문장 결합을 통한 소프트웨어 테스트 데이터 생성
(A Generation of Software Test Data Using Statement Combination)

곽동규·조용윤·김영철·유재우
숭실대학교 컴퓨터학과
{coolman, yycho, yckim}@ss.ssu.ac.kr, cwyoo@computing.ssu.ac.kr

요 약

테스트 데이터 생성 단계는 소프트웨어 테스팅 과정에 있어 매우 중요하며 많은 노력을 요구한다[1]. 본 논문에서는 pascal이나 C와 같은 절차적 프로그래밍 언어에 있어서 실행의 단위가 되는 문장들간의 결합구조 정보를 통해 테스트 데이터를 자동 생성하는 방법을 제안한다. 이를 위해 본 논문에서는 프로그램의 문장을 집합으로 정의하고 집합의 원소간에 연산을 통해 프로그램 코드의 구조 정보를 추상 구문 트리(Abstract Syntax Tree)로 표현한다. 또한, 본 논문은 생성된 추상 구문 트리의 구조 정보에 따라 사용자가 입력한 실행 경로를 XML 파일로 표현하여 실행 경로에 영향을 주는 조건식을 추출하고 연산하기 위한 알고리즘을 제안한다. 따라서 제안하는 테스트 데이터 자동 생성 방법은 기존의 테스팅 적합성 기준을 판별하는 과정을 필요로 하지 않는다. 또한, 실행 경로 정보에 대한 XML 파일을 통해 조건식의 추출과 연산과정을 자동화 할 수 있는 가능성을 제시하여 소프트웨어 테스트 데이터 생성에 관련된 자동화 도구 개발에 기여할 수 있을 것으로 기대된다.

1. 서 론

소프트웨어의 복잡도가 증가함에 따라 신뢰성이 높은 소프트웨어 생산이 어려워지고 있다. 신뢰성이 높은 소프트웨어란 개발자의 요구사항에 합당한 동작을 하는 소프트웨어를 의미한다. 높은 신뢰성을 가진 소프트웨어의 생산이 어려워짐에 따라 소프트웨어의 신뢰성을 측정하기 위한 테스트 방법이 요구되고 현재 소프트웨어의 테스트는 전체 소프트웨어 개발의 50%에 해당하는 경비를 지출하고 있다.[1] 소프트웨어를 테스트하기 위한 방법에 관한 연구는 많이 되었고 상용화된 제품[2][3][4]이 많지만 테스트를 위한 테스트 데이터를 자동으로 작성하는 방법에 관한 연구는 아직 미비하다.

Pascal이나 C와 같은 절차적 언어로 작성된 소프트웨어는 함수로 이루어져 있고 각 함수간의 순차적인 실행을 통해 문제를 해결한다. 따라서 신뢰성 높은 소프트웨어를 얻기 위해서는 함수에 대한 적절한 테스트가 필수적이다. 함수는 문장의 제어문을 통한 결합으로 구성되어 있고, 실행 경로에 따라 그 실행 여부가 결정된다. 따라서 소프트웨어의 함수를 테스트하기 위해서는 소프트웨어의 경로를 분석하고 조건 제약식

을 추출하여 연산을 통해 경로에 따른 테스트 데이터를 생성할 수 있는 방법이 요구된다. 일반적으로 함수를 구성하는 문장들은 연속적인(linear)구조와 반복적인(recursive)구조의 두 가지 결합 형태로 이루어져 있다. 또한, 연속적인 구조를 갖는 문장들의 결합 형태는 선행 문장이 실행되고 후행 문장이 실행되는 순차적인(sequential) 결합 형태와 선행 문장과 후행 문장이 조건에 따라 선택적으로 실행되는 선택적인(optional) 결합 형태로 구분되어 진다.

본 논문은 프로그램 소스의 문장을 대수적 집합으로 표현하고, 집합 원소간의 관계를 문장의 결합 특성에 따라 연산 식으로 나타낸다. 생성된 연산 식은 프로그램의 구조 정보를 쉽게 표현하기 위해 추상 구문 트리로 표현된다. 본 논문에서는 생성된 추상 구문 트리를 기반으로 테스트 대상이 되는 실행 경로를 선택, 지정하는 알고리즘을 제안한다. 또한, 실행 경로를 연산으로 정의하고 문장이 실행되는 조건인 속성과 연산을 통해 실행 대상이 되는 조건을 찾는 알고리즘을 제안한다.

한번 생성한 경로에 따른 테스트 데이터는 XML[5]로 저장하여 테스트 로그로 사용한다. 기존의 테스트 데이터 생성 절차는 테스팅 적합성 기준을 선정하고 테스트 데이터를 선정 또는 생성하는 기준을 정하여 이 기준에 합당한 입력 데이터를 테스트 데이터로 사용한다[6]. 본 논문이 제안하는 방법은 사용자가 프로그램 소스를 분석하여 테스트 대상이 되는 경로를 입력으로 받는다. 실행 경로 작성 시 문장 단위로 실행 가능한 경우만을 입력으로 받기 때문에 테스팅에 적합성 기준을 선정하는 과정을 제거할 수 있다. 또한, 사용자가 지정한 경로에 따라 테스트 대상이 되는 소프트웨어 프로그램의 입력을 찾을 수 있다. 또한, 사용자가 지정한 경로와 생성된 테스트 데이터를 XML의 형태로 로그를 남겨 분석 결과 분석에 유용하다.

본 논문은 2장에서 관련 연구를 소개하고 3장에서 문장 결합을 통한 테스트 데이터 자동 생성 방법을 보인 후 4장 실험에서 제안한 방법을 실험하는 예제를 보인다. 그 후 5장 결론 및 향후 연구 과제에 대해 논한다.

2. 관련 연구

2.1. 일반적인 테스트 데이터 생성

기존의 테스트 데이터를 생성하기 위한 연구 방법은 두 가지로 나누어진다. 한 가지는 모형 검사 방법과 다른 한 가지는 정리 증명 방법이다[6][7][8]. 모형 검사 방법은 기본적으로 유한 상태 공간을 다루는 방법으로 완전한 자동화를 할 수 있다는 장점을 가지고 있다. 하지만, 무한한 모든 경우를 다룰 수 없고 초기의 유한한 상태 공간을 사용자가 지정해야한다는 단점을 가지고 있다. 반면 정리 증명 방법은 무한한 상태 공간을 다룰 수 있다는 장점이 있지만 무한 공간을 다루기 때문에 완전히 자동화하기 어려워 사용자가 증명 과정에 개입해야 한다는 단점이 있다[6].

2.2 임의의 값을 이용한 테스트 데이터 생성[9]

Bird는 “Automatic generation of random self-checking test cases”에서 임의의 값을 통해 테스트 케이스를 생성하고 테스트 대상에 대해 실험을 실시하였다. 이 방법은 테스트 데이터를 쉽게 생성할 수 있는 장점을 가지고 있지만 선택적인 실행 경로에 따른 테스트 데이터를 생성하기 위해서는 반복적인 많은 작업을 요구하는 단점을 가지고 있다.

2.3 함수 최소화 방법을 이용한 테스트 데이터 생성[10]

Korel은 임의의 값을 사용하는 방법이 경로에 따른 테스트가 어려운 약점을 해결하기 위해 임의의 값을 생성하여 실행하는 방법을 변형하여 개발자가 지정한 경로에 합당한 경우에 테스트를 진행하고 경로에 합당하지 않으면 테스트를 중지하고 다른 테스트 데이터를 생성하는 방법을 제안하였다. 이 방법은 기존의 임의의 값을 이용한 방법이 가지고 있는 경로에 따른 테스트 데이터를 결정할 수 없는 약점을 극복하였지만 임의 값이 선택한 경로를 실행하지 않을 경우 비효율적인 테스트 데이터를 생성한다는 약점을 가지고 있다.

2.4 일차 연립방정식을 이용한 테스트 데이터 생성[11]

Gupta는 실행 경로에 따른 테스트 데이터를 생성하기 위해서 블록의 실행 경로를 결정하는 제어문의 조건문을 추출한다. 추출된 조건문의 제약식의 계산을 일차 연립방정식으로 적용하여 연립방정식의 해를 계산함으로써 지정된 경로에 따른 테스트 데이터를 생성하는 방법을 고안하였다. 이 방법은 추출된 조건문의 분기 범위를 계산하여 주어진 경로에 따라 올바른 분기가 일어나는 테스트 데이터를 일차 연립방정식의 근사 해를 통해 생성하게 한다. Gupta의 방법은 기존의 한 번에 하나의 분기 조건만을 고려한 방법에 비해 적은 연산을 통해 테스트 데이터를 생성할 수 있는 장점이 있다. 하지만 Gupta가 제안한 방법은 분기에만 적용 가능하고 반복과 같은 제어문에 대한 분석을 고려하고 있지 않았고 제어문에 영향을 주는 명령문의 추출방법에 대해서는 소개하고 있지 않다.

3. 문장결합을 통한 테스트 데이터 자동 생성

본 논문에서 제안하는 테스트 데이터 생성 방법은 자동화를 지원하기 위해 테스트 대상이 되는 프로그램 코드와 실행 경로를 입력받아 테스트 데이터를 출력하기 위한 일련의 알고리즘과 모듈로 구성된다.

3.1 테스트 데이터 자동 생성

그림 1은 본 논문에서 제안하는 테스트 데이터 자동 생성을 위한 전체적인 모듈 개념도 이다.

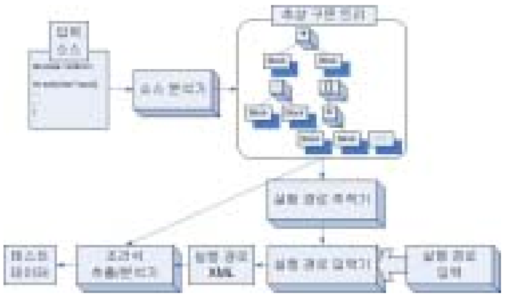

그림 1. 테스트 데이터 자동 생성 모듈

소스 분석 모듈은 문장 결합 정의에 따라 프로그램 소스를 연산 식으로 표현하고 추상 구문 트리를 생성한다. 실행 경로 추적 모듈과 실행 경로 입력 모듈은 생성된 추상 구문 트리를 바탕으로 제안된 알고리즘을 통해 테스트 대상이 되는 실행 경로를 결정한다. 조건식 추출/분석기는 결정된 실행 경로와 추상 구문 트리를 입력으로 하고 출력으로 테스트 데이터를 생성한다.

3.2 문장 결합과 연산 정의

절차적 프로그래밍 언어는 문장의 결합으로 구성되어 있다. 문장의 결합은 문장의 연속이나 문장이 제어문에 포함되는 구조를 갖는다. 문장의 실행 구조를 파악하기 위해서는 제어문을 포함하고 있는 문장의 결합 구조를 분석하여야 한다. 본 논문에서는 표준 ANSI C[12]의 문법을

기준으로 문장(statement)을 집합의 원소로 정의하고 그 특성을 분석한다. 정의 1은 ANSI C 문법의 문장(statement)의 집합 S의 정의이다.

정의 1. 문장의 집합 S
$S = \{ b \mid \langle statement \rangle \Rightarrow^{*} s \}$
단, $\langle statement \rangle \in N$ □

정의 1에서 정의한 집합 S의 원소는 프로그램 실행 시 실행 단위가 되고 프로그램은 집합 S의 원소가 제어문을 통해 결합되어 실행된다. 또한, 문장의 실행 경로는 제어문의 조건식을 통해 결정된다. 그러므로 제어문의 조건식을 정의 2와 같이 정의한다.

정의 2. 조건식의 집합 C
$C = \{ c \mid \langle selection-statement \rangle \Rightarrow^{*}$
"if" (c) <statement> or
$\langle iteration-statement \rangle \Rightarrow^{*}$
"while" (c) <statement> }
단,<selection - statement>, <statement>,<iteration - statement> ∈ N □

정의 2의 조건식 집합 C의 원소는 모듈에서 프로그램 실행 경로를 결정짓는 요소로써, while문과 if문의 조건식을 원소로 갖는 집합이다. C 언어의 while문은 반복문이고 if문은 대표적인 분기문이다. 또한, 모든 반복문은 while문으로 변환할 수 있고 모든 분기문은 if문으로 전환할 수 있다[13]. 즉, 정의 1의 문장 집합 S 원소간의 결합과 정의 2의 조건식을 이용하여 프로그램 코드를 분석할 수 있다. 그러므로 이와 같은 문장과 조건식의 관계를 정의 3에서 연산으로 정의한다.

정의 3. 제어 연산
①. “ [$statement_1$] [$statement_2$] ”
= $statement_1 \bullet statement_2$ 이고,
②. “ if[condition][$statement_1$] else[$statement_2$] ” = $statement_1^{<condition>} + statement_2^{<\sim condition>}$ 이며,
③. “ while [condition] [$statement_1$] ”
= $(statement_1)^{<condition>}$이다.
또한, $statement_1 \bullet statement_2$, $statement_1^{<condition>} + statement_2^{<\sim condition>}$,$(statement_1)^{<condition>}$ ∈ S 이다.
단, $statement_1$, $statement_2$ ∈ S, condition ∈ C □

프로그램 코드는 문장의 결합으로 이루어져 있는데 그 결합은 세 가지로 분류할 수 있다. 첫째, 정의 3의 ①과 같이 연속적으로 결합하여 선행 문장이 실행된 후 후행 문장이 실행되는 선형적인 결합이고 둘째, ②와 같이 두 문장이 연속적으로 결합하여 조건에 따라 하나의 문장이 실행되면 다른 하나의 문장이 실행되지 않는 특성을 가지는 if문의 결합이다. 그리고 세 번째 결합의 형태는 ③에서와 같이 조건에 따라 문장이 반복적으로 실행되는 while문의 결합으로 분류할 수 있다. 그러므로 정의 3 제어 연산은 프로그램 코드에서의 모든 제어문을 연산으로 표현한다.

3.3 문장결합 표현식과 구조적 정보

그림 1에서 소스 분석 모듈은 테스트 대상이 되는 프로그램 코드를 입력 받아 정의 1, 2, 3을 통해 문장 구조 표현식을 생성한다. 생성된 표현식은 구조 정보를 표현하기에는 다소 어려운 수식 형태를 가지고 있다. 따라서 소스 분석 모듈은 생성된 표현식의 구조 정보를 표현하기 위해 추상 구문 트리를 생성한다. 그림 2는 정의 3의 연산에 대해 소스 분석 모듈이 생성한 표현식과 그에 대한 구조 정보를 나타내는 추상 구

문 트리이다.

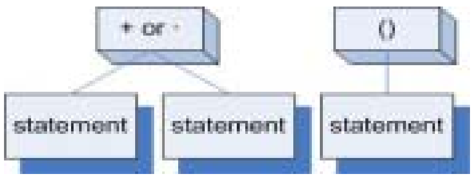

그림 2. 표현식의 추상 구문 트리

그림 2에서 “+”은 왼쪽 자식 문장과 오른쪽 자식 문장이 선택적으로 실행되는 결합이고 “•”은 왼쪽 자식 문장이 실행되고 오른쪽 자식 문장이 연속적으로 실행되는 결합을 의미한다.

본 논문에서는 그림 2와 같은 추상 구문 트리를 생성하기 위해 YACC[14]을 이용하여 파서를 생성하는 알고리즘을 제안한다. 표 1은 제안한 형태의 추상 구문 트리를 생성하는 파서 생성 YACC 소스의 일부이다.

표 1. 추상 구문 트리를 생성하는 YACC 코드

```
                              ...
statement_list
              :statement  {$$ = $1;}
              |statement_list  statement ;
              {$$ = AndOp($1, $2);}
selection_statement
              : 'if' '(' expression ')' statement
                'else' statement;
              {$$ = OrOp($3, $5, $7);}
iteration_statement
              : 'while' '(' expression ')' statement
              {$$ = RepeatOp($3, $5);}
                              ...
```

표 1에서 소개한 YACC 소스의 일부는 ANSI C 프로그래밍 언어를 파싱하는 전체 YACC 소스와 함께 테스트 대상이 되는 프로그램 코드를 분석하여 연산 노드를 생성한다. 표 2는 연산 노드를 생성하는 간단한 C 소스이다.

표 2. 추상 구문 트리 생성 알고리즘

```
andNode AndOp(statement l, statement r){
 andNode p;              p->l_link = l;
 p->r_link = r;          return p; }
orNode OrOp(statement l, condition c, statement r){
 orNode p;               p->l_link = l;
 p->r_link = r;          p->c_link = c;
 return p; }
repeatNode RepeatOp(statement s, condition c){
 repeatNode p;           p->link = s;
 p->c_link = c;          return p; }
```

3.4 실행 경로 선택

소개한 소스 분석기를 통해 생성된 추상 구문 트리는 실행 경로 추적 모듈의 입력으로 사용된다. 실행 경로 추적 모듈은 추상 구문 트리를 분석하고 실행 경로 입력 모듈은 사용자에게 실행 경로를 입력 받는다. 이때, 사용자는 생성된 추상 구문 트리의 구조 정보로부터 선택 가능한 경로에 대한 정보를 얻을 수 있다. 따라서 사용자는 실행 경로를 결정적으로 선택할 수 있다. 이것은 기존의 테스팅 적합성 기준을 판별하는 과정을 제거해 보다 효율적인 테스트 실행 경로 설정이 가능하다.

정의 4의 연산은 실행 경로를 표현하는 연산이다. 정의 1의 집합 S을 이용해 실행 경로에 관한 연산을 정의한다.

정의 4. 실행 경로 결정을 위한 연산

①. s_1이 실행된 후 s_2가 실행되는 연산은 “ $s_1^{<c_1>} \oplus s_2^{<c_2>}$ ” 이다.

②. 조건 c_1과 c_2 이 만족할 경우 s_1이 실행된 후 s_2가 실행되는 연산은 “ $s_1^{<c_1>} \odot s_2^{<c2>}$ ” 이다.

단, s_1, $s_2 \in S$, c_1, $c_2 \in C$ □

정의 4는 실행에 특성에 따라 문장이 결합하는 구조를 두 연산으로 정의한다. 실행 경로는 문장의 제어에 따라 두 가지 특성을 가지고 있다. 한 가지는 정의 4의 ①과 같이 조건에 상관없이 먼저 실행된 문장의 조건에 상관없이 후에 실행되는 문장이 실행되는 문장의 결합이 있다. 다른 한 가지는 ②와 같이 선 실행된 문장의 조건이 다음에 실행되는 문장에 영향을 주는 경우가 있다. ②에서 정의한 “$s_1^{<c_1>} \odot s_2^{<c2>}$”의 연산이 실행되기 위해서는 조건식 “($c_1$ && c_2)”가 만족해야한다. 또한, 정의 4의 ①식 “$s_1^{<c_1>} \oplus s_2^{<c2>}$”이 실행은 조건에 상관없이 실행되지만 조건이 있는 경우 “(c_1 && c_2)”의 조건을 갖는다. 왜냐하면, s_1과 s_2가 모두 실행되는 경로이므로 두 조건이 모두 참일 경우에만 경로 표현식의 실행 경로에 따라 실행되기 때문이다.

3.5 조건식 추출과 연산

사용자로부터 받은 실행 경로는 XML 문서의 형태로 저장한다. 이것은 실행경로에 영향을 줄 수 있는 조건식의 추출과 연산을 쉽게 할 수 있다. 표 3은 실행 경로를 XML로 저장하기 위한 DTD이다.

표 3. 실행 경로 XML의 DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT function (statement | operator)*>
<!ELEMENT statement (#PCDATA)>
<!ATTLIST statement
            id CDATA #REQUIRED
            condition CDATA #REQUIRED>
<!ELEMENT operator (#PCDATA)>
<!ATTLIST operator
            type CDATA #REQUIRED>
```

테스트 대상이 되는 프로그램 코드의 구조를 표현하고 있는 추상 구문 트리와 실행 경로를 저장하는 XML 문서는 조건식 추출/분석기의 입력이 된다. 조건식 추출/분석기는 입력된 추상 구문 트리와 실행 경로를 바탕으로 실행 경로에 영향을 주는 조건식을 추출하고 3.1에서 소개한 조건식의 연산 방법에 따라 조건식을 연산하여 변수의 범위를 찾아내고 변수의 범위에 따라 입력의 범위를 찾는다.

4. 실 험

본 논문에서 절차적 언어인 ANSI C 언어로 작성된 프로그램 소스를 분석하여 테스트 데이터를 생성하는 과정을 예제로 보인다. 실험에서 프로그램 소스를 제안한 정의 1, 2, 3에 따라 표현식으로 나타내고 구조 정보를 위한 추상 구문 트리를 생성한다. 또한, 정의 4를 통해 실행 경로를 추출하여 XML로 저장하고 조건식을 추출한다. 마지막으로 추출된 조건식의 연산에 따라 사용자가 입력한 실행경로에 대한 테스트 데이터를 생성한다. 표 4는 테스트 대상이 되는 간단한 C 코드이다.

표 4. C 프로그램 예제 코드

```
int input[10];

int binarySearch(int low,int high,int target){
    int mid = (low + high)/2;/*statement0*/
    while(low <= high){
        mid = (low + high) / 2;/*statement1*/
        if(input[mid] == target){
            return mid;/*statement2*/
        }else{
            if(target > input[mid]){
                low = mid + 1;/*statement3*/
            }else{/*statement4*/}
            if(target < input[mid]){
                high = mid - 1;/*statement5*/
            }else{/*statement6*/}
        }
    }
    return -1;/*statement7*/
}
```

표 4의 프로그램 코드는 input에 있는 배열에서 특정 자료를 찾아내는 이진 탐색 프로그램의

함수이다. 표 4의 제어문 구조를 분석하여 앞에서 제안한 연산으로 표현하면 표 5와 같은 표현식으로 나타난다. 표현식을 단순화하기 위해 statement는 s로 표기한다.

표 5. 코드를 분석한 표현식

$$s0\cdot(\{s1\cdot\{s2^{<input[mid]==target>}+\{\{s3^{<target>input[mid]>}+s4^{<target<=input[mid]>}\}s5^{<target<input[mid]>}+s6^{<target>=input[mid]>}\}\}\}^{<input[mid]!=target>}\}^{<low<=high>})\cdot s7$$

표 5의 표현식은 프로그램으로 분석하기 어렵기 때문에 그림 2에서 소개한 추상 구문 트리의 형태로 나타낸다. 그림 3은 표 5의 표현식을 나타낸 추상 구문 트리이다.

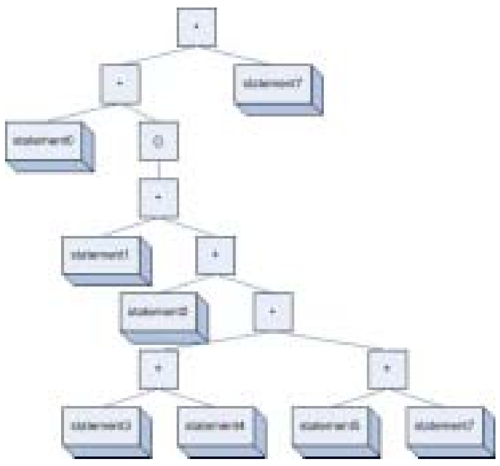

그림 3. 표 2 예제에 따른 추상 구문 트리

그림 3과 같은 형태로 분석된 추상 구문 트리를 분석하여 사용자에게 실행 경로를 입력으로 받으면 표 6과 같은 실행 경로를 나타내는 표현식을 생성할 수 있다.

표 6. 실행 경로를 나타내는 표현식

$$s_0 \oplus (s_1^{low \le high} \odot s_3^{(input[mid] \neq target) \text{and} (target > input[mid])} \oplus s_6^{target \ge input[mid]}) \oplus (s_1^{low \le high} \odot s_4^{(input[mid] \neq target) \text{and} (target \le input[mid])} \oplus s_5^{target < input[mid]}) \oplus (s_1^{low \le high} \odot s_2^{input[mid] = target})$$

표 6과 같은 표현식은 프로그램으로 다루기가 어렵기 때문에 실행 경로를 표 7과 같은 XML 문서로 작성 가능하다. 표 7은 실행 경로를 나타내는 XML의 일부이다. 표 7의 XML에서 ‘<’나 ‘>’의 문자를 사용하면 잘 정의된(well-defined) XML 문서가 되지 않는다. 그러므로 ‘<’ 문자를 ‘[’문자로 사용하고 ‘>’문자는 ‘]’ 문자로 사용한다.

표 7. 실행 경로를 나타내는 XML의 일부

```
<?xml version="1.0" encoding="UTF-8"?>
<function>
<statement id="s0" condition=""/>
<operator type="or"/>
<statement id="s1" condition="low[=high"/>
<operator type="and"/>
<statement id="s3" condition="(input!=target)and
  (target]input[mid])"/>
                    ...
</function>
```

표 7의 실행 경로와 그림 3의 추상 구문 트리를 이용하여 표 8과 같은 조건식과 조건식에 영향을 주는 문장을 추출할 수 있다.

표 8. 조건식과 추출문장

실행 문장	조건식	추출문장
s_0		
s_1	low_0<=$high_0$	
s_3	(input[mid_0]!=$target_0$)and ($target_0$>input[mid_0])	mid_0 = (low_0+$high_0$) / 2;
s_6	$target_0$>=input[mid_0]	
s_1	low_1<=$high_0$	low_1 = mid_0 + 1;
s_4	(input[mid_1]!=$target_0$)and ($target_0$<=input[mid_1]	mid_1 = (low_1+$high_0$) / 2;
s_5	$target_0$<input[mid_1]	
s_1	low_1<=$high_1$	$high_1$ = mid_1 - 1;
s_2	input[mid_2]==$target_0$	mid_2 = (low_1+$high_1$) / 2;

표 8의 조건식과 추출문장의 변수는 변수의 값이 변화하는 과정을 설명하기 위해 변수에 대한 수열로 표현한다. 표 4의 예제에서 input배열을 고정시키고 실험한 결과는 표 9와 같다. input배열의 고정 값은 0부터 9까지의 정수이고, low와 high는 배열의 시작과 끝을 가르치는 인덱스이므로, 실제 입력 데이터는 target이다. 표 9는 표 6의 경로에 따라 생성된 테스트 데이터이다.

표 9 테스트 데이터 생성

index	target	low	high	mid
0	5	0	9	4
1		5		7

표 9와 같은 테스트 데이터는 테스트 대상이 되는 소스의 결정적 테스트 데이터로 제공된다.

5. 결 론

본 논문은 프로그램 소스 문장의 결합 구조를 대수적 연산식으로 표현하고 그것을 통해 테스트 데이터를 자동으로 생성하는 방법을 제안한다. 본 논문은 생성된 표현식의 구조정보를 이용하기 위해 추상 구문 트리 생성 알고리즘을 이용한다. 제안한 추상 구문 트리는 실행 경로를 사용자로부터 결정적으로 입력 받기 위한 기본적인 자료구조로 사용된다. 본 논문에서는 추상 구문 트리의 구조 정보와 사용자 실행 경로 입력 정보를 통해 실행 경로를 결정하기 위한 알고리즘을 제안한다. 따라서 제안한 알고리즘과 방법은 기존의 테스팅 적합성 판별 단계를 거치지 않고 테스트 데이터를 자동 생성할 수 있다. 본 논문에서 제안한 선택된 실행 경로를 XML 문서로 표현하여 로그로 사용하기 위한 DTD를 제안한다. 또한, 입력 받은 실행 경로의 조건식을 추출하기 위한 알고리즘을 제안한다. 본 논문은 제안한 알고리즘과 방법에 대한 실험을 위해 소스 분석 모듈, 실행 경로 추적 모듈, 실행 경로 입력 모듈, 조건식 추출/분석 모듈로 구성된 일련의 테스트 데이터 자동 생성 과정을 보였다.

본 논문에서 제안한 테스트 데이터 자동 생성 방법은 프로그램 소스를 정형화된 대수적 표현식으로 표현하고 이것을 통한 자동화 가능성을 제시함으로써, 테스트 데이터 생성 관련 어플리케이션 개발에 도움이 될 것으로 기대된다.

참 고 문 헌

[1] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 2nd edition, 1990.
[2] http://www-306.ibm.com/software/awdtools/ test/realtime
[3] HcMillan, G. J., *Design and Validation of Computer Protocols, Prentice Hall*, 1991.
[4] Larsen, K, Pettersson, P., and Yi W., "UPPAAL in a Nutshell", Springer *International Journal of Software Tools for Technology Transfer*, 1(1+2), 1997.

[5] http://www.w3.org/xml

[6] 정인상, “소프트웨어 테스팅을 위한 테스트 데이터의 자동 생성” *정보과학회지 제19권 제11호*, pp.10~18, 2001. 11.

[7] McMillan, K L, *Symbolic Model Checking An Approach to the State Explosion Problem*, Kluwer Academy, 1993.

[8] Griffioen, D. and Huisman, M, "A Comparison of PVS and Isabelle/HOL", *Theorem Proving in Higher Order Logics: 11th International Conference*, 1998.

[9] Bird, D.L. and Munoz, C. U., "Automatic generation of random self-checking test cases", *IBM Systems Journal, Vol. 22*, pp.229~245, 1983.

[10] Korel, B., "Automated Software Test Data Generation", *IEEE Trans. on Software Eng*, Vol. 16, no 8, pp.870~879, 1990.

[11] Gupta, N., Mathur, A, and Soffa, M. L., "Automated test data generation using an iterative relaxation method", *In Proceedings of Foundations of Software Engineering*, ACM Press, Nov. 1998.

[12] http://www.lysator.liu.se/c/

[13] Steve McConnell, *CODE COMPLETE*, 1993.

[14] http://dinosaur.compilertools.net/

곽 동 규

2001년 서경대학교 응용수학과(학사).
2002년~2004년 숭실대학교 컴퓨터학과(석사).
2004년~현재 숭실대학교 컴퓨터학과(박사 과정).
관심분야 : 프로그래밍 언어, 컴파일러, XML.

조 용 윤

1995년 시립 인천대학교 전자계산과(학사).
1996년~1998년 숭실대학교 전자계산학과(석사).
1999년~현재 숭실대학교 컴퓨터학과(박사 과정).
관심분야 : 프로그래밍 언어, 컴파일러, XML, HCI, 컴퓨터 통신.

김 영 철

1990년 한남대학교 전자계산과(학사).
1994년~1996년 숭실대학교 전자계산학과(석사).
1996년~2003년 숭실대학교 컴퓨터학과(박사).
현재 (주) 뉴스텍시스템즈 이사, 명지전문대학 겸임교수.
관심분야 : 프로그래밍 언어, 컴파일러, 망관리, XML, 컴퓨터 통신.

유 재 우

1976년 숭실대학교 전자계산학과(학사).
1985년 한국과학기술원 전자학과(박사).
1983년~현재 숭실대학교 컴퓨터학부 교수.
1986~87년, 1996~97년 코넬대학교, 피츠버그대학교 객원교수.
1999~2000년 한국정보과학회 프로그래밍언어 연구회 위원장.
관심분야 : 프로그래밍 언어, 컴파일러, 인간과 컴퓨터 상호작용.