복잡한 자료 구조를 지니는 프로그램에 대한 슬라이싱

류호연* · 강원임 · 박재흥
경상대학교 전자계산학과

Program Slicing in the Presence of Complicated Data Structure

Ho-Yeon Ryu* · Won-Im Kang · Jae-Heung Park
Department of Computer Science, Gyeoungsang National University

요 약

프로그램 슬라이싱은 프로그램의 특정 문장에서 변수에 영향을 주는 문장들을 분해하는 방법으로, 디버깅, 최적화, 프로그램 유지 보수, 테스팅, 재사용 부품 추출 그리고 프로그램 이해를 포함하는 여러 응용 분야들에서 그 유용성을 확인할 수 있다. 본 논문은 포인터 변수와 포인터 변수에 의해 참조되는 동적 객체, 구조체, 배열을 포함하는 프로그램에 대한 정확한 슬라이스 생성에 관한 연구이다. 포인터 변수와 같은 복잡한 자료 구조를 가지는 프로그램에서 포인터 변수나 포인터 변수가 참조하는 객체의 상태를 파악하기 위해서 객체 참조 상태 그래프를 제시한다.주님

1. 서론

ANSI-C언어에서의 포인터나 배열과 같은 복잡한 자료 구조에 대한 정적 분석은 어렵다.[3,4] 포인터 변수가 참조하게 되는 객체1)에 대한 정확한 정보나 배열 첨자값이 변수로 나타날 경우 정확한 위치 정보, 그리고 동적 메모리 할당 시 생성되는 객체의 수는 실행 시간에 결정되기 때문이다.

실제 프로그램 상에서 포인터나 배열, 구조체와 같은 복잡한 자료 구조를 발견할 수 있다. 배열 요소나 구조체의 필드는 포인터의 간접 참조 연산을 이용하여 접근될 수 있으며, 대부분의 포인터 연산식은 복잡한 연산식을 포함하지 않는다 본 논문에서는 포인터 변수 연산식이나 배열 첨자값에 수식이 나타나는 경우를 허용한다.

본 논문은 포인터 변수와 포인터 변수에 의해 처리되는 동적 객체, 구조체 그리고 배열을 포함하는 프로그램에 대한 정확한 슬라이스 생성에 관한 연구이다 복잡한 자료 구조를 지닌 객체들간의 참조 상태를 표현하기 위하여 객체 참조 상태 그래프(ORSG;Object Reference State Graph)를 이용한다.

본 논문의 구성은 다음과 같다 2장에서는 포인터와 관련된 연구들을 분석하고 문제점을 제시하며, 3장에서는 본 논문에서 제안하는 ORSG를 정의한다. 4장에서는 이를 적용한 예를 보이고, 5장에서는 결론 및 향후 과제를 보인다.

2 연구 배경

프로그램 슬라이싱 속도를 향상시키거나 정확한 슬라이스 생성을 위한 연구는 많은 진행되고 있으나, 포인터와 같은 복잡한 자료 구조가 나타나는 프로그램에 대한 슬라이싱 연구는 드물다. 본 장에서는 복잡한 자료 구조 분석에 대한 기존의 연구들을 기술하고, 문제점을 제시한다

[Lyle]에 의해 제안된 포인터 상태 서브그래프(PSS; Pointer State Subgraph)는 포인터를 다루는 프로그램에서 포인터의 객체 참조 상태를 표현하기 위한 그래프이다 원래 프로그램 상에서 포인터 변수에 값을 할당하는 문장들만을 추출하여 포인터 변수에 대한 슬라이스를 구할 수 있다. PSS의 각 노드 상의 포인터 상태 함수(Pointer State Function)는 현재 노드에서의 포인터 변수를 객체로 사상시킨다. 그러나 PSS는 포인터 변수에 대한 상태 변화에만 관심이 있기 때문에 그 외의 배열이나 동적 객체들에 대한 상태 정보는 생성할 수는 없다[3].

[Livadas]는 동적 객체와 다차원 배열을 포함하는 다양한 자료 구조를 분석하기 위한 저장소 상태 그래프(SSG, Storage State Graph)를 제안하였다. ANSI-C 언어에서의 복잡한 객체들간의 자료 종속성 정보(data dependency information)를 계산할 수 있다. SSG는 복잡한 자료 구조 변수가 나타나는 각 문장 수행 후의 참조 상태를 표현한 것으로, 각 프로그램 지점 p에 대해 IN(p)와 OUT(p)의 두 개의 그래프를 포함한다 IN(p)는 p 실행 전에 존재하는 모든 메모리 패턴들의 집합을 나타내며, OUT(p)는 p 실행 후 발생하게 된 IN과의 변화를 나타낸다 [4]. 즉, SSG는 각 문장마다 두 개의 그래프의 쌍으로 이루어지는 반면에 ORSG는 프로그램 당 하나의 그래프만 생성된다

3. 객체 참조 상태 그래프(Object Reference State Graph)

본 장에서는 복잡한 자료 구조를 지닌 객체들간의 참조 상태를 표현할 수 있는 ORSG를 정의한다 ORSG는 복잡한 자료 구조를 지닌 객체에 대하여 기존의 방법들보다 더 정확한 정적 정보를 생성할 수 있으며, 나아가 더 정확한 슬라이스 생성 유용하다

3.1 ORSG

다음은 프로그램 P의 ORSG 정의이다

정의 3.1 (객체 참조 상태 그래프; ORSG) . 객체 참조 상태 그래프는 포인터 변수를 선언하는 문장(i.e., int **x)과 포인터 변수에 값을 할당하는 문장(i.e., x=&y, malloc 함수 호출), 그리고 배열 원소와 구조체 필드 접근 연산이 발생하는 문장을 분석함으로써 생성된다.

ORSG는 기본적으로 세 종류의 노드들을 지니며, 이들은 각각 세 종류의 부노드(subnode)로 분할되어질 수 있다 ORSG의 노드들은 [Livadas]가 제안한 노드 종류와 동일하다 ORSG는

1) 변수 선언에 의해 정적으로 할당되거나, malloc 함수 호출에 의해 동적으로 할당되는 메모리 영역을 객체(object)라 한다

변수(variable) 노드, 힙(heap) 노드, 가상 힙(virtual heap) 노드로 이루어진다 변수 노드는 이름 있는 메모리 구성 부분을 의미하며 선언된 변수들이다. 힙 노드는 프로그램 실행 시 동적으로 생성되어지는(malloc 함수 호출로 인한) 메모리 구성 부분이며, 힙 영역에 할당되는 객체의 자료형은 그 객체를 가리키는 포인터 변수의 자료형과 동일하다고 가정한다 가상 힙 노드는 배열의 구성 요소나 구조체의 필드처럼 정적으로 할당되지만 이름을 지니지 않는 메모리 구성 부분이다 세 가지 기본 노드들은 각각 스칼라, 포인터, 구조체 부노드들로 분할되어질 수 있으며, <표 1>은 ORSG에서 9가지 노드 종류를 나타낸다

Node Name	Examples
Scalar Variables	int x,
Point variable	float *x,
Structure Variable	struct node1 x,
Scalar Heap	x = (float *)(malloc(sizeof(float)),
Point Heap	x = (float **)(malloc(sizeof(float *)),
Structure Heap	x = (struct node1) (malloc(sizeof(struct node1))),
Scalar Virtual heap	struct node1 x, (field X.i is Virtual node) int x[10], (body array x is Virtual Node)
Point Virtual heap	struct node2 x, (field X.i is Virtual node) int *x[10], (body array x is Virtual Node)
Structure Virtual heap	struct node3 x, (field X.i is Virtual node) struct node3 x[10]; (body array x is Virtual Node)
struct node1 { int i, } struct node2 { int *i, }	struct node3 { struct node1 i, }

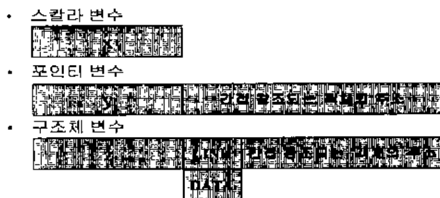
<표 1> The nine kinds of nodes in ORSG

ORSG에서의 노드 n1에서 n2 간의 간선은 n2 노드는 n1으로 표현된 포인터 변수나 힙 영역에 할당된 포인터 변수에 의해 참조되어질 수 있음을 의미한다. 모든 포인터 노드들은 자신과 관련된 간접 참조 레벨을 지니고 있다 이는 포인터 변수 선언문에서 쉽게 결정되어진다

3.2 ORSG 노드 표현 스키마

스칼라 변수에 대한 노드는 하나의 필드만을 지니며 자신의 이름이 레이블링 된다. 포인터 변수에 대한 노드는 두 개의 필드를 지닌다. 첫 번째 필드는 자신의 이름이 레이블링 되며, 두 번째 필드는 다른 노드로의 포인터 값이다. 구조체 변수에 대한 노드는 구조체 변수 이름이 레이블링 되는 하나의 필드를 나타내는 노드와 구조체의 각 필드를 표현하는 부노드로 구성된다 예를 들어,

```
int x;
int *y;
struct node {
        int *link,
        int data;
} z;
```



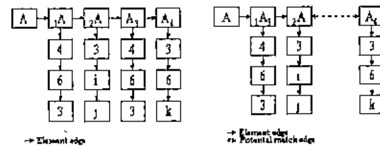
<그림 1> ORSG 노드 스키마의 예

에 대한 ORSG 노드 표현 스키마는 <그림1>과 같이 나타낼 수 있다.

동적 메모리 할당 함수에 의해 힙 영역에 위치한 객체의 스칼라, 포인터, 구조체 힙 노드들도 변수 노드들의 경우와 유사하다. 단, 이름이 레이블링 되는 필드는 자신이 할당된 프로그램 상의 문번호가 레이블링 된다

가상 힙 노드인 배열 원소나 구조체 필드들의 접근 문장에서 배열 원소를 배열 객체(array object)라 하고, 구조체 필드들은 구조체 객체(struct object)라고 한다 배열 객체들은 차수가 하나의 노드로 표현되는 링크드 리스트 구조로 표현된다 각 배열 객체의 첫 번째 노드는 배열 객체의 식별자가 레이블링 되고, 두 번째 노드부터 배열 객체의 차수가 하나의 노드로 표현되는 링크드 리스트이다. <그림2>는 3차원 배열을 포함하는 프로그램과 대응 ORSG이다.

```
        int x, y, z;
        int A[10][10][10],
        int i, j, k;
        sub() {
①              A[4][6][3]=x;
②              A[3][i][j] =x;
③              b=A[4][6][3],
④              c=A[3][6][4], }
               <a>
```



<그림 2> 다차원 배열과 ORSG

다음은 두 배열 객체 간의 정확한 일치(exact match)와 잠재적 일치(potential match)를 식별하는 방법이다

두 배열 객체 ai와 aj는 다음의 조건을 만족하면 정확히 일치(exact match)한다고 한다

① 어떤 객체도 수식을 포함해서는 안 된다

② $N_{i,k}$ = $N_{j,k}$, for 1≤k≤len(ai), $N_{i,k}$는 배열 객체 ai의 k번째 노드의 레이블이다 len(ai)는 배열 객체의 첫 번째 노드를 제외한 차수 노드의 길이이다

두 배열 객체 ai와 aj는 다음의 조건을 만족하면 잠재적으로 일치(potential match)한다고 한다

for 1≤k≤len(ai),

① $N_{i,k}$나 $N_{j,k}$ 둘 중 하나는 수식이거나,

② $N_{i,k}= N_{j,k}$

배열 객체 노드들이 정확히 일치 관계를 지니고 있다면, 두 노드는 합병되어질 수 있다 잠재적 일치 관계를 지니고 있다면, ORSG상의 배열 객체간에 잠재적 에지(potential edge)를 연결한다 <그림2-b>에 대하여 합병되어진 배열 객체의 ORSG는 <그림2-c>와 같다. 구조체 객체도 배열 객체와 유사하게 표현되어진다 단, 구조체 객체의 노드들은 상수값을 지니지 않는다

3.3 OSF

프로그램 상의 포인터 변수 할당문과 포인터 간접 참조 연산자가 나타나는 문장에 대해, 각 포인터 변수들은 객체 상태 함수(OSF; Object State Function)를 지닌다 이 함수는 포인터 변수를 객체로 사상시킨다

정의 3.2 (객체 상태 함수, OSF) : 문장 n에 나타나는 포인터 변수 v에 대하여, OSF P(n, v)는 v가 참조할 수 있는 객체의 집합이다.

$P_1(n, v)$의 경우, *v의 형태로 v가 1-레벨 참조할 수 있는 객체의 집합을 의미한다 P(n, v)는 다음과 같이 단계적으로 분해되어진다.

$P_0(n, v) = \{ v \}$

$P_1(n, v) = \{ \text{object } 0 \mid v \text{ holds the address of } o \}$

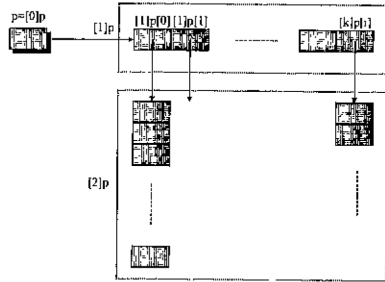
$P_k(n, v) = \{ x \mid x \in P_1(n, y) \wedge y \in P_{k-1}(n, v) \}$

$P_k(k>1)$의 값은 P_1의 값으로부터 유도되어 진다

3.4 가상 변수(Dummy Variable)

[Jingyue]는 포인터의 포인터 변수에 의해 접근 가능한 메모리 영역을 다루기 위해 각 포인터 변수에 대한 가상 변수를 사용하였다[2] 예를 들어, ANSI-C에서 **p 형식의 포인터 변수 p에 대한 가상 변수는 (1)p와 (2)p이다 포인터 변수 앞에 붙은 상수 값은 포인터 변수 p의 간접 참조 레벨 수를 의미한다 이때 p=(0)p이다

본 논문에서는 위의 방식과는 조금 다른 포인터 변수 연산식에 대한 가상 변수 표현법을 사용한다 다음 그림은 이 가상 변수를 나타낸 것이다



<그림 3> Dummy Variable의 예

k-레벨 간접 참조하는 포인터 변수 p에 대하여, [k]p[i]로 나타낸다(k≥0,i는 0 혹은 변수명 집합). k값을 좌첨자라 하고, I값을 우첨자라 한다 k는 간접 참조 레벨수를 나타내기 위한 음이 아닌 정수값이다 i는 포인터 변수 연산식에 나타난 변수들을 원소로 지니는 집합이거나, 연산식이 나타나지 않을 경우 0(zero)을 값으로 지닌다. 예를 들어, 포인터 변수 p에 대한 *(p+i)과 같은 포인터 변수 연산식이 나타나는 경우 p에 대한 가상 변수는 [1]p[i]로 나타낸다 *p의 경우는 [1]p[0]이고 좌·우첨자가 0(zero)인 경우는 생략 가능하다 즉 *p = [1]p로 표현한다

다음 표를 살펴보자. 첫 번째 열은 원시 프로그램 문장들이고, 두 번째와 세 번째 열은 각 문장에서의 계산된 분석 정보이다 네 번째 열은 슬라이싱 기준 c=<8, x>로 주어졌을 때 계산한 RIN 정보이다

		Def	Ref	RIN
1	*(p+i) = c1;	[1]p[i]	p, i, c1	p, i, j
2	*(p+j) = c2,	[1]p[j]	p, j, c2	p, i, j, [1]p[i]
3	if(c)			p, i, j, [1]p[i], [1]p[j]
4	k = i,	k	i	p, i, [1]p[i]
5	else			
6	k = j,	k	j	p, j, [1]p[j]
7	x = *(p+k),	x	p, k, [1]p[k]	p, k, [1]p[k]
8	printf(x)		x	

<표 2> 새로운 표기법을 적용한 예

<표 2>을 자세히 살펴보면, RIN(7)에 포함된 가상 변수 [1]p[k]가 RIN(6)에서는 [1]p[j]로 바뀌었다 즉, 6번 문장에서 k의 값이 j의 값에 직접적인 영향을 받기 때문에 가상 변수의 우첨자 리스트에 나타나는 k 이름도 j로 바뀐다 따라서 [1]p[k]는 [1]p[j]로 재명명 되어진다.

먼저 문자를 원소로 지니는 집합 S에서 원래 원소 a를 원소 b로 교체하여 새로운 집합 S'을 생성하는 Rep 연산을 다음과 같이 표기한다

$a \in S,\ S' = Rep(S)_{a \to b}$

이 기호로서 가상 변수의 재명명에 대한 공식을 정의하면 다음과 같다.

슬라이싱 기준 c가 주어졌을 때, m∈IMP(n)에 대하여,

$\exists a \in S,\ Ref(m) \neq \emptyset,\ a \in Def(m) \wedge [k]p[S] \in RIN(n)$

$\Rightarrow [k]p[S'] \in RIN(m),\ a \in S,\ S' = Rep(S)_{a \to Ref(m)}$

4 슬라이싱

다음은 앞서 제안한 내용을 슬라이싱에 적용한 예이다 <그림4>는 ⑩번 문장 수행 후, 포인터 변수 min에 대한 ORSG이다

```
/* call   func (&a, &b, &c)
return the square of the minimum of a, b, and c */
int func (int *x, int *y, int *z)
{
① int  *min,  square,
w;
② w = *x;
③ if (*y < w)
④    w = *y;
⑤ min = &w,
⑥ if (*z < *min)
⑦    min = z;
⑧ w = *min;
⑨ square = w*w,
⑩ return square,
}
```

<그림 4> 포인터 변수 min에 대한 ORSG

포인터 변수 min의 상태를 변경하는 문장은 ⑤, ⑦이며, P_1(⑤, min) = {w}, P_1(⑦, min) ={c}이다 ⑧번 문장에서의 P_1(⑧, min)의 값은 P_1(⑤, min) ∪ P_1(⑦, min) ={w, c}가 된다 슬라이싱 기준을 c=<⑧, w>로 두었을 때, 포인터 변수 min뿐만 아니라 min이 참조하는 객체에 대한 슬라이스도 포함되어져야 한다. 즉, c=<⑧, min, w, c>가 된다. 슬라이스에 포함되는 문장은 ①에서 ⑧번까지 이다

5. 결론 및 향후 과제

프로그램 슬라이싱은 특정 계산에 관련 있는 문장 추출에 기반을 둔 프로그램 분해 방법이다[1]. 본 논문은 포인터 변수와 같은 복잡한 자료 구조를 지닌 프로그램에 대한 정확한 슬라이스 생성에 관한 연구이다 포인터 연산식과 배열 원소, 구조체 필드 접근 연산이 존재하는 경우 기존의 방법보다 더 정확한 자료 정보를 생성함으로서 더 정확한 슬라이스를 생성할 수 있다.

향후 과제로는 동적 객체의 무제한적 생성 시에 발생하게 되는 슬라이싱 문제를 해결하는 것과 자동화된 슬라이서 도구 개발이다

【 참고 문헌 】

[1] Mark Weiser, "Program Slicing", IEEE Trans Software Eng., Vol SE-10, No 4, pp 352-357, July 1984

[2] Jingyue Jiang, Xiling Zhou, David Robson, "Program Slicing For C-The Problems in Implementation", Proc IEEE International Conf Software Maintenance, pp 182-190, 1991.

[3] James R. Lyle, David Binkley, "Program Slicing in the Presence of Pointers", Preceedings of the 3RD Annual Software Engineering Research Forum, November 11-12 1993.

[4] Pandos E Livadas, Adam Rosenstein, "Slicing in the Presence of Pointer Variables", Ghinsu Project Technical reoprt, 1995.

[5] Daivd R Chase, Mark Wegman, F Kenneth Zadeck, "Analysis of Pointers and Structures", ACM, 1990