동적 슬라이싱 기법을 이용한 프로그램 버젼들의 통합 알고리즘

박 순 형† · 정 은 이†† · 박 만 곤†††

요 약

일반적인 소프트웨어 시스템은 새로운 요구와 오류의 발견으로 인해 계속적인 개발과 확장 그리고 수정이 요구된다. 그러므로 소프트웨어의 재사용이 가능한 컴포넌트들을 통합하여 새로운 소프트웨어 시스템으로 재구성할 수 있다면 소프트웨어의 생산성, 품질 및 비용을 개선시킬 수 있을 것이다. 이를 위해 프로그램의 통합을 효율적으로 진행시킬 수 있는 통합 프로그램의 개발과 연구는 필요할 것이다. 두 개의 프로그램을 통합하여 하나의 프로그램으로 만들기 위하여 각 프로그램이 서로 일치하는 부분이 존재할 것이고 서로 비 간섭임이 확인되어야만 한다. 기존의 논문에는 비 간섭 기준 대상 프로그램의 크기가 너무 커서 실제로 통합이 가능한 많은 프로그램들이 통합을 하지 못하는 경우가 많았다. 본 논문에서는 동적 슬라이싱 기법을 사용하여 비 간섭 기준 대상 프로그램의 크기를 축소시킴으로서 프로그램들의 통합 가능성을 상당히 높일 수 있는 효율적인 알고리즘을 제시하였고, 기존의 기법 보다 우수함을 실제 예를 통하여 보였다

An Algorithm for Integrating Versions of Programs using Dynamic Slicing

Soon-Hyung Park† · Eun-Yi Jung†† · Man-Gon Park†††

ABSTRACT

For the operation of the practical software systems, the development, extension and modification phases are successively needed through the new requirements added and their errors debugging detected. Therefore, if it is possible to integrate the reusable software components and to reorganize them into the new software systems, we can improve the productivity, quality, and cost-effectiveness for the development of software systems. For these reasons, it will be required to research and develop methodology and tools for programs integration which make some programs merged effectively. In the process of merging two versions of a program into one, we can consider that some parts of the two versions are equivalent under the non-interference between them. According to the previous studies the program selected to determine the criterion of non-interference was very large, so we have difficulty in integrating many programs practically. In this paper, we propose a new improved algorithm which can reduce the related program size of non-interference criterion by use of dynamic slicing techniques for integration of two versions of a program, and prove the efficiency of the proposed algorithm by showing some examples.

† 정 회 원 : 동의공업대학 전자계산과 교수
†† 준 회 원 : 춘해대학 멀티미디어정보과 교수
††† 종신회원 : 부경대학교 컴퓨터멀티미디어공학부 교수

논문접수 : 1999년 3월 30일, 심사완료 : 2000년 2월 14일

1. 서 론

프로그램의 개발과 수정을 효율적으로 수행하기 위하여 기존의 프로그램들 중에서 재사용 가능한 컴포넌트들을 통합하거나 보완되는 부분만을 새로 개발한 다음 이것을 기존의 프로그램들과 통합하는 방식을 취하는 경우가 많이 있다. 그러므로 프로그래머는 여러 부분이 서로 연관되어져 있는 소프트웨어들을 효율적으로 통합할 수 있는 기법의 필요성에 직면하게 된다. 따라서 생산성이 높은 프로그램 통합 방법의 개발과 연구는 매우 필요할 것이다. 통합 프로그램을 산출하기 위해서는 통합 대상 프로그램의 각 특징을 결합한 다음 서로의 비 간섭 여부를 확인하는 절차가 필요하다[1, 7, 8].

기존의 통합 기법들은 프로그램 비 간섭의 기준으로 정적 슬라이스 기법을 사용하였기 때문에 통합이 가능한 프로그램들도 비 간섭 기준을 충족하지 못하여 통합을 못하는 사례가 많이 발생하였다[4, 9, 10]. 프로그램 슬라이싱은 기준 노드의 특정 변수 var의 값에 직 간접적으로 영향을 끼치는 프로그램 P에 있는 모든 명령문들을 찾는 것이다[2, 3, 11]. 본 논문에서는 통합을 위한 비 간섭 기준을 산출하기 위해 동적 슬라이싱 기법을 사용함으로써 간섭 조건을 완화시켰고, 두 통합 대상 프로그램의 통합 조건으로 비 간섭 여부를 확인하기 위한 *Base* 프로그램이 필요 없기 때문에 통합 단계를 줄임으로서 통합 효율을 높일 수 있는 알고리즘을 제시하였다. 그리고, 기존의 기법 보다 통합 가능 확률이 매우 우수함을 실제 예를 통해 보였다.

제2장에서는 실제적인 통합 환경에서 동적 슬라이싱 기법의 도입 필요성에 대해 설명하였다. 3장에서는 비 간섭 기준을 제시하였고, 비 간섭 점증에 사용될 different nodes, 동적 슬라이스와 실행이력 등을 설명하였다. 4장에서는 기존의 프로그램 통합 기법 5단계를 실제 예를 들어 자세히 설명하였고, 5장에서는 본 논문에서 제시한 동적 슬라이싱을 통한 프로그램 통합 기법에 대해 다루었으며, 알고리즘과 함께 실제 예를 통해 자세히 설명하였다. 6장에서는 본 논문에서 제시한 기법의 장점과 효율성에 대해 기존 기법과 비교 설명하였다.

2. 동적 슬라이싱 기법의 필요성

프로그램 재사용이란 새로운 프로그램을 작성하기 위해 기존의 모듈을 통합하여 우리가 필요로 하는 프로그램을 만드는 것이다. 그러나, 통합 대상이 되는 프로그램 모듈에는 우리가 실제적으로 필요로 하는 것이 전체가 아닌 단지 일부분만 필요로 하는 경우가 많다. 그러므로 통합하는 과정에 동적 슬라이싱 기법을 적용하여 우리가 필요한 부분만 선택하는 것이 필요하다. 다음의 예를 살펴보자 (그림 1)은 보증 수수료를 산출하는 예제 프로그램이다.

```
      begin
S1        read (guarantee),
S2        read (type)
S3.       if type = "중소기업"
          then
S4.             rate := 0.01
          else
S5.             rate := 0.015
          end_if
S6:       commission := rate * guarantee;
S7        write (commission)
      end
```

(그림 1) 예제 프로그램

(그림 1)의 프로그램은 모든 기업에 대해 보유기술을 담보로 보증사업을 하는 기술신용보증회사의 프로그램 모듈인데 비해 이 모듈을 이용한 통합 프로그램 사용자의 회사는 중소기업만을 대상으로 하는 중소기업보증회사라면 위 프로그램 중 중소기업에 해당하는 부분만 필요하다. 그러므로 S2의 입력자료 type의 값으로 "중소기업"을 준다면 프로그램 통합 대상이 되는 프로그램 모듈의 크기는 작아질 것이고 따라서 통합의 효율성은 커지게 된다. (그림 1)의 예제 프로그램에서 슬라이싱 기준 S7에 대한 동적 슬라이싱 프로그램 P를 구하면 다음과 같다

$$P/_{S7} = \{S1, S4, S6, S7\}$$

본 논문에서는 입력 자료까지 고려한 동적 슬라이싱 개념을 사용함으로서 보다 현실적인 통합 기법을 제공하였다.

3. 비 간섭 기준과 슬라이스

(1) different nodes

어떤 대상 프로그램 P를 기준으로 할 때 또 다른

대상 프로그램과의 사이에 존재하는 差(difference)의 집합을 슬라이스한 결과를 의미한다.

- P_B에 대한 P_A의 different nodes $DN_{A,B}$는 P_A에서 P_B와의 nodes의 差의 집합을 슬라이스한 결과이다.

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

(2) 실행 이력

실행 이력이란 주어진 시험사례를 실행하는 동안 방문된 순서에 의한 일련의 노드들인 $\{n_1, n_2, \ldots, n_m\}$의 집합이다. 입력 x에 대한 프로그램 P의 실행 이력 H_x에 포함된 position p에서 노드 Y는 Y^p로 표현된다[2, 3].

(3) 동적 슬라이스

기준 변수를 c라 한다면 H_x와 c에 관한 P의 동적 슬라이스는 어떤 실행이 실행의 종료 시점에서 관찰되어진 것처럼 c의 값에 어떤 영향을 끼치는 H_x에서의 모든 명령문들의 집합이다. P/c는 c를 기준 변수로 하는 프로그램 P에서의 슬라이스의 집합이다[5, 6].

- P_M/ $DN_{A,B}$는 $DN_{A,B}$를 기준 변수로 하는 P_M에서의 슬라이스의 집합을 의미한다.

(4) 비 간섭 기준

통합 프로그램 P_M내에서 보존되어진 A와 B의 변화된 동작을 위해 유지되어져야 하는 비 간섭 기준은 "조건(1) and 조건(2)"이다.

- 조건(1) $P_M/DN_{A,B}=P_A/DN_{A,B}$
- 조건(2) $P_M/DN_{B,A}=P_B/DN_{B,A}$

(5) DEF와 USE

DEF(n)은 노드 n에서 바뀌어진 값을 가진 변수의 집합이고, USE(n)은 노드 n에서 사용된 값을 가진 변수의 집합이다. 프로그램 P의 $LastVar(P)$는 assign type에서의 last DEF(n)과 write type의 USE(n)의 집합이다.

(6) def-order nodes

같은 변수를 정의하는 assign type 노드의 집합이다.

4. Susan Horwitz의 프로그램 통합 기법

기존의 프로그램 통합 기법 중 대표적인 Susan Horwitz의 자동통합 프로그램 버전의 의미기반 분의 설계에서 주장한 통합 기법을 소개한다[1, 4]

4.1 Susan Horwitz의 통합 단계

(1) 1 단계

두 개의 프로그램의 비 간섭 여부를 확인하기 위해 두 프로그램의 공통 부분으로 구성된 *Base* 프로그램(P_{Base})을 만든다. 두 프로그램(프로그램 *A*, 프로그램 *B*)을 *Base* 프로그램의 변형(variant) 프로그램이라고 부른다.

(2) 2 단계

P_{Base}에 대한 P_A와 P_B에서의 different node인 $DN_{A,Base}$와 $DN_{B,Base}$를 결정한다.

(3) 3 단계

앞 단계에서 만들어진 different nodes를 기준 변수로 사용하여 P_A와 P_B를 결합한 통합 프로그램 P_M을 만든다

(4) 4 단계

*A*와 *B*가 *Base*와 관련하여 간섭을 하는지를 결정하며, 비 간섭 기준은 "조건(1) and 조건(2)"이다.

- 조건(1) : $P_M/DN_{A,Base}=P_A/DN_{A,Base}$
- 조건(2) $P_M/DN_{B,Base}=P_B/DN_{B,Base}$

(5) 5 단계

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

4.2 Susan Horwitz 통합기법의 문제점

(1) 정적 슬라이싱(static slicing) 기법을 사용하여 비 간섭 기준을 만들었기 때문에 비 간섭 기준 파일의 크기가 커진다. 비 간섭 기준의 파일이 커질수록 통합을 위해 체크해야할 범위가 그만큼 커지기 때문에 통합 가능성이 낮아진다

(2) 통합 대상이 되는 프로그램에 대한 base program

이 존재해야 하거나 존재하지 않는다면 만들어야 한다. base program은 비 간섭기준을 만드는데 사용되며 비 간섭 여부를 확인하는 과정에서도 사용된다.

4.3 예 제

(1) 1 단계

(그림 2)는 Base 프로그램이고, 이 프로그램의 두 변형 프로그램이 (그림 3)과 (그림 4)에 나타나 있다.

```
        begin
S1          SUM := 0;
S2:         I := 3
S3:         while I < 11
            do
S4:             SUM := SUM + I;
S5:             I := I + 3
S6:         while_end
S7:         write(SUM)
        end.
```

(그림 2) *Base* 프로그램

```
        begin
A1:         PROD :=1;
A2:         SUM := 0;
A3:         I := 3
A4:         while I < 11
            do
A5:             PROD := PROD * I;
A6:             SUM := SUM + I;
A7:             I := I + 3
            while_end
A8:         write(SUM);
A9:         write(PROD)
        end.
```

(그림 3) 변형 프로그램 *A*

```
        begin
B1:         SUM := 0;
B2:         N := 0;
B3:         I := 3
B4:         while I < 11
            do
B5:             SUM := SUM + I;
B6:             N := N + 1;
B7:             I := I + 3
            while_end
B8          write(SUM);
B9:         write(N)
        end
```

(그림 4) 변형 프로그램 *B*

(2) 2 단계

① $DN_{A,Base}$

```
PROD := 1;
PROD := PROD * I;
write(PROD)
```

② $DN_{B,Base}$

```
N := 0;
N := N + 1;
write(N)
```

(3) 3 단계

P_A와 P_B를 결합하여 만들어진 통합 프로그램 P_M이 (그림 5)에 나타나 있다.

```
        begin
M1:         PROD :=1;
M2:         SUM := 0;
M3:         N := 0;
M4:         I := 3
M5:         while I < 11
            do
M6:             PROD := PROD * I;
M7:             SUM := SUM + I;
M8:             N := N + 1;
M9:             I := I + 3
            while_end
M10:        write(SUM);
M11:        write(PROD);
M12:        write(N)
        end.
```

(그림 5) 통합 프로그램 *M*

(4) 4 단계

*A*와 *B*가 *Base*와 관련하여 간섭을 하는지를 비 간섭 기준을 통해 결정한다.

① $P_M/DN_{A,Base}$

```
        begin
M1:         PROD :=1;
M4:         I := 3
M5:         while I < 11
            do
M6              PROD :=PROD * I;
M9              I := I + 3
            while_end
M11         write(PROD)
        end
```

② $P_A/DN_{A,Base}$

```
        begin
A1:         PROD :=1;
```

```
A3:    I := 3
A4:    while I < 11
       do
A5:       PROD :=PROD * I;
A7:       I := I + 3
       while_end
A9:    write(PROD)
     end.
```

- ① = ②이므로 비 간섭 기준(1)을 만족

③ $P_M/DN_{B,Base}$

```
     begin
M3:    N := 0;
M4:    I := 3
M5:    while I < 11
       do
M8:       N = N + 1;
M9:       I := I + 3
       while_end
M12:   write(N)
     end
```

④ $P_B/DN_{B,Base}$

```
     begin
B2:    N := 0;
B3:    I := 3
B4:    while I < 11
       do
B6:       N := N + 1;
B7:       I := I + 3
       while_end
B9:    write(N)
     end
```

- ③ = ④이므로 비 간섭 기준(2)를 만족
- 비 간섭 조건(1)과 (2)를 모두 만족시키므로 프로그램 A와 B에는 통합 프로그램 M에 대하여 간섭이 존재하지 않는다.

(5) 5 단계

(그림 5)의 통합 프로그램 M에서 deforder node가 존재하지 않으므로 (그림 5)는 완전한 통합 프로그램이다.

5. 효율적인 프로그램 통합 기법

5.1 효율적인 프로그램 통합 단계

본 논문에서 제시한 프로그램 통합 기법은 크게 3부분으로 나눈다

(1) 통합 대상 프로그램의 기준변수 결정:
통합 대상 프로그램 A와 B간의 different nodes 즉, 기준변수를 구하는 단계

(2) 프로그램의 통합

(3) 대상 프로그램의 비 간섭 여부 결정
대상 프로그램 A와 B의 변화된 동작이 P_M내에서 보존되어지는 것을 확실하게 증명하는 단계로서 우리는 P_A와 P_B의 동적 슬라이스를 P_M과 비교함으로써 비 간섭 여부를 결정한다. 비 간섭 기준은 3장에서 소개되어 있다.

그리고, 통합 단계의 구체적인 절차 7단계는 다음과 같다.

(1) 대상 프로그램에서 더 이상 영향을 줄 수 없는 라스트 변수(*LastVar*)를 각각 구한다.
- LastVar(A)
- LastVar(B)

(2) 대상 프로그램들의 different nodes(DN) 즉, 기준변수를 각각 구한다
- $DN_{A,B}$:
  $LastVar(A)$에서 $LastVar(B)$의 差
  $DN_{A,B} = LastVar(A) - LastVar(B)$
- $DN_{B,A}$:
  $LastVar(B)$에서 $LastVar(A)$의 差
  $DN_{B,A} = LastVar(B) - LastVar(A)$

(3) 대상 프로그램들의 실행이력(H)을 각각 구한다.
- H_A
- H_B

(4) 대상 프로그램들의 동적 슬라이스를 각각 구한다
- $P_A/DN_{A,B}$
- $P_B/DN_{B,A}$

(5) backtracking 방법으로 두 프로그램간의 등적 link를 구한다.
- 동적 링크 관계도
- 마킹 변수에 의한 매칭 결과표

(6) 매칭 결과표를 이용하여 통합 프로그램을 산출한다.

(7) 대상 프로그램과 통합 프로그램간의 비 간섭 여부를 각각 확인

5.2 효율적인 프로그램 통합 알고리즘

```
procedure FindCreterion
    LastVar(a, lat_b, last_a, cre_b)
    LastVar(b, last_b)
    Creterion(last_a, last_b, cre_a)
    Creterion(last_b, last_a, cre_b)
end.

procedure IntegrateProg
    DynamicMark(a, in_data, cre_a, mark_a)
    DynamicMark(b, in_data, cre_b, mark_b)
    MatchTbl(mark_a, mark_b, match_prog)
    MergeProg(match_prog, depend_data,
                marge_prog)
end.

procedure CheckInterfere
    DynamicSlice(a, in_data, cre_a, slice_a)
    DynamicSlice(merge_prog, in_data, cre_a,
                slice_merge)
    if (slice_a = slice_merge) then
      DynamicSlice(b, in_data, cre_b, slice_b)
      DynamicSlice(merge_prog, in_data,
                cre_b, slice_merge)
      if (slice_b = slice_merge) then
        return(merge_prog)
      else
        return("false")
      end_if
    else
      return("false")
    end_if
end.
```

프로그램 *FindCreterion*은 통합 대상 프로그램의 기준변수를 구하는 프로그램으로 2개의 함수로 구성되어 있다. *LastVar*은 인수1 프로그램에서 *LastVar*(인수1)을 찾는 함수이며 *Creterion*은 인수1에서 인수2에 대한 差의 집합을 구하여 인수3에 저장하는 함수이다.

프로그램 *IntegrateProg*는 통합 프로그램을 만드는 프로그램으로서 3개의 함수로 구성되어있다. *DynamicMark*는 in_data을 입력으로 하는 인수1에서 인수3을 기준 변수로 하는 동적 슬라이스를 구하기 위한 마킹 테이블(인수4)을 만드는 함수이며 함수 결과에 대한 예가 <표 1>에 나타나 있다. *MatchTbl*은 마킹 테이블(인수1, 인수2)들 간에 동적 링크 관계를 이용하여 match 프로그램(인수3)을 만드는 함수이며 그 예가 <표 2>에 나타나 있다 *IntegrateProg*는 match 프로그램(인수1)을 종속 관계(인수2)를 이용하여 통합 프로그램(인수3)을 만드는 함수이다 프로그램 *CheckInterfere*는 두 대상 프로그램의 비 간섭 여부를 결정하는 프로그램으로 비 간섭이면 merge_prog를 그렇지 않으면 "false"를 return한다 *DynamicSlice*는 in_data를 입력으로 하는 인수1(프로그램)에서 인수3을 기준 변수로 하는 동적 슬라이스(인수4)을 구하는 함수이다. 이렇게 구해진 동적 슬라이스를 이용하여 비 간섭 여부를 판단한다.

5.3 예 제

통합 대상 프로그램 A는 (그림 6)에 있는 프로그램 A이고 통합 대상 프로그램 B는 (그림 7)에 있는 프로그램 B라고 가정하자

(1) 프로그램 A와 B의 *LastVar*을 구한다.

- $LastVar(A)$ = {A10, A11}
- $LastVar(B)$ = {B9, B10}

(2) different nodes 결정

- $DN_{A,B}$ = {A10}
- $DN_{B,A}$ = {B10}

(3) 프로그램 A와 B의 실행이력(H)을 구한다.

- V = 3이고, N = {-2, 4, -3}이라면

H_A = {1, 2, 3^1, 4^1, 5^1, 6^1, 8^1, 9^1, 10, 3^2, 4^2, 5^2, 8^2, 9^2, 10, 3^3, 4^3, 5^3, 6^2, 8^3, 9^3, 10, 3^4}

H_B = {1, 2, 3, 4^1, 5^1, 6^1, 7^1, 8^1, 9^1, 4^2, 5^2, 6^2, 7^2, 8^2, 9^2, 4^3, 5^3, 6^2, 7^3, 8^3, 9^3, 4^4, 10}

```
        begin
A1:         read(V)
A2          I := 1;
A3.         N := 1
A4:         while (I <= V)
            do
A5:             read(N)
A6.             if (N < 0)
                then
A7:                 T := f1(N)
                else
A8:                 T := f2(N)
                end_if;
A9:             Y := f3(T);
A10.            write(Y).
A11             I := I - 1
            end_while;
        end.
```

(그림 6) 예제 프로그램 A

```
        begin
B1:         read(V)
B2:         I := 1;
B3:         Q := 0
B4:         while (I <= V)
            do
B5:             read(N)
B6:             T := f1(N);
B7:             Y := f3(T);
B8:             Q := Q + Y;
B9:             I := I + 1
            end_while
B10:        write(Q)
        end.
```

(그림 7) 예제 프로그램 B

(4) A와 B의 동적 슬라이스를 구한다.

- $P_{A}/DN_{1,B}$
  = {A1, A2, A4, A5, A7, A9, A10, A11}
- $P_{B}/DN_{B,4}$
  = {B1, B2, B3, B4, B5, B6, B7, B8, B9, B10}

〈표 1〉 동적 링크 관계도

프로그램 A			동적링크	프로그램 B		
순번	실행 이력	마킹		마킹	실행 이력	순번
1	A1	✓	→	✓	B1	1
2	A2	✓	→	✓	B2	2
3	A3			✓	B3	3
4	A4	✓	→	✓	B4	4
5	A5				B5	5
6	A6				B6	6
7	A7				B7	7
8	A9				B8	8
9	A10				B9	9
10	A11	✓	→	✓	B4	10
11	A4	✓	→	✓	B5	11
12	A5				B6	12
13	A6				B7	13
14	A8				B8	14
15	A9				B9	15
16	A10		→	✓	B4	16
17	A11	✓				
18	A4				B5	17
19	A5	✓	→	✓	B6	18
20	A6			✓	B7	19
21	A7	✓	→	✓	B8	20
22	A9	✓			B9	21
23	A10	✓			B4	22
24	A11			✓	B10	23
25	A4					

〈표 2〉 마킹 변수에 의한 매칭 결과표

순번	Node 번호
1	A1, B1
2	A2, B2
3	B3
4	A4, B4
5	A11, B9
6	A5, B5
7	A7
8	B6
9	A9, B7
10	A10
11	B8
12	B10

(5) backtracking 방법으로 두 프로그램간의 동적 link를 구한다

- <표 1>은 프로그램 A와 B에 대한 동적 링크 관계도이다.
- <표 2>는 <표 1>의 동적 링크 관계도에서 마킹 변수에 의한 매칭 결과표이다.

(6) <표 2>의 매칭 결과표를 이용하여 통합 프로그램 M을 산출한다

- (그림 8)은 통합 프로그램 M이다.

```
        begin
M1:         read(V)
M2:         I := 1;
M3:         Q := 0
M4:         while (I <= V)
            do
M5:             read(N)
M6:             T := f1(N);
M7:             Y := f3(T);
M8:             write(Y);
M9              Q := Q + Y;
M10:            I := I + 1
            end_while
M11:        write(Q)
        end
```

(그림 8) 통합 프로그램 M

(7) 대상 프로그램의 비 간섭 여부 결정

① A와 M간의 비 간섭 여부 확인

㉠ $P_{M}/DN_{A,B}$

```
        begin
M1:         read(V).
```

```
M2:      I := 1
M4:      while (I <= V)
         do
M5:           read(N);
M6:           T := f1(N);
M7:           Y := f3(T);
M8:           write(Y);
M10:          I := I + 1
         end_while
      end.
```

㉡ $P_A/DN_{A,B}$

```
      begin
A1:       read(V);
A2:       I := 1
A4:       while (I <= V)
          do
A5:            read(N);
A7:            T := f1(N);
A9:            Y := f3(T);
A10:           write(Y);
A11:           I := I + 1
          end_while
      end.
```

- ㉠=㉡이므로 비 간섭 기준(1)을 만족

② B와 M간의 비 간섭 여부 확인

㉢ $P_M/DN_{B,A}$

```
      begin
M1:       read(V);
M2:       I := 1;
M3:       Q := 0
M4:       while (I <= V)
          do
M5:            read(N);
M6:            T := f1(N);
M7:            Y := f3(T);
M9:            Q := Q + Y;
M10:           I := I + 1
          end_while;
M11:      write(Q)
      end.
```

㉣ $P_B/DN_{B,A}$

- ㉢=㉣이므로 비 간섭 기준(2)를 만족

③ 비 간섭 여부 결론

- 비 간섭 조건(1)과 (2)를 모두 만족시 키므로 프로그램 A와 B에는 통합 프로그램 M에 대하여 간섭이 존재하지 않는다

6. 비교 및 고찰

본 논문의 통합기법이 효율적임을 증명하기 위해 5.3에서 예제로 사용한 대상 프로그램을 Susan Horwitz 기법에 적용시켜 통합여부를 살펴보자. 즉, 통합 대상 프로그램 A는 (그림 6)에 있는 예제 프로그램 A이고 통합 대상 프로그램 B는 (그림 7)에 있는 예제 프로그램 B라고 가정한다.

(1) 1 단계

(그림 9)는 (그림 6)과 (그림 7)에 있는 통합 대상 프로그램의 *Base* 프로그램이다.

```
      begin
S1:       read(V);
S2:       I := 1
S3:       while (I <= V)
          do
S4:            read(N);
S5:            I := I + 1
          end_while
      end.
```

(그림 9) *Base* 프로그램

(2) 2 단계

① $DN_{A,Base}$

```
N := 1,
if (N < 0)
T := f1(N);
T := f2(N);
Y := f3(T),
write(Y)
```

② $DN_{B,Base}$

```
Q := 0;
T := f1(N);
Y := f3(T);
Q := Q + Y;
 write(Q)
```

(3) 3 단계

P_A와 P_B를 결합하여 통합 프로그램 P_M을 만든다. 통합 프로그램 P_M이 (그림 10)에 나타나 있다.

```
        begin
M1:         read(V)
M2:         I := 1;
M3:         N := 1;
M4:         Q := 0
M5:         while (I <= V)
            do
M6:             read(N)
M7:             if (N < 0)
                then
M8:                 T := f1(N)
                else
M9:                 T := f2(N)
                end_if;
M10:            T := f1(N);
M11:            Y := f3(T);
M12:            write(Y);
M13:            Q := Q + Y;
M14:            I := I + 1
            end_while
M15:        write(Q)
        end
```

(그림 10) 통합 프로그램 M

(4) 4 단계

A와 B가 $Base$와 관련하여 간섭을 하는지를 비 간섭 기준을 통해 결정한다.

① $P_M/DN_{A,Base}$

```
        begin
M1:     read(V)
M2:     I := 1
M5:     while (I <= V)
        do
M6:         read(M)
M10:        T := f1(N);
M11:        Y := f3(T),
M12:        write(Y);
M14         I := I + 1
        end_while
    end.
```

② $P_A/DN_{A,Base}$

```
    begin
A1:     read(V)
A2:     I := 1;
A3:     N := 1
A4:     while (I <= V)
        do
A5:         read(N)
A6:         if (N < 0)
            then
A7:             T := f1(N)
            else
A8:             T := f2(N)
            end_if;
A9:         Y := f3(T);
A10:        write(Y);
A11:        I := I + 1
        end_while
    end.
```

- ① ≒ ②이므로 비 간섭 기준(1)을 위배한다. 따라서, 비 간섭 기준의 and 조건을 위배한다. 그러므로 프로그램 A와 B에는 통합 프로그램 M에 대하여 서로 간섭이 있다. 그러므로 통합 될 수 없다.

그러나, 4.3절의 예제와 같은 동일한 조건에서 본 논문에서 제시한 동적 기법을 통한 통합은 비 간섭이므로 통합 가능함을 알 수 있었다. 그리고, 본 논문에서 제시한 기법은 통합할 때 필요 없는 부분은 동적 슬라이싱 단계에서 삭제할 수 있다. 그러므로 본 논문에서 제시한 기법이 기존의 기법보다 효율적이다.

7. 결 론

본 논문에서는 실행이력을 통한 동적 슬라이싱 방법을 사용한 프로그램의 효율적인 통합 기법을 소개하였으며, 예제 프로그램을 통해 기존의 기법 보다 효율적임을 증명하였다. 예제 프로그램은 프로그램의 기본 구성요소인 입력문, 출력문, 반복문, 분기문 그리고, 할당문으로 구성되어있다. 본 논문에서 제시한 기법은 기존의 Susan Horwitz 기법에 비해 네 가지 면에서 효율적이다.

첫째, 기존의 기법에서는 기준 변수 결정을 위해 Base 프로그램이 미리 만들어졌으나 본 논문에서는 Base 프로그램이 필요 없기 때문에 Base 프로그램을 만드는 단계를 줄일 수 있다.

둘째, 자료 종속관계를 고려하여 통합 프로그램을 작성함으로써 통합 후 def-order를 고려할 필요가 없다. 그러므로, 프로그램 통합 단계에서 실행 순서에 대해 검증 단계를 거쳐야하는 기존의 기법 보다 단계를 줄일 수 있어 실행 효율을 높일 수 있다.

셋째, 기존의 기법은 프로그램 그 자체만을 통합하기 때문에 통합 대상이 되는 프로그램의 크기가 매우 컸으나, 본 논문에서는 동적 슬라이스를 통해 통합 대상

프로그램의 크기를 상당히 줄임으로서 비 간섭 대상의 크기가 감소함에 따른 통합 가능성을 상대적으로 높였다. T = 정적 프로그램의 집합, D = 동적 프로그램의 집합이라고 한다면, $D \supset T$가 되고, $\sigma = D - T$에서 σ가 동적 프로그램에서 정적 프로그램을 제외한 프로그램 부분들의 집합이라면 정적 통합에서는 σ 만큼의 비 통합 가능성을 가지게 된다.

넷째, 기존의 통합 기법은 단순한 프로그램의 통합이었는데 비해 본 논문에서 제시한 기법은 필요 없는 프로그램 노드는 삭제하는 프로그램 수정(modification) 기능까지 포함시켰다.

(그림 6)의 예제 프로그램 A와 (그림 7)의 예제 프로그램 B를 기존의 통합 기법을 사용하여 통합한 결과가 (그림 10)에 나타나있다. 그러나, (그림 10)의 통합 프로그램은 $P_{M'}/DN_{A,Base}$와 $P_{N'}/DN_{A,Base}$가 다르기 때문에 비 간섭 조건을 만족시키지 못해 통합이 불가능하다. 즉, 예제 프로그램 A에서 변화된 내용이 통합 프로그램 M에서 보존되지 않았기 때문에 간섭이 발생하였다. 그러나, (그림 6)의 예제 프로그램 A와 (그림 7)의 예제 프로그램 B를 본 논문에서 제시한 기법을 사용하여 통합한 통합 프로그램 (그림 8)은 비 간섭이므로 통합 가능함을 알 수 있었다. 이와 같이 본 논문에서 제시한 통합 기법을 사용하면 실행 효율과 통합 가능 확률을 매우 높일 수 있으며 프로그램의 수정 효과까지 얻을 수 있다.

참 고 문 헌

[1] David Binkley, Susan Horwitz, and Thomas Reps, "Program Integration for Languages with Procedure Calls," ACM Tran. on Software Engineering and Methodology 4, 1, pp.3-35, Jan. 1995.

[2] S. H, Park and Park, M. G., "An efficient dynamic program slicing algorithm and its Application," Proc. of the IASTED International Conference, Pittsburgh, Pennsylvania, pp.459-465, May 1998.

[3] 박순형, 박만곤, "소프트웨어 테스팅을 위한 동적 프로그램 슬라이싱 알고리즘의 효율성 비교", 정보처리논문지 제5권 제9호, pp.2323-2334, 1998.

[4] Susan Horwitz, Jan Prins, and Thomas Reps. "Integrating noninterfering version s of programs," ACM Trans on Progra mming Languages and Systems, pp.345-387, July 1989

[5] Karl J. Ottentein and Linda M. Ottentein. "The program dependence graph in a software development environment." Proc of the ACM SIG SOFT/SIGPLAN Symposium on Practical Software Development Environments, Pittaburgh, Pennsylvania, April 1984

[6] Mark Weiser. "Programmers use slices when debugging," Communications of the ACM, pp.446-452, July 1982.

[7] V. Berzins, "Software merge: Models and methods for combining changes to programs," Journal of Systems Integration, Vol.1, No.2, pp.121-141, August 1990.

[8] V. Berzins, Luqi, and A. Yehudai, "Using transformations in specification-based prototyping," IEEE Transactions on Software Engineering, Vol. Tec 436-452, May 1993.

[9] T. Rep and W. Yang, "The semantics of program slicing and program integration," Proceedings of the Colloquium on Current Issues in Programming Languages, Barcelona, Spain, pp.360-374, March 1989

[10] T Rep, "Algebraic properties of program integration," Proceedings of the Colloquium on Current Issues in Programming Languages, Copenhagen, Denmark, pp.326-340, May 1990.

[11] Bodan Korel, "Computation on Dynamic Program Slices for Unstructured Programs," IEEE Trans. on Software Engineering, Vol.23, No.1, pp.17-34, January 1997.

박 순 형

e-mail sbpark@dit.ac.kr

1981년 울산대학교 공과대학 전자계산학과(학사)

1985년 숭실대학교 대학원 전자계산학과(석사)

1997년~현재 부경대학교 대학원 전자계산학과 박사과정 수료

1981년~1983년 현대미포조선(주) 전산실 근무

1987년~현재 동의공업대학 전자계산과 교수

관심분야 : 소프트웨어공학 및 제공학, 소프트웨어 테스팅, 비지니스 프로세스 재공학, 정보시스템 분석 및 설계, 비주얼 프로그래밍 기법, 멀티미디어 정보시스템 개발방법론

정 은 이

e-mail jey@choonhae.ac.kr

1995년 부경대학교 전자계산학과(공학사)

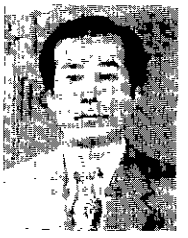
1997년 부경대학교 대학원 전자계산학과(이학석사)

2000년 부경대학교 대학원 전자계산학과(이학박사)

1997년~1999년 동명대학 전자계산과 겸임교수, 제인소프트웨어 대표

2000년~현재 춘해대학 멀티미디어정보과 교수

관심분야 : 소프트웨어공학 및 제공학, 소프트웨어 신뢰성 및 안전성 공학, 웹 기반 멀티미디어 학습 정보시스템, 멀티미디어 정보시스템, 멀티미디어 소프트웨어공학 등



박 만 곤

e-mail mpark@dolphin.pknu.ac.kr

1976년 경북대학교 수학교육학과 졸업(이학사)

1987년 경북대학교 대학원 전산통계학과(이학박사)

1980년~1981년 경남정보대학 전자계산학과 교수

1990년~1991년 영국 리버풀대학교 전자계산학과 객원교수

1992년~1993년 미국 켄사스대학교 컴퓨터공학과 교환교수

1996년 호주 사우스 오스트레일리아대학교 컴퓨터 및 정보과학부 객원교수

1995년 몽골 컴퓨터배핑 전문가로 외무부 파견

1997년 중국 산동성 칭루 정보시스템구축 전문가로 외무부에 의해서 파견

1997년~현재 콜롬보플랜 기술자교육대학(Colmbo Plan Staff College, 필리핀 마닐라에 본부) 정보기술 및 통신학처장으로 정부파견

1981년~현재 부경대학교 컴퓨터 멀티미디어 공학부 교수

관심분야 : 소프트웨어공학 및 제공학, 소프트웨어 신뢰성 및 안전성공학, 비지니스 프로세스 재공학, 멀티미디어 정보시스템, 소프트웨어품질공학, 소프트웨어 메트릭스, 소프트웨어테스팅 및 감사, 결함허용 소프트웨어 시스템