《2024 软件测试》大作业要求

—— 变异测试 & 模糊测试方向选题

文献综述选题

1、变异测试选题

（1）变异测试优化技术综述

分析并讨论现有工作是从哪些入手（如：变异体的选择、约简、执行、分析），采用何种方法优化变异测试技术的。

（2）变异测试应用综述

变异测试在众多垂直领域（如回归测试优化、测试生成、AI 测试等）领域均有应用。试分析并总结这些相关研究的规律并给出有趣的见解。

（3）变异技术综述

变异技术是自动化测试中最常见也是最有效的技术之一，在变异测试、模糊测试、基于搜索的测试等领域中均有应用。试总结归纳变异技术在现有研究工作中的研究动机和设计思路，辨析不同研究工作的侧重点。

2、模糊测试选题

（1）模糊测试技术中[特定环节]技术综述

[特定环节]部分主要参考课上介绍的框架图，包括种子调度、种子排序、种子选择、测试生成、能量调度、插装组件等，也可以是更粗粒度的环节，如预处理技术（种子选择、插装、Sanitizer 技术等）、后处理技术（漏洞分析、种子复现、种子库复用等）。选择本题目的小组在完成作业和填写选题时应当将[特定环节]**替换**成具体的内容。

（2）定向模糊测试技术综述

定向模糊测试是模糊测试的重要分支。与通用的模糊测试技术不同，定向模糊测试更加关注如何更快到达待测程序的特定区域（称为目标点），被广泛应用于漏洞复现、补丁验证、静态报告验证等关键领域。关于定向模糊测试的相关知识可以参考这篇论文。

（3）[变异式|生成式]模糊测试技术综述

变异式（Mutation-based）和生成式（Generation-based）是模糊测试主流的测

试生成方法，前者关注生成的效率，后者侧重生成的有效性。试总结现有工作的研究思路和研究动机，提出自己的见解。

（4）[垂直领域]模糊测试技术综述

模糊测试以其强大的漏洞挖掘能力和高可拓展性著称，被广泛应用在多个垂直领域。在本选题中，[垂直领域]可以填入内核（Linux Kernel）、嵌入式系统（Internet of Things, IoT）、固件（Firmware）、协议（Protocol）、RESTful API 等。选题小组可以自行检索并选定感兴趣的垂直领域开展综述

代码实现选题一：覆盖率引导的变异式模糊测试工具

2023 软件测试搭建的模糊测试 Demo，供参考：Github、Gitee

1 题目概述

选题小组应参考 SOTA 模糊测试工具 AFL++，使用 Python 或 Java 语言（如果想使用其他语言请发邮件和助教沟通）实现自己的~~简化版~~的覆盖率引导的变异式模糊测试工具。该工具应包含 7 个组件（其中 6 个需实现）：~~插装组件~~（直接使用 afl-cc 实现）、测试执行组件、执行结果监控组件、变异组件 、种子排序组件、能量调度组件、评估组件，工具框架图如下所示：

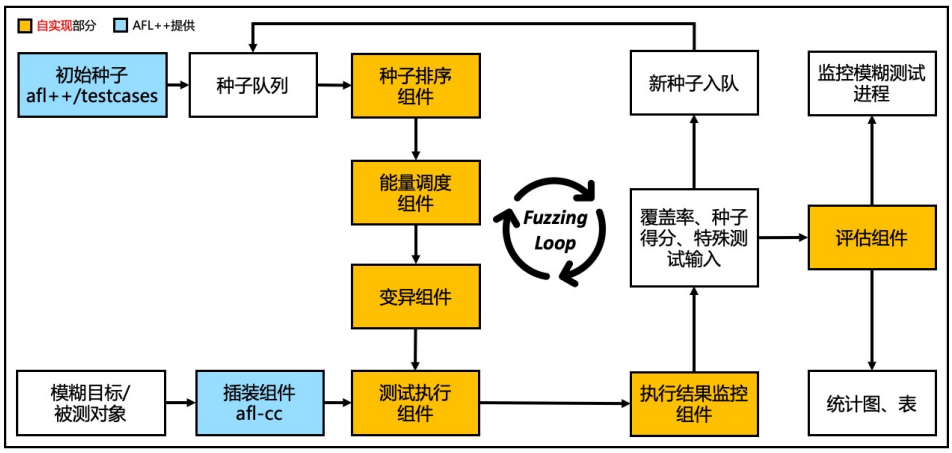

图 1 覆盖率引导的变异式模糊测试工具

2 实现要点

2.1 参考 AFL++实现

选题小组应首先学习 SOTA 模糊测试工具 AFL++的用法，在了解 AFL++使用方法的情况下实现自己的模糊测试工具。选题小组应掌握 afl-cc 和 afl-fuzz 的使用方法，并基本了解 AFL++插装原理（afl-cc.c 和 afl-compiler-rt.o.c）。

2.2 模糊目标/待测程序

本选题指定了 10 个模糊目标。选题小组应当在所有 10 个模糊目标程序上，使用自己的工具执行 24 小时的模糊测试，并绘制统计图表。统计图表的绘制由评估组件完成。

2.3 模糊目标的编译与插装

选题小组需要掌握基本的编译知识，学会使用 make（包含在 build-essential

中）、cmake 等构建工具，使用 afl-cc 对被测项目进行编译，完成对模糊目标的插装。编译环境应使用 Ubuntu 22.04，该环境下 llvm 的版本为 14.0.0。部分变异编译过程中会用到的依赖如下所示：

```
apt-get install -y libtool build-essential cmake python3 gcc llvm clang file binutils
```

C/C++项目编译的难度较大，通常需要迭代多次才能最终完成编译。选题小组应当仔细阅读项目文档，学会检索和使用大模型，必要时询问助教。一些编译时可能会用到的指令如下所示：

```
./autogen.sh # 生成 configure 文件

./configure --disable-shared # 生成 Makefile

cmake -S <src_dir> -B <build_dir> -G "Unix Makefiles" #使用 cmake 生成 Makefile

nm -C <target_bin> | grep afl # 检查二进制中是否包含 afl 符号

AFL_DEBUG=1 <target_bin> # 通过 afl debug 设置检验插装是否成功
```

2.4 种子排序组件

除按入队顺序选择（可以认为是不排序）外，至少再实现一种非纯随机排序方法，例如：按照覆盖率排序、按照执行时间排序、启发式排序等。详情可参考 afl-fuzz.c#main()函数中 perform_dry_run()之后的代码。

2.5 能量调度组件

实现启发式能量调度方法，至少能够依据覆盖率调整不同种子的能量，详情可参考 afl-fuzz.h#calculate_score()。

2.6 变异组件

实现 AFL 基础变异算子（bitflip、arith、interest、havoc、splice），详细说明见 AFL 文档#Stage Progress，代码可参考 afl-fuzz.h#fuzz_one()，afl-fuzz.h#fuzz_one_original()等函数。加分项：基于语法/结构的变异算子

2.7 测试执行组件

创建子进程，运行模糊目标，记录执行时间、次数等信息。C 的 fork()，Python 的 subprocess 库，Java 的 ProcessBuilder 类。可参考 afl-fuzz.h# fuzz_run_target()。

2.8 执行结果监控组件

基本要求：能够为工具打印日志，记录覆盖率、执行速度等统计数据，保存特殊测试用例等，实现效果参考 AFL++（图 2~图 5）。此外，还要注意学习和理解 AFL++是如何记录并传递覆盖率信息的（图 6）。

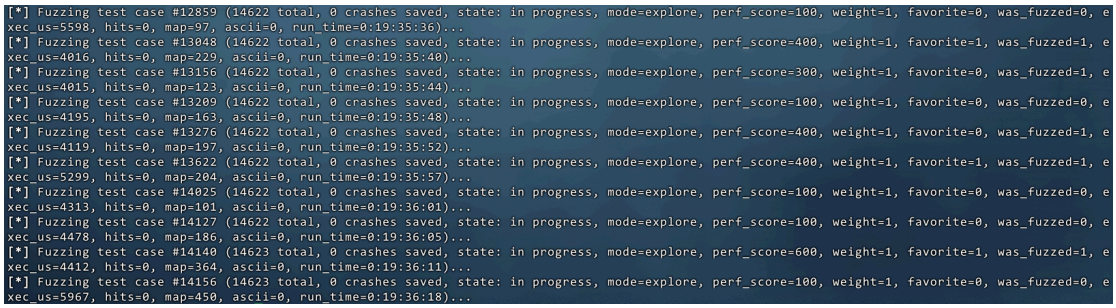

图 2 AFL++运行日志（AFL_NO_UI=1）

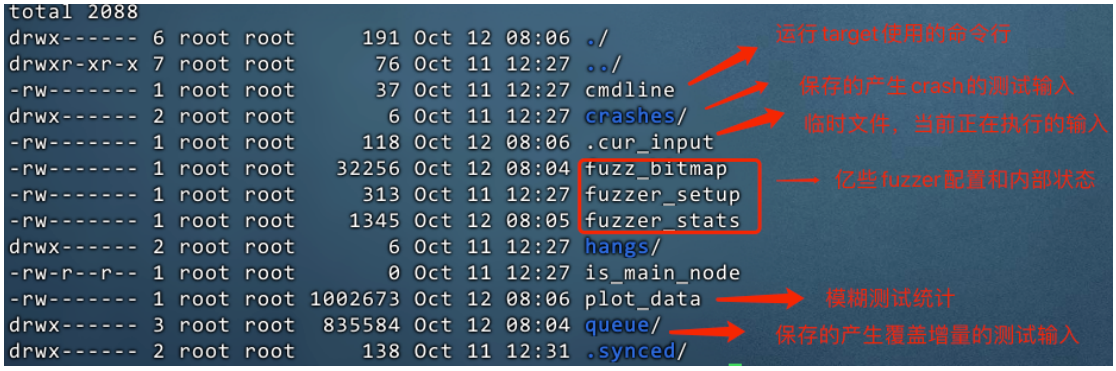

图 3 AFL++运行输出（-o 选项）

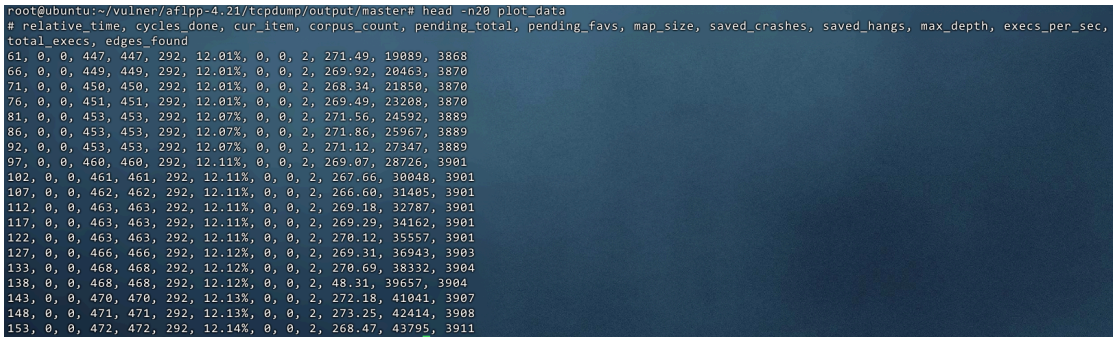

图 4 AFL++统计数据文件 plot_data

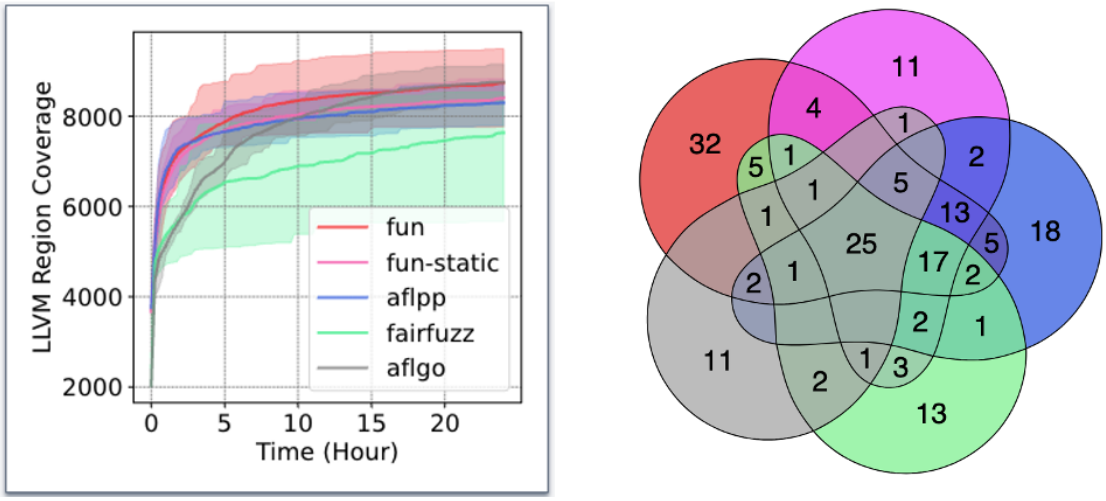

图 5 覆盖曲线图（左）和漏洞 Venn 图（右）举例

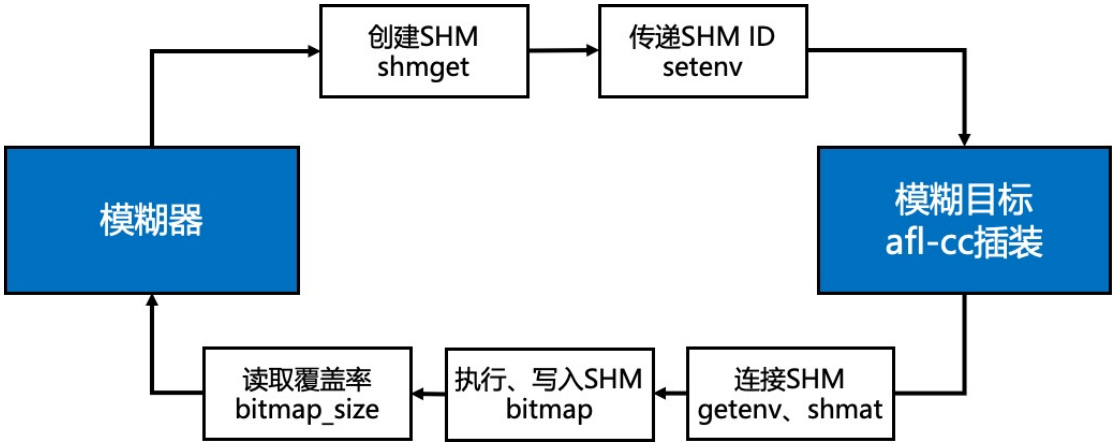

图 6 AFL++覆盖率收集原理示意图

3 验收内容

（1）源代码：业务代码和流程代码，如 Java、Python、Shell、R 等。

（2）开发日志：Markdown 文件（devlog.md）。记录开发过程中的进度安排、任务分配、遇到的难题和解决的过程等。注意描述的逻辑性。

（3）项目文档：Markdown 文件（README.md）。介绍项目的设计方案（架构、流程、类层次设计、使用方法等）和使用方法，可参考：仓库 1、仓库 2。描述应清晰、详实，按照“总-分”结构介绍。

（4）开箱即用的环境：使用 Docker，基础镜像为 ubuntu:22.04。提供可访问的 Dockerhub 链接，或是提供可本地一键构建的 Dockerfile。环境中应该囊括本选题指定的全部 10 个模糊目标（选题 Github 仓库）、每个模糊目标运行 24 小时之

后产生的测试数据（参考 AFL++的输出）、数据分析脚本等。选题小组提交的环境应该能够让绝大部分用户在配合 README.md 的情况下轻松使用工具和开展数据分析。

（5）评估结果：一组统计图表（pdf、png、csv 等），最少包括 10 张覆盖率曲线图，每张对应一个模糊目标。

（6）演示视频：演示工具的使用方法，控制在 8 分钟以内。视频上传到 B 站，视频链接贴在 README.md 里。

代码实现选题二：基于大模型的模糊测试驱动生成工具（探索向）

参考论文：2024-ISSTA[1]，2024-CCS[2]

1 题目概述

模糊驱动是测试执行的入口，是模糊测试的重要组件。然而，人工编写模糊测试驱动程序不仅耗时耗力，还需要大量的领域知识，要求编程人员需要对被测软件（通常是 C/C++标准库）有深入的了解。模糊驱动本质上是一段代码，而现代大模型又具备强大的代码生成能力。因此，本课题的主要目标就是探索如何驱动大模型自动生成模糊驱动，提升模糊测试的有效性（例如：提升代码覆盖）。

2 实现要点

基于大模型的模糊测试驱动生成的大致流程可参考图 7（简化自[1]-Figure2，Figure 3）。选题小组需要为选题 Github 仓库中提供的 libjpeg-turbo、libpng、libxml2 三个项目生成模糊驱动（加分项：支持其他项目，例如 libcap）。在完成作业的过程中，选题小组应首先开展前期调研，了解各种大模型（如 gpt-4，glm-4，codellama 等模型）的代码生成能力和交互方式，选定具备代码生成能力的大模型，进而围绕此大模型，实现基于大模型的模糊测试驱动生成。除需要掌握大模型 API 调用和提示工程等技能外，选题小组还需要（1）了解 libfuzzer 的基本原理，（2）掌握 libfuzzer 编译模糊驱动、生成模糊目标的方法，以及（3）学会合理运用 llvm-cov 等工具对模糊驱动/模糊测试的有效性进行评估。

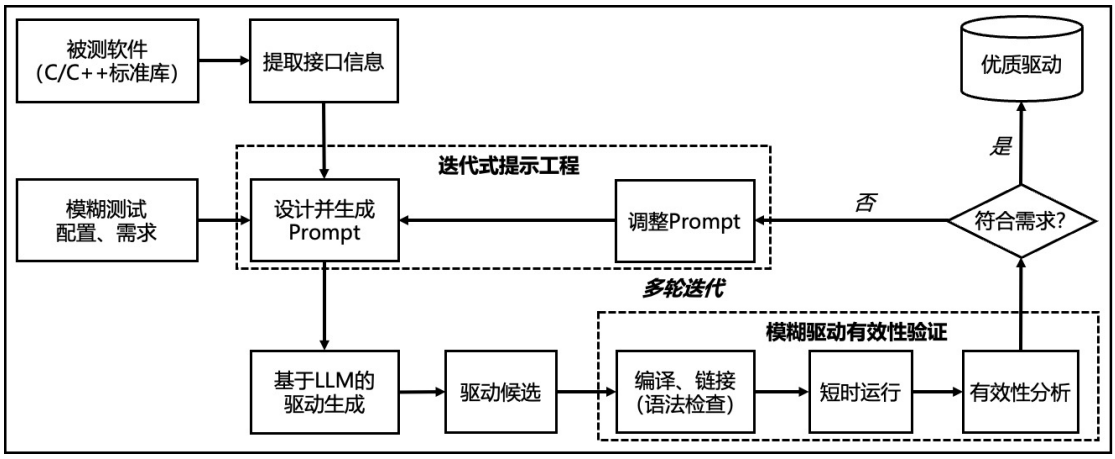

图 7 基于大模型的模糊驱动生成流程示意

3 验收内容

（1）源代码：业务代码和流程代码，如 Java、Python、Shell、R；生成的模糊驱动代码（各种 LLVMTestOneInput），以及编译这些模糊驱动所需的脚本。

（2）开发日志：Markdown 文件（devlog.md）。记录开发过程中的进度安排、任务分配、遇到的难题和解决的过程等。注意描述的逻辑性。

（3）项目文档：Markdown 文件（README.md）。介绍项目的设计方案（架构、流程、类层次设计、使用方法等）和使用方法，可参考：仓库 1、仓库 2。描述应清晰、详实，按照“总-分”结构介绍；模糊驱动编译指南。

（4）开箱即用的环境：使用 Docker，基础镜像为 ubuntu:22.04。提供可访问的 Dockerhub 链接，或是提供可本地一键构建的 Dockerfile。环境中应包含所有生成的模糊驱动和编译得到的模糊目标、每个模糊目标运行 24 小时之后产生的测试数据（参考 AFL++的输出）。选题小组提交的环境应该能够让绝大部分用户在配合 README.md 的情况下轻松使用工具和开展数据分析。

（5）评估结果：一组统计图表（pdf、png、csv 等），最少包括每个模糊驱动的覆盖率曲线图，以及模糊驱动的覆盖分布对比。

（6）演示视频：演示工具的使用方法，控制在 8 分钟以内。视频上传到 B 站，视频链接贴在 README.md 里。

联系方式

钱瑞祥，**qrx_at@163.com**

希望大家都可以养成先想后问的习惯，避免低级问题！

与此同时，学会使用大模型（一个永远不会骂你的好老师……）。