확장된 종속차트를 사용한 효율적인 점진 평가 방법

An Efficient Incremental Evaluation Technique Using an Extended Dependency Chart

한 정 란*
Hun, Jung lan

요 약

프로그램의 생산성을 향상시키기 위해 프로그램 개발 단계에서 소요되는 비용을 최소화하려는 연구가 다양하게 진행되고 있다. 점진 평가는 프로그램을 수정할 경우, 전체 프로그램을 다시 평가하는 대신 수정한 부분과 그 부분에 영향 받는 부분만을 다시 평가하는 방법이다. 점진 평가 방법은 전체 프로그램을 다시 평가하지 않기 때문에, 프로그램 개발 환경의 실행 효율성 측면에서 고려해 볼 때 매우 중요하다.

본 논문에서는 명령형 언어를 위해 제시된 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 그 알고리즘의 정확성을 증명하고 실험을 통해 점진 평가의 효율성을 분석한다.

Abstract

There have been numerous researches in minimizing the total cost of program development in order to improve productivity of the programs. The incremental evaluation is the evaluation method of reevaluating only affected parts instead of reevaluating overall program when the program has been changed. Since the incremental evaluation method carries its advantage, the method itself is very important when considering the execution efficiency of the program developing environment.

This paper extends the dependency chart for an imperative language and presents the extended version of dependency chart which can be applied to the Object Oriented Programming Languages such 'Java'. This paper also presents the incremental evaluation algorithm for Object Oriented Programming Languages and proves its correctness, analyzing the efficiency of incremental evaluation by the simulation.

☞ Key words : incremental evaluation, dependency chart, incremental evaluation algorithm, dependency graph, object oriented programming languages

1. 서 론

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 평가 방법에 따라 프로그램을 번역하면 프로그램 전체를 다시 번역하지 않더라도 전체 프로그램을 번역한 것과 동일한 결과를 얻을

* 종신회원 : 협성대학교 경영정보학과 부교수
jlhan@uhs..ac.kr
[2008/07/30 투고 - 2008/07/31 심사 - 2008/10/13 심사완료]

수 있다.

점진 평가를 수행하기위해, 수정된 부분과 그 부분에 영향을 받아 변경되는 부분을 찾아내어 이 부분만을 다시 평가해야하고 이를 위해 점진 속성 평가 방법[1]을 사용한다. 프로그램에서 실행되는 결과는 변수의 값을 계산하고 저장하여 출력하는 것이고 변수의 값을 나타내는 속성에 대한 종속성(dependency)만 관리하면 훨씬 효율적인 점진 평가를 수행할 수 있다. 반면, 기존의 점진 속성 평가 방법에서 사용된 종속 그래프(dependency graph)의 경우 속성 문법(attribute grammar)에 기반을 두어 프로그램의 의미 구조를 나타내기 위해 프로그램에 소속된 모든 속성에 대해 종속성을 나타내어 종속 그래프가 상당히 복잡하고 수정이 일어날 때마다 모든 속성에 대해 변화를 파급(propagation)시켜야 하므로 비효율적이라 할 수 있다. 예로서, 변수만 고려하더라도 각 변수들에 대해 이름, 값, 환경 등의 속성이 있고 이러한 세 가지 속성의 종속성을 표시하고 관리해야하고 변화가 생길 때마다 이들 속성의 변화를 파급시켜야 하므로 상당히 복잡하다. 특히, 종속 그래프나 형식적인 의미 명세를 통한 기존의 점진 평가 방법들의 경우, 객체 지향 개념이 활발하게 사용되기 전에 제시된 방법이어서 객체를 처리하는 방법이 구체적으로 기술되어 있지 않아 자바와 같은 객체지향언어에 적용하는 것이 어려운 실정이다. 객체를 처리하기 위한 속성들에 대한 종속성을 종속 그래프로 나타낸다 하더라도 객체를 다루기 위한 더 많은 속성들에 대한 종속성을 유지해야 하므로 효율적이지 못하고 점진 속성 평가 과정이 더욱 복잡해진다. 따라서, 객체를 적절하게 처리하는 점진 평가 방법에 대한 연구가 필요하고 객체지향언어에 대해 점진 평가를 효율적으로 수행할 수 있는 방법이 제시되어야 하는데 이러한 연구들은 미미한 실정이고 동적 의미를 형식적으로 명세한 연구들은 진행 중에 있다[2, 4].

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. 확장된 종속 차트는 의미 구조에 직접적으로 영향을 주는 변수의 값을 나타내는 속성을 중심으로 종속성을 표시하여 변화를 추적하는 과정이 효율적으로 수행된다. 2장에서는 관련연구들을 기술하고 3장에서는 객체 지향 개념을 다룰 수 있도록 확장된 종속차트를 제시한다. 4장에서는 객체를 다루어 처리하기 위한 점진 평가 알고리즘을 제시하여 알고리즘의 정확성을 증명한다. 5장에서는 실험을 통해 점진 평가의 효율성을 분석한다.

2. 관련 연구

점진 평가를 수행하는 많은 연구들이 진행되었고 이러한 점진 시스템들 중에서 ALOE 에디터[6]나 Gandalf 프로젝트[7]에서는 언어의 추상 구문과 관련하여 작용 루틴을 작성하여 점진 평가를 수행하였다.

POE 에디터[8]는 속성 문법을 사용하여 문법에서 자동적으로 유도되는 속성들 간의 종속 정보를 이용해서 주어진 속성 문법에서 최소한으로 필요한 속성이 다시 평가되어 변경된 부분에 영향 받는 부분만을 평가하게 된다.

Cornell Program Synthesizer[5]도 속성 문법을 사용하여 속성들 간에 종속성을 표현하고 변화가 생겼을 때 변화된 속성에 종속하는 속성들을 점진 속성 평가 방법을 사용하여 찾아내게 된다.

통합된 점진 프로그래밍 환경인 Galaxy[9]에서는 변경된 부분에 대한 부트리를 마크하여 마크된 부트리만을 다시 파싱하는 방법을 사용하고 있다. 이 시스템에서는 Galaxy 언어에 대한 점진적 스캐닝과 점진적 파싱을 위해 사용되어질 알고리즘과 방법을 제시하여 시간과 공간적인 면에서 효율적으로 실행되고 있다.

Alphonse[10]는 큰 네트워크에서 상호 관련된 계산의 재사용이 허용되는 점진적 갱신 기법을 사용한 시스템이다. 변화가 일어났을 때 변화되기 전

의 그전 값과 중간 단계 계산의 새로운 결과를 비교하는 경우 변화에 대한 파급을 일시 중단하는 정지 파급(Quiescence propagation)을 수행한다. Alphonse는 변화에 의존하는 종속성을 기억하는데 필요한 부기(bookkeeping)를 자동적으로 수행하여 점진적 구현을 생성하는 자동 점진 시스템이다.

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. 실제로 영향을 주는 속성은 변수의 값을 나타내는 속성인데, 기존의 연구에서는 프로그램에서 사용된 모든 속성들 간의 종속성을 종속 그래프(dependency graph)로 나타내어 수정된 부분에서 생긴 변화를, 값을 나타내는 속성뿐만 아니라 모든 속성에 대해 그래프상의 다른 부분으로 변화를 파급시켜야 하므로 변화를 추적하는 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 종속성을 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.

3. 확장된 종속 차트

종속차트는 종속성을 나타내기 위한 유향(directed) 그래프이다[1]. 기존의 종속차트를 확장하여 자바와 같은 객체지향 언어를 처리하는 객체와 클래스에 대한 정보를 추가한다. 이후 확장된 종속차트를 편의상 간략히 줄여 종속차트라 명명한다. 그림 1에서처럼 기존의 종속차트에 객체를 처리하는 부분 즉, 소속클래스, 객체 리스트, 소속 메서드에 대한 정보가 추가로 들어간다. 각 클래스에 소속된 변수들에 대해 종속성을 나타내려면 기존의 종속차트를 확장하여 변수들이 소속된 객체에 대한 소속정보를 갖도록 해야 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. 필수적인 변수들을 평가하기 위해 확장된 종속차트에도 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

(그림 1) 확장된 종속차트

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

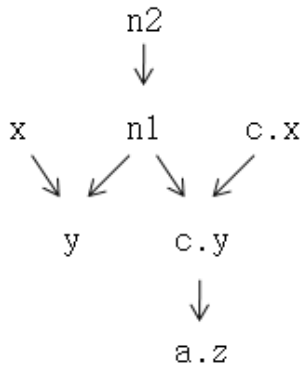
평가될 필요성이 있는지 그 여부를 나타내기 위한 정보가 들어가고 CON과 FLAG 필드에 따라 변수의 평가 여부를 결정하게 된다. 예로서 반복문에서 게이트 속성의 값이 거짓이라면 그 WHILE 루프 속에 값이 변경된 변수가 존재하더라도 그 값을 다시 계산할 필요가 없다. 즉, WHILE 루프의 조건이 거짓이라면 루프를 수행하지 않아도 되기 때문이다. 마찬가지로 IF 문의 조건식에 해당되는 값이 거짓이라면 참일 때 수행되는 명령문에 속한 변수가 변경되었다고 해도 그 값을 다시 계산할 필요가 없다[1].

```
public class Dependency {
  int x = 1, y = 2, z;      --- ①
   ...
  void methodEX() {
    int n1, n2 = 10;        --- ②
    n1 = n2 + 2;            --- ③
    if (n1 > 10)
       y = n1 * x;          --- ④
    ...
   }
  ...
  public static void main(String args[]) {
   Dependency a, b, c;
   a = new Dependency();
   b = new Dependency();
   c = new Dependency();
   ...
   c.methodEX();            --- ⑤
   a.z = c.y + 10;          --- ⑥
   ...}
```

(그림 2) 자바 프로그램

Dependency 클래스의 객체 a, b, c 각각은 객체 속성변수인 멤버 변수로 x, y, z를 갖는다. 종속 링크는 배정문의 우변에 있는 변수의 값이 변할 때 좌변의 변수 값이 변하게 되므로 이를 종속 링크에 반영한다. n2의 값의 변화에 따라 n1의 값이 변하므로 n2 → n1 으로 링크를 생성한다[1].

변수의 종속성을 표현한 종속 링크만을 갖는 종속차트는 그림 3과 같다.



(그림 3) 확장된 종속차트의 종속링크

그림 2의 ⑤번 메서드 호출에 의해 c.x → c.y 의 링크가 존재하고 만일 a.methodEX(); 로 변경되면 c.x → c.y 대신 a.x → a.y 로 변경된다.

종속 차트의 다른 필드를 나타내면 표 1과 같다.

(표 1) 확장된 종속 차트의 필드 정보

변수	소속 클래스	객체리스트	소속메서드	CON	FLAG
n1	Dependency		methodEX		I
n2	Dependency		methodEX		I
x	Dependency	a, b, c	methodEX		I
y	Dependency	a, b, c	methodEX	n1	T
c.x	Dependency	a, b, c	main		I
c.y	Dependency	a, b, c	main	n1	T
a.z	Dependency	a, b, c	main		I

변수 y의 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)

을 만들어 클래스와 관련된 소속정보를 넣는다. 특히, 그림 2의 ⑤번과 ⑥번 명령문의 순서가 바뀐 경우에 대비해 변화 추적을 올바르게 수행하기 위해 프로그램의 라인 정보를 VT의 마지막 필드에 추가한다. 예로서 그림 2의 ③번 명령문에 대해서만 종속차트를 구성하면 그림 4와 같다. ③번 명령문이 그림 2의 프로그램에서 11 라인에 있다고 가정했을 때 VT의 맨 마지막 필드에 라인 정보가 들어간다.

n2	Dependency		methodEx		I	11	→ n2
n1	Dependency		methodEx		I	11	→ n1

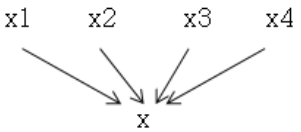
(그림 4) 종속 차트

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 깊이우선탐색(DFS)으로 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 평가하지 않더라도 전체 프로그램을 평가한 것과 동일한 결과를 얻을 수 있다.

변수 값이 변할 때 점진 평가를 수행하기 위해 종속링크를 찾아 가는 것이 중요하다. 만일 여러 변수들에 변화가 생겼을 경우 각 변수들에 대해 순차적으로 종속 링크를 추적하여 영향받는 변수들을 찾아 그 명령문들을 점진 평가하게 된다. 그림 5와 같이 여러 변수인 x1, x2, x3, x4가 동시에 하나의 변수 x에 영향을 주는 경우도 종속 차트의 종속 링크를 따라 가면 쉽게 찾을 수 있다.

x = x1 + x2 * x3 - x4;



(그림 5) 예제 명령문과 종속성

그림 2의 프로그램을 중심으로, 자바와 같은 객체지향언어에서 변수의 값이 변경될 수 있는 경우를 살펴보면 여러 가지가 있다.

(1) 지역 변수의 변경

가장 간단한 변수 값의 변화는 지역 변수와 관련된 값의 변화이다.

②번 명령문에서 n2 값을 15로 변경하면 n1 값이 바뀌게 되므로 종속 링크를 따라가서 ③번 명령문 n1 = n2 + 2; 을 다시 평가하면 n1 값을 변경할 수 있다. n1 값이 바뀌면 ④번 명령문을 다시 평가하여 y값도 변하게 된다. 두 번째 경우에 자세히 언급하겠지만 y는 객체 속성변수이고 methodEX()에 포함된 명령문에 속한 변수이므로 ⑤번 명령문에서처럼 methodEX()를 호출하는 객체 c의 y 값만 바뀌도록 다시 평가한다.

(2) 객체 속성 변수의 변경

①번 명령문에서 객체 속성변수인 x, y, z의 값을 변경하면 x, y, z를 객체 속성변수로 갖는 객체 a, b, c의 x, y, z도 변하게 된다. 만일 x 값을 변경했을 때 ④번 명령문에서처럼 y 값도 변하는데 ④번 명령문은 n1값이 10보다 클 경우만 평가하게 된다. 이를 위해 CON 필드와 FLAG 필드를 확인하여 ④번 명령문을 다시 평가하게 된다. 그런데 ⑤번 명령문에서처럼 methodEX()를 호출하는 객체 c의 y 값만 바뀌게 된다. 즉 객체 a와 객체 b는 methodEX()를 호출하지 않으므로 y 값은 변하지 않게 된다. 만일 ⑥번 명령문에서 수정이 일어나서 z 값이 변경될 경우 ⑥번 명령문을 다시 평가해야 하고 객체 a의 z 값이 변하도록 다시 평가한다.

(3) 객체이름의 변경

⑥번 명령문 a.z = c.y + 10; 에서 a.z 대신 c.z로 객체 이름을 변경하면 해당 객체의 속성 변수에 변화가 생기게 되고 종속 링크도 변경해야 한다. 객체 c의 z값이 바뀌도록 ⑥번 명령문을 다시 평가한다.

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

점진평가를 위해 사용된 용어들을 정의하면 다음과 같다.

<정의 1> 종속성

B = A 식이 존재하면 우변에 있는 임의의 변수 A는 좌변에 있는 변수 B에 종속한다고 한다. 만일 A가 B에 종속한다면 종속 차트에 B → A 패스(path)가 존재하게 된다. 또한 종속 차트에 B → A 패스(path)가 존재한다면 A가 B에 종속하게 된다.

<정의 2> change-set

값이 변경된 변수에 종속하는 변수들 즉 변경된 변수에 영향받는 변수들의 집합이 change-set이다.

<정의 3> evaluated-set

change-set에 속한 변수들 중 조건 속성의 값에 따라 프로그램 수행 상 실행될 필요가 있는 변수들의 집합이 evaluated-set이다.

```
Algorithm IncrementalALGM(VT, DC, X);
/* let DC be the dependency chart */
/* let VT be the variable table in the DC */
/* 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;
                 // DFS is the Depth First Search.
                insert P into change-set;
                mark of P := true;
              end;
        end;
endFor;
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
    endCase
endFor;
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
 endCase
endFor;
```

(그림 6) 점진 평가 알고리즘

그림 6의 점진 평가 알고리즘은 크게 두 부분으로 나눠지는데 전반부에서는 명령형 언어에서의 점진 평가처럼 필수 속성을 갖는 변수들을 찾아내는 과정이다. 최적의 평가를 수행하려면, 종속 차

트의 CON과 FLAG 값을 고려해 평가될 필요가 있는 필수 속성을 갖는 변수들만 다시 평가해야 한다. 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. 후반부에서는 객체를 처리하기 위해 새롭게 확장된 부분으로 evaluated-set에 속한 변수들 중 지역변수인지, 객체 속성변수인지 파악하여 해당되는 평가를 수행하고 객체이름을 변경한 경우 해당 객체의 속성변수가 변하도록 평가한다.

그림 2의 프로그램을 중심으로 점진 평가를 수행하는 과정을 살펴본다. 만일 ①번 문장에서 x값을 변경하고 ②번 문장에서 n2값을 변경했다고 가정한다면 점진 평가 알고리즘은 변경된 모든 변수들에 대해 for each 반복문으로 change-set를 찾아내고 change-set에 속한 모든 변수들에 대해 for each 반복문으로 필수 속성을 갖는 변수들로 evaluated-set를 구성하여 전반부를 수행한다. 먼저 x부터 살펴보면, 그림 2의 종속차트의 종속링크를 통해 알 수 있듯이 y값이 바뀌게 됨을 알 수 있고 y가 change-set에 들어간다. n2도 같은 방법으로 n1, c.y, a.z이 바뀌게 됨을 알 수 있고 이들이 change-set에 들어간다. change-set에 속한 각 변수에 대해 for each 반복문으로 evaluated-set을 찾아내어 필수적인 변수를 가려낸다. 이들 중 y나 c.y를 제외한 나머지 변수들은 CON 필드에 값이 없으므로 모두 evaluated-set에 들어간다. 반면, y나 c.y는 CON 필드와 FLAG 필드에 값이 있으므로 이 변수들에 대해서만, 다시 평가될 필수 변수인지 알기 위해 표2에서 볼 수 있듯이 CON 필드와 FLAG 필드를 조사하는데 y의 FLAG가 T 이므로 심벌 테이블의 n1 값을 참조하여 조건식인 (n1 > 10)이 참이면 y는 evaluated-set에 들어간다. 즉 y는 필수 변수가 되고 c.y도 같은 방법으로 심벌 테이블의 n1 값을 참조하여 조건식인 (n1 > 10)이 참이면 y는 evaluated-set에 들어간다.

알고리즘의 후반부로 넘어가서 객체에 관한 정보에 따라 평가를 수행하는데 전반부에서 구성한 evaluated-set에 속한 모든 변수들에 대해 for each 반복문으로 평가를 수행하게 된다. 이는 evaluated-set에 속한 변수가 지역변수, 객체 속성변수, 객체, 클래스, 메서드일 경우 앞부분에서 언급되었듯이 각각에 해당되는 평가를 수행하게 된다.

제시된 점진 평가 방법의 올바름을 증명하려면 전반부와 후반부가 올바름을 증명하면 된다. 전반부의 경우 명령형 언어에서 evaluated-set에 속한 변수들만 평가했을 때 올바른 값을 갖게 된다는 것이 이미 증명되어있다[1]. 후반부에서, 클래스 이름이나 메서드 이름이 변경된 경우는 종속차트에서 클래스 이름이나 메서드 이름을 찾아서 변수의 소속 정보를 변경한다. evaluated-set에 속한 변수들 중에는 지역 변수나 객체 속성변수나 객체 이름일 경우가 있고 각 경우에서 객체에 따른 평가가 올바른지 다음 정리를 통해 증명한다.

<정리 1> evaluated-set에 속한 모든 지역변수들을 점진 평가했을 때 그 지역변수들은 정확한 값을 갖는다.

<증 명> evaluated-set에 속한 지역변수들을 x1, x2, ... xn라 가정하자. 각 xi는 <정의 1>과 <정의 3>에 의해 변경된 변수에 영향 받는 변수이고 다시 평가해야하는 필수적인 변수이다. 각 xi는 <정의 1>에 의해 변경된 변수들에 종속하는 변수들이므로 이러한 종속하는 변수들은 값이 변하는 변수들이고 다시 평가해야만 정확한 값을 갖게 된다. 또한, 종속차트의 종속링크를 통해 추적된 변수이므로 올바른 추적 경로를 통해 발견된 변수이다. 따라서, 종속차트의 라인정보를 따라 각 xi가 포함된 명령문들 S1, S2, ... Sn을 찾을 수 있고, 각 Si 명령문을 다시 평가하면 그 명령문에 소속된 각 xi는 정확한 값을 갖게 된다.

<정리 2> 객체 속성변수가 변경되고 그 객체 속

성변수가 evaluated-set에 속해서 점진 평가한 후 그 변수는 정확한 값을 갖는다.

<증 명> 객체 속성변수가 변경되고 그 객체 속성변수가 evaluated-set에 속했다는 것은 <정의 3>에 의해 변경된 값에 영향 받고 다시 평가해야하는 필수적인 객체 속성 변수라는 사실을 알 수 있다. 이러한 객체 속성변수가 소속된 객체를 찾아 다시 평가된 값을 변경해 줘야 각 객체는 정확한 객체 속성변수 값을 갖게 된다.

임의의 x를 객체 속성변수라 가정하고 x가 소속된 객체를 XX라 가정하자. 종속차트의 VT에서 x를 찾을 수 있고 종속차트의 라인 정보에 따라 x가 속한 명령문을 찾을 수 있 다. x가 소속된 명령문을 다시 평가해야 x는 올바른 값을 갖게 된다. x가 올바른 값을 갖도록 하기위해 x가 소속된 객체 XX를 종속차트에서 찾을 수 있고 점진 평가과정을 거쳐 x가 소속된 해당 명령문은 다시 평가하게 된다. 점진 평가를 수행하여 다시 계산된 x 값은 객체 XX에 속한 객체 속성변수가 변경되도록 저장하여 다시 평가된 값이 심벌 테이블에 들어간다. 따라서 객체 속성변수가 변경되더라도 각 객체는 정확한 객체 속성변수 값을 갖게 된다.

<정리 3> 객체 이름이 변경되고 evaluated-set에 속했을 때 점진 평가 후 각 객체 속성 변수들은 정확한 값을 갖는다.

<증 명> 객체 이름이 변경되어 evaluated-set에 속하면 각 객체 속성변수들은 종속차트에 변경된 객체로 소속 정보가 바뀌게 되고 종속 링크도 변경이 된다. 임의의 XX를 evaluated-set에 속한 객체라 가정하고 그 객체에 속한 객체 속성변수를 x1, x2, ... xn 이라 가정하자. 각 xi 는 종속 차트에 있는 정보에 따라 작성되고 링크를 쫓아가서 객체 이름이 변경된다. <정의 3>에 의해 각 xi는 값이 변하게 될 영향 받는 필수적인 객체 속성변수들이고 이 변수들을 종속차트에서 찾을 수 있고 종속차트의 라인 정보에 의해 각 xi가 속한 해당 명령문을 찾을 수 있다. 각 xi가 올바른 값을 갖도록 하기 위해, 각 xi가 소속된 명령문들을 다시 실행하여 점진평가를 수행해야한다. 그런데, 각 xi의 명령문들은 종속차트의 라인정보를 통해 찾을 수 있고 이들 명령문을 다시 평가하여 각 xi는 올바른 값을 갖게 된다. 따라서 객체 속성변수인 각 xi는 정확한 값을 갖게 된다.

<정리 1>, <정리 2>, <정리 3>에 의해 evaluated-set에 속한 모든 것을 점진 평가했을 때 각 변수들은 정확한 값을 갖는다는 것을 알 수 있고 제시된 알고리즘은 정확하다는 사실을 확인할 수 있다. 기존의 연구를 통하여 실제로 점진 평가 방법으로 번역했을 때 전체 프로그램을 번역한 것과 같은 결과를 얻는다는 사실을 알 수 있었다[1].

5. 점진 평가의 효율성 실험

객체 지향 언어에 대해서 점진 평가를 수행하는 기존의 연구가 없어 기존 연구와 비교하여 분석하는 과정에 애로사항이 있어서, 프로그램 전체를 평가했을 때와 본 연구에서 제시한 점진 평가 방법으로 평가 했을 때를 비교하여 점진 평가의 효율성을 분석한다. 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. 명령형 언어를 비롯한 자바의 경우도 실행 결과에 직접적으로 영향을 미치는 명령문들 중에는 배정문, 조건문 및 반복문이 있고 이들 명령문들이 가장 빈번하게 사용되는 명령문들이어서 프로그램의 수정 유형으로 선택하여 점진 평가의 효율성을 분석한다.

(표 2) 프로그램 점진 평가 결과 (단위:%)

프로그램 유형	배정문	조건문	반복문	호출문	평균	표준 편차
P1	30	12	31	35	27	10.23
P1	89	3	90	91	68	43.51
P2	33	32	38	41	36	4.24
P2	93	94	95	96	95	1.29
P3	6	7	76	71	40	38.74
P3	8	13	88	82	48	43.13
P4	9	17	88	81	49	41.51
P4	5	9	89	83	47	45.71
평균	34	23	74	73	51	
표준편차	36.74	29.83	25.25	22.58	21.3	

표 2에서 네 가지 프로그램 유형에 대해 각 유형별로 2가지 형태의 프로그램을 사용하는데 소량의 자료를 사용한 경우와 대량의 자료를 사용한 경우이다. 표 2의 단위는 프로그램 전체를 평가할 경우를 100%로 놓았고 해당하는 유형의 점진 평가를 수행할 경우에 점진 평가되는 프로그램의 비율을 %로 표시하고 있다. 표2의 맨 윗줄에 있는 '평균'과 '표준편차'는 네 가지 유형의 수정인 배정문, 조건문, 반복문, 메서드 호출문의 점진 실행 효율의 평균을 구하고 표준편차를 구한 것이다. 맨 마지막 줄의 '평균'과 '표준편차'는 네 가지 수정 유형별로 모든 프로그램 유형의 점진 평가의 평균을 구하고 표준편차를 구한 것이다. P1과 P2 프로그램의 경우 반복문이 실행 결과를 얻는 핵심 구문이 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. 표준 편차를 고려할 경우 P2 프로그램이 네 가지 수정 유형에 관계없이 점진 실행이 비슷한 효율성을 나타냄을 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. 따라서, 반복문의 경우 점진 평가를 수행할 때 다른 명령문 유형보다 효율적이지 못하다는 사실을 알 수 있다.

배정문이나 조건문의 경우 반복문이나 메서드 호출문보다 표준편차가 조금 높은 사실에서 배정문이나 조건문이 수정되었을 때 점진 실행 효율에 편차가 심하다는 사실을 알 수 있다. 이는 배정문이나 조건문에서의 수정이 반복문에 영향을 미치지 않는 경우는 점진 실행 효율성이 아주 높기 때문이다.

평균적으로 고려해볼 때 배정문의 수정이 반복문까지 영향을 주는 경우도 있지만 영향을 주지 않는 경우도 있으므로 51%의 점진 실행 효율이 나타난다. 즉 프로그램 전체를 평가하는 것을 100%로 놓았을 때 전체 프로그램의 51%만 실행하여 상당히 효율적인 점진 평가를 수행하게 된다. 점진 평가의 효율성 분석에는 종속 차트를 실시간으로 저장하고 갱신하는데 소요되는 시간은 고려하지 않지만 이 시간을 추가한다하더라도 제시된 점진 평가 방법이 전체 프로그램을 다시 평가하는 것보다 더 효율적이라는 사실을 확인할 수 있다.

6. 결 론

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 영향

을 주는 변수의 값을 나타내는 속성을 중심으로 종속성을 표시하여 변화를 추적하는 과정이 효율적으로 수행된다. 객체 지향언어에서 점진 평가를 수행하는 알고리즘을 제시하고 그 알고리즘의 정확성을 증명하였고 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.

참 고 문 헌

[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] David A. Watt and Deryck F. Brown, "Formalizing the Dynamic Semantics of Java", 2006.

[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.

◐ 저 자 소 개 ◑

한 정 란

1985년 이화여자대학교 전자계산학과 졸업(학사)
1987년 이화여자대학교 대학원 컴퓨터공학과 졸업(석사)
1999년 이화여자대학교 대학원 컴퓨터공학과 졸업(박사)
1999～현재 협성대학교 경영정보학과 부교수
관심분야 : 형식언어론, 전자상거래, XML, 웹 서비스 등
E-mail : jlhan@uhs..ac.kr