제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

함수 호출을 포함한 반복문으로부터 생성된 실행트리 탐색방법

김영국, 고훈준, 유원희
인하대학교 전자계산공학과

e-mail : odin1004@hotmail.com

A Traversal Method of an Execution Tree built from Loops Including Function Calls

Young-Kook Kim, Hoon-Joon Kouh, Weon-Hee Yoo
Dept of Computer Science and Engineering, Inha University

요 약

소프트웨어 개발에서 프로그램의 검증과 확인을 위한 기본적인 방법중에 하나로 디버깅이 있다. 디버깅의 대표적인 기술중에 하나인 알고리즈믹 디버깅에서는 실행트리라는 구조를 사용한다. 실행 트리란 프로그램을 실행하면서 프로그램내의 모든 함수의 호출 관계를 나타낸 트리 구조이다. 본 논문에서는 실행트리에서 반복문안에 함수 호출이 있을 경우에 나타나는 문제점을 지적한다. 실행트리의 문제점은 반복문안에 함수 호출이 있을 경우 실행트리의 노드가 같은 내용이 반복해서 나타난다. 이와 같은 경우에 함수 호출에 대한 반복된 사용자의 응답을 줄일 수 있는 방법을 본 논문에서 제시한다.

1. 서론

현대의 사회에서는 과거와 달리 소프트웨어를 빠르게 개발하는 것이 아니라 오류가 없는 소프트웨어의 개발에 중점을 두고 있다. 따라서 프로그램의 검증과 확인 그리고 유지보수가 차지하는 비중이 점차 커져가고 있다. 소프트웨어 개발에서 프로그램의 검증과 확인을 위한 기본적인 방법은 테스트와 디버깅이다. 프로그램 디버깅은 소프트웨어 개발 작업의 많은 노력과 비용을 차지하고 있다.

대표적인 디버깅 기술중에 알고리즈믹 디버깅 기술은 Sharpiro[1]에 의해 처음으로 제안된 소프트웨어 디버깅 기술이다. 알고리즈믹 디버깅은 프로그램을 실행하고 프로그램의 시작부터 종료까지 모든 함수의 호출관계를 실행 트리(Execute Tree)로 구성하고 이 트리를 하위 레벨에서 상위 레벨로 탐색하면서 프로그램 내에 포함된 논리적 오류를 하향식 방법으로 발견하는 반자동화 기술이다[2]. 이후에 GADT[3], IDT[4], HDT[5]로 확장되었다.

이들 방법에 사용되는 실행 트리는 프로그램의 실행 순서에 따라 루트 노드로부터 하향식으로 구성된다. 이때 호출된 함수는 자식 노드가 되고, 호출한 함수는 부모 노드가 된다. 그리고 임의의 함수에서 여러 개의 함수를 호출한 경우에는 왼쪽에서 오른쪽으로 구성되며 형제 노드가 된다.

프로그램의 크기가 커질수록 실행트리가 비례해서 커지게 된다. 실행 트리를 구성할 경우 문제점이 나타나게 된다. 실제적으로 실행 트리의 문제점에 해결 방안을 목적으로 만들어진 수많은 생각들이 있다. 그중에 실제 구현 방법에서 사용되는 2가지 방법으로 회소 추적(Thin tracing)과 분해 추적(Piecemeal tracing)이 있다[6].

앞에서 설명된 해결방안이 있음에도 불구하고 실행트리에서 반복문안에 함수 호출이 있을 경우에 대한 해결방안에 대한 연구는 아직까지 없었다. 따라서 본 논문은 반복문안에 함수 호출로 생성된 실행트리에서 사용자의 응답 횟수를 줄일 수 있는 방법을 제안한다.

본 논문의 효과적인 전개를 위하여 다음과 같이 전개한다. 2장에서는 관련 연구부분에서 실행 트리와 실행 트리가 가진 문제점을 지적한다. 3장에서는 효율적인 탐색 방법에 대해서 설명한다. 그리고 4장은 3장에 대한 예를 들어 설명한다. 5장은 결론으로 이루어진다.

2. 관련 연구

2.1 실행 트리

실행 트리는 프로그램을 실행하면서 프로그램 내

의 모든 함수의 호출관계를 나타낸 트리 구조로 [정의 1]과 같다.

[정의 1] 프로그램 P에 대한 실행 트리 T는 다음과 같이 정의한다.

T = (N, E, S)

N : 유한 노드의 집합
E : 노드의 관계를 나타내는 순서쌍인 간선의 집합
S : 시작노드

실행 트리는 프로그램의 실행 순서에 따라 루트 노드(프로그램의 메인 함수)로부터 하향식으로 구성된다. 이때 호출된 함수는 자식 노드가 되고, 호출한 함수는 부모 노드가 된다. 그리고 임의의 함수에서 여러 개의 함수를 호출한 경우에는 왼쪽에서부터 오른쪽으로 구성되며 형제 노드가 된다.

트리의 각 노드는 [정의 2]와 같이 함수 이름과 입력과 출력 매개변수의 이름과 값을 통하여 구성된 추적 정보(trace information)를 저장하는 함수로 구성된다.

[정의 2] N에 포함된 노드 n은 다음과 같이 정의한다.

n = (p, in, out)

p : 프로시저(함수)의 이름
in : 입력변수의 집합
out : 출력변수의 집합

이때 i번째의 입력 변수는 순서쌍(x_i, xv_i)로 표현되고, i번째의 출력 변수는 순서쌍 (y_o, yv_o)으로 표현된다. 이때 x_i는 입력변수 이름이고 y_o는 출력변수 이름이다. 그리고 x_i, y_o는 변수 이름이다. 그리고 xv_i, yv_o는 그 변수들의 값이다. 함수 p는 입력 in에 의해 실행되고 출력 out을 반환한다. 따라서 노드가 k개 있다고 가정하면, 노드의 집합 N은 다음과 같다.

$$N = \{ (p_1, in_1, out_1), (p_2, in_2, out_2), \cdots, (p_k, in_k, out_k)\}$$

함수 p_1는 입력 in_1에 의해 실행되고 출력 out_1을 반환하고 함수 p_2는 입력 in_2에 의해 실행되고 출력 out_2를 반환한다. 이런 방법으로 p_k는 입력 in_k에 의해 실행되고, 출력 out_k을 반환한다. 그리고 $0 \le i \le j \le k$ 을 만족하는 (p_i, in_i, out_i)와 (p_j, in_j, out_j)을 각각 n_i와 n_j라고 가정하고 n_i에서 n_j을 호출한다면 n_i는 부모 노드, n_j는 자식 노드가 된다. 그리고 두 노드 사이의 간선은 (n_i, n_j)$\in$ E로 표현할 수 있다. 이것은 노드 n_i로부터 노드 n_j로 잠재적인 제어의 이동과 대응된다. 따라서 실행 트리 T는 시작 노드 S로부터 마지막 단말 노드까지 왼쪽에서 오른쪽으로 함수의 호출 순서에 따라 구성 된다[7].

2.2 문제점

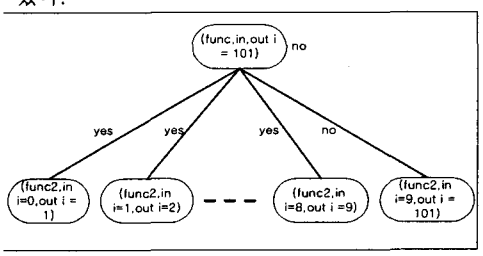
알고리즈믹 디버깅에서 함수 내에 반복문에서 함수를 n번 호출 할 때, 사용자가 호출한 함수에서 n번째에서 오류가 나타날 경우 n번 탐색해야하는 문제점이 있다. [그림 1]의 예제를 이용하여 설명한다.

```
func(){
   i = 0;
   while(i < 10) {
      i = func2(i);
      }
   return i;
}

func2(int i){
    y = 100;
    if (i == 9) i = y++; // i = y--; 가 옳은 구문
    else i++;
    return i;
}
```

[그림 1] 프로그램 예제

[그림 1]의 경우 func함수에서 반복문을 통해서 func2함수를 10번 호출한다. 그러나 9번의 호출동안은 오류가 없는 구문은 else구문만 실행된다. 10번째 호출에서 if문이 실행되고 오류가 발견된다. [그림 1]과 같은 예제를 구성하면 [그림 2]와 같이 구성할 수 있다.



[그림 2] 실행트리

[그림 2]에서 같은 내용에 대한 사용자의 응답을 9번을 'yes'라고 대답한다. 그리고 10번째에서 오류를 찾을 수 있다. 같은 대답을 반복하는 것은 비효율적이다. 만약 [그림 1]의 프로그램에서 함수 호출의 횟수가 n번 일어나면 오류의 내용을 알기 위해서는 n-1번의 'yes'라고 대답을 한다. 이러한 문제점을 해결하는 방법으로 사용자의 응답을 줄이는 방법을 3장에서 제시한다.

3. 실행트리의 효율적인 탐색 방법

사용자의 응답을 줄일 수 있는 방법은 크게 두 가

지로 나뉜다.

- 트리 노드의 같은 내용을 건너뛰는 방법.
- 트리노드 구성 시 같은 내용은 트리 노드 구성에서 제외시키는 방법.

이렇게 두 가지 해결 방법으로 나뉜다. 여기서 논의 할 내용은 우선 첫 번째 실행 트리 노드의 같은 내용을 건너뛰는 방법을 어떻게 논리적으로 실행 트리 노드에서 건너뛰는 방법에 대해서 서술한다. 두 번째로는 실행트리 노드 구성 시 같은 내용은 트리 노드구성에서 제외시키는 방법에 대해서 설명한다.

우선 첫 번째 해결 방법인 실행 트리 노드의 같은 내용을 건너뛰는 방법에 대해서 설명한다. [정의 3], [정의 4] 그리고 flag를 이용해서 나타낸다. flag는 조건문과 반복문의 실행순서를 나타내는 변수이다.

[정의 3] 실행 트리의 노드 N에 포함된 노드 n은
n = (p , in , out , TN)
p : 함수의 이름
in : 입력변수들의 집합
out : 출력변수들의 집합
TN : 실행 테이블의 번호

[정의 4] 실행 테이블(Execute Table)의 구성
(TN, combination, execute)
TN : 실행 테이블 번호
combination : 실행 문장들의 조합의 집합(비트 열)
execute : 실행 횟수

반복문 내에서 함수의 호출 구문을 찾는다. 호출된 함수 안에서 반복문과 조건문이 나타날 경우 다음과 같이 combination을 표현 할 수 있다.

- 반복문이나 조건문이 호출된 함수 내에 있을 경우 flag를 이용하여 반복문이나 조건문의 범위에 차례대로 숫자를 부여한다.
- 반복문이나 조건문이 중첩되어 있지 않을 경우 flag의 숫자를 이용하여 combination값의 하위비트부터 차례로 순서대로 실행비트로 표시한다.
- 조건문과 반복문이 중첩되어 있을 경우 조건문과 반복문의 순서에 따라 flag값에 맞는 combination 값의 실행비트로 표시한다.

combination값의 표시방법에 대한 예제로 [그림 1]을 이용해서 표현하기 위해 조건문에 flag를 표시한 내용은 [그림 3]과 같다.

```
func2(int i){
    y = 100;
    if (i == 9) i = y++; //  flag := 1
    else i++;  //flag :=  2
    return i;
}
```

[그림 3] flag를 표현한 프로그램

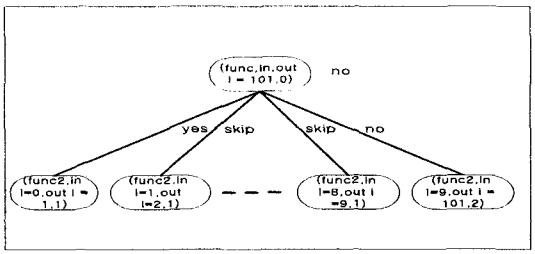
여기서는 조건문이 반복문의 중첩되어 있는 구문이 없으므로 각각의 flag값이 combination값을 조정하는 역할을 한다. 여기서는 임의로 combination을 8bit로 사용하고 실행비트는 1로 표현한다. 따라서 combination값은 00000001, 00000010 이 된다.

combination값은 유일하게 표현된다. 따라서 combination값의 개수만큼 실행테이블에 TN이 만들어 진다. 사용자의 응답이 요구될 때 실행테이블을 확인한다. 실행테이블에서 TN값에 속하는 execute 값이 0보다 크면, TN의 값에 속하는 노드는 사용자의 응답을 건너뛴다. 반대의 경우 사용자의 응답을 묻고 TN의 값에 해당하는 execute값에 1을 더한 후 다음 노드로 이동한다. 두 번째 실행 트리 노드 구성 시 같은 내용은 트리 노드구성에서 제외시키는 방법을 설명한다. 여기서도 [정의 3]과 [정의 4]를 사용한다. 따라서 앞의 내용과 같이 실행 테이블을 사용한다.

반복문에서 함수의 호출 구문을 찾는다. 호출된 함수에서 반복문과 조건문이 나타날 경우 앞에서 사용한 방법을 통해 combination값을 알 수 있다. 이렇게 표현된 combination값은 유일하게 표현된다. combination값의 개수만큼 실행테이블에 TN이 만들어 진다. 이렇게 만들어진 TN값을 비교하여 트리 노드의 생성 시 TN값이 같을 경우 노드를 생성하지 않고 execute의 횟수를 증가 시킨다. 반면에 TN값이 일치하지 않을 경우 새로운 노드를 생성하고 실행테이블에 내용을 반영한다.

4. 실행트리의 효율적인 탐색 방법에 대한 예제

[그림 1]에 대한 실행트리를 구성해보자. 첫 번째로 트리 노드의 같은 내용을 건너뛰는 방법에 대해서 설명한다.



[그림 4] 실행트리에 실행테이블을 적용한 트리

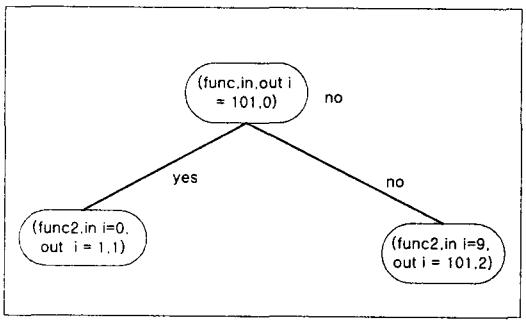
[그림 2]에서 표현한 flag를 실행테이블 구성에서 이용한다. flag는 독립적이므로 combination값에서 실행을 표현하는 값을 1이라고 하면, 1의 값은 각각의 combination에서 1번만 존재한다. combination을 이용하여 실행테이블을 구성하면 [표 1]과 같다. 여기서 실행되는 내용에 따라 TN값을 각각의 노드에 부여한다.

[표 1] 노드 구성에 사용되는 실행테이블 구성

TN	combination	execute
1	000001	0
2	000010	0

이렇게 구성된 실행트리에서 탐색하는 방법은 다음과 같다. 같은 내용을 건너뛰기 위해서 실행 테이블에서 execute값을 비교한다. execute값이 0보다 크면 execute값을 1증가 시킨 후 사용자의 응답을 묻지 않고 건너뛴다. 반대로 execute값이 0이라면 execute값을 1증가 시키고 사용자의 응답을 묻는다.

두 번째로 트리노드 구성 시 같은 내용은 트리 노드 구성에서 제외시키는 방법에 대해서 설명한다.



[그림 5] 노드 구성에서 제외하는 실행 트리

[그림 2]에서 표현한 프로그램을 실행 테이블을 구성하면서 노드를 생성한다. 구성 노드를 생성할 때 실행테이블의 내용은 [표 2]와 같다. 그리고 실행 트리에서 노드 생성을 위해서는 실행테이블의 combination값을 비교하고 같은 경우 노드를 생성하지 않고, execute값을 1씩 증가 시킨다. 실행테이블의 combination값이 없을 경우 노드는 새로 생성되고 새로운 TN값을 부여 받는다. 그리고 execute값을 0으로 초기화 시켜준다.

[표 2] 노드 구성에서 제외하는 실행테이블 구성

TN	combination	execute
1	000001	9
2	000010	1

5. 결론

실행 트리에서 반복문안에서 함수 호출은 비효율적으로 탐색되어 왔다. 여기서는 함수 호출에서 같은 내용에 대해서 사용자의 응답을 요구하지 않고 건너뛰거나 실행 트리 구성 시 같은 노드를 제거하는 방법을 사용했다. 그 결과 사용자에게 응답을 요구하는 실제 횟수가 상당히 많이 감소하게 되었다. 그러나 여기서 두 가지 제시한 방법 중에 실제 사용시에는 같은 내용에 대해서 건너뛰는 방법이 실제 사용자가 쉽게 사용할 수 있었다. 같은 노드에 대한 내용을 트리 구성 시 삭제하면 실행테이블에서 실제 노드의 실행횟수는 알려주지만 프로그램 이해도가 떨어지게 되었다.

참고문헌

[1] E. Sapriro, "Algorithmic Program Debugging," MIT Press, Cambridge, May ,1983

[2] N. Shamehri, M. Kamkar, and P. Fritzson, "Semi-automatic Bug Localization in Software Maintenance," Proceedings of the IEEE Conf. on Software Maintenance, CSM-90,pp30-36, San Diego, November, 1990

[3] P. Fritzson, N. Shahmhrl, and M. Kamkar, "General Algorithmic Debugging and Testing," ACM Programming Language and Systems, December 1992.

[4] Z. Alexin, T. Gyim'othy, G. Kokai,"IDT: Integrated System for Debugging and Testing Prolog Programs," Proceedings of the Fourth Symposium on Programming Languages and Software Tools, pp.312-323, Hungary, June 9-10,1995.

[5] H. J. Kouh, K. T. Kim, W. H. Yoo, "Hybrid Debugging Method: Algorithmic + Step-wise Debugging,",The 2002 International conference on Software Engineering Research and Practice(SERP'02), pp. 342-347, 2002. 6

[6] H. Nilsson and P. Frizson,"Lazy Algorithmic Debugging: Ideas for Practical Implementation," AADEBUG'93, the first international workshop on Automated and Algorithmic Debugging, LNCS 749, Springer-Verlag, 1993

[7] H. J. Kouh, W. H. Yoo, "The Efficient Debugging System for Locating Logical Errors in Java Programs", Computational Science and Its Application - ICCSA 2003, Montreal, Canada, Springer-Verlag, LNCS2667, pp. 684-693, 2003. 5