# Concolic Testing 기반 프로그램 상호작용 슬라이싱 기법 설계

서강복○, 김덕엽, 이우진
경북대학교 컴퓨터학부, 소프트웨어기술연구소
○e-mail:seokang13@naver.com, ejrduq77@naver.com, woojin@knu.ac.kr

# Design of interactive slicing method based on Concolic Testing

Kang Bok Seo○, Deok-Yeop Kim, Woo-Jin Lee
School of CSE & SWRC, Kyungpook National University

요 약

프로그램 슬라이싱은 처음 발표된 이후로 지금까지 다양하게 연구되어 테스팅, 디버깅, 개발, 유지 보수 등 다양한 분야에서 사용되어 왔다. 프로그램 슬라이싱을 수행하기 위해서는 기준이 되는 변수에 대해 소스코드 내에 존재하는 모든 라인들에 대해 의존성을 계산하여 프로그램 슬라이싱을 수행하게 된다. 하지만 지정된 변수에 대해서만 의존성 계산을 수행하기 때문에 프로그램을 실행 가능하게 하는 정보들에 대해서는 누락될 수 있고 이 경우의 프로그램 실행은 보장되지 않을 수 있다. 이러한 문제를 개선하기 위해 본 논문에서는 concolic testing 기법을 이용하여 기준이 되는 변수에 대한 의존성 있는 구문들을 추출하고 프로그램 실행에 요구되는 구문들을 같이 추출해낼 수 있는 프로그램 상호작용 슬라이싱 기법을 제안한다.

## 1. 서론

프로그램 슬라이싱이 처음 발표[1]되고 난 뒤 다양한 기법들이 연구되고 있다[2-4]. 프로그램 슬라이싱은 테스팅, 디버깅, 개발, 유지 보수 등 다양한 분야에서 사용되어 왔다. 프로그램 슬라이싱을 수행하기 위해서는 동적이나 정적으로 특정 변수에 대한 의존성을 계산한 뒤 소스코드의 각 부분을 분리해내야 한다. 하지만 지정된 변수에 대해서만 의존성 계산을 수행하기 때문에 프로그램을 실행 가능하게 하는 정보들에 대해서는 누락될 수 있다. 특히 지정된 변수와는 관련이 없지만 프로그램이 수행되기 위해 필요한 데이터나 변수들에 대해서는 의존성 계산을 통해서는 분별해내기 어렵다.

이러한 문제점은 화이트 박스 테스트 기법 중 하나인 concolic testing[5] 기법을 활용하여 개선될 수 있다. concolic testing 기법은 소스코드에 존재하는 모든 분기를 방문하는 것을 목표로 수행되는 테스트 기법으로 소스코드에 존재하는 분기마다 요구되는 입력을 자동으로 생성하며 테스트를 수행한다. concolic testing 기법의 수행 방법은 가상머신을 이용한 방법과 프로브 코드를 이용하는 방법 2가지가 존재하는데 본 논문에서는 프로브 코드 기반의 concolic testing 기법을 활용한 슬라이싱 기법을 제안한다. concolic testing을 수행하기 위해 각 라인별로 프로브 코드가 삽입되는데 이 프로브 코드에 실행된 라인들을 표시하는 데이터를 추가하여 테스팅이 진행되는 동안 실행된 라인들을 표시할 수 있다. concolic testing이 정상적으로 수행될 경우 해당 프로그램의 실행이 보장되고 실행된 구문들을 추출하면 지정된 변수에 의존성을 가지는 구문들과 프로그램 실행을 위해 요구되는 구문들 모두를 추출해낼 수 있다. 본 논문에서는 concolic testing 기법을 이용한 프로그램 상호작용 슬라이싱 기법을 소개한다.

## 2. 관련 연구

### 2.1 프로그램 슬라이싱

프로그램 슬라이싱이 발표된 이후로 지금까지 다양하게 연구되는 기법이다. 프로그램 슬라이싱은 크게 정적 슬라이싱[6]과 동적 슬라이싱[6]으로 구분되는데 정적 슬라이싱은 프로그램에 존재하는 구문들에 대해 특정 변수에 영향을 미치는 구문들을 프로그램을 실행하지 않고 추출해내는 기법이다. 동적 슬라이싱은 정적 슬라이싱과 반대로 프로그램을 실행하면서 특정 변수에 영향을 미치는 구문들을 추출하는 기법이다. 하지만 기존의 슬라이싱은 지정된 변수와는 관련이 없지만 프로그램이 수행되기 위해 필요한 데이터나 변수들에 대해서는 의존성 계산을 통해서는 분별해내기 어려워 수행 가능한 프로그램으로 추출하기 어렵다.

### 2.2 Concolic Testing

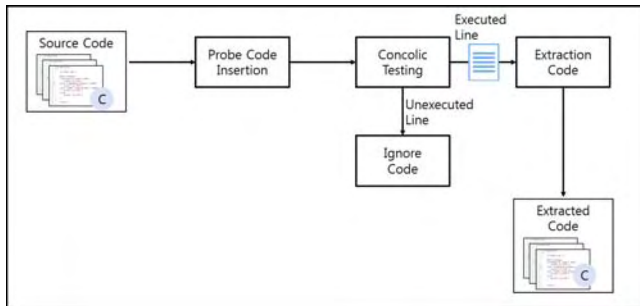
concolic testing 기법은 소프트웨어에 존재하는 모든 분기를 방문하기 위해 테스트 입력을 생성하여 테스트를

진행하는 기법으로 현재 다양한 분야의 연구에 적용되고 있다[7]. 본 논문에서 사용된 CREST[8] 도구는 concolic testing 기법의 대표적인 도구로서 다양한 연구에 사용되고 있다. concolic testing 기법은 소스코드의 양이 많아질수록 테스트에 요구되는 자원이 늘어나 규모가 큰 소프트웨어에는 적용하기가 쉽지 않다는 제한점이 존재한다.

## 3. 본론

### 3.1 Concolic Testing 기반 프로그램 상호작용 슬라이싱

본 장에서는 concolic testing을 이용하여 동적 프로그램 슬라이싱 기법의 구조를 그림 1과 같이 제안한다. 먼저 슬라이싱을 위한 소스코드에 대해 concolic testing을 위한 전처리를 진행하여 소스코드를 재생성 한다. 이 과정에서 슬라이싱의 기준이 되는 변수를 지정하고 소스코드의 각 라인별로 프로브 코드가 삽입된다. 이렇게 재생성된 소스코드를 concolic testing 도구로 테스트를 진행하게 되면 지정된 변수의 값이 변경되어 입력되면서 테스트가 진행되게 된다. 이 과정에서 각각의 프로브 코드에 실행된 라인을 로그로 남겨 실행된 라인들을 구분할 수 있다. 이 로그 파일을 이용하여 각각의 소스코드 파일에서 실행된 라인들을 추출하여 소스코드를 재생성하면 기준 변수에 대한 의존성 있는 구문들과 프로그램 실행에 요구되는 구문들도 같이 추출할 수 있다.




(그림 1) concolic testing 기반 프로그램 상호작용 슬라이싱 구성도

본 논문에서 제안한 concolic testing 기반의 동적 프로그램 슬라이싱을 수행하는 경우 지정된 변수의 의존성에 따른 구문과 프로그램 실행에 요구되는 각 구문들을 구분할 수 있는데 그림 2는 기존의 프로그램 슬라이싱의 예시이다. 왼쪽의 원본 소스코드에서 write(sum)구문을 기분으로 슬라이싱한 결과를 오른쪽에 보였다. 이 예시에서는 특정 구문을 기준으로 슬라이싱을 수행하였는데 프로그램의 로직 부분만을 표시하고 이외에 실제 프로그램을 실행하기 위한 요소들이 누락되어 있다. 기본적인 문법과 관련된 사항이나 괄호, 다른 코드나 모듈과의 상호작용, 환경 등 프로그램 로직에서는 드러나지 않고 실제 개발 환경에서 프로그램 실행 시 문제를 발생시킬 수 있는 요소들이 존재한다. 이러한 요소들이 누락될 경우 프로그램이 실행되지 않거나 프로그램이 정상적으로 동작하지 않아 슬라이실된 프로그램에 영향을 미칠 수 있다.

```
int i;
int sum = 0;
int product = 1;
int w = 7;

for(I = 1; i<N; ++i){
   sum = sum + i + w;
   product = product * i;
}
write(sum);
write(product);
```

```
int i;
int sum = 0;
int product = 1;
int w = 7;

for(I = 1; i<N; ++i){
   sum = sum + i + w;

}
write(sum);
```

(그림 2) 기존의 프로그램 슬라이싱

하지만 본 논문에서 제시하는 기법은 의존성 계산이 아닌 테스팅을 통해 실제 수행되는 구문만을 추출하여 슬라이싱을 진행하여 프로그램의 실행이 보장되는 슬라이싱을 진행할 수 있다. 표 1은 수도 코드(pseudo code)나 알고리즘 표현 시엔 드러나지 않고 실제 소스코드에서 나타나는 부분이지만 누락될 경우 프로그램 실행에 문제가 될 수 있는 요소들을 나타낸다. 표 1에 존재하는 내용 이외에도 누락 시 프로그램 오류를 발생시키는 요소들이 많이 존재하나 자주 고려되는 요소들을 나타내었다.

<표 1>. 누락 시 프로그램 실행에 영향을 주는 요소

| 구분 | 누락 시 발생하는 문제 |
|---|---|
| 헤더파일 | 변수, 함수 등의 호출 시 오류 발생 |
| {} | 문법 오류로 인한 오류 발생 |
| 외부 모듈 | 외부 모듈과의 상호작용 시 오류 발생 |

## 4. 결론

프로그램 슬라이싱은 처음 발표된 이후로 지금까지 다양하게 연구되어 테스팅, 디버깅, 개발, 유지 보수 등 다양한 분야에서 사용되어 왔다. 프로그램 슬라이싱을 수행하기 위해서는 지정된 변수에 대해 소스코드 내에 존재하는 모든 라인들에 대해 의존성을 계산해서 프로그램을 슬라이싱을 수행한다. 하지만 지정된 변수에 대한 의존성만을 계산하다 보니 프로그램 실행에 요구되는 구문들이 누락되어 프로그램의 실행이 보장되지 않을 수 있다.

이러한 문제점을 개선하기 위해 본 논문에서는 concolic testing 기반의 프로그램 상호작용 슬라이싱 기법의 구조를 제안하였다. 이 기법은 기존의 동적 프로그램 슬라이싱과 다르게 기준 변수의 의존성 있는 구문들과 더불어 프로그램 실행에 요구되는 구문들을 같이 추출하여 실행이 가능한 프로그램으로 슬라이싱해낼 수 있다. 또한 concolic testing 도구가 개발된 언어에 대해 추가적인 개발 없이 수행할 수 있다는 장점이 있다. 향후 연구로는 해

당 기법을 구현하여 기존의 프로그램 슬라이싱과 제안된 기법의 소요 자원을 비교하여 제안된 기법의 효율성을 시험하고자 한다.

* 본 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.NRF-2017R1D1A3B04035880)

### 참고문헌

[1] M. Weiser "Program slicing", ICSE '81 Proceedings of the 5th international conference on Software engineering, pp. 439-449, 1981.
[2] Baowen Xu, Ju Qian, Xiaofang Zhang, Zhongqiang Wu, Lin Chen, "A brief survey of program slicing", ACM SIGSOFT Software Engineering, Vol. 30, Issue 2, pp. 1-36, 2005.
[3] Hung Viet Nguyen, Christian Kästner, Tien N. Nguyen, "Cross-language program slicing for dynamic web applications", Proc. of the 10th Joint Meeting on Foundations of Software Engineering, pp. 369-380, 2015.
[4] Sandrine Blazy, Andre Maroneze, David Pichardie, "Verified Validation of Program Slicing", Proc. of the Conference on Certified Programs and Proofs, pp. 109-117, 2015.
[5] K. Sen, D. Marinov, G. Agha, "CUTE: a concolic unit testing engine for C," Proc. of European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, pp. 263-272, 2005.
[6] K. Galagher, D. Binkley, "Program Slicing", Frontiers of Software Maintenance
[7] R. Kannavara, C. Havlicek, B. Chen, M. Tuttle, K. Cong, S. Ray, and F. Xie, "Challenges and Opportunities with concolic testing", in IEEE National Aerospace Conference / Ohio Innovation Summit & IEEE Symposium on Monitoring & Surveillance Research (NAECON-OIS), pp. 374-378, 2015.
[8] J. Burnim and K. Sen, "Heuristics for scalable dynamic test generation," in Technical Report UCB/EECS-2008-123, 2008.