객체 지향 언어를 위한 점진 평가 방법 분석

한정란
협성대학교 경영정보학과
e-mail:jlhan@uhs.ac.kr

Analysis of Incremental Evaluation Technique For Object Oriented Language

Junglan Han
Dept of Management and Information System, Hyupsung University

요 약

프로그램의 생산성을 향상시키기 위해 프로그램 개발 단계에서 소요되는 비용을 최소화하려는 목적으로 점진 평가를 사용하고 있다. 점진 평가는 전체 프로그램을 다시 평가하는 대신 수정한 부분과 그 부분에 영향 받는 부분만을 다시 평가하는 방법이다.

본 논문에서는 기존의 종속 차트(dependency chart)를 확장하여 객체 지향언어인 자바 같은 언어에서 점진 평가를 수행할 수 있도록 확장된 종속 차트를 제시한다. 객체 지향언어에서 점진 평가를 수행하는 알고리즘을 제시하고 실험을 통해 점진 평가의 효율성을 분석한다.

1. 서론

점진 평가는 프로그램을 수정할 경우, 전체 프로그램을 다시 평가하는 대신 수정된 부분과 그 부분에 의해 영향 받게 될 부분을 분석하여 수정된 부분과 그 부분에 영향 받는 부분만을 다시 평가하는 방법이다. 프로그램의 생산성을 향상시키기 위해 프로그램 개발 단계에서 소요되는 비용을 최소화하려는 연구가 필요하고 이러한 연구들이 다양하게 진행되고 있다. 프로그램의 일부분이 수정될 때마다 전체 프로그램을 다시 평가하는 것은 소요되는 시간과 공간적인 측면에서 비효율적이라 할 수 있다. 점진 평가 방법은 전체 프로그램을 다시 평가하지 않고 수정된 부분과 그 부분에 영향 받는 부분만 평가하기 때문에, 프로그램 개발 환경의 실행 효율성 측면에서 고려해 볼 때 매우 중요하다.

점진 평가를 수행하기위해, 수정된 부분과 그 부분에 영향을 받아 변경되는 부분을 찾아내어 이 부분만을 다시 평가해야하고 이를 위해 점진 평가 방법[1]을 사용한다. 점진 평가를 수행하는 많은 연구들이 진행되었고 이러한 연구들 중에서 속성 문법을 사용한 기존의 연구에서는 수정된 부분에서 생긴 변화, 즉 변수 값이 변경될 때 그 변수를 사용한 다른 변수의 값도 바뀌는 영향 받는 변수들을 찾아내는 변화 파급(change propagation) 과정을 통해 점진 속성 평가(incremental attribute evaluation)를 수행한다. 프로그램에서 사용된 속성들 간의 종속성을 종속 그래프(dependency graph)로 나타내어 수정된 부분에서 생긴 변화를 다른 부분으로 파급(propagation)시키는 과정이 아주 복잡하게 진행된다[1].

복잡한 변화 파급 과정을 단순하게 하기 위해서 본 연구에서는 동적 의미 구조에 직접적으로 영향을 주는 변수의 값을 나타내는 속성들을 중심으로 속성들 간의 종속성(dependency)을 나타내어 종속 차트를 작성하고 작성된 차트와 속성 값을 고려하여 효과적인 평가를 수행한다.

수식에 나오는 변수들에 속한 속성들 중 변수의 실제 값을 나타내는 속성은 그 변수 값이 변함에 따라 프로그램의 의미 구조에 직접적으로 영향을 주게 된다. 본 논문에서는 변수의 값을 나타내는 속성을 중심으로 속성간의 종속성을 고려하고 객체지향언어를 처리하기 위한 확장된 종속 차트를 제시하고자 한다.

본 논문에서는 명령형 언어를 위해 제시된 종속 차트(dependency chart) 사용 기법[1]을 확장하여 객체 지향언어인 자바 같은 언어에서 점진 평가를 수행할 수 있도록 확장된 종속 차트를 제시한다. 객체 지향언어에서 점진 평가를 수행하는 알고리즘을 제시하고 실험을 통해 점진 평가의 효율성을 분석한다.

2. 확장된 종속 차트

종속차트는 종속성을 나타내기 위한 유향(directed) 그래프이다[1]. 자바와 같은 객체지향 언어를 처리하려면 객체와 클래스에 대한 정보가 필요하다. 각 클래스에 소속된 변수들에 대해 종속성을 나타내려면 기존의 종속차트를 확장하여 변수들이 소속된 객체에 대한 소속정보를 갖도록 필드를 추가해야 한다. 각 변수의 종속성을 나타내기 위해 선언된 클래스와 특정 클래스형을 갖는 객체들의 리스트를 나타내고 그 변수가 속한 메서드 이름을 표시하고 다음에 영향을 주는 변수를 종속링크로 연결한다. 확장된 종속차트에도 필수적인 변수들을 평가하기 위해 기존의 종속차트에 있던 조건 속성을 표시하는 CON 필드와 조건 속성의 형을 나타내는 FLAG 필드가 들어간다[1].

변수	소속 클래스	객체 리스트	소속 메서드	CON	FLAG	종속 링크

자바에서 동일한 이름의 변수가 각 클래스마다 나올 수 있으므로 각 변수를 구별하기 위해 변수가 선언된 소속 클래스를 표시해야한다. 같은 클래스형을 갖는 객체가 여러 개 생성될 수 있고 각 객체마다 객체 속성변수(멤버 변수)를 관리하므로 클래스이름과 객체 리스트를 배열로 저장하는 것이 필요하다. 소속 메서드는 지역 변수로 선언된 경우 메서드마다 동일한 변수를 선언할 수 있으므로 이를 구별하기 위해 표시한다. 종속 링크는 어떤 변수 값이 변함에 따라 값에 변화가 생길 수 있는 변수를 연결하는 링크이다. CON 필드는 조건 속성을 나타내는 것이고 FLAG 필드는 각각의 조건 속성의 형을 구분 짓기 위해 필요한 항목이다[1]. IF 조건 식일 경우 true(T)와 false(F)를 주고 게이트 속성일 경우 G 란 값을 주어 구분한다. 그 밖의 경우 I가 들어간다[1].

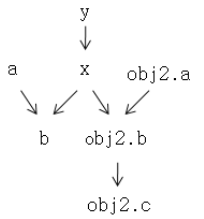
점진 평가의 효율성을 높이기 위해 종속 차트에 평가될 필요성이 있는지 그 여부를 나타내기 위한 정보가 들어가고 CON과 FLAG 필드에 따라 변수의 평가 여부를 결정하게 된다. 예로서 반복문에서 게이트 속성의 값이 거짓이라면 그 WHILE 루프 속에 값이 변경된 변수가 존재하더라도 그 값을 다시 계산할 필요가 없다. 즉, WHILE 루프의 조건이 거짓이라면 루프를 수행하지 않아도 되기 때문이다. 마찬가지로 IF 문의 조건식에 해당되는 값이 거짓이라면 참일 때 수행되는 명령문에 속한 변수가 변경되었다고 해도 그 값을 다시 계산할 필요가 없다[1].

```
public class Example {
  int a = 10, b = 5, c;      --- ①
   ...
  void Compare() {
    int x=1, y = 7;          --- ②
    x = y * 5;               --- ③
    if (x > 30)
      b = x * a;             --- ④
    ...
  }
  ...
  public static void main(String args[]) {
   Example obj1, obj2;
   obj1 = new Example();
   obj2 = new Example();
   ...
   obj2.Compare();           --- ⑤
   obj2.c = obj2.b - 5;      --- ⑥
}
```

(그림 1) 자바 프로그램

Example 클래스의 객체 obj1, obj2 는 객체 속성변수인 멤버 변수로 a, b, c를 갖는다. 종속 링크는 배정문의 우변에 있는 변수의 값이 변할 때 좌변의 변수 값이 변하게 되므로 이를 종속 링크에 반영한다. y의 값의 변화에 따라 x의 값이 변하므로 y → x 으로 링크를 생성한다[1].

변수의 종속성을 표현한 종속 링크만을 갖는 종속차트는 (그림 2)와 같다.



(그림 2) 종속차트의 종속링크

(그림 1)의 ⑤번 메서드 호출에 의해 obj2.a → obj2.b 의 링크가 존재하고 만일 obj1.Compare(); 로 변경되면 obj2.a → obj2.b 대신 obj1.a → obj1.b 로 변경된다.

종속차트의 다른 필드를 나타내면 <표 1>과 같다.

<표 1> 종속 차트의 필드 정보

변수	소속 클래스	객체 리스트	소속 메서드	CON	FLAG
x	Example	·	Compare	·	I
y	Example	·	Compare	·	I
a	Example	obj1, obj2	Compare	·	I
b	Example	obj1, obj2	Compare	x	T
obj2.a	Example	obj1, obj2	main	·	I
obj2.b	Example	obj1, obj2	main	x	T
obj2.c	Example	obj1, obj2	main	·	I

변수 b의 경우만 조건속성이 있는 if문에 속해 있고 조건식이 참일 경우만 평가하게 되므로 FLAG는 T이다. 나머지 변수들은 조건속성은 없으므로 CON 값은 없고 FLAG에는 I가 들어간다.

종속차트를 실제로 구현할 경우 각 변수들을 쉽게 찾아서 변화를 추적하기 위해 변수 테이블(VT)를 만들어 클래스와 관련된 소속정보를 넣는다.

4. 점진 평가 알고리즘

프로그램에서 올바른 실행 결과를 얻으려면 프로그램에 사용된 변수들의 값의 변화를 추적하여 계산하고 그 값을 출력하는 과정을 거치게 된다. 점진 평가에서는 변수 값을 수정하는 명령문이 나오면 종속차트의 종속링크를 통해 그 변수 값에 영향 받아 변하게 될 변수들을 추적하여 영향 받는 명령문들을 다시 평가한다. 따라서 변수 값이 변할 때 종속링크를 찾아 변수 값이 변하는 명령문을 다시 실행하여 점진 평가를 수행하면 전체 프로그램을 다시 평가하

지 않더라도 전체 프로그램을 평가한 것과 동일한 결과를 얻을 수 있다.

변수 값이 변할 때 점진 평가를 수행하기 위해 종속링크를 찾아 가는 것이 중요하고 자바와 같은 객체지향언어에서는 좀 더 복잡한 과정들을 거치게 된다.

변수의 값이 변경될 수 있는 경우를 (그림 1)의 프로그램을 중심으로 살펴보면 여러 가지가 있다.

1. 지역 변수의 변경

가장 간단한 변수 값의 변화는 지역 변수와 관련된 값의 변화이다.

②번 명령문에서 y 값을 15로 변경하면 x 값이 바뀌게 되므로 종속 링크를 따라가서 ③번 명령문 x = y * 5; 을 다시 평가하면 x 값을 변경할 수 있다. x 값이 바뀌면 ④번 명령문을 다시 평가하여 b 값도 변하게 된다. 두 번째 경우에 자세히 언급하겠지만 b는 객체 속성변수이고 Compare()에 포함된 명령문에 속한 변수이므로 ⑤번 명령문에서처럼 Compare()를 호출하는 객체 obj2의 b 값만 바뀌게 된다.

2. 객체 속성 변수의 변경

①번 명령문에서 객체 속성변수인 a, b, c의 값을 변경하면 a, b, c를 객체 속성변수로 갖는 객체 obj1, obj2의 a, b, c도 변하게 된다. 만일 a 값을 변경했을 때 ④번 명령문에서처럼 b 값도 변하는데 ④번 명령문은 x값이 30보다 클 경우만 평가하게 된다. 이를 위해 CON 필드와 FLAG 필드를 확인하여 ④번 명령문을 다시 평가하게 된다. 그런데 ⑤번 명령문에서처럼 Compare()를 호출하는 객체 obj2의 b 값만 바뀌게 된다. 즉 객체 obj1은 Compare()를 호출하지 않으므로 b 값은 변하지 않게 된다. 만일 ⑥번 명령문에서 수정이 일어나서 c 값이 변경될 경우 ⑥번 명령문을 다시 평가해야 하고 객체 obj2의 c 값이 변하게 된다.

3. 객체이름의 변경

⑥번 명령문 obj2.c = obj2.b - 5; 에서 obj2.c 대신 obj1.c로 객체 이름을 변경하면 해당 객체의 속성 변수에 변화가 생기게 되고 종속링크도 변경되어야 한다. 객체 obj1의 c값이 바뀌도록 ⑥번 명령문을 다시 평가해야한다.

그 외 클래스이름이나 메서드이름이 변경될 경우 해당 클래스나 메서드를 종속차트에서 찾아 소속 변수들의 클래스이름이나 메서드이름을 변경한다.

```
Algorithm IncrementalALGM(VT, DC, X);
  / let VT be the variable table in the dependency chart /
  / let DC be the dependency chart /
  / let X be changed variables /
 initialize mark flag;
 for each X do
   if X = variable in VT then
      begin
        P := X in DC;
        if not (mark of P) then
           begin
             P := P' successor using DFS;
             insert P into change-set;
             mark of P := true;
           end;
      end;
 od;
 for each b ∈ change-set do
    case FLAG in DC of
     G, T : if con.val then insert b into evaluated-set
     F    :
        if not (con.val) then insert b into evaluated-set
     I    : insert b into evaluated-set
     esac
 od;
 for each c ∈ evaluated-set do
  case c of
    local variable : evaluate c
    object member variable :
      find object of its methods call
      each associated object member variable evaluates
    object name :
      each associated object member variable evaluates
    class name :
      find each variable whose class field equals to class
      name in DC
      change its information in dependency chart
    method name :
      find each variable whose method field equals
      to method name in DC
      change its information in dependency chart
  esac
 od;
```

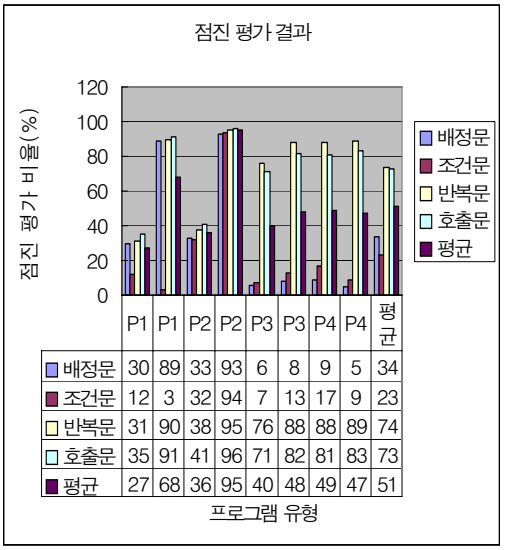
(그림 4) 점진 평가 알고리즘

종속 차트의 CON과 FLAG 값을 고려해 평가될 필요가 있는 필수 속성만을 다시 평가하여 최적의 평가를 수행한다. 변경된 변수에 의해 값이 변할 변수들을 모아 change-set을 구성하고 change-set에 있는 모든 변수에 대해 평가될 필요성이 있는지를 적절하게 검사해서 evaluated-set을 구성한다[1]. 각 변수의 CON 필드와 FLAG 필드를 조사하여 CON 필드에 있는 속성 값에 따라서 evaluated-set이 구성되고 evaluated-set에 속한 필수적인 변수만 평가하게 된다. evaluated-set에 속한 변수들 중 지역변수인지 객체 속성변수인지 파악하여 해당되는 평가를 수행하고 객체이름을 변경한 경우 해당 객체의 속성변수가 변하도록 평가한다.

5. 점진 평가의 효율성 실험

자바로 작성된 네 가지 유형의 프로그램에 대해 모의실험을 통해 점진 평가의 효율성을 분석한다. 합계와 평균을 계산하는 프로그램(P1), 최대값과 최소값을 찾는 프로그램

(P2), 급료를 계산하는 프로그램(P3), 배열로 선형 리스트를 구현하는 프로그램(P4)에 대해 실험하고 네 가지 명령문인 배정문, 조건문, 반복문, 메서드 호출문에 대해 변수 값을 수정하였고 그 수정으로 점진 평가하는 명령문의 비율을 비교하여 수정하는 명령문 유형에 따라 점진 평가의 효율성을 분석한다.



	P1	P1	P2	P2	P3	P3	P4	P4	평균
배정문	30	89	33	93	6	8	9	5	34
조건문	12	3	32	94	7	13	17	9	23
반복문	31	90	38	95	76	88	88	89	74
호출문	35	91	41	96	71	82	81	83	73
평균	27	68	36	95	40	48	49	47	51

(그림 5) 프로그램 점진 평가 결과

(그림 5)에서 P1과 P2 프로그램의 경우 반복문이 실행결과를 얻는 핵심구문이 되고 자료 수에 비례하여 반복적으로 실행하여 결과를 얻으므로 소량의 자료에 대한 평균 점진 실행비율은 각각 27%, 36% 인데 반해 대량의 자료에 대한 평균 점진 실행 비율은 68%, 95%로 자료 수에 비례하여 점진 평가 비율이 높아짐을 알 수 있다. P2의 비율이 더 높은 이유는 조건문이 프로그램의 결과를 얻는 핵심 명령문이기 때문에 조건문의 수정이 P1보다 더 큰 영향을 미치게 됨을 알 수 있다.

P3과 P4 프로그램의 경우도 핵심 구문이 반복문이지만 P1과 P2와는 달리 배정문에 있는 변수를 수정하더라도 반복문에 있는 변수에 영향을 주지 않는 경우도 있으므로 점진 실행 비율이 자료 수에 비례하여 영향을 미치지 않음을 알 수 있다. 프로그램의 명령문을 수정하는 유형 중 반복문의 수정은 점진 평가에 큰 영향을 주고 다른 명령문 유형의 수정이 반복문에 영향을 미치는 경우 점진 실행에 더 큰 영향을 주게 된다는 사실을 확인할 수 있다. 평균적으로 고려해볼 때 배정문의 수정이 반복문까지 영향을 주는 경우도 있지만 영향을 주지 않는 경우도 있으므로 51%의 점진 실행 비율이 나타난다. 프로그램 전체를 평가하는 것을 100%로 놓았을 때 점진 평가하는 것이 더 효율적이라 사실을 확인할 수 있다.

6. 결론

본 논문에서는 점진 평가를 수행하기 위해 명령형 언어를 위해 제시된 종속 차트(dependency chart) 사용 기법[1]을 확장하여 객체 지향언어인 자바 같은 언어에서 점진 평가를 수행할 수 있도록 확장된 종속 차트를 제시하였다. 제시된 종속 차트는 프로그램의 의미 구조에 직접적으로 영향을 주는 변수의 값을 나타내는 속성을 중심으로 종속성을 표시하여 변화를 추적하는 과정이 효율적으로 수행된다. 객체 지향언어에서 점진 평가를 수행하는 알고리즘을 제시하고 모의실험을 통해 점진 평가가 전체 프로그램을 다시 평가하는 것보다 효율적이라는 사실을 확인하였다.

네 가지 프로그램 유형에 대해서 실험하였고 네 가지 명령문인 배정문, 조건문, 반복문, 메서드 호출문에 대해 수정한 후 다시 평가될 명령문 수를 중심으로 점진 평가의 효율성을 검토해 본 결과 이들 명령문의 수정 유형들이 반복문에 영향을 줄 경우, 전체를 평가하는 것을 100%로 놓을 때 점진 평가 비율이 88% ~ 96%로 다소 효율성이 떨어지지만 네 가지 수정 유형을 평균적으로 고려할 때 51%의 실행 효율성을 나타내어 프로그램 전체를 평가하는 것보다 훨씬 효율적이라는 사실을 알 수 있다.

참고문헌

[1] 한정란 “작용 식 기반 점진 해석” Ph. D Thesis 이화여대 1999.

[2] 한정란 “객체 지향 언어를 위한 의미 명세” 인터넷정보학회 논문지, 제8권 5호, pp.35~43, 2007.

[3] 한정란, 최성 “동적 의미 분석에 의한 점진 해석기 구축” 인터넷정보학회 논문지, 제5권 6호 , pp.111~120 , 2004.

[4] Junglan Han, "Action Equations for Object Oriented Programming Language", In Proceeding of ICUT International Conference, 2007.

[5] T. Teitelbaum and T. Reps "The Cornell Program Synthesizes: A Syntax-directed Environment" Communication ACM Vol 24(9) pp. 563~573 1981.

[6] Raul Medina Rora and David S. Notkim "ALOE users' and implementers' guide" Carnegie-mellon Computer Science Depart. Research Report CS-81-145 1981.

[7] A. N. Habermann "The Gandalf Research project" Computer Science research Review Carnegie-Mellon University 1979.

[8] Charles N. F., Greg J. and Jon M. "An Introduction to Editor Allen Poe" Univ. Wisconsin-Madison TR 451 1981.

[9] John F. Beetem and Anne F. Beetem "Incremental Scanning and Parsing With Galaxy" IEEE Transactions on Software Engineering Vol. 17 No. 7 pp. 641~651 1991.

[10] Roger Hoover "Alphonse : Incremental Computation as a Programmer Abstraction" ACM SIGPLAN Notices pp. 261~272 1992.