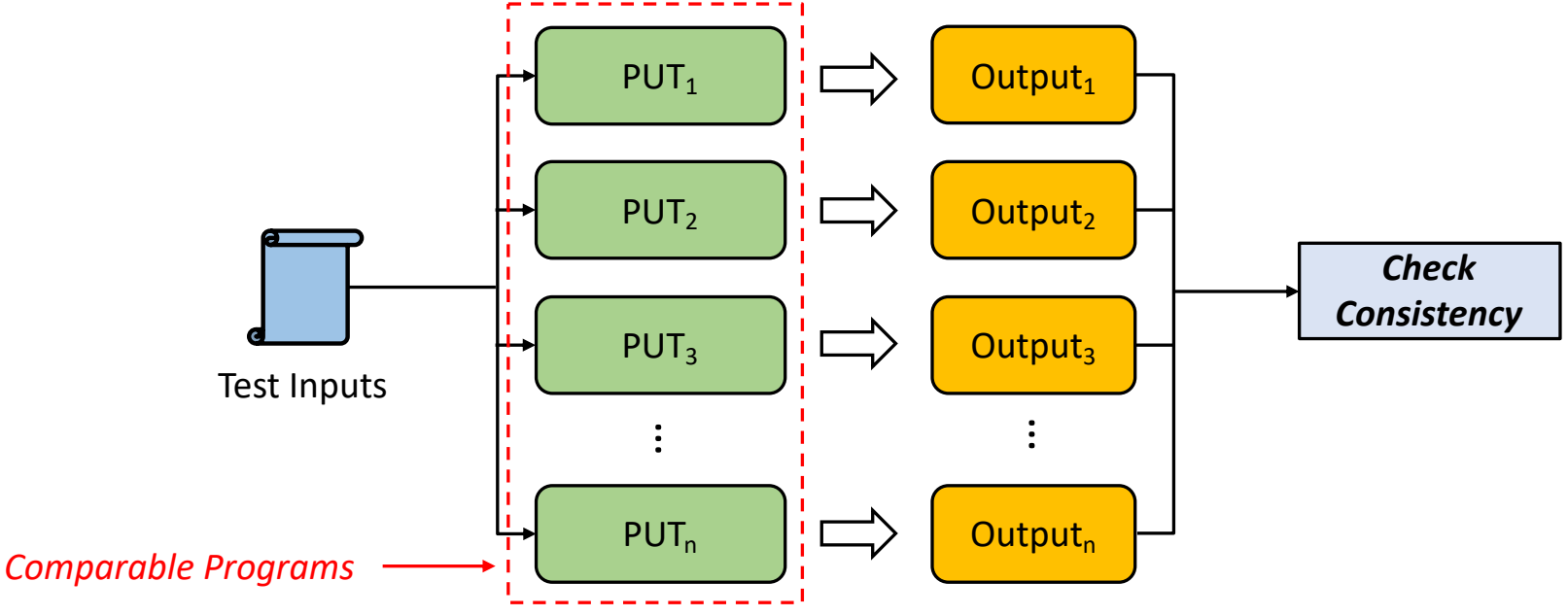
PIT要求的用于判断变异杀死的Junit断言

- **挑战**：变异测试和模糊测试存在天然的冲突
  - 问题1：如何保证执行的效率？ → 变异体筛选 + 并行执行
  - 问题2：如何判断变异杀死？ → 差分测试

```
@Test(timeout = 4000)
public void test69() throws Throwable {
    IntegerBloomFilter integerBloomFilter0 = new IntegerBloomFilter(2242.643, (-421));
    integerBloomFilter0.clear();
    integerBloomFilter0.clear();
    int int0 = integerBloomFilter0.getFilterSize();
    assertEquals(6736, int0);
    double double0 = integerBloomFilter0.getExpectedFalsePositiveProbability();
    assertEquals((-16), integerBloomFilter0.getBitsPerElement());
    assertEquals(Double.POSITIVE_INFINITY, integerBloomFilter0.getCurrentFalsePositiveProbability(), 0.01);
    assertEquals(2180.0231231218227, double0, 0.01);
}
```

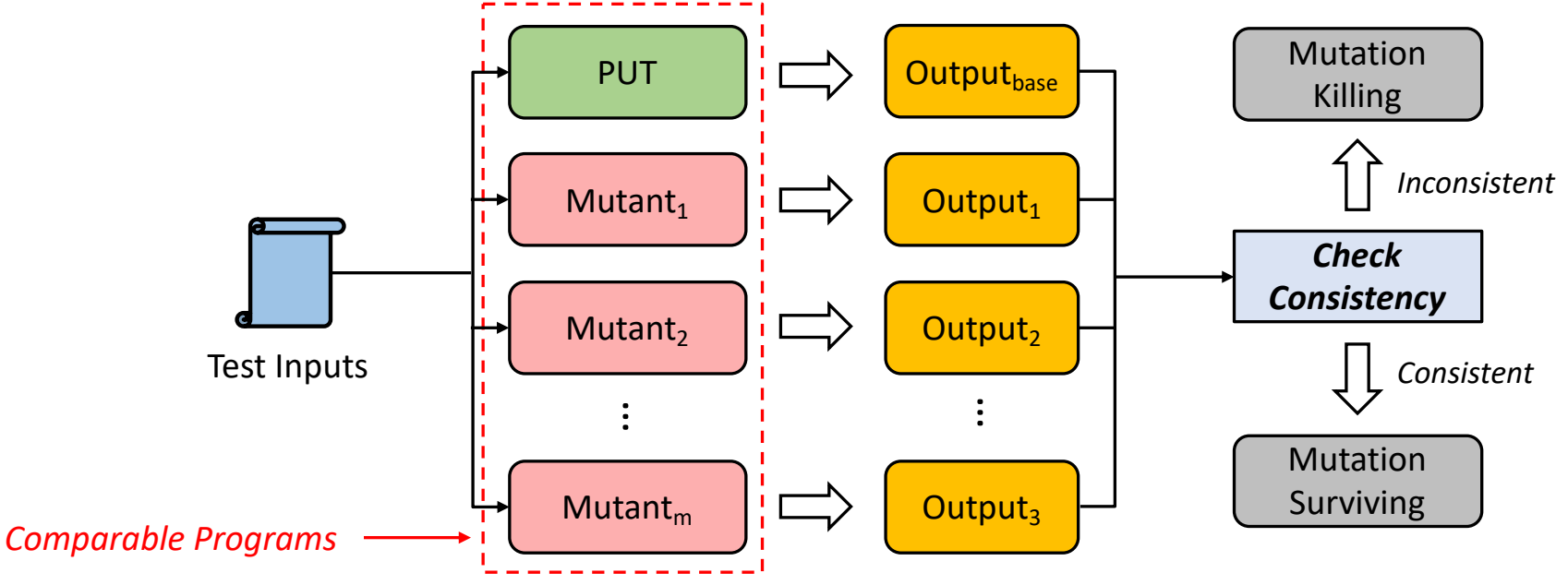
PIT要求的用于判断变异杀死的Junit断言

- **差分测试**：为变异测试提供测试预言



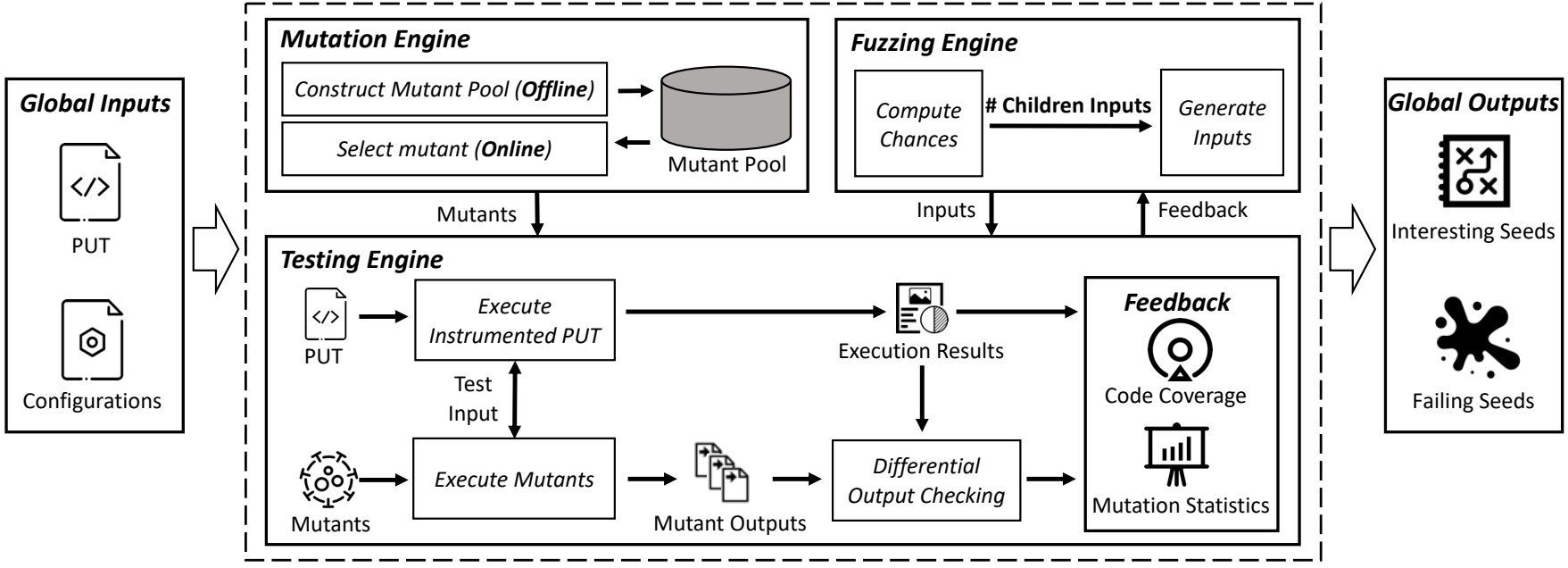
差分测试一般框架

- **差分测试**：为变异测试提供测试预言



核心组件：差分输出比较

变异测试+模糊测试



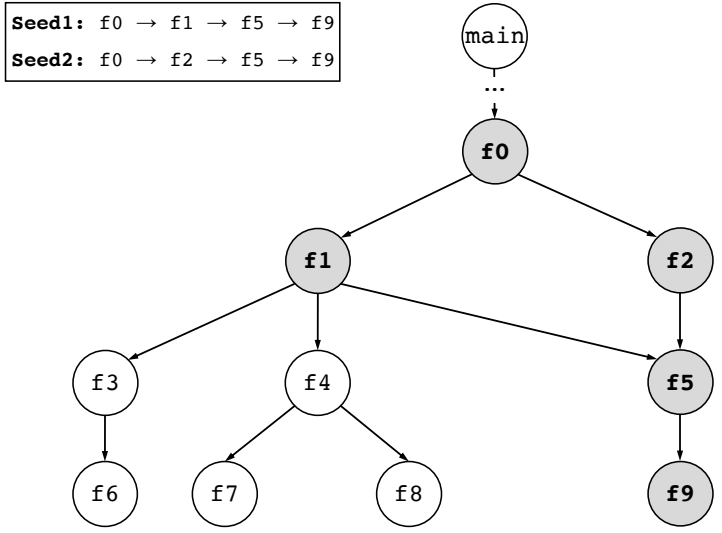
结合变异测试的覆盖引导的灰盒模糊测试框架

函数重要度+模糊测试

- AFLFun：利用函数重要度增强灰盒模糊测试
  - **动机**：待测函数亦有差距！不同待测函数的重要度不同

Caller	Callee	First Occur
parse_statement (total:27)	parse_return	mjs.c:13329
	parse_if	mjs.c:13344
...	...	...
parse_if (total: 27)	cs_log_print_prefix	mjs.c:12893
	cs_log_printf	mjs.c:12893
	mjs_set_errorf	mjs.c:12894
	pnext	mjs.c:12894
	parse_expr	mjs.c:12896
	emit_byte	mjs.c:12898
	emit_init_offset	mjs.c:12898
	parse_block_or_stmt	mjs.c:12903
	mjs_bcode_insert_offset	mjs.c:12920
...	...	...
parse_return (total: 7)	mjs_set_errorf	mjs.c:12941
	cs_log_print_prefix	mjs.c:12941
	cs_log_printf	mjs.c:12941
	pnext	mjs.c:12941
	parse_expr	mjs.c:12941
	emit_byte	mjs.c:12942
...	...	...

Mjs项目中存在的调用关系不均衡现象



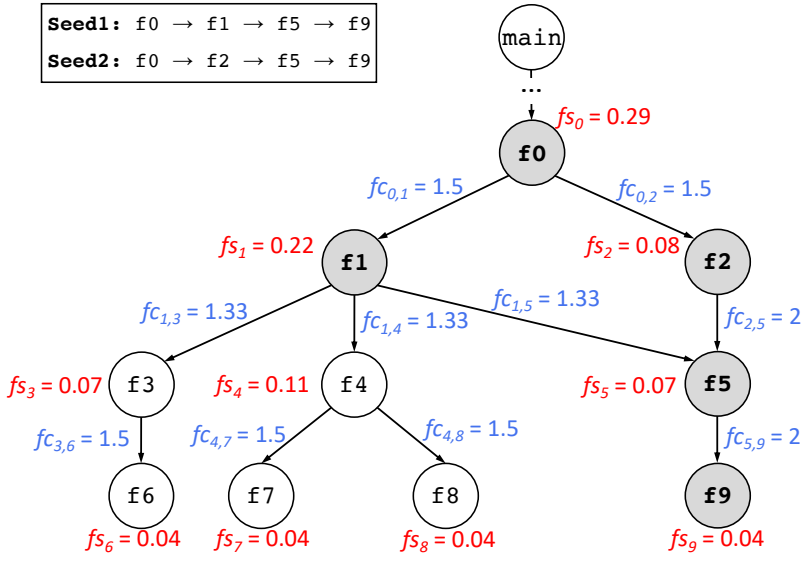
种子对调用图的覆盖

函数重要度+模糊测试

- AFLFun：利用函数重要度增强灰盒模糊测试
  - **挑战**：量化函数重要度，为执行重要函数的种子分配更多资源

Caller	Callee	First Occur
parse_statement (total:27)	parse_return	mjs.c:13329
	parse_if	mjs.c:13344
...	...	...
parse_if (total: 27)	cs_log_print_prefix	mjs.c:12893
	cs_log_printf	mjs.c:12893
	mjs_set_errorf	mjs.c:12894
	pnext	mjs.c:12894
	parse_expr	mjs.c:12896
	emit_byte	mjs.c:12898
	emit_init_offset	mjs.c:12898
	parse_block_or_stmt	mjs.c:12903
	mjs_bcode_insert_offset	mjs.c:12920
...	...	...
parse_return (total: 7)	mjs_set_errorf	mjs.c:12941
	cs_log_print_prefix	mjs.c:12941
	cs_log_printf	mjs.c:12941
	pnext	mjs.c:12941
	parse_expr	mjs.c:12941
	emit_byte	mjs.c:12942
...	...	...

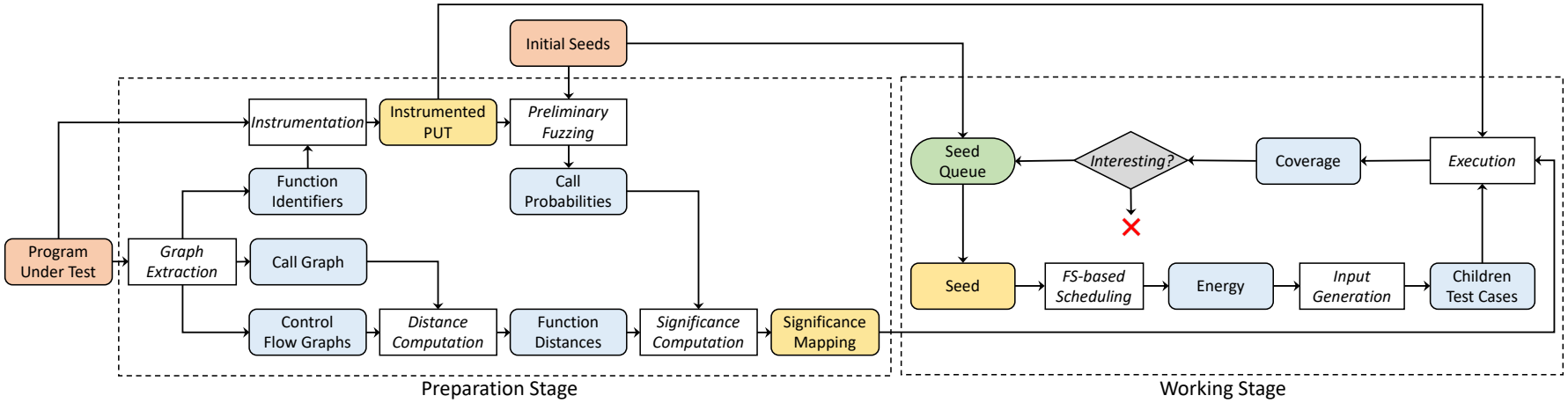
Mjs项目中存在的调用关系不均衡现象



重要度感知的函数调用图

函数重要度+模糊测试

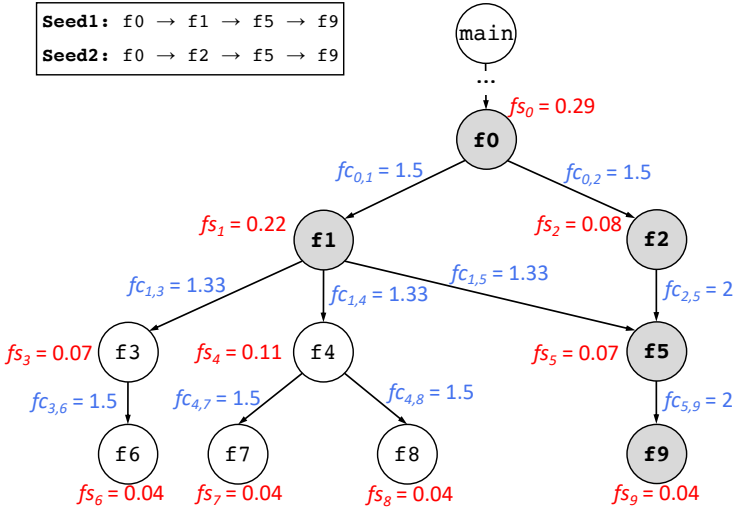
- AFLFun：流程框架



函数重要度感知的灰盒模糊测试流程框架

函数重要度+模糊测试

- AFLFun：计算重要度（FS，Function Significance）
  - **动静态结合**：采用静态函数距离和动态调用概率描述函数关系
  - **中心度分析**：采用PageRank算法推动重要度流向处于中心函数



重要度感知的函数调用图

$$d(f_s, f_t) = \Phi(f_s, f_t) \cdot \Psi(f_s, f_t) \tag{4}$$

$$p(f_s, f_t) = \frac{v(f_s)}{\sum_{\forall f_i \in \mathbb{F}} v(f_i)} \cdot \frac{v(f_s, f_t)}{\sum_{\forall f_i \in L(f_s)} v(f_s, f_i)} \tag{5}$$

提取动静态程序特征

$$\tilde{d}(f_s, f_t) = \frac{d(f_s, f_t) - \text{minD}}{\text{maxD} - \text{minD}} \tag{6}$$

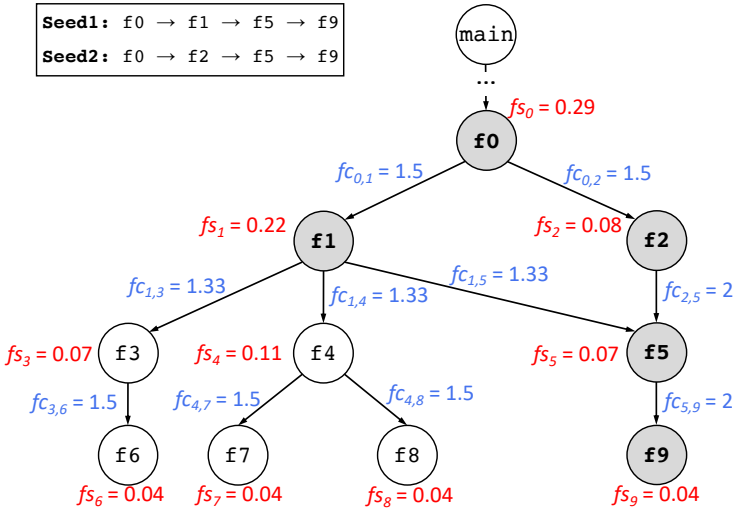
$$\tilde{p}(f_s, f_t) = \frac{p(f_s, f_t) - \text{minP}}{\text{maxP} - \text{minP}} \tag{7}$$

$$r(f_s, f_t) = \beta \cdot (1 - \tilde{d}(f_s, f_t)) + (1 - \beta) \cdot \tilde{p}(f_s, f_t) \tag{8}$$

描述函数关系

函数重要度+模糊测试

- AFLFun：计算重要度（FS，Function Significance）
  - **动静态结合**：采用静态函数距离和动态调用概率描述函数关系
  - **中心度分析**：采用PageRank算法推动重要度流向处于中心函数



重要度感知的函数调用图

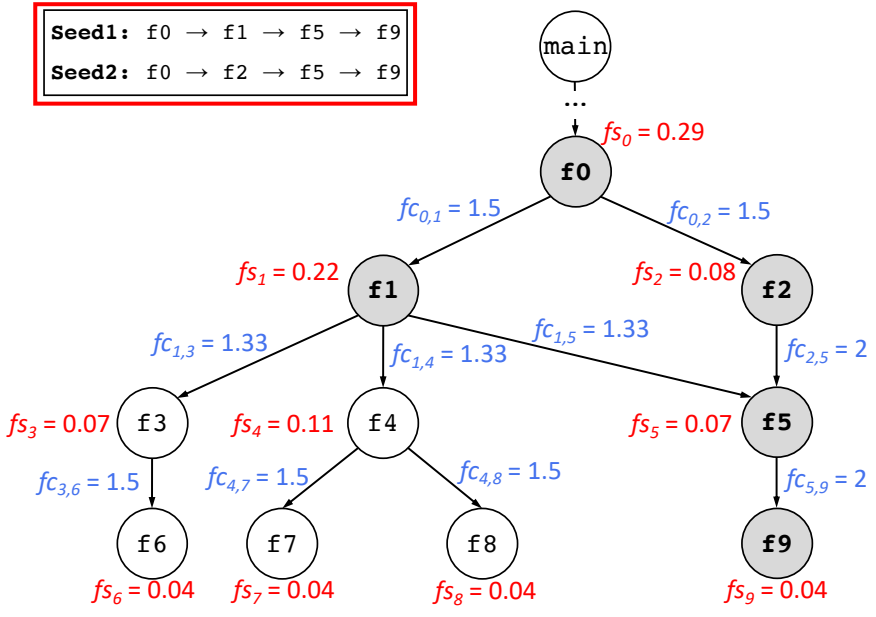
$$m_{ij} = \begin{cases} 0 & \text{if } f_j \notin L(f_i) \\ \frac{r(f_i,f_j)}{\sum_{f_k \in L(f_i)} r(f_i,f_k)} & \text{otherwise} \end{cases} \quad (10)$$

$$FS(f_i) = (1-\alpha) + \alpha\left(\sum_{f_j \in L(f_i)} \frac{FS(f_j)}{|B(f_j)|}\right) \quad (11)$$

计算函数重要度

函数重要度+模糊测试

- AFLFun：能量调度



重要度感知的函数调用图

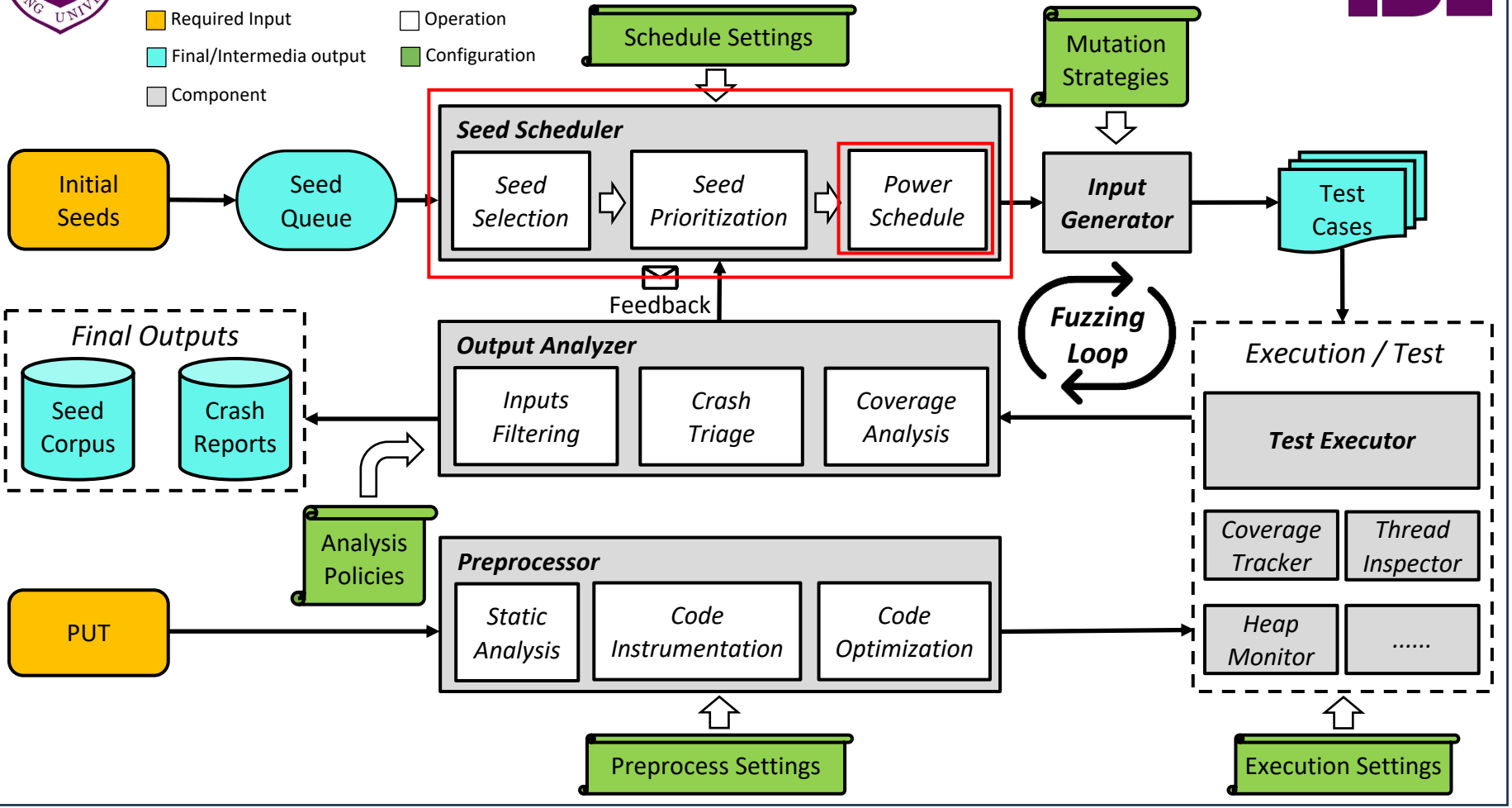
Algorithm 2 FS-based Power Scheduling

Input: Seed s, Significance Mapping M
Output: Assigned Energy e

1: **global variables**
2: $maxS \leftarrow$ MIN, $minS \leftarrow$ MAX
3: **end global variables**
4: $B \leftarrow$ GETCOVEREDBBS(s)
5: $e \leftarrow$ ASSIGNBYBB(B)
6: **if** ISFAVORED(s) **then**
7: $e \leftarrow$ AMPLIFYFAVORED(e)
8: **end if**
9: $F \leftarrow$ GETEXERCISEDFUNCTIONS(s)
10: $sig \leftarrow 0$
11: **for** f in F **do** ▷ *Accumulate FS values*
12: $sig \leftarrow sig +$ LOOKUPFS(f, M)
13: **end for**
14: **if** $sig > maxS$ **then** ▷ *Update globals*
15: $maxS \leftarrow sig$
16: **else if** $sig < minS$ **then**
17: $minS \leftarrow sig$
18: **end if**
19: $\rho \leftarrow$ COMPUTEPOWERFACTOR(sig, $maxS$, $minS$)
20: $e \leftarrow$ ADJUSTSIGNIFICANT(e, ρ)
21: **return** e

伪代码：基于函数重要度的能量调度

模糊测试框架


Required Input
Operation
Final/Intermedia output
Configuration
Component
Schedule Settings
Mutation Strategies
Seed Scheduler
Seed Selection
Seed Prioritization
Power Schedule
Initial Seeds
Seed Queue
Input Generator
Test Cases
Feedback
Fuzzing Loop
Final Outputs
Seed Corpus
Crash Reports
Output Analyzer
Inputs Filtering
Crash Triage
Coverage Analysis
Execution / Test
Test Executor
Coverage Tracker
Thread Inspector
Heap Monitor
.....
Analysis Policies
Preprocessor
Static Analysis
Code Instrumentation
Code Optimization
PUT
Preprocess Settings
Execution Settings

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

Thank you!