시스템 종속 그래프를 이용한 C# 프로그램의 객체 분할

강성관*
*인하대학교 정보통신공학과
e-mail : kskk1111@empal.com

Object Slicing of C# Programs Using System Dependence Graph

Sung-Kwan Kang*,
* Dept. of Information Technology & Telecommunication Engineering, Inha-University

요 약

프로그램 분할은 분할 기준으로써 언급된 어떤 관심의 시점에서 계산되어진 값에 잠재적으로 영향을 미치는 프로그램의 부분들을 얻어내는 방법이다. 객체 지향 프로그램의 분할(slicing)은 객체가 메소드를 호출할 때 한 객체의 모든 데이터 멤버들을 실매개변수들(actual parameters)로 전달함으로써 데이터 멤버들을 구별한다. 그러나, 실제적으로 데이터 멤버들의 일부분만이 메소드에서 사용되어진다. 또한, 기존의 분할 방법들은 한 클래스의 메소드들에 있는 문장들만을 분할하는 것이다. 클래스, 객체, 상속, 다형성, 동적 바인딩과 같은 객체 지향 프로그램의 독특한 특징들 때문에 문장 분할이 객체 지향 프로그램에 적용되는 것은 부적당하다. 본 논문에서는 객체 지향 언어중 가장 최근에 나왔으며 활용도가 높아지고 있는 C# 프로그램에 시스템 종속 그래프(System Dependence Graph)를 확장 및 적용한다.

1. 서론

프로그램 분할(slicing)은 디버깅, 복귀 테스팅, 프로그램 이해, 역공학과 같은 소프트웨어 유지보수의 여러 분야에 적용되고 있다[1]. 프로그램 분할은 프로그램 지점 p에서 정의되거나 사용되어진 프로그램 변수 v의 값에 영향을 미칠 수 있는 프로그램 문장과 술어의 집합이다[1, 2]. 어떤 프로그램을 분할할 때 표현식 <v, p>는 분할 기준으로서 알려진다. Horwitz는 시스템 종속 그래프(System Dependence Graph)와 프로시져 상호간의 분할을 계산하기 위하여 시스템 종속 그래프에서 두 단계 그래프 도달가능성 알고리즘을 개발했다[3]. Larsen과 Harrold, Tonella는 다양한 언어 특징을 표현하기 위하여 시스템 종속 그래프를 확장해 왔고 더 정교한 분할을 용이하게 하기 위하여 종속 그래프의 변화를 제안해왔다[4, 5].

그러나 클래스, 객체, 상속, 다형성, 동적 바인딩과 같은 객체 지향 개념은 명령형 언어 프로그램에 대하여 발전해 온 표현과 분석 기법이 객체 지향 프로그램에 적용되는 것을 부적당하게 만든다. Larsen 과 Harrold 는 객체 지향 프로그램의 특징들 중에서 몇 가지를 표현하기 위하여 시스템 종속 그래프를 변형하여 사용하였다[4]. 이러한 기존의 방법이 객체 지향 프로그램의 몇가지 특징을 표현하기 위한 기법들을 제공할 지라도 몇가지 면에서 분할 방법들이 개선될 수 있다. 본 논문에서는 객체 지향 C# 프로그램의 효율적인 분할을 위한 방법을 제안한다.

2. 시스템 종속 그래프를 이용한 분할

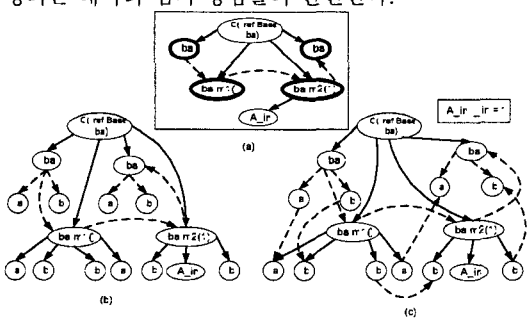
시스템 종속 그래프는 각 프로시져에 대하여 한 개의 프로시져 종속 그래프를 포함한다. 프로시져 종속 그래프는 정점들이 문장 또는 술어부 표현식인 그래프로서 프로시져를 표현한다[6]. 데이터 종속(data dependence) 간선은 문장 또는 표현식들 사이의 데이터의 흐름을 표현한다. 제어 종속 간선은 문장 또는 표현식의 실행이 종속되는 제어 조건을 표현한다[7]. 데이터 종속과 제어 종속은 프로그램의 CFG(Control Flow Graph)의 관점에서 정의된다[1]. CFG 는 프로그램에서 각 문장과 제어 술어부에 대한 노드를 포함한다. 분할 알고리즘은 두개의 단계로 이루어진다. 첫번째 단계 동안에, 알고리즘은 parameter_out 간선들을 제외하고 모든 간선들을 따라서 역방향으로 순행하고 도달된 정점들을

표시한다. 두번째 단계 동안에, 알고리즘은 호출과 parameter_in 간선들을 제외하고 모든 간선들을 따라서 첫번째 단계 동안에 표시된 모든 정점들로부터 역방향으로 순행하고 도달된 정점들을 표시한다. 분할은 첫번째와 두번째 단계 동안에 표시된 정점들의 합집합이다.

3. 시스템 종속 그래프 확장

클래스들을 표현하는 가능한 방법은 똑 같은 프로그램 지점에서 생성된 모든 객체들을 동등한 클래스로 그룹화하는 것이다. 그러므로, 똑 같은 프로그램 문장에 의하여 생성된 객체들의 집합을 표현하는 하나의 객체를 이용한다.

데이터 멤버들에 의하여 도입된 두개의 연속적인 메소드 호출 사이의 데이터 종속은 메소드 호출 지점에서 실매개변수들 사이의 데이터 종속으로써 표현한다. 더 정확한 데이터 종속을 계산하기 위하여 트리가 객체를 표현하기 위하여 사용될 때 사용 또는 정의는 데이터 멤버 정점들과 관련된다.



[그림 1] 시스템 종속 그래프의 확장

트리에 의해서 표현된 객체가 이용되면 트리에 있는 모든 데이터 멤버 정점은 데이터 멤버의 이용과 관련된다. 트리에 의하여 표현된 객체가 정의된다면 트리에 있는 모든 데이터 멤버 정점은 데이터 멤버의 정의와 관련된다. 제안하는 알고리즘은 매개변수 객체와 관련된 종속이 계산되어지는 방법을 보여준다.

4. 객체 분할(Object Slicing)

객체 지향 프로그램에서, 객체가 클래스로부터 사례화될 때 클래스의 데이터 멤버들과 메소드들은 그 객체에 대하여 사례화 된다. 그러나, 본 논문에서는 메소드의 모든 사례에 대하여 단 한가지 표현만을 이용하기 때문에 Horwitz가 제안한 것과 같은 분할 알고리즘에 의하여 확인된 한 메소드에 있는 문장들의 집합은 그 메소드의 다른 사례에 있는 문장들을 포함한다[6]. 디버깅과 프로그램 이해와 같은 작업에 대하여 분할주체는 한번에 하나의 객체에 관심을 집중하고 싶어한다. 이것을 하기 위하여 분할 기준에 영향을 미칠 수 있는 특정 객체의 메소드에 있는 문장들을 확인하기를 시도하며 이것을 **객체 분할(object slicing)**이라고 부른다.

객체 분할은 분할 기준에 영향을 미칠 수 있는 객체의 메소드들에 있는 문장들을 확인함으로써 더 정확한 분할하기 위해서, 분할 주체는 고려되어지는 객체와 관련된 시스템 종속 그래프의 부분을 단지 순행할 필요가 있다. 객체가 매개변수로서 사용될 때 객체를 트리로써 표현한다. 트리의 루트 노드는 객체를 표현한다. 루트의 자식 노드들은 객체의 데이터 멤버들을 표현한다. 트리의 간선들은 객체와 그것의 데이터 멤버들 사이의 데이터 종속을 표현한다.

p가 지점이고 v가 변수라고 하면 분할기준 <v, p, Object, Class>에 대하여 객체 O를 분할하기 위하여 먼저 <v, p, Object, Class>에 대한 시스템 종속 그래프로부터 분할을 얻는다. 그리고 나서, 분할 알고리즘에 의하여 표시된 O의 메시지-전달 호출지점을 확인한다. 이러한 호출 지점에 의하여 호출된 메소드들은 <v, p, Object, Class>에 영향을 미칠 수 있는 O의 메소드들 뿐이다. 다음에, 선택된 호출 지점에서의 호출 때문에 분할에 포함된 이러한 메소드들에 있는 문장들을 확인한다. 이렇게 확인된 문장들은 O의 분할이다.

분할 알고리즘에 의하여 표시된 O의 메시지-전달 호출 지점을 확인하기 위하여, O가 구성되어진 메소드 또는 프로시져에 있는 O의 객체 흐름 부그래프의 정점들을 검사한다. O가 또 하나의 메소드 또는 프로시져에게 매개변수로써 전달된다면 호출 지점에서 결합하는 간선을 순행하고 호출된 메소드 또는 프로시져에 있는 형식 매개변수 객체의 객체 흐름 부그래프를 계속 검사한다. 순행은 객체의 모든 호출지점이 검사될 때까지 계속한다.

종속 관계를 분석할 때 클래스의 프로시져 종속 그래프는 모든 클래스의 사례들에 의하여 공유된다. 분할 기준이 주어졌을 때 분할 알고리즘에 의하여 확인된 메소드에 있는 문장들은 그 메소드의 다른 사례들에 있는 문장들을 포함한다.

객체 지향 프로그램에서 분할을 지켜볼 수 있는 객체 분할(object slicing)을 이용한다면 객체 분할에 대한 기법은 상당히 효율적이다. 단지 전체 프로그램을 일

단 분할하는 것을 필요로 한다. 그리고 나서 객체를 분할의 유효성과 효율을 평가하는 것을 고려한다.

5. 결론 및 향후 연구 방향

본 논문은 Larsen과 Harrold가 객체 지향 프로그램에 대하여 이용했던 시스템 종속 그래프를 객체 분할을 위하여 확장하였다. 객체명을 루트로하고 객체가 갖는 멤버변수들을 자식노드 정점으로하는 트리로써 표현하였다. 확장된 시스템 종속 그래프는 C#과 JAVA와 같은 언어에 존재하는 객체를 포함하여 객체 지향 프로그램의 특징을 표현한다. 본 논문에서는 분할 기준에 영향을 미칠 수 있는 객체의 메소드들에 있는 문장들만을 확인하는 Zhenqiang의 객체 분할(object slicing)의 문제점을 지적하였다. 확장된 시스템 종속 그래프를 기초로 하는 객체 분할은 여러 장점이 있다.

첫째, 다른 객체들에 대한 데이터 멤버들을 구별하고 매개변수로써 사용된 객체들이 가진 데이터 멤버들 사이의 가짜 데이터 종속 관계를 막을 수 있다. 둘째, 제안한 시스템 종속 그래프 구성 알고리즘이 더 큰 프로그램에서 더 효율적인 분할이 가능할 수 있도록 하는데 이전 방법보다 더 효율적이다. 셋째, 어떤 프로그램에 대하여 제안한 기법이 객체 대 객체로 분할과정에 있는 문장을 사용자가 검사하도록 한다는 것이다.

확장된 시스템 종속 그래프와 객체 분할은 다른 객체들에 속해있는 데이터 멤버들을 구별한다. 제안한 방법은 객체가 매개변수로써 사용될 때 조차도 데이터 멤버들을 표현하고 호출 지점과 매개변수에서 다형성의 영향을 고려한다. 본 논문에서는 또한 객체 지향 프로그램에 대하여 확장된 시스템 종속 그래프를 구성하기 위하여 효율적인 알고리즘을 제안하였다. 객체 분할은 사용자가 분할 기준에서 특정 객체의 영향을 검사하는 것을 가능하게 함으로써 더 큰 규모의 프로그램에 대하여 디버깅과 프로그램 이해에 대한 더 좋은 지원을 제공한다.

향후 연구에서는 제안한 시스템 종속 그래프 구성 알고리즘의 실행 비용을 줄이는 방법의 연구가 필요하다. 또한, 객체 지향 프로그램을 표현하는 시스템 종속 그래프와 두 단계 그래프 도달가능성 알고리즘을 이용하여 객체 지향 프로그램에서 정적 분할의 유효성과 효율을 평가하는 것을 고려해 본다.

참 고 문 헌

[1]Frank Tip, "A survey of program slicing techniques", Journal of Programming Languages, Vol.3(3), September 1995, pp.121-189

[2]Mark Weiser, "Program Slicing." IEEE transaction on Software Engineering, Vol,Se-10, No.4, July 1984, pp. 52-357

[3]Susan Horwitz, Thomas Reps,and David Binkley,"Interprocedural Slicing Using Dependence Graphs", ACM Transactions on Programming Languages and Systems, 1990, 12(1), pp. 26-60

[4]Loren Larsen and Mary Jean Harrold, " Slicing Object-Oriented Software" , Department of Computer Science, Clemson University, ICSE' 96, Mar. 1996, pp. 495-505

[5]Paolo Tonella, et al. " Flow in-sensitive C++ pointers and polymorphism analysis and its application to slicing." In 19th International Conference on Software Engineering, May 1997, pp. 433-443

[6]Zhenqiang Chen and Baowen Xu, " Slicing Object-Oriented Java Programs" , ACM SIGPLAN Notices, Vol.36(4), April 2001, pp. 33-40

[7]Karl J. Ottenstein , Linda M. Ottenstein , " The Program Dependence Graph In a Software Development Environment" , In Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments(1984), SIGPLAN Notices 19(5), pp.177-184.

[8]David Byers, Marian Kamkar "Static Slcing and Parametric Polymorphism", Department of Computer Science, Linköpings universitet, Sweden, Source Code Analysis and Manipulation, 2001. Proceedings. First IEEE International Workshop on 10 Nov. 2001, pp.179 - 184