소프트웨어 테스팅을 위한 동적 프로그램 슬라이싱 알고리즘의 효율성 비교

박 순 형† · 박 만 곤††

요 약

어떤 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. 본 논문에서는 이러한 동적 프로그램 슬라이스(dynamic program slices)를 산출하는 마킹 알고리즘을 제시하였고 이것을 프로그래밍한 뒤 예제 프로그램을 적용시켜 구현하였다. 구현 결과는 실행 이력에 대한 마킹 테이블(marking table), 동적 종속 그래프(Dynamic Dependence Graph) 그리고, 축소 동적 종속 그래프(Reduced Dynamic Dependence Graph)로 나타내었다. 그리고, 본 논문에서 제시한 효율적인 동적 슬라이스 생성을 위한 마킹 알고리즘과 동적 종속 그래프가 기존의 기법 보다 더 효율성이 높다는 것을 보였다.

On the Efficiency Comparison of Dynamic Program Slicing Algorithm for Software Testing

Soon-Hyung Park† · Man-Gon Park††

ABSTRACT

Software engineers generally analyze the program behavior under the test-case that revealed the error, not under any generic test case. In this paper we discuss the dynamic slice consisting of all statements that actually affect the value of a variable occurrence for a given program input. We propose an efficient algorithm to make dynamic program slices. The efficiency of this algorithm is evaluated on some developed programs. Results are shown by a marking table of execution history, Dynamic Dependence Graph, and Reduced Dynamic Dependence Graph. Consequently, the efficiency of the proposed algorithm is also presented by the comparison with algorithm that was announced previously.

1. 서 론

프로그램 슬라이싱은 어떤 포인터 p에서 변수 var의 값에 직 간접적으로 영향을 끼치는 프로그램 P에 있는 모든 명령문들을 찾는 것이다[1,10,11]. 따라서, 프로그램 슬라이싱은 프로그램 테스팅, 디버깅, 병렬 처리, 분산 그리고 유지 보수와 같은 분야에서 유용하게 응용된다[2,3,4]. 본 논문은 프로그램 테스팅 및 디버깅 분야에 유용하다.

정적 슬라이스(static slice)는 주어진 변수의 값에 영향을 주는 모든 명령문의 집합이고, 동적 슬라이스(dynamic slice)는 주어진 프로그램 입력에 대해 발생된 변수의 값에 실제적으로 영향을 주는 모든 명령문

† 정 회 원 : 동의공업대학 전자계산과 교수
†† 정 회 원 : 부경대학교 전자계산학과 교수
논문접수 : 1998년 10월 21일, 심사완료 : 1998년 7월 7일

들로 구성되어 있다[5,6,12].

프로그램 명령문 S에서 변수 var의 값을 디버깅하는 동안에 종종 오류가 발견된다. 이런 경우에 프로그래머는 시험 사례(test case)를 통해 프로그램을 분석한다. 정적 슬라이싱(static slicing)은 입력에 대한 변수의 값에 영향을 끼치는 모든 명령문들과 관련이 있기 때문에, 현재 입력 값에 영향을 끼치는 모든 명령문들에 관련된 동적 슬라이싱(dynamic slicing)이 실제 테스팅 과 디버깅에 더 적합하다고 할 것이다. 결과적으로 보면 정적 슬라이싱(static slicing)은 실행과 직접 관련이 없는 노드를 보유하게 된다.

본 논문에서는 효과적인 동적 슬라이스를 구하는 마킹 알고리즘을 제시하였고, 실제 구현한 결과를 보였다. 2장에서는 정적 슬라이싱과 동적 슬라이싱에 대해 정의하였고 프로그램 종속 그래프(program dependence graph)와 동적 종속 그래프(dynamic dependence graph)를 통해 정적 슬라이싱에 비해 동적 슬라이싱이 더 효율적임을 설명하였다. 3장에서는 기존 동적 슬라이싱 기법에 대해 설명하였으며 4장에는 효과적인 동적 프로그램 슬라이싱 알고리즘을 제시하였고 프로그래밍 언어를 사용하여 실제 구현한 결과를 마킹 테이블(marking table), 동적 의존 그래프, 축소 동적 의존 그래프(reduced dynamic dependence graph)를 통해 나타내었다. 5장에서는 본 논문에서 제시한 효율적인 동적 슬라이싱 알고리즘이 기존 알고리즘에 비해 효율적 임을 비교 고찰하였다.

2. 정적 프로그램 슬라이싱과 동적 프로그램 슬라이싱

프로그램 구조는 하나의 flow graph G=(N,A,s,e)로써 표현된다.

1) N : 노드들의 집합

2) A : arc의 집합으로써 N에 binary relation을 가짐.

3) s와 e는 각각 유일한 시작과 마지막 노드이다.

시작 노드 s에서 어떤 노드 k까지 (k∈N) path는 n_1=s, n_q=k 그리고 (n_i, n_{i+1})∈A, (1≤i<q)인 일련의 노드 {n_1, n_2, ..., n_q}이다.

어떤 한 프로그램의 프로그램 종속 그래프는 각각의 간단한 명령문 (예를 들어 assignment, read, write)에 대해 한 개의 노드를 가지며, 각각의 제어 술부 표현(control predicate expression : 예를 들어 if then else, while-do내에서의 조건 표현)에 대해서도 한 개의 노드를 갖는다. 프로그램 종속 그래프는 간선(edge)들로 연결되며 간선에는 두 가지 종류 즉, 자료 종속 간선(data-dependence edges)과 제어 종속 간선(control-dependence edges)[7,8]이 있다.

정점(vertex) v_i으로부터 정점 v_j 까지 자료 종속 간선은 정점 v_i에서 실행된 결과는 정점 v_j에서 실행된 값에 직접 종속된다. 즉, 정점 v_i 의 실행 결과를 구하기 위해 v_j에서 정의 되어진 변수 var을 사용한다. 그리고, v_i으로부터 정점 v_j 까지 제어 종속 간선은 노드 v_i가 노드 v_j에서 술어 표현의 boolean 결과에 따라 실행할 수도 안할 수도 있다는 의미이다.

실행 이력이란 주어진 시험사례를 실행하는 동안 방문된 순서에 의한 일련의 정점들인 {v_1, v_2, ... , v_n}의 집합이다. (그림 3)의 예제 프로그램 2에서 N = 2일 때 실행 이력 H_x에 대해 $H_x(9)$ = 5, $H_x(14)$ = 9로 나타낼 수 있다. H_x 내에 있는 position p에서 노드 Y (즉, $H_x(p)$ = Y)는 Y^p로 표현된다. 이와 같이 상위 첨자를 사용 하므로써 실행 이력에서 같은 노드의 다중 발생을 구별할 수 있다. 따라서, 실행이력 H_x는 {1, 2, 3, 4, 5^1, 6^1, 7^1, 8^1, 5^2, 6^2, 7^2, 8^2, 5^3, 9}가 된다.

정적 슬라이스는 노드 n에서 var에 도달하는 모든 것을 찾은 후 그 노드에서 시작하는 프로그램 종속 그래프를 운행(traversing) 함으로써 쉽게 구성될 수 있다. 따라서, (그림 1)의 예제 프로그램 1에 있는 명령문 10의 변수 Y에 대한 정적 슬라이스는 {1, 2, 3, 5, 6, 8}이 된다.

동적 슬라이스는 동일한 프로그램의 입력에 대해 어떤 실행위치 q에서 관심 변수에 관련된 원래의 프로그램과 그것의 결과가 일치하는 프로그램의 실행의 부분이다. 프로그램 입력 x에 대해 실행된 프로그램 P의 동적 슬라이싱 기준은 C=(x, I^q, V)이다. I는 실행 이력 H에서 position q의 명령문이다. V는 P에 있는 변수의 집합이다[11]. 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[9].

같은 슬라이싱 기준에 대해서도 많은 상이한 동적 슬라이스 기법들이 있다. 그리고 그 목적은 명령문의

최소의 수를 가진 슬라이스를 찾는 것이다.

(그림 1)의 예제 프로그램 1에 있는 명령문 10의 변수 Y에 대한 동적 프로그램 슬라이싱(dynamic program slicing)을 위해 X = -1일 때 시험 사례를 살펴보자. 실행 이력 (execution history)은 {1, 2, 3, 4, 10, 11}이 된다. 기준변수가 명령문 10의 Y이므로 Y에 대한 직전 정의는 명령문 3이 된다. 명령문 3으로부터 운행하면 {1, 2, 3}을 얻을 수 있다. 그러므로 동적 슬라이스는 {1, 2, 3}이 된다. (그림 2)는 (그림 1)의 예제 프로그램 1에 대한 동적 프로그램 슬라이싱 결과를 동적 종속 그래프로 나타낸 것이다. 동적 종속 그래프에서는 제일 처음 모든 노드들을 점으로 된 원으로 나타낸 뒤 그 중에 해당 시험 사례에 대한 실행 이력에 해당하는 노드 들은 가는 실선으로 된 원으로 나타내었다. 그리고, 동적 프로그램 슬라이스(dynamic program slices)에 해당되는 노드들은 굵은 실선으로 된 원으로 표시하였다. 만일 명령문 10에서 Y의 값이 틀린 결과가 나왔다면 이것은 명령문 3의 f1에서 오류가 있던지 아니면 명령문 2에서 오류가 있다고 할 수 있다. 확실히 동적 슬라이스 {1, 2, 3}은 정적 슬라이스{1, 2, 3, 5, 6, 8} 보다 버그의 범위를 국한시키는데 도움을 준다[1,9].

3. 기존 동적 프로그램 슬라이싱 기법

3.1 Agrawal & Horgan 기법

주어진 실행이력에 대한 변수에 관해 동적 슬라이스를 구하기 위해 Agrawal 과 Horgan이 제안한 3 가지 기법을 동적 종속 그래프를 통해 설명한다.

(1) 기법 1

ㄱ) 그래프에서 모든 노드들은 시작 시점에서 점선으로 그린다.

ㄴ) 명령문들이 실행 됨에 따라 발생하는 새로운 종속에 대응하는 간선 들을 실선으로 만든다.

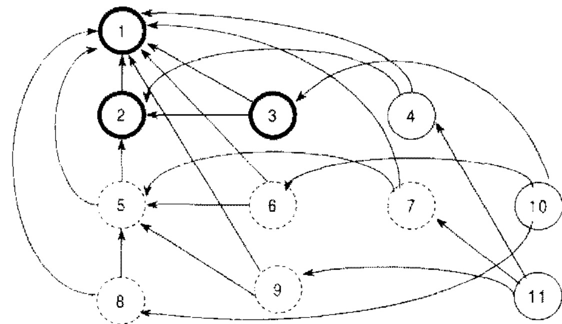
ㄷ) 실선으로 표시된 간선을 따라 운행하고 도달된 노드들을 굵은 실선으로 만든다.

(2) 기법 2

실행 이력에서 어떤 한 명령문의 각 발생들에 대해서 해당 명령문에 종속성 간선을 가진 단지 한 개의 명령문 만을 새로운 노드로 만든다.

```
        begin
S1:         read(X);
S2:         if (X < 0)
            then
S3:             Y := f1(X);
S4:             Z := g1(X);
            else
S5:             if (X = 0)
                then
S6:                 Y := f2(X);
S7:                 Z := g2(X);
                else
S8:                 Y := f3(X);
S9:                 Z := g3(X);
                end_if;
            end_if;
S10:        write(Y);
S11:        write(Z);
        end.
```

(그림 1) 예제 프로그램 1
(Fig. 1) Example Program 1



(그림 2) 예제 프로그램 1의 동적종속 그래프
(Fig. 2) DDG of Example Program 1

(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) 기법 1의 적용

(그림 3)의 예제 프로그램 2에 시험 사례로서 N = 1 일 때 변수 Z에 대한 동적 슬라이스를 산출하기 위해 기법 1을 적용 시킨다. 이 때, 실행 이력은 {1, 2, 3, 4, 5^1, 6, 7, 8, 5^2, 9}이 된다. 적용 시킨 결과는 (그림 4)에 나타나 있다.

(5) 기법 2의 적용

(그림 5)의 예제 프로그램 3에 시험 사례로서 N=3 이고 X={-4, 3, -2} 일 때 변수 Z에 대한

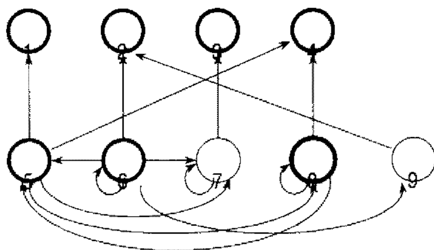
동적 슬라이스를 산출하기 위해 기법 2을 적용 시킨다. 이 때, 실행 이력은 {1, 2, 3^1, 4^1, 5^1, 6^1, 8^1, 9^1, 10, 3^2, 4^2, 5^2, 7, 8^2, 9^2, 10, 3^3, 4^3, 5^3, 6^2, 8^3, 9^3, 10, 3^4}이 된다. 기법 2를 적용 시킨 결과는 (그림 6)에 나타나 있다.

(6) 기법 3의 적용

(그림 5)의 예제 프로그램 3에 시험 사례로서 N=3 이고 X={-4, 3, -2} 일 때 변수 Z에 대한 동적 슬라이스를 산출하기 위해 기법 3을 적용 시킨다. 기법 3을 적용 시킨 결과는 (그림 7)에 나타나 있고, 특히 이 그래프를 축소 동적 종속 그래프라고 한다.

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

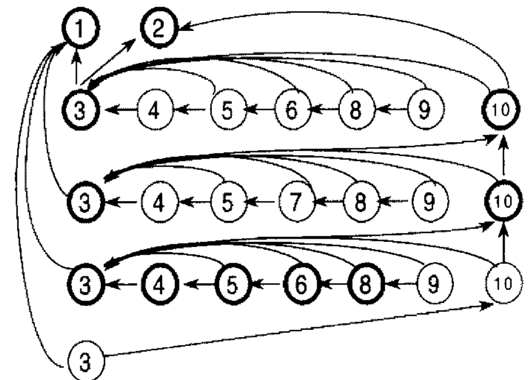
(그림 3) 예제 프로그램 2
(Fig. 3) Example Program 2



(그림 4) 예제 프로그램 2의 동적 종속 그래프
(Fig. 4) DDG of Example Program 2

```
      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 5) 예제 프로그램 3
(Fig. 5) Example Program 3



(그림 6) 예제 프로그램 3의 동적 종속 그래프
(Fig. 6) DDG of Example Program 3

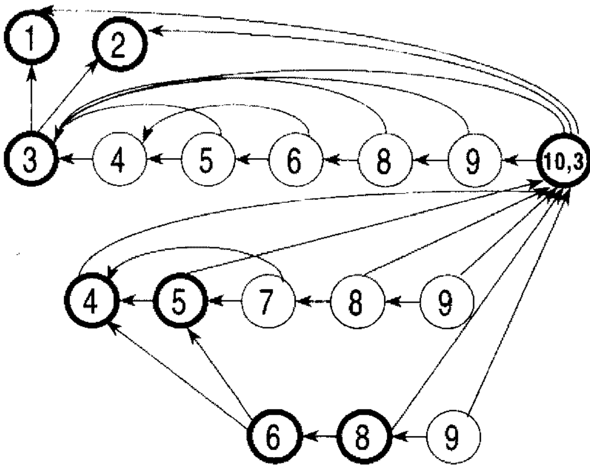
4. 효율적인 동적 프로그램 슬라이싱 기법

본 논문에서는 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)과 동적 종속 그래프와 축소 동적 종속 그래프로 나타내었다.

하나의 시험 사례를 통한 테스트를 위해 프로그램 P의 실행 이력을 H_x 그리고 주어진 변수를 v라 한다면 H_x와 v에 관한 P의 동적 슬라이스는 어떤 실행이 실행의 종료 시점에서 관찰되어진 것처럼 v의 값에 어떤 영향을 끼치는 H_x에서의 모든 명령문들의 집합이다. 따라서 동적 프로그램 슬라이싱을 구하기 위해서는 실행 이력과 슬라이싱 기준이 먼저 주어진다.

본 논문에서 프로그래밍 언어의 구조는 프로그래밍 언어 ML에서 유사 구조로 채택된 let-in구조를 사용한다[13].

let <declarations> in <expression>



(그림 7) 예제 프로그램 3의 축소 동적 종속 그래프
(Fig. 7) RDDG of Example Program 3

4.1 효율적인 프로그램 슬라이싱을 위한 입력 자료

동적 프로그램 슬라이싱 알고리즘을 구현하기 위해 실행 이력 자료와 노드 관련 자료가 필요하다.

실행 이력 자료란 주어진 시험 사례에서 실행되는 동안 관련되는 노드들을 실행 순서 대로 나열을 한 것으로 실행 이력 자료를 선행하는 첫번째 값은 슬라이싱의 기준 변수명이다. 노드 관련 자료란 각 명령문에 해당되는 노드 들의 구성 요소를 저장해 놓은 자료를 말하며 노드 번호, 노드 type, DEF, 그리고 USE로 구성된다. 노드 type에는 입력, 서술, 판단, 반복, 출력 등이 있으며, DEF(n)은 노드 n에서 바뀌어진 값을 가진 변수의 집합이고, USE(n)은 노드 n에서 사용된 값을 가진 변수의 집합이다.

4.2 효율적인 동적 프로그램 슬라이스 마킹 알고리즘

동적 프로그램 슬라이싱 알고리즘에는 두 개의 기준 변수 테이블과 한 개의 초기 정의 용 변수 테이블이 운영된다. 두 개의 기준 변수 테이블이란 서술 기준 변수 테이블(*DefnBaseNode*)과 제어 기준 변수 테이블(*PredBaseNode*)을 말하는데 서술 기준 변수 테이블은 직전 서술 명령문의 USE에 관련된 변수의 집합이고, 제어 기준 변수 테이블은 직전 제어 관련 명령문의 DEF에 관련된 변수로써 슬라이싱 제어 기준이 되는 변수들의 집합을 말한다. 초기 정의용 변수 테이블(*InitDefnNode*)은 사용되는 변수들의 직접 관련성 여부를 노드 type과 함께 체크를 하기 위한 것이다. 노드 type은 다음과 같이 분류된다.

NodeType	DefnType	Assign (서술,할당)	InitDefn
			AssgnDefn
	PredType	Input (입력)	
		Output (출력)	
		Repeat (반복)	
		Decision (판단)	

본 논문에서 제시한 효율적인 동적 프로그램 슬라이스(*ReachableStmts*)를 구하는 마킹 알고리즘은 다음과 같다.

```
ReducedDynamicDep(<prevhist | next>)=
   let ReducedDynamicDep( prevhist ) = ( V, A,
          ReachableStmts, DefnBaseNode,
             PredBaseNode ),
       D = U DefnBaseNode( var ),
       C = U PredBaseNode ( x ),
   in if InitSw = 0 then
          SearchBaseVar( BaseVar, ReachableStmt,
                  Def )
          InitSw = InitSw + 1
       end if,
       AssgnProc( NodeType, Def, Use,
                  DefBaseNode),
       ControlProc( NodeType, Def,
                  DefnBaseNode, PredBaseNode )

SearchBaseVar( BaseVar, ReachableStmt, Def ) =
   let R = { v } ∪ Ux→n ReachableStmts( x )
   in if BaseVar = Def( Var ) then R,
           AddDefnNode( DefnBaseNode, Use )
      else SearchBaseVar( BaseVar,
                 ReachableStmt, Def( next ))
      end if

AssgnProc( NodeType, Def, Use,
                  DefnBaseNode ) =
   let R = { v } ∪ Ux→n ReachableStmts( x ),
       n = first node number which depends
              Def( x )
   in if NodeType( x ) = DefnType then
         if Def( x ) ⊆ D ∪ C then R
```

```
        if Def( x ) ∈ D then
          DelDefnNode( DefnBaseNode, Def )
          DelPredNode( PredBaseNode, Def )
        end if
        AddDefnNode( DefnBaseNode, Use )
      else
        if NodeType( x ) ∈ InitDefnNode
        then
            RelatedInitVar( n, NodeType, Def,
                    Use, ReachableStmt )
        end if
      end if
    end if

ControlProc( NodeType, Def, DefnBaseNode,
             PredBaseNode ) =
  let R = { υ } ∪ ∪x∈D ReachableStmts( x )
  in if NodeType( x ) = PredType then
      if NodeType( x ) = 'Repeat' then
        if ( RepCnt > 1 )  then R
        end if
      end if
      RepCnt = RepCnt + 1
      AddPredNode( PredBaseNode, Use )
    end if
    if NodeType( x ) = 'Input' then
      if Def ( x ) ∈ D ∪ C then R
        DelDefnNode( DefnBaseNode, Def ),
        DelPredNode( PredBaseNode, Def )
      end if
    end if
AddDefnNode( DefnBaseNode, Var ) =
   ∪x∈D Var( x ) ∪ ∪x∈D DefnBaseNode( x )

DelDefnNode( DefnBaseNode, Var ) =
   ∪x∈D DefnBaseNode( x ) - ∪x∈D Var( x )

AddPredNode( PredBaseNode, Var ) =
   ∪x∈D Var( x ) ∪ ∪x∈D PredBaseNode( x )

DelPredNode( PredBaseNode, Var ) =
   ∪x∈D PredBaseNode( x ) ∪ ∪x∈D Var( x )

RelatedInitVar( n, NodeType, Def,
                Use, ReachableStmt )
  if NodeType( n ) = AssgnDefn ∩
        InitDefnNode( x ) = Def( n ) then ∅
  end if
  if NodeType( n ) = AssgnDefn ∩
       InitDefnNode( x ) = Use( n ) then
             ReachableStmt( x )
  end if
  if NodeType( n ) = 'Input' ∩
        InitDefnNode( x ) = Def( n ) then ∅
  end if
  if NodeType( n ) = 'Output' ∩
        InitDefnNode( x ) = Use( n )
  then ReachableStmt( x )
  end if
  if NodeType( n ) = 'Decision' ∩
      InitDefnNode = Use( n ) then
            ReachableStmt( x )
  end if
  if NodeType( n ) = 'Repeat' ∩
      InitDefnNode( x ) = Use( n ) then
            ReachableStmt( x )
  end if
```

4.3 적용 사례 1

(그림 3)의 예제 프로그램 2에 동적 프로그램 슬라이스 마킹 알고리즘을 적용시켜 보자. 시험 사례 N=1 인 경우 실행 이력은 {1, 2, 3, 4, 5^1, 6, 7, 8, 5^2, 9}이 된다. 여기에서 슬라이스 기준이 position 10의 노드 9에 있는 변수 Z일 때 동적 슬라이싱을 산출해 보자. 노드 관련 자료는 <표 1>에 나타나 있고 알고리즘의 실행 순서는 다음과 같다.

〈표 1〉 예제 프로그램 2의 노드 관련 자료

〈Table 1〉 The data of related nodes for the Example Program 2

노드 번호	TYPE	DEF	USE
1	입력	N	
2	서술	Z	
3	서술	Y	
4	서술	I	
5	반복		I , N
6	서술	Z	Z , Y
7	서술	Y	Y
8	서술	I	I
9	출력		Z

① 슬라이싱 기준의 실행순번이 10이므로 실행 이력에서 노드 9가 선택된다. 그리고, 변수Z에 직접 영향을 끼치는 노드를 구하기 위해 DEF가 Z인 노드를 찾는다.

② 노드 6이 슬라이스에 포함되며, 기준 변수 테이블은 {Z, Y}가 된다.

③ 노드 5는 슬라이싱 여부가 처음 체크되는 반복 제어술부 노드이기 때문에 슬라이스에 포함되지 않는다. 그러나 USE 변수 I와 N은 기준 변수 테이블에 포함된다. 기준 변수 테이블은 {Z, Y, I, N}이 된다.

④ 노드 4는 DEF가 I이고 기준 변수 테이블에 I가 있으므로 슬라이스에 포함된다. 기준 변수 테이블은 {Z, Y, N}이 된다.

⑤ 노드 3은 DEF가 Y이고 기준 변수 테이블에 Y가 있으므로 슬라이스에 포함된다. 기준 변수 테이블은 {Z, N}이 된다.

⑥ 노드 2는 DEF가 Z이고 기준 변수 테이블에 Z가 있으므로 슬라이스에 포함된다. 기준 변수 테이블은 {N}이 된다.

⑦ 노드 1은 DEF가 N이고 기준 변수 테이블에 N이 있으므로 슬라이스에 포함된다. 기준 변수 테이블은 blank가 된다.

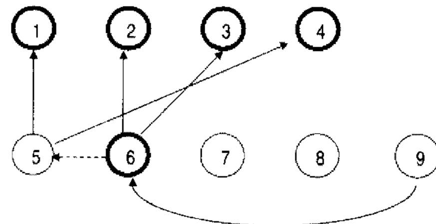
그러므로 동적 슬라이스는 {1, 2, 3, 4, 6}이 된다.

〈표 2〉 예제 프로그램 2의 동적 슬라이싱 결과인 마킹 테이블

〈Table 2〉 Marking table that draw results of dynamic slicing algorithm

순번	실행 이력	Marking
1	1	✓
2	2	✓
3	3	✓
4	4	✓
5	5	
6	6	✓
7	7	
8	8	
9	5	
10	9	

동적 프로그램 슬라이스 마킹 알고리즘을 적용시킨 결과인 마킹 테이블은 <표 2>에 나타나 있으며 이 결과를 동적 종속 그래프로 표현하면 (그림 8)과 같다. $<v_k, v_{k-1}>$의 간선이 점선인 경우에는 v_k에 대해 직접적인 종속관계는 없으나 $USE(v_{k-1})$는 새 기준변수 테이블에 저장된다. 이 경우는 v_{k-1}이 첫 번째 실행되는 반복노드일 때 발생한다. 따라서 (그림 8)에서 점선으로 표시된 노드는 종속 값이 ∅ 이므로 동적 슬라이스에 포함되지 않는다.



(그림 8) 예제 프로그램 2의 효율적인 동적 종속 그래프

(Fig. 8) EDDG of Example Program 2

4.4 적용 사례 2

(그림 9)의 예제 프로그램 4에 동적 프로그램 슬라이싱 알고리즘을 적용시켜 보자. 시험사례는 N = 3이고 X = {-2, 4, -3} 인 경우 실행 이력은 {1, 2, 3, 4^1, 5^1, 6^1, 7^1, 9^1, 10^1, 11^1, 4^2, 5^2, 6^2, 8, 9^2, 10^2, 112, 43, 53, 63, 72, 93, 103, 113, 44}가 된다. 여기에서 슬라이스 기준이 position 23의 노드 10에 있는 변수 Z일 때 동적 슬라이싱을 산출해 보자. 노드 관련 자료는 <표 3>에 나타나 있다. 동적 프로그램 슬라이스 마킹 알고리즘을 적용시킨 결과인 마킹 테이블은 <표 4>에 나타나 있으며, 이 결과를 동적 종속 그래프로 표현하면 (그림 10)과 같다. 그리고, 동일한 종속성을 가진 노드가 이중으로 존재하지 않는 축소 동적 종속 그래프로 표현하면 (그림 11)과 같다.

```
      begin
S1:       read(N);
S2:       I := 1;
S3:       X := 1;
S4:       while (I <= N)
          do
S5:           read(X);
S6:           if (X < 0)
              then
S7:               Y := f1(X);
              else
S8:               Y := f2(X);
              end_if;
S9:           Z := f3(Y);
S10:          write(Z);
S11:          I := I + 1;
          end_while;
      end.
```

(그림 9) 예제 프로그램 4

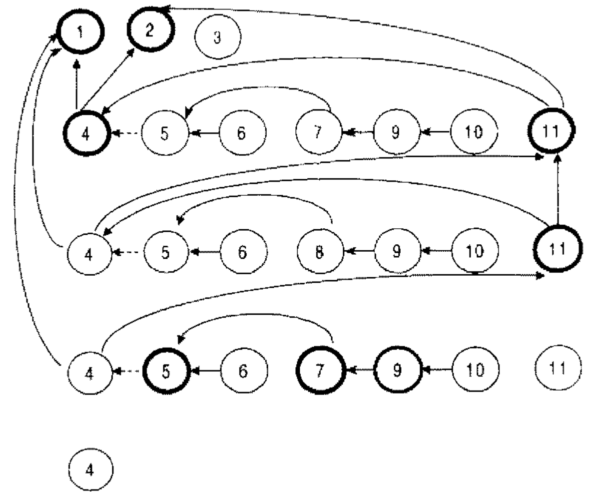
(Fig. 9) Example Program 4

〈표 3〉 예제 프로그램 4의 노드 관련 자료
〈Table 3〉 The data of related nodes for the Example Program 4

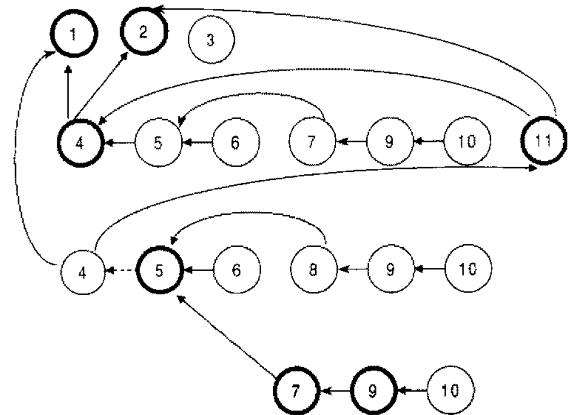
노드 번호	TYPE	DEF	USE
1	입력	N	
2	서술	I	
3	서술	X	
4	반복		I , N
5	입력	X	
6	판단		X
7	서술	Y	X
8	서술	Y	X
9	서술	Z	Y
10	출력		Z
11	서술	I	I

〈표 4〉 예제 프로그램 4의 동적 슬라이싱 결과인 마킹테이블
〈Table 4〉 Marking table that draw results of dynamic slicing algorithm for the Example Program 4

순번	실행 이력	Marking
1	1	✓
2	2	✓
3	3	
4	4	✓
5	5	
6	6	
7	7	
8	9	
9	10	
10	11	✓
11	4	✓
12	5	
13	6	
14	8	
15	9	
16	10	
17	11	✓
18	4	
19	5	✓
20	6	
21	7	✓
22	9	✓
23	10	
24	11	
25	4	



(그림 10) 예제 프로그램 4의 효율적인 동적 종속 그래프
(Fig. 10) EDDG of Example Program 4



(그림 11) 예제 프로그램 4의 효율적인 축소
(Fig. 11) ERDDG of 동적 종속 그래프Example Program 4

5. 동적 프로그램 슬라이싱 기법의 비교 및 고찰

본 논문에서 제안한 동적 슬라이싱 기법이 기존의 기법과 비교하여 실제로 노드 수가 줄어듦을 증명하기 위해 기존의 기법 중 대표적인 Agrawal & Horgan 기법을 선택하였다. 그리고, Agrawal H. 과 Horgan J.가 발표한 논문[1]에서 적용한 프로그램 예를 그대로 본 논문에 적용을 하였다. 그리고, 비교 결과를 table로 나타내었다.

5.1 비교 사례 1(예제 프로그램 2를 중심으로)

(그림 3)의 예제 프로그램 2에 시험 사례로서 N = 1 일 때 슬라이스 기준이 position 10의 노드 9에 있는 변수 Z에 대한 프로그램 슬라이스를 구하는 경우를

Agrawal & Horgan 기법과 본 논문에서 제안한 동적 슬라이스 마킹 기법을 사용하여 서로 비교해 보았다.

Agrawal & Horgan 기법을 사용한 결과가 (그림 4)에 나타나 있고, 본 논문에서 제안한 기법을 사용한 결과가 (그림 8)에 나타나 있다. 그리고, 두 기법에 대한 비교 결과표가 <표 5>에 나타나 있다. <표 5>의 결과에서 보면 5 개의 명령문으로 산출 결과가 나온 본 논문에서 제안한 기법 {1, 2, 3, 4, 6}이 7 개의 명령문으로 산출 결과가 나온 Agrawal & Horgan 기법 {1, 2, 3, 4, 5, 6, 8}보다 더 효율적 임을 알 수 있다.

〈표 5〉 예제 프로그램 2의 동적 슬라이싱 결과인 마킹 테이블 비교

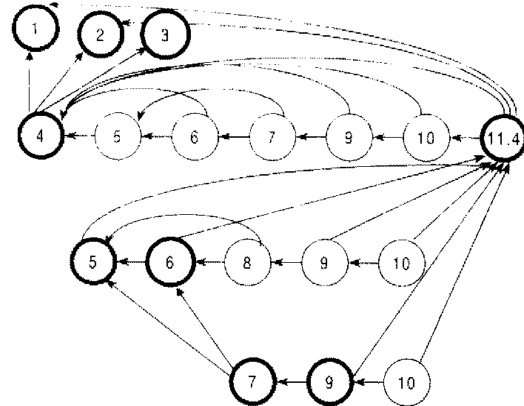
〈Table 5〉 The comparison of dynamic program slicing for the Example Program 2

순번	실행이력	Agrawal & Horgan's 기법	제안한 동적 슬라이싱 기법
1	1	✓	✓
2	2	✓	✓
3	3	✓	✓
4	4	✓	✓
5	5	✓	
6	6	✓	✓
7	7		
8	8	✓	
9	5		
10	9		

5.2 비교 사례 2(예제 프로그램 4를 중심으로)

(그림 9)의 예제 프로그램 4에 시험 사례로서 N = 3 이고 X = {-2, 4, -3} 일 때 슬라이스 기준이 position 23의 노드 10에 있는 변수 Z에 대한 프로그램 슬라이스를 구하는 경우를 Agrawal & Horgan 기법과 본 논문에서 제안한 동적 슬라이싱 기법을 사용하여 서로 비교해 보았다.

Agrawal & Horgan 기법을 사용한 결과가 (그림 12)에 나타나 있고, 본 논문에서 제안한 기법을 사용한 결과가 (그림 11)에 나타나 있다. 그리고, 두 기법에 대한 비교 결과표가 <표 6>에 나타나 있다. <표 6>의 결과에서 보면 7 개의 명령문으로 산출 결과가 나온 본 논문에서 제안한 기법 {1, 2, 4, 5, 7, 9, 11}이 9 개의 명령문으로 산출 결과가 나온 Agrawal & Horgan 기법 {1, 2, 3, 4, 5, 6, 7, 9, 11}보다 더 효율적 임을 알 수 있다.



(그림 12) 예제 프로그램 4의 Agrawal & Horgan의 축소 동적 종속 그래프

(Fig. 12) RDDG of Agrawal & Horgan for Example Program 4

5.3 고찰

제어 술부 노드(*while-do, if-than-else*에서의 조건 노드)는 무조건 슬라이스에 포함시키는 기존의 동적 슬라이싱 기법에 비해 본 논문에서는 노드에 직접 영향을 미치지 않는 제어 술부 노드는 제외시키는 방식이므로 제어 술부 노드라 하여도 주어진 슬라이스 기준 변수에 직접 종속되지 않으면 슬라이스에 포함 시키지 않는다. 그러므로 동적 슬라이스의 크기가 줄어들게 되며 따라서, 기존의 슬라이싱 기법 보다 효율적임을 알 수 있다. 그리고, 본 논문에서 제안한 동적 슬라이싱 기법은 프로그램을 구성하는 노드가 특히 다음과 같은 환경일 때 기존 알고리즘 보다 효율성이 높아짐을 알 수 있다.

(1) 직접 종속되지 않는 변수 초기 정의문 노드

(2) 반복분의 반복 실행 회수가 아주 적은 프로그램에서 반복분의 제어 술부 노드(*while-do*에서의 조건 노드)

(3) 직접 종속되지 않는 판단 제어 술부 노드 (*if-than-else*에서의 조건 노드)

따라서, 프로그램에서 제어 술부 노드가 많으면 많을수록 기존 동적 기법들에 비해 더 효율적이 된다.

6. 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. 본 논문에서는 이러한 동적 프로그램 슬라이스를 만드는 마킹 알고리즘을 제시하였고 프로그래밍 언어를 사용하여 동적 프로그램 슬라이스 마킹 알고리즘을 프로그래밍한 뒤 예제 프로그램을 적용시켜 구현하였다. 동적 프로그램 슬라이스 마킹 알고리즘에는 세 개의 테이블이 운영된다 - 서술 기준 변수 테이블, 제어 기준 변수 테이블 그리고, 초기 정의용 변수 테이블. 구현 결과는 실행 이력에 대한 마킹 테이블, 동적 종속 그래프 그리고, 축소 동적 종속 그래프로 나타내었다.

〈표 6〉 예제 프로그램 4의 동적 슬라이싱 결과인 마킹 테이블 비교
〈Table 6〉 The comparison of dynamic program slicing for the Example Program 4

순번	실행 이력	Agrawal & Horgan's 기법	제안한 동적 슬라이싱 기법
1	1	✓	✓
2	2	✓	✓
3	3	✓	
4	4	✓	✓
5	5		
6	6		
7	7		
8	9		
9	10		
10	11	✓	✓
11	4	✓	✓
12	5		
13	6		
14	8		
15	9		
16	10		
17	11	✓	✓
18	4	✓	
19	5	✓	✓
20	6	✓	
21	7	✓	✓
22	9	✓	✓
23	10		
24	11		
25	4		

구현에 필요한 입력 양식에는 두 가지 종류가 있다. 이 두 가지는 실행 이력 자료와 노드 관련 자료를 말하는데, 노드 관련 자료는 노드 번호, 노드 type, DEF, 그리고 USE로 구성된다. 그리고, 마지막에는 제안한 동적 슬라이싱 기법과 기존의 Agrawal & Horgan 기법을 비교하여 본 논문에서 제안한 기법이 마킹 테이블이나 동적 종속 그래프를 통해 더 효율적임을 보였다.

향후 연구 과제는 원시 프로그램을 분석하여 실행 이력 자료를 실제 생성 시키는 것이다. 이렇게 함으로써 본 논문에서 제시한 동적 프로그램 슬라이스 산출 프로그램과의 연계를 통해 완벽한 구현을 할 수 있다.

참 고 문 헌

[1] Hiralal. Agrawal and J. R. Horgan. "Dynamic Program Slicing", Proc. ACM SIGPLAN'90 Conf. Programming Lang. Design and Implementaion, pp.246-256, 1990.

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

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

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

[5] 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.

[6] Susan Horwitz, T. Reps and David Binkley, "Interprocedural Slicing using Dependence Graph", ACM Tran. on Programming Languages and Systems, Vol.12, No.1, 1990.

[7] 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.

[8] Karl J. Ottentein and Linda M. Ottentein. "The

program dependence graph in a software development environment.", Proc. of the ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments, Pittaburgh, Pennsilvania, April 1984.

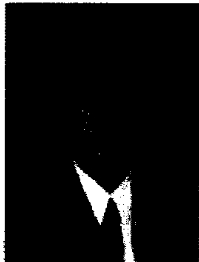
[9] Bodan Korel, "Computation od Dynamic Program Slices for Unstructured Programs", IEEE Trans. on Software Engineering, Vol.23, No.1, pp.17-34, January 1997.

[10] Cheng J, "Slicing Concurrent Programs.", Proc. First Int'l Workshop Automated and Algorithmic Debugging, Linkoping, Sweden, pp.244-261, 1993.

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

[12] Gupta R., Harrold M. and Soffa M, "An Approach to Regression Testing Using Slicing.", Conf. Software Maintenance, pp.299-308, 1992.

[13] Robin Milner, Mads Tofte, and Robert Harper, "The Definition of Standard ML.", The MIT Press, Cambridge, MA, 1990.



박 순 형

1981년 울산대학교 공과대학 전자계산학과(학사)

1985년 숭실대학교 대학원 전자계산학과(석사)

1997년~현재 부경대학교 대학원 전자계산학과 박사과정

1981년~1983년 현대미포조선(주) 전산실 근무

1987년~현재 동의공업대학 전자계산과 부교수

관심분야 : 소프트웨어 테스팅 및 디버깅, 비지니스 프로세스 재공학

박 만 곤

1976년 경북대학교 수학교육학과(이학사)

1987년 경북대학교 대학원 전산통계학과(이학박사)

1980년~1981년 경남공전 전산학과 교수

1990년~1991년 영국리버풀대학 전자계산학과 객원교수

1992년~1993년 미국 캔사스대학교 컴퓨터공학과 교환교수

1996년 호주 사우스 오스트레일리아대학 컴퓨터 및 정보과학부 객원교수

1995년 몽골 컴퓨터매핑 전문가로 외무부 파견

1997년 중국 산동성 정부 시스템구축 전문가로 외무부 파견

1981년~현재 부경대학교 전자계산학과 교수

1998년~현재 필리핀 마닐라 CPSC 정보기술 및 통신학부 책임 교수로 외무부 파견 근무 중

관심분야 : 소프트웨어공학 및 재공학, 소프트웨어 신뢰성 및 안전성 공학, 비즈니스 프로세스 재공학, 소프트웨어 품질 공학, 소프트웨어 메트릭스, 소프트웨어 테스팅 및 감사, 결함허용 소프트웨어 시스템, 멀티미디어 정보시스템 등.