韓國 軍事運營分析 學會誌
第14卷 第1號, 1988. 6.

Program Slice 生成 模型에 關한 硏究

(A Study on the Program Slicing Model)

尹 昌 燮*
車 榮 憲*
鄭 昌 模*

Abstract

Many programmers start debugging by reading the faulty program from start to bottom without investigating carefully the erroneous program. Expert programmers, however, trace backward from a particular variable in a specific statement to identify all possible sources of influence on the value of variable (program slice).

Weiser proposed a slicing algorithm(method) that is complex, iterative and still in modification [3, 4]. This paper presents a method to generate a program slice by use of matrix computation which represents all possible slices of the program. The matrix representation of a program is soundly based on the graph theory of data dependency.

1. 序 論

프로그램 디버깅의 가장 중요한 目標는 소프트웨어 誤謬의 原因을 찾아서 修正하는 것이다. 그런데 이 디버깅을 修行하는 노력에는 비록 동일한 敎育을 받았거나, 유사한 經驗을 가졌다 할지라도 프로그래머의 能力에 따라 상당한 차이가 있다. 일반적으로 프로그래머들이 디버깅을 할 때 誤謬가 發生한 프로그램의 出力을 주의깊게 觀察한 다음에 誤謬가 있는 프로그램 전체를 top-down으로 자세하게 살펴보는 傾向도 있다.[8] 그러나 디버깅의 가장 效果的인 方法은 誤謬가 發生한 地點에서부터 시작하여 誤謬를 發生하게 한 일련의 關聯된 文章들을 살펴보는 것으로써, 이는 control flow를 backward로 찾는 것이다. 이를 working backward

* 國防大學院

라고 한다.[2] 따라서 프로그램이 control flow만으로 表現되어 있는 것보다는 프로그램의 誤謬狀態에 따라 表現될 수 있다면 디버깅하는 데 매우 有用하게 사용될 수 있다. 이러한 觀點에서 Weiser가 프로그램 slice를 提案하게 되었고, 이를 利用하면 디버깅이 훨씬 용이하다.

Weiser의 프로그램 slice는 原始 프로그램을 control flow graph로 변환한 다음 이를 利用하여 變數들의 相關關係 探索을 통하여 구한 原始프로그램의 部分的 集合으로 구하였다. 그러나 變數들의 相關關係가 잘 表現된 DDG(Data Dependency Grap)[5]를 利用하게 되면 보다 쉽게 프로그램 slice를 구할 수가 있다.

本 論文의 目的은 이 DDG를 利用하여 Slice를 구하는 模型을 提示하고자 한다. 제2장에서 프로그램 slice의 槪要와 旣存의 生成方法을 살펴본 다음 제3장에서는 DDG의 理論的 考察을 통해 DDG와 프로그램 slice의 關係를 살펴보고, 제4장에서 DDG를 利用한 프로그램 slice의 生成模型을 提示한다.

프로그램 slice는 對象 프로그램이 structured, unstructured(GOTO사용)이거나 module간(inter procedure), module 내부 (intra procedure) 등을 다루어야 한다. 그러나 本 論文에서는 對象 프로그램을 structured, module 내부만으로 한정하여 다루고 있다.

II. 프로그램 Slice의 生成方法

이 장에서는 Weiser가 提示한 프로그램 slice에 관한 背景 및 定義 그리고 特性에 관하여 설명한다.

1. 프로그램 Slice의 槪要

Weiser는 프로그램에 誤謬를 임의로 挿入하여 숙련된 프로그래머들에게 디버깅을 요구한 결과, 誤謬가 發生한 地點으로부터 그 變數에 影響을 주지 않는 文章들을 除去하면서 作業(프로그램 slicing)하는 過程을 발견하였다.[2]

프로그램 slice란 프로그램 slicing을 통하여 남은 文章들, 즉 誤謬가 發生한 文章에서 關聯되는 變數에 影響을 미치는 文章들로만 構成되는 프로그램의 部分的인 集合이다. 따라서 이 프로그램 slice는 프로그램 施行狀態 중에서 특별히 주어진 入力에서는 原始 프로그램을 代表할 수 있는 獨立的인 프로그램이라 할 수 있다.

```
1     BEGIN
2     READ(X, Y)
3     TOTAL : =0.0
4     SUM : 0.0
5     IF X < =1
        THEN
6           SUM : =Y
      ELSE BEGIN
7           READ(Z)
8           TOTAL : =X * Y
          END
9     WRITE(TOTAL, SUM)
10    END.
```

예제.1. 예제 프로그램

프로그램내에는 slicing條件(criterion)에 따라 서로 상이한 많은 프로그램 slice가 存

在한다. 또한 프로그램에는 原始 프로그램 자체가 프로그램 slice가 되는 적어도 하나 이상의 프로그램 slice가 반드시 存在하게 된다. 이와 같은 프로그램 slicing槪念을 설명하기 위하여 위 예제.1프로그램을 利用한다.

예제.1의 10번째 文章에서 Z變數를 중심으로 디버깅하고자 프로그램 slicing條件을 〈10, {Z}〉로 提示한 경우, 生成되는 프로그램에 대한 slice는 예제.1의 '가'와 같다.

```
 1      BEGIN
 2      READ(X, Y)
 5      IF X < =1
            THEN
            ELSE
 7              READ(Z)
10      END.
```

예제.1의 가. Slice on Criterion 〈10, {Z}〉

프로그램을 디버깅하는 프로그래머들은 이 예제 프로그램에서 10번째 文章의 Z變數가 7번째 文章에서 定義되었고, 이는 또한 5번째 文章인 IF문의 ELSE 實行經路上에서 發生하였음을 발견하게 된다. IF문의 ELSE 部分은 X變數에 의하여 결정된 것이므로 X 變數의 入力에 해당되는 2번째 文章도 프로그램 slice에 包含하게 된다. 프로그램 slice에 包含되는 2, 5, 7번째 文章들과, 關係되는 프로그램의 文法에 맞추어 1, 10번째 文章을 包含하여 완전한 프로그램 slice를 만들게 된다.

예제.1의 '나'와 '다'도 동일한 方法에 의하여 프로그램 slice를 生成한 경우인데 '나'의 경우는 8번째 文章에서 X變數는 2번째 文章에서 定義되었으므로 2번째 文章만을 프로그램 Slice에 包含하였다. '다'의 경우는 10번째 文章에서 TOTAL 變數를 중심으로 프로그램 Slice를 찾고자 하는 것인데 5번째 IF문에 의하여 TOTAL 變數의 값이 3번째 文章의 初期値를 갖거나, 8번째 文章의 X와 Y變數에 의한 값을 가질 수 있으므로 3, 5, 8번째 文章이 프로그램 slice에 包含된다. 또한 5번째 文章에서 參照된 X變數의 入力에 해당되는 2번째 文章을 包含하여 프로그램 slice를 구하면 '다'와 같은 결과를 얻을 수 있고, 이를 利用하여 디버깅을 하게된다.

```
 1      BEGIN
 2      READ(X, Y)
10      END.
```

예제.1의 나. Slice on Ctiterion 〈8, {X}〉

```
 1      BEGIN
 2      READ(X, Y)
 3      TOTAL : =0.0
 5      IF X < =1
            THEN
            ELSE
 8              TOTAL : =X * Y
10      END.
```

예제.1의 다. Slice on Criterion 〈10, {TOTAL}〉

2. 旣存의 프로그램 slicing方法

Weiser의 프로그램 slicing方法은 어떤 文

章에 影響을 주는 直·間接 關係變數(R)와 分岐文章(Branch Statement : B)을 통하여 프로그램 slice를 구하는 方法이라 할 수 있다.[3, 4]

1) 關係變數

slicing의 條件과 變數들 간의 關係를 예를 들어 설명하면,

```
    Y : =X
      :
      :
n:  Z : =Y
```

에서 첫번째 文章 이전의 X의 값이 n번째 文章 이후의 Z의 값에 影響을 준다. 이러한 경우에 X가 文章 n에서 구하고자 하는 slice의 直接 關係變數라고 한다.

定義 : slicing條件을 C=⟨i,V⟩라 할 때,

i : 文章번호

V : 프로그램내의 變數들

이러한 경우 關係變數 $R^0_c(n)$은 다음과 같이 定義된다. ····· (1)

1. n=i and v is in V

혹은 2. n이 m의 immediate predecessor일 때

a) v is in BEF(n) and there is a w in both DEF(n) and $R^0_c(m)$

혹은 b) v is not in DEF(n) and v is in $R^0_c(m)$

(REF(n), DEF(n)은 n번째 文章에서 각각 參照되었거나 定義된 變數의 集合)

2) 프로그램 slice

關係變數가 구해지면 이 R^0_c에 의해서 프로그램 slice에 包含될 文章들, S^0_c는

S^0_c = all nodes such that $R^0_c(m) \cup DEF(n) \neq \phi$ ····· (2)

(ϕ기호는 空集合을 나타냄)

그러나 S^0_c에서는 分岐文章, B에 대한 사항은 包含되지 않으므로, 다음 公式에 의해 分岐文章을 식별한다.

B^0_c = all nodes b such that $n \in S^0_c$ and $n \in INFL(b)$ ····· (3)

(INFL(b)는 b文章의 Boolean expression의 影響내에서 存在하는 文章들의 集合)

그런 다음 B^0_c에 影響을 주는 文章들, 즉 criterion에 명시된 變數에 間接的으로 影響을 주는 間接 關係變數를 구한 다음 다시 프로그램 slice에 包含될 文章들을 식별한다.

$BC(b) = \langle b, REF(b) \rangle$ ····· (4)

$R^{i+1}_c(n) = R^0_c(n) \cup R^0_{BC(b)}(n)$ ····· (5)

($R^{i+1}{}_c(n)$는 直接 關係變數에 間接 關係變數가 包含되는 關係變數이고, i+1은 間接的인 影響度를 의미한다.)

$S^{i+1}{}_c$ = all nodes n such that $n \in B^i{}_c$ or $R^{i+1}{}_c(m) \cap DEF(n) = \phi$ ····· (6)

$B^{i+1}{}_c$ = all node b such that $n \in S^{i+1}{}_c$ and $n \in INFL(b)$ ····· (7)

위에서 定義된 내용을 예제 2를 이용하여 설명하면 아래와 같다.

```
1       BEGIN
2       K = 1
3       WHILE K < 10 DO
            BEGIN
4               X = K + 1
5               K = K + 1
            END
6       Z = X + Y
7       WRITE(Z)
8       END.
```

예제.2. 예제 프로그램

예제.2는 Z變數의 값을 계산하는 것으로서, Z變數의 값은 X와 Y變數의 값을 합하여 구하는데, Y變數의 값은 初期値를, X變數의 값은 K變數의 값이 10보다 클때까지 반복한 다음 K變數의 값을 합하여 Z變數의 값을 구하는 프로그램이다.

〈표1〉 Criterion = 〈4, {X}〉에 의한 參照表

n	DEF	REF	R^0_C	INFL	$R^0_{BC(b)}$	R^1_C
2	K		{X}	ϕ	ϕ	{X}
3		K	{X}	{4,5}	{K}	{X,K}
4	X	K	{X}	ϕ	{K}	{X,K}
5	K	K	{X}	ϕ	{K}	{X,K}
6	Z	X,Y		ϕ	{K}	{X,K}
7		Z		ϕ		

예제.2에 Weiser의 프로그램 slicing方法을 適用하기 위하여 표.1과 같은 參照表를 사용하고 있다. 이 參照表는 4번째 文章의 X變數를 중심으로 프로그램 slicing條件을 提示하였을 때 프로그램 slice를 구하기 위한 것이다. 이 參照表에서 n의 열은 文章番號이고, DEF, REF 열은 각각 n번째의 文章에서 定義되었거나 參照된 變數들을 말한다. 直接關係變數, R^0_C 열은 위 公式(1)을 適用하여, 즉 프로그램 slicing條件에서 주어진 〈4, {X}〉와 關係되는 文章別 直接 關係變數만을 찾은 것이고, 이 R^0_C와 DEF가 公式(2)에 適用되어 分岐文章이 包含되지 않은 slice인 S^0_C가 구해진다. INFL은 分岐文章의 影響을 받는 文章들을 말하는 것으로, 分岐文章이 아닌 文章에서의 INFL은 空集合이 된다. 이 INFL과 S^0_C가 公式(3)에 適用되어 프로그램 slice에 包含될 B^0_C를 결정한다. 그 다음 公式(4)에 의하여 새로운 프로그램 slicing條件인 BC(b), 즉, BC(b) = 〈3, {k}〉가 결정되고, 이를 다시 公式(1)에 適用하여 參照表의 $R^0_{BC(b)}$의 값을 生成한다. 이 $R^0_{BC(b)}$와 R^0_C를 公式(5)에 適用하여 R^1_C를 生成하는데, R^1_C의 '1'은 예제 프로그램의 4번째 文章 X에 間接的으로 影響을 주는 정도를 의미한다. 다시 公式(6)과 (7)를 반복 適用하여 앞에서 구

한 프로그램 slice와 그 다음에 구한 프로그램 slice가 같을 경우에 作業을 종료한다. 이와 같은 作業을 통하여 구하여진 프로그램 slice가 예제.2의 '가'와 같다.

```
1       BEGIN
2       K = 1
3       WHILE K < 10 DO
            BEGIN
4               X = K + 1
5               K = K = 1
            END
8       END.
```

예제.2의 가. 프로그램 Slice on Criterion <4, {X}>.

위에서 살펴 본 바와 같이 Weiser의 方法은 프로그램 slicing條件에 따라 프로그램 내의 變數들간의 關係를 찾기위해 原始 프로그램을 control flow graph로 만든 다음, 이를 가지고 文章과 文章間의 實行經路를 Search하여 直·間接 關係變數를 찾아 프로그램 slice를 만드는 方法이다.

III. Data Dependency Graph의 理論的 考察[5]

1. DDG의 概要

Bieman[5]이 提示한 DDG는 프로그램내의 모든 變數의 dependency를 表現하는 directed graph이다.

定義 : DDG = <V, E>, where V is a set of variable definition in program P and E is a set of data dependency

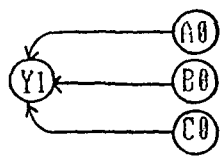
DDG에서 Vertex(V)는 變數의 定義를 말하고, Edge(E)는 變數와 變數간의 dependency를 말한다. dependency에는 direct dependency와 control dependency의 두가지 유형이 있는데, 이에 대한 설명은 아래와 같다.

direct dependency는 Y=f(X)에서 Y와 X의 關係(Y는 從屬變數, X는 獨立變數)이고, control dependency는 分岐文章의 Boolean expression에 사용된 變數와 그 變數의 眞僞값에 따라 發生한 實行經路상에서 定義된 變數들과의 關係를 말한다.

이를 예를 들어 설명하면 아래와 같다.

예1) direct dependency

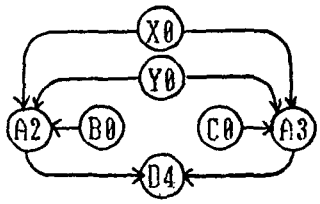
```
1 Y : =f(A, B, C)
```



變數 Y는 1번째 文章에서 初期値 A, B, C 變數들의 값에 의해 定義되므로 direct dependency가 存在한다.

예2) control dependency

```
1    IF X < Y
2        THEN A : =B
3        ELSE A : =C
4    D : =A
```



2번째 文章의 A變數는 B變數의 初期値, 3번째 文章의 A變數는 C變數의 初期値에 의해서 定義되므로 A2와 B0, A3와 C0 사이에는 direct dependency가 存在한다. 그리고 1번째 文章에서 參照된 X變數와 Y變數의 初期値에 의하여 결정된 眞僞값에 따라 實行經路가 정해지므로 實行經路상에 存在하는 定義된 變數, 즉 A2와 A3에는 X0와 Y0에서의 control dependency가 存在한다.

2. DDG의 構成方法

*Bieman*은 文章n에서 Live Definition, L(n)과 Structured Control Influence Definition, C(n), 그리고 Unstructured GOTO Control Influence Definition, G(n)을 구하여 L U C(n) U G(n)을 生成한다. 이 L U C(n) U G(n)이 Source Definition (S)가 되고, 이 S로부터 각 文章에서 定義된 Destination Definition으로 edge가 生成되어 DDG를 構成한다.

Bieman이 提案한 DDG 模型은 原始 프로그램을 compile 단계인 scanning 過程에서 生成되는 자료들을 利用함으로 DDG를 구할 수 있게한다. 이 DDG 生成過程에 대하여 언급하기 전에 關聯되는 몇가지 용어를 定義하면 다음과 같다.

定義 : 1. Live Definition : 하나의 變數 定義가 n번째 文章까지의 實行經路상에서 再定義 되지 않는다면 그 變數의 定義는 n번째 文章까지 이르는 각각의 文章에서 Live Definition된다고 한다.[9]

2. $\ell(x, i)$: 프로그램에서 定義된 變數들 중에서 x번째 文章이 實行되기 직전에 live되는 變數들의 集合을 말한다.

3. $\ell(x, f)$: 프로그램에서 x번째 文章이 實行된 직후에 live되는 모든 變數들의 集合이다.

그림.1은 II.2.의 예제.2의 프로그램을 DDG로 表現한 것이다.

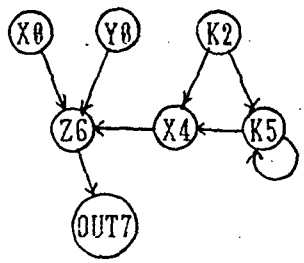


그림.1. 예제.2에 대한 DDG

그림.1과 같은 DDG를 生成하기 위하여 먼저 예제 프로그램의 文章번호에 따라 표.2에서 보는 것과 같은 Live Definition Table을 만든다.

<표3>에서의 Destination Definition은 각 文章에서 定義된 變數들을 나타내고, L은 각 文章에 live되는 變數들 중에서 參照되는 變數들로서, 표.2에서 찾는다. C(n)은 control influence가 있는 곳에서만 나타나는 것으로, 예를 들어 4번째 文章에서의 X變數는 2번째 文章의 K變數와 5번째 文章의 K

〈표2〉 Live Definition Table(for 예제.2)

1(1,i)	X0,Y0	1(1,f)	X0,Y0
1(2,i)	X0,Y0	1(2,f)	X0,Y0,K2
1(3,i)	X0,Y0,K2,X4,K5	1(3,f)	X0,Y0,K2,X4,K5
1(4,i)	X0,Y0,K2,X4,K5	1(4,f)	Y0,K2,X4,X5
1(5,i)	Y0,K2,X4,K5	1(5,f)	Y0,X4,K5
1(6,i)	X0,Y0,K2,X4,K5	1(6,f)	X0,Y0,K2,X4,K5,Z6
1(7,i)	X0,Y0,K2,X4,K5	1(7,f)	X0,Y0,K2,X4,K5,Z6
1(8,i)	X0,Y0,K2,X4,K5	1(8,f)	X0,Y0,K2,X4,K5,Z6

〈표3〉 DDG생성 Table

Destination Definition	L	C(n)	G(n)	Source DEF LUC(n) UG(n)
K2	•	•	•	•
X4	K2,K5	K2,K5	•	K2,K5
K5	K2,K5	K2,K5	•	K2,K5
Z6	X0,Y0,X4	•	•	X0,Y0,X4
OUT7	Z6	•	•	Z6

變數의 값의 影響을 받고 있음을 알 수 있다. 〈표3〉에서의 Source DEF은 L과 C(n), 그리고 G(n)의 合集合으로 Destination Definition으로 edge를 그려주면 (그림1)에서 보는 것과 같은 DDG를 얻을 수 있다.

3. DDG와 slice의 關係

DDG는 프로그램에서 이용되는 變數들(data objects)간의 關係를 분석하는 추상적 模型이다. Bieman은 DDG를 이용하여 프로그램의 資料關係 복잡도(data dependency complexity)를 측정하는데 사용하였다.[9]

이것은 Weiser의 slice 개념에서의 變數간의 影響, 즉 data dependencey 關係는 잘 묘사하고 있지만, control flow에 관한 내용이 빠져 있으므로 완전한 프로그램 slice는 구할 수 없다.

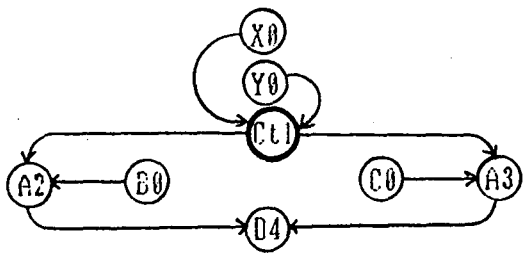
Ⅳ. 프로그램 slice 生成模型의 提示

1. 模型의 概要

DDG는 프로그램 slice 개념에 있어서 어떤 變數에 影響을 주는 變數들을 모두 나타내고 있지만 control flow를 결정하는 文章(分岐文章)은 찾을 수가 없다. 다시 말해서

어떤 變數가 分岐文章의 影響권 내, 즉 influence내에 있다면 그 變數는 分岐文章에 從屬되어 있다고 볼 수 있다. 따라서 이와같이 control이 define되는 分岐文章은 Boolean값이 定義되는 것이므로, DDG내에 node로 표시하여 변형된 DDG(MDDG : modified DDG)를 生成하고, 이 MDDG를 利用하면 프로그램 slice를 찾아 낼 수 있다.

III.1의 예2)에 대한 MDDG는 아래 (그림 2)와 같다.



(그림2) MDDG

2. Graph Theory의 適用

앞 절에서 언급된 MDDG에 관한 이론을 graph 表現으로 정리하면 다음과 같다.

定義 : MDDG = ⟨N, E⟩, where

N is a set of node which are representing a variable definition or control definition

E is representing a data dependency

이 MDDG를 adjacency matrix, A로 表現하면, A의 構成要素인 a_{ij}는,

$$a_{ij} = \begin{cases} 1 & , \text{ if} \langle n_i, n_j \rangle \in E \\ 0 & , \text{ otherwise} \end{cases}$$

로 表現되고, 이 MDDG의 reachability matrix, R은,

$$R = A \cup_{k=1}^{n-1} A^k$$

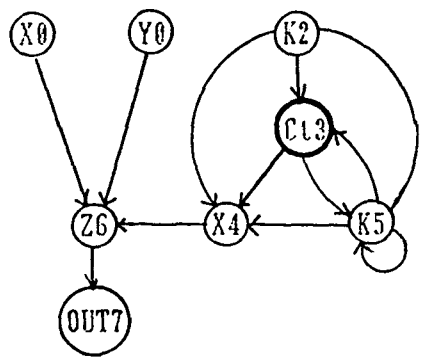
$$\text{, where } r_{ij} = \begin{cases} 1 & , \text{ if } a_{ij} = 1 \text{ or } a^k{}_{ij} = 1 \\ 0 & , \text{ otherwise} \end{cases}$$

$$A^{k-1}.A \quad , \ k = 2, 3, 4, \cdots$$

$$\text{slice matrix(SM)} = R^T$$

이 R을 transpose시킨 slice matrix ($SM = R^T$)에서 열의 값이 '1'로 나타나는 행의 變數들은 그 행에 影響을 미치는 變數들이므로, 그 열에 해당하는 變數와 direct dependency가 있거나 또는 control dependency가 있다는 것을 뜻한다. 따라서 SM을 利用하여 프로그램 slice를 구할 수 있다.

II.2의 예제.2를 MDDG로 表現하고 위의 이론에 따라 프로그램 slice를 구하면 (그림 3)과 같다.



(그림3) 예제.2에 대한 MDDG

(그림3)은 本 論文에서 提示한 control이 define되는 分岐文章을 Node로 추가하여 表現한 MDDG이다. 따라서 이 MDDG를 앞에서 설명한 graph 이론에 의해 다음과 같은 slice matrix, SM으로 表現할 수 있다.

slice matrix, SM은,

$$SM = \begin{array}{c} \\ X0 \\ Y0 \\ K2 \\ Ct3 \\ X4 \\ K5 \\ Z6 \\ OUT7 \end{array} \begin{array}{c} \begin{array}{cccccccc} X0 & Y0 & K2 & Ct3 & X4 & K5 & Z6 & OUT7 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

과 같다.

이 SM을 가지고 4번째 文章에서 X變數에 影響을 주는 變數들을 찾아보면, X4 열에서 '1'이 나타난 K2, Ct3, K5가 X4에 影響을 주는 것들임을 알 수 있다. 따라서 아래와 같은 프로그램 Slice를 얻을 수 있다.

프로그램 slice on criterion<X4>

slice : S = {K2, Ct3, K5}

여기에 프로그램 文法體系에 맞게 1, 8번째 文章을 包含하면 Weiser가 말한 프로그램 slice와 같은 프로그램 slice를 구하여 낼 수 있다. 즉, SM의 열에 '1'이 나타난다면 그 행에 연관된 變數가 그 열에 影響을 미치고 있다는 것을 의미한다. 다른 예를 든다면, 7번째 문장의 OUT 變數에 影響을 주고 있는 것은 OUT7 열이 모두 '1'이므로 프로그램 전체가 됨을 알 수 있다. 다시말해서 프로그램 자체가 프로그램 slice가 되는 것이다.

V. 模型의 檢證 및 評價

MDDG를 통하여 生成한 프로그램 slice는 Weiser의 algorithm에 의하여 구한 것과 근본적으로 같다. MDDG에 의한 프로그램 Slice는 i번째 文章에서 定義된 變數에 影響을 미치는 文章에서 定義된 變數들을 선택하는 것이므로, Weiser의 algorithm에 의하여 구한 프로그램 slice와 같게 된다.

따라서 MDDG에서의 criterion은, MC = $\langle V_{di}, [V_{ri}] \rangle$로, V_{di}는 i번째 文章에서 定義되는 變數를 의미하고, V_{ri}는 i번째 文章에서 參照되는 變數를 의미하는 것으로 필요시 사용한다.

Weiser의 algorithm은 지정된 文章에서 주어진 變數에 대한 프로그램 slice 하나만을 生成해 내지만, MDDG를 利用한 본 模型에서는 한번의 作業으로 프로그램내의 모든 프로그램 slice를 生成해 낼 수 있다는 特性이 Weiser의 方法과 비교하여 그 장점이 될 수

있다.

VI. 結 論

지금까지 本 論文에서는 프로그램 slice를 生成하는데 있어서 旣存의 方法이었던 Weiser의 algorithm에 의한 方法과는 달리, DDG를 변형한 MDDG로부터 matrix 연산을 통해 프로그램 slice를 찾는 方法을 提示하였다.

Weiser의 algorithm에 의한 方法은 한번에 하나의 프로그램 slice를 구하여 내지만, MDDG를 利用한 본 模型에서는 한번에 프로그램내의 모든 프로그램 slice를 生成하므로서 소프트웨어 개발 중 디버깅시 하나의 變數에 대한 디버깅도 가능하며, 變數의 상호 연계에 의한 또 다른 變數에 대한 디버깅까지도 동시에 가능하다.

Weiser가 提示한 프로그램 slice에 관한 Metric[1] 表現 또한 本 模型에서 가능하므로 이를 통해 프로그램 복잡도에 관한 연구도 가능하다.

本 論文에서는 structured 프로그램에서 module 내부흐름만을 취급하였다. DDG의 特性상 unstructured 프로그램 및 module간에서도 表現이 가능함으로 추후 연구되어야 할 것이다.

參考文獻

1. Mark Weiser, "Program Slicing" IEEE Trans. S.E. 1981. pp.439-449.
2. Mark Weiser, "Programmers use Slices When Debugging" ACM Communication Vol.25. July 1982. pp. 446-452.
3. Mark Weiser, "Program Slicing" IEEE Trans. S.E. Vol. SE-10 NO-4. July 1984. pp.352-357.
4. Hareton K.N.Leung & Hassan K. Reghbati, "Comments on Program Slicing" IEEE Trans. S.E. Vol. SE-13 NO-12 DEC. 1987. pp.1370-1371.
5. James M.Bieman, "Measuring Software Data Dependency Complexity" 1984. ph.D. Thesis. Univ of Louisiana.
6. James M.Bieman & William R. Edwards, "Experimental Evaluation of the Data Dependency Graph for Use in Measuring Software Clarity" Proceedings of the Eighteenth Annual Hawaii International Conference on System Science 1985, pp.171-276.
7. J.P.Tremblay & R.Manohar, "Discrete Mathematical Structures with Applications to Computer Science" pp. 168-493.
8. Gould, J.D. "Some psychological evidence on how people debug computer programs" International J. of Man-Machine Studies 7.1. jan 1975, pp.151-182.
9. Hecht, Matthew S., "Flow Analysis of Computer Programs" Elsevier North-Holland 1977, pp.13-22.