




# 연구논문/작품 제안서

2024 년도 제 2 학기

|            |   |
|------------|---|
| 논문/작품      | ○논문 ( 0 ) ○작품 ( ) ※ 해당란에 체크   |
| 제목         | Enhancing Symbolic Execution. FeatPARA: A Unified Approach for Path and State Optimization through Integrated Learning Mechanisms |
| GitHub URL | <a href="https://github.com/kimmail99/FeatPARA_2025_08.git">https://github.com/kimmail99/FeatPARA_2025_08.git</a>                 |
| 팀원명단       | 김문수  학번: 2020315614                            |

2024 년 10 월 11 일

지도교수 : 차 수 영 서명

## Abstract

심볼릭 실행(Symbolic Execution)은 소프트웨어 테스트에서 매우 강력한 도구로 사용되지만, 여전히 경로 폭발(Path Explosion)과 상태 폭발(State Explosion)문제로 인해 탐색 효율성에 한계가 있습니다. 이를 해결하기 위해 다양한 탐색 방법론이 제시되었습니다. PARADYSE는 경로 탐색 휴리스틱을 자동으로 생성하여 탐색 경로를 최적화하고, FeatMaker는 상태 특성(State Features)을 학습하여 상태의 탐색 우선순위를 설정함으로써 경로를 찾아가는 데 기여합니다. 두 도구 모두 경로와 상태 탐색을 최적화하지만, 각각 다른 방식으로 탐색을 수행합니다. 본 연구에서는 PARADYSE와 FeatMaker의 장점을 결합하여 경로와 상태를 더욱 효율적으로 탐색할 수 있는 새로운 도구를 제안합니다. 이 도구는 이중 학습 메커니즘을 통해 경로와 상태 탐색을 통합하여 성능을 극대화하며, 코드 커버리지와 버그 탐지 성능에서 더 나은 결과를 기대할 수 있습니다.

## 서론

### 소프트웨어 테스트의 중요성

현대 소프트웨어는 복잡성이 크게 증가함에 따라 그 규모와 기능이 매우 다양해지고 있습니다. 이로 인해 소프트웨어 테스트의 중요성 또한 비약적으로 커지고 있습니다. 소프트웨어 테스트는 소프트웨어의 신뢰성, 안정성, 성능을 보장하는 데 핵심적인 역할을 하며, 특히 대규모 시스템에서는 작은 결함이 치명적인 결과를 초래할 수 있습니다. 따라서 모든 가능한 실행 경로를 테스트하고, 잠재적인 결함을 사전에 발견하는 것이 매우 중요합니다.

이러한 맥락에서 심볼릭 실행(Symbolic Execution)은 소프트웨어 테스트에서 매우 강력한 방법론으로 자리잡았습니다. 심볼릭 실행은 프로그램 내의 모든 실행 경로를 탐색하고, 해당 경로에서 발생할 수 있는 다양한 입력 조건을 자동으로 생성하는 방식으로, 높은 코드 커버리지를 달성할 수 있는 중요한 도구입니다. 특히 심볼릭 실행은 프로그램 내에 존재하는 잠재적 버그를 효과적으로 찾아낼 수 있기 때문에, 많은 연구자와 개발자들이 선호하는 기술 중 하나입니다.

### 심볼릭 실행의 동작 원리

심볼릭 실행의 핵심은 프로그램 입력값을 심볼릭 변수로 치환하는 데 있습니다. 이를 통해 프로그램이 실행될 수 있는 모든 경로를 가능한 변수 조합에 따라 탐색하게 됩니다. 각 실행 경로는 특정한 조건에 따라 분기되며, 심볼릭 실행은 이러한 분기를 따라 여러 상태를 유지하면서 탐색을 계속합니다. 각 경로에 대해 SMT 솔버(Satisfiability Modulo Theories Solver)를 사용하여 경로 조건을 검증하고, 만족할 수 있는 조건을 찾아내면 해당 경로를 추적합니다. 이렇게 탐색된 경로를 바탕으로, 자동으로 테스트 케이스를 생성함으로써 코드 커버리지를 극대화할 수 있습니다.

심볼릭 실행의 장점 중 하나는, 수동으로 수행하기 어려운 대규모 테스트 케이스 생성을 자동화할 수 있다는 점입니다. 이를 통해 코드 커버리지를 높이고, 일반적인 테스트에서 놓칠 수 있는 복잡한 경로를 탐색함으로써 더 많은 결함을 발견할 수 있습니다. 그러나 이러한 강력한 도구임에도 불구하고, 심볼릭 실행은 여러 중요한 한계에 직면하고 있습니다. 그 중

에서도 경로 폭발(Path Explosion)과 상태 폭발(State Explosion)문제는 심볼릭 실행의 가장 큰 장애물로 꼽힙니다.

### 경로 폭발 문제

경로 폭발(Path Explosion)문제는 심볼릭 실행 도구가 직면하는 가장 큰 난제 중 하나입니다. 이는 프로그램 내에서 발생하는 분기(branching)때문에 발생하는 문제로, 프로그램이 실행될 수 있는 경로의 수가 기하급수적으로 증가한다는 데 그 본질이 있습니다. 예를 들어, 프로그램이 if-else조건문을 다수 포함할 경우, 각 조건문마다 새로운 분기가 발생하게 되며, 각 분기마다 탐색해야 할 경로의 수가 배로 늘어납니다. 이러한 분기가 프로그램 전반에 걸쳐 누적되면, 결국 탐색해야 할 경로의 수가 기하급수적으로 증가하게 됩니다. 이러한 경로 폭발 문제는 심볼릭 실행의 주요 병목 현상으로 작용하며, 심지어 소규모 프로그램에서도 모든 경로를 탐색하는 것이 현실적으로 불가능해질 수 있습니다.

예를 들어, 간단한 if-else조건문이 10개만 존재해도  $2^{10}(=1024)$ 개의 경로가 발생할 수 있으며, 이 경로들이 중첩되면 탐색해야 하는 경로의 수는 수천, 수만 개에 이를 수 있습니다. 이러한 상황에서 모든 경로를 탐색하는 것은 시간과 자원의 측면에서 매우 비효율적일 뿐만 아니라 사실상 불가능합니다. 따라서 심볼릭 실행에서 모든 경로를 탐색할 수 없을 때, 어떤 경로를 탐색할 것인지를 결정하는 것이 매우 중요합니다. 이는 심볼릭 실행의 성능을 좌우하는 중요한 요소로 작용합니다.

### 상태 폭발 문제

상태 폭발(State Explosion)문제는 경로 폭발 문제와 밀접한 관련이 있습니다. 경로 폭발 문제가 프로그램의 분기로 인해 경로의 수가 급격히 증가하는 문제라면, 상태 폭발 문제는 각 경로에서 발생하는 상태의 수가 급격히 증가하는 문제입니다. 프로그램이 실행되는 동안 각 상태는 프로그램의 특정 시점에서의 메모리 상태, 변수 값, 경로 조건 등을 포함하며, 이러한 상태가 계속해서 누적되면서 상태 폭발 문제가 발생하게 됩니다.

예를 들어, 프로그램의 분기가 많을수록 상태도 그만큼 복잡해지고, 여러 경로에서 발생하는 다양한 상태를 모두 탐색하는 것은 매우 어렵습니다. 상태 폭발 문제는 프로그램의 복잡성이 증가할수록 더 심각해지며, 모든 상태를 탐색하려면 매우 많은 시간과 자원이 필요합니다. 이러한 이유로, 심볼릭 실행에서는 어떤 상태가 더 탐색할 가치가 있는지를 결정하는 것이 매우 중요한 과제가 됩니다.

### 연구의 필요성

이러한 경로 폭발과 상태 폭발 문제는 심볼릭 실행의 성능에 큰 영향을 미치며, 이를 해결하지 않으면 심볼릭 실행 도구의 잠재력을 충분히 발휘할 수 없습니다. 경로와 상태를 모두 효과적으로 탐색하는 것은 매우 중요한 과제입니다. 기존의 심볼릭 실행 도구들은 주로 경로 탐색 또는 상태 탐색 중 하나에 중점을 두고 있으며, 이로 인해 탐색의 효율성이 제한됩니다.

PARADYSE와 FeatMaker는 각각 경로 탐색과 상태 탐색에 중점을 두고 있지만, 두 접근 방식을 결합함으로써 경로와 상태 탐색의 효율성을 동시에 극대화할 수 있는 가능성이 있습니다. 본 연구는 이 두 도구의 장점을 결합한 새로운 도구를 제안함으로써, 경로와 상태 탐색 모두에서 더 높은 성능을 달성하고, 기존 도구들이 직면한 한계를 극복하고자 합니다.

## 2. 선행연구 및 기술현황

### 2.1 선행연구

KLEE는 심볼릭 실행(Symbolic Execution) 분야에서 널리 사용되는 대표적인 도구로, 프로그램의 다양한 실행 경로를 탐색하고 테스트 케이스를 자동으로 생성하여 코드 커버리지를 극대화하는 데 기여해왔습니다. KLEE는 프로그램의 입력을 심볼릭 변수로 대체하고, 가능한 모든 실행 경로를 탐색하여 프로그램 내의 분기(branch)와 조건문을 자동으로 분석합니다. 이 과정에서 KLEE는 SMT 솔버(Satisfiability Modulo Theories Solver)를 사용하여 경로 조건을 검증하고, 만족할 수 있는 조건을 찾아내어 각 경로를 추적함으로써 높은 효율성을 보여줍니다.

그러나 KLEE는 경로 폭발(Path Explosion)상태 폭발(State Explosion)문제라는 두 가지 주요 한계에 직면하고 있습니다. 경로 폭발 문제는 프로그램 내의 분기로 인해 탐색해야 할 경로의 수가 기하급수적으로 증가하는 현상을 의미하며, 이는 심볼릭 실행의 주요 병목 현상으로 작용합니다. 이러한 문제를 해결하기 위해, 다양한 탐색 방법론이 연구되고 있습니다.

PARADYSE는 경로 탐색을 최적화하기 위해 설계된 도구로, 심볼릭 실행 도구에서 경로 탐색 휴리스틱을 자동으로 생성합니다. PARADYSE는 프로그램의 특성에 맞춘 파라미터화된 휴리스틱을 학습하여 중요성이 높은 경로를 우선 탐색함으로써 경로 폭발 문제를 완화하고, 더 높은 코드 커버리지와 효율적인 탐색을 달성할 수 있습니다. PARADYSE는 심볼릭 실행 도구인 KLEE와 통합되어 있으며, 다양한 실험 결과에서 기존의 탐색 방법들보다 더 높은 브랜치 커버리지와 버그 탐지 성능을 제공하는 것으로 확인되었습니다.

FeatMaker는 상태 폭발 문제를 해결하기 위한 도구로, 프로그램의 각 상태에 대한 특성을 학습하고, 이를 기반으로 상태 탐색의 우선순위를 설정하여 중요한 상태를 먼저 탐색합니다. FeatMaker는 상태 특성을 분석하고, 이를 바탕으로 상태 간의 차이를 구분하여 더 중요한 상태에 더 높은 우선순위를 부여합니다. 상태 특성은 프로그램의 경로 조건(path condition)과 상태 변화에 따라 동적으로 생성되며, 심볼릭 실행 도중 축적된 데이터를 활용하여 상태 탐색의 우선순위를 조정합니다. 이를 통해 상태 폭발 문제를 완화하고, 심볼릭 실행의 탐색 성능을 개선하는 데 기여합니다.

### 2.2 기술현황

현재 심볼릭 실행을 개선하기 위한 다양한 도구들이 개발되고 있으며, 그 중에서도 KLEE, PARADYSE, FeatMaker가 대표적입니다. 이 도구들은 모두 심볼릭 실행의 탐색 성능을 향상시키기 위해 설계되었으며, 각 도구는 고유의 강점을 가지고 있습니다.

KLEE는 대표적인 심볼릭 실행 도구로, 프로그램의 실행 경로를 자동으로 탐색하고 테스트 케이스를 생성하는 데 특화되어 있습니다. KLEE는 특히 다양한 분기와 조건문을 처리하는 데 있어 우수한 성능을 보이며, 소규모에서 중대형 프로그램에 이르기까지 폭넓게 사용됩니다.

PARADYSE는 경로 탐색을 최적화하기 위해 자동으로 경로 탐색 휴리스틱을 생성하는 도구로, 탐색 경로의 우선순위를 효율적으로 결정하여 경로 폭발 문제를 완화하려고 합니다. PARADYSE는 프로그램의 복잡한 경로에서 더 높은 효율성을 발휘하며, 심볼릭 실행에서 경로 폭발 문제를 해결하는 데 중요한 역할을 합니다.

FeatMaker는 프로그램의 상태 특성을 학습하여 각 상태에 대한 탐색 우선순위를 설정함으로써, 상태 폭발 문제를 해결합니다. FeatMaker는 상태 탐색의 효율성을 높이고, 대규모 프로그램에서의 탐색 성능을 향상시키는 데 기여합니다.

### 2.3 과제 제안목표와 방향

본 연구의 목표는 PARADYSE와 FeatMaker의 장점을 결합하여 경로 탐색과 상태 탐색을 동시에 최적화하는 새로운 도구를 설계하는 것입니다. 이 도구는 경로 탐색과 상태 탐색을 모두 통합적으로 고려하여, 두 가지 탐색 전략이 상호작용할 수 있도록 하는 방법론을 제시합니다. FeatMaker는 상태 특성을 학습하면서 경로를 탐색하는 방식으로 작동하므로, 두 도구의 탐색 전략을 결합하는 것이 효과적입니다.

## 3. 작품/논문 전체 진행계획 및 구성

이 연구의 전체 진행 계획은 새로운 도구의 설계, 구현, 평가를 포함한 여러 단계를 포함합니다. 각 단계는 다음과 같이 구성되며, 각 요소는 심볼릭 실행 도구의 탐색 성능을 향상시키기 위한 구체적인 목표와 방법론을 제시합니다.

### 3.1 이중 학습 전략

내용: 본 연구에서는 FeatMaker의 상태 특성 학습과 PARADYSE의 경로 휴리스틱 학습을 동시에 적용하여, 경로와 상태라는 두 가지 측면에서 탐색 전략을 최적화합니다. 이를 통해 두 학습 메커니즘이 보완적으로 작용하여 탐색 성능을 극대화하는 방법을 개발합니다.

구체적 목표:

프로그램이 여러 상태에 도달할 수 있는 경로를 탐색할 때, 경로의 중요도를 평가하고 그 경로 내에서 가장 중요한 상태를 찾습니다.

경로 탐색 중 상태 특성을 고려하여, 각 상태의 중요도를 재조정하고, 탐색 우선순위를 효율적으로 설정합니다.

### 3.2 가중치 기반 탐색 통합

내용: 새로운 도구에서는 FeatMaker에서 사용되는 상태 특성 가중치와 PARADYSE에서의 경로 탐색 휴리스틱을 결합하여 탐색 우선순위를 정합니다.

구체적 목표:

각 상태에 가중치를 부여하는 과정에서 경로의 중요성도 함께 반영하고, 이를 통해 탐색의 우선순위를 더욱 정밀하게 결정합니다.

상태와 경로 간의 상호작용을 통해 더 많은 코드 커버리지와 버그 탐지 성능을 개선하는 방법론을 적용합니다.

### 3.3 피드백 기반 탐색 최적화

내용: 새로운 도구는 피드백 루프를 설계하여 탐색 중 얻은 데이터를 기반으로 경로와 상태의 탐색 전략을 동시에 개선합니다.

구체적 목표:

탐색할 때마다 경로와 상태를 동적으로 평가하고, 이를 통해 점진적으로 성능을 향상시킵니다.

예를 들어, 특정 경로에서 더 많은 버그를 발견할 경우, 해당 경로와 연결된 상태의 중요도를 높이고, 중요하지 않은 상태는 탐색에서 제외하여 효율적인 탐색을 수행합니다.

### 3.4 실험 및 성능 평가

내용: 새로운 도구의 성능을 평가하기 위해 다양한 벤치마크 프로그램을 사용하여 실험을 수행합니다.

Benchmarks: (sqlite-3.33.0, gawk-5.1.0, gcal-4.1, find-4.7.0, grep-3.4, diff-3.7, du-8.32, make-4.3, patch-2.7.6, ptx-8.32, expr-8.32, csplit-8.32, ls-8.32, trueprint-5.4, combine-0.4.0)

구체적 목표:

기존의 PARADYSE, FeatMaker와 비교하여, 코드 커버리지와 버그 탐지 성능에서 개선된 성과를 측정합니다.

탐색 성능을 정량적으로 분석하고, 이중 학습 전략과 피드백 기반 최적화의 효과를 평가합니다.

### 3.5 결과 분석 및 논의

내용: 실험 결과를 분석하여 새로운 도구의 효과성을 평가하고, 기존 도구들과의 성능 차이를 논의합니다.

구체적 목표:

탐색 성능의 개선 사항을 정리하고, 연구의 기여도를 명확히 합니다.

향후 연구 방향에 대한 제안 및 기존 도구들의 한계를 극복하기 위한 방법론을 제시합니다.

이 연구는 심볼릭 실행 도구의 성능을 극대화하기 위한 구체적인 계획을 가지고 있으며, 각 단계는 연구 목표 달성을 위한 필수적인 요소로 구성됩니다. 이를 통해 코드 커버리지와 버그 탐지 성능의 향상을 기대할 수 있습니다.

## 4. 기대효과 및 개선방향

### 1. 기대효과

본 연구에서 제안하는 새로운 도구는 PARADYSE와 FeatMaker의 장점을 결합하여 심볼릭 실행의 탐색 성능을 극대화하는 것을 목표로 하고 있습니다. 이 도구의 기대효과는 다음과 같습니다:

#### (1) 효율적인 리소스 사용

경로와 상태를 동시에 최적화함으로써, 탐색 과정에서 발생하는 불필요한 연산을 줄일 수 있습니다. 이로 인해 CPU와 메모리 등의 리소스를 효율적으로 사용할 수 있으며, 심볼릭 실행의 실행 속도를 개선할 수 있습니다.

#### (2) 높은 코드 커버리지

새로운 도구는 기존의 심볼릭 실행 도구들보다 더 높은 코드 커버리지를 달성할 것으로 기

대됩니다. 특히, 중요한 상태와 경로를 우선적으로 탐색함으로써, 잠재적인 결함을 사전에 발견할 가능성을 높입니다.

### (3)적시 피드백 제공

실시간 피드백 루프를 도입하여 탐색 중 얻은 데이터를 바탕으로 경로와 상태의 중요도를 동적으로 조정할 수 있습니다. 이를 통해 각 테스트 실행에서 얻은 정보를 즉시 반영하여 더 효과적인 탐색 전략을 수립할 수 있습니다.

## 2. 개선방향

새로운 도구가 개발된 후에는 다음과 같은 개선 방향이 필요합니다:

### (1)확장성과 유연성 확보

다양한 소프트웨어 환경에서 사용할 수 있도록 도구의 확장성과 유연성을 높이는 작업이 필요합니다. 이는 새로운 기능이나 알고리즘을 추가하는 데 용이하도록 설계되어야 합니다.

### (2)성능 비교 및 검증

기존의 심볼릭 실행 도구와의 성능 비교 실험을 통해, 새로운 도구의 효과성을 정량적으로 평가하고 검증해야 합니다. 다양한 벤치마크 프로그램을 통해 결과를 분석하고, 필요한 경우 알고리즘을 수정하는 피드백 과정을 구축해야 합니다.

### (3)사용자 친화적인 인터페이스 개발

최종 사용자가 도구를 쉽게 사용할 수 있도록 사용자 인터페이스(UI)를 개선하고, 문서화 및 교육 자료를 제공하는 방향으로 나아가야 합니다. 이는 연구 결과를 실용화하는 데 기여할 것입니다.

## 5. 기타

### 5.1 팀원간의 역할분담

본 연구는 혼자 진행될 예정이며, 모든 작업을 직접 수행할 계획입니다. 아래는 개인의 역할과 주요 실행 계획입니다.

<표1>

|     |   |
|-----|---|
| 이름: | 김문수   |
| 학번: | 2020315614  |
| 역할: | 연구 및 문헌 조사, 기술적 설계 및 도구 개발, 실험 및 성능 평가, 결과 분석 및 문서화 |

실행계획:

1단계: 연구 및 문헌 조사

관련된 심볼릭 실행 도구들(PARADYSE, FeatMaker 등)에 대한 문헌 조사를 진행하여 현재 기술 현황을 파악합니다.

2단계: 기술적 설계 및 도구 개발

새로운 도구의 구조를 설계하고, 이중 학습 메커니즘 및 피드백 루프를 구현합니다.

3단계: 실험 및 성능 평가

개발된 도구를 사용하여 다양한 벤치마크 프로그램에 대해 성능 실험을 진행합니다.

4단계: 결과 분석 및 문서화

실험 결과를 분석하고, 연구 결과를 정리하여 논문 형식으로 작성합니다.

## 5.2 비용 분석

본 연구는 혼자 진행되므로, 인건비는 제외하며, 필요한 비용을 아래와 같이 분석합니다.

<표2>

|           |              |
|-----------|--------------|
| 소프트웨어 비용: | 0(오픈소스 사용)   |
| 하드웨어 비용 : | 0(개인보유 장비사용) |

## 6. 참고문헌

[1] Jaehan Yoon and Sooyoung Cha. 2024. FeatMaker: Automated Feature Engineering for Search Strategy of Symbolic Execution. Proc. ACM Softw. Eng., Vol. 1, No. FSE, Article 108 (July 2024), 22 pages. <https://doi.org/10.1145/3660815>.

[2] Sooyoung Cha, Seongjoon Hong, Jiseong Bak, Jingyoung Kim, Junhee Lee, and Hakjoo Oh. 2022. Enhancing Dynamic Symbolic Execution by Automatically Learning Search Heuristics. IEEE Transactions on Software Engineering, 3640–3663 .

[3] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI '08), 209–224 .

[3] Koushik Sen, Darko Marinov, and Gul Agha. 2005. CUTE: A Concolic Unit Testing Engine for C. In Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE '05), 263–272 .

[4] Mark Harman, Yue Jia, and Yuanyuan Zhang. 2015. Achievements, Open Problems and Challenges for Search Based Software Testing. In 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST).