동적 심볼릭 수행과 유전 알고리즘을 사용한 테스트 생성 기법 비교

김윤호, 김문주

KAIST
대전광역시 유성구 구성동 373-1
kimyunho@kaist.ac.kr, moonzoo@cs.kaist.ac.kr

요약: 현대 사회에서 소프트웨어가 널리 사용됨에 따라 소프트웨어의 오류에 대한 사회 비용 문제와 소프트웨어 테스트의 중요성이 점차 커지고 있다. 하지만 현재의 테스트 관행은 테스트 케이스를 작성하는데 많은 인력과 비용이 들고, 효과적인 테스트 케이스를 작성하기 어려운 문제가 있다. 따라서 효과적인 테스트 케이스를 생성하기 위한 기법이 많이 제안되었다.

본 논문에서는 테스트 케이스 자동 생성 기법 가운데 가장 각광받고 있는 동적 심볼릭 수행과 유전 알고리즘 기법을 비교 분석하였다. 각 알고리즘의 개요를 설명하고 적용 가능성, 테스트 케이스 생성 효과, 테스트 케이스 생성 성능을 비교 분석하였다.

핵심어: 심볼릭 수행, 유전 알고리즘, 소프트웨어 테스트

1. 서론

현대 사회에서 소프트웨어가 널리 사용됨에 따라 소프트웨어의 오류에 따른 사회적 비용의 증가 문제가 크게 대두되고 있다. 2002 년 NIST 의 조사에 따르면[1] 소프트웨어 오류로 인한 사회적 손실 비용이 당시 약 600 억 달러에 달했으며, 이 중 약 1/3 이 기본적인 소프트웨어 테스트 과정을 통해 방지할 수 있는 오류인 것으로 밝혀졌다. 따라서 소프트웨어의 오류를 검출하기 위한 소프트웨어 테스트의 중요성이 점점 커지고 있다.

소프트웨어 테스트의 성패는 얼마나 효과적인 테스트 케이스를 작성하느냐에 달려있다. 하지만 현재 널리 사용되고 있는 테스트 관행은 테스트 케이스를 개발자가 임의로 작성하거나, 특정 사용 시나리오를 기반으로 작성하고 있기 때문에 소프트웨어의 복잡한 동작을 체계적으로 테스트하기 어렵고 충분한 테스트 케이스를 작성하는 데 많은 인력과 비용이 필요하다.

이와 같은 문제점을 해결하기 위해 소프트웨어 테스트 케이스를 자동으로 생성해 주는 기법들이 많이 제안되었다. 그 중에서 본 논문에서는 동적 심볼릭 수행을 활용한 테스트 케이스 생성 기법과 유전 알고리즘을 활용한 테스트 케이스 생성 기법을 소개하고 이 둘을 비교하도록 한다.

동적 심볼릭 수행(dynamic symbolic execution, 혹은 concolic testing)[2-8] 은 테스트 케이스를 생성하는 문제를 제약 만족 문제의 형태로 표현하고 풀어낸다. 동적 심볼릭 수행은 먼저 테스트 대상 프로그램을 실행하고 심볼릭 수행을 통해 현재 실행한 경로의 경로 제약 조건을 생성한다. 이 현재 실행한 경로 제약 조건을 기반으로 아직까지 실행하지 않은 새로운 경로를 표현하는 경로 제약 조건을 만들고 풀어냄으로써 새로운 실행 경로를 수행하는 테스트 케이스를 생성한다.

유전 알고리즘(genetic algorithm)[9-13] 테스트 케이스를 생성하는 문제를 최적해를 찾는 문제로 간주하고 풀어낸다. 유전 알고리즘은 먼저 주어진 테스트 평가 기준(test criteria)을 기반으로 테스트 케이스에 대한 적합도 함수(fitness function)를 생성한다. 이 적합도 함수의 최적해에 해당하는 테스트 케이스를 찾기 위해 유전자가 진화하는 것과 같이 테스트 케이스를 선택, 교차, 변이 시킴으로써 최적해에 해당하는 테스트 케이스로 진화시켜 나간다.

동적 심볼릭 수행과 유전 알고리즘을 사용한 테스트 생성 기법은 서로 상이한 방법을 통해 테스트 케이스를 생성하기 때문에 그 장, 단점이 서로 다르다. 하지만, 그 동안 동적 심볼릭 수행 기법과 유전 알고리즘 기법을 테스트 케이스 생성 관점에서 비교한 연구는 많이 이루어지지 않아서 어떤 프로그램에 어떤 기법을 사용하는 것이 좋을지 알기 어려웠다. 따라서 각 기법의 특성을 자세하게 분석하고 비교하는 연구가 이 기법을 사용하고자 하는 개발자와 기법을 발전시키고자 하는 연구자

모두에게 있어 도움이 되리라 생각한다.

2. 기본 개념

2.1 동적 심볼릭 수행

동적 심볼릭 수행은 기존의 정적 심볼릭 수행[14]의 한계를 극복하기 위해 동적 분석을 결합한 테스트 생성 기법이다. 동적 심볼릭 수행은 실제 프로그램을 실행하면서 프로그램 경로를 따라 경로 제약 조건을 생성한다.

동적 심볼릭 수행은 크게 여섯 단계로 나눌 수 있다.

1. 심볼릭 변수 선언
   이 단계에서는 타겟 프로그램의 어떤 입력 변수를 심볼릭 변수로 사용할 것인지를 정의한다. 심볼릭 수행 경로는 이 단계에서 선언한 심볼릭 변수로 표현된다.

2. 타겟 프로그램 수정(instrumentation)
   타겟 프로그램을 수정하여 프로그램이 실행될 때 어떤 경로를 수행하는지를 기록하는 probe 를 삽입한다. 프로그램이 종료되었을 때, probe 가 기록한 실행 경로를 기반으로 현재 실행한 경로의 경로 제약 조건을 생성한다.

3. 타겟 프로그램 실행
   Probe 를 삽입한 수정된 타겟 프로그램을 실행한다. 처음에는 임의로 생성된 입력값을 사용하고 그 후에는 동적 심볼릭 수행을 통해 생성된 입력값을 사용해서 타겟 프로그램을 수행한다.

4. 경로 제약 조건 생성
   타겟 프로그램이 실행되는 동안 실행 경로를 따라 동적 심볼릭 수행을 한다. 만약 대입문이 수행되면 1 단계에서 선언한 심볼릭 변수가 갱신되어야 하는지 확인하고 필요한 경우에 심볼릭 변수값을 갱신하며, 분기문이 수행되면 심볼릭 경로 조건 c_i 를 기록한다. 프로그램의 수행이 모두 종료되면 분기문이 수행될 때 기록한 심볼릭 경로 조건을 실행 순서대로 논리곱으로 연결하여 경로 제약 조건 $\phi = c_1 \wedge c_2 \ldots \wedge c_n$ 을 생성한다.

5. 새로운 경로 제약 조건 생성
   4 단계에서 생성한 경로 제약 조건 ϕ 를 사용해 새로운 입력값을 생성하기 위한 새로운 경로 제약 조건 ψ 를 생성한다. 새로운 경로 제약 조건 ψ 는 ϕ 에서 하나의 심볼릭 경로 조건을 부정하고 부정한 심볼릭 경로 조건 이후의 경로 조건들을 제거해서 생성한다. 즉, ψ 는 어떤 j 에 대해서 $c_1 \wedge c_2, \ldots, \wedge c_j$ 로 표현된다. 어떤 경로 조건을 부정할 것인지는 심볼릭 경로 탐색 전략에 따라 달라진다.

6. 새로운 입력값 생성
   5 단계에서 생성한 새로운 경로 제약 조건 ψ 를 SMT solver[15] 와 같은 제약 조건 해찾기 도구를 사용해서 풀어내고 새로운 입력값을 생성한다. 만약 ψ 를 만족하는 해가 없으면 5 단계로 돌아가 새로운 경로 제약 조건을 다시 생성하고, ψ 를 만족하는 해가 있는 경우, 이 해를 사용해서 새로운 테스트 케이스를 생성한 다음 3 단계로 돌아간다.

동적 심볼릭 수행을 통한 테스트 케이스 생성 과정을 세 값의 최대값을 찾는 예제 프로그램을 사용해 좀 더 구체적으로 설명하면 다음과 같다.

```
int main() {
 int x, y,z, max_num=0;
 SYM_INT(x); // Declaration of x, y, z
 SYM_INT(y); // as symbolic integer
 SYM_INT(z); // variables
   if(x >= y) { // SYM_COND(x,y, ">=");
     if(y >= z) { // SYM_COND(y,z, ">=");
       max_num = x;
     } else { // SYM_COND(y,z, "<");
     if (x >= z){ // SYM_COND(x,z, ">=");
       max_num = x;
     } else { // SYM_COND(x,z, "<");
       max_num = z;
     }
   }
 } else {...}
 printf("%d is the largest number among\
 {%d,%d,%d}", max_num, x,y,z);
 // SMT_Solve();
}
```

1. 심볼릭 변수 선언

사용자가 SYM_INT() 함수를 통해서 변수 x, y, z 를 심볼릭 변수로 선언한다.

2. 타겟 프로그램 수정
동적 심볼릭 수행 도구가 각 분기문의 then 분기와 else 분기에 SYM_COND() probe 를 삽입하여 실행 시간에 어떤 분기를 실행하고 어떤 심볼릭 변수를 사용했는지 기록한다.

3. 타겟 프로그램 실행
초기 입력값으로 임의의 값을 주고 타겟 프로그램을 실행한다. 여기서는 x, y, z 가 각각 1, 1, 0 의 값을 초기값으로 갖는다고 가정한다.

4. 경로 제약 조건 생성
주어진 입력값으로 프로그램을 실행시킬 경우 $x \geq y$ 이고 $y \geq z$ 이기 때문에 $\phi = x \geq y \wedge y \geq z$ 인 경로 제약 조건이 생성된다.

5. 새로운 경로 제약 조건 생성
두 경로 조건 중 하나를 부정하여 새로운 경로 제약 조건 $\psi = x \geq y \wedge y < z$ 를 생성한다.

6. 새로운 입력값 생성
5 단계에서 생성한 ψ 를 풀어서 새로운 입력값 1, 1, 2 를 생성한다. 이 입력값을 갖고 3 단계로 돌아가 3~6 단계를 더 이상 새로운 실행경로를 수행하지 못할 때까지 반복한다.

2.2 유전 알고리즘

2.2.1 유전 알고리즘 과정

유전 알고리즘은 유전자의 진화 원리를 기반으로 하여 구한 해를 적합도 함수를 기준으로 최적의 해로 진화시켜나가는 알고리즘이다. 유전 알고리즘을 활용한 테스트 생성 기법은 테스트 케이스를 유전 알고리즘을 통해 생성해야 할 해로 설정하고, 임의로 주어진 테스트 케이스를 진화시켜 주어진 테스트 평가 기준에 적합한 테스트 케이스를 생성하는 것을 목표로 한다.

유전 알고리즘을 사용한 테스트 생성 기법은 다음 다섯 단계로 나뉜다. 각 단계에서 사용하는 적합도 평가, 테스트 케이스 선택, 교배, 변이에 대해서는 2.2.2 절에서 자세하게 설명한다.

1. 초기 테스트 케이스 집단 풀 생성
주어진 테스트 케이스를 사용하거나 혹은 임의로 생성된 테스트 케이스를 사용해서 초기 테스트 케이스 집단 풀을 생성한다.

2. 테스트 케이스 집단 풀의 각 테스트 케이스의 적합도 평가
테스트 케이스 집단 풀의 각각의 테스트 케이스의 적합도를 평가한다.

3. 적합도를 기반으로 테스트 케이스 선택
2 단계에서 평가한 적합도를 사용해서 테스트 케이스를 선택하고 선택 받지 못한 테스트 케이스를 집단 풀에서 제거한다.

4. 선택된 테스트 케이스 교배를 통한 새로운 테스트 케이스 집단 풀 생성
3 단계에서 선택된 테스트 케이스를 교배하여 새로운 테스트 케이스 집단 풀을 생성한다.

5. 테스트 케이스 집단 풀의 일정 개체를 돌연변이 시킴
테스트 케이스 집단 풀에 돌연변이를 추가하여 집단 풀에 다양성과 추가한다. 돌연변이를 통해 테스트 케이스 탐색 공간을 넓힐 수 있다.

2.2.2 유전 알고리즘 매개변수

- 적합도 평가 방법

테스트 케이스의 적합도 평가 함수는 접근 레벨(approach level)과 분기 거리(branch distance) 를 조합하여 생성한다[13]. 접근 레벨은 CFG(Control-Flow Grpah)에서 실행하고자 하는 목표 노드와 현재 테스트 케이스가 실행한 노드들 사이의 가장 가까운 거리로 정의된다. 분기 거리는 목표 노드와 접근 레벨이 가장 낮은 분기문의 조건식과 입력값을 사용해서 계산한다. 예를 들어 다음 예제 코드에 대해 TC1(a = 10, b = 15, c = 30)과 TC2(a = 5, b = 7, c=20)가 있다고 하면

```
if (a==10)
  if (b==15)
    if(c==20)
      /* target */
```

TC1 의 접근 레벨은 1, 분기 거리는 |30-20| 이 되고 TC2 의 접근 레벨은 3, 분기 거리는 |10-5| 가 된다.

- 선택 방법

선택 방법에는 크게 룰렛 선택, 토너먼트 선택, 순위기반 선택, 임의 선택이 있다. 룰렛 선택 방법에서는 테스트 케이스의 적합도 합의 크기를 갖는 룰렛을 만들고, 각 테스트 케이스는 자신의 적합도에 비례하여 룰렛의 공간을 할당 받는다. 원하는 개수만큼의 테스트 케이스를 선택할 때까지 룰렛에 화살을 쏘아서 화살이 맞는 테스트 케이스를 선택한다. 토너먼트 선택은 테스트 케이스를 임의로 배치하여 토너먼트 대진표를 만들고 적합도가 높은 테스트 케이스를 승자로 하는 토너먼트 게임을 진행하여 상위 순위의 테스트 케이스를 선택하는 방법이다. 순위 기반 선택은 적합도를 기준으로 테스트 케이스를 정렬하여 상위 테스트 케이스를 선택하고, 임의 선택은 적합도와 상관 없이 임의로 테스트 케이스를 선택하는 방법이다.

- 교배 방법

교배 방법에는 한 점 교배(one-point crossover), 여러 점 교배(multi-point crossover), 균등 교배(uniform crossover) 방법이 있다. 한 점 교배는 테스트 케이스의 임의의 한 점을 기준으로 두 테스트 케이스를 교환하는 방법이다. 예를 들어 두 테스트 케이스 TC1(5, 10, 15)과 TC2(5, 20, 30) 이 있고, 기준점을 첫 번째와 두 번째 값 사이로 잡았을 때 교배 후 생성되는 테스트 케이스는 TC1′(5, 20, 30) 과 TC2′(5, 10, 15) 이다. 여러 점 교배는 한 점 교배와 비슷하나 기준점이 한 점이 아닌 여러 점을 선택하여 교환하는 방법이다. 균등 교배는 테스트 케이스의 각 항목에 대해서 부모 중 어느 하나의 항목을 임의로 가져오는 교배 방법을 말한다.

- 돌연변이 방법

돌연변이는 낮은 확률로 테스트 케이스의 일부분을 임의로 수정하여 테스트 케이스를 변경시키고 테스트 케이스 집단의 다양성을 넓힌다. 테스트 케이스를 수정하는 확률에 따라 균등 변이와 비균등 변이로 나눌 수 있다. 균등 변이는 테스트 케이스를 변경할 확률을 일정하게 유지하는 돌연변이 방법이다. 비균등 변이는 유전 알고리즘의 후반부로 갈수록 변이 확률을 줄이는 돌연변이 방법으로 유전 알고리즘의 후반부로 갈수록 테스트 케이스 집단의 적합도 수준이 높아 돌연변이로 인한 적합도 향상을 기대하기 어렵다는 가정에 기반한다.

3. 비교 기준

본 장에서는 동적 심볼릭 수행을 활용한 테스트 생성 기법과 유전 알고리즘을 활용한 테스트 생성 기법의 비교 기준을 설명한다.

3.1 기법 적용 가능성(applicability)

두 기법을 실제 프로그램에 적용하여 테스트 케이스를 생성하기 위해서는 타겟 프로그램의 입력 변수를 설정하고 매개변수를 조율(parameter tuning)해야 한다.

특정 실행 경로를 수행하는 테스트 케이스를 생성하는 문제는 비결정적(undecidable)인 문제기 때문에 문제를 정확하게 해결하는 알고리즘이 존재하지 않는다. 동적 심볼릭 수행을 활용한 테스트 생성 기법과 유전 알고리즘을 활용한 테스트 생성 기법 테스트 생성 기법 모두 휴리스틱을 활용한 근사해를 구해 나가는 과정으로 볼 수 있다. 다른 근사 알고리즘과 마찬가지로 테스트 생성 기법 역시 입력값인 프로그램과 조율 가능한 매개변수의 값에 따라서 휴리스틱의 성능 및 효과가 크게 달라지기 때문에 매개변수의 조율은 중요한 비교 기준이 된다.

동적 심볼릭 수행의 매개변수로는 크게 탐색 전략과 경로 조건 제약식을 표현하는 기반 논리가 있고, 유전 알고리즘의 매개변수로는 테스트 케이스 집단 풀의 크기, 적합도 함수 생성 방법, 개체 선택, 교배, 돌연변이 방법 등이 있다.

3.2 효과(Effectiveness) 비교

동적 심볼릭 수행 기법과 유전 알고리즘을 활용한 테스트 케이스 생성 기법 모두 테스트 케이스를 생성하기 어려운 구문과 쉬운 구문을 갖고 있다. 동적 심볼릭 수행 기법의 정확한 심볼릭 경로 조건식을 만들기 어려운 프로그램 구문이나, 정확한 심볼릭 경로 조건식을 만들더라도 SMT solver 로 푸는데 시간이 오래 걸리거나 정확하게 풀 수 없는 구문이 이에 해당한다. 유전 알고리즘의 경우, 접근 레벨이 커지면 커질수록 테스트 케이스를 생성하기 어렵다.

3.3 성능(Efficiency) 비교

동적 심볼릭 수행 기법의 성능은 심볼릭 실행 경로를 기록하기 위해 probe 가 차지하는 시간과 경로 제약 조건식을 풀기 위해 SMT solver 가 차지하는 시간에 크게 좌우된다. 특히 SMT solver 가

차지하는 시간이 전체 테스트 생성 시간의 50%를 넘어[16] 심볼릭 제약 조건식을 어떻게 빨리 풀어내느냐가 동적 심볼릭 수행 기법이 얼마나 빨리 수행되느냐를 좌우한다고 볼 수 있다.

유전 알고리즘의 경우 사용하는 매개변수에 따라 조금씩 다르지만, 생성된 테스트 케이스를 사용해 타겟 프로그램을 실행하고 적합도를 계산하는 데 많은 시간을 사용한다.

4. 비교 분석

4.1 기법 적용 가능성

4.1.1 입력 변수의 길이와 형태

동적 심볼릭 수행 기법을 적용하기 위해서 먼저 사용자는 어떤 입력이 심볼릭 변수로 선언되어야 할 것인가를 고려해야 한다. 최대의 효과를 보기 위해선 가능한 모든 입력 변수를 심볼릭 변수로 선언하는 것이 좋지만, 심볼릭 변수가 많아질수록 심볼릭 경로 조건식을 만들기 위한 probe 의 실행 시간이 오래 걸리고, 경로 제약 조건식의 길이가 길어져서 경로 제약 조건식을 푸는데 오랜 시간이 걸린다.

```
void test_analyze () {

char str[10]={0,};
for(i=0; i < 9; i++)
        SYM_char[i]; // 심볼릭 입력 선언

analyze(str);
}
```

예를 들어 위의 test_analyze() 함수를 보면 최대 9 의 길이를 갖는 문자열 str 을 심볼릭 입력으로 선언하여 테스트 타겟 프로그램인 analyze 함수를 수행하였다. 만약 analyze() 함수가 길이 10 이상인 문자열을 분석하는 데 있어 오류가 있다면, test_analyze() 로는 이와 같은 오류를 분석할 수 없다. 하지만 입력 문자열의 길이를 10 이상으로 늘린다면, 그에 따라 프로그램의 실행 가능한 경로가 많아지고, 동적 심볼릭 수행이 수행되는데 더 많은 시간이 걸리게 된다. 따라서 비용 대비 효과를 극대화 하기 위해선 사용자가 타겟 프로그램을 이해하고 최적의 심볼릭 입력을 설정해야 하는 문제가 있다.

반면 유전 알고리즘의 경우 생성되는 입력의 길이나 종류에 실행시간이 큰 영향을 받지 않기 때문에 최대한 많은 입력을 생성하도록 할 수 있다. 따라서 입력 설정을 위해서는 동적 심볼릭 수행이 유전 알고리즘보다 더 많은 사용자 지식을 요구하고 적용 가능성이 떨어진다고 볼 수 있다.

4.1.2 매개변수 조율

적용 가능성을 평가하기 위한 또 다른 관점은 얼마나 많은 매개변수를 조율해야 하는가 이다. 두 알고리즘은 모두 휴리스틱을 기반으로 하기 때문에, 매개변수의 조율에 따라 성능 및 효과가 크게 차이날 수 있다.

동적 심볼릭 수행 기법의 경우 조율할 수 있는 매개변수는 경로 탐색 기법과 경로 제약 조건을 표현하기 위한 논리식 방법이 있다. 경로 탐색 기법으로는 모든 실행 가능한 경로를 탐색하기 위한 DFS 기법, 분기 커버리지 달성을 목표로 하는 하이브리드 기법[17], CFG 기반 휴리스틱 기법[18], 적합도 기반 휴리스틱 기법[19] 등이 있으며 경로 제약 조건을 표현하기 위한 논리식의 종류로는 크게 선형 정수 표현식과 bit-vector 표현식이 있다. 경로 탐색 기법은 테스트 목표에 따라 결정할 수 있고, 그 종류가 한정적이기 때문에 모두 적용해 본 후에 그 결과를 비교 분석할 수 있다. 경로 제약 조건을 표현하기 위한 논리식의 경우, 프로그램이 주로 선형 정수 표현식으로 표현되고 빠른 성능이 필요한 경우 선형 정수 표현식을, bit 수준의 연산자와 정확도가 필요한 경우는 bit-vector 표현식을 사용할 수 있다. 이와 같이 사용하고자 하는 목적에 따라 사용하고자 하는 매개변수가 비교적 명확하기 때문에 사용자가 필요한 경우 쉽게 조율할 수 있다.

유전 알고리즘의 경우 동적 심볼릭 수행 기법에 비해 더 많은 매개변수 조율이 필요하다. 먼저 테스트 생성 과정에서 사용되는 테스트 케이스 집단 풀의 크기를 설정해야 하고 적합도를 어떻게 평가하고, 어떤 선택, 교배, 돌연변이 방법을 사용해서 테스트 케이스를 진화시킬 것인지를 선택해야 한다. 동적 심볼릭 수행과 달리 각 선택, 교배, 돌연변이 기법이 특별한 목적성을 갖고 개발된 것이 아니기 때문에 타겟 프로그램에 어떤 기법이 적당한지는 직접 실행해 보고 결정해야 한다. 또한 각 기법마다 기준이 되는 값(예를 들어, 한 점 교배 기법에서 어떤 기준점을 잡을 것인가, 돌연변이 기법에서 돌연변이 율은 몇으로 설정할 것인가 등)에 따라서도 성능 및 효과가 크게 달라지기 때문에 사용자는 매개변수를 조율하는 데 더 많은 노력을 들여야 한다.

따라서 실제 개발 환경에 얼마나 쉽게 적용

가능한지를 살펴보는 적용 가능성을 따져 봤을 때 동적 심볼릭 기법이 유전 알고리즘보다 더 적용 가능성이 높다고 볼 수 있다.

4.2 효과 비교

동적 심볼릭 수행 기법의 경우 심볼릭 경로 조건식을 생성할 수 없거나, 혹은 심볼릭 경로 조건식을 생성하더라도 SMT solver 로 풀기 어려운 경우 테스트 생성 효과가 떨어질 수 있다. 심볼릭 경로 조건식을 생성할 수 없는 경우는 내부 구현을 알 수 없는 외부 바이너리 함수를 사용해서 probe 가 심볼릭 경로 조건식을 표현하지 못하는 경우와 심볼릭 메모리 모델이 표현하기 어려운 복잡한 포인터 연산 및 메모리 연산을 사용하는 경우, 그리고 부동소수점 연산을 사용하는 경우가 있다. 예를 들어 부동소수점 연산의 경우에는 현재 컴퓨터에서 부동소수점 연산을 하는데 사용하는 IEEE 754[20] 표준을 표현하고 풀 수 있는 SMT solver 가 아직 개발되지 않았기 때문에 부동소수점 연산을 사용한 프로그램의 심볼릭 경로 조건식을 생성할 수 없다.

유전 알고리즘의 경우 테스트 케이스를 진화시키는 과정에서 입력 값의 형태와 수행하는 연산의 종류에 크게 제약을 받지 않기 때문에 동적 심볼릭 수행의 효과가 떨어지는 프로그램을 테스트 하는데 효과적일 수 있다. 하지만 동적 심볼릭 수행과 달리 중첩 분기문을 수행하는 테스트 케이스를 생성하는 데 효과적이지 않다. 예를 들어 다음과 같은 중첩 분기문이 있는 경우를 살펴보자.

```
if (a==10)
  if (b==15)
    if(c==20)
    /* target */
```

동적 심볼릭 수행의 경우 최대 4 번의 경로 제약 조건을 풀고 나면 목표 노드를 수행할 수 있지만 유전 알고리즘을 사용하면 선택, 교배, 돌연변이를 통해 $(2^{-32})^3$ 의 확률을 통과해야 목표 노드를 수행할 수 있다. 특히 유전 알고리즘의 적합도는 기존에 생성된 테스트 케이스를 평가하는데 만 사용되지 새로운 테스트 케이스를 생성하는 데 사용되지 않기 때문에 교배, 변이 과정에서 항상 더 좋은 적합도를 갖는 테스트 케이스를 생성한다는 보장이 없다.

4.3 성능 비교

동적 심볼릭 수행 기법의 성능은 SMT solver 가 얼마나 빠른 시간 내에 경로 제약 조건을 풀어서 테스트 케이스를 생성할 수 있는가에 크게 좌우된다. 따라서 동적 심볼릭 수행의 매개변수 가운데 경로 제약 조건을 표현하기 위한 논리식을 무엇을 사용했느냐에 따라 성능이 크게 좌우된다. 선형 정수 표현식으로 표현된 제약 조건의 해를 찾는 효율적인 알고리즘이 개발되었으며[21] 선형 정수 표현식으로 기술할 수 없는 경로 제약 조건은 생성되지 않기 때문에 bit-vector 를 사용해 표현한 경로 제약 조건보다 제약 조건의 길이가 더 짧다. 따라서 일반적으로 동일한 실행 경로에 대해 경로 제약 조건을 생성하고 SMT solver 를 사용해서 해를 구할 때, 선형 정수 표현식으로 경로 제약 조건을 기술하는 것이 bit-vector 를 사용해 경로 제약 조건을 기술하는 것 보다 효과는 떨어지지만 성능을 향상시킬 수 있다.

유전 알고리즘의 경우 적합도를 계산하기 위해 테스트 케이스에 대해 타겟 프로그램을 실행하는 시간이 테스트 케이스를 생성하는데 드는 시간의 대부분을 차지한다. 따라서 테스트 케이스 집단 풀의 크기를 몇으로 설정한 것 인지와 얼마나 많은 횟수만큼 진화 과정을 반복할 것인지가 성능에 큰 영향을 미치게 된다. 따라서 짧은 진화과정을 통해 좋은 테스트 케이스를 만들 수 있도록 선택, 교배, 변이와 같은 매개변수의 설정을 잘 해줘서 진화 과정의 횟수를 줄이고 테스트 케이스의 집단 풀을 작게 유지해서 각 진화 단계의 시간을 줄여주는 것이 유전 알고리즘을 사용한 테스트 케이스 생성 성능을 향상시키는 데 도움이 된다.

4.4 비교 분석 결론

4.1 절에서 4.3 절까지, 동적 심볼릭 수행과 유전 알고리즘을 테스트 생성 기법 측면에서 비교한 결과를 정리하면 다음 표 1 과 같다.

	동적 심볼릭 수행	유전 알고리즘
기법 적용 가능성	● 입력 변수의 길이와 형태를 면밀히 지정해야 함 ● 매개변수 조율이 크게 중요치 않음	● 입력 변수의 길이와 형태에 큰 영향을 받지 않음 ● 많은 매개변수 조율 필요

효과	●타겟 프로그램이 포인터 연산, 부동소수점 연산, 외부 바이너리 함수 호출을 사용하는 경우 효과가 떨어짐	● 타겟 프로그램이 다중 중첩 제어구조를 사용하는 경우 효과가 떨어짐
성능	● 심볼릭 경로 조건 표현식을 풀어서 입력 생성하는 데 많은 시간이 듬	● 진화 및 변이를 위해 테스트 케이스를 계속 생성하고 실행하는 데 많은 시간이 듬
장점	● 테스트 케이스 생성 성능이 좋고 사용하기가 쉬움	● 프로그램 구문에 따른 효과 저하가 동적 심볼릭 수행보다 적음
단점	● 복잡한 포인터 연산, 부동소수점 연산, 외부 바이너리 함수 호출을 사용하는 프로그램을 효과적으로 테스트 하지 못함	● 테스트 케이스 생성 속도가 느리고 사용하는데 어려움

표 1 동적 심볼릭 수행과 유전 알고리즘 비교

5. 결론

본 논문에서는 테스트 케이스 자동 생성 기법 중에서 널리 사용되고 있는 동적 심볼릭 수행과 유전 알고리즘을 소개하고 장, 단점을 비교 분석하였다. 비교 분석 결과 매개변수 조율을 통한 성능 및 효과의 조절이 쉬운 동적 심볼릭 수행이 유전 알고리즘에 비해 적용 가능성이 더 높았으며 효과 비교에서는 중첩 분기 구조가 복잡한 프로그램의 경우 동적 심볼릭 수행이, 외부 라이브러리 함수를 많이 활용하고 복잡한 포인터 연산 및 부동소수점 연산이 많은 경우에는 유전 알고리즘이 더 적합하였다. 성능 비교에서는 각 기법의 매개변수를 어떻게 설정하느냐에 따라 성능 차이가 크게 날 수 있으며 동적 심볼릭 수행 기법의 경우 SMT solver 의 성능 향상이, 유전 알고리즘의 경우는 진화 반복 횟수와 테스트 케이스 집단 풀의 크기를 줄이는 것이 전체적인 테스트 케이스 생성 성능을 높이는데 크게 도움이 되는 것을 알 수 있었다.

본 논문에서 살펴봤듯이 동적 심볼릭 수행과 유전자 알고리즘은 테스트 케이스를 생성하는 데 있어 서로 상이한 특징을 갖고 있으며 테스트 케이스 생성 효과를 비교했을 때 서로 상호 보완할 수 있는 가능성을 보이고 있다. 따라서 차후 연구를 통해, 동적 심볼릭 수행이 약한 부분은 유전 알고리즘을, 유전 알고리즘이 약한 부분은 동적 심볼릭 수행을 활용하는 hybrid 테스트 생성 기법을 개발하여 효과 좋은 테스트 케이스 생성 기법을 개발할 수 있을 것이다.

참고문헌

[1] National Institute of Standards and Technology, “The Economic Impacts of Inadequate Infrastructure for Software Testing,” Planning Report 02-3 May 2002.

[2] P. Godefroid, N. Klarlund, and K. Sen, “DART: Directed Automated Random Testing,” ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, 2005

[3] K. Sen, D. Marinov, and G. Agha, “CUTE: A Concolic Unit Testing Engine for C,” International Symposium on the Foundations of Software Engineering, 2005

[4] CREST Project Page http://code.google.com/p/crest

[5] P. Godefroid, M. Y. Levin, and D. Molnar, “Automated Whitebox Fuzz Testing”, Annual network and Distributed System Security Symposium, 2008

[6] N. Tillmann, and J. Halleux, “Pex-White Box Test Generation for .NET,” International Conference on Tests and Proofs, 2008

[7] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler, “EXE: Automatically Generating Inputs of Death”, ACM Conference on Computer and Communications Security, 2006

[8] C. Cadar, D. Dunbar, and D. Engler, “Klee: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs”, USENIX Symposium on Operating System Design and Implementation, 2008

[9] A. Baresel, H. Sthamer, and M. Schmidt,

"Fitness Function Design to Improve Evolutionary Structural Testing," Genetic and Evolutionary Computation Conference, 2006

[10] L. Bottaci, "Instrumenting Programs with Flag Variables for Test Data Search by Genetic Algorithm," Genetic and Evolutionary Computation Conference, 2002

[11] N. Mansour and M. Salame, "Data Generation for Path Testing," Software Quality Journal, Vol. 12, No. 2, pp. 121-134, 2004

[12] G. McGraw, C. Michael, and M. Schatz, "Generating Software Test Data by Evolution," IEEE Transactions on Software Engineering, Vol. 27, No. 12, pp. 1085-1110, 2001

[13] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary Test Environment for Automatic Structural Testing," Information and Software Technology, Vol. 43, No. 14, pp. 841-854, 2001

[14] J. C. King "Symbolic Execution and Program Testing," Journal of the ACM, Vol.19, No. 7, pp. 385-394, 1976

[15] SMT-LIB: The Satisfiability Modulo Theories Library, http://combination.cs.uiowa.edu/smtlib/.

[16] M. Kim, Y. Kim, and Y. Choi, "Concolic Testing of the Multi-sector Read Operation for Flash Storage Platform Software," Formal Aspects of Computing, in print

[17] R. Majumdar and K. Sen, "Hybrid Concolic Testing," International Conference on Software Engineering, 2007

[18] J. Burnim and K. Send, "Heuristics for Dynamic Test Generation," International Conference on Automated Software Engineering, 2008

[19] T. Xie, N. Tillmann, P. Halleux, and W. Schulte, "Fitness-Guided Path Exploration in Dynamic Symbolic Execution," International Conference on Dependable Systems and Networks, 2009

[20] IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008

[21] B. Dutertre and L. Moura, "A Fast Linear-arithmetic Solver for DPLL(T)," International Conference on Computer Aided Verification, 2006