효율적인 동적 시스템 종속 그래프

박순형·박만곤

부경대학교 컴퓨터 멀티미디어 공학부

Efficient Dynamic System Dependence Graph

Soon-Hyung Park·Man-Gon Park
Faculty of Computer and Multimedia Engineering
PuKyong National University

요 약

전통적인 기존의 슬라이싱 기법들은 종속 그래프를 통해 슬라이스를 산출하였고, 슬라이스의 정확성을 입증하였다. 그러나, 기존의 종속 그래프 기법은 정적 슬라이싱 기법을 바탕으로 하기 때문에 프로시져 간의 매개변수별 자료 전달 링크를 나타내기 위하여 많은 정점들과 간선들이 필요하다. 그래서 그래프가 매우 복잡하다. 본 논문에서는 어떤 작업을 처리하기 위해 관련된 여러 개의 프로그램으로 구성된 소프트웨어 시스템의 슬라이싱을 수행하기 위한 동적 시스템 종속 그래프의 표현법에 대해 제안하였다. 그리고, 본 논문에서 제안한 동적 시스템 종속 그래프 기법과 기존의 프로그램 종속 그래프 기법에 대한 복잡도 측정 공식을 제안하였으며, 동적 시스템 종속 그래프 기법이 기존의 기법에 비해 그래프의 복잡도가 작아 효율적임을 보였다.

1. 서론

프로그램 슬라이싱(program slicing)은 프로그램 디버깅 처리에서 해당 프로그램 중에서 관심이 있는 변수에 대해 관련된 명령문만을 모아놓은 또 다른 프로그램을 제공해 줌으로서 디버깅에 대한 효율성을 높여주는 기술이다. 즉, 어떤 포인터 p에서 변수 var의 값에 직 간접적으로 영향을 끼치는 프로그램 P에 있는 모든 명령문들을 찾는 것이다[6,7,10,15]. 시스템 슬라이싱(system slicing)은 관련된 프로그램들을 모아놓은 소프트웨어 시스템에서의 관심이 있는 변수에 대한 슬라이싱이다.

기존의 슬라이싱 기법들은 정확한 슬라이스를 산출하기 위한 슬라이싱 과정으로 종속 그래프를 사용하였다. 그러나, 기존의 종속 그래프 특히, 시스템 종속 그래프는 프로시져 간의 자료전달을 표시하기 위하여 많은 정점들과 간선들이 필요하기 때문에 매우 복잡하다. 그러므로 실제로 프로그래머나 테스터들이 적용하기에는 매우 어렵다. 그리고, 기존의 연구들은 대부분 정적 슬라이스를 산출하기 위한 효율적인 종속 그래프 표현법이 주류를 이루고 있다. 만일 슬라이싱 대상 프로그램이 1개 일 경우 정적 슬라이스를 산출하는 그래프를 프로그램 종속 그래프, 대상 프로그램이 여러 개일 경우 정적 슬라이스를 산출하는 그래프를 시스템 종속 그래프라 한다[2,3,11,12].

본 논문에서는 기존의 동적 종속그래프 개념을 확장하여 동적 시스템 종속 그래프를 나타내는 효율적인 표기법을 제안하였다.

2 장에서는 기존 슬라이싱 기법에 대한 종속 그래프에 대해 기술하였으며 3 장에는 본 논문에서 제안한 동적 시스템 슬라이싱에 대한

동적 시스템 종속 그래프에 대해 기술하였다. 4장에서는 제안된 슬라이싱 기법과 기존의 슬라이싱 기법과의 효율성을 비교 분석을 하였다.

2. 종속 그래프

정확한 프로그램 슬라이스의 산출을 확인하는 방법으로 전통적으로 종속그래프를 이용한 표현법을 사용한다. 따라서, 프로그램 슬라이싱 기법이 발달함에 따라 여러 가지의 종속 그래프 표현법이 소개되었다. 그러므로 본 단원에서는 종속그래프를 설명함으로서 기존의 프로그램 슬라이싱에 대해 살펴본다.

프로그램 슬라이싱 기법은 프로그램 입력값에 따른 실행이력의 적용여부에 따라 정적 프로그램 슬라이싱과 동적 프로그램 슬라이싱으로 나눈다. 정적 프로그램 슬라이싱은 슬라이싱 기준에 주어진 프로그램의 한 지점에서 변수에 영향을 줄 수 있는 모든 노드들을 추출하는 방법으로서 프로그램 종속 그래프 기법 혹은 시스템 종속 그래프 기법을 사용하여 정적 프로그램 슬라이싱을 산출할 수 있다. 동적 프로그램 슬라이싱은 프로그램의 입력에 대해 어떤 실행위치 q에서 관심 변수에 관련된 원래의 프로그램과 그것의 결과가 일치하는 프로그램 실행의 부분을 산출하는 것으로 동적 프로그램 슬라이싱을 위하여 동적 종속 그래프 기법을 사용하여 슬라이스를 산출할 수 있다.

2.1 프로그램 종속 그래프 기법

프로그램 종속 그래프(PDG : Program Dependence Graph)는 Susan Horwitz, Tomas Reps, David Binkley에 의해 제안된 슬라이싱 방법으로 원시 프로그램을 자료 흐름 분석과 제어 흐름 분석을 통하여 프로그램 종속성 그래프(PDG : Program Dependence Graph)로 변환하여 슬라이스를 산출하는 방법이다. 프로그램 종속 그래프는 간선(edge)들로 연결되며 간선의 종류에는 정점(vertex) v_i에서 실행된 결과가 정점 v_j에서 실행된 값에 직접 종속하는 자료 종속 간선(data-dependence edges)과 노드 v_i가 노드 v_j에서 술어 표현의 boolean 결과에 따라 실행할 수도 혹은 안 할 수도 있는 제어 종속 간선(control-dependence edges)으로 나눌 수 있다[2,3].

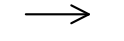
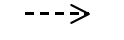
2.2 시스템 종속 그래프 기법

시스템 종속 그래프(SDG : System Dependence Graph)는 시스템의 주프로그램을 표현한 프로그램 종속 그래프와, 시스템의 보조 프로시져를 표현한 프로시져 종속 그래프, 그리고 약간의 추가 간선들을 포함한다. 추가 간선들은 두 가지로 분류된다. (1) 호출문과 호출된 프로시져 사이의 직접 종속을 표현한 간선. (2) 호출에 따른 이행적 종속을 표현한 간선.

그리고, 프로그램 종속 그래프에서 정점은 크게 procedure node와 parameter node로 나눌 수 있다.

procedure node	procedure name vertex	PV	
	general vertex	GV	
	call vertex	CV	
parameter node	call parameter	CP	
	procedure parameter	PP	

edge는 크게 세 가지로 나눌 수 있다.

control, data	
intra-procedure flow	
inter-procedure flow	

그러나, 매개변수 개수의 2배만큼의 정점들이 추가되기 때문에 전체 정점들의 수가 증가되고, 또 그들간에 간선으로 연결되기 때문에 그래프가 복잡해지는 단점이 있다.

(그림 1)의 예제 프로그램에 대한 SDG가 (그림 2)에 나타나 있다.

3. 동적 시스템 종속 그래프

본 논문에서 제시한 동적 시스템 종속 그

래프(DSDG: Dynamic System Dependence Graph)는 주어진 입력 값에 대한 실행이력을 산출한 뒤 슬라이싱 기준노드를 중심으로 종속성 간선을 가지는 노드들을 표현하는 그래프로서 동적 프로그램 슬라이싱을 위하여 동적 종속 그래프 기법을 사용하여 효율적으로 슬라이스를 산출할 수 있다. 동적 프로그램 슬라이싱은 프로그램의 입력에 대해 어떤 실행위치 q에서 관심 변수에 관련된 원래의 프로그램과 그것의 결과가 일치하는 프로그램 실행의 부분을 산출한다.

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. 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, ... , v_n}의 집합이다.

동적 시스템 종속 그래프는 정점(vertex)과 간선(edge)으로 구성되며 초기의 정점은 가는 실선으로 나타내며 슬라이스된 정점은 굵은 실선으로 나타낸다. 초기의 모든 간선은 가는 점선으로 표시되며, 슬라이스된 노드를 연결하는 간선의 종류에는 3가지 종류가 있다.

intra 종속 간선	———
inter 종속 간선	▬▬▬
return 종속 간선	▬ ▬ ▬

(그림 1)의 예제 프로그램에 대한 DSDG가 (그림 3)에 나타나 있다.

4. 종속 그래프의 효율성 비교

기존의 시스템 종속 그래프와 본 논문에서 제시한 동적 시스템 종속 그래프의 복잡도를 측정하여 비교하면 다음과 같다.

(그림 1) 예제 프로그램

```
 1  program Main
 2    x=1
 3    y=1
 4    i=1
 5    while i<=2 do
 6       call A(x,y,z,i)
      loop
 7    print z
 8  end

 9  procedure A(a,b,c,d)
10    input k
11    a=k**2
12    call IFF(a,b,c)
13    call Increment(d)
14  end

15  procedure  IFF(p,q,r)
16    if p<0 then
17      q=f1(p)
18      r=f2(q)
      else
19      q=f3(p)
20      r=f4(q)
      end if
21  end

22  procedure Increment(e)
23    e=e+1
24  end
```

4.1 시스템 종속 그래프의 복잡도

SDG의 정점들에 대한 복잡도는 다음과 같다.

· {∑σ(SDG)}
= {PV + GV + CV + ⌈CP × 2⌉ +
⌈PP × 2⌉}

SDG의 edge들에 대한 복잡도는 다음과 같다.

· {∑τ(SDG)}
= {∑data, control edge} +
{∑intra-procedure flow edge} +
{∑inter-procedure flow edge}
= {⌈GV × 2⌉ + ⌈CP × 2⌉ +

⌈PP × 2⌉} + {⌈CP × 2⌉} +
{⌈(PV − 1) + PP × 2⌉}

・ϕ(SDG)
= {$\sum\sigma$(SDG)} + {$\sum\tau$(SDG)}
= 2PV + 3GV + CV + 6CP + 6PP − 1

4.2 동적 시스템 종속 그래프의 복잡도

본 논문에서 제시한 동적 종속 기법(DSDG)에 대한 종속 그래프의 복잡도 ϕ는 다음과 같다.

・$\sum\sigma$(DSDG)
= PV + GV + CV

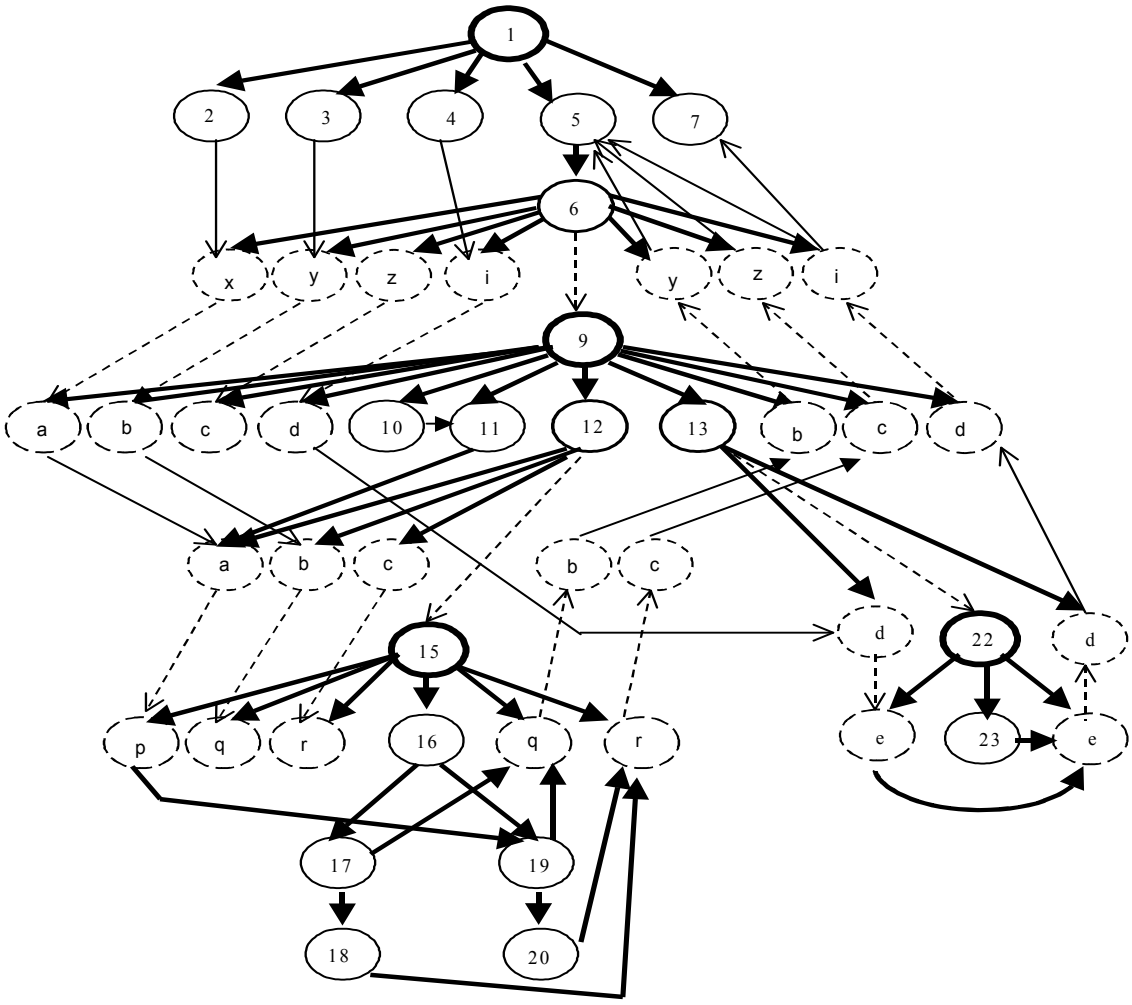
+ {$\sum$*return종속간선*}
= {⌈2(GV + CV) − 1⌉} + {CV} + {CV}

・ϕ(DSDG)
= {$\sum\sigma$(*동적기법*)} + {$\sum\tau$(*동적기법*)}
= PV + 3GV + 5CV − 1

4.3 종속 그래프 복잡도 비교

기존의 시스템 종속 기법(SDG)과 본 논문에서 제시한 동적 종속 기법(DSDG)에 대한 복잡도의 差는 다음과 같다.

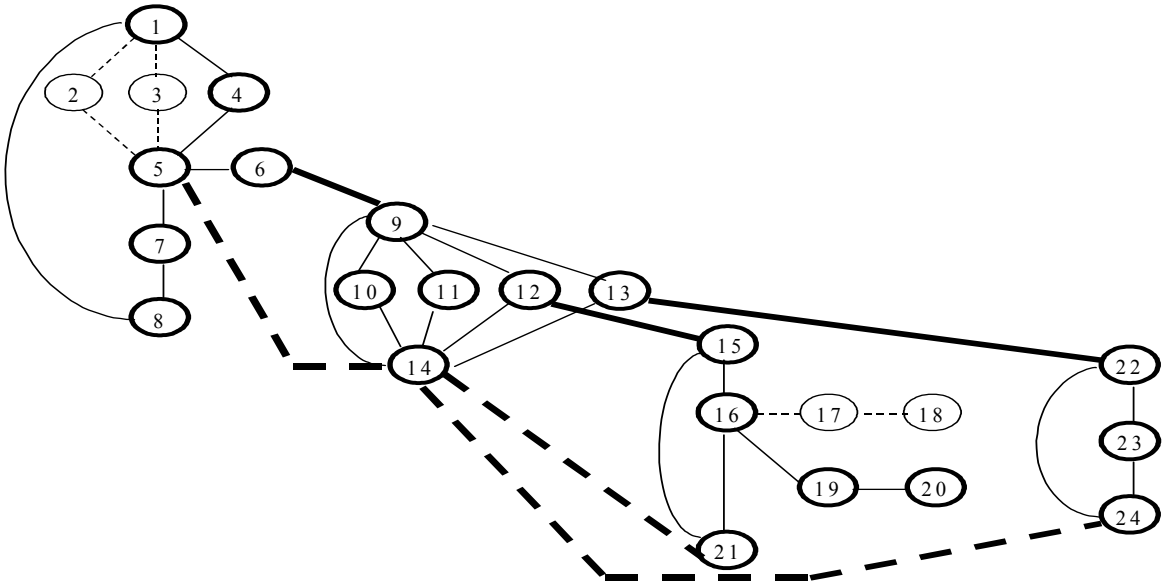
・ϕ(SDG) − ϕ(DSDG)
= PV − 5CV + 6CP + 6PP



(그림 2) 예제 프로그램 1의 SDG

・$\sum\tau$(DSDG)
= {$\sum$*intra종속간선*} + {$\sum$*inter종속간선*}

PV와 CV의 크기가 CP 혹은 PP에 비해 매우



(그림 3)예제 프로그램 1의 DSDG

작으므로

・ϕ(SDG) − ϕ(DSDG) = 6(CP + PP)

그러므로 기존의 시스템 종속 기법이 본 논문에서 제시한 동적 종속 기법에 비해 복잡도가 CP와 PP의 6배 만큼 크다는 것을 알 수 있다.

기존의 시스템 종속 기법과 본 논문에서 제시한 동적 종속 기법에 대한 (그림 1)의 예제 프로그램의 종속 그래프의 복잡도 ϕ는 다음과 같다. GV의 괄호안의 숫자는 DSDG의 GV의 개수이다.

프로그램 순번	PV	GV	CV	PP	CP
1	1	5(6)	1	0	4
2	1	2(3)	2	4	4
3	1	5(6)	0	3	0
4	1	1(2)	0	1	0

・ϕ(SDG) = ⌈(8 + 39 + 3 + 48 + 48 − 1)⌉
= ⌈145⌉ = 129

・ϕ(DSDG) = ⌈(4 + 51 + 15 − 1)⌉
= ⌈69⌉ = 59

5. 결론

기존의 슬라이싱 기법은 프로그램 종속그래프(PDG) 혹은 시스템 종속 그래프(SDG)를 통해 슬라이싱 한다. 그러나, 시스템 종속 그래프를 작성할 때 프로시져와 호출문의 매개변수의 최대 2배 만큼 그래프의 정점들이 추가되고 이 정점들을 연결하는 간선들도 최대 4배 만큼 증가되므로 그래프의 복잡성은 프로시져와 호출문의 매개변수의 최대 6배 만큼 증가된다. 그러므로 프로시져들의 수가 많을수록 관련 매개변수의 개수가 많을수록 그래프의 복잡성은 매우 증가한다. 그러나, 본 논문에서 제시한 동적 시스템 종속 그래프를 이용하면 그래프의 복잡성이 줄어든다.

본 논문에서 제시한 동적 시스템 종속 그래프 기법과 정적 기법에 의한 기존의 시스템 종속 그래프 기법을 그림 1의 예제 프로그램에 적용한 결과 본 논문에서 제시한 복잡성 측정 공식에 적용하면 동적 시스템 종속 그래프가 시스템 종속 그래프 보다 복잡성이 52% 감소됨을 알 수 있다. 그리고, 실제 그림 2와 그림 3에 나타난 종속 그래프를 비교하면 동적 시스템 종속 그래프가 시스템 종속 그래프에 비해 복잡성이 54% 줄어듦을 알 수 있다. 그러므로 본 논문에서 제시한 동적 시스템 종속 그래프 기법이 기존의 기법에 비해 복잡성을 줄여주는 효율적인 표현법임을 알 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 Development Environments, Pittaburgh, Pennsylvania, 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 Programming 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.

[13] Das, M. "Partial evaluation using dependence graphs.", Ph.D. dissertation and Tech. Rep. TR-1362, Computer Sciences Department, University of Wisconsin, Madison, WI, February 1998.

[14] Melski, D. and Reps, T., "Interprocedural path profiling", In Proc. of CC '99: 8th Int. Conf. on Compiler Construction, (Amsterdam, The Netherlands, Mar. 22-26, 1999), Lecture Notes in Computer Science, Vol. 1575, S. Jaehnichen (ed.), Springer-Verlag, New York, NY, 1999, pp.47-62.