한국산학기술학회논문지
Vol. 10, No. 5, pp. 985-991, 2009

CFG를 이용한 소프트웨어 테스트 케이스의 자동 생성

윤성희[1*]
[1]상명대학교 컴퓨터소프트웨어공학과

Automatic Generation of Software Test Cases using CFG

Yoon, Sung Hee[1*]
[1]**Department of Computer Software Engineering, Sangmyung University**

요 약 문법 기반의 테스트 케이스 자동 생성(GBTG) 알고리즘은 문법 G를 기반으로 G에 의해 수용되는 문장들의 집합 L(G), 즉 소프트웨어 테스트를 위한 테스트 케이스를 생성한다. 문맥 자유 문법으로 기술되는 대부분의 언어들은 순환적 성격을 가지며, 일반적으로 G에 의해 생성되는 집합 L(G)는 너무 커서 생성된 케이스들을 모두 실행시킬 테스트 대상으로 삼는다는 것은 현실적이지 못하다. 본 논문에서는 CFG에 의해 자동 생성되는 테스트 케이스의 집합의 크기를 통제하면서도 테스트의 주요 범위를 커버하기 위하여 문맥 자유 문법을 보강하는 문법외적 요소인 태그들을 개발하고 실험하였다.

Abstract A grammar-based test case generation (GBTG) algorithm takes a grammar G and generates a subset of the language L(G) accepted by G, and called test cases for software product testing. As most languages specified with CFG are recursive, usually L(G) is so large that it is not practical to execute all the of the generated cases. Therefore, this paper presents some "tags" : extra-grammatical annotations which are designed to restrict the generation. A number of control mechanisms have been developed to prune the number of test cases generated while still producing a test set that covers the majority of inputs to the system.

Key Words : Software test automation, test cases generation, CFG

1. 서론

소프트웨어 프로젝트의 수행과정에서 시간이 매우 많이 소모되는 과정 중 하나는 테스트 데이터(test data)를 생산하고 그 결과를 검증하는 과정이다. 소프트웨어 개발자의 입장에서나 소비자의 입장에서나 소프트웨어 오류(error)를 미리 발견하고 수정함으로써 얻을 수 있는 비용절감의 효과는 매우 크다. 소프트웨어 테스트 과정을 통하여 소프트웨어 생산물이 예상하는 만큼 잘 작동되는지를 검증해야 하며, 테스트 과정에서는 어떤 예상되는 타당한 테스트 데이터를 입력하고 기대되는 결과가 산출되어야 하는지를 찾아야 한다. 즉, 테스트 케이스(test cases)란 바로 소프트웨어의 동작을 테스트하기 위해 주어지는 입력이며, 소프트웨어의 기능이나 성능에 대하여 던지는 일종의 질문으로서 학문분야 용어의 정의에 의하면 “특정 프로그램의 경로나 소프트웨어의 요구사항 확인 등 일정한 목적을 위하여 개발된 테스트 입력, 실행 조건, 예상 결과의 집합체”라고 한다. 테스트 케이스는 얼마나 오류를 잘 발견할 수 있도록 구성되었으며 또 그 규모가 최소화된 집합으로 구성되는가가 테스트 과정의 효율을 좌우한다[1,2,6,11]. 좋은 테스트 케이스는 테스트 목적에 맞는 오류를 충분히 잘 드러내고 최소한의 비용과 노력으로 원하는 테스트를 커버(cover)할 수 있어야 한다. 이와 같은 오류추출성(readable)과 커버리지(coverage) 속성 외에도 경제성(minimize cost), 수정가능성(up-to-date), 구조성(structural), 문서화(documented) 등의 속성이 좋은 테스트 케이스의 속성으로 알려져 있다[2].

일반적인 테스팅 과정은 예상되는 입력인 테스트 케이

본 논문은 2008년 상명대학교 교내연구비지원에 의하여 연구되었음.
*교신저자 : 윤성희(shyoon@smu.ac.kr)
접수일 09년 02월 27일 수정일 (1차 09년 04월 13일, 2차 09년 05월 01일) 게재확정일 09년 05월 27일

스의 생성, 테스트 케이스에 대한 기대되는 출력, 테스트 케이스의 실제 실행과 출력 확보, 기대한 결과와 실제 결과의 비교 과정이 반복적으로 진행된다. 그럼에도 불구하고 이와 같은 테스트 과정을 거치면서 소프트웨어가 갖는 오류를 발견할 수는 있지만, 오류가 없음을 증명하는 것은 아니다[4,8]. 반면에 전통적인 소프트웨어 테스트 방법에서 테스트 케이스(test case)의 수동 생성과정은 시간과 비용의 소모가 매우 큰 과정이다. 따라서 소프트웨어 테스팅(testing) 연구 목표의 중요한 한 방향은 소프트웨어 툴(tool)을 이용하여 성(automatic generation) 방법이며 소프트웨어의 오류를 최대한 발견할 수 있는 테스트 케이스를 자동 생성하는 것이라고 할 수 있다[11].

본 논문에서는 소프트웨어 테스트 케이스의 자동 생성을 위해 전통적인 언어 분석 도구이며 생성 도구인 문맥자유문법(CFG : Context Free Grammar)을 기반으로 하며 생성 범위를 통제하기 위해 태그를 이용하는 테스트 케이스의 자동 생성 방법과 두 가지 개발 소프트웨어에서의 실험을 소개한다.

2. 관련 연구

문법 기반의 테스트 케이스 생성(GBTG : Grammar-based Test case Generation)에 관한 연구로서 블랙박스 테스트 접근 방법으로 구분될 수 있다. 생성의 기반이 되는 문맥 자유문법(CFG: context-free grammar)은 문맥 자유 언어에 대한 형식적 명세(formal specification)이다. 문맥 자유 문법은 전통적으로 어떤 문자열이 주어진 문법의 언어에 속하는지를 판별하기 위하여 분석(analysis) 도구로 사용되어 왔다. 특히 입력문자열을 파싱(parsing)하고 계산(evaluation)을 위한 내부 표현인 파스 트리(parse tree)를 생성할 수 있는지를 결정하는 과정으로서 프로그래밍 언어의 컴파일(compile) 과정에서 매우 중요하게 연구되어 왔다[4,5,15,17]. 반면에 소프트웨어 테스트 과정으로서는 컴파일의 성능을 테스트하기 위해서 주어진 문법의 언어에 속하는 문자열을 테스트용 입력으로 생성하는 반대 방향의 과정이 필요하다. 문맥자유문법(CFG: Context Free Grammar) 기반으로 기술된 문법 규칙들을 Top down 및 left-to-right 방식의 자동 유도(derivation) 알고리즘에 의해 테스트 데이터를 생성하며, 깊이 우선 탐색(depth-first-traversal) 순서로 적용되는 과정에서 백트래킹(backtracking)을 일으키는 일반적인 문법기반 생성(generation) 알고리즘이 기본이 된다[5,14,16].

문법 기반 테스트 방법으로 테스트 케이스를 자동 생성하기 위해서 테스터는 BNF(Backus-normal Form) 형식을 갖는 문맥 자유 문법(Context Free Grammar) G로 테스트 케이스의 구문(syntax)을 명세한다. 또한 테스트 케이스의 자동 생성기는 이 문법 G에 의해 수용되는 언어 L(G)를 생성하는 도구가 된다. 이와 같이 소프트웨어 테스트 도구로서 응용될 경우 문맥 자유 문법은 입력 언어의 구문을 명세하고 테스트 케이스의 문장을 자동 생성하는 것뿐만 아니라 네트워크 프로토콜(network protocol)이나 웹 테스팅(web testing)을 위한 XML과 같은 일련의 데이터 형식을 표현하기 까지 폭넓게 그 문법을 기술하게 위해 사용될 수 있다[7,9,10]. 이러한 경우 문법은 테스트 케이스(test case)를 생성하기 위한 구문적 명세(syntax specification)이며, 응용에 따라서 테스트 케이스는 고급 프로그래밍 언어의 코드처럼 복잡할 수도 있으면 이진코드처럼 단순한 형식이 될 수도 있다.

문맥자유문법의 실질적인 응용을 위해서는 테스트 케이스의 자동 생성 과정에서 순환(recursion)에 의한 무한 생성을 제어해야 하며, 유도 트리(derivation tree)의 깊이를 제어해야 하고 경우에 따라서는 문법외적 조검은 검사하는 제약이 필요하다. 이를 위하여 문맥 자유 문법의 규칙에 부가되는 다양한 태그들을 설계하고, 이 태그를 이용하여 테스트 데이터의 형식과 종류와 전체 크기를 제어할 수 있도록 문법을 보강하는 방법을 적용해야 한다.

여러 관련연구에서 볼 수 있듯이 문법기반 테스트케이스 생성은 컴파일러 테스트분야에서 널리 적용되어 왔으며, 주로 형식적 명세를 정의하는 데 중심을 두어왔다[5,7,16,17]. 이 경우에 생성되는 테스트 케이스는 프로그래밍언어의 단위별로 테스트를 위한 입력프로그램이 된다[4,8,10,15] 문맥자유문법의 무한반복적 생성 메커니즘에 제어기능을 부가하는 연구도 많이 소개되고 있으며 이들 연구들은 대부분 유도과정에서 트리의 깊이와 순환의 깊이를 중심으로 통제하는 방법을 논하고 있으나 실제 소프트웨어 도메인에서의 실험에 대한 결과는 구체적으로 소개되고 있지 않다[13,14,16]. 이론적 한계를 벗어나서 문법기반 테스트 케이스의 생성방법을 실제 구현된 소프트웨어의 테스트에서 사용되기 위해서는 테스트케이스의 생성과정에 보다 적극적인 통제가 필요하다. 본 연구는 특히 생성규칙 각각에 비중을 달리 두어 선별적으로 적용할 수 있는 태그와 내부 코드를 삽입함으로써 문법외적 조건을 검사하면서 규칙을 적용하는 태그 등으로 확장 설계하여 생성의 범위를 테스터의 요구에 맞게 조정할 수 있게 하였다.

3. 생성을 위한 문맥 자유 문법과 태그

기술된 문맥자유문법에 따라 순환적 알고리즘(recursive)과 백트래킹 메커니즘(backtracking mechanism)에 의해 테스트 케이스를 자동 생성하는 이론적인 생성 방법은 실제로는 그대로 적용되기 어렵다. 특히 대부분 순환적 정의에 의해 강력한 표현력을 갖는 문맥자유문법은 생성 과정에서 본질적으로 무한한 개수의 테스트 케이스를 생성하면서 종료할 수 없는 상황에 빠지게 되므로 실질적이고 효율적인 생성을 위해서는 다음과 같은 중요한 몇 가지에 문제에 초점을 두어 해결해야 한다.

- 무한한 개수의 테스트 케이스를 생성하고 테스트할 수 없으므로 생성되는 문자열, 즉 테스트 케이스의 개수를 제한할 수 있어야 한다.
- 매우 유사한 테스트 케이스들이 반복적으로 생성되지 않도록 생성 과정, 즉 제어의 방향을 통제할 수 있어야 한다.
- 문맥 의존적(context-sensitive) 성질을 일부 명세할 수 있어야 한다. 문법외적 요소로서 특정 속성을 값으로 유지하거나 검사할 수 있어야 한다.

본 논문에서는 소프트웨어 테스트를 위한 테스트 케이스의 자동 생성을 위해 문맥자유 문법과 생성 알고리즘을 기본으로 사용하면서, 위와 같은 제약성을 극복하기 위해 생성 과정을 통제할 수 있도록 문법 규칙에 부가될 수 있는 문법 외적인(extra grammatical) 요소인 다양한 태그(tag)를 제안한다. 문법에 추가된 태그는 목적에 따라서 불린식(boolean expression)처럼 단순할 수도 있고, 내포된 코드(embedded code)의 형식을 가질 수도 있다. 테스트 케이스 생성을 위한 태그는 세 가지 카테고리로 구분될 수 있다. 언어 트리의 순회 과정에서 특정 경로를 제약하거나 일반적인 생성 과정을 중지시키는 태그들, 만족시키는 경로를 따르도록 순회 순서를 변경하는 태그들, 생성 과정에서 특정 성질에 대해 추적하기 위한 태그들로 성질을 구분된다.

3.1 Count 태그

Count 태그는 문법 규칙이 생성 과정에서 사용되는 회수를 제한하기 위한 간단한 태그이다. 다음의 예 1에서 [Count 2] 태그에 의해 첫 번째 규칙이 선택되고 확장되는 회수를 제한하며, 이 문법이 생성하는 테스트 케이스의 집합은 {0, 00}로 제한된다.

```
[count 2]  Zeros  →  Zeros '0'
           Zeros  →  '0'
```

[예 1] count 태그를 갖는 간단한 문법

3.2 Weight 태그

Weight 태그는 랜덤 생성 알고리즘(random generation algorithm)이 적용될 때, 규칙의 선택에 대해 우선권을 부여하기 위한 것이다. 예 2에서 [Weight 2] 태그가 부여된 두 번째 규칙은 첫 번째 규칙에 비해서 랜덤 선택 과정에서 선택될 확률을 두 배로 갖게 된다. 또한 첫 번째 규칙은 순환 규칙이고 두 번째 규칙은 순환 생성을 마감할 수 있는 규칙이므로 위와 같은 태그의 부착을 통해서 생성되는 테스트 케이스의 형태가 무모하게 반복되는 긴 문자열이 될 가능성을 크게 줄여준다. 이러한 성질은 문맥자유 문법의 순환적 본질을 통제하여 이론적 배경에서 실질적 응용으로 전환하는 데 중요한 역할을 할 수 있다.

```
[weight 1] Zeros  →  Zeros '0'
[weight 2] Zeros  →  '0'
```

[예 2] Weight 태그를 갖는 문법

만약 [Weight 0] 태그가 부착된 규칙이 있다면 그 규칙은 테스트 케이스 생성과정에서 선택되지 않을 것이며, 테스트 케이스 생성 과정에서 테스트 범위(scope)를 제한하는 중요한 수단이 된다.

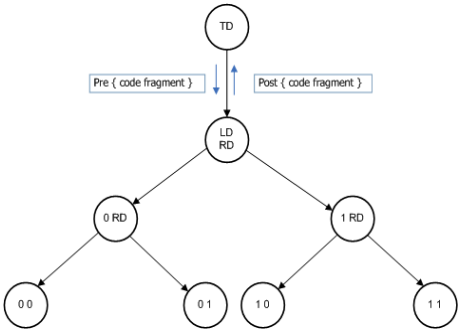
3.3 Guard 태그

Guard 태그는 불린식(boolean expression)의 형식을 갖는 태그로서, 문법 규칙이 선택되어 생성을 위해 확장되기 전에 식의 값을 평가한다. 불린식의 값이 참(true)이면 규칙에 의해 확장되는 생성이 계속될 것이며, 거짓(false)인 경우 다른 규칙을 선택하게 된다. 이 태그는 문맥 자유 문법이 해결하지 못하는 일부 문맥 의존적(context sensitive)인 성질을 검사하기 위해 사용되기도 한다. 태그 Guard는 내부 데이터 구조의 값을 참조하거나 검사하기도 하고, 생성 트리(generation tree)에서 상속되거나 계산되는 값들을 참조하거나 검사한다. 예를 들면, 생성되는 테스트 케이스의 길이를 속성과 그 값으로 지정하고, 새로운 규칙에 의해 확장될 때 그 값을 참조하고 변경할 수 있으므로 이 태그는 생성되는 트리의 특정 경로를 제한하는 종류의 태그이다.

3.4 Action 태그

Action 태그는 문법 규칙에 부가된 코드부(code segment)

이다. Action은 생성 과정에서 규칙이 선택되고 확장되기 전에 실행되는 pre-Action과 확장된 후에 실행되는 post-Action으로 구현된다. 그림 1에서는 생성 과정이 하향(top-down)으로 확장될 때 pre-Action이 실행되고, 상향으로 복귀할 때 실행되는 post-Action의 적용되는 방향을 보여준다.



[그림 1] pre-Action 태그와 post-Action 태그

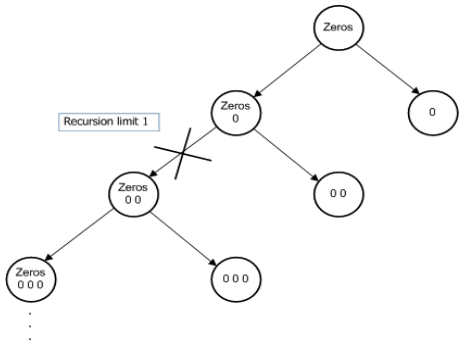
Guard 태그와 비슷하게 Action 태그도 생성되는 트리, 즉 테스트 케이스의 생성 트리에서 여러 가지 데이터 값을 참조하고 계산한다. Action 태그는 생성 과정을 매우 강력하게 통제할 수 있는 태그이므로 생성되는 테스트 케이스의 문맥 조건을 검사하거나 길이를 계산하거나 그 실행 결과 값을 예측하는 등 여러 가지 다양한 목적으로 사용될 수 있다.

3.5 Recursion 태그

문맥 자유 문법은 본질적으로 순환 규칙(recursive rules)에 의해 강력한 표현력(expressive power)을 갖는다. 반면에 특히 깊이 우선(depth first) 생성 알고리즘을 적용할 때 순환 규칙을 제어하지 않으면 무한 순환에 빠지는 상황을 피할 수 없다. 실질적인 테스트 케이스의 생성을 위해 순환의 회수를 제한하는 recursion 태그를 아래와 같은 방법을 사용한다. 아래 예에서, 첫 번째 규칙은 순환 적용이 한번만 가능하게 제한하며, 그 이후에는 비순환 규칙만 선택되게 제한하는 순환 제한(recursion limit)을 위한 태그이다.

[recursion 1] Zeros → Zeros '0'
Zeros → '0'

[예 3] 순환 제한 recursion 태그의 예



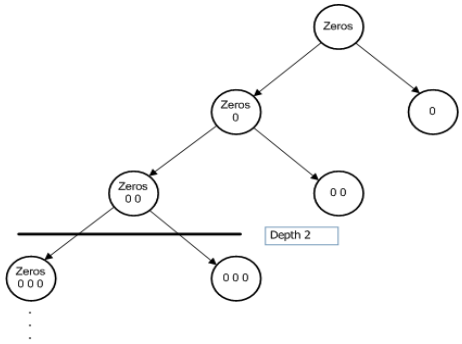
[그림 2] recursion 태그에 의한 생성 경로의 제어

3.6 Depth 태그

Depth 태그는 생성되는 테스트 케이스의 트리의 깊이(depth)를 제한함으로써 전체적으로는 생성되는 테스트 케이스 전체 집합의 크기를 조절할 수 있다. Depth의 계산은 태그가 부착된 해당 nonterminal로부터 시작되며, 특정 깊이 이하로는 테스트 케이스의 생성이 가능하지 않으므로 depth에 대한 임계값(threshold)을 갖게 된다. 여기서 예로 드는 간단한 문법에서의 임계값은 1이 된다.

[depth 2] Zeros → Zeros '0'
Zeros → '0'

[예 4] depth 제한 태그의 문법 예



[그림 3] depth 제한 태그에 의한 생성 경로의 제어

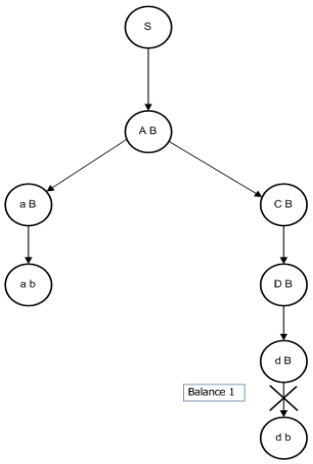
3.7 Balance 태그

앞에서 예로 든 간단한 문법에서만 보더라도 생성되는 테스트 케이스의 생성 트리는 종종 불균형적으로 한쪽으로 치우치는 형태를 갖게 된다. 유한 개수의 다양한 형태의 테스트 케이스를 생성하기 위해 이와 같은 형태의 생성을 피하고 생성되는 트리의 균형(balance)을 유지할 수 있도록 생성 과정을 제어할 수 있어야 한다. Balance 태

그는 서로 다른 nonterminal들 사이에서도 생성 트리의 깊이의 균형을 제어할 수 있는 태그이다. 두 기호들에 의해 생성되는 깊이의 차이가 태그에 지정된 값을 넘지 않도록 생성 과정이 제어된다.

```
[balance 1]   Skewed → A B
                       A → C
                       A → 'a'
                       B → 'b'
                       C → D
                       D → 'd'
```

[예 5] balance 제어를 위한 태그의 문법 예



[그림 4] balance 태그에 의한 생성 경로의 제어

다음의 [그림 5]는 태그를 갖는 CFG의 생성 알고리즘을 간략하게 보여주고 있다.

```
//some produciton rules are augmented with tags
// tag = {tag_name, tag_argument}
// each sentential forms has data structure D of
// derivation tree and tag
Tlist gen(S)
    if S is empty
        return []
    head = S[0]
    tail = S[1:]
    if head is a nonterminal
        foreach rule R in G where R.lhs = head
            foreach s0 in gen(R.rhs)
                foreach s1 in gen(tail)
                 if s0+s1 satisfies {tag_name, tag_argument}
                    yield s0+s1
                     updates derivation tree information D
    else
        foreach s1 in gen(tail)
          if head+s1 satisfies {tag_name, tag_argument}
            yield head + s1
```

[그림 5] 태그를 갖는 CFG의 생성 알고리즘

4. 실험

본 논문에서 소개하고 있는 태그된 문맥 자유 문법에 기반한 테스트 케이스의 생성 방법을 실험하기 위하여 두 가지 다른 종류의 소프트웨어를 대상으로 실험하였다. 우선 태그 부착의 영향력을 보기 위해서 단순한 연산 문법을 이용하여 depth에 따른 생성(시작깊이=1) 과정을 실험한 결과는 표 1과 같다. G1, G3, G3에서는 문법에서 단말 또는 비단말 항의 수가 늘어나면 동일하거나 유사한 규칙의 반복 적용에 의해 생성되는 케이스의 수가 크게 증가함을 볼 수 있다. G1'의 경우는 G1에 비하여 순환에 의해 급증하는 생성을 통제할 수 있다.

[표 1] depth와 recursion을 통제한 생성

depth	G1	G2	G3	G1’
2	1	3	6	1
3	2	42	156	1
4	10	8148	105,144	3
5	170	268,509	overflow	21

G1 : 단말 a와 산술단항연산자 -, 산술이진연산자 +를 표현하는 문법
G2 : 문법 G1을 확장하여 단말 a, b, c와 산술이진연산자 +, -, *, /을 표현하는 문법
G3 : 문법 G2를 확장하여 단말 1,2,3을 포함하는 문법 순환제어의 영향력을 보기 위해 G’을 다음과 같이 설정.
G1’ : G1문법에서 산술단항연산자 -의 Recursion 태그가 2인 문법

본 연구의 실험을 위해 언어 정보 처리 분야에서 자연어 질의를 입력으로 하는 웹 검색엔진[3]을 테스트하기 위한 입력 질의어를 자동 생성하였다. 이 연구는 본래 검색엔진을 사용하는 다수의 사용자들로부터 수집된 실제 자연어 질의 문장을 통계적으로 분석하여 입력 대상으로 삼아 성공적으로 수행된 연구이다. 다만 본 논문에서 소개하는 테스트 케이스의 자동 생성 과정을 실험하기 위해서는 질의 문장을 생성하는 태그된 문법과 자동 생성

기를 이용하여 태그의 집합을 달리하면서 생성되는 문장의 유형을 분류하였다. 질의문장들의 유형은 실제 수집된 자연어 질의들을 분류한 기존의 실험을 기반으로 하였으며, 문법으로는 문법단말 21개과 43개 생성 규칙을 사용하였으며 생성결과의 문장은 2000개로 제한하였다. 또한 단순히 단말어휘만 바뀌는 경우 질의문장 분석에 거의 영향을 주지 않으므로 어휘의 수는 132개로 제한하였다. 태그를 사용하지 않은 기본 문법과 서로 다른 네 가지의 태그셋을 이용한 문법에 의한 생성의 분포를 표 2에서 보이고 있다. 특히 고려할 점은 자연어의 특성 상 입력분포가 고른 것이 이상적임을 의미하는 것은 아니라는 점과 태그셋1에 의한 결과가 유형 1과 2가 큰 비중을 차지하는 실제 사용자 입력분포와 가장 유사한 유사하다는 것이다.

[표 2] 실험문법과 질의문장 생성(문장개수)

문장유형	1	2	3	4	5	6
기본문법	824	215	432	324	192	13
태그셋1	620	543	376	226	121	114
태그셋2	442	468	267	398	212	213
태그셋3	228	326	144	442	476	384
태그셋4	276	296	316	298	366	448

위 실험은 다양한 형식의 질의 문장들을 테스트 케이스로서 자동 생성하는 기능을 수행할 수 있음을 보여줌과 동시에 집중적으로 테스트하고자 원하는 유형을 생성하도록 태그셋을 조정할 수 있음을 보여주고 있다. 이 실험은 언어 생성기를 이용하여 초기의 본질적 응용 분야인 언어 처리를 위한 테스트 케이스를 생성하는 분야에서 실험하였음에 의미가 있다고 본다.

다른 도메인의 실험으로는 UPnP(Universal Plug and Play) 장비의 성능을 테스트하는 연구에서 그 입력 패킷을 자동 생성하기 위하여 태그된 문맥 자유 문법을 기술하고 생성기의 성능을 실험하였다[9]. 특히 구문적 오류가 있는 메시지를 통해 시스템의 성능을 분석하는 실험이다. 정확한 메시지뿐만 아니라 각 세션에서 나타날 수 있는 패킷의 형식에 고의적인 오류를 자동 생성하여 장비의 성능을 테스트 하는 과정에서 사용될 수 있다.

```
<start> ::=
<syn> <bfp> <syn-ack> <ack> <fin-ack> <fin-ack> <ack> |
<syn> <syn-ack> <bfp> <ack> <fin-ack> <fin-ack> <ack> |
<syn> <syn-ack> <ack> <bfp> <fin-ack> <fin-ack> <ack> |
<syn> <syn-ack> <ack> <fin-ack> <bfp> <fin-ack> <ack> |
<syn> <syn-ack> <ack> <fin-ack> <fin-ack> <bfp> <ack> |
<bfp> ::= <bfp1> | <bfp2> | ... | <bfp46>
<syn>::= client tx 0 0 0 0 1 0 0
<syn-ack> ::= server tx 0 1 0 0 1 0 0
.
.
.
<ack> ::= server tx 0 1 0 0 0 0 0
<bfp1> ::= client tx 0 0 0 0 0 0 0
<bfp2> ::= client tx 0 0 0 0 0 1 0
.
.
.
<bfp46> ::= client tx 1 1 1 1 1 1 0
```

[그림 6] 오류포함 메시지 생성 문법 예

그림 6의 단순화한 문법 예에서 생성에 대한 태그 제어 없이는 47^5 가지의 테스트 케이스를 생성하게 될 것이며, 이중 상당 부분은 실제로 의미가 없는 오류 유형에 해당되므로 테스트 케이스 집합 크기 35개부터 47^5 사이로 생성의 범위를 통제할 수 있다는 점이 중요하다. 반면에 이 실험의 범위는 제어되는 테스트 케이스의 생성에 관한 것이므로 장비의 동작성능에 대한 실험은 본 연구의 내용에 관련되지 않으므로 서술하지 않는다.

5. 결론

문법 기반의 테스트 케이스 자동 생성 방법은 프로그래밍 언어의 컴파일러 개발, JAVA 가상 기계의 테스팅, VLSI 시뮬레이터 테스팅, 산업기반 시설인 SCADA(supervisory control and data acquisition) 시스템에 대한 테스팅, 네트워크 프로토콜의 테스팅 등 매우 다양한 소프트웨어 개발 과정에서 활용이 시도되어 왔다. 이론적 배경의 문맥 자유 문법을 넘어서 실질적인 응용에서 자동 생성된 테스트 케이스의 집합이 유효할 수 있도록 태그를 설계하는 것이 본 연구의 목적이다. 본 논문에서 소개한 문맥 자유 문법을 보강하는 다양한 태그들을 통해 유한한 개수의 테스트 케이스의 생성하면서도 테스트의 주요 범위를 포함하는 테스트 케이스를 테스트할 수 있는 방법이다. 특히 자연어 질의 문장을 테스트 케이스로 생성하는 실험과 이진 패킷을 테스트 케이스로 생성하는 실험 등 상이한 종류의 두 소프트웨어를 테스트하는 위한 실험을 통해 폭넓은 활용 가능성을 볼 수 있었다.

향후 계속될 연구로는 각 태그의 지정 값의 변화와 생성되는 테스트 케이스 집합의 크기, 특히 테스트 케이스

의 커버리지(coverage)에 대한 상관 관계를 분석하는 것이며, 또한 다양한 종류의 소프트웨어 시스템에서 테스트 케이스의 자동 생성을 실험함에 따라 추가적인 태그의 설계와 태그 간 충돌에 의한 부작용 여부를 확인하는 것이다.

현재의 실험 대상 외에도 현재 개발되고 있는 많은 소프트웨어들이 문맥 자유 문법과 태그에 의해 기술된 문법과 생성기를 통해 자동으로 많은 테스트 케이스를 얻을 수 있을 것으로 기대된다. 특히 본문에서 소개된 태그들의 값을 다양하게 지정할 수 있음으로써 소프트웨어의 테스터가 생성되는 테스트 케이스의 길이나 복잡도, 또 테스트 케이스 전체 집합의 크기 등을 자유롭게 통제함으로써 실험의 범위를 조정할 수 있는 것이 본 연구의 성과로 평가된다.

참고문헌

[1] 배현섭, “소프트웨어 시험 단계별 자동화 지원 도구”, 정보과학회지, 제23권 제3호, 2005.

[2] 서광익, 최은만, “블랙박스 테스트 케이스의 리엔지니어링”, 정보처리학회 논문지, 제 13-D권, 제4호, p. 39-44, 2006.

[3] 윤성희, “질의어 의미정보와 사용자 피드백을 이용한 웹 검색엔진의 성능 향상”, 한국산학기술학회논문지, 제8권 제2호, p.280-285, 2007.

[4] S. M. Austin, D. R. Wilkins, and B. A. Wichmann. An Ada program test generator. Proceedings of the conference on TRI-Ada 91, p.320-325, 1991.

[5] G. Bernot, M. Gaudel, and B. Marre. Software testing based on formal specifications: a theory and a tool. Software Engineering Journal, 6(6):387-405, 1991.

[6] C. Brown, S. Maghsoodloo, and W. Deason. A cost model for determining the optimal number of software test cases. IEEE Trans. Soft. Eng., 15(2):218-221, 1989.

[7] C. Burgess. The automated generation of test cases for compilers. Journal of Software Testing, Verification, and Reliability, 4(2):81-99, 1994.

[8] C. Burgess and M. Saidi. The automatic generation of test cases for optimizing Fortran compilers. Information and Software Technology, 38(2):111-119, 1996.

[9] D. Hoffman and K. Yoo, A Framework for firewall test automation, In ASE 05: Proc. of Intl. Conf. of Automated software engineering, p.96-104, 2007.

[10] W. Homer and R. Schooler, Independent testing of compiler phases using a test case generator. Software-practice and experience, 19(1):53-62, 1989.

[11] D. Ince. The automatic generation of test data. The Computer Journal, 30(1):63-69, 1989.

[12] Minsik Kim, Jangbok Kim, Kevin Yoo, “Testing UPnP Internet Gateway Devices with Faulty Packets," Proceedings of the 6th WSEAS Int'l Conf. on Circuit, systems, Electronics, Control & Signal Processing, p.29-37, Cairo, Egypt, 2007.

[13] R. Lammel and W. Schulte. Controllable combinatorial coverage in grammar-based testing. In Proc. Test Com, pages 19–38. Springer-Verlag, LNCS 3964, 2006.

[14] P. M. Maurer. Generation test data with enhanced context-free grammars. IEEE Software, 7(4):50-55, 1990.

[15] V. Murali and R. K. Shyamasundar. Sentence generator for a compiler for pt, a pascal subset. Software-Practice and Experience, 13(9):857-869, 1983.

[16] E. G. Sirer and B. N. Bershad. Using production grammars in software in software testing. In 2nd conference on domain-specific languages, p.1-13, ACM Press, 1999.

[17] J. Paakki. Attribute grammar paradigms: a high-level methodology in language implementation. ACM Computing Surveys, 27(2):196-255, 1995.

윤 성 희(Yoon, Sung Hee) **[정회원]**

- 1987 2월 : 서울대학교 컴퓨터공학과 (공학사)
- 1989 2월 : 서울대학교 대학원 컴퓨터공학과 (공학석사)
- 1993 8월 : 서울대학교 대학원 컴퓨터공학과 (공학박사)
- 1994 3월 ~ 현재 : 상명대학교 컴퓨터소프트웨어공학과 교수

<관심분야>
언어정보처리, 정보검색, 소프트웨어 테스팅