다중 기준변수를 사용한 동적 프로그램 슬라이싱 알고리즘의 효율성 비교

박 순 형† · 박 만 곤††

요 약

프로그램에서 오류가 발견되었을 때 프로그래머는 어떤 시험 사례(test case)를 통해 프로그램을 분석한다. 이처럼 현재 입력 값에 영향을 끼치는 모든 명령문들에 관련된 동적 슬라이싱(dynamic slicing)과 이를 구현하는 기술은 실제 테스팅 및 디버깅 분야에서 매우 중요하다고 할 것이다. 지금까지의 동적 슬라이싱은 슬라이싱 기준 변수가 1 개 일 때의 경우에 대해서만 연구해 왔다. 그러나, 실제적인 테스팅 및 디버깅 분야에서는 슬라이싱 기준이 되는 변수가 2 개 이상인 경우가 아주 많이 발생한다. 따라서 슬라이싱 기준 변수가 n 개일 때 각 변수 별로 슬라이싱을 구하는 다중기준변수를 사용한 프로그램 슬라이싱을 구현하는 알고리즘 개발 및 구현은 매우 필요하다. 본 논문에서는 슬라이싱 기준 변수가 n개 일 때 동적 프로그램 슬라이스(dynamic program slices)를 만드는 알고리즘을 제시하였고 프로그래밍 언어를 사용하여 동적 프로그램 슬라이싱 알고리즘을 프로그래밍한 뒤 예제 프로그램을 적용시켜 구현하였다. 구현 결과는 실행 이력에 대한 마킹 테이블(marking table)과 동적 종속 그래프로 나타내었다. 그리고, 본 논문에서 제시한 다중기준변수 동적 슬라이스 생성을 위한 마킹 알고리즘이 기존의 단일기준변수 기법 보다 실제적인 테스팅 환경에서 더 우수함을 보였다.

On the Efficiency Comparison of Dynamic Program Slicing Algorithm using Multiple Criteria Variables

Soon-Hyung Park† · Man-Gon Park††

ABSTRACT

Software engineers are used to analyse the error behavior of computer programs using test cases which are collected for the testing phase when software errors are detected. In actual software testing and debugging, it is important to adopt dynamic slicing technique which is concerned on all the statements to be affected by the variables of current inputs and to use technique of its implementation. The traditional dynamic slicing has focused on the single slicing criterion algorithm. It has been thought that it is needed to develope and implement algorithm for used multiple criteria variables program slicing, which finds every slicing criterion variable where it is used multiple criteria variables. In this paper, we propose an efficient algorithm to make dynamic program slices when it has used multiple criteria variables. The results of the implementation are presented by the marking table on execution history and the dynamic dependence graph. Also we can find that the proposed dynamic program slicing approach using multiple criteria variables is more efficient than the traditional single case algorithm on the practical testing environment.

1. 서 론

프로그램 슬라이싱은 프로그램 디버깅 처리에서 해당 프로그램에 관련된 부분들에 대해 관련 범위를 좁혀주기 위한 효과적인 기술이다. 즉, 어떤 포인터 p에서 변수 var의 값에 직·간접적으로 영향을 끼치는 프로그램 P에 있는 모든 명령문들을 찾는 것이다[3, 6, 11, 12]. 프로그램 슬라이싱의 기본적인 기법으로 정적 슬

† 정 회 원 : 동의공업대학 전자계산과 교수
†† 종신회원 : 부경대학교 컴퓨터멀티미디어공학부 교수
논문접수 : 1999년 2월 18일, 심사완료 : 1999년 7월 20일

라이스(static slice) 기법과 동적 슬라이스(dynamic slice) 기법이 있다.

정적 슬라이스 기법은 프로그램이 디버깅되는 조건에서 주어진 실행보다 오히려 주어진 프로그램의 변수 값에 영향을 주는 모든 가능한 실행들에 대해 구축되어지는데 비해 동적 슬라이스 기법은 그 프로그램에 의해 취해진 경로를 정확히 측정함으로써 프로그램의 변수 값에 실제적으로 영향을 주는 실행들에 대해 구축되어진다[1, 4, 10]. 정적 슬라이싱은 동적 슬라이싱에 비해 슬라이스의 크기가 크고 슬라이스의 정확성이 떨어지므로 본 논문에서는 동적 슬라이스를 기반으로 한다.

동적 슬라이스를 산출하기 위한 몇 가지의 기법들이 제안되어졌으나[2, 5, 8, 9] 지금까지의 동적 슬라이싱에 대한 연구는 주로 슬라이싱 기준이 1개 일 때의 경우에 대해서만 연구해 왔다. 실제적인 테스팅 및 디버깅 분야에서는 슬라이싱 기준이 되는 변수가 2개 이상인 경우가 많이 발생한다.

만일 n개의 기준변수를 전체를 슬라이싱 한다면 각각의 슬라이싱 기준에 동시에 종속되는 명령문이 존재하게 된다. 그러므로 슬라이싱 운행경로를 줄일 수 있어 각 기준 변수 별로 슬라이싱을 구하는 경우에 비해 훨씬 효율적인 슬라이싱을 할 수 있다.

따라서, 슬라이싱 기준 변수가 n개일 때 기준 변수 1개 씩 슬라이싱을 구한다면, 실제로 프로그램 전체의 디버깅 차원에서 보면 이에 소요되는 시간과 노력은 n 배 보다 더 많이 필요하게 된다. 그러므로 다중 기준변수를 적용할 수 있는 프로그램 슬라이싱을 구현하는 알고리즘 개발 및 구현에 대한 연구는 매우 필요하다 할 것이다. 본 논문에서는 슬라이스의 기준 변수가 n개일 때 효과적인 동적 슬라이스를 구하는 마킹 알고리즘을 제시하였고, 실제 구현한 결과를 보였다. 2장에서는 기존 동적 슬라이싱 기법에 대해 설명하였으며 3장에는 슬라이싱 기준 변수가 n개일 때 효과적인 동적 프로그램 슬라이싱 알고리즘을 제시하였고 프로그래밍 언어를 사용하여 실제 구현한 결과를 마킹 테이블(marking table)을 통해 나타내었다. 4장에서는 본 논문에서 제시한 다중 기준변수 동적 슬라이스 마킹 알고리즘이 기존의 알고리즘과 단일 기준변수 동적 슬라이싱 알고리즘에 비해 효율적임을 비교 사례를 통해 비교 분석을 하였다.

2. 동적 프로그램 슬라이싱 기법

동적 슬라이스는 동일한 프로그램의 입력에 대해 어떤 실행위치 q에서 관심 변수에 관련된 원래의 프로그램과 그것의 결과가 일치하는 프로그램 실행의 부분이다. 프로그램 입력 x에 대해 실행된 프로그램 P의 동적 슬라이싱 기준은 C = (x, I^q, v)이다. I^q는 실행 이력 H에서 position q의 명령문이다. v는 I^q에 있는 변수이다[7]. 슬라이싱 기준 C에 대한 프로그램 P의 동적 슬라이스는 0 개 이상의 명령문을 삭제함으로써 P로 부터 얻어지는 정확하고 실행 가능한 프로그램 P′이다. 그리고 프로그램이 수행되어졌을 때 입력 x는 H_x 에 있는 I^q의 v 값이 H'_x에서 $I^{q'}$의 v 값과 같은 경우일 때 대응하는 실행 위치 q′가 존재하는 실행이력 H'_x를 산출한다[4].

실행 이력이란 주어진 시험사례를 실행하는 동안 방문된 순서에 의한 일련의 정점들인 $\{v_1, v_2, \ldots\ldots, v_n\}$의 집합이다. H_x 내에 있는 position p에서 노드 Y(즉, Hx(p)=Y)는 Yp로 표현된다. 이와 같이 상위 첨자를 사용함으로서 실행 이력에서 같은 노드의 다중 발생을 구별할 수 있다.

2.1 Korel & Laski 기법

하나의 슬라이싱 기준에 대한 동적 슬라이스의 계산을 위한 알고리즘이 (그림 1)에 나타나 있다. 처리순서는 다음과 같다.

- 이 알고리즘에서 모든 동작들은 맨 처음 마크되지 않고 방문되지 않은 상태에서 초기화된 후 I^q의 가장 최근 정의된 노드를 찾아 마크한다.
- while-loop의 각 반복에 대해 마크만 되고 방문되지 않는 하나의 동작 X^*는 선택된다. 그리고 방문되어질 때 set 한다 (line 5와 6 참조).
- X^*에서 사용된 모든 변수들의 모든 마지막 정의는 확인된 후 마크된다(line 7). 이 단계는 동작들 사이에서 자료 종속(Z^t)을 발견하는데 대응한다.
- Z^t 와 X^*사이에서 하나의 제어종속이 존재하는 모든 동작들은 마크된다(line 8).
- T_x에 있는 노드 X의 모든 동작들은 마크된다(line 9).
- while-loop에 의해 모든 마크된 동작들이 전부 방문될 때까지 반복된다[4].

1 입력 x에 대한 프로그램 P를 실행하여 위치 q까지의 실행 이력 T_x를 기록한다.
2 T_x 에 있는 모든 동작들을 마크되지 않고 방문되지 않은 상태로 초기화시킨다.
3 P의 가장 최근 정의 Y^n를 찾은 후 Y^n 를 마크한다.
4 **while** T_x에 마크는 되었으나 방문되지 않는 동작이 존재한다.
do
5 T_x 에서 마크는 되었으나 방문되지 않는 한 개의 동작 X^k를 선택한다.
6 X^k는 방문되었음을 표시한다.
7 $v \in U(X^k)$를 만족하는 모든 변수들에 대해 v의 최근 정의(last definition) Y^m를 찾은 후 마크한다.
8 Z 와 X^k 사이에서 하나의 제어 종속이 존재하는 그러한 모든 동작 Z를 마크한다.
9 노드 X에 대한 모든 동작들을 마크한다.
10 **end_while**
11 T_x에 있는 마크되지 않는 모든 동작들의 노드를 제거한 후 P로부터 구축된 동적 슬라이스를 보여준다.

(그림 1) Korel & Laski 동적 슬라이싱 알고리즘

2.2.2 Agrawal & Horgan 기법

주어진 실행이력에 대한 변수에 관해 동적 슬라이스를 구하기 위해 Agrawal 과 Horgan이 제안한 세 가지 기법을 동적 종속 그래프를 통해 설명한다[2].

(1) 기법 1

ㄱ) 그래프에서 모든 노드들은 시작 시점에서 점선으로 그린다.

ㄴ) 명령문들이 실행됨에 따라 발생하는 새로운 종속에 대응하는 간선들을 실선으로 만든다.

ㄷ) 실선으로 표시된 간선을 따라 운행하고 도달된 노드들을 굵은 실선으로 만든다.

(2) 기법 2

실행 이력에서 어떤 한 명령문의 각 발생들에 대해서 해당 명령문에 종속성 간선을 가진 단 한 개의 명령문만을 새로운 노드로 만든다.

(3) 기법 3

실행 이력에서 명령문의 모든 발생에 대해 새로운 노드를 만드는 대신 동일한 종속성을 가진 또 다른 노드가 존재하지 않을 경우에 한해서 새로운 노드를 만든다.

(4) 적 용

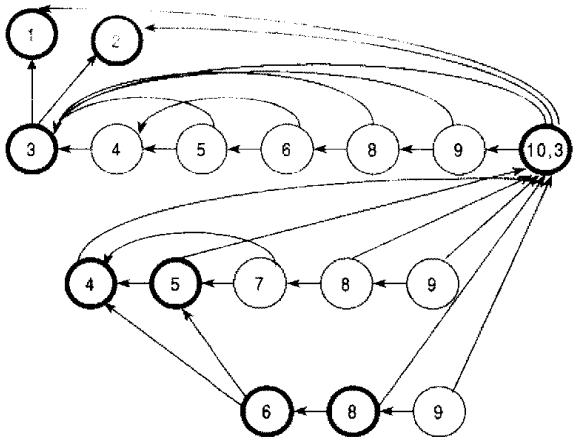
(그림 2)의 예제 프로그램 1에 시험 사례로서 N = 3 이고 X = {-4, 3, -2} 일 때 변수 Z에 대한 동적 슬라이스를 산출하기 위해 기법 2를 적용시킨다. 이 때, 실행 이력은 {1^1, 2^1, 3^1, 4^1, 5^1, 6^1, 8^1, 9^1, 10^1, 3^2, 4^2, 5^2, 7^1, 8^2, 9^2, 10^2, 3^3, 4^3, 5^3, 6^2, 8^3, 9^3, 10^3, 3^4}이 된다. 위의 기법들을 적용시킨 결과는 (그림 3)에 동적종속그래프(DDG : Dynamic Dependence Graph)로 나타내었으며, 굵은 원은 슬라이스된 노드들이다.

3. 다중 기준변수 동적 프로그램 슬라이싱 기법

다중 기준변수의 동적 슬라이스는 동일한 프로그램의 입력에 대해 어떤 실행위치 q에서 관심 변수들에 관련된 원래의 프로그램과 그것의 결과가 일치하는 프로그램의 실행의 부분이다. 프로그램 입력 x에 대해 실행된 프로그램 P의 다중 기준 동적 슬라이싱의 기준은 MC = (x, I, V)이다. I는 실행 이력 H에서 기준변수들의 포인터에 있는 명령문들의 집합이다. V는 I에 있는 기준변수들의 집합이다[11]. 슬라이싱 기준 MC에 대한 프로그램 P의 동적 슬라이스는 0 개 이상의 명령문을 삭제함으로써 P로 부터 얻어지는 정확하고 실행가능한 프로그램 P'이다.

```
        begin
S1:         read(N)
S2:         I := 1;
S3          while (I <= N)
            do
S4:             read(X)
S5:             if (X < 0)
                then
S6                  Y := f1(X);
                else
S7:                 Y := f2(X);
                end_if;
S8:             Z := f3(Y);
S9:             write(Z);
S10:            I := I + 1;
            end_while;
        end.
```

(그림 2) 예제 프로그램 1



(그림 3) 예제 프로그램 1의 동적 종속 그래프

본 논문에서는 슬라이싱 기준 변수가 n개일 때 효율적인 동적 프로그램 슬라이스를 산출하기 위한 마킹 알고리즘을 제시하고 이 알고리즘을 프로그래밍 언어를 통해 실제 구현하였다. 그리고 그 결과를 실행 이력에 대한 마킹 테이블(marking table)로 나타내었다. 동적 프로그램 슬라이싱을 구하기 위해서는 실행 이력과 n개의 테스트할 변수 명들이 먼저 주어진다.

3.1 표기법

프로그램을 구성하는 명령문의 구조는 다음과 같이 표현할 수 있다.

Program → *Declarations* **begin** *Stmt_list* **end**
Stmt_list → *Stmt*; *Stmt_list*
Stmt → *Simple_stmt* | *If_stmt* | *While_stmt*
Simple_stmt → *Assgn_stmt* | *Read_stmt* | *Write_stmt*
Assgn_stmt → *Var* := *Exp*
Read_stmt → **read**(*Var*)
Write_stmt → **write**(*Var*)
If_stmt → **if** (*Predicate_exp*) **then** *Stmt_list* **else** *Stmt_list* **end_if**
While_stmt → **while** (*Predicate_exp*) **do** *Stmt_list* **end_while**
Predicate_exp → *Exp*
Exp → *Exp Binary_op Exp* | *Unary_op Exp* | *Var* | *Const*

3.2 입력 자료

동적 프로그램 슬라이싱 알고리즘을 구현하기 위해 세 개의 입력 자료가 필요하다 - 실행 이력 자료, 관련 자료, 기준 변수 자료.

실행 이력 자료란 주어진 시험 사례에서 실행되는 동안 관련되는 노드들을 실행 순서대로 나열을 한 것을 말하며, 노드 관련 자료란 각 명령문에 해당되는 노드들의 구성 요소를 저장해 놓은 자료를 말하며 노드 번호, 노드 *type*, *DEF*, 그리고 *REF*로 구성된다. 노드 *type*에는 입력, 서술, 판단, 반복, 출력 등이 있으며, *DEF*(*n*)은 노드 *n*에서 바뀌어진 값을 가진 변수의 집합이고, *REF*(*n*)은 노드 *n*에서 사용된 값을 가진 변수의 집합이다. 그리고, 기준 변수 자료란 슬라이싱의 기준 변수의 위치와 변수명들에 관한 자료이다.

3.3 마킹 알고리즘

본 논문에서 제시한 다중 기준변수를 사용한 동적 프로그램 슬라이스(*MarkTbl*)를 구하는 마킹 알고리즘은 다음과 같다.

```
begin
  read HistoryFile(Iorder, HistorySize)
  read RelatedFile(NodeType, Def, Ref)
  read CriterionFile(InitBaseNo, InitBaseVar)
  Icnt = 1
  while (Icnt not > HistorySize) do
    I = Iorder(Icnt)
    if (i = InitBaseNo) then
      AddBaseTbl(BaseValTbl, InitBaseVar)
    end_if
    if (NodeType(i) = 'Input') then
      ProcInput (Def(i), BaseVarTbl,
                 MarkTbl(Icnt))
    end_if
    if (NodeType(i) = 'Assign' ∩
      Ref(i) = blank) then
      ProcInput (Def(i), BaseVarTbl,
                 MarkTbl(Icnt))
    end_if
    if (NodeType(i) = 'Assign' ∩
      Ref(i) not = blank) then
      ProcAssign (Def(i), Ref(i), BaseVarTbl,
                  MarkTbl(Icnt))
    end_if
    if (NodeType(i) = 'Repeat') then
      ProcessRepeat
    end_if
    if (NodeType(i) = 'Decision') then
```

```
        ProcessDecision
      end_if
      Icnt = Icnt + 1
   end_while
end.

ProcInput (Def(i), BaseVarTbl, MarkTbl(Icnt))
begin
   if (Def(i) ∈ BaseVarTbl) then
      DelBaseVarTbl(BaseVarTbl, Def(i))
      MarkTbl(Icnt) = 'O'
   end_if
end.

ProcAssign (Def(i), Ref(i), BaseVarTbl,
            MarkTbl(Icnt))
begin
   if (Def(i) ∈BaseVarTbl) then
      DelBaseVarTbl(BaseVarTbl, Def(i))
      MarkTbl(Icnt) = 'O'
      AddBaseVarTbl(BaseVarTbl, Ref(i))
   end_if
end.

ProcRepeat (Ref(i), BaseVarTbl, MarkTbl(Icnt))
begin
   MarkTbl(Icnt) = 'O'
   AddBaseVarTbl(BaseVarTbl, Ref(i))
end.

ProcessDecision (TrueTbl, FalseTbl, Ref(i),
                 BaseVarTbl, MarkTbl(Icnt))
begin
   if (TrueTbl ∈ MarkTbl ∩
      FalseTbl ∈ MarkTbl) then
      MarkTbl(Icnt) = 'O'
      AddBaseVarTbl(BaseVarTbl, Ref(i))
   end_if

AddBaseVarTbl(BaseVarTbl, Var)
end.
begin
   BaseVarTbl = BaseVarTbl + Var
end.

DelBaseVarTbl(BaseVarTbl, Var)
begin
   BaseVarTbl = BaseVarTbl - Var
end.
```

4. 다중 기준변수 동적 프로그램 슬라이싱 기법의 적용 및 비교

4.1 적용 예제

(그림 4)의 예제 프로그램 2에 다중 기준변수 동적 프로그램 슬라이싱 알고리즘을 적용시킬 수 있다. N=3이고 X={-2, 4, -3} 일 때 실행이력은 {1^1, 2^1, 3^1, 4^1, 5^1, 6^1, 7^1, 8^1, 9^1, 10^1, 13^1, 14^1, 15^1, 16^1, 17^1, 18^1, 6^2, 7^2, 8^2, 11^1, 12^1, 13^2, 14^2, 15^2, 16^2, 17^2, 18^2, 6^3, 7^3, 8^3, 9^2, 10^2, 13^3, 14^3, 15^3, 16^3, 17^3, 18^3, 6^4}이다. 15^3의 Q, 16^3의 U 와 17^3의 Z에 대한 프로그램 슬라이스를 구하는 경우를 본 논문에서 제안한 다중 기준변수 동적 슬라이싱 기법을 사용하여 적용시켜 보았다. 노드 관련 자료는 <표 1>에 나타나 있다.

〈표 1〉 예제 프로그램 2의 노드 관련 자료

노드 번호	TYPE	DEF	REF
1	입력	N	
2	서술	I	
3	서술	T	
4	서술	Y	
5	서술	Z	
6	반복		I , N
7	입력	X	
8	판단		X
9	서술	Y	X
10	서술	Z	Y
11	서술	Y	X
12	서술	Z	Y
13	서술	Q	Y , T
14	서술	U	Y , N
15	출력		Q
16	출력		U
17	출력		Z
18	서술	I	I

4.2 비교 사례 1

(그림 4)의 예제 프로그램 2에 시험 사례로서 N=3이고 X={-2, 4, 3}일 때 변수 Q, U 와 Z에 대한 프로그램 슬라이스를 구하는 경우를 Agrawal & Horgan 기법과 본 논문에서 제안한 다중 기준변수 동적 슬라이스 마킹 기법을 사용하여 서로 비교 해 보았다.

Agrawal & Horgan 기법을 사용하여 변수 Q ,U와 Z에 대한 프로그램 슬라이스를 구하는 경우 그 결과가 (그림 5)에 나타나 있고, 본 논문에서 제안한 다중변수 기법을 사용한 결과가 (그림 6)에 나타나 있다.

그리고, Agrawal & Horgan 기법과 제안 기법에 대한 마킹 비교 결과표가 <표 2>에 나타나 있다. <표 2>의 결과에서 보면 13개의 명령문으로 산출 결과가 나온 본 논문에서 제안한 기법이 16개의 명령문으로 산출 결과가 나온 Agrawal & Horgan 기법 보다 더 효율적임을 알 수 있다.

4.3 비교 사례 2

3개의 기준변수에 대한 각각의 마킹된 운행노드와 마킹 노드 개수는 다음과 같다.

기준노드	마킹된 운행 노드	마킹노드 개수
Q	33, 31, 29, 28, 27, 17, 16, 6, 3, 2, 1	11
U	34, 31, 29, 28, 27, 17, 16, 6, 3, 1	10
Z	32, 31, 29, 28, 27, 17, 16, 6, 3, 1	10
	합 계	31

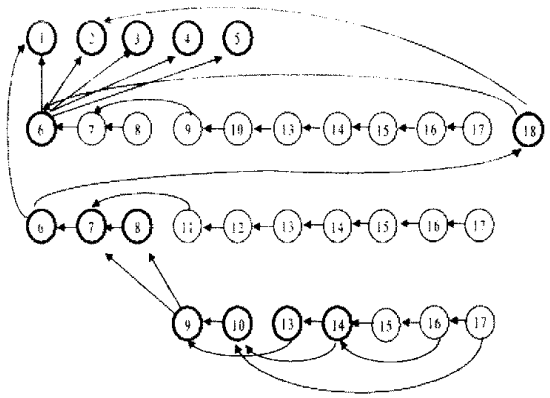
본 논문에서 제안한 알고리즘을 3개의 기준변수에 적용한 기준변수테이블의 변수에 대한 실제 운행 노드 상에서 중복 분석 그래프는 (그림 7)과 같다. (그림 7)의 중복 분석 그래프에서 보는 바와 같이 Q, U, Z가 동시에 실행한다면 마킹 노드 개수는 다음과 같다.

기준노드	마킹된 운행 노드	마킹노드 개수
Q, U, Z	34, 33, 32, 31, 29, 28, 27, 17, 16, 6, 3, 2, 1	13

따라서, Q, U, Z가 동시에 실행한다면 각각 실행할 때보다 마킹 노드의 개수가 약 42%정도 줄어들게 되어 2.4배의 효율성이 높아진다는 것을 알 수 있다.

```
        begin
S1:       read(N);
S2:       I := 1;
S3:       T := 1;
S4:       Y := 0;
S5:       Z := 0;
S6:       while (I <= N)
          do
S7:           read(X);
S8:           if (X < 0)
              then
S9:               Y := f1(X);
S10               Z := f2(Y);
              else
S11:              Y := f3(X);
S12:              Z := f4(Y);
              end_if;
S13:          Q := f5(Y,T);
S14:          U := f5(Y,N);
S15:          write(Q);
S16:          write(U);
S17:          write(Z);
S18:          I := I + 1;
          end_while;
        end.
```

(그림 4) 예제 프로그램 2



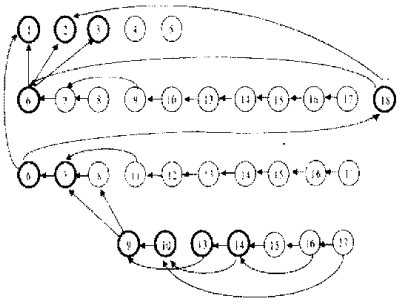
(그림 5) 예제 프로그램 2의 변수 Q, U, Z에 대한 Agrawal & Horgan의 동적 종속 그래프

〈표 2〉 예제 프로그램 2의 동적 슬라이싱 결과인 마킹테이블 비교

순 번	실행이력	다중기준변수를 사용한 Agrawal & Horgan's 기법 (Q, U, Z)	제안된 다중기준변수를 사용한 동적 슬라이싱 기법 (Q ,U, Z)
1	1	✓	✓
2	2	✓	✓
3	3	✓	✓
4	4	✓	
5	5	✓	
6	6	✓	✓
7	7		
8	8		
9	9		
10	10		
11	13		
12	14		
13	15		
14	16		
15	17		
16	18	✓	✓
17	6	✓	✓
18	7		
19	8		
20	11		
21	12		
22	13		
23	14		
24	15		
25	16		
26	17		
27	18	✓	✓
28	6	✓	✓
29	7	✓	✓
30	8	✓	

〈표 2〉 계속

순 번	실행이력	다중기준변수를 사용한 Agrawal & Horgan's 기법 (Q, U, Z)	제안된 다중기준변수를 사용한 동적 슬라이싱 기법 (Q ,U, Z)
31	9	✓	✓
32	10	✓	✓
33	13	✓	✓
34	14	✓	✓
35	15		
36	16		
37	17		
38	18		
39	6		
노드수		16	13



(그림 6) 예제 프로그램 2의 변수 Q, U, Z에 대한 제안 동적 종속 그래프

4.4 비교 분석

4.4.1 알고리즘에 따른 상대효율성 측도1

- 상대효율성 측도1 (RER1)

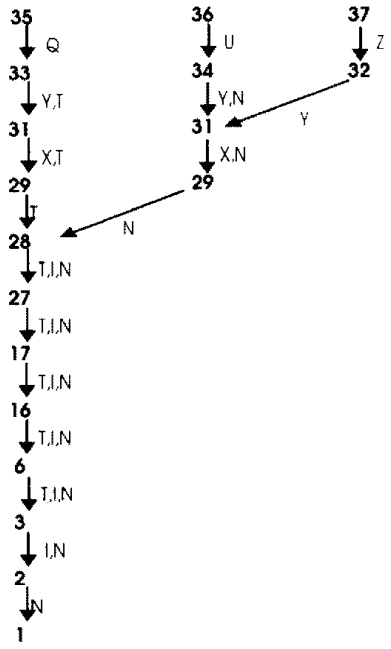
$$= \frac{\text{제안알고리즘의 평균운행노드 개수}}{\text{기존알고리즘의 평균운행노드 개수}}$$

- 평균운행노드 개수(NVTN)

$$= \frac{\sum_{i=1}^{n} NTN(i)}{n}$$

- NTN(Number of Travering nodes)
  = 운행노드 개수

알고리즘 \ 기준변수 개수		1	2	3
기존알고리즘	NVTN	12.67	14.33	16
	RER1	1.0	1.0	1.0
제안알고리즘	NVTN	10.33	11.67	13
	RER1	0.815	0.814	0.813



(그림 7) 기준변수를 이용한 중복 분석 그래프

4.4.2 기준변수 개수에 따른 상대효율성 측도2

- 상대효율성 측도2 (RER2)

$$= \frac{n-case\text{의 평균운행노드의 합}}{1-case\text{의 평균운행노드의 합}}$$

- 평균운행노드의 합(SVTN)
  = 평균운행노드개수 × 실행횟수

알고리즘 \ 기준변수개수		1	2	3
기존알고리즘	SVTN	38.01	28.66	16
	RER2	1.0	0.754	0.421
제안알고리즘	SVTN	30.99	23.34	13
	RER2	1.0	0.753	0.419

5. 결 론

동적 슬라이스는 주어진 프로그램 입력에 대해 발생된 변수 값에 실제적으로 영향을 주는 명령문들로 구성되어 있다. 그러므로 어떤 시험 사례를 통해 프로그램을 분석하는 디버깅 분야에서는 동적 슬라이싱이 정적 슬라이싱 보다 더 유용하게 사용될 수 있다. 그러나 지금까지의 동적 슬라이싱은 슬라이싱 기준이 1개일 때의 경우에 대해서만 주로 연구해 왔다. 실제적인 테스팅 및 디버깅 분야에서는 슬라이싱 기준이 되는

변수가 2개 이상인 경우가 아주 많이 발생한다. 따라서 슬라이싱 기준 변수가 n개일 때 각 변수별로 슬라이싱을 구하는 다중 기준 프로그램 슬라이싱을 구현하는 알고리즘 개발 및 구현은 매우 필요하다고 할 것이다. 본 논문에서는 슬라이싱 기준 변수가 n개일 때 동적 프로그램 슬라이스를 만드는 마킹 알고리즘을 제시하였고 프로그래밍 언어를 사용하여 동적 프로그램 슬라이스 마킹 알고리즘을 프로그래밍 한 뒤 예제 프로그램을 적용시켜 구현하였다. 구현 결과는 실행 이력에 대한 마킹 테이블과 동적 종속 그래프로 나타내었다. 그리고, 본 논문에서 제시한 다중 기준변수 동적 슬라이스 생성을 위한 마킹 알고리즘과 동적 종속 그래프를 기존의 Agrawal & Horgan 기법과 상대효율성 측도 비교 테이블을 통해 비교 분석한 결과 슬라이스 마킹 변수들이 약 81%로 줄어들었음을 알 수 있었다. 그리고, 기준변수가 1개일 경우에 대해 기준변수가 2개인 경우와 3개인 경우를 비교 분석한 결과 슬라이스 마킹 변수들이 각각 약 75%와 약 42%로 줄어들었음을 알 수 있었다. 그리고, 기존 알고리즘과 제안된 알고리즘과 비교했을 때 슬라이스 마킹 변수들이 최대 약 34%로 줄어들었음을 알 수 있었다. 이것으로 미루어 기준 변수의 수가 많으면 많을수록 다중 기준변수 슬라이싱 알고리즘의 효과는 더 커지게 됨을 알 수 있었다. 그리고, 경험있는 소프트웨어 엔지니어들은 적당히 복수개의 기준변수를 선택하는 것이 실제적으로 효율적인 소프트웨어 테스팅 작업을 수행할 수 있기에 기준 변수의 수를 여러 개 선택하는 제안된 슬라이스 알고리즘의 효과가 더 크다는 것도 알 수 있었다.

향후 연구 과제는 원시 프로그램을 분석하여 실행 이력 자료를 실제 생성시키는 프로그램과 노드 관련 자료를 자동 생성시키는 프로그램을 개발하는 것이다. 이렇게 함으로써 본 논문에서 제시한 다중 기준변수 동적 프로그램 슬라이스 산출 프로그램과의 연계를 통해 완벽한 구현을 할 수 있다.

참 고 문 헌

[1] B. Korel, "Computation of Dynamic Program Slices for Unstructured Programs," IEEE Trans. on Software Engineering, Vol.23, No.1, pp.17-34, January, 1997.

[2] Jeanne Ferrante, Karl J. Ottentein, and Joe D. Warren. "The program dependence graph and its uses in optimization." ACM Trans. on Programming Languages and Systems, pp.319-349, July, 1987.

[3] Karl J. Ottentein and Linda M. Ottentein. "The program dependence graph in a software development environment.", Proc. of the ACM SIG SOFT /SIGPLAN Symposium on Practical Software Develpment Environments, Pittaburgh, Pennsy lvania, April, 1984.

[4] Korel B. and S. Yalamanchili, "Forward Derivation of Dynamic Slices.", Proc. Int'l Symp. Software Testing and Analysis, Seattle, pp.66 -79, 1994.

[5] Kamkar. M., "Interprocedural Dynamic Slicing with Applications to Debugging and Testing.", PhD thesis, Linkoping Univ., 1993.

[6] Mark Weiser, "Programmers use slices when debugging," Communications of the ACM, pp.446-452, July 1982.

[7] Mark Weiser. "Program slicing," IEEE Trans. on Software Engineering, pp.352-357, July, 1984.

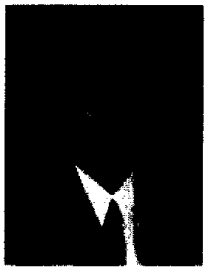
[8] Park, S. H. and Park, M. G., "An efficient dynamic program slicing algorithm and its Application.," Proc. of the IASTED International Conference, Pittsburgh, Pennsylvania, pp459-465, May, 1998.

[9] 박순형, 박만곤, "소프트웨어 테스팅을 위한 동적 프로그램 슬라이싱 알고리즘의 효율성 비교", 정보처리논문지 제5권 제9호, pp.2323-2334, 1998.

[10] Gupta R. , Harrold M. and Soffa M., "An Approach to Gegression Testion Using Slicing.", Conf. software Maintenance, pp.299-308, 1992.

[11] Susan Horwitz, Jan Prins, and Thomas Reps, "Integrating noninterfering versions of programs," ACM Trans. on Progra mming Languages and Systems, pp.345-387, July, 1989.

[12] Susan Horwitz, "Identifying the semantic and textual differences between two versions of a program," Proc. of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation, pp.234-245, 1990.



박 순 형

e-mail : shpark@dit.ac.kr

1981년 울산대학교 공과대학 전자계산학과(학사)

1985년 숭실대학교 대학원 전자계산학과(석사)

1997년~현재 부경대학교 대학원 전자계산학과 박사수료

1981년~83년 현대미포조선(주) 전산실 근무

1987년~현재 동의공업대학 전자계산과 교수

관심분야 : 소프트웨어 테스팅 및 디버깅, 비지니스 프로세스 재공학

박 만 곤

e-mail : mpark@dolphin.pknu.ac.kr

1976년 경북대학교 수학교육학과 졸업(이학사)

1987년 경북대학교 대학원 전산통계학과(이학박사)

1980년~1981년 경남공업전문대학 전자계산학과 교수

1990년~1991년 영국 리버풀대학 전자계산학과 객원교수

1992년~1993년 미국 캔사스대학교 컴퓨터공학과 교환교수

1995년 몽골 컴퓨터매핑 전문가로 외무부 파견

1996년 호주 사우스 오스트레일리아대학 컴퓨터 및 정보과학부 객원교수

1997년 중국 산동성 정부 시스템구축 전문가로 외무부 파견

1998년 필리핀 마닐라 CPSC 정보기술 및 통신학부 책임교수로 외무부 파견

1981년~현재 부경대학교 컴퓨터멀티미디어공학부 교수

관심분야 : 소프트웨어공학 및 재공학, 소프트웨어 신뢰성 및 안전성공학, 비지니스 프로세스 재공학, 멀티미디어 정보시스템, 소프트웨어품질공학, 소프트웨어 메트릭스, 소프트웨어 테스팅 및 감사, 결함허용 소프트웨어 시스템