자료 흐름 분석 기법을 이용한 효율적인 프로그램 통합에 관한 연구

박 순 형*

A Study on the Efficient Program Integration using Data Flow Analysis Method

Soon-Hyung Park*

Abstract

To take the re-use of software, we need to study the efficient integration method of source programs. When the source programs are merged, it is required the steps of verification for any non-interference on non-identical parts of them. The traditional techniques of the program integration verify non-interference of source programs through the simple comparison of statements of source programs. We propose the efficient integration method using data flow analysis in the programs. A study comparing test results from the traditional method and the proposed method has found that the proposed method is more efficient than the traditional method.

Keyword : program re-use, program slicing, program integration, program dependence graph

* 한국산업관계연구원

1. 서 론

통합 대상 프로그램들을 서로 통합하여 하나의 새로운 통합 프로그램을 생성하기 위해서는 통합 대상 프로그램들의 서로 일치하지 않는 부분이 각 통합 대상 프로그램들이나 통합 프로그램에 대해 서로 간섭하지 않아야 한다[4]. 그러나 기존의 통합 기법은 프로그램을 구성하고 있는 명령문들을 단순히 비교하는 방법으로써 비 간섭 여부에 대한 확인을 시도하였기 때문에 통합이 가능한 프로그램들도 비 간섭 기준을 충족하지 못하여 통합을 못하는 사례가 많이 발생하였다. 본 논문에서는 프로그램 내에서의 변수들의 변화를 추적하는 방법으로 프로그램의 구조를 분석하고 프로그램을 구성하는 명령문들의 구조를 분석하는 방법을 통해 프로그램 통합의 가능성을 높여주는 방법을 제시하였다.

2장에서는 프로그램 통합 과정에서 적용되어질 프로그램 통합 가능성에 대한 비 간섭 기준을 제시하였다. 3장에서는 기존의 프로그램 통합 기법에 대해 설명하였고 4장에서는 본 논문에서 제시한 효율적인 프로그램 통합 기법에 대해 다루었으며 5장에서는 본 논문에서 제시한 기법의 효율성에 대해 기존의 기법과 비교 설명하였다.

2. 프로그램 통합기준

(1) Different nodes

어떤 통합 대상 프로그램 P를 기준으로 할 때 또 다른 통합 대상 프로그램과의 사이에 존재하는 差(difference)의 집합을 슬라이스한 결과를 의미한다. 통합 대상 프로그램P_B에 대한 통합 대상 프로그램 P_A의 different nodes $DN_{A,B}$는 P_A에서 P_B와의 nodes의 差의 집합을 슬라이스한 결과이다. 이때 v는 program을 구성하는 vertex 즉, node이며, V는 vertices 즉, node들의 집합이다.

$$DN_{A,B} = \{v \in V(P_A),\ V(P_B) \mid (P_A/v) - (P_B/v)\}$$

(2) 비 간섭 기준

통합 대상 프로그램 P_A와 P_B에 대한 통합 프로그램이 P_M 이고 프로그램 P_A와 P_B에 대한 공통 루틴의 집합 P_{AB} 를 PC라고 할 때 PM/DNA,C는 DNA,C를 기준 변수로 하는 프로그램PM에서의 슬라이스의 집합을 의미한다.

통합 프로그램 PM내에서 보존되어진 PA 와 PB 의 변화된 동작을 위한 유지되어져야 하는 비 간섭 기준은 “조건(1) and 조건(2)” 이다.

- **조건**(1) : $P_M/DN_{A,C} = P_A/DN_{A,C}$ when $P_C = P_{AB}$
- **조건**(2) : $P_M/DN_{B,C} = P_B/DN_{B,\ C}$ when $P_C = P_{AB}$

(3) 프로그램 종속 그래프(Program Dependence Graph)

프로그램 종속 그래프(PDG) 기법은 원시 프로그램을자료 흐름 분석과 제어 흐름 분석을 통하여 프로그램 종속 그래프로 변환하여 슬라이스를 산출하는 방법이다. 프로그램 종속 그래프는 두 가지 종류의 간선들에 의해 연결된 정점들의 방향성 그래프이다[2].

3. 프로그램 통합기법

기존의 프로그램 통합 기법 중 대표적인 프로그램 통합 기법을 소개한다[1][3].

(1) 통합 대상 프로그램의 비 간섭 여부를 확인하기 위해 프로그램 P_A와 P_B의 공통 부분으로 구성된 *Base* 프로그램 P_{Base}를 만든다.

(2) P_{Base}에 대한 P_A와 P_B의 different nodes인 $DN_{A,Base}$와 $DN_{B,Base}$를 생성한다.

(3) 프로그램 P_A와 P_B를 결합한 통합 프로그램 P_M을 만든다.

(4) different nodes인 $DN_{A,Base}$와 $DN_{B,Base}$에서

산출된 기준변수를 사용하여 P_A와 P_B가 P_M과 관련하여 간섭을 하는지를 결정한다.

(5) 통합 프로그램 P_M이 정확하게 실행하는지를 점검한다. 즉, P_M의 노드 배열순서를 조정하여 완전한 통합 프로그램을 만든다.

4. 효율적인 프로그램 통합 기법

(1) 두 개의 통합 대상 프로그램 P_A와 P_B에 대한 축약 프로그램(shortening program) P_{SA}와 P_{SB} 를 각각 만든다. 축약 프로그램이란 자료 흐름의 변화를 최소화 시켜놓은 프로그램이다.

(2) 축약 프로그램 P_{SA}와 P_{SB} 의 프로그램 슬라이싱 기준변수들에 대한 슬라이싱을 한 후 각각의 프로그램 슬라이스를 산출한다.

(3) 축약 프로그램 P_{SA}와 P_{SB} 의 프로그램 슬라이싱 기준변수들에 대한 프로그램 슬라이스들의 프로그램 종속 그래프(PDG)를 그린다.

(4) 축약 프로그램 P_{SA}의 기준변수들에 대한 PDG와 축약 프로그램 P_{SB} 의 기준변수들에 대한 PDG를 상호비교하여 graph 구조가 서로 같고 graph를 구성하는 node들의 구성 type이 서로 같은 graph들로 축약 프로그램 P_{SA}와 P_{SB} 에 대한 *Base*프로그램 P_{Base}를 구성한다.

(5) 축약 프로그램 P_{SA}의 기준변수들에 대한 PDG와 축약 프로그램 P_{SB} 의 기준변수들에 대한 PDG를 상호비교하여 graph 구조가 서로 다르거나 graph 구조가 같더라도 graph를 구성하는 node들의 구성 type이 서로 다른 graph들을 사용하여 두 개의 축약 프로그램 P_{SA}와P_{SB} 의 different nodes인 $DN_{SA,Base}$와 $DN_{SB,Base}$를 생성한다.

(6) 축약 프로그램 P_{SA}와 P_{SB}를 결합한 통합 프로그램 P_M을 생성한다.

(7) different nodes인 $DN_{SA,Base}$와 $DN_{SB,Base}$에서 산출된 기준변수를 사용하여 P_{SA}와P_M이 서로 간섭을 하는지 그리고 P_{SB}와 P_M이 서로 간섭하는지를 확인한다. P_{SA}와 P_M그리고 P_{SB}와 P_M모두가 서로 간섭하지 않으면 P_M이 통합 프로그램이 된다.

5. 고찰

본 논문의 통합기법이 효율적임을 증명하기 위해 통합 대상 프로그램을 중심으로 기존의 프로그램 통합 기법과 본 논문에서 제시한 기법을 비교해 본다. (그림1)은 통합 대상 프로그램이다.

```
      begin
S1:   MULTIPLE := 1;
S2:   SUM := 0;
S3:   MIDSUM := 0;
S4:   H := 10;
S5:   I := 0
S6:   while I < 10
      do
S7:     K := I;
S8:     MULTIPLE := MULTIPLE * H;
S9:     SUM := SUM + K;
S10:    MID := SUM / 2;
S11:    MIDSUM := MIDSUM + MID;
S12:    I := I + 3
      while_end
S13:  write(SUM);
S14:  write(MIDSUM);
S15:  write(MULTIPLE)
    end.
```

(그림 1) 통합 대상 프로그램 P_S

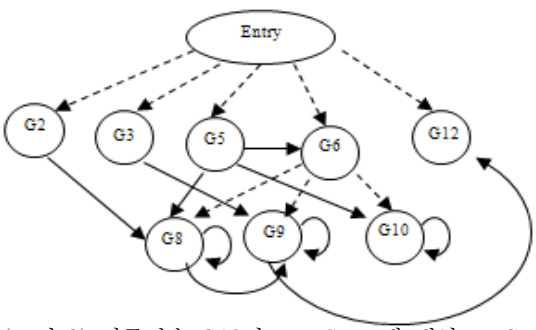
통합 대상 프로그램 P_S 에 대한 축약 프로그램 P_G 가 (그림 2)에 나타나 있다.

```
      begin
G1:   MULTIPLE := 1;
G2:   SUM := 0;
G3:   MIDSUM := 0;
G4:   H := 10;
G5:   I := 0
G6:   while I < 10
      do
G7:     MULTIPLE := MULTIPLE * H;
G8:     SUM := SUM + I;
G9:     MIDSUM := MIDSUM + SUM / 2;
```

```
G10:      I := I + 3
        while_end
G11:    write(SUM);
G12:    write(MIDSUM);
G13:    write(MULTIPLE)
      end.
```

(그림 2) 축약 프로그램 P_G

기존의 통합기법처럼 compact한 프로그램을 산출하지 않고 통합기법을 수행한다면 통합 대상 프로그램에 대한 생성된 *Base* 프로그램 이 완전한 프로그램이 아닐 수 있으며 그 결과 통합 프로그램에서도 기준변수에 대한 정확한 값을 산출할 수가 없는 경우가 있을 수 있다. 그리고 통합 대상 프로그램들의 변수명을 비교해서 통합 가능 부분을 산출하는 기존의 통합기법이 아닌 통합 대상 프로그램의 프로그램 종속 그래프를 통한 프로그램 구조를 비교하여 통합 가능 부분을 산출하는 본 논문이 통합 가능성을 좀 더 높일 수 있다. (그림 2)의 축약 프로그램에서 슬라이싱 기준변수 G12의 MIDSUM에 대한 프로그램 종속 그래프가 (그림 3)에 나타나 있다.



(그림 3) 기준변수 G12의 MIDSUM에 대한 PDG

6. 결 론

프로그램 재사용이란 새로운 프로그램을 작성하기 위하여 기존에 개발되어 사용되고 있는 프로그램들 중 필요로 하는 일부분의 프로그램 모듈들을 통합하여 우리가 필요로 하는 프로그램을 만드는 것이다. 기존의 프로그램 통합 방법은 프로그램 구성하는 형식 과 프로그램 로직에 대한 검토를 하지 않고 단지 통합 대상 프로그램을 구성하는 명령문들을 단순 비교하는 방식으로 통합여부를 검증하는 방식을 취하였다. 그러므로 프로그램 통합 가능성이 낮았다. 본 논문에서는 통합 대상 프로그램의 로직을 추적하여 축약 프로그램을 만든 후 프로그램 통합을 시도하는 방법을 제시함으로써 프로그램 통합 가능성을 높이는 효율적인 통합 방법을 제시하였다. 실제 예에서 살펴보면 통합 대상 프로그램 P_E와 P_F를 기존의 통합 기법을 사용하여 통합을 시도한 결과 두 통합 대상 프로그램의 공통 명령문들의 집합인 *Base* 프로그램이 완벽한 프로그램의 골격을 갖추지 못한 것을 알 수 있었다. 그리고 통합 프로그램 또한 원하는 기준 변수를 슬라이싱 한 결과 기준 변수의 원하는 결과를 산출할 수 없는 것도 발견할 수 있었다. 그러나 본 논문에서 제시한 기법 사용하여 통합 대상 프로그램들의 축약 프로그램을 만든 후 통합 단계를 시도하면 완존한 통합 프로그램을 산출할 수 있음을 알 수 있었다. 따라서 본 논문에서 제시한 프로그램 통합 기법이 통합 대상 프로그램들의 통합 가능성을 높여준다는 사실을 알 수 있었다.

참 고 문 헌

[1] David Binkley, Susan Horwitz, and Thomas Reps "Program Integration for with Procedure Calls", ACM on Software Engineering, pp.3-35, Jan. 1995.

[2] Darren Atkinson and William Griswold "Implementation techniques for efficient data-flow analysis of large programs", In Proc. International Conference on Software Maintenance, 2001.

[3] Ioana, Manolescu, Luc Bouganim, and Erpc Simon, "Efficient Data and Program Integration using Binding Patterns", INRIA-00072348, August 2001.

[4] M. Burgin, "Logical Tools for Program Integration and Interoperability", Softwae Engineering and Applications(SEA), Mit Cambridge, USA, November 2004.