HGU LINC 3.0 Leaders in Industry-university Cooperation

2023년도 2학기 캡스톤디자인 결과 보고서

과제구분	☐ 일반 캡스톤디자인 ☐ 기업연계 캡스톤디자인 ☐ 기업연계 기반 학제간 융합 또는 글로벌 캡스톤디자인				
교과목명	캡스톤 디자인1				
과 제 명	자동테스팅 도구 SymCC의 성능과 사용성 개선				
팀 명	HK1				
지도교수	**소속학과**	전산전자공학부	**성 명**	홍신	
기업정보	**기업명**		**성 명**		
	연락처		**E-mail**		

	No.	성명	소속학과	학년	학번	연락처	E-mail
참여학생	**1 (팀장)**	한나린	전산전자공학부	3	22100768	010-4972-3119	lynnie21@handong.ac.kr
	2	하정원	전산전자공학부	3	21900780	010-5355-7081	jeongwon19@handong.ac.kr
	3	이진주	전산전자공학부	3	22100579	010-4351-9182	22100579@handong.ac.kr
	4	함상훈	전산전자공학부	3	22000796	010-4304-7266	22000796@handong.ac.kr

본인은 기업연계 캡스톤디자인 과제를 성실하게 지도하고
이에 따른 결과보고서를 제출합니다.

2023년 12월 6일

지도교수 : 홍 신 (인)

한동대학교 산학연협력 선도대학(LINC 3.0) 사업단장 귀하

설계 요약

문제정의	동적 심볼릭 실행을 통해 프로그램을 대상으로 테스트 입력을 자동으로 생성하는 자동 테스팅 도구인 SymCC의 탐색 휴리스틱과 사용자 인터페이스를 개선함으로써 실제 C/C++ 프로그램 테스팅에서 활용성을 개선하고자 한다.	
설계목표	1. 기존의 동적 심볼릭 테스팅 연구에서 제안된 다양한 탐색 휴리스틱을 SymCC 기반으로 구현하여 SymCC의 성능 향상 2. SymCC의 다양한 파라미터를 쉽게 튜닝할 수 있는 인터페이스 개선 3. SymCC 성능 문제가 발생하는 코드 영역을 비쥬얼하게 파악할 수 있는 인터페이스 개선 4. SymCC를 위한 테스트 드라이버를 쉽게 작성할 수 있는 인터페이스 개선	
해당 설계요소	기능 설계	SymCC 프레임워크 분석 및 인터페이스 개선
	개념 및 상세설계	C/C++ 소스코드를 입력으로 받으며, SymCC 과정을 거쳐 변환된 코드 및 보고서 또는 문서를 출력으로 제공한다.
	구현 및 제작	SymCC 프레임워크 분석을 위해 코드 Instrumentation을 분석하고, 현재 탐색 알고리즘의 성능을 평가하고, 새로운 탐색 알고리즘을 구현한다. 인터페이스 개선을 위해 SymCC 테스팅 과정 리포트 기능을 개발하고, 성능 문제 프로파일링 기능을 개발하며, 테스트 드라이버 작성 인터페이스를 개발한다.
	시험 및 평가	전체적인 사용성을 평가하기 위해 사용자가 쉽게 접근하고 기능을 이해하는지, 인터페이스가 직관적이고 사용하기 편한지, 프로파일링 기능이 유용한 정보를 제공하는지 등을 평가한다. 또한, 새로운 탐색 알고리즘이 기존보다 성능을 향상시켰는지, 테스트 드라이버 생성 기능이 효율적으로 작동하는지, 성능 문제를 식별하고 해결하는 데 도움이 되는지 등을 종합적으로 고려해 평가한다.
	기타	
제약조건	개발환경	SymCC, LLVM, Clang, Visual Studio Code
	운영환경	Ubuntu 18.04.6 LTS
	제작비용 및 기간	2023-2학기, 2024-1학기로 약 10개월의 기간
	사용성 및 심미성	사용자 친화적인 인터페이스 및 성능 측정이 가능한 기능 구현
	사회 및 윤리성	개인 정보 보호, 보안 취약점 탐지, 악의적인 사용 방지, 공정성과 균형, 저작권 및 지적 재산권
	기타	
예상 산출물	프로파일링 분석 보고서, 개선된 인터페이스	
요약문	본 프로젝트에서는 자동 테스팅 도구인 SymCC의 탐색 휴리스틱과 사용자 인터페이스를 개선함으로써 실제 C/C++ 프로그램 테스팅에서 활용성을 개선하고자 한다. 기존의 동적 심볼릭 테스팅 연구에서 제안된 다양한 탐색 휴리스틱을 SymCC 기반으로 구현해 성능을 향상하고, 다양한 파라미터를 쉽게 튜닝하고, 성능 문제가 발생하는 코드 영역을 비쥬얼하게 파악하고, 테스트 드라이버를 쉽게 작성할 수 있는 인터페이스로 사용성을 개선하는 것을 목표로 한다.	

Summary of Capstone Design Project Proposal

<table>
<tr><td>Problem Definition</td><td colspan="2">The goal is to enhance the practical utility of SymCC, an automated testing tool employing dynamic symbolic execution to generate test inputs for programs, by improving its exploration heuristics and user interface. This improvement aims to boost its effectiveness in testing real C/C++ programs.</td></tr>
<tr><td>Design Objectives</td><td colspan="2">1. Implement various exploration heuristics proposed in prior dynamic symbolic testing research into SymCC to enhance its performance.
2. Improve the interface to allow easy tuning of various parameters within SymCC.
3. Enhance the interface to visually identify areas in code where SymCC performance issues arise.
4. Improve the interface to facilitate easy creation of test drivers for SymCC.</td></tr>
<tr><td rowspan="5">Design Elements</td><td>Function Design</td><td>Analysis of the SymCC framework and interface enhancement</td></tr>
<tr><td>Conceptual Design</td><td>SymCC framework accepts C/C++ source code as input and provides transformed code and reports or documents as output after going through the SymCC process.</td></tr>
<tr><td>Implementation</td><td>To analyze the SymCC framework, we will review its code instrumentation methods, evaluate the current exploration algorithm's performance, and implement new algorithms as needed. For interface enhancement, we'll develop a testing process report feature within SymCC, enabling detailed insights into executed test cases and coverage. Additionally, we'll create a performance issue profiling functionality to identify bottlenecks and problematic areas. Lastly, we'll design an intuitive interface for easy test driver creation and management, streamlining the generation and maintenance of test cases for targeted programs. These efforts aim to enhance SymCC's functionality, usability, and effectiveness in dynamic symbolic testing scenarios.</td></tr>
<tr><td>Test and Experiment</td><td>To evaluate the overall usability, we assess whether users can easily access and comprehend the functionality, whether the interface is intuitive and user-friendly, and if the profiling feature provides valuable information. Additionally, we consider whether the new exploration algorithm enhances performance compared to the previous one, if the test driver generation feature operates efficiently, and if it aids in identifying and resolving performance issues. Comprehensive evaluation involves considering these factors collectively to assess the overall usability and make necessary improvements.</td></tr>
<tr><td>Etc</td><td></td></tr>
<tr><td rowspan="2">Constraints</td><td>Development Env.</td><td>SymCC, LLVM, Clang, Visual Studio Code</td></tr>
<tr><td>Operating Env.</td><td>Ubuntu 18.04.6 LTS</td></tr>
</table>

	Cost and Period	10 months (2023-Fall to 2024-Spring semesters)
	Usage and	Implementing a user-friendly interface and performance measurement functionality.
	Usability and Aesthetics	Privacy protection, security vulnerability detection, prevention of malicious use, fairness and balance, copyright and intellectual property rights.
	Etc	
예상 산출물	Profiling analysis report, improved interface.	
요약문	In this project, we aim to improve the usability of SymCC, an automated testing tool, for practical C/C++ program testing by enhancing its exploration heuristics and user interface. We will implement various exploration heuristics proposed in existing dynamic symbolic testing research to improve performance, facilitate easy tuning of parameters, visually identify performance-critical code areas, and enhance usability with an interface that allows easy creation of test drivers.	

결과 보고서

목차

1. 프로젝트 개요
   1.1. 문제 정의 (Problem Definition)
   1.2. 목적과 목표 (Goals and Objectives)
   1.3. 제약 조건 (Constraints)
   1.4. 최종(예상)산출물 (Expected Outcomes)
   1.5. 프로젝트 이해 관계자 (Key Project Stakeholders)
   1.6. Project Requirements

2. 프로젝트 배경 정보
   2.1. 기존의 유사 제품에 대한 조사
      2.1.1. KLEE(KLEE Symbolic Execution Engine)
         2.1.1.1. 주요 특징
         2.1.1.2. 장점
         2.1.1.3. 단점
      2.1.2. AFL(American Fuzzing Lop)
         2.1.2.1. 주요 특징
         2.1.2.2. 장점
         2.1.2.3. 단점
   2.2. 기존의 연관 논문 및 특허 조사
      2.2.1. "Symbolic execution with SymCC: Don't interpret, compile!" by Sebastian Poeplau and Aurelien Francillon, EURECOM
      2.2.2. "Learning to Explore Paths for Symbolic Execution" by Jingxuan He, Gishor Sivanrupan, Petar Tsankov, Martin Vechev
   2.3. 관련된 전공 지식 분야
      2.3.1. 정적 및 동적 분석 기술
      2.3.2. 컴파일러 및 LLVM 기술
      2.3.3. 보안 및 위약점 분석

3. 개념 디자인과 전략
   3.1. 필요 기능 (Functions)
      3.1.1. SymCC 테스팅 과정 리포트 기능
      3.1.2. SymCC 성능 문제 프로파일링 기능 개발
      3.1.3. 테스트 드라이버 작성 인터페이스 개발
   3.2. 구현 전략 (Methodology)
   3.3. 프로토타입 (Prototype)
      3.3.1. SymCC 테스팅 과정 리포트 기능 개념 디자인

3.3.2. SymCC 성능 문제 프로파일링 기능 개념 디자인
3.3.3. 테스트 드라이버 작성 인터페이스 개념 디자인
3.4. 초기 실험 (Pilot Test)

4. 프로젝트 추진
4.1. 수행한 작업
4.1.1. 배경지식 스터디
4.1.2. 개념 숙달을 위한 과제 수행
4.2. 추후 계획
4.2.1. 구현 및 제작
4.2.2. 시험 및 평가
4.3. 팀 협력 방안과 팀원 별 역할

5. 필요한 자원
5.1. 구현 및 실험에 예상되는 소요 장비 또는 부품 리스트
5.2. 프로젝트 수행에 필요한 예산 내역

6. 첨부
6.1. 인용 자료 및 참고 문헌
6.2. 실험 데이터, 수식 전개, 증명 등 세부 기술적인 사항들
6.3. 기타 첨부자료

1. 프로젝트 개요

1.1. 문제 정의 (Problem Definition)

SymCC는 동적 심볼릭 실행을 통해 프로그램을 대상으로 테스트 입력을 자동으로 생성하는 Whitebox Testing 도구로 C, C++, Rust, Object-C 등 다양한 시스템 프로그램에 적용이 가능한 오픈소스 테스팅 엔진이다. SMT 기반 분석을 통해 랜덤 테스팅으로 도달이 어려운 복잡한 실행경로에 대한 테스트 생성이 가능하며, 기존 동적 심볼릭 테스팅 도구가 최근 C 코드 지원이 어려운 것과 달리, LLVM 프레임워크 기반으로 다양한 C 계열 프로그램에 적용이 가능하다. 본 프로젝트에서는 자동 테스팅 도구인 SymCC의 탐색 휴리스틱과 사용자 인터페이스를 개선함으로써 실제 C/C++ 프로그램 테스팅에서 활용성을 개선하고자 한다.

1.2. 목적과 목표 (Goals and Objectives)

기존의 동적 심볼릭 테스팅 연구에서 제안된 다양한 탐색 휴리스틱을 SymCC 기반으로 구현하여 SymCC의 성능을 향상하고, SymCC의 다양한 파라미터를 쉽게 튜닝할 수 있는 인터페이스, SymCC 성능 문제가 발생하는 코드 영역을 비쥬얼하게 파악할 수 있는 인터페이스, SymCC를 위한 테스트 드라이버를 쉽게 작성할 수 있는 인터페이스 개선 등을 통해 사용성을 개선한다.

1.3. 제약 조건 (Constraints)

Concolic Testing 알고리즘의 명시적인 개선이 이루어져야 하며, SymCC 실행 과정을 프로파일링 및 모니터링하고 성능 bottleneck을 식별할 수 있는 새로운 UI를 제공해야 한다. 또한 테스팅의 결과는 사용자에게 직관적으로 보여져야 한다.

1.4. 최종(예상)산출물 (Expected Outcomes)

프로파일링 분석 보고서 및 사용자 친화적이고 직관적으로 개선된 인터페이스를 산출할 것으로 예상한다.

1.5. 프로젝트 이해관계자 (Key Project Stakeholders)

SymCC 프레임워크를 개발하고 유지/보수하는 개발자 및 소프트웨어 엔지니어들과 테스터 및 QA 엔지니어들, 실제 C/C++ 프로그램 개발자로 SymCC를 활용해 소스코드를 테스트하고자 하는 최종 사용자들이 본 프로젝트의 이해관계자들이다.

1.6. Project Requirements

SymCC 프레임워크 분석을 통해 기능, 구조, 동장 방식 등을 자세히 분석하고 강점과 약점을 파악해 개선할 수 있는 부분을 식별한다. 또한 현재 사용자 인터페이스의 기능 및 평가와 요구사항을 알아보고, 사용자 친화적이며 직관적이고 편리한 인터페이스를 설계한다.

2. 프로젝트 배경 정보

2.1. 기존의 유사 제품에 대한 조사

2.1.1. KLEE(KLEE Symbolic Execution Engine)

KLEE(KLEE Symbolic Execution Engine)는 LLVM 프레임 워크 위에서 동작하는 심볼릭 실행 도구로, C 및 C++ 프로그램에 대해 동적으로 분석한다. KLEE는 소스 코드 레벨에서 심볼릭 실행을 지원하며, 프로그램의 모든 가능한 실행 경로를 탐색하고 테스트 케이스를 생성하는데 사용된다.

2.1.1.1. 주요 특징

1) LLVM 기반

   KLEE는 LLVM 프레임워크를 기반으로 하며, 소스 코드를 LLVM 비트코드로 변환하여 분석한다. 이를 통해 다양한 플랫폼에서 사용할 수 있다.

2) 심볼릭 변수 및 심볼릭 메모리

   KLEE는 프로그램의 입력 변수 및 메모리를 심볼릭 변수 및 심볼릭 메모리로 다루어, 실행 경로를 추상화하고 분석한다.

3) 실행 경로 추상화

   KLEE는 각 분기 지점에서 발생하는 조건을 심볼릭으로 다루어 실행 경로를 추상화하고, 가능한 모든 경로를 탐색한다.

4) 테스트 케이스 자동 생성

   KLEE는 프로그램의 다양한 실행 경로를 탐색하면서, 테스트 케이스를 자동으로 생성한다. 이는 특히 소프트웨어 검증 및 테스트에 유용하다.

5) 제약 조건 해결

   KLEE는 실행 경로에서 발생하는 제약 조건을 해결하여, 입력 값을 찾아낸다.

2.1.1.2. 장점

1) LLVM 기반

   KLEE는 LLVM 프레임워크를 사용하여 다양한 플랫폼에서 동작 가능하며, 컴파일러 최적화 및 코드 분석이 효율적이다.

2) 실행 경로 추상화

   KLEE는 실행 경로를 정교하게 추상화하고 다양한 분기 조건을 탐색하는 데 강점이 있다.

3) 테스트 케이스 생성

   KLEE는 자동으로 테스트 케이스를 생성하여 코드를 효율적으로 테스트하는 데 도움이 된다.

2.1.1.3. 단점

1) 실행 속도

   KLEE는 대규모 프로젝트에서 느린 실행 속도를 보일 수 있다.

2) 경로 폭발 문제

KLEE는 다양한 실행 경로를 탐색하는 과정에서 많은 반복문과 분기로 나누어지면서 심볼릭 변수의 복잡성과 수 때문에 경로 폭발 문제가 발생할 수 있어 모든 경로를 완전히 탐색하는 것은 어려울 수 있다.

2.1.2. AFL(American Fuzzy Lop)

AFL(American Fuzzy Lop)은 Fuzz Testing을 기반으로 하는 자동화된 보안 테스트 도구로, 소프트웨어 프로그램에 있는 버그와 취약점을 찾기 위해 사용된다. SymCC와 다르게 심볼릭을 이용하는 것이 아닌 무작위 값을 생성해서 테스트한다.

2.1.2.1. 주요 특징

1) Fuzz Testing

   AFL은 입력값의 무작위 조합을 생성하여 프로그램을 테스트하는 퍼지 테스팅의 방식을 사용한다. 이를 통해 예상치 못한 입력 조합이나 실행 경로를 찾아내어 프로그램의 취약점을 발견한다.

2) 트리 기반 Fuzz Testing

   AFL은 트리 기반의 퍼지 테스팅을 수행하여 실행 경로를 추적하고, 이를 기반으로 더 효율적인 테스트 케이스를 생성한다. 이는 일반적인 무작위 Fuzzing보다 더 높은 코드 커버리지를 제공한다.

3) 자동으로 최적화된 테스트 케이스 생성

   AFL은 특정 입력값이 프로그램을 얼마나 "흥미롭게" 만드는지 측정하고, 이를 기반으로 최적화된 테스트 케이스를 자동으로 생성한다.

4) 멀티쓰레드 지원

   AFL은 여러 입력값을 병렬로 실행하면서 높은 성능을 제공한다.

5) 통계 및 리포트 기능

   AFL은 테스트 진행 상황에 대한 통계와 리포트를 제공하여 사용자가 진행 상황을 모니터링하고 분석할 수 있다.

2.1.2.2. 장점

1) 자동화된 테스트 케이스 생성

   자동으로 테스트 케이스를 생성하므로 사용자가 직접 입력값을 작성할 필요가 없다.

2) 좋은 코드 커버리지

   트리 기반의 Fuzz Testing 더 높은 코드 커버리지를 제공할 수 있다.

3) 오픈 소스

   오픈 소스로 제공되어 무료로 사용할 수 있다.

2.1.2.3. 단점

1) 모든 종류의 취약점을 찾지 못할 수 있음

   AFL이 특정 조건에서만 흥미로운 입력값을 생성하므로 모든 종류의 취약점을 찾지 못할 가능성이 크다.

2) 일부 프로그램에는 적용이 어려울 수 있음

일부 프로그램에서는 Fuzz Testing이 적용되기 어려울 수 있으며, 특히 복잡하거나 특수한 조건을 요구하는 경우에는 효과적이지 않을 수 있다.

2.2. 기존의 연관 논문 및 특허 조사

2.2.1. "Symbolic execution with SymCC: Don't interpret, compile!" by Sebastian Poeplau and Aurélien Francillon, EURECOM

SymCC는 심볼릭 실행의 속도 문제를 극복하기 위한 LLVM 기반 C 및 C++ 컴파일러로, 이진 코드에 콘콜릭 실행을 통합한다. SymCC는 KLEE와 비교해 최대 3배, 평균 12배 빠르며, 최근 성능 개선이 이뤄진 Qsym에 비해 최대 2배, 평균 10배 빠르다. 실제 프로젝트에서 적용 시 높은 코드 커버리지를 유지하며, OpenJPEG에서 두 취약점을 발견하여 해당 프로젝트에 보고되었다.

2.2.2. "Learning to Explore Paths for Symbolic Execution" by Jingxuan He, Gishor Sivanrupan, Petar Tsankov, Martin Vechev

이 연구는 심볼릭 실행(Symbolic Execution) 기술에서 발생하는 경로 폭발 문제를 해결하기 위한 새로운 전략인 "Learch"를 제안한다. Learch는 학습 기반의 전략으로, 주어진 실행 시간 동안 코드 커버리지를 최대화하는 것을 목표로 삼아 심볼릭 실행에서 유망한 상태를 선택한다.

2.3. 관련된 전공 지식 분야

2.3.1. 정적 및 동적 분석 기술

SymCC와 유사한 도구들은 프로그램을 정적 및 동적으로 분석하는 기술을 기반으로 한다. 프로그램의 실행 경로를 예측하고 버그를 찾는 능력이 필요하다.

2.3.2. 컴파일러 및 LLVM 기술

SymCC는 컴파일러 기술을 기반으로 하고 있으며, LLVM 프레임워크를 사용한다. 컴파일러 이론과 LLVM의 구조를 이해하는 것이 필요하다.

2.3.3. 보안 및 취약점 분석

SymCC와 유사한 도구들은 보안 취약점을 찾는 데 중점을 두고 있다. 보안 및 취약점 분석 기술에 대한 이해가 필수적이다.

3. 개념 디자인과 전략

3.1. 필요 기능 (Functions)

3.1.1. SymCC 테스팅 과정 리포트 기능

SymCC를 사용해 테스팅을 진행했을 때, 세부 과정을 알 수 없는 한계점이 존재한다. SymCC 테스팅 과정 리포트 기능을 개발함으로써 테스트의 결과만 고려하는 것이 아니라, 세부 과정도 고려할 수 있도록 환경을 마련한다.

1) 테스트 결과 추적 및 분석을 가능하게 한다. SymCC 테스팅을 통해 얻은 결과를 기록하고 추적할 수 있는 리포트 기능은 개발자들이 프로젝트의 현재 상태를 파악하는 데 도움을 준다. 이는 테스트 결과를 분석하여 버그나 오류를 식별하고 해결할 수 있다.
2) 두 번째는 보다 좋은 SW 품질 관리를 가능하게 한다. 테스팅 과정 리포트를 작성하면 소프트웨어의 품질을 관리할 수 있다. 리포트를 통해 어떤 테스트가 수행되었는지, 어떤 결과가 도출되었는지를 파악할 수 있다. 이를 통해 개발자들은 품질 관리를 위한 개선 작업을 수행할 수 있다.
3) 효율적인 협업을 가능하게 한다. 협업과 의사소통을 용이하게 하기 때문이다. 테스팅 과정 리포트는 개발자들 간의 협업과 의사소통을 원활하게 도와준다. 리포트를 통해 테스트 결과를 공유하고 논의할 수 있으며, 테스트에 대한 추가적인 조치나 수정이 필요한 경우에도 리포트를 참고하여 작업할 수 있다. 위의 이유들로 인해 소프트웨어 개발 과정에서 테스팅 과정 리포트 기능은 필수적이다. 이 기능을 통해 개발자들은 소프트웨어의 품질을 관리하고 문제를 해결할 수 있으며, 효율적인 협업과 의사소통을 이룰 수 있다.

3.1.2. SymCC 성능 문제 프로파일링 기능 개발

SymCC 성능 문제 프로파일링 기능은 소프트웨어의 성능 향상과 자원 관리에 중요한 역할을 한다. 이를 통해 개발자들은 성능 문제를 신속하게 파악하고 최적화할 수 있으며, 사용자에게 더 나은 성능과 경험을 제공할 수 있다.

1) 성능 최적화을 가능하게 한다. 소프트웨어의 성능 최적화는 사용자 경험과 시스템 효율성을 향상시키는 데 중요하다. SymCC 성능 문제 프로파일링 기능을 통해 프로그램 실행 시간, 메모리 사용량, 함수 호출 등과 같은 성능 관련 정보를 수집하고 분석할 수 있다. 이를 통해 어떤 부분에서 성능 문제가 발생하는지 파악하고, 해당 부분을 최적화하여 소프트웨어의 성능을 향상시킬 수 있다.
2) 자원 관리를 할 수 있다. SymCC 프로젝트에서는 자원 관리가 중요하다. 성능 문제 프로파일링 기능을 사용하여 메모리 누수, 자원 낭비, 비효율적인 알고리즘 등을 식별할 수 있다. 이를 통해 자원 사용을 최적화하고, 시스템의 안정성과 성능을 개선할 수 있다.
3) 성능을 튜닝할 수 있다. 성능 문제 프로파일링은 SymCC 의 성능 튜닝에 도움을 준다. 프로파일링 결과를 분석하여 성능 개선이 필요한 부분을 확인하고, 해당 부분을 수정하여 최적화할 수 있습니다. 이를 통해 SymCC 의 실행 시간을 단축하고, 사용자 경험을 개선할 수 있습니다.
4) 성능을 평가할 수 있다. 성능 문제 프로파일링은 다양한 구현 방식이나 알고리즘의 성능을 평가하는 데에도 사용될 수 있습니다. 프로파일링 결과를 분석하여 각 방식의 성능을 비교하고 평가할 수 있다. 이를 통해 개발자는 성능이 우수한 방식을 선택하고 적용할 수 있다.

3.1.3. 테스트 드라이버 작성 인터페이스 개발

테스트 드라이버 작성 인터페이스 개발은 소프트웨어 개발 과정에서 테스트의 효율성과 신뢰성을 높이는 데에 필요하다. 이를 통해 개발자는 품질 좋은 소프트웨어를 개발하고, 사용자에게 안정적이고 신뢰할 수 있는 제품을 제공할 수 있다.

1) 테스트 커버리지를 확보할 수 있다. 테스트 드라이버 작성 인터페이스를 개발하면 소프트웨어의 테스트 커버리지를 확보할 수 있다. 테스트 드라이버는 특정 모듈이나 컴포넌트를 테스트하기 위해 필요한 환경을 구성하고, 해당 모듈에 대한 테스트를 자동화하는 역할을 한다. 이를 통해 소프트웨어의 다양한 부분을 효과적으로 테스트하고, 테스트 커버리지를 높일 수 있다.
2) 자동화된 테스트 수행을 할 수 있다. 테스트 드라이버 작성 인터페이스를 개발하면 자동화된 테스트를 수행할 수 있다. 테스트 드라이버는 특정 모듈 또는 컴포넌트를 호출하고, 예상된 결과와 실제 결과를 비교하여 테스트를 수행한다. 이를 통해 반복적이고 복잡한 테스트 작업을 자동화하여 시간과 노력을 절약할 수 있다.
3) 테스트 환경을 제어할 수 있다. 테스트 드라이버 작성 인터페이스를 개발하면 테스트 환경을 제어할 수 있다. 필요한 데이터를 설정하고, 모의 객체를 생성하거나 외부 시스템과의 상호작용을 시뮬레이션할 수 있다. 이를 통해 테스트의 신뢰성을 높이고, 테스트 결과의 일관성을 유지할 수 있다.
4) 버그 식별 및 수정이 용이하다. 테스트 드라이버 작성 인터페이스를 개발하면 버그를 식별하고 수정하는 것에 도움이 된다. 테스트 드라이버를 통해 예상치 못한 동작이나 오류를 발견하고, 이를 개발팀에 보고하여 수정할 수 있다. 이를 통해 소프트웨어의 품질을 향상시킬 수 있다.

3.2. 구현 전략 (Methodology)

위 기능을 구현하기 위해서는 먼저 SymCC 프레임워크를 분석한다. 코드 Instrumentation 분석을 통해서 오픈소스 소프트웨어인 SymCC 를 분석한다. 그 후 현재 SymCC 가 사용하고 있는 탐색 알고리즘의 성능을 평가하고, 새로운 탐색 알고리즘을 구현한다. 이후 테스팅 과정 리포트 기능, 성능 문제 프로파일링 기능, 테스트 드라이버 작성 인터페이스를 개발한다.

3.3. 프로토타입 (Prototype)

3.3.1. SymCC 테스팅 과정 리포트 기능 개념 디자인

Figure 1 에 나와 있는 바와 같이 테스팅 결과에 대한 리포트 기능을 구현한다.

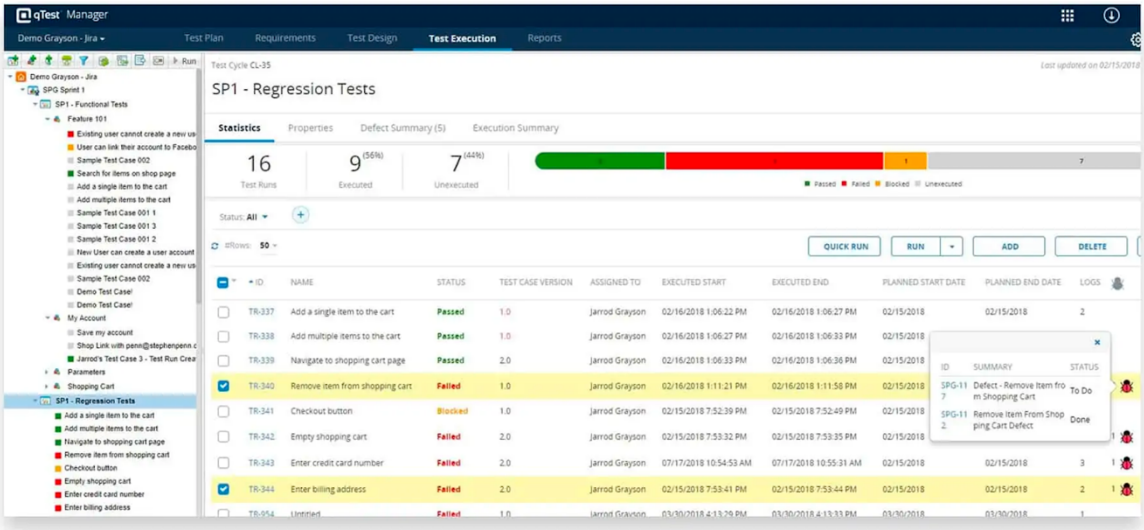

Figure 1. Interface of qTest

3.3.2. SymCC 성능 문제 프로파일링 개념 디자인

Figure 2 에 나와 있는 바와 같이, SymCC 의 각 기능 별 성능에 대한 프로파일링 제공을 구현한다.

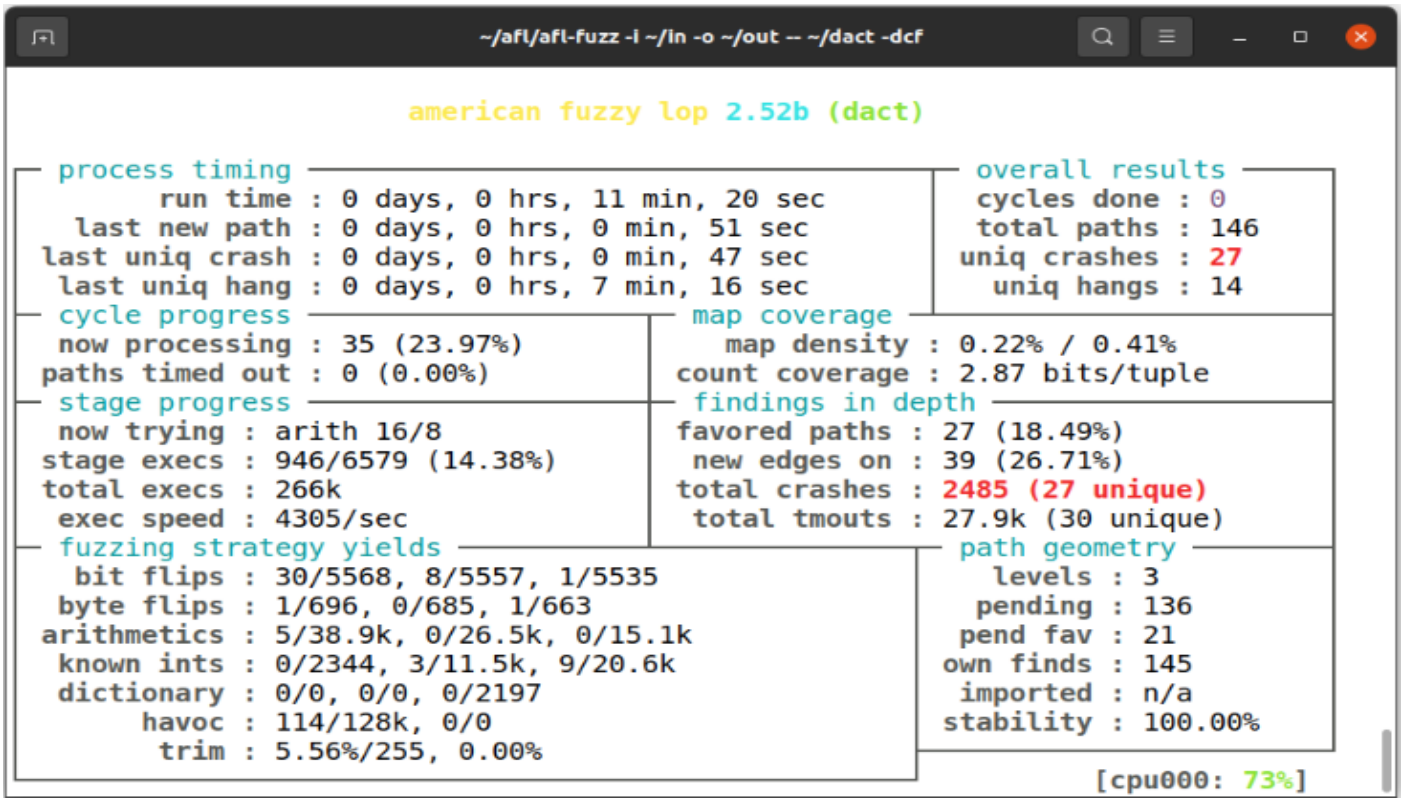

Figure 2. Result of Fuzzing using AFL

3.3.3. 테스트 드라이버 작성 인터페이스 개발 개념 디자인

Figure 3 에 나와 있는 바와 같이, 테스트 드라이버의 과정을 보다 편리하게 하기 위해 인터페이스를 개발해 사용하는 것을 구현한다.

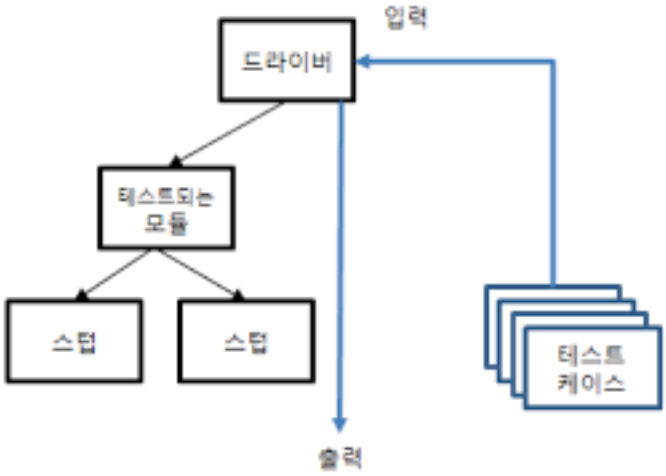

Figure 3. Flow Chart of Test Driver

3.4. 초기 실험 (Pilot Test)

본 프로젝트는 아직 스터디 단계로, 초기 실험 단계에 이르지 못했다.

4. 프로젝트 추진

4.1. 수행한 작업

4.1.1. 배경지식 스터디

1) Software Testing

Software Testing은 소프트웨어 개발 과정에서의 품질 관리 방법론으로, 생성된 프로그램 코드에 다양한 값을 입력하고 실제 작업을 수행하여 이를 관찰하여 원하는 품질을 달성하는 데 사용된다. 현재 IT 산업에서 경제 성장을 이끌고 있는 글로벌 산업 구조에서는 이것이 가장 중요한 생산 활동 중 하나로 간주될 수 있다.

2) SymCC

SymCC는 프로그램 소스 코드의 심볼릭 실행을 지원하는 컴파일러로서의 역할을 하는 open source Whitebox Testing 도구이다. '심볼릭 실행'은 프로그램을 실제로 실행하지 않고 코드의 모든 가능한 실행 경로를 분석하는 기술이다. SymCC의 경우, SMT(공리적 Satisfiability Modulo Theories) 기반 분석을 사용하여 무작위 테스트로 도달하기 어려운 복잡한 실행 경로에 대한 테스트 생성이 가능하다. 더불어 LLVM 프레임워크를 기반으로 하며, 프로그램 소스 코드의 컴파일 중에 심볼릭 실행을 위한 정보와 기능을 추가하여 더 빠르고 효율적인 심볼릭 실행 성능을 제공한다. 이는 다양한 C 계열 프로그램에 적용할 수 있다.

3) Coverage

Coverage는 테스트 경로를 검증하는 기준 중 하나이다. 테스트 경로란 노드들의 sequence이며 각 노드는 연관된 code entity의 테스트 케이스에 따른 실행에 의해 cover된다. 그래프 형식으로 나타난 테스트 경로를 CFG(Control Flow Graph)라고 하며, 이는 Graph coverage의 기본이 된다. Graph coverage의 몇 가지 측정 기준이 있다. Node

coverage, Edge coverage, Edge pair coverage 등이 있다. Node coverage의 경우 실행 플로우에 의해 커버된 노드의 비율을 나타낸다. Statement coverage가 그 예시이다. 만약 어떤 테스트 케이스의 실행이 모든 노드를 커버할 때 Test requirement가 만족되었다고 말할 수 있다. 이는 해당 케이스가 100% node coverage를 가진다는 것을 의미한다. Edge coverage는 실행 플로우에 의해 커버된 Edge의 비율을 나타낸다. Edge coverage의 개념은 Node coverage를 포함한다. 왜냐하면 "모든 테스트 세트가 100% Edge coverage를 달성한다"면 항상 100% node coverage를 달성할 것이기 때문이다.

4) Gcov

Gcov는 C 프로그램의 코드 coverage를 측정하는 데 사용되는 gcc의 확장 기능이다. -coverage 플래그를 사용한 명령어로 프로그램의 실행 흐름에 관한 정보를 포함하는 gcno 파일과 실행 파일을 생성할 수 있다. 이렇게 나온 실행 파일은 정상적으로 컴파일할 때의 실행 파일 크기와 비교하여 더 큰 용량을 갖는데, 위에서 언급한 추가적인 정보들을 포함하기 때문이다. 명령어를 사용해 컴파일한 실행 파일을 이용하여 test case를 실행하면 .gcda 파일이 생성되는데, 이 파일은 test case의 실행 중 어떤 라인들이 실행되었는지에 대한 정보를 저장한다. 또한 여러 번 실행하면 실행된 line들의 정보가 누적되게 된다. 그리고 gcov 명령어를 사용하여 gcda에 측정된 코드 coverage를 확인할 수 있으며, line coverage 뿐만 아니라 branch coverage와 function coverage도 확인할 수 있다.

5) LLVM

LLVM 은 LLVM 은 Low Level Virtual Machine 의 약자로, 프로그래밍 언어의 컴파일러 및 코드 최적화 도구를 개발하기 위한 오픈 소스 프로젝트이다. 프론트엔드, 백엔드 및 미들엔드로 구성되어 있다. 프론트엔드는 다양한 프로그래밍 언어로 작성된 소스 코드를 LLVM 의 중간 언어로 변환하는 역할을 한다. 백엔드는 LLVM 의 중간 언어를 목표로 하는 실제 코드로 변환하며, 이로써 여러 종류의 목표 코드를 생성할 수 있다. 미들엔드는 LLVM 의 중간 언어를 최적화하고 변환하는 역할을 담당하며, 이를 통해 코드의 효율성을 향상시킬 수 있다. LLVM 은 강력한 코드 최적화 기능을 제공하며, 중간 언어로서 LLVM IR 을 사용한다. LLVM IR 은 반 어셈블리 코드와 반 프로그래밍 언어로 구성되어 있어, 이를 통해 프로그래머는 고수준의 추상화와 함께 효과적인 코드 최적화를 수행할 수 있다. 또한 LLVM 은 산화제를 통해 실행 중에 발생하는 다양한 오류를 감지하고 보고하는 기능을 제공한다. 이는 메모리 오류, 데이터 경쟁, 초기화되지 않은 메모리 등을 검출하여 소프트웨어의 안정성을 향상시킨다. LLVM 은 컴파일러 기술의 발전과 코드 최적화에 기여하여 많은 프로그래밍 언어 및 프로젝트에서 널리 사용되고 있다. LLVM IR 을 사용하면 프로그래머는 각 모듈을 만들 수 있으며 LLVM 옵티마이저에서 많은 패스를 사용할 수 있다. LLVM sanitizer 는 메모리 오류, 데이터 경쟁, 초기화되지 않은 메모리 등 실행 오류를 감지하고 보고하는 역할을 한다. Sancov 는 LLVM sanitizer 를 기반으로 커버리지를 측정하는 도구이며, 커버리지 모드를 선택하고 자체 커버리지 도구를 사용할 수 있다.

4.1.2. 개념 숙달을 위한 과제 수행

학습한 개념들을 익히고 실습해보기 위해 비정기적인 과제를 수행하고, 수행 내용을 정기 미팅 시간에 발표하였다. 과제 수행을 위한 팀 결성은 파트너를 랜덤하게 교체하며 2인 1조로 이루어졌다.

1) Program failure 찾기

libxml2, libpng, curl 등 잘 알려진 오픈소스 프로젝트 또는 라이브러리의 이슈 트래커(Issue Tracker)에 올라온 다양한 이슈들을 확인 및 분석하고, 다양한 failure case 종류 중 최소 다섯 가지 이상의 샘플을 찾아 발표한다.

2) PIE(Execution, Infection, Propagation)

PIE 모델을 적용하여 분석 가능한 예제 코드와 test case들을 만들어보고 각 test case들에서 Execution, Infection, Propagation 이 나타나는 양상을 분석하고 발표한다.

3) CFG(Control Flow Graph)

오픈소스 프로젝트들 중 C++ 언어로 작성된 소스 코드로부터 이미지 형태의 CFG(Control Flow Graph)를 만들어주는 툴을 3가지 이상 찾고 또다른 오픈소스 소프트웨어에 대한 예제를 2가지 이상 만들어본 뒤, 각 툴들에 의해 만들어진 CFG를 비교 분석하여 발표한다.

4) Gcov

Gcov를 사용하여 replace 프로그램과 5000여개의 test case들 중 가장 많은 line coverage, branch coverage를 갖는 8개 case의 조합을 찾고 그 방법에 대해 발표한다.

5) Argparse 라이브러리

주어진 프로그램의 test case 쉘 파일을 하나씩 분리하고 Gcov를 사용하여 각각 test case에 대한 edge coverage 측정 후 메모리 주소 형식으로 나온 edge pair들을 실제 라인 넘버와 매칭하여 저장하고 방법론, 결과에 대하여 발표한다.

6) Libxml2 라이브러리

Configuration을 실행하고 생성된 실행 파일로 지정된 test case들을 실행시켜 보면서 제시된 질문들에 답을 찾고 그 과정에 대하여 발표한다.

7) Prime Path Coverage

오픈소스 프로젝트들 중 복잡한 loop 형태가 드러나는 프로젝트를 찾아 CFG를 만들고 Prime path를 찾는다. 임의의 direct graph가 입력되었을 때 Prime path를 찾는 알고리즘을 구성해 프로그램을 작성하고, 흥미로운 예제를 찾아 발표한다.

캡스톤디자인1 프로젝트 기간 동안, 위의 과제들을 차례로 수행하며 개념적 연구 및 학습에 적절한 실습을 병행함으로써 깊이를 더했고, 이후 연구를 위한 툴 사용의 숙련도를 높이는 과정을 밟았다. 또한 수행 내용과 과정에 대한 발표를 매 랩미팅마다 진행하며 설명을 위한 보다 깊은 이해를 위해 노력하였고 과제의 진행과 문제해결 흐름을 타인에게 설명하는 능력을 향상시키고, 프레젠테이션의 숙련도를 높일 수 있었다.

4.2. 추후 계획

4.2.1. 구현 및 제작

1) 코드 Instrumentation 분석

- 현재 SymCC 프레임워크에서 사용되고 있는 코드 Instrumentation 기술을 자세히 조사한다.
- 코드 Instrumentation이 어떻게 동작하고, 어떤 종류의 정보를 수집하는지를 파악한다.
- 현재 사용 중인 Instrumentation 기법의 장점 및 한계를 평가한다.

2) 탐색 알고리즘 성능 평가 및 새로운 알고리즘 구현

- SymCC 프레임워크의 현재 탐색 알고리즘의 성능을 측정하고 분석한다.
- 성능 평가 결과를 토대로 개선이 필요한 부분을 식별하고, 새로운 탐색 알고리즘을 설계하고 구현한다.
- 구현한 새로운 알고리즘의 성능을 기존 알고리즘과 비교하여 평가한다.

3) SymCC 테스팅 리포팅 기능 개발

- SymCC의 테스팅 프로세스를 자동화하여 리포트를 생성하는 기능을 개발한다.
- 리포트에는 각 테스트의 성공 여부, 수행된 동작, 오류 메시지 등을 자세히 기록하여 사용자에게 효과적인 피드백을 제공한다.
- 테스팅 결과를 분석하고 개선점을 도출할 수 있는 리포팅 기능을 구현한다.

4) 성능 문제 프로파일링 기능 개발

- SymCC 프레임워크에서 발생하는 성능 문제를 식별하고 프로파일링하는 기능을 개발한다.
- 성능 문제의 원인을 분석하고, 코드의 특정 부분이나 알고리즘이 어떻게 작동하는지 추적할 수 있는 기능을 구현한다.
- 프로파일링 결과를 시각적으로 제공하여 사용자가 성능 향상을 위한 조치를 취할 수 있도록 한다.

5) 테스트 드라이버 작성 인터페이스 개발

- SymCC 프레임워크를 사용한 테스트 케이스를 자동으로 실행하는 테스트 드라이버의 인터페이스를 개발한다.
- 테스트 드라이버는 SymCC를 적절히 호출하고 테스트 결과를 수집하여 사용자에게 제공한다.
- 사용자가 테스트를 효과적으로 수행하고 결과를 확인할 수 있도록 직관적이고 효율적인 인터페이스를 디자인한다.

4.2.2. 시험 및 평가

사용성 평가 기준은 다음과 같다.

1) 사용자가 쉽게 접근하고 기능을 이해할 수 있는가

2) 인터페이스가 직관적이고 사용하기 편리한가

3) 프로파일링 기능이 유용한 정보를 제공하는가

4) 개발한 탐색 알고리즘이 기존보다 성능을 향상시켰는가

5) 테스트 드라이버 생성 기능이 효율적으로 작동하는가

6) 성능 문제의 식별과 해결에 도움이 되는가

위 기준들의 만족 비율을 종합하여 SymCC의 활용성 향상 및 개선 정도를 확인할 수 있다.

4.3. 팀 협력 방안과 팀원 별 역할

배경지식 스터디와 관련 개념 및 내용의 학습 과정 전반은 모든 팀 구성원이 동일하게 진행하였다. 비정기적으로 나오는 과제 수행을 위해서는 매 과제마다 랜덤하게 파트너를 변경하여 2인 1조로 진행됐다.

5. 필요한 자원

5.1. 구현 및 실험에 예상되는 소요 장비 또는 부품 리스트

개인 PC, 서버, 개발 및 운영 환경에 필요한 툴 등

5.2. 프로젝트 수행에 필요한 예산 내역

따로 필요한 예산이 없음

6. 첨부

6.1. 인용 자료 및 참고 문헌

[1] Poeplau, S. (2020). *Symbolic execution with {SymCC}: Don't interpret, compile!* https://www.usenix.org/conference/usenixsecurity20/presentation/poeplau

[2] USENIX. (2020, September 14). *USENIX Security '20 - Symbolic execution with SymCC: Don't interpret, compile! Distinguished Paper Award Winner Presentation Video.* YouTube. https://www.youtube.com/watch?v=hOuY2tNz6Q8

[3] E. S. (n.d.). *GitHub - eurecom-s3/symcc: SymCC: efficient compiler-based symbolic execution*. GitHub. https://github.com/eurecom-s3/symcc

[4] Jingxuan He, Gishor Sivanrupan, Petar Tsankov, and Martin Vechev. 2021. Learning to Explore Paths for Symbolic Execution. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21). Association for Computing Machinery, New York, NY, USA, 2526–2540. https://doi.org/10.1145/3460120.3484813

6.2. 실험 데이터, 수식 전개, 증명 등 세부 기술적인 사항들

6.3. 기타 첨부자료