프로그램 슬라이싱과 나씨-슈나이더만 차트를 이용한 구조적 프로그램의 이해를 위한 도구 개발

박승득, 박만곤
부경대학교 전자계산학과

요 약

본 논문은 원시 프로그램에 오류가 생겼을 때 신속한 오류검출, 소프트웨어의 유지보수, 유연한 테스팅 등을 목적으로 하는 프로그램 슬라이싱과 원시 프로그램의 복잡한 제어 흐름이나 데이터 흐름을 이해하고 프로그램의 테스팅 속도 향상과 오류위치를 파악하는 데 도움을 주는 구조적 순서도의 개념을 도입하여, C 언어에 대한 할당문(Assignment Statement)과 복합 제어 문(Compound Control Statement)으로 된 원시 프로그램의 입력에 대해 프로시듀어(Procedure) 내에서의 구조적 순서도인 나씨-슈나이더만 차트(Nassi-Shneiderman Chart)를 자동 생성하고 실제 관심있는 변수에 대해 정적 슬라이싱(Static Slicing)을 하는 도구를 개발함으로써 보다 빠르고 정확한 프로그램 구조의 이해에 도움을 주고자 한다.

Ⅰ. 서 론

프로그램 슬라이싱(Program Slicing)은 관심있는 변수들의 값에 직접·간접적으로 영향을 주는 모든 문장들을 찾는 방법이다. 이 방법은 프로그램의 데이터 흐름과 제어 흐름을 분석함으로써 구현되어질 수 있는 데, 이에 대한 개념은 Weiser(1979, 1984)에 의해 처음 제안되었다. 여기서 제안된 프로그램 슬라이싱을 하기 위한 슬라이싱 기준(*Criterion*)은 $<p, V>$이다. p는 원시 프로그램 내의 문장(Statement)의 위치이고 V는 변수(variable)들의 집합이다. 슬라이싱 기준 $<p, V>$로 슬라이싱해서 나온 결과를 프로그램 슬라이스(Program Slice)라 한다. 이는 위치 p에서 V에 속한 변수들의 값에 영향을 주는 원시 프로그램의 모든 문장들로 구성되어지며, 슬라이싱하기 전의 원시 프로그램보다 크기가 작고 실행 가능한 원시 프로그램이 된다. 따라서 프로그램 슬라이싱은 디버깅, 테스팅, 유지보수, 그리고 프로그램의 이해에 유용하다.

이와 비슷하게, 구조적 순서도는 원시 프로그램의 구조를 도식적으로 표현함으로써 원시 프로그램의 가독성을 높이고 복잡한 데이터 흐름이나 제어 흐름을 이해하며 프로그램의 테스팅 속도 향상과 오류위치를 파악하는 디버깅에 도움을 준다(Nassi and Shneiderman, 1973; Woodward, 1986).

프로그램 슬라이싱에 있어서 대표적인 슬라이싱 도구로는 UNRAVEL(Lyle and Wallace 등, 1997), SPYDER(Agrawal 등, 1990) 등이 있으며, 구조적 순서도에는 DiaGen(Minas 등, 1995), Garnet(Myers 등, 1990) 등과 같은 다이어그램 에디터를 위한 생성기가 있으나 프로그램 슬라이싱과 구조적 순서도를 동시에 고려한 도구는 아직 없으며, 본 논문의 구조적 슬라이싱 도구(Structured Slicing Tool)는 C 언어에 대한 할당문(Assignment Statement)과 복합 제어 문(Compound Control Statement)으로 된 원시 프로그램에 대해 구조적 순서도인 나씨-슈나이더만 차트를 자동으로 생성하고, 관심 있는 변수에 대해 정적 슬라이싱 알고리즘을 적용하여 프로시듀어(Procedure) 내에서 나씨-슈나이더만 차트와 슬라이싱 결과를 보여 주는 자바 언어(Java Language)로 구현한 윈도우 응용 프로그램이다.

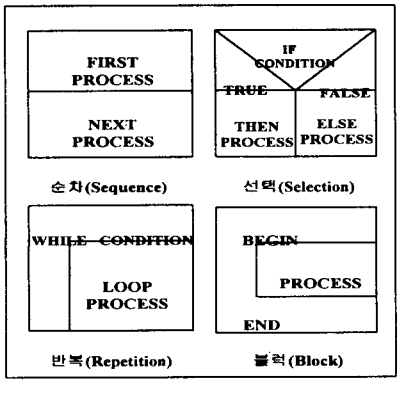
본 논문의 구성은 먼저 구조적 순서도인 나씨-슈나이더만 차트(Nassi and Shneiderman, 1973)에 대한 기본 개념을 2장에서 설명하고, 프로그램 슬라이싱에 대한 기본 개념을 3장에서 설명한다. 4장에서는 C 언어에 대한 할당문(Assignment Statement)과 복합 제어 문(Compound Control Statement)으로 된 원시

프로그램의 입력에 대해 나씨-슈나이더만 차트를 생성하는 구조적 슬라이싱 도구(Structured Slicing Tool)의 구현 알고리즘과 구현된 예를 설명한다. 마지막으로 5장에서는 구조적 슬라이싱 도구에 대한 유용성과 확장성을 언급하고 결론을 맺는다.

Ⅱ. 구조적 순서도

구조적 순서도는 원시 프로그램을 기능이나 구조에 따라서 여러 개의 모듈로 분할하고 이 모듈의 흐름을 파악하여 순서도로 표현한다. 처리 방법은 먼저 프로그램 전체에 대한 각 모듈의 처리순서나 관계 등을 표현하고, 각각의 모듈의 구조와 처리흐름을 순서에 맞게 나타낸다.

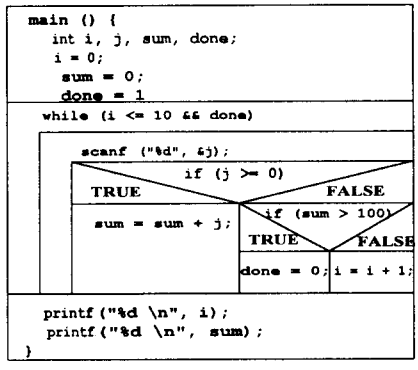
본 논문에서 사용된 구조적 순서도는 나씨-슈나이더만 차트이며 논리의 기술에 중점을 두고 도형으로 표현하는 방법이다. 일반적으로 원시 프로그램은 구조적 기법을 이용하여 코딩(Coding)이 되어진다. 따라서 본 논문에선 네 가지 제어 논리 구조(Woddward, 1986)를 이용하여 원시 프로그램을 순서도로 표현한다. 각각의 제어 논리 구조는 순차, 선택, 반복, 블록구조 등이며 <그림 1>과 같이 박스(Box)의 형태로 나타난다. 이러한 구조적 순서도를 사용하여 표현하면 원시 프로그램을 쉽게 파악할 수 있으며 이 구조적 순서도를 참조하여 프로그램을 코딩(Coding)하면 시간과 노력을 훨씬 줄이고 생산성을 향상시킬 수 있다. 그리고 구조적 순서도는 교육용, 유지보수용 문서로 활용할 수 있는 장점이 있다. <그림 3>은 <그림 2>의 C 언어로 된 원시 프로그램을 이용하여 나씨-슈나이더만 차트를 표현한 것이다.



<그림 1> 나씨-슈나이더만 차트의 기본구조

```
main () {
    int i, j, sum, done;
    i = 0;
    sum = 0;
    done = 1;
    while (i <= 10 && done) {
        scanf ("%d", &j);
        if (j >= 0)
            sum = sum + j;
        if (sum > 100)
            done = 0;
        else
            i = i + 1;
    }
    printf("%d \n", i);
    printf("%d \n", sum);
}
```

<그림 2> C 언어로 된 원시 프로그램 예제



<그림 3> C 언어로 된 원시 프로그램의 나씨-슈나이더만 차트

2.1 구조적 순서도 표현 방법

구조적 순서도의 표현방법에는 기본적인 순차, 선택, 반복, 블록구조가 있고, 선택구조의 내부에 선택구조가 존재하는 다중선택구조가 있다. C 언어에서 박스(Box)를 구성하는 명령문은 아래의 <표 1>과 같다.

<표 1> C 언어에서 박스(Box)를 구성하는 명령어

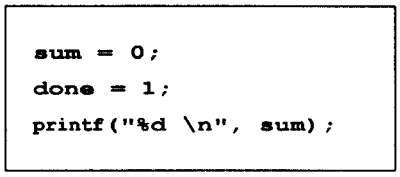
종 류	명 령 문
블럭구조	프로그램이나 함수의 시작
순차구조	선언문, 할당문 등
선택구조	if-else 문, case 문 등
반복구조	while 문, for 문 등

2.1.1 순차구조의 표현방법

원시 프로그램을 읽어서 분기명령문이나 반복명령문 이외의 명령문이면 이는 순차로 처리되는 명령문이므로 박스를 표시하고 읽은 명령문을 박스내에 삽입한다. 만일 이전의 명령문이 순차로 처리되는 명령문이었으면 이미

박스가 그려졌으므로 박스를 그리는 대신 기존의 박스를 알맞은 크기로 확장하여 박스에 추가한다. <그림 3>는 순차구조를 표현한 예이다.

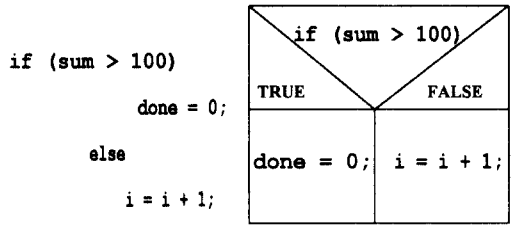
```
sum = 0;
done = 1;
printf("%d \n", sum);
```



<그림 4> 순차구조의 표현

2.1.2 선택구조의 표현방법

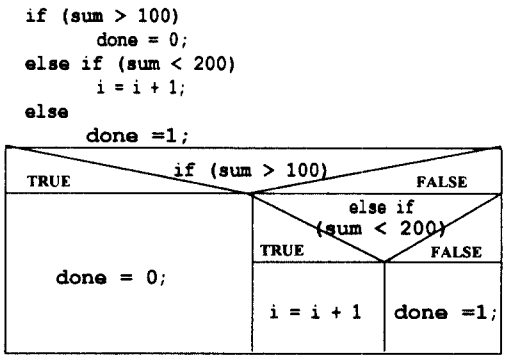
프로그램 내에 명령문이 if와 같은 선택구조의 명령문이면 이에 따른 분기처리를 표현한다. 먼저 분기하기 위한 조건을 나타내고 그 아래에 박스를 두 개로 나누어서 나타낸다. 좌측에는 조건의 값이 참일 경우의 처리를 나타내고 우측에는 조건의 값이 거짓일 경우에 대한 처리를 표시하도록 한다. 이때 조건문장이나 처리문장의 크기를 계산하여 박스를 알맞게 나타내도록 한다. <그림 5>는 프로그램을 선택구조로 표현한 예이다.



<그림 5> 선택구조의 표현

2.1.3 다중선택구조의 표현방법

다중 선택구조는 선택구조의 내부에 else if 문과 같은 여러 개의 선택구조가 중첩되어 있는 형태이다. 이때 여러 개의 조건문을 순서도로 표현하기 위해서는 단순한 선택구조를 표현할 때보다는 더욱 복잡해진다. 그러나 선택구조가 조건에 의하여 분기될 때는 반드시 반으로 나누어져서 처리되므로 하부의 선택구조에서도 마찬가지로 주어진 박스의 구조를 이분하여 처리하도록 한다. 다음 <그림 6>의 예는 다중선택구조를 표현한 것이다.

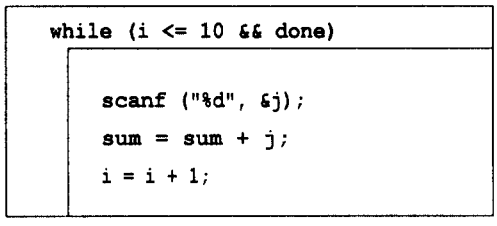


<그림 6> 다중 선택구조의 표현

2.1.4 반복구조의 표현방법

이 구조는 while 문의 형태로서 조건을 먼저 검사하고 나서 해당 처리부분을 수행하는 형태로서 먼저 반복구조를 나타낼 박스를 표시하고 상단에 반복 수행할 조건을 나타낸다. 그 다음으로는 박스 내부의 처리를 나타내는 박스를 우측 하단에 맞추어서 나타내고 이 안에 처리 명령어를 추가하는 방법으로 표현한다. 다음 <그림 7>의 예는 while 문의 예이다.

```
while (i <= 10 && done) {
    scanf ("%d", &j);
    sum = sum + j;
    i = i + 1;
}
```



<그림 7> 반복구조의 표현

Ⅲ. 프로그램 슬라이싱

프로그램 슬라이싱(Program Slicing)에는 주어진 변수의 값에 영향을 주는 모든 문장들의 집합을 찾는 정적 슬라이싱(Static Slicing)과 주어진 프로그램 입력에 대해 발생된 변수의 값에 실제적으로 영향을 주는 모든 문장들의 집합을 찾는 동적 슬라이싱(Dynamic Slicing)으로 크게 나눌 수 있다.

3.1 정적 슬라이싱(Static Slicing)

Weiser(1979, 1984)에 의해 제안된 정적 슬라이싱은 원시 프로그램 내의 문장(statement)의 위치인 p와, 변수(variable)들의 집합 V로 이루어진 슬라이싱 기준 <p, V>로 슬라이싱하는 것을 말한다. 즉, 프로그램 내의 위치 p에서 변수 V에 속한 변수들의 값에 영향을 주는 모든 원시 프로그램 내의 문장들을 찾아내는 것을 말한다. 다음은 정적 슬라이싱에 쓰이는 몇 가지 정의이다.

- **DEF**(n): 문장 n에서 정의된 값을 가진 변수들의 집합.
- **REF**(n): 문장 n에서 사용된 값을 가진 변수들의 집합.

예를 들어, <그림 8>의 왼쪽과 같은 C 언어로 된 원시 프로그램에 대해 슬라이싱 기준 <14, *output1*>이 주어진다면, 정적 슬라이싱을 하는 과정을 살펴보면, 우선 프로그램의 실행 순서에 따라 라인 14번의 선임자(Predecessor)를 찾는다. 여기서 라인 14번의 선임자는 라인 13번이 되며, 라인 13번에서 *output1*이 정의되었다면, 라인 13번을 슬라이스에 포함을 시키고 라인 13번에서 사용된 변수(z)를 포함한 슬라이싱 기준을 새롭게 재구성한다. 즉, 라인 13번에서 *output1*이 정의되었으므로 사용된 변수 z를 포함한 슬라이싱 기준은 <13, z>로 재구성된다. 여기서 새롭게 재구성된 슬라이싱 기준 <13, z>에 대해서 라인 13번의 선임자는 라인 12번이 되며, 라인 12번에서 변수 z가 정의되었으므로 라인 12번을 슬라이스에 포함시키고 라인 12번에서 사용된 변수(할당문의 오른 쪽 변수 z)에 대한 슬라이싱 기준은 <12, z>로 된다. 여기서 라인 11번은 슬라이싱 기준 <12, z>에 대해서 변수 z가 정의되지 않았으므로 슬라이스에 포함을 시키지 않고 슬라이싱 기준은 <11, z>로 바뀌게 된다. 이어서 라인 11번의 선임자는 라인 10번이고, 변수 z가 정의되었으므로 라인 10번을 슬라이스에 포함시키고 라인 10번에서 사용된 변수 {y, b}에 대한 슬라이싱 기준을 각각 <10, y>, <10, b>로 재구성하고, 프로그램의 실행순서에 따라 선임자를 찾아서 프로그램의 처음 라인이 될 때까지 계속 반복 계산을 한다.

<그림 8>은 C 언어로 된 원시 프로그램에 대해 슬라이싱 기준 <14, *output1*>에 대한 슬라이싱의 결과를 보여준다.

원시 프로그램	슬라이싱된 프로그램
1	1
2 int input1, input2;	2
3 int a, b y, z;	3
4 int output1, output2;	4
5 scanf("%d", &input1);	5 scanf("%d", &input1);
6 a = input1;	6 a = input1;
7 scanf("%d", &input2);	7 scanf("%d", &input2);
8 b =input2;	8 b =input2;
9 y = a + 1;	9 y = a + 1;
10 z =y +b;	10 z =y +b;
11 output2 = z + 1;	11
12 z = z * z;	12 z = z * z;
13 output1 = z;	13 output1 = z;
14 printf("%d\n", output1);	14
15 printf("%d\n", output2);	15

<그림 8> 슬라이싱 기준 <14, *output1*>에 대한 정적 슬라이싱 예제와 결과

3.2 동적 슬라이싱(Dynamic Slicing)

정적슬라이싱이 어떤 변수에 대한 모든 문장들에 관심을 가진다면 Korel과 Laski에 의해서 소개된 동적 슬라이싱(Korel and Laski, 1988)은 디버깅 실행에서 어떤 특별한 잘못된 실행을 처리하고 그 결과로 나타나는 오류의 원인이 되는 위치에 관심을 가지며, 입력의 집합보다는 오히려 어떤 한 개의 주어진 프로그램 입력에 대하여 해당 프로그램의 행위를 보존하는 한 개의 슬라이스를 찾는 것을 말한다. 즉, 어떤 슬라이싱 기준에 대해 실제 입력 값을 대입해서 실행된 원시 프로그램의 문장만을 고려해서 슬라이싱하는 방법을 동적 슬라이싱이라 한다.

동적 슬라이싱의 계산은 현재 대부분 후향분석(Backward Analysis)에 근거를 두고 있다. 즉, 실제 입력 값에 대한 프로그램의 실행순서가 처음 기록되어진 후 동적 슬라이스 알고리즘이 자료·제어 종속을 나타내는 실행순서의 반대 순서로 분석될 때 동적 슬라이스가 산출된다. 비교를 위해 <그림 9>는 슬라이싱 기준 <11. *product*>에 대한 정적 슬라이싱 결과이다.

<그림 10>의 C 언어로 된 예제 프로그램에서 슬라이싱 기준 <11, *product*>에 대해 동적 슬라이싱을 하기 위해 변수 n = -1 일 때 시험 사례(Test Case)를 살펴보자. 이때 실행순서는 라인 1, 2 , 3, 4, 10, 11의 순서로 프로그램이 실행된다.

이제 슬라이싱 기준 <11, *product*>에 대해 동적 슬라이싱을 해보면, 라인 11번의 선임자는 라인 10번이고 변수 *product*가 정의되지 않았으므로 새로운 슬라이싱 기준 <10, *product*>가 재구성되고 라인 10번을 거쳐 라인 4

번에서 변수 *product*가 정의되었으므로 라인 4번을 슬라이스에 포함하고 라인 1번까지 슬라이싱을 수행한다. 따라서 결과는 <그림 10>과 같고 확실히 <그림 9>와 같은 정적 슬라이싱보다 버그의 범위를 국한시키는 데 도움을 준다.

원시 프로그램	슬라이싱된 프로그램
1 scanf("%d", &n); 2 i = 0; 3 sum = 0; 4 product = 1; 5 while (i <= n){ 6 sum = sum + i; 7 product =product * i; 8 i = i + 1; 9 } 10 printf("%d", sum); 11 printf("%d", product);	1 scanf("%d", &n); 2 i = 0; 3 4 product = 1; 5 while (i <= n){ 6 7 product =product * i; 8 i = i + 1; 9 } 10 11 printf("%d", product);

<그림 9> 슬라이싱 기준 <11, *product* >에 대한 정적 슬라이싱

원시 프로그램	슬라이싱된 프로그램
1 scanf("%d", &n); 2 i = 0; 3 sum = 0; 4 product = 1; 5 while (i <= n){ 6 sum = sum + i; 7 product =product * i; 8 i = i + 1; 9 } 10 printf("%d", sum); 11 printf("%d", product);	1 2 3 4 product = 1; 5 6 7 8 9 10 11

<그림 10> 슬라이싱 기준 <11, *product*>이고 n = -1일 때의 동적 슬라이싱

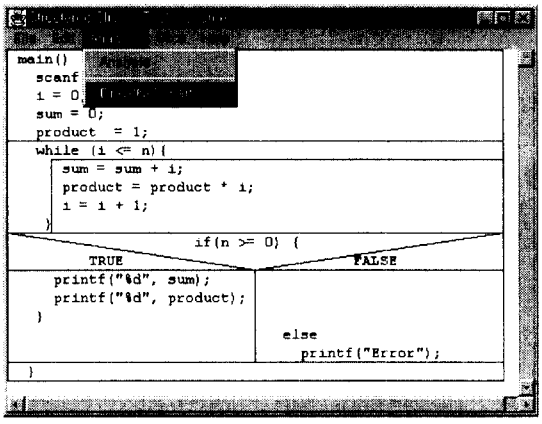
Ⅳ. 구조적 슬라이싱 도구 (Structured Slicing Tool)의 구현

2장에서 설명된 구조적 순서도인 나씨-슈나이더만 차트와 3장에서 설명된 프로그램 슬라이싱의 기본 개념을 이용한 구조적 슬라이싱 도구(Structured Slicing Tool)는 C 언어에 대한 할당문과 복합 제어문으로 된 원시 프로그램의 입력에 대해 자바 언어로 구현한 윈도우 응용 프로그램이다. 다음 <표 2>는 구조적 슬라이싱 도구의 메뉴에 대한 설명이다.

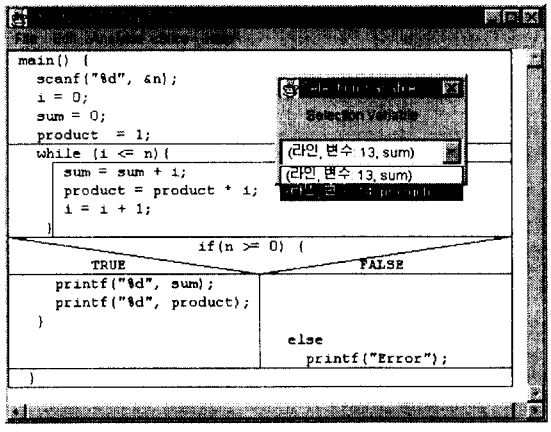
<그림 11>은 File 메뉴의 Open 부메뉴를 선택하여 C 언어로 된 원시 프로그램을 화면에 나타내고 Analysis 메뉴의 Draw(N-S chart) 부메뉴를 선택하여 화면에 뿌려진 원시 프로그램에 대해서 나씨-슈나이더만 차트를 생성한 화면이다. <그림 12>는 Slice 메뉴의 Selection Variable 부메뉴를 선택해서 다이얼로그 창을 띄워 관심있는 변수를 선택하고 선택한 변수에 대해 슬라이싱된 결과는 굵은 글씨체로 보여주는 것이 <그림 13>이다.

<표 2> 메뉴의 종류와 기능

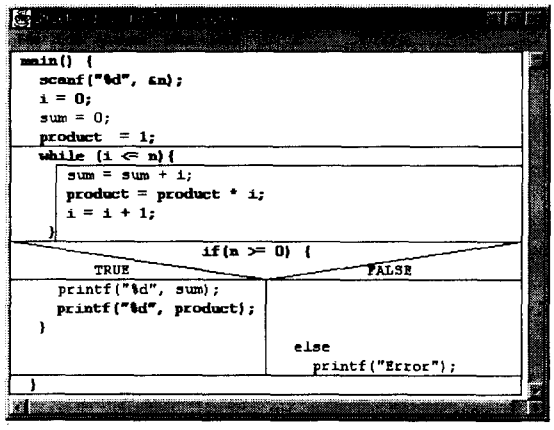
종	류	기 능
메뉴	File	New, Open, Save, Save as, Print, Quit의 부메뉴를 가지고 있으며, Open 부메뉴를 통해 원시 프로그램을 선택한다.
	Edit	Undo, Redo, Copy, Cut, Paste, Find, Replace의 부메뉴를 가진 편집 메뉴이다.
	Ana-lysis	원시 프로그램의 파싱 분석 기능을 가진 Analysis, 원시 프로그램의 나씨-슈나이더만 차트를 그려주는 Draw(N-S chart)의 부메뉴를 가진다.
	Slice	슬라이싱할 관심있는변수를 선택하는 Selection Variable의 뷰메뉴를 가진다.
	Help	프로그램에 대한 About의 부메뉴를 가진다.



<그림 11> 나씨-슈나이더만 차트의 생성



<그림 12> 다이얼로그 창에서 변수 선택



<그림 13> 슬라이싱 기준 <14, *product*>에 대한 정적 슬라이싱의 결과

4.1 나씨-슈나이더만 차트의 구현

나씨-슈나이더만 차트를 구현하기 위해 원시 프로그램의 구문을 분석하여야 한다. 구조적 슬라이싱 도구는 이 과정을 Analysis 메뉴를 통해서 분석하는 데 이 분석과정에서 사용하는 것은 Java Compiler Compiler™ (JavaCC)인데 JavaCC는 자바 응용 프로그램과 함께 사용하기 위한 파서 생성기이다.

JavaCC 파서 생성기에서는 순차구조, if 문의 선택구조, while 문의 반복구조를 분석하여 스트링 값으로 저장하고 JavaCC 파서 생성기로부터 받은 프로그램의 시작과 끝, while 문의 시작과 끝, if 문의 시작과 끝 라인이 기록된 스트링 값은 Analysis 메뉴의 Draw(N-S chart)라는 부메뉴가 선택되어지면 스트링 값의 키워드를 분석해서 각각의 구조에 대한 알고리즘(장경식 and 박만곤, 1996)을 통해서 처리를 하게 된다. <그림 14>은 Analysis 메뉴의 Analysis 부메뉴를 선택할 때 실행되는 자바 소스 코드이다.

4.2 프로그램 슬라이싱의 구현

구조적 슬라이싱 도구(Structured Slicing Tool)를 구현하기 위해서 C 언어에 대한 할당문(Assignment Statement)과 복합 제어 문(Compound Control Statement)으로 된 원시 프로그램 P와 원시 프로그램 내에서의 위치(Location) L, 즉 라인이고 V는 프로그램 내의 변수(Variable)인 슬라이싱 기준이 $<L, V>$로 주어지면, 라인의 위치 L과 L에 대한 정의 변수(Definition Variable)와 참조 변수(Reference Variable)들에 대한 분석은 4.1절에서 설

```
public void actionPerformed(ActionEvent evt) {
 String cmd = evt.getActionCommand();
 if (cmd == null)
     return;
 if (cmd.equals(analysisString)) {
  if(!textCanvas.openready){
    AnalysisDialog ad=new AnalysisDialog(textEdit);
    ab.show();
  }
  ps=new Parser(textEdit.filename); //파일 네임 입력
  ps.count = 0;  // Parser class의 카운터 초기화
  //파서의 라인을 캔버스의 스트링에 대입
  textCanvas.pred = ps.Line;
  ps.Line =""; // 라인 초기화
  textCanvas.analysis = true;
 }
 else if (cmd.equals(drawString)){
  if(!textCanvas.openready || !textCanvas.analysis){
   DrawDialog ab = new DrawDialog(textEdit);
   ab.show();
   }
 else if(textCanvas.openready
           && textCanvas.analysis){
   textCanvas.drawChart = true;
   textCanvas.update();
 }
 else return;
 }
}
```

<그림 14> Analysis 메뉴의 Analysis 부메뉴를 구현한 자바 소스 코드

명한 JavaCC 파서 생성기로부터 얻으며, 라인의 위치 L과 변수 집합 V는 Weiser(1979, 1984)의 개념을 재정의하여 사용하였다(Lyle and Wallace, 1997). 다음은 프로그램 슬라이스들을 구성하는 데 쓰이는 정의이다.

- ***Defs(n)***: 문장 n에서 정의(할당)된 변수들의 집합.
- ***Refs(n)***: 문장 n에서 참조되어진 변수들의 집합.
- ***Req(n)***: 문장 n과 함께 슬라이스(slice) 내에서 포함되어진 문장들의 집합.

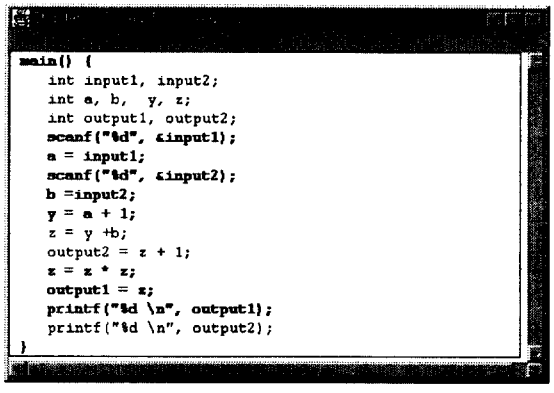
우선, 문장 n을 문장 m의 선임자라면, *def(n)*과 슬라이싱 기준은 다음 <그림 15>와 같은 식으로 슬라이스 S를 정의한다.

$$S_{<m,v>} = \begin{cases} S_{<n,v>} & \text{if} \notin \text{def}\,s(n) \\ \{n\} \bigcup S_{<n,x>} \forall x \in refs(n) & otherwise \end{cases}$$

<그림 15> 슬라이스 S의 정의

<그림 16>는 <그림 15>의 슬라이스 정의를 적용해서 <그림 8>의 왼쪽 원시 프로그램

을 슬라이싱 기준 <14, *output1*>에 대해 구조적 슬라이싱 도구를 이용해서 슬라이싱한 결과를 보여주고 <그림 16>의 라인 5-9번, 라인 12-14번은 프로시듀어(Procedure) 내의 중괄호로 둘러싸기 위해 *req(n)*의 집합인 라인 1번과 16번을 필요로 하는 것을 보여준다.



```
main() {
   int input1, input2;
   int a, b,  y, z;
   int output1, output2;
   scanf("%d", &input1);
   a = input1;
   scanf("%d", &input2);
   b =input2;
   y = a + 1;
   z = y +b;
   output2 = z + 1;
   z = z * z;
   output1 = z;
   printf("%d \n", output1);
   printf("%d \n", output2);
}
```

<그림 16> 슬라이싱 기준 <14, *output1*>에 대한 슬라이싱 결과

복합 제어문(Compound Control Statement)은 다른 문장의 실행을 직접적으로 제어하는 조건(Condition)을 가진 문장이다. 간단한 복합 문장(중괄호로 둘러싸인 문장)과 프로시듀어 문장으로 둘러싸인 중괄호는 조건이 없는 복합 제어 문장과 같이 취급한다. if 문, while 문, for 문과 같은 제어문장들은 제어문장에 의해 좌우되는 어떤 문장이 슬라이스에 포함될 때마다 슬라이싱 기준 <*n*, *refs(n)*>에 대한 슬라이스는 원래의 슬라이스에 추가되어진다. 다음의 <그림 17>은 복합 제어문에서 $v \in \mathrm{def}\,s(n)$이기 위한 슬라이싱 규칙이다.

$$S_{\langle m,\, v\rangle} = \{n\} \cup \left(\bigcup_{x \in refs(n)} S_{\langle n,\, x\rangle}\right) \cup \left(\bigcup_{y \in refs(k)} \bigcup_{k \in req(n)} S_{\langle k,\, y\rangle}\right)$$

<그림 17> $v \in \mathrm{def}\,s(n)$을 위한 슬라이싱 규칙

본 논문의 구조적 슬라이싱 도구(Structured Slicing Tool)에서 Slice 메뉴의 Selection Variable 부메뉴를 통해 <그림 16>와 <그림 17>의 정의와 규칙을 구현하였고, <그림 18>은 Slice 메뉴의 Selection Variable 부메뉴를 선택했을 때 실행되는 변수 선택 다이얼로그 창에 대한 자바 소스 코드이고 <그림 19>은 다이얼로그 창에서 변수를 선택했을 때 실행되는 자바 소스 코드이다.

```
public SelectionDialog(Frame parent) {
   super(parent,"Selection Variable",true);
   setLayout(new BorderLayout());
   Panel pp = new Panel();
   pp.setLayout(new FlowLayout());
   add("North", pp);
   Label label = new Label(" Selection Variable ");
   pp.add(label);
   Panel p = new Panel();
   p.setLayout(new FlowLayout());
   add("Center", p);
   // printf 다음의 변수를 선택하는 박스를 띄움
   selectionVariable = ps.selectionVariable;
   ll = ps.ll;
   count = ps.count;
   ps.count = 0; // Parser class의 카운터 초기화
   for( int i = 0; i < count; i++) {
     c.addItem(" (라인, 변수: " + ll[i] +", "
                    +selectionVariable[i] + ") ");
   }
   ps.Line =""; // 라인 초기화
   c.addItemListener(this);
   p.add(c);
   setSize(170, 100);
   setLocation(300,200);
   addWindowListener(this);
   textE = new TextEdit();
   textC = textE.getCanvas();
}
```

<그림 18> Selection Variable 다이얼로그 창에 대한 자바 소스 코드

```
public void itemStateChanged(ItemEvent e) {
    // 슬라이스 라인을 -1로 초기화
     for(int s = 0; s< textC.sliceLine.length; s++)
        textC.sliceLine[s] = -1;
   for(int s = 0; s< textC.sliceLine.length; s++)
      //슬라이스 된 라인을 굵게 화면에 출력
       textC.sliceLine[s] = sL.sliceLine[s];
     textC.slice = true;
     textC.update();
     dispose();
}
```

<그림 19> Selection Variable 다이얼로그 창에서 변수를 선택했을 때의 자바 소스 코드

V. 결 론

구조적 순서도와 프로그램 슬라이싱은 디버깅, 테스팅, 프로그램의 이해에 매우 유용하며 구조적 순서도와 프로그램 슬라이싱을 통한 학습의 효과는 매우 크다.

이러한 구조적 순서도와 프로그램 슬라이싱의 유용성을 결합한 본 논문의 구조적 슬라이싱 도구(Structured Slicing Tool)는 구조적 순서도의 한 종류인 나씨-슈나이더만 차트를 자동적으로 생성해주며 현재 소프트웨어 개발

추세에 맞게 GUI(Graphic Use Interface) 개념을 적용하여 플랫폼 독립성의 언어적 특성을 가진 자바 언어로 개발한 윈도우 응용 프로그램이다. 이 도구를 이용하면 프로그램의 흐름이나 구조를 효과적이고 빠르게 파악함으로써 학습자들이나 개발자들에게 시간 절약과 소프트웨어 생산성을 향상시켜 줄 것이다.

앞으로의 과제는 C 언어에 대한 포인터(Pointer), 선언 구조(Declared Structure), 동적 구조(Dynamic Structure), 포인터에 의한 구조 변수의 참조와 할당, 그리고 구조적 환경에서 표현하기 힘든 프로시듀어 호출 간의 파라미터 패싱(Parameter Passing) 등의 구조를 표현할 수 있는 구조적 순서도의 작성과 동시에 정적, 동적 프로그램 슬라이싱을 할 수 있는 알고리즘의 개발과 적용이 필요하다.

참고문헌

[1] 장경식, 박만곤, "원시 프로그램의 이해를 위한 구조적 순서도 작성도구 개발(I)", 한국정보처리학회 '96추계 학술논문 발표집, 제 3권, 2호, 1996, pp. 442-449.

[2] B. Korel and J. Laski, "Dynamic Program Slicing", *Information Processing Letters*, Oct. 1988.

[3] B. A. Myers, D. A. Giuse, R. B. Dannenberg, et al., "Garnet-Comprehensive support for graphical, highly interactive user interfaces", *Computer*, Vol. 23, Nov. 1990, pp. 71-84.

[4] Chapin Ned, "New format for flowcharts", *Software-Practice and Experience*, Vol. 4, Oct. 1974, pp. 341-357.

[5] H. Agrawal, J. R. Horgan, "Dynamic Program Slicing", *Proc. ACM SIGPLAN'90 Conf. Programming Language Design and Implementation*, 1990, pp. 246-256.

[6] H. Agrawal, R. A. DeMillo, and E. H.Spafford, "Dynamic Slicing in the Presence of Unconstrained Pointers", *Proc. of the 4th ACM Symposium on Testing, Analysis and Verification*, Oct. 1991, pp. 60-73.

[7] http://www.sun.com/suntest/JavaCC, JavaCC(Java™ Compiler Compiler™), Java Parser Generator.

[8] J. Lyle and D. Wallace, " Using the Unravel Program Slicing Tool to Evaluate High Integrity", *Software Quality Week*, May 1997.

[9] K. Gallgher and J. Lyle, "Using Program Slicing in Software Maintenance", *IEEE Trans. Software Engineering*, Vol. 17, No. 8, Aug. 1991, pp. 751-761.

[10] M. Weiser, "Program Slicing: Formal, Psychological and Practical Investigations of an Automatic Program Abstraction Method", PhD thesis, The University of Michigan, Ann Arbor, Michigan, 1979.

[11] M. Weiser, "Progam Slicing" *IEEE Trans. Software Eng.*, Vol. 10, July 1984, pp. 352-357.

[12] M. Minas and G. Viehstaedt, "DiaGen: A generator for diagram editors providing direct manipulation and execution of diagrams", *IEEE Computer Society Press*, Sept. 1995.

[13] Nassi I. and Shneiderman B., "Technical contributions: Flowchart techniques for structured programming", *ACM Sigplan Notices*, Vol. 8, No. 8, Aug. 1973, pp. 12-16.

[14] R. Gupta, M. J. Harrold and M. L. Soffa, "An approach to regression testing using slicing", *In Proc. of the IEEE Conf. on Software Maintenance*, 1992, pp. 299-30.

[15] S. Horiwitz, T. Reps and D. Binkley, "Interprocedural Slicing Using Dependence Graphs", *ACM Trans. Programming Languages and Systems*, Vol. 12, No. 1, Jan. 1990, pp. 26-60.

[16] Woodward M. R., "The Use of Nassi-Schneiderman Chart and Supporting Tools in Software Engineering Education", *Compute. Educ.*, Vol. 11, Sept. 1986, pp. 267-279.