C# 프로그램의 정적 분할

강성관*, 고훈준**, 김기태***, 조선문***, 유원희***
*인하대학교 정보통신공학과
**경인여자대학 컴퓨터정보기술학부
***인하대학교 컴퓨터정보공학과

e-mail:kskk1111@empal.com

Static Slicing of C# Programs

Sung-Kwan Kang*, Hoon-Joon Kouh**, Ki-Tae Kim,
Sun-Moon Jo, Weon-Hee Yoo
*Dept. of Information Technology & Telecommunication Engineering, Inha-University
**Dept. of Computer Information Technology, Kyungin Women's College School
***Dept. of Computer Science & Engineering, Inha University

요 약

C# 언어로 작성된 프로그램에서 기존의 객체 지향 프로그램에서 이용하던 정적 분할 방법을 적용한다. 기존의 두 경로 그래프 도달 가능성 분할 알고리즘을 적용하였을 때 프로시져들 간의 전이적인 종속 관계를 표현하는 요약 간선만을 이용하면 두 번째 경로에서 역 추적 할 때 모호성이 발생한다. 이러한 모호성은 C#의 이벤트, 델리게이트(delegate)들과 메소드의 다형적 호출 관계에서 발생될 수 있다. 본 논문은 호출된 프로시져의 호출하는 문맥을 설명하기 위하여 호출 지점에서 요약 간선 및 경로 간선을 이용하여 C#에서 다형적 호출에 대한 시스템 종속성 그래프(system dependence graph)에 대한 새로운 표현을 제안한다. 이 방법은 다형적 호출에서 발생하는 모호성을 해결할 수 있다.

1. 서론

프로그램 분할(slicing)은 분할 기준(criterion)으로써 언급된 어떤 관심의 시점에서 계산되어진 값에 잠재적으로 영향을 미치는 프로그램의 부분들을 얻어내는 방법이다.

Horwitz, Reps와 Binkley는 분할을 계산하기 위하여 시스템 종속성 그래프, 상호절차적 프로그램 표현과 시스템 종속성 그래프를 이용한 두 경로 그래프 도달 가능성 분할 알고리즘을 적용하였다[1]. 이 알고리즘을 C#으로 작성된 프로그램에 적용하여 시스템 종속성 그래프를 구성하였을 때 이벤트, 델리게이트, 메소드 간의 다형적 호출 관계에서 모호성이 발생한다.

이러한 모호성은 C#에서의 상호 절차적 프로그램 표현을 위한 프로시져 종속성 그래프에서 호출된 프로시져의 호출 문맥을 표현하기 위하여 사용하는 요약 간선들을 표현할 때 발생한다.

본 논문에서는 효율적인 분할 알고리즘이 적용될 수 있도록 C#을 이용한 프로그램에 대하여 경로 간선 이라고 하는 새로운 개념이 추가된 시스템 종속성 그래프의 구성을 제안한다.

2. 분할의 개념

프로그램 분할은 프로그램의 제어 흐름과 자료 흐름에 기반을 둔 기술로 M.Weiser에 의해 처음으로 제안된 방법이다[2].

분할은 주로 프로그램의 이해와 분석, 프로그램의

유지 보수와 복잡성 측정 및 프로그램의 디버깅과 테스트를 할 때 검사해야 하는 프로그램 코드의 양을 줄이기 위한 방법 등으로 사용되고 있으며 현재 여러 분야에서 사용되고 있는 기술이다[3].

프로그램 시점 p와 변수 x와 관련한 프로그램의 분할은 시점p에서 x의 값에 영향을 미칠 수 있는 프로그램의 모든 문장과 술어부로써 구성된다. 따라서 프로그램 분할은 프로그램에서 0개 이상의 문장을 제거함으로써 얻을 수 있는 실행 가능한 프로그램으로 정의 될 수 있다[4].

분할은 일반적으로 정적 분할(static slicing)과 동적 분할(dynamic slicing)로 구분할 수 있으며, 정적 분할은 Weiser, Ottenstein, Horwitz에 의해 제안된 분할 방법을 말한다. Ottenstein은 자료 종속성 그래프와 제어 종속성 그래프를 모두 표현할 수 있는 프로그램 종속성 그래프(program dependence graph:PDG)를 정의하고, PDG에서 그래프 도달 가능성 문제(graph reachability problem)로 분할을 구하였다[5].

Horwitz는 프로그램 종속성 그래프를 확장하여 프로시져 사이의 종속성을 표현하는 시스템 종속성 그래프를 제시하고 이를 이용하여 프로시져 사이의 분할을 구하였다[6].

최근에는 Krishnaswamy가 객체지향 프로그램에서 프로그램 분할을 수행하기 위해서 시스템 종속성 그래프에 클래스 계층 그래프(class hierarchy graph:CHG)를 정의하여 적용한 객체지향 프로그램 종속성 그래프(object-orientedprogram dependence graph:OPDG)를 제안하였다[7]. 그리고 Larsen과 Harrold은 C++로 작성된 프로그램의 분할을 위해서 시스템 종속성 그래프에 클래스의 종속성을 고려한 클래스 종속성 그래프(class dependence graph : CLDG)를 이용한 분할을 소개하였다[8].

3. C# 에서의 다형적 호출에 대한 표현

프로시져 종속성 그래프의 정점들은 정점들이 간선들의 2가지 타입인 제어 종속 간선과 데이터 종속 간선으로써 연결된 주어진 프로시져의 문장과 술부에 대한 표현이다.

본 논문에서는 이벤트, 델리게이트, 메소드 간의 다형적 호출에서 호출 문맥을 보존하기 위하여 요약 간선을 이용한 두 경로 도달 가능성 알고리즘을 적용할 때 발생하는 모호성을 해결하고자 한다.

본 논문은 Horwitz, Reps와 Binkley가 분할을 계산하기 위하여 사용한 시스템 종속성 그래프, 상호절차적 프로그램 표현 그리고 시스템 종속성 그래프를 이용한 두 경로 도달가능성 분할 알고리즘을 적용하였다. 이 분할 알고리즘은 호출된 프로시져의 호출 문맥을 표현하기 위하여 호출 지점에서 요약 정보 즉, 요약 간선을 이용하기 때문에 이전의 알고리즘보다 더 정확한 상호절차적 분할을 계산한다.

하나의 이벤트와 몇 개의 관심있는 델리게이트들로 이루어진 불완전한 시스템에 대하여 델리게이트가 참조하고 있는 공용 메소드(public methods)에 대한 모든 가능한 호출을 시뮬레이션하는 프로시져 종속성 그래프를 구성한다.

이벤트는 객체지향 프로그램에서 굉장히 중요한 부분이다. 일반적인 개념은 어떤 이벤트가 일어날 때 통보를 받아야 하는 문장이 있다는 것이다. GUI 프로그래밍은 이러한 상황으로 가득 차 있다. 프로그램에서 사용자가 메뉴를 선택하고 버튼을 누르는 것이 바로 이벤트이다. 이벤트가 발생했을 때 해야 할 일을 정의한 메소드를 호출하게끔 연결시켜야 하는데 이 때 델리게이트를 이용한다. 이벤트가 발생하면 델리게이트 인스턴스의 참조값을 갖고 해당 메소드를 찾아간다.

기존의 자바에서도 이벤트와 델리게이트의 개념이 있는데 대부분의 이벤트는 GUI 관련 컴포넌트에서 발생한다. 자바에서는 GUI 관련 컴포넌트들을 AWT 패키지로 제공하고 있다. 그러므로, 프로그래머가 패키지내에 있는 프로그램들을 수정이나 편집할 필요가 없었다. 하지만, C#에서는 프로그래머가 임의로 이벤트를 정의하고 이벤트에 델리게이트의 인스턴스를 추가 및 삭제할 수 있다.

C#에서 델리게이트를 사용하려면 사용하고자 하는 델리게이트를 우선 정의해야 한다. 그리고 나서 정의한 델리게이트의 인스턴스를 생성하고 호출한다.

4. 델리게이트 인스턴스 생성 부분

위에서 서술한 바와 같이 델리게이트의 인스턴스는 메소드의 참조값을 가지고 있고 이벤트는 그 참조값을 여러 개 가지고 있다가 동시에 실행시킨다. 이때 델리게이트 인스턴스 생성 부분에 있어서 세가지 방식으로 다르게 표현될 수 있다.

델리게이트의 인스턴스를 생성하는 형식은 아래와 같이 new 키워드를 이용하여 할 수 있고 형식에서 호출될 메소드 이름의 위치에 다음과 같이 세가지 형태의 다른 메소드들이 올 수가 있다.

[형식]

델리게이트이름 인스턴스이름 = new 델리게이트이름(호출될 메소드이름) ; (예제) MyDelegate df = new MyDelegate(dm.func)

(1) 정적 메소드
(2) 클래스 인스턴스의 멤버 메소드
(3) 같은 형의 다른 델리게이트 인스턴스

다음 그림[4-1]에 나오는 예제 프로그램은 (3)번에서의 경우를 나타내는 프로그램이다. 그림[4-1]에서 각각의 호출 정점 C1,C3에는 호출된 메소드의 정점 E2,E3에 있는 formal_in(F1_in, F2_in)과 formal_out(F1_out,F2_out)과 결합하기 위한 actual_in정점(A1_in,A2_in)과 actual_out(A1_out, A2_out)정점들이 있다.

한 쌍의 형식적인 formal_in, actual_in 그리고 formal_out, actual_out 정점들은 각각 parameter_in 간선과 parameter_out 간선으로 연결된다. 요약 간선과 경로 간선은 actual_in 과 actual_out 정점들 사이에 일어나는 종속의 전이적인 흐름을 표현한다.

[그림 4-1]과 같이 델리게이트가 호출할 메소드 이름 대신에 같은 형의 다른 델리게이트 인스턴스를 넣어주고 분할 기준을 C2 문장의 변수 b라고 하자. 기존의 객체 지향 프로그램에서 사용하였던 두 경로 그래프 도달 가능성 알고리즘은 다음과 같다.

[알고리즘 4-1]두 경로 그래프 도달가능성 알고리즘

첫번째 경로
parameter_out 간선들을 제외하고 분할 기준점에서 모든 간선을 가로질러 뒷방향으로 추적하고 도달된 그 정점들을 표시한다.
두번째 경로
parameter_out 간선을 따라서 호출된 메소드(또는 프로시져) 안으로 내려가면서 모든 간선을 가로질러 첫번째 경로 동안에 표시된 모든 정점들로부터 뒷방향으로 추적하고 도달된 정점들을 표시한다.

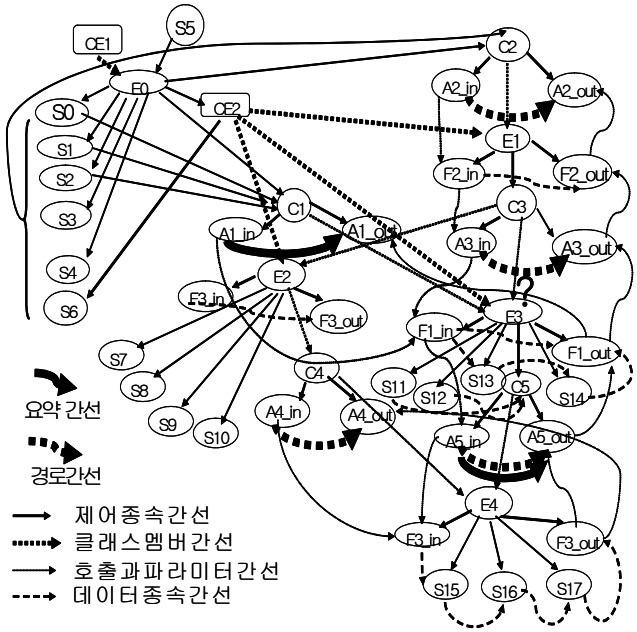
다음 그림[4-1]에서 각각의 코드는 종류별로 다음과 같이 분류할 수 있다.

S_n:문장, C_n:호출하는 메소드, E_n :호출된 메소드, CE_n : 클래스 진입점

```
 using System;
 namespace Oper
  {
S5: delegate void MyDel (double value) ;
CE2:class Oper
    {
S6:   public event MyDel myevent ;
E1:   public void func(double n)
      {
C3:       myevent(n) ;
      }
E2: public static void MulByTwo(double value)
    {
S7:     Oper op= new Oper () ;
S8:     double end_weight;
S9:     double result = value * 2 ;
C4:     end_weight= op.WeightByTwo(value) ;
S10:    Console.WriteLine(" 가중값: {0 }, 결과값: {1}" ,
                              end_weight,result) ;
    }
E3: public static void Square(double value)
    {
S11:  Oper  op= new Oper () ;
S12:  double end_weight;
S13:  double result = value * value ;
C5:   end_weight= op.WeightByTwo(value) ;
S14:  Console.WriteLine(" 가중값: {0 }, 결과값: {1
                  }" , end_weight,result) ;
    }
E4: public double WeightByTwo(double  k)
    {
S15:   double after_weight ;
S16:   after_weight = k * 2 +0.5 ;
S17:   return after_weight ;
    }
  }
CE1: class MainClass
     {
E0:   static void Main(string[] args)
      {
S0:     double  a= 3.5, b= 2.5 ;
S1:     Oper  ec = new Oper () ;
S2:     MyDel df = new  MyDel(Oper.Square) ;
C1:     df(a) ;
S3:     ec.myevent += new MyDel(Oper.MulByTwo);
S4:     ec.myevent += new MyDel(df) ;
C2:     ec.func(b) ;
      }
    }
}
```

[그림 4-1] 예제 프로그램

이 [알고리즘 4-1]을 아래 [그림 4-2]의 시스템 종속성 그래프에 적용하였을 때 두 번째 경로에서



[그림 4-2] 경로 간선을 도입한 [그림4-1] 프로그램의 시스템 종속성 그래프

정점 E3에서 두 정점 C1과 C3로 가게 되는데 정점 C1은 분할 기준에 있는 변수 b와는 관련이 없는 정점이므로 분할할 때 포함되지 말아야 하지만 E3와 호출 간선으로 연결되어 있어 포함이 되므로 정확한 분할이 이루어지지 않는다. 즉, 정점 E2에서 모호함이 발생한다. 이때 델리게이트의 인스턴스를 이용하여 메소드를 호출하는 부분(C1->E3->C5->E4)은 기존의 객체 지향 프로그램에서 이용하였던 요약 간선으로 표현하고 이벤트 호출하는 부분부터 델리게이트가 메소드를 호출하는 부분은 경로 간선으로 표현하여(C2->E1->C3->(E2,E3)->(C4,C5)->E4) 두 부분을 구분하여 프로시져간의 전이적인 종속 관계를 표현하면 호출하는 문장과 호출된 메소드 사이에서 발생하는 모호성을 해결할 수 있다.

5.결론 및 향후 연구 방향

C# 프로그램에 정적 분할을 적용하였을 때 이벤트와 멀티캐스트 델리게이트 그리고 메소드간의 다형적 호출에서 발생하는 모호성은 요약 간선과 경로 간선을 이용하여 전이적인 종속 관계를 구분하여 표현할 수 있었다. 하지만, 두 경로 도달 가능성 알고리즘을 적용하였을 때 모호성이 발생하는 부분에서 분할 기준으로 삼은 정점으로 추적하기 위한 알고리즘에 대한 연구가 필요하다.

참고문헌

[1] Horwitz SReps TBinkley DInterprocedural Slicing Using Dependence Graphs ACM Transactions on Programming Languages and Systemspages.

[2] M.Weiser, "Program Slicing." IEEE saction on Softwarengineering,Vol,Se-10,No.4,pp 52-357,July 1984

[3] M.Kamkar, *An Overview of Static and Dynamic Slicing*, Research Report, Dept. of Computer Science, Linkoping University, Sweden, 1991

[4] G.A. Venkatesh, "The Semantic Approach to Program Slicing, *"Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation PLDI '91, SIGPLAN Notices 26(6),* pp. 107-119, Toronto, Ontario, Canada, June 26-28 1991

[5] F.Tip, *"A survey of program slicing techniques," Journal of Programming Languages, Vol.3(3), pp.121-189,* Chapman and Hall, London, September 1995

[6] K. J. Ottenstein, and L. M. Ottenstein, "The Program Dependence Graph in a Software Development Environment, "Proceeding of the ACM SIGSOFT/SIGPLAN *Software Engineering Symposium on Practical Software Development Environments,SIGPLAN Notices Vol.19(5),* pp.177-184, April 1984

[7] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural Slicing Using Dependence Graphs," *Proceedings of the ACM SIGPLAN'88 Conference on Programming Language Design and Implemention* ,Atlanta, Georgia, June,1988. *SIGPLAN Notices Vol. 23(7),* pp.35-46, July 1988

[8] *A. Krishnaswamy, "Program Slicing : An Application of Object-Oriented Program Dependency Graphs,"* Technical Report TR94-108, Department of Computer Science, Clemson University, 1994