C# 프로그램의 정적 분할에서의 모호성 해결

강성관*, 고훈준**, 김기태, 조선문, 유원희
*인하대학교 정보통신공학과
**경인여자대학 컴퓨터정보기술학부
인하대학교 컴퓨터정보공학과
e-mail : kskk1111@empal.com

solution of ambiguity in 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 프로그램을 정적으로 분할할 때 기존의 객체 지향 프로그램에 이용하던 방법을 일반적으로 적용할 수 있다. 그러나, 기존의 두 정로 그래프 도달 가능성 분할 알고리즘을 적용하였을 때 프로시져들 간의 전이적인 종속 관계를 표현하는 요약 간선만을 이용하면 두 번째 경로에서 역추적 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, Reps 와 Binkley 가 제안한 두 경로 그래프 도달 가능성 분할 알고리즘을 [1] C#으로 작성된 프로그램에 적용하여 시스템 종속성 그래프를 구성하였을 때 이벤트, 델리게이트, 메소드 간의 다형적 호출 관계에서 모호성이 발생한다[1].

본 논문에서는 C# 프로그램에서 이벤트, 델리게이트, 메소드 간의 다형적 호출에서 호출 문맥을 보존하기 위하여 요약 간선만을 이용하여 두 경로 그래프 도달 가능[1]성 알고리즘을 적용할 때 발생하는 모호성을 해결하고자 한다.

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 시점 p 와 변수 x 와 관련한 분할은 시점 p 에서 x 의 값에 영향을 미칠 수 있는 모든 문장과 술어부로써 구성되며 실행 가능한 프로그램으로 정의 될 수 있다[2]. 시스템 종속성 그래프에서 데이터의 흐름을 나타내는 간선들은 다음과 같이 두 가지로 나타낼 수 있다[3].

제어 종속 간선(Control Dependency Edge)은 보통 지위 우월(post-dominance)의 관점에서 정의되며 노드 j 가 i 에 대하여 제어 종속일 조건은 다음과

[1] .본 연구는 한국과학재단 목적기초연구(R05-2004-000-11694-0)지원으로 수행되었음.

같다[4].

1)노드 i 부터 STOP 까지 모든 경로가 노드 j 를 통하여 지나간다면 CFG 에 있는 노드 j 는 노드 i 에 대하여 지위 우월(post-dominated)이라고 한다.

2)경로 P 에 있는 어떤 k 가(k .≠ i) j 에 대하여 post-dominated 도록 하는 i 부터 j 까지 경로 P 가 존재한다. 제어 종속 간선은 SDG 에서 호출 지점과 호출된 프로시져 사이의 직접적인 종속을 표현할 때 사용한다.

데이터 종속 간선(Data Dependence Edge)의 개념은 다음과 같은 변수 x 가 존재한다면 노드 j 는 노드 i 에 대하여 데이터 종속이다.

1)변수 x 가 노드 i 부터 j 에서 정의되어지고 참조되어진다.

2)노드 i 부터 j 까지 x 의 정의에 간섭하지 않는 경로가 하나 존재한다[5].

3.C# 에서의 다형적 호출에 대한 표현

시스템 종속성 그래프에서 하나의 이벤트가 발생하여 멀티캐스트 델리게이트에 의해 여러 메소드가 호출되는 정점에서 파라미터의 전이적인 종속 관계를 표현하기 위하여 새로운 간선인 경로(Path Edge)간선을 도입한다.경로 간선(path Edge)을 사용하는 경우는 하나의 델리게이트가 여러 개의 메소드(프로시져)를 호출할 때이다.

두 경로 그래프 도달 가능성 알고리즘을 적용하여 슬라이싱의 역추적 과정에서 요약 간선만을 사용할 경우 공통으로 호출된 호출 지점에서 모호성이 발생하므로 이 모호성을 해결하기 위하여 경로 간선을 도입한다.

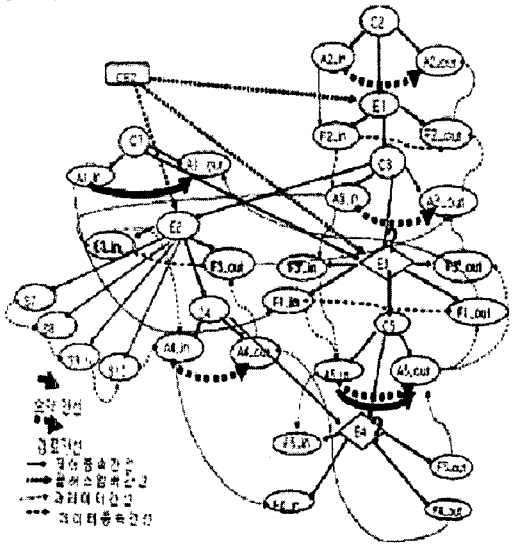
4. 델리게이트 인스턴스 생성 부분

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

[그림 4-1]의 시스템 종속성 그래프에서 분할 기준을 C2 문장의 변수 b 라 하고 두 정로 그래프 도달 가능성 알고리즘을 적용하였을 때 두 번째 경로에서 정점 E3 에서 두 정점 C1 과 C3 로 가게 된다. 그런데, 정점 C1 은 분할 기준에 있는 변수 b 와는 관련이 없는 정점이므로 분할할 때 포함되지 말아야 하지만 E3 와 제어 종속 간선으로 연결되어 있어 포함이 되므로 정확한 분할이 이루어지지 않는다.



[그림 4-1]경로 간선을 도입한 프로그램의 시스템 종속성 그래프

즉, 정점 E3 에서 모호성이 발생한다. 마찬가지로 C1 문장의 변수 a 를 분할 기준으로 하면 역추적할 때 정점 E4 에서 모호성이 발생한다. 이 모호성을 해결하기 위한 알고리즘은 다음과 같다.

<모호성을 해결하는 알고리즘 >

Input : Program P,and slicing criterion $< s_0, V >$

s_0 :슬라이싱 기준이 되는 하나의 문장

V : s_0 에서 정의되거나 사용되어진 모든 변수들의 집합

Output : The slice Program on slicing criterion $< s_0, V >$: SP

```
Procedure MarkVerticesOfSlice(G,S)
declare
G : a system dependence graph
S : sets of vertices in G
AllEdges , Kinds : a set of kinds of edges
begin
/*Pass1: Except for parameter_out Edges and traverses
back along all edges from slice criterion */
 SP:= φ
MarkReachingVertices(G,S, AllEdges-{parameter_out
        SP=SP∪{(y,v)},trace(y,v)
     od
 od
```

```
 edges} )
/* Pass 2: The algorithm descends into called
methods(or procedures) along the parameter_out edges
*/
MarkReachingVertices(G,S,AllEdges)
end

procedure MarkReachingVertices(G,V,Kinds )
declare
   G: a system dependence graph
   V : a set of vertices in G
   x,v,w,a,b : vertices in V
   E_n : n번 메서드
   WorkList : a set of Edges to be sliced in G
CM(Common Method):CM = C_i I C_j (i,j = 0,1,2,3,…)
C_i : i 번째 호출 , C_j : j 번째 호출
procedure trace (a,b)
  begin
  if (b->a) ∉ WorkList then insert {(a,b)} into
     WorkList fi
  end
begin
 WorkList := φ
 while V ≠ φ    do
     Select and remove a vertex v from V
     Mark v
     for each vertex w that is a predecessor of v
         in G such that there is an edge w-> v is
         in Kinds do
         insert (w,v) into
         SP=SP Y {(w,v)} , trace(v,w) od
    switch v
      case v∈ ActualOutVertices(G) :
        for each x such that (x->v) ∈
            parameter_outEdges do insert (x,v) into
            SP=SP Y {(x,v)} , trace( v,x)
            od
      end case
      case v ∈ FormalOutVetices(G) :
       for each
    C_i ∈Callers(Proc(w)) do
if(n(FormalOutVertices( E_n ))>=2) //CM(Common Method)
    let x=CorrespondingActualIn( C_i ,w)
      y=CorrespondingActualOut( C_i ,w) in G
      insert (v,y) into
      SP=SP Y {(v,y)} , trace( y,v )
    end let
  else
   for each y such that (v->y) ∈
            parameter_outEdges
      do
         insert (y,v)  into
end case
default:
   for each x such that (x → v)
        ∈ (Data Dependency Edges(G)
             Y Control Edges(G))
        do
           insert (x,v) into
           SP=SP Y {(x,v)},trace(v,x)
        od
   end case
  end switch
  return(SP)
 od
end
```

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 표현하므로써 해결할 수 있었다. 앞으로,본 논문에서 제안한 알고리즘에 대한 실험적인 검증 작업이 필요하다.

참고문헌

[1]S.Horwitz,T.Reps,and D.Binkley,"Interprocedu ral Slicing Using Dependence Graphs, " ACM Transactions on Programming Languages and Systems pages.

[2]M.Weiser, "Program Slicing." IEEE saction on Softwarengineering,Vol,Se-10,No.4,pp 5 2-357,July 1984

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

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

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