객체지향 프로그램 슬라이싱에 관한 연구

방극인, 박영옥, 이 준

조선대학교 컴퓨터 공학과

A Study on the Object-Orient Program Slicing

Geuk-In Bang, Young-ok Park, Joon Lee

Dept. of Computer Engineering, Chosun Univ.

요 약

일반적인 소프트웨어 시스템은 새로운 요구와 오류의 발견으로 인해 계속적인 개발과 확장 수정이 요구되며, 어떤 프로그램에 특정 명령문의 변수에 대한 관련 명령문을 찾고 싶을 때 프로그래머는 입력자료의 값에 대해 프로그램의 실행 궤도 추적을 통해 프로그램을 분석한다.

그러므로 본 논문은 기존의 프로그램 슬라이싱 방법에 객체지향 프로그램 슬라이싱을 적용하여 프로그램의 실행 궤도을 통하여 객체지향 그래프와 슬라이싱의 알고리즘을 보인다. 객체지향 프로그램 종속성 그래프는 클래스 종속성 그래프와 클래스 계층구조 그래프로 구성된다.

여기에 제안된 알고리즘은 쉽게 확장이 가능하며 프로그램이 점진적으로 개발되는 경우에 유리하게 사용될 수 있다.

ABSTRACT

When general software system wants to need continuous Dept of Software Engineering by discovery of new request and mistake, extension correction, and find connection command statement about variable of specification imperative sentence in some program, programmer analyzes program through practice orbit chase of program about value of input data.

This paper that see therefore applies object intention program Slicing to existent program Slicing method, and express practice orbit of program in object intention subordination graph and show process that become Slicing.

Proposed algorithm defenses crab extension is possible and can be used advantageously in case program is developed gradually as well as.

Ⅰ. 서 론

프로그램 슬라이싱은 기준 노드의 특정 변수의 값에 직, 간접 적으로 영향을 끼치는 프로그램 P에 있는 모든 명령들을 찾는 것이다. 그리고 객체지향 소프트웨어 개발은 독립적이고 확장성을 가진 소프트웨어 부품들간의 관련성을 통하여 새로운 소프트웨어 개발을 위한 방안으로 제기 되고 있으며, 또한 소프트웨어에 대한 일반적인 사용자의 요구를 충족시키기 위하여 많은 기능을 가지고 방대한 소프트웨어가 개발되고 있다. 그러나 대체적으로 사용자는 소프트웨어의 일반적인 기능만을 사용할 뿐 많은 기능을 가지고 있는 소프트웨어에서 사용하지 않고 불필요한 기능을 방치하면서 실행의 효율성을 떨어뜨리는 경우가 많이 발생한다. 그러므로 프로그램 유지보수 작업의 일환으로 현행 소프트웨어 시스템에 대해 지속적으로 프로그램을 개발하고 확장 할 때 기존의 컴포넌트을 재사용 하거나 효율적으로 프로그램을 테스팅 할수 있는 기법을 이용할 수 있다면 소프트웨어의 품질은 매우 향상될 것이다. 이를 위해 기존의 프로그램에 대해 정확히 분석하여 필요로 하는 부분들을 정확하고 효율적인 프로그램 슬라이싱 기법을 적용한다면, 프로그램의 전체 크기를 줄이면서 실행의 효율성을 증대하고, 불필요한 기능을 줄이면서 프로그램이 명확해지면서, 디버그나 분석이 용이해진다. 또한 최적화된 프로그램을 이용함으로 프로그램의 로딩시간을 감소시킬 수 있다.

따라서 본 논문에서는 객체지향 언어인 C++언어를 대상으로 기본 설계단위가 되는 모듈을 기준으로 종속 그래프와 모듈 사이의 파라 메터들을 표현하고 불필요한 클래스정보의 멤버함수 및 멤버 데이터를 제거하는 내용을 중심으로 슬라이

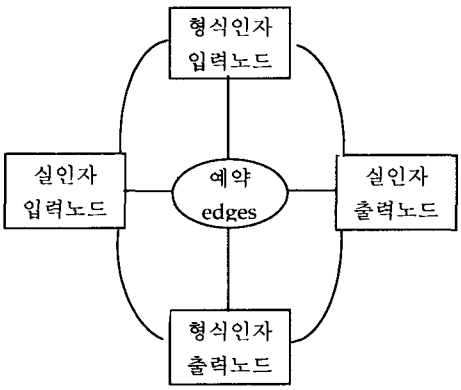
싱 관계를 고찰하고자 한다.

II. 관련 연구

프로그램 슬라이싱은 특정한 계산에 필요한 정보를 신속하게 생성하기 위한 방법을 개발하는 것이다. Weiser는 처음으로 슬라이스 개념을 소개하였다. 그의 슬라이싱 알고리즘은 제어 흐름 그래프 상에서 데이터 흐름 분석을 이용한다. 프로그램 P의 슬라이스는 슬라이싱 기준 C =< s, V >과 관계있다. 여기서 s와 V는 P에 있는 프로그램 포인터와 변수 집합이다. Ottenstein등은 그래프 도달성 알고리즘을 이용하여 프로그램을 프로그램 종속성 그래프로 표현한 후 이것의 종속성 에지를 순회 함으로써 효율적으로 슬라이스를 계산할수 있음을 보였다. 이 절에서는 절차간 슬라이싱과 객체지향 프로그램의 슬라이싱 방법 및 프로시쥬어 정의 그래프을 소개하고자 한다.

2.1. 절차간 슬라이싱

슬라이싱에 있어서 객체 지향 언어의 특성이 클래스 단위 구성을 가지며 이 클래스 단위는 많은 멤버 수와 멤버 데이터로 이루어지고 있기 때문에 이들간의 관계를 표현하기 위해 절차를 슬라이싱에 이용된다. 시스템 종속성 그래프는 프로그램 종속성 그래프들의 모임으로 호출되는 프로시쥬어의 종속성 그래프를 포함한다. 시스템 종속성 그래프는 하나의 메인 프로시쥬어와 호출되는 프로시쥬어로 구성된 완전한 프로그램을 가정한다. 인자의 전달 방식이 값-결과로 전달된다고 가정 하여 기술하면 (그림 1)과 같다.



(그림 1) 인자전달 표현

시스템 종속성 그래프는 프로시쥬어의 호출을 표현하기 위하여 새로운 노드와 에지를 추가 하여 종속성 그래프를 연결한다. 프로시쥬어의 호출은 호출 노드로 표현되며 인자 전달을 표현하기 위해 네가지 종류의 인자 노드가 존재한다. 인자 흐름 그래프는 호출 부분에 실인자-입력노드와 실인자 출력 노드를 표현하며, 이들은 호출 노드에 제어 종속적이다. 호출되는 부분에는 형식인자-입력 노드와 형식인자-출력 노드를 표현하며, 이들 프로시쥬어의 입력 노드에 제어 종속적이다. 실인자-입력노드와 형식인자-입력 노드는 모든 인자에 대해서 각각 존재하며, 실인자-출력노드와 형식인자-출력 노드는 호출 결과에 수정되어지는 인자에 대해서만 존재한다. 시스템 종속성 그래프는 요약 에지라는 새로운 에지를 갖는다. 이것은 호출 부분에 있는 실인자-입력 노드로부터 실인자-출력 노드로의 에지이며 호출로 인한 이행 종속성을 표현한다. 직관적으로 인자 x의 초기 값이 인자 y의 최종값을 정의 하는데 참조되었으며, x의 실인자-입력노드로부터 y의 실인자-출력노드로 향하는 요약 에지가 존재한다.

2.2. 객체지향 프로그램 슬라이싱

Harrold 등의 연구가 절차간 프로그램을 표현하고 이것에 대한 슬라이싱 방법을 연구한 것인데 반에 Larsen 등의 연구는 Harrold등의 연구를 토대로 프로그램 표현을 객체지향 프로그램을 지원하도록 확장하고 여기에 슬라이싱을 적용한 것이다.

Harrold 등의 연구에서 슬라이싱 기준은 <p,x>로 p는 프로그램 문장이며 x는 p에서 참조되거나 정의 되는 변수이다. 객체지향 소프트웨어의 개발자는 객체 상태의 정의와 참조를 위한 인터페이스를 제공하여 사용자로 하여금 객체의 상태를 규정짓는 데이터 멤버들을 직접 다루지 않게 하고 있다. 따라서 객체지향 소프트웨어에서 단순이 값을 리턴하는 메소드의 호출은 변수의 참조라고 볼수 있다. 이러한 점을 고려하여 Larsen등의 연구에서 슬라이싱 기준은 Harrold등의 연구와는 약간 다른데, 슬라이싱 기준은 <p,x>이며 여기서 p는 문장이고 x는 p에서 정의 또는 참조된 변수이거나 p에서 호출되는 메소드이다. Harrold 등의 연구와 마찬가지로 시스템 종속성 그래프가 호출 문맥을 위해 표시되는 요약에지를 가지므로 슬라이스를 계산하기 위해 2단계 알고리즘을 이용한다.

역방향 슬라이싱의 첫 번째 단계에서는 요약 에지를 통해 호출 노드와의 이행 종속을 추적할 수 있으며 두 번째 단계에서는 인자 출력에지를 따라 호출되는 메소드로 내려가는 방법이다

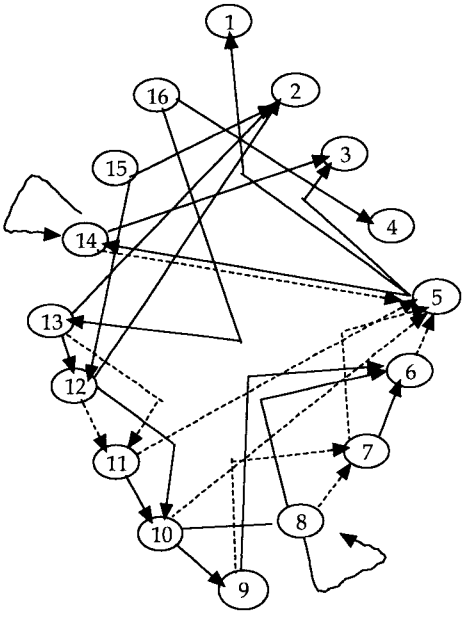
2.3. 프로시쥬어 정의 그래프

프로그램의 어느 정점 P에서 정의 되거나 사용되어지는 변수 V에 영향을 미치는 프로그램 문장의 집합으로 정의되고, 한 프로시쥬어에 대한 종속그래프를 프로그램 정의 그래프(PDG:Procedure Dependence Graph)라고 한다.

```
      int m, a, i, b, x, y, z;
1.    scanf("%d",&m);
2.    a = 0;
3.    i = 1;
4.    b = 2;
5.    while(i <= m){
6.     scanf("%d",&x);
7.     if(x <= 0)
8.        y=x+5;
      else
9.        y=x-5;
10.   z=y+4;
11.  if(z>0)
12      a=a+z;
       else
13.     b=a+5;
14.   i = i+1;
   }
15.   printf("%d\n", a);
16.   printf("%d\n", b);
}
```

(그림 2) 예제 프로그램1

(그림 3)은 (그림 2)의 알고리즘을 이용하여 정적 슬라이싱하는 과정을 나타내고 있다.



(그림 3) 프로그램 정의 그래프 표현

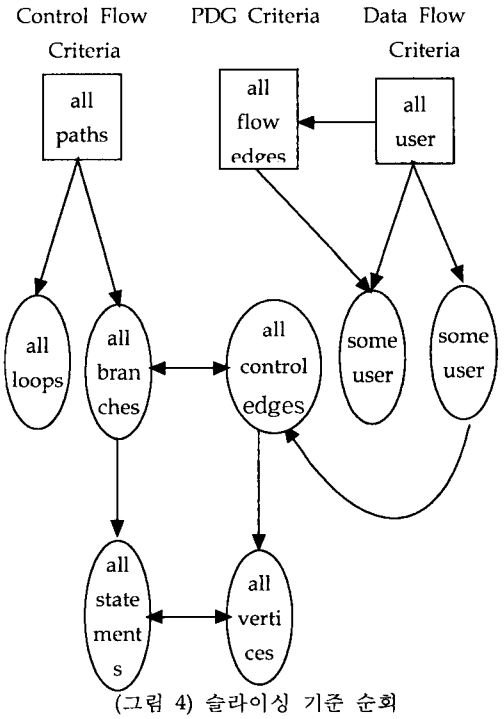
프로시쥬어 종속그래프는 자료 흐름 분석과 제어흐름 분석을 통해 프로시쥬어 종속 그래프로 나타낼 수 있다.

프로시쥬어 종속 그래프는 각각의 명령문에 대해 한 개의 노드를 가진다. 노드들은 간선들로 연결되며 간선에는 두 가지 종류 즉 자료 정의 간선과 제어 정의 간선이 있다. 정점 vi부터 정점 vj까지 자료 종속 간선은 정점 vi에서 실행된 결과는 정점vj에서 실행된 값에 직접 종속된다. 위의 알고리즘에 15의 변수 a에 대한 정적 슬라이스(2-12)가 된다.[1]

III. 객체지향 프로그램 슬라이싱

3.1. 슬라이싱의 기준 순회

프로그램 슬라이싱 기법은 변수의 의존 관계와 제어 흐름을 분석하여 어떤 주어진 출력 변수 들에 직접 간접적으로 영향을 미치는 프로그램 문장들을 추출해 내는 기법으로 1984년 Weiser에 의해 처음으로 소개되었다. 슬라이싱 기준<n,V>가 주어졌을 때 프로그램 문장 n이 수행되기 전에 변수 V에 영향을 미치는 문장은 무엇인가에 대한 답을 제공하는 것이다 이러한 기준 순회 슬라이싱중 프로시쥬어 슬라이싱은 (그림 3)에서 보여주었고 (그림 4)은 슬라이싱에 영향을 미치는 기준 제어 흐름, 기준 프로시쥬어 정의 그래프, 기준 데이터 흐름 3가지의 순회를 보여주고 있다.[8]



(그림 4) 슬라이싱 기준 순회

3.2. 클래스 정의 그래프

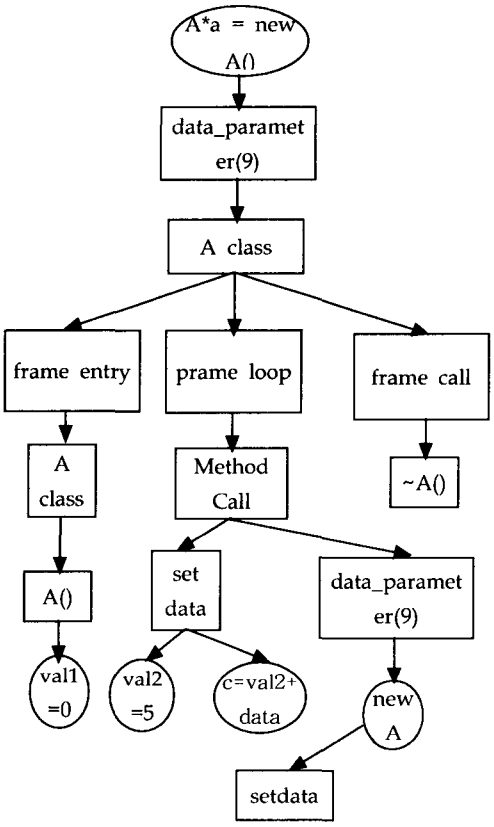
CDG(Cass Dependence Graph)는 단위 클래스 단위로 그래프가 구성되며 모듈화 되지 않고 각 클래스 단위로 표현되기 때문에 (그림 5)의 알고리즘을 (그림 6)과 같이 표현 할수 있다. 이러한 구성 형태는 몇 가지 문제점을 가지고 있다. 사용되는 문장은 한 클래스를 대상으로만 표현되기 때문에 설계단위가 되는 모듈에서의 클래스들의 관계가 적절하게 표현되지 못하고 문장의 호출에 의해 사용되는 다른 클래스 내의 문장들이 중복해서 포함됨으로 노드의 중복 가져온다.

객체지향 프로그램에서 프로그램 분석과 재사용을 쉽게 하기 위해서 프로그램의 각 클래스를 각각의 독립된 개체로 표현하여 객체지향 프로시쥬어 정의 그래프 정보 저장소에 저장함으로서 프로그램의 점진적인 개발과 갱신을 용이하게 할 수 있다. (그림 6)의 그래프는 클래스의 메소드에 대한 각각의 프로그램 정의 그래프로 구성되고 또한 메소드는 클래스 멤버를 간선으로 연결된다.

```
1:    class A{
2:    private : A* p; int val;
3:    public:
4:    A(){
5:     val1 = 0;}
6:    ~A(){}
7:    void setdata(int data){
8:    int val2 = 5;
9:     C = val2 + data;}
       };
10:   void data_parameter(){
11:   p = new A;
12:   p ->setdata(val1);
       }
      };
13:   main()
      {
14:   p *a;
15:   a = new A();
16:   p -> data_parameter();
17:   cout << a; "\n";
18:     return 0;
       }
```

(그림 5) 예제 프로그램2

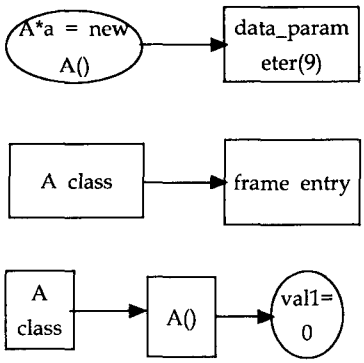
객체지향 소프트웨어에서 클래스는 new연산자, 선언등을 통해서 새로운 객체를 생성할수 있다. (그림 6)과 같이 A포인터 = new A()와 같은 문장을 사용하여 A클래스의 객체를 생성하여 클래스와 메인함수를 이용하여 각 메소드와 독립된 프로시쥬어의 종속성 그래프를 표현한다. 프로시어 정의 그래프는 각 문장 정점에 대하여 제어 종속성 간선과 각 문장의 변수들에 대한 자료 종속성 간선으로 표현하는 점에서는 단일 프로시쥬어의 프로그램 정의와 같으나 차이점을 들자면 클래스 객체의 생성을 표현하기 위한 객체생성 호출 간선 다형성 호출을 표현하기 위한 다형성 호출간선 클래스 메소드 호출을 표현하기위한 메소드 호출등의 간선은 해당 클래스의 헤더 정점과 호출정점간의 연결을 표현한다.



(그림 6) 클래스 정의 그래프

3.3. 클래스 정의 그래프를 이용한 슬라이싱

프로그램 슬라이싱은 두가지 목적을 가진다. 하나는 프로그램 포인터 P가 주어졌을 때 P에 영향을 주는부분 q을 찾아내는 것으로 q의 행위가 p의 행위에 영향을 준다는 사실을 밝히는 것이다. 또 다음은 하나의 프로그램 포인터 p가 주어졌을 때 p의 영향을 받는 부분 q을 밝히는 것으로 p의 행위가 q에 영향을 준다는 사실이다. 제안된 프로그램에서는 이러한 관계를을 표현하여 클래스 계층 그래프에 정점 사이의 변이 클래스 간의 관계를 나타내므로 정점간의 도달 가능성을 파악 함으로서 클래스간의 관계를 알 수 있다. 다음 (그림 7)은 메인 프로그램를 이용하여 클래스 정의 그래프를 기준으로 하여 프로그램 슬라이싱 되는 과정을 보인다.



(그림 7) 슬라이싱 표현

이렇게 구성된 객체지향 슬라이싱 알고리즘을 이용하여 표현하면 파리메터 아웃 에지를 제외한 모든 에지를 이용하여 후방향 검색을 통하면서 도달 가능 한 정점을 찾는다. 이러한 슬라이싱 진행과정을 표현 하기위해 (그림 7)과 같이 슬라이스 대상 위의 알고리즘 (그림 6) 명령문 16을 적용 시키면 (7-8-9-10-11-12)과 같이 슬라이싱 된다. 이렇게 슬라이싱 되는 과정을 살펴보면 정적 슬라이스는 주어진 기준 변수에 영향을 끼치는 모든 노드를 찾는 것이고, 객체지향 슬라이스는 주어진 프로그램 입력에 대해 발생된 변수 값에 실제적으로 영향을 주는 명령문들로 구성되어 있다. 그러므로 어떤 시험 사례를 통해 프로그램을 분석하는 디버깅 분야에서는 객체지향 슬라이싱이 정적 슬라이싱 보다 유용하게 사용될 수 있을 뿐만 아니라 클래스 정의 그래프를 이용하면 객체의 확장이 용이해지고 정점과 간선들을 감소시킬 수 있다.

IV. 결 론

지금까지 연구되어 왔던 고전적인 슬라이싱이나 다른 변형된 슬라이싱은 출력 데이터를 선택하고 그와 관련된 프로그램 문장을 포함하는데 초점을 두었다. 이러한 슬라이싱은 주로 오류 검출, 유지보수, 유연한 테스팅을 위한 연구로 이루어져 왔다. 본 논문에서는 후방향 및 전방향을 이용하여 객체지향 프로그램의 분석 정보를 표현할 수 있는 클래스 정의 그래프와 실재 예제프로그램을 보이고 슬라이싱되는 과정을 표현하여 분석 과정을 보임으로서 객체지향 프로그램 슬라이스가 정적슬라이싱보다 더 효과적임을 보였다. 앞으로 연구과제로는 기본 조작수를 최소화 하기 위한 클래스 계층 구조 재구성의 최적화 문제에 집중 되어야 할 것이다.

참고문헌

[1] G.B.Mund, R. Mall and S. Sarkar, An efficient dynamic program slicing technique, Information and Software Technology Vol44, Issue2, pp124-125, 2002

[2] Sebastian Danicic, Mark Harman and Yoga Sivagurunathan, A parallel algorithm for static program slicing, Information Processing Letters Vo56, Issue6, pp310-311, 1995

[3] Gerardo Canfora, Aniello Cimitile and Andrea De Lucia, Conditioned program slicing, Information and Software Technology North-Holland Vol40, Issues11-12, 597-599, 1998

[4] Franz Wotawa, On the relationship between model-based debugging and program slicing, Artificial Intelligence Vol135, Issues1-2, pp128-129, 2002

[5] Margaret Ann Francel and Spencer Rugaber, The value of slicing while debugging, Science of Computer Programming Vol40, Issues 2-3, pp153-154, 2001

[6] Wan Kwon Lee, In Sang Chung, Gwang Sik Yoon and Yong Rae Kwon, Specification-based program slicing and its applications, Journal of Systems Architecture Vol47, Issue5, pp438-440, 2001

[7] Mark Harman and Keith Brian Gallagher, Program slicing, Information and Software Technology Vo40, Issues11-12, pp578-579, 1998

[8] David Binkley, The application of program slicing to regression testing, Information and Software Technology Vol40, Issues11-12, pp585, 1998

[9] David W. Binkley and James R. Lyle, Application of the Pointer State Subgraph to Static Program Slicing, Journal of Systems and Software Vol40, Issue1, pp23-24, 1998

[10] Mariam Kamkar, An Overview and Comparative Classification of Program Slicing Techniques, Journal of Systems and Software Vol 31, Issue3, 209-210, 1995