목적 지향 콘콜릭 테스팅
(Goal-oriented Concolic Testing)

정 인 상 † 박 정 규 ††
(Insang Chung) (Jungkyu Park)

요 약 콘콜릭 테스트는 높은 테스트 커버리지를 달성하기 위해 실제 프로그램 수행과 심볼릭 수행을 결합하여 테스트 데이터를 생성한다. CREST는 콘콜릭 테스팅을 구현한 대표적인 open-source인 테스트 도구이다. 그러나 현재 CREST는 기본적으로 프로그램의 모든 가능한 실행 경로들을 탐색하는 것을 목적으로 한다. 이 때문에 특정 분기 또는 블록만을 테스트하는 경우에는 비효율적일 수 있다. 이 논문에서는 프로그램 상의 한 분기 또는 블록을 주고 이를 실행할 수 있는 테스트 데이터를 생성하는 목적 지향 콘콜릭 테스트 방법을 제안한다.

키워드 : 콘콜릭 테스팅, 목적 지향 테스트 데이터 생성, CREST

Abstract Concolic testing generates test data by combining concrete program execution and symbolic execution to achieve high test coverage. CREST is a representative open-source test tool implementing concolic testing. Currently, however, CREST aims at exploring all possible execution paths. In case of testing a specific branch or block, thus, it can be ineffective. This paper suggests a goal-oriented concolic testing that generates test data to execute a given branch or block.

Key words : Concolic Testing, Goal-oriented Test Data Generation, CREST

1. 서 론

테스트 데이터 생성 방법들은 이 방법들은 사용자가 테스트할 프로그램 경로를 제공하는지에 따라 경로 지향(path-oriented) 방법과 목적 지향 방법(goal-oriented)으로 분류할 수 있다. 경로지향 방법은 프로그램의 제어 흐름이 복잡하거나 반복문이 존재하는 경우에 경로를 선정하는 작업 자체가 용이하지 않을 뿐만 아니라 주어진 프로그램 경로가 실행이 불가능한 경우, 즉 경로를 실행할 수 있는 입력 값이 존재하지 않는 경우에는 입력 값을 찾기 위해 많은 시간과 노력이 소요될 수 있다는 단점이 있다. 이와는 달리 목적 지향(goal-oriented) 테스트 데이터 생성 방법은 특정 프로그램 경로를 제공하는 대신에 프로그램 상의 한 블록 또는 분기를 주고 이를 실행할 수 있는 테스트 데이터를 생성하는 방법이다[1]. 따라서 사용자가 일일이 프로그램 경로를 선정하는 부담이 없으며 경로 기반 테스트에서는 사용자가 정한 프로그램 경로가 실행 불가능하다면(즉, 주어진 경로를 실행할 수 있는 입력 값이 존재하지 않는다면) 사용자가 다른 경로를 선택해야 하였으나 목적 기반 테스트에서는 주어진 프로그램 포인트(i.e., 블록 또는 분기)를 실행할 수 있는 다른 경로를 자동으로 탐색하여 입력 값을 생성할 수 있다.

최근에는 특정 프로그램 경로나 포인트 대신에 프로그램의 모든 경로들을 탐색하는 콘콜릭 테스트가 제안되었다[2]. 이 방법은 동적 테스트 방법과 심볼릭 수행을 결합하여 높은 테스트 커버리지를 달성하기 위해 개발되었다. 콘콜릭 테스트는 우선 (무작위로 생성된) 입력으로 프로그램을 수행한다. 이 때 입력에 의해 실행된 경로를 따라 심볼릭 수행을 하여 프로그램 경로 제약 조건을 생성한다. 이렇게 생성된 경로 제약 조건은 프로그램의 커버리지를 높이기 위해 이전과는 다른 프로그램 경로를 수행할 수 있는 테스트 데이터를 산출하도록

· 본 연구는 2010년도 한성대학교 교내연구비 지원과제임

† 종신회원 : 한성대학교 컴퓨터공학과 교수
insang@hansung.ac.kr
†† 비 회 원 : 한성대학교 컴퓨터공학과
mjk0208@naver.com

논문접수 : 2010년 7월 5일
심사완료 : 2010년 8월 31일

수정한다. 이러한 과정은 프로그램의 모든 경로가 실행되거나 사용자가 지정한 종료 조건을 만족할 때까지 반복된다.

CREST는 콘콜릭 데스트를 기반으로 C 프로그램을 대상으로 테스트 데이터를 자동으로 생성하는 대표적인 open-source 테스트 도구이다[3]. 그러나 현재 CREST는 기본적으로 프로그램의 모든 가능한 실행 경로들을 탐색하는 것을 목적으로 한다. 이 때문에 특정 분기 또는 블록만을 테스트하는 경우에는 비효율적일 수 있다. 이 논문에서는 프로그램 상의 한 블록 또는 분기를 주고 이를 실행할 수 있는 테스트 데이터를 생성하는 목적 지향 콘콜릭 테스트 방법을 제안한다. 제안된 방법은 프로그램의 자료 흐름 정보를 이용하여 주어진 프로그램 포인트의 실행에 영향을 미치는 영문장들을 식별한다. 실험 결과를 통해 제안된 기법이 효율적으로 테스트 데이터 생성함을 보인다.

이 논문은 다음과 같이 구성된다. 2장에서는 콘콜릭 테스트를 수행하는 절차에 대해 보다 자세하게 기술한다. 3장에서는 이 논문에서 제안한 목적 지향 콘콜릭 테스트 방법에 대해 설명한다. 4장에서는 실험결과를 기술하고 제안된 기법의 효율성을 보인다. 마지막으로 5장에서 결론 및 향후 연구에 대해 소개한다.

2. CREST

CREST는 당시 UC Berkely 대학에서 재직중이던 K. Sen 교수에 의해 개발된 콘콜릭 테스트 도구이다[3]. 이 도구는 심볼릭 수행을 위해 C로 작성된 프로그램을 입력으로 받아 CIL[4]를 사용해서 프로브(probe)를 삽입하고 보다 분석하기 단순한 프로그램으로 변환한다.. 테스트 데이터 생성을 위한 첫 번째 단계로 우선 각 입력 변수에 무작위 값이 할당되어 프로브가 삽입된 프로그램이 실행된다. 이 때 실행된 경로를 따라 심볼릭 수행을 하며 심볼릭 수행을 한 결과는 심볼릭 맵을 통해 관리된다. 심볼릭 맵은 각 변수에 대한 심볼릭 표현식이 있으며 배정문을 만날 때마다 삽입된 프로브를 통해 이 표현식은 갱신된다. 물론 프로그램 실행이 처음 시작된 경우에는 각 입력변수에 대한 심볼릭 표현식은 어떤 배정문도 실행되기 전이기 때문에 단순한 심볼릭 값이 된다.

심볼릭 맵외에도 CREST는 실행 경로에 따른 경로 제약식을 저장하기 위한 자료 구조 Φ를 관리한다. Φ는 초기 값으로 'true'가 주어진다. 만약 조건문 'if p then ...'을 처리하는 경우에 실제 실행된 경로가 p가 참이 될 때에는 조건식 p에 해당하는 심볼릭 표현식 ξ(p)이 현재 Φ에 있는 식과 'and'로 결합되어 Φ에 저장된다. 실행된 경로가 p가 거짓인 경우에는 심볼릭 표현식 'not ξ(p)'이 결합된다. 즉, Φ를 만족하는 입력 값은 지금까지 실행된 경로를 실행하는 테스트 데이터가 된다.

CREST는 이전에 실행된 프로그램 경로와는 다른 경로를 실행하기 위해 Φ를 구성하는 심볼릭 표현식에서 하나를 선택하여 부정하여 새로운 경로 제약식을 생성한다. 예를 들어 $<p_0, p_1, ..., p_{k-1}>$이 실제 실행된 경로라고 하자. CREST는 분기 $p_j(0 \le j < k)$를 선정한다. 이 논문에서는 분기를 테스트 문장과 테스트 문장이 참 또는 거짓이 될 때 실행되는 첫 번째 문장으로 구성된 쌍으로 정의한다. 이 때 ϕ_l을 이 분기를 수행하기전의 경로 제약식이라 하고 ϕ_t를 이 분기에 의해 생성된 심볼릭 표현식이라고 하자. CREST는 이전 경로와는 다른 경로를 실행하는 테스트 데이터를 찾기 위해 $(\wedge_{\phi \in \Phi l}\phi) \wedge (\text{not } \phi_t)$를 만족하는 해를 구한다. 만약 이 새로운 경로 제약식을 만족하는 입력 값으로 프로그램을 실행하면 실행 경로는 $<p_0, p_1, ..., \text{paired}(p_j), p'_{j+1}, ..., p'_m>$일 것이다. 여기에서 paired(pj)는 p_j의 짝 분기를 나타낸다. 즉, 어떤 조건식에서 p_j가 참인 분기라면 paired(p_j)는 거짓인 분기를 나타내며 p_j가 거짓인 분기라면 paired(p_j)는 참인 분기를 나타낸다. CREST는 이 과정을 모든 경로들에 대한 테스트 데이터를 찾을 때까지 반복하거나 지정한 회수만큼 반복한다.

3. 목적지향 콘콜릭 테스트

이 장에서는 모든 경로를 실행하는 테스트 데이터를 생성하는 콘콜릭 테스트를 수정하여 특정 프로그램 분기를 실행하는 테스트 데이터를 찾는 목적지향 콘콜릭 테스트 방법(GCT, Goal-oriented Concolic Test)에 대해 기술한다.

3.1 용어 정의

이 절에서는 목적 지향 콘톨릭 테스트, 즉 GCT 수행 절차에 필요한 용어들을 정의한다. 분기 l에 대해서 paired(l)이 실행 경로 π상에 있고 l를 통해 목표 분기 t에 도달할 수 있는 프로그램 경로가 존재할 때 분기 l은 실행 경로 π에 관해 유도 분기(guiding branch)라 한다. $GB_t(\pi)$는 실행 경로 π상에 존재하는 목표 분기 t에 관한 유도 분기들의 집합을 나타낸다. $D(s_i)$는 문장 s_i에서 정의된 변수 집합을 나타내고 $U(s_j)$는 문장 s_j에서 사용 또는 참조되는 변수 집합을 나타낸다. 분기 l=(p, q)에 대해 <l>은 문장 q를 통해 도달할 수 있는 문장들의 집합을 나타낸다. 즉, 테스트 문장 p에 대해 이행적으로 제어 의존적(transitive control-dependent)[5]인 문장들 등에서 q가 실행될 때 실행될 수 있는 문장들이다. 다음은 GCT가 기반으로 하고 있는 자료 흐름 개념에 대한 정의이다.

정의 1. 문장 s_i는 s_j에 다음 조건을 만족할 때 경로 $p=(s_1, s_2, ..., s_k)$를 통하여 *직접적으로 영향을 미친다*고

말한다.

• $v \in U(s_i)$이고 $v \in D(s_j)(i<j)$인 변수 v가 존재한다.

• 모든 $s_m(j<m<i)$에 대해 $v \notin D(s_m)$

정의 2. 경로 $p=(s_{i1}, s_{i2}, ..., s_{in})$로부터 다음 조건을 만족하는 시퀀스 $<k_1, k_2, ..., k_r>$를 추출할 수 있다면 문장 s_{i1}는 s_{in}에 *영향을 미친다*고 말한다.

• $s_{i1}=k_1$, $s_{in}=k_r$

• for all j, $1<j<r$, k_j는 k_{j+1}에 직접적으로 영향을 준다.

3.2 GCT 수행절차

이 논문에서 제안하는 목적 지향 콘콜릭 테스트는 자료 흐름 정보를 이용하여 주어진 분기의 실행에 영향을 미칠 수 있는 문장들을 식별한다. GCT는 이렇게 식별된 문장들의 실행을 제어하여 목표 분기를 수행하도록 유도한다. 다음은 3.1절에서 기술한 정의에 바탕을 두고 특정 분기를 실행하는 테스트 데이터를 찾아내는 GCT 수행 절차이다.

1) 무작위로 생성된 입력을 사용하여 콘콜릭 테스트 수행한다. 이 때 실행된 프로그램 경로를 π_e라 한다.
2) 실행경로 π_e가 목표분기 t에 도달하였다면 종료한다.
3) 만약 목표 분기 t에 도달하지 않았다면 $GB_t(\pi_e)$로 부터 유도 분기(guiding branch)를 l_g를 선정한다. 이 논문에서는 목표 분기와의 거리를 선정 기준으로 사용하였으며 가장 가까운 분기를 선정하였다.
4) l_t paired(l_g)
5) 이 때 ϕ_l을 l_t를 수행하기전의 경로 제약식이라 하고 ϕ_t를 l_t에 의해 생성된 심볼릭 표현식이라고 하자.
6) E $(\wedge_{\phi \in \Phi l}\phi) \wedge (\text{not } \phi_t)$.
7) E를 만족하는 해를 구하여 이를 입력으로 프로그램을 실행한다. 만약 실행 경로가 목표 분기 t를 수행한다면 종료한다. 만약 이전 경로와 다른 경로 π_e'를 수행한다면 π_e π_e'에 대해 2)번 단계부터 수행한다.
8) 만약 E를 만족하는 해를 구할 수 없다면 $GB_t(\pi_e)$가 공집합이 될 때까지 단계 3)번을 수행한다.
9) 만약 이전 경로 π_e와 동일한 경로를 수행한다면 분기 l_t의 테스트 문장에 *영향을 미치는* 문장들을 식별한 후 이 문장들을 $AFF(l_t)$라 한다. 이 때 실행 경로 π_e 상에서 분기 l_t 바로 이전에 실행된 분기 l에 대해서 다음을 검사한다.
   9-a) 만약 분기 l 때문에 l_t의 실행이 영향을 받았다면, 즉, $<l> \cap AFF(l_t) \neq \varnothing$라면 l_t l로 한 후에 5)번 단계부터 수행한다.
   9-b) 만약 $<l> \cap AFF(l_t)=\varnothing$이고 $<paired(l)> \cap AFF(l_t) \neq \varnothing$라면 9-a)와 마찬가지로 l_t l로 한 후에 5)번 단계부터 수행한다.
   9-c) 만약 $<l> \cap AFF(l_t)=\varnothing$이고 $<paired(l)> \cap AFF(l_t) =\varnothing$라면 실행 경로 π_e상에서 분기 l 바로 이전에 실행된 분기 l'에 대해서 l l' 한 후에 이 단계를 반복 수행한다. 만약 π_e상에 더 이상의 분기가 존재하지 않는다면 절차를 종료한다.

GCT 수행 절차에서 9)번 단계는 목표 분기 실행 이전에 실행해야 되는 분기들을 자료 흐름 정보를 이용하여 식별한다. 9-a)와 9-b)는 목표 분기에 현재 미치는 영향이 목표 분기 실행에 도움이 되지 않았기 때문에 목표 분기 실행에 영향을 미치는 다른 경로를 탐색하는 단계이다.

최근에 콘콜릭 테스트를 이용하여 리그레션 테스트를 수행한 연구 결과가 Rothermel 등에 의해 발표되었다[6]. 이 방법은 이 논문에서 제안된 방법과 유사하게 특정 분기(집합)을 수행하는 테스트 데이터를 생성한다. 그러나 다음과 같은 점에서 이 논문에서 제안한 방법과 차이가 있다. 이 논문은 개발 과정에서 사용되는 테스트 방법인데 반해 [6]의 방법은 수정된 프로그램에 대한 리그레션 테스트라는 점이다. 즉, 리그레션 테스트는 이미 한 번 테스트가 되어 기존의 테스트 데이터 suite가 있다는 점을 가정하고 있지만 이 논문의 방법은 그러한 가정이 없다. 이는 만약 기존의 테스트 suite가 존재하지 않는 경우에는 [6]의 방법을 적용할 수 없다는 사실을 의미한다. 기술적으로도 목적 분기를 찾기 위해 Rothermel의 연구는 기존 테스트 데이터가 수행한 트레이스 정보를 이용하지만 이 논문은 자료 흐름 정보를 이용하여 탐색한다.

3.3 예제

그림 1의 foo 함수에서 분기 (8, 9)를 실행하는 테스트 데이터를 GCT를 사용하여 찾는 과정을 살펴보자. 이 함수는 두 개의 정수형 배열 a와 b를 입력으로 받는다. 이 분기를 실행하기 위해서는 배열 a의 원소 중의 하나가 10 이어야 하지만 b 값은 상관이 없다. 첫 번째 단계로 두 배열의 크기가 10인 경우에 a[0], a[1], ..., a[9], b[0], ..., b[9]에 무작위 값을 할당하여 프로그램을 실행한다(단계 1).

만약 모두 0으로 할당되어 프로그램이 실행되었다면 첫 번째 실행에서 목표 분기가 실행되지 않았으며 분기 (8, 14)가 실행된다(단계 3). paired((8, 14))는 목표 분기 (8, 9)와 같아 이 경우에는 유도 분기로도 선정된다. 만약 $a[i](0 \leq i \leq 9)$와 $b[i](0 \leq i \leq 9)$가 각각 심볼릭 값 α_i와 β_i 값이 할당되었다면 다음과 같은 경로 제약식 E가 생성된다(단계 6): $E=(\alpha_0 \neq 10) \wedge (\alpha_1 \neq 10) \wedge ... \wedge (\alpha_9 \neq 10)$. 여기에서 주목할 사실은 변수 fa에 대한 심볼릭 표현식, i.e., $not(fa \neq 1)$이 E에 반영이 되지 않는다는 점이다. 그 이유는 fa가 입력 변수에 관한 표현식으로 표현되지 않기 때문이다. 따라서 E를 만족한 해를 구해 입력으로 프로그램을 실행하여도 처음 경로와 동일한 경로가 실

행된다(단계 9).

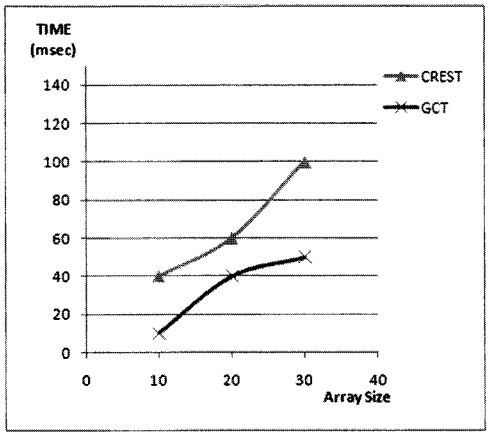
이 단계에서 실행 경로 상에서 분기 (8, 14) 이전에 실행되었던 분기 l=('a[9]≠10', 'i=i+1')에 대해 테스트 문장 (fa==1)에 영향을 미치는 문장을 조사한다. 이러한 문장은 <l>에는 존재하지 않고 <paired(l)>, i.e., 'fa=1'에 존재하기 때문에 9-b) 단계가 수행된다. 따라서 5)번, 6)번 단계로부터 다음과 같은 새로운 경로 제약식이 생성된다: $E=(a_0\neq 10)\wedge(a_1\neq 10)\wedge\ldots\wedge\text{not}\ (a_9\neq 10)$. 이 경로 제약식을 만족하기 위해서는 a[9]가 10이어야 한다. 따라서 새로운 E를 만족하는 입력 값은 목표 분기 (8, 9)를 실행할 수 있다.

4. 실험 결과

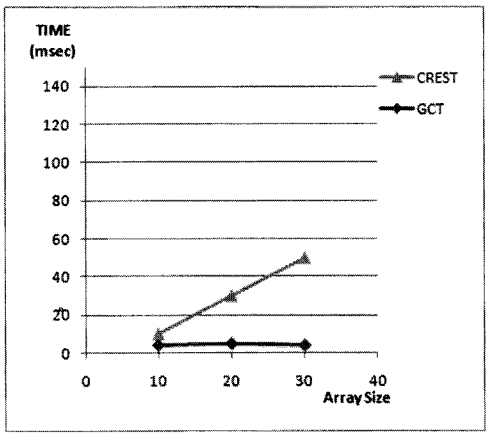
이 절에서는 이 논문에서 제안한 목적 지향 콘콜릭 테스트 데이터 생성 방법의 효용성을 보이기 위해 그림 1에 주어진 함수 foo에 대해 CREST와 비교하여 실험을 수행한 결과를 기술한다. 실험환경은 Linux Ubuntu 9.04를 탑재한 Pentium 4 (4GB 메모리, 2.5GHz) PC에서 수행하였다.

마지막 조건문에서 fb가 1, 즉, 참이 되는 경우가 목표 분기이며 이를 실행하기 위해서는 입력 배열 a의 원소 중의 하나가 10을 가지고 b 배열의 모든 원소들이 10이 되는 경우이다. 변수 fa나 fb와 같이 참이나 거짓을 가지는 부울리언 변수로 프로그램의 분기 조건에 사용되는 변수를 플래그 변수라 한다. 플래그 변수를 사용하는 프로그램에서 기존의 많은 테스트 데이터 탐색 알고리즘의 성능은 랜덤 테스트를 수행하는 경우와 동일한 것으로 밝혀졌다[7].

그림 2(a)는 그림 1의 프로그램에 대해 CREST 방식과 이 논문에서 제안하는 GCT 방식을 배열 크기에 따라 목표 분기를 실행하는 테스트 데이터를 생성하는데까지 걸리는 시간을 비교한 그래프이다. [3]에서 소개한 CREST의 7가지 탐색 알고리즘 중에서 DFS(Depth-First Search)를 제외한 나머지 탐색 알고리즘들은 주어진 목표 분기를 실행할 수 있는 테스트 데이터를 생성하지 못하였다. 따라서 이 논문에서는 CREST의 탐색 알고리즘으로 DFS를 가정한다. 실험에 사용된 예제 프로그램의 배열의 크기는 a, b 두 배열 모두 10, 20, 30까지로 한정하였다. 그래프에서 볼 수 있듯이 GCT가 목표 분기를 실행하는 테스트 데이터를 찾는 시간에서 대략적으로 CREST의 절반정도의 시간만이 소요된 사실을 알 수 있다. 이러한 성능 차이는 목표 분기에 도달하기 전에 탐색해야 하는 경로가 많은 경우에 확연하게 드러난다. 이를 확인하기 위해 그림 1의 프로그램을 입력 배열 a와 b 모두 원소 중의 하나가 10을 가지는 경



(a)



(b)

그림 2 (a) CREST와 GCT의 성능 비교 (b) 수정된 프로그램에 대한 성능 비교

```
void foo(int a[], int b[], int n) {
1:      i = 0;
2:      fa = 0;
3:      fb = 0;

4:      while(i < n){
5:              if(a[i] == 10)
6:                      fa = 1;
7:              i = i + 1;
        }

8:      if(fa == 1){
9:              fb = 1;
10:             i = 0;
11:             while(i < n){
                        if(b[i] != 10)
12:                             fb = 0;
13:                     i = i + 1;
                }
        }

14:     if(fb == 1)
15:             fprintf(stderr,"GOAL!\n"); /* 목표 분기 */
}
```

그림 1 예제 프로그램

우에 목표 분기를 수행할 수 있도록 프로그램을 변경하여 보자. 그림 2(b)는 이와 같이 프로그램을 변경하였을 때의 실험 결과를 보여준다.

이와 같은 결과는 CREST는 목표 분기에 도달하기 전에 탐색해야하는 경로들을 모두 탐색한 후에 목표 분기에 도달한다. 즉, "조건 if (a[i]==10) ..."에 의해 배열 크기만큼 실행 분기가 생성되며 (i.e., 'if (a[0]==10)...', 'if (a[1]==10) ...',...,'if (a[9]==10)...') CREST는 이 분기들을 모두 탐색하기 때문이다. 반면에 GCT는 생성된 실행 분기 중에서 하나만을 실행하는 테스트 데이터를 탐색하면 충분하다. 특히 이 경우에는 GCT의 성능은 배열의 크기와는 크게 의존적이지 않음을 알 수 있다.

5. 결론 및 향후 연구

이 논문에서는 콘콜릭 테스트 방법을 기반으로 특정 분기를 실행할 수 있는 테스트 데이터를 생성하는 목적 지향 콘콜릭 테스트 방법에 대해 기술하였다. 기본적으로 콘콜릭 테스팅은 프로그램을 테스트 할 때 모든 실행 경로들을 심볼릭 수행기법을 사용하여 테스트 하는 것은 많은 비용을 초래할 수 있다. 랜덤 테스트와 같은 저비용의 테스트 방법을 사용하여 우선 테스트한 후에 여전히 실행이 안된 프로그램이나 블록들만을 대상으로 테스트 데이터를 생성하는 방식이 보다 효과적일 수 있다. 목적 지향 콘콜릭 테스트 방법은 이 때 적용할 수 있는 효과적인 방법으로 간주된다. 실험 결과에서 목적 지향 콘콜릭 테스트 방법은 목표 분기에 앞서 탐색해야 하는 공간이 매우 큰 경우에는 콘콜릭 테스트에 비해 매우 효과적임을 보였다. 향후 연구에서는 이 방법을 포인터를 사용하는 프로그램과 통합 테스팅을 지원할 수 있도록 한 개 이상의 프로시듀어에 걸친 프로그램을 처리할 수 있도록 확장할 계획이다.

참 고 문 헌

[1] J. Edvardsson, "A Survey on Automatic Test Data Generation," In *Proc. the Second Conf. on Computer Science and Engineering*, pp.21-28, 1999.

[2] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed automated random testing," In *Proc. of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation* (PLDI), 2005.

[3] J. Burnim, K. Sen, "Heuristics for Dynamic Test Generation," In *the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2008.

[4] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer, "CIL: Intermediate Language and Tools for Analysis and transformation of C Programs," In *Proc. of Conference on compiler Construction*, pp.213-228, 2002.

[5] J. Ferrante, K. Ottenstein, and J. Warren, "The Program Dependence Graph and its Use in Optimization," *ACM Trans. Softw. Eng., Methodology*, vol.2, no.9, pp.319-349, 1987.

[6] Z. Xu and G. Rothermel, "Directed Test Suite Augmentation," In *Proc. of 16th APSEC*, pp. 406-413, 2009.

[7] L. Bottaci, "Instrumenting Programs with Flag Variables for Test Data Search by Genetic Algorithm," In *Proc. of the Genetic and Evolutionary Computation Conf.(GECCO'02)*, pp.1337-1342, NY, USA, July 2002.

정 인 상

1987년 서울대학교 컴퓨터공학과(학사) 1989년 한국과학기술원(KAIST) 전산학과(석사). 1993년 한국과학기술원(KAIST) 전산학과(박사). 1994년~1999년 한림대학교교수. 1999년 9월~현재 한성대학교 컴퓨터공학과교수. 관심분야는 소프트웨어공학 소프트웨어테스팅

박 정 규

2009년 한성대학교 컴퓨터공학과 졸업 2009년~현재 한성대학교 컴퓨터공학과 석사과정. 관심분야는 소프트웨어공학 소프트웨어테스팅