Vol.9, No.8 pp.229~234 pISSN: 2287-5905 https://doi.org/10.3745/KTSDE.2020.9.8.229

CRESTIVE-DX: Design and Implementation of Distrusted Concolic Testing Tool for Embedded Software

Hyerin Leem† · Hansol Choe† · Hyorim Kim†† · Shin Hong†††

ABSTRACT

This paper presents the design and the implementation of CRESTIVE-DX, a concolic testing tool that distribute the concolic testing process over the embedded target system and the host system for efficient test generation of a target embedded program. CRESTIVE-DX conducts the execution of a target program on the target embedded system to consider possible machine-dependent behaviors of a target program execution, and conducts machine-independent parts, such as search-strategy heuristics, constraint solving, on host systems with high-speed computation unit, and coordinates their concurrent executions. CRESTIVE-DX is implemented by extending an existing concolic testing tool for C programs CREST. We conducted experiments with a test bed that consists of an embedded target system in the Arm Cortex A54 architecture and host systems in the x86-64 architecture. The results of experiments with Unix utility programs Grep, Busybox Awk, and Busybox Ed show that test input generation of CRESTIVE-DX is 1.59 to 2.64 times faster than that of CREST.

Keywords : Concolic Testing, Test Generation, Embedded Software Testing, Symbolic Execution, Automated Testing

CRESTIVE-DX: 임베디드 소프트웨어에 대해 테스트케이스 생성을 지원하는 분산 Concolic 테스팅 도구

임 혜 린† · 최 한 솔† · 김 효 림†† · 홍 신†††

요 약

본 논문은 임베디드 소프트웨어에 대한 Concolic 테스팅을 효과적이고 효율적으로 지원하기 위해 임베디드 타겟(target) 시스템과 호스트(host) 시스템의 분산적이고 동시적으로 테스트 생성을 위한 작업을 수행하는 Concolic 테스팅 도구의 설계와 구현 사례를 소개한다. 소개하는 테스트 케이스 생성 도구는 Concolic 테스팅 과정 중 (1) 임베디드에 종속적인 특성을 갖는 테스트검증 대상 프로그램의 실행 부분은 임베디드 타겟 시스템에서 수행하고, (2) 시스템에 비종속적인 실행 부분인 탐색 전략, 제약식 해법기 실행 과정은 계산성능이 좋은 호스트 시스템에 분산하고, 독립적인 단계를 동시적으로 실행하도록 기존 Concolic 도구를 개선하였다. Arm Cortex A54 아키텍쳐의 임베디드 타겟 시스템과 x86-64 아키텍쳐의 호스트 시스템을 대상으로 본 기법을 구현하여 오픈소스 C 프로그램의 Grep, Busybox Awk, Busybox Ed를 대상으로 실험한 결과, 기존 도구 보다 1.59~2.64배 테스트케이스 생성속도가 향상됨을 확인할 수 있었다.

키워드 : Concolic Testing, 테스트 생성, 임베디드 소프트웨어 테스팅, 심볼릭 실행, 자동 테스팅

1. 서 론

임베디드 시스템이 사회 전반의 다양한 영역에 이용됨에 따라 임베디드 시스템을 구동하는 임베디드 소프트웨어의 오류가 사회의 안전을 위협하는 새로운 요인으로 대두되고 있으며[1], 이를 제거하기 위한 임베디드 소프트웨어 검증 기술의 수요가 계속적으로 증가하고 있다. 일반적인 소프트웨어 검증 상황과 달리, 임베디드 소프트웨어에 대한 검증은 특정 하드웨어 장치에 종속적인(비표준적) 기능과, 제약적인 하드웨어 자원 등을 필수적으로 반영하여 수행돼야 한다. 이러한 이유로, 표준적 프로그램 의미론에 기반한 기법이나 높은 계산 비용이 소요되는 기존의 소프트웨어 검증 기법을 임베디드 시스템에 직접적으로 적용하기 어려우며, 이로 인해 실제 임베디드 소프트웨어 검증 과정은 비체계적 관행에 그치는 경우가 많다.

본 논문은 임베디드 소프트웨어에 대한 Concolic 테스팅

※ 본 논문은 2020 한국소프트웨어공학 학술대회(KCSE 2020)에서 "임베디드 소프트웨어에 대한 테스트케이스 생성을 지원하는 분산 Concolic 테스팅 도구의 설계와 구현"을 제목으로 발표된 논문을 확장한 것임.
※ 본 연구는 정부 재원으로 소프트웨어중심대학 지원사업(2017-0-00130)과 한국연구재단의 지원(NRF-2017R1C1B1008159, NRF-2017M3C4A7068179, NRF-2020R1C1C1013512)을 받아 수행됨.
† 비 회 원 : 한동대학교 정보통신공학과 석사과정
†† 비 회 원 : 한동대학교 전산전자공학부 학사과정
††† 정 회 원 : 한동대학교 전산전자공학부 조교수
Manuscript Received : April 20, 2020
Accepted : June 6, 2020
* Corresponding Author : Shin Hong(hongshin@handong.edu)

을 효과적이고 효율적으로 지원하기 위해, 임베디드 타겟(target) 시스템과 호스트(host) 시스템을 분산하여 동시적으로 테스트 생성 작업을 수행하는 Concolic 테스팅 도구의 설계와 구현 사례를 소개한다.

Concolic 테스팅[2, 3]은 테스트 대상 프로그램으로부터 추출한 동적 실행 경로식을 SMT 해법기로 계산하여 새로운 테스트 케이스를 생성하는 기법이다. Concolic 테스팅 기법은, Z3[4] 등 고도로 발달된 SMT 해법기를 실행 경로식 계산에 효율적으로 활용함으로써, 대규모 코드로 이루어진 소프트웨어의 테스팅 자동화에도 높은 확정성을 보이며 이용되고 있다[5-7]. 이와 같은 현재 Concolic 테스팅 도구는 테스트 대상 프로그램, SMT 해법기, 테스트 생성 알고리즘을 같은 시스템 상에서 수행하도록 구성되어 있는데, 이를 임베디드 소프트웨어 테스팅에 적용하기 위해서는 다음 두 가지 방식이 가능하다:

- 임베디드 시스템에 Concolic 테스팅 도구 전체를 이식한 후 실행시키는 방법: SMT 해법기 등 많은 자원이 소요되어, 제한적인 하드웨어 상황에서 테스트 생성이 불가하거나 테스트 생산성이 현저히 낮은 문제가 발생함.
- 호스트 시스템에서 테스트 대상의 일부 코드만을 가상적으로 실행: 성능이 호스트 시스템에서 임베디드 타겟 시스템을 대신하는 테스트 스텁(test stub)을 작성해 일부 코드만을 실행시킬 수 있다. 이 경우, 임베디드 시스템을 모델링하는 비용이 소요되며, 실제 하드웨어 동작과 간극이 있을 경우, 결함 검출에 실패할 수 있다.

이와 같이 기존 Concolic 테스팅 도구가 임베디드 소프트웨어를 대상으로 갖는 한계점을 효과적으로 해결하기 위하여, 본 논문에서는 Concolic 테스팅 과정 중 (1) 하드웨어 및 임베디드 시스템에 종속적인 특성을 갖는 테스트검증 대상 프로그램의 실행 부분을 임베디드 타겟 시스템에서 수행하고, (2) 하드웨어나 시스템에 종속적이지 않게 실행이 가능한 테스트 케이스 생성 탐색전략 실행, 제약식 해법기 실행 과정은 계산성능이 좋은 호스트 시스템에 분산하여, 독립적인 단계를 동시적으로 실행하는 개선된 Concolic 알고리즘과 이를 구현한 도구를 소개한다.

본 연구는 기존에 개발된 C 프로그램 대상 Concolic 테스팅 도구인 CREST를 확장하여 제안한 기법을 Arm Cortex A54 아키텍처를 갖는 임베디드 타겟 시스템과 x86-64 아키텍처를 갖는 호스트 시스템을 대상으로 구현하였다. 실제 오픈소스 C 프로그램을 대상으로, 여러 가지 탐색 전략을 사용하여 수행한 실험을 통해 제안한 기법이, 임베디드 시스템 상에서 기존 Concolic 테스팅 도구 전체를 실행하는 기존 방식과 비교하여, 평균 2.5배 테스트 케이스 생성 속도를 향상함을 확인할 수 있었다.

본 논문이 탐구한 주제는 다음과 같이 요약할 수 있다:

- 테스트 대상 프로그램의 실행과 테스트 입력 실행 도출을 위한 계산을 2개의 서로 컴퓨팅 노드에서 동시에 실행할 수 있도록 기존의 순차적 Concolic 테스팅 알고리즘을 분산적 알고리즘으로 확장한 새로운 Concolic 테스팅 알고리즘을 제시하였다.
- 제안한 알고리즘을 실제 C 프로그램 대상 Concolic 테스팅 도구로 구현하는 방안을 제시하였고, 기존의 Concolic 테스팅 도구인 CREST를 기반으로 하여 실제 도구인 CRESTIVE-DX를 구현하였다.
- CRESTIVE-DX를 실제적인 임베디드 개발 환경에서 3개의 오픈소스 UNIX 유틸리티 프로그램을 대상으로 한 테스트 케이스 생성에 적용하였고, 그 성능을 기존의 Concolic 테스팅 도구와 비교 평가하는 실험을 수행하였다. 실험 결과, 제안한 기법이 임베디드 프로그램에 대한 Concolic 테스팅의 효율성을 증대함을 확인할 수 있었다.

```
ConcolicTesting(P, in_0)
1   in ← in_0
2   while ¬timeout() {
3       t ← start RunProgram(P, in)
4       φ ← receive from t
5       do {
6           ψ ← nextpath(φ)
7       } while UNSAT(ψ)
8       in ← SOLVE(ψ)
9   }

RunProgram(P, in)
11  φ ← run(P, in)
12  send φ to ConcolicTesting
```

Fig. 1. General Concolic Testing Algorithm

2. 배경: 순차적 Concolic 테스팅 알고리즘

Fig. 1은 기존 연구에서 제시해 온 일반적인 Concolic 테스팅의 수행 과정을 간략히 설명하는 알고리즘이다. Concolic 테스팅 기법은 테스트 대상 프로그램(P)와 초기 테스트 케이스(in_0)를 입력으로 받은 후, 테스팅에 주어진 시간(자원)이 소진될 때까지(2행) 테스트 입력 생성을 반복한다(3-8행). Fig. 1에서 설명하고 있는 바와 같이, 기본적인 Concolic 테스팅 알고리즘은 동일한 시스템 내에서 각 단계를 순차적으로 실행함으로써 구동되도록 설계되었다.

각 테스트 케이스의 생성을 위해서는 우선 직전에서 생성한 테스트 케이스(처음의 경우 in_0)로 테스트 대상 프로그램을 실행하는 과정이 필요하다. 이를 위해서, ConcolicTesting은 새로운 프로세스(t)를 생성(fork) 한다(3행). 생성된 프로세스(RunProgram)는 테스트 대상 프로그램을 실행시켜 해당 실행 경로의 조건식(ϕ)를 추출(11행)하여 이를 ConcolicTesting에 전달한다(12행). ConcolicTesting은 추출한 실행 경로 조건식을 변형하여 차기 목표 경로조건식(ψ)을 도출 (5행)한 다음, SMT 해법기로 목표 경로식이 만족 가능한 지를 검사하고(6행), 만족 가능할 경우, 이를 만족하는 해를 구하여 테스

```
Host(P, in_0)
1  in ← in_0
2  while ¬timeout() {
3     send in to Target
4     φ ← receive from Target
5     do {
6        ψ ← nextpath(φ)
7     } while UNSAT(ψ)
8     in ← SOLVE(ψ)
9  }
```

(a) Algorithm that Run on the Host System

```
Target(P, in_0)
11 while ¬timeout() {
12    t ← start RunProgram(P)
13    in ← receive from Host
14    send in to t
15    φ ← receive from t
16    send φ to Host
17 }

RunProgram(P) {
21 in ← receive from Target
22 φ ← run(P, in)
23 send φ to Target
```

(b) Algorithm that Run on the Embedded Target System

Fig. 2. Distributed Concolic Testing Algorithm

트 케이스를 생성하는 단계(7행)를 거쳐 이루어진다. 이때, 테스트 대상 프로그램을 실행하는 run 연산(3행)은 테스트 대상 프로그램의 실행 과정과 메모리 상태를 관찰하여 실행 경로식을 생성하는 과정을 포함한다. 관찰한 경로 조건식에서 차기 목표 경로조건식을 도출하는 nextpath 연산(5행)은 탐색 전략을 구현한 것으로, 관찰한 경로 조건식을 변형하여 도출 가능한 여러 대안 중 목표달성(예: 분기 커버리지)에 유리할 것으로 판단되는 조건식을 선택하는 휴리스틱이다(예: DFS, CFG, Random Negation). 마지막으로 UNSAT과 SOLVE 명령은 Z3와 같은 SMT 해법기를 실행하는 명령이다.

기존에 제시된 Concolic 테스팅 기법은 Fig. 1에서 설명하는 Concolic 테스팅 알고리즘 전체를 단일한 환경에서 수행하는 상황에 한정하여 개발되었다. 이로 인하여, 테스트 대상 프로그램 실행을 위해 계산 속도, 메모리, 자원 활용에 제약이 큰 임베디드 프로그램을 대상으로 기존 Concolic 테스팅 기법을 적용하는데 효과성 및 효율성에 있어서 한계가 큰 상황이다(1절에서 논의). 이러한 문제를 해결하기 위해서는 테스트 대상 프로그램의 실행과 테스트 실행 도출을 위한 계산이 서로 다른 환경에서 동시적으로 수행 가능 하도록 Concolic 테스팅 알고리즘의 확장이 요청된다.

3. 분산형 Concolic 테스팅 기법

본 논문은 실제 임베디드 시스템 상에서 테스트 대상 프로그램을 실행하면서 동시에 탐색전략 휴리스틱, SMT 해법기 등 계산비용이 높은 Concolic 테스팅 알고리즘을 효율적으로 구동하기 위해서, Fig. 1의 기본 알고리즘을 호스트 시스템 실행 부분과 타겟 시스템 실행 부분으로 분산하고, 독립적인 부분은 병렬화한 알고리즘을 제안한다.

타겟(target) 시스템은 테스트 대상 프로그램이 내장되고 운영될 실제 임베디드 시스템이다. 임베디드 시스템을 모델링하기 위한 비용을 없애고 미탐이나 오탐 없이 정확한 결함 검출이 가능한 테스팅을 위해서 테스트 대상 프로그램(P)를 실제 하드웨어/시스템 종속적 명령의 직접적인 실행이 가능한 타겟 시스템에서 실행하여 실제적인 실행 경로식을 추출해야 다. 반면, 임베디드 시스템 특성상 타겟 시스템은 속도가 느리거나 메모리 자원에 제약이 크기 때문에, 테스트 대상 프로그램 실행 이외의 추가적인 컴퓨팅/메모리 자원 사용이 최소화되어야 한다. 호스트(host) 시스템은 계산 속도가 빠르고 메모리 자원이 많은 시스템으로, nextpath 연산이나 SMT 해법기가 수행하는 UNSAT, SOLVE 연산 등 하드웨어/시스템과 독립적인 연산을 맡아서 수행하게 된다.

Fig. 2는 본 논문이 제안하는 분산형 Concolic 테스팅 알고리즘으로, Fig. 2(a)의 Host는 호스트 시스템에서, Fig. 2(b)의 Target은 임베디드 타겟 시스템에서 동시에 실행되는 과정이다. 이 때, Fig. 2(b)의 RunProgram은 임베디드 타겟 시스템에서 Target에 의하여 하나의 프로세스(process)로 생성되어 실행되는 부분을 나타낸다. 테스트 검증 대상과 초기 테스트 케이스가 주어지면 Host는 호스트 시스템에서 Target은 타겟 시스템에서 동시에 실행된다.

Fig. 2의 Host는 Fig. 1의 ConcolicTesting과 같이, timeout이 만족될 때까지(2행) 테스트 케이스 생성을 반복한다. 각 테스트 케이스 생성에서 Host는 직전 단계에서 생성한 테스트 입력(in)을 타겟 시스템으로 전송(send)한다(3행). 그 후, Host는 타겟 시스템으로부터 경로 조건식(ϕ)가 주어지길 기다려서 타겟 시스템으로부터 전달받은 후, Fig. 1에서와 마찬가지로, 탐색 전략(nextpath, 6행) 및 SMT 해석기를 이용해 새로운 테스트 케이스를 계산한다(7-8행).

Fig. 2의 Target은 타겟 시스템에서 timeout이 만족될 때까지(11행) 반복적으로 타겟 프로그램을 실행시키는 역할을 한다. 각 테스트 케이스 실행을 위하여, Target은 먼저 타겟 시스템에 새로운 프로세스를 생성하여 RunProgram을 시작한다(12행). 새로운 프로세스 생성이 끝나고 실행된 RunProgram은 Target으로부터 추후 테스트 케이스를 전달받기를 기다린다(21행). 이와 동시에 Target은 Host로부터 테스트 케이스를 기다려 수신한다(13행). Target은 Host로부터 새로운 테스트 케이스를 수신 받은 후, 생성한 프로세스(t)에 전달(14행)하여 테스트 대상 프로그램이 실행이 되도록 하고(22행), 해당 프로세스가 실행 결과 전송(23행)하는 경로조건식을 수신하기까지 기다린다(15행). Target은 RunProgram으로부터 수신한 경로실행식을 Host에 전달함으로써 하나의 테스트 케이스 실행 사이클이 종료된다.

이 때, Host에서 새로운 테스트 케이스를 생성하고 (5-8행) 이를 Target에서 수신하는(13행) 단계와 Target에서 테

스트 대상 프로그램을 실행을 위해 새로운 프로세스를 만드는 단계(12행)은 독립적인 과정이므로, 제안하는 분산 알고리즘에서는 이들을 각각 호스트 시스템과 타겟 시스템에서 동시에 병렬적으로 실행한다. 실제 이 두 과정은 SMT 해법기를 구동하고, 시스템콜을 호출하여 프로세스 생성을 수행하기 위해 시간이 많이 소요되는 과정으로, 이들을 병렬 처리함으로써 전체 Concolic 테스팅 시간을 단축한다.

4. 분산 Concolic 테스팅 도구 구현 및 실험

4.1 도구의 구현

본 연구는 제시한 분산 Concolic 테스팅 기법인 CRESTIVE-DX[1]는 C 프로그램 대상 오픈소스 Concolic 테스팅 도구인 CREST[8]를 확장한 구현하였으며, Arm Cortex A54 아키텍처의 임베디드 타겟 시스템과 x86-64 아키텍처의 호스팅 시스템 상황에 적용하였다.

CRESTIVE-DX는, 2절에서 설명한 바와 같이, 단일 시스템에서 순차적인 과정을 통해 Concolic 테스팅을 수행하도록 설계되어 있다. CREST는 크게 (1) 테스트 대상 프로그램 수정 모듈, (2) 동적 심볼릭 실행 수행 모듈, 그리고 (3) 테스트 대상 프로그램 실행을 통한 테스트 케이스 생성 모듈, 이렇게 세 가지 모듈로 구성되어 있으며, 각 모듈의 대략은 다음과 같다:

- 테스트 대상 프로그램 수정 모듈: C 프로그램 소스코드의 각 명령에 해당 명령의 실행 정보를 추출하는 탐침(probe)을 삽입(instrument)한 후[9], 테스트 대상 프로그램을 빌드하는 과정에서 탐침에 동적 심볼릭 실행을 수행하는 라이브러리를 연결(bind)한다.
- 동적 심볼릭 실행 수행 모듈: 테스트 대상 프로그램에 삽입된 탐침에 의해서 심볼릭 실행의 각 단계에 대한 명령이 호출되어, 테스트 대상 프로그램이 실행하는 경로에 조건식이 도출된다. 결과적으로, 테스트 대상 프로그램의 실행이 종료될 때, 해당 경로에 대한 조건식이 부산물로 생성된다.
- 테스트 케이스 생성 모듈: 테스트 케이스로 테스트 대상 프로그램을 실행시켜 경로조건식을 생성하고, 생성된 경로 조건식을 Concolic 탐색 전략[9]과 SMT 해법기를 이용해 새로운 테스트 케이스를 생성하는 전체 과정을 운전한다.

본 연구에서는 3절에 제안한 분산형 Concolic 테스팅 기법을 구현하기 위해 CREST를 다음과 같이 수정/확장하여 CRESTIVE-DX를 구성하였다:

- **경로조건식 인코딩 통일:** 임베디드 타겟 시스템에서 테스트 대상 프로그램의 실행에서 도출된 경로조건식을 호스트 시스템에서 처리하기 위해 경로조건식 도출 단계에서부터 아키텍처와 관계없이 인코딩이 동일하게 이루어지도록 수정하였다. 구체적으로는, LIA 제약식 표현에 사용되는 숫자는 일관적으로 8 바이트 정수형으로 표현되도록 통일하고, 시스템에서 이보다 작은 숫자를 할당하여 사용할 경우 변환(upcasting)되어 저장되도록 하였다.
- **테스트 대상 프로그램 실행 모듈과 테스트 케이스 생성 모듈의 분할:** 기존의 테스트 케이스 생성 모듈에서 테스트 대상 프로그램을 실행시키는 부분은 임베디드 타겟 시스템에서 실행되는 프로그램으로, 탐색 전략 및 SMT 해법기를 통한 테스트 케이스 생성 부분은 호스트 시스템에서 실행되는 프로그램으로 분할하였다. 그리고 분할된 두 부분은 TCP 통신을 통해서 테스트케이스 교환, 경로조건식 교환이 이루어지도록 하였다. 이 때, 교환되는 테스트케이스와 경로조건식 데이터 크기가 비교적 작은 편이기 때문에, 두 모듈 간의 신속한 통신이 가능하도록 Nagle 알고리즘을 사용하지 않는 TCP 연결을 구성하여 사용하였다(i.e., TCP_NODELAY).
- **테스트 프로그램 실행을 위한 프로세스 생성과 테스트 케이스 생성의 병렬화:** 2.2절에서 설명한 바와 같이, 테스트 프로그램 실행을 위해 새로운 프로세스를 만드는 과정은 운영체제가 여러 자원을 할당하는 작업을 포함하므로 임베디드 타겟 시스템 측면에서 많은 시간이 소요되는 과정이다. 또한, 주어진 경로 조건식을 바탕으로 차기 테스트 케이스를 만드는 과정 역시 호스팅 시스템 입장에서는 많은 시간이 소요되는 시간이므로, 이 둘이 동시적으로 실행되도록 프로그램 구조를 구성하였다. 즉, 호스트 시스템에서 i번째 테스트 케이스를 도출하는 동시에, 타겟 시스템에서는 i번째 테스트 대상 프로그램 실행을 위해 프로세스 생성을 마치고 실제 테스트 대상 프로그램을 구동한다. 이 때, 테스트 대상 프로그램은 첫 번째 입력을 받는 지점에서 i번째 테스트 케이스가 전송되기를 기다렸다가, 호스트 시스템으로부터 테스트 케이스가 주어질 때 실행을 시작하도록 동기화한다.

Table 1. Time Spent Creating a Test Case(sec)

	Random Negation		Uniform Random		DFS	
	CRS	CDX	CRS	CDX	CRS	CDX
Grep	190.2 (15.38)	**77.0 (39.0)**	126.2 (23.8)	**50.3 (59.6)**	177.9 (16.9)	**67.1 (44.7)**
Awk	169.77 (29.5)	**106.5 (46.9)**	161.6 (30.9)	**101.5 (49.3)**	194.8 (25.7)	**117.3 (42.6)**
Ed	121.0 (41.3)	**59.6 (83.9)**	125.8 (39.7)	**59.9 (83.5)**	151.5 (33.0)	**59.3 (84.3)**

Table 2. Time Spent per Course using Uniform Random(sec)

	(a) Search Strategy		(b) SMT Sover		(c) Execution and Communication	
	CRS	CDX	CRS	CDX	CRS	CDX
Grep	7.5	**1.0**	107.8	**62.6**	177.9	**65.1**
Awk	10.3	**1.6**	64.1	**35.2**	194.8	**117.3**
Ed	9.2	**1.0**	40.7	**19.7**	101.7	**38.6**

1) CRESTIVE-DX (CREST ImproVed-Distributed eXecution): https://github.com/ARISE-Handong/crestive/tree/dx

이외에도 CRESITVE-DX는 CREST와 달리 SMT 해법기로 Z3[4]를 이용하도록 수정하였으며, Concolic 드라이버 작성의 편의를 위한 API를 추가했다.

4.2 실험 셋팅

본 연구에서는 제안한 기법을 구현한 도구가 실제 임베디드 소프트웨어 테스팅 상황에서 Concolic 테스팅을 가능하게 하며, 기존 방식에 비해서 테스트 케이스 생산 효율성을 향상하였는지 평가하기 위해 다음과 같은 실험을 구상했다.

- **테스팅 환경:** 임베디드 타겟 시스템은 Arm Cortex A54 아키텍쳐인 Raspberry Pi 3 B+로, CPU의 클럭속도는 1.4 GHz이며 메모리는 0.94 GB로 Raspbian 4.19.75 운영체제로 사용하는 시스템을 사용했다. 호스팅 시스템은 x86-64 아키텍쳐인 Intel CPU로 클럭속도는 3.3 GHz이며 메모리는 8 GB를 갖고 Ubuntu 16.04를 운영체제로 사용하는 시스템을 이용하였다. 두 시스템은 1 Gbps 스위치를 사용하여 Ethernet으로 네트워킹을 구성하였다.
- **비교 대상:** 본 실험에서는 제안한 기법을 임베디드 타겟 시스템과 호스팅 시스템에 설치한 방법(CRESTIVE-DX 혹은 'CDX'로 지칭)와 SMT 해법기를 포함하여 기존 CREST 전체를 임베디드 타겟 시스템 상에서 수행한 방법('CRS'로 지칭)을 구성하였다. CRESTIVE-DX와 CREST 모두 SMT 해법기로 Z3 4.4.1를 사용하였다.
- **테스트 대상 프로그램:** 본 실험에서는 Concolic 테스팅 연구에서 널리 사용되는 오픈소스 C 프로그램인 Grep 1.2 (10959 LOC)와 Busybox 내 Awk (3322 LOC), Ed (1115 LOC)를 테스트 대상으로 사용하였다. 이들 프로그램은 Concolic 테스팅 연구의 실험에서 테스트 대상으로 자주 사용되는 프로그램이다. Concolic 테스팅 수행을 위해 Grep의 경우 심볼릭 변수 25개, Awak은 심볼릭 변수 30개 Ed는 심볼릭 변수 30를 선언하는 테스트 드라이버를 만들어 사용하였다. 초기 테스트 입력은 간단한 형태의 임의의 문자열을 사용하였다.
- **테스트 생성:** 각 테스트 대상 프로그램에 대하여 CREST와 CRESTIVE-DX를 세 가지 탐색 전략, 즉 DFS, Uniform Random, Random Negation을 적용하여 Grep은 3000개, Awak과 Ed는 각각 5000개의 테스트 케이스 생성을 수행하고, 이에 소요된 시간을 측정하였다. 이 때, 확률적 요소가 포함된 Uniform Random 및 Random Negation 탐색 전략의 경우, 동일한 실험을 총 10회 반복하여 결과의 평균을 구하였다. 최종적으로는 각 상황에서, CREST와 CRESTIVE-DX가 단위시간(1초) 당 생성한 테스트 케이스 생성 개수를 비교하였다.

4.3 실험 결과

Table 1은 CREST와 CRESTIVE-DX가 3개의 테스트 대상 프로그램에 대하여 3종의 탐색 전략을 이용하여 Grep은 3000개, Awk과 Ed는 각각 5000개 테스트 케이스를 생성하는 데 소요된 시간과 1초당 테스트 케이스 생성 개수(괄호)를 나타낸다.

Table 1의 결과는 본 연구가 제안한 기법(CDX)는 기존 CREST를 그대로 적용한 방법과 비교하여, 최소 1.59배(Awk, Random Negation)에서 최대 2.64배(Grep, DFS) 초당 테스트 케이스 생성 속도가 향상하였음을 알 수 있다. 특히 DFS 탐색 전략을 사용했을 때, 평균적인 테스트 케이스 생성속도 향상은 2.29배로, 가장 큰 폭의 향상이 관찰되었다. 이는 DFS 탐색 전략 수행에 소요 되는 계산이 Random Negation이나 Uniform Random보다 많기 때문에, 컴퓨팅 성능이 좋은 환경에서 실행한 효과가 더 두드러지게 나타난 것으로 추정된다.

Table 2는 Uniform Random 전략을 사용하는 상황에서 CRS와 CDX가 소요하는 시간을 (a) 탐색 전략 실행 시간, (b) SMT해법기 구동 시간, (c) 테스트 대상 프로그램 실행 시간 및 네트워크 통신 시간(네트워크는CDX만 해당)을 나누어 측정한 결과다. 실험 결과, 탐색 전략 실행이 소요한 시간은 기존의 10~16% 수준으로 감소함을 보이는데, 이는 탐색 전략 또한 SMT 해법기와 마찬가지로, 계산 비용이 많이 소요되는 과정이기 때문이다. 또한, 상대적으로 더 뛰어난 호스트의 컴퓨팅 성능 덕분에 SMT 해법기 실행 시간은 기존의 48~58%로 감소하였다. 테스트 대상 프로그램 실행 및 통신에 소요한 시간은 기존의 37~60% 수준으로 감소하였다. CRESTIVE-DX에는 기존 기법보다 통신으로 오버헤드가 발생하였으나, 병렬화의 효과로 대상 프로그램 실행 및 통신 전반에 소요한 시간은 오히려 감소함을 알 수 있다.

추가로, 위 실험에서 DFS와 같이 비결정적 요소가 없는 결정적(deterministic)인 탐색 전략으로 테스트 케이스 생성 과정에서 탐색하는 실행 경로를 일대일로 면밀히 비교한 결과, CDX가 CRS와 동일한 경로를 탐색함을 작동함을 확인할 수 있었으며, 이를 통해 CDX과 CRS의 알고리즘을 올바르게 구현함을 검증할 수 있었다. 또한, 비결정성(non-determinism)이 있는 탐색 전략을 사용한 실험에서, 같은 테스트 대상 프로그램을 같은 탐색전략으로 실행하는 CRS와 CDX의 커버리지 달성을 비교한 결과, 대체로 같은 값이 나옴을 추가로 확인할 수 있었다.

5. 결 론

본 논문은 임베디드 소프트웨어의 효과적이고 효율적인 테스팅을 임베디드 타겟 시스템과 호스트 시스템의 분산적 Concolic 테스팅 기법을 제안하고, 이를 CREST 도구를 확장하여 구현하여 테스트 생성 속도가 1.59~2.64배 향상됨을 실험적으로 확인한 결과를 소개하였다.

본 논문에서 제안한 CRESTIVE-DX는 TCP/IP가 지원되지 않는 임베디드 시스템을 대상으로 UART, JPEG 통신을 활용한 분산 Concolic 테스팅 환경을 구성하는 연구, MCU와 같이 극단적으로 하드웨어 자원이 제약적인 임베디드 시스템을 대상으로 한 Concolic 테스팅 적용, 탐색 알고리즘의 병렬화를 통한 Concolic 증대 등 임베디드 시스템을 대상으로 한 테스팅 자동화 연구에 향후 활용할 수 있다.

References

[1] N. Zhang, S. Demetriou, X. Mi, W. Diao, K. Yuan, P. Zong, F. Qian, X. Wang, K. Chen, Y. Tian, C. A. Gunter, K. Zhang, P. Tague and Y. Lin, "Understanding IoT Security Through the Data Crystal Ball: Where We Are Now and Where We Are Going to Be," *CoRR*, Vol. abs/1703.09809, 2017.

[2] K. Sen, D. Marinov and G. Agha, "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-13)*, pp.263-272, Sep. 2005.

[3] C. Cadar, D. Dunbar and D. Engler, "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)*, pp.209-224, Dec. 2008.

[4] The Z3 Theorem Prover [Internet], https://github.com/Z3Prover/z3

[5] Y. Kim, Y. Kim, T. Kim, G. Lee, Y. Jang and M. Kim, "Automated Unit Testing of Large Industrial Embedded Software Using Concolic Testing," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE'13)*, pp.519-528, Nov. 2013.

[6] Y. Park, S. Hong, M. Kim, D. Lee and J. Cho, "Systematic Testing of Reactive Software with Non-deterministic Events: A Case Study on LG Electric Oven," in *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, Vol.2, pp.29-38, May 2015.

[7] Y. Kim, D. Lee, J. Baek and M. Kim, "Concolic Testing for High Test Coverage and Reduced Human Effort in Automotive Industry," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP'19)*, pp.151-160, May 2019.

[8] J. Burnim, CREST [Internet], https://github.com/jburnim/crest

[9] G. C. Necula, S. McPeak, S. P. Rahul and W. Weimer, "CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs," in *Proceedings of the 11th International Conference on Compiler Construction (CC'02)*, pp.213-228, Apr. 2002.

[10] J. Burnim and K. Sen, "Heuristics for Scalable Dynamic Test Generation," in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*, pp.443-446, Sep. 2008.

임 혜 린

https://orcid.org/0000-0001-5055-8772
e-mail : 22031007@handong.edu
2020년 한동대학교 전산전자공학부(학사)
2020년 ~ 현 재 한동대학교 정보통신공학과 석사과정
관심분야 : 소프트웨어 공학, 테스팅 자동화, 심볼릭 실행

최 한 솔

https://orcid.org/0000-0001-6272-618X
e-mail : 21831008@handong.edu
2018년 한동대학교 전산전자공학부(학사)
2018년 ~ 현 재 한동대학교 정보통신공학과 석사과정
관심분야 : 소프트웨어 공학, 테스팅 자동화, 심볼릭 실행

김 효 림

https://orcid.org/0000-0002-4094-3113
e-mail : 21600193@handong.edu
2016년 ~ 현 재 한동대학교 전산전자공학부 학사과정
관심분야 : 소프트웨어 공학

홍 신

https://orcid.org/0000-0003-4217-6031
e-mail : hongshin@handong.edu
2007년 한국과학기술원 전산학(학사)
2010년 한국과학기술원 전산학(석사)
2015년 한국과학기술원 전산학(박사)
2016년 ~ 현 재 한동대학교 전산전자공학부 조교수
관심분야 : 소프트웨어 공학, 프로그램 분석, 자동 테스트 입력 생성, 자동 소프트웨어 디버깅