C 프로그램의 이해를 지원하는 동적 조각화 알고리즘

김태희* • 강문설**

Dynamic Slicing Algorithms for Understanding of C Programs

Tae-Hee Kim* • Moon-Seol Kang**

본 연구는 2004년 동신대학교 교내연구비 지원으로 수행되었음.

요 약

동적 조각화 기법은 프로그램을 이해하기 쉬운 조각단위로 분해하여 소프트웨어 개발자나 유지보수자가 프로그램을 쉽게 이해할 수 있도록 지원한다. 본 논문에서는 프로그램을 조각화하는데 걸리는 시간을 단축할 수 있는 동적 조각화 알고리즘을 제안하였다. 모든 문장에 대해 배정연산자를 기준으로 우측에서 사용되는 참조변수집합과 좌측에서 사용되어 변수의 값에 영향을 미치는 변경변수집합을 산출하고, 변수선언부분의 모든 변수에 대해 변수간 관련성(VV)을 작성하였다. 이를 활용하여 제안한 알고리즘을 적용하면 동적조각을 추출할 수 있다. 제안한 알고리즘의 성능을 기존방법과 비교한 결과 기준변수의 개수가 3개 이상인 경우에는 문장의 평균 비교횟수를 35% 감소시킴으로써 동적조각을 추출하는 과정에 소요되는 시간을 단축시킬 수 있었다.

ABSTRACT

Dynamic slicing method decomposes a program into slices and supports to be understood programs easily by software developer or maintainer. In this paper, we propose dynamic slicing algorithm to reduce time to decompose a program. We produce reference-variable set used in right and modify-variable set used in left on the basis of the assignment operator of all sentences and extract Inter-Variable Relationship(VV) for all variables of variable declaration. Proposed algorithm extracts dynamic slices by using them and execution trace of program. In conclusion, proposed algorithm improved the performance by reducing the time to extract dynamic slices by decreasing average comparison count of sentence when the number of criterion variables is three or more.

키워드

동적 조각화(Dynamic Slicing), VV(Inter-Variable Relationships), 변경변수 집합, 참조변수 집합

1. 서 론

소프트웨어를 이해하는 작업은 소프트웨어와 관련된 분석, 설계, 구현 등의 모든 작업 중에서 중요한 역할을 수행하고, 유지보수와 재사용 활동에 깊이 관련되어 있다. 프로그램 전체를 이해하는 작업은 결코 쉬운 일은 아니기 때문에 보다 용이하게 프로그램을 효율적으로 이해하는 방법이 요구되고 있다. 복잡하고 규모가 큰 프로그램을 유지보수자나 개발자들이 이해하고 조작하기 용이하도록 분해하는 기법으로 프로그램 조각화가 있다[5, 7]. 한 개의 프로그램을 여러 개의 작은 컴포넌트로 분해하여 유지보수자나 개발자들이 참조해야 하는 세부적인 컴포넌트들을 감소시

* 동신대학교 정보생활과학대학 디지털콘텐츠학과 **광주대학교 공과대학 컴퓨터학과

접수일자 : 2004. 11. 22

킴으로써 코드에 대한 이해를 증진시킨다. 조각화 알고리즘은 정적으로 가능한 정보만을 사용하는 정적 조각화와 특정 입력에 대한 프로그램 실행 추적결과를 대상으로 변수 값에 영향을 미치는 문장을 추출하는 동적 조각화로 분류된다[3, 4].

함수 호출, 포인터 변수, 배열참조에 기인하는 정적인 문제점을 갖고 있는 정적 조각화 기법보다 임의의 입력 값에 의한 프로그램 수행결과를 활용하여 조각을 추출하는 동적 조각화 기법을 많이 활용하는 경향이 있다. 본 논문의 2장에서는 관련연구를 기술하고, 3장에서는 제안한 동적조각화 알고리즘을 기술한다. 4장에서는 Korel의 방법과 문장의 비교횟수를 분석한 결과를 제시하고, 마지막으로 5장에서는 결론과 향후 연구방향에 대하여 논한다.

2. 관련연구

2.1 Korel의 블록화 방법

Korel[6]은 프로그램에서 일련의 문장을 블록화하는 방법을 사용하여 기준변수와 관련 있는 문장을 추출하는 동적조각화 방법을 제안하였다. 이 방법은 문장의 번호와 실행 순서가 명시된 실행추적결과를 사용하고, 각 문장들을 묶음으로 블록화하였다. 분기문과 관련된 문장들에 대해서는 j-entry / j-exit, r-entry / j-exit, j-entry / r-exit의 조건을 부여하고, 나머지 임의의 문장들에 대해서는 r-entry / r-exit의 조건을 부여하였다. 임의의 문장과 비교하여 기준변수와 관련 있는 문장을 찾고, 그 문장이 포함되어 있는 블록의 나머지 문장에 대해서도 기준변수와의 관련성을 조사하고자 문장들을 비교하였다. 문장의 많은 비교횟수로 실행시간이 길어진 단점이 있다.

2.2 글로벌 조각화(Global Slicing)

글로벌 조각화 알고리즘은 D/U(Definition /Use) 프로그램 표현방법을 사용하여 데이터 의존성과 제어의존성 모두를 나타내었다[2]. 문장의 종류에 따라 조건문은 문장번호 앞에 'p'를 붙이고(p3, p6 등), 출력변수는 문장번호 앞에 'o'를 붙여서(o12, o15 등) 동적 조각화에 필요한 정보를 작성하였다. 매 문장마다 정의된 변수명과 문장번호를 함께 기술하였다. 포인터변수는 메모리 위치의 동적 의존성을 찾아야 하므로 포인터 변수명과 d1을 함께 기술하고 그 이후의 포인터변수 표현은 d2로 하나씩 증가시켜 나간다. 함수 호출은 함수명과 매개변수, 매개변수의 위치를 함께 표기하고, 함수의 반환값을 함수명과 함께 표기하였다. 이 알고리즘은 최근에 수행된 명령어에 영향을 미치는 문장의 집합을 계산하여 저장하기 때문에 원래 소스코드보다 적은 공간을 요구한다. 이 방법은 종류별로 영문기호를 다르게 표현해야하는 약간의 번거로움을 수반하고 구문에 포함되는 than, else, do와 같은 키워드를 모두 제외시키고 문장의 번호를 부여하기 때문에 최종 동적 조각을 추출 후 문장의 배열을 요구하는 단점이 있다.

2.3 제한된 전처리 동적 조각화

이 알고리즘[1]은 프로그램을 수행 추적한 결과를 추적블록(trace block)으로 나누고, 그 블록의 끝부분에 변수이름과 메모리 주소에 대해 밖으로 드러난 요약 정보(summary information)를 저장하였다. 이러한 정보는 역방향 추적 수행을 하는 동안 산출된다. 요약정보는 블록단위로 저장되기 때문에 조각을 추출하는 과정에서 그 블록이 실행되지 않으면 조각화 과정에 소요되는 노력이 줄어들 수 있었다. 따라서 추적결과를 대상으로 조각을 추출하는 방법보다 문장 비교에 소요되는 비용과 요약정보의 크기가 감소됨을 장점으로 한다. 이는 비트백터(bit vector)를 사용하여 요약정보를 저장하기 때문이다. 이 방법은 그 블록에 포함되는 한 개의 문장이 동적조각으로 추출되는 경우, 불필요함에도 불구하고 그 블록 전체가 조각에 포함되어야 하는 단점을 갖는다.

3. 동적 조각화 알고리즘

3.1 프로그램 구성과 기준문장

본 논문에서 동적 조각을 추출하는 대상 프로그램은 C 언어로 작성된 프로그램이다. 변수의 타입은 배열과 포인터를 포함한 정수형, 실수형, 문자형 등을 대상으로 하고, if, while, repeat-until, for 등의 제어·반복 구조를 포함한다. 함수간의 조각화에서는 함수의 인수 전달 기법에서 값에 의한 호출(call-by-value)과 포인터 인수를 고려하여 정확한 조각을 추출한다. 조각화하기 위해서 입력으로 들어가는 프로그램은 일정한 형식을 갖추어야 한다. 첫째, 변수 선언과 변수 초기화하는 문장은 하나의 문장에 하나의 변수가 선언되도록 해야 한다. 설정된 조각화 기준과 프로그램의 나머지 문장을 비교하여 동적 조각을 추출할 때 하나의 문장에 변수가 두 개 이상 선언되는 경우에는 하나의 변수 때문에 필요하지 않은 나머지 변수들이 모두 비교되어지고, 조각화 기준과 관련이 없는 변수가 포함

될 수 있기 때문이다. 둘째, if식에서 if조건식, then문, else문을 코딩할 때와 switch문 에서 case 문을 코딩할 때 각각 한 문장으로 취급하여 문장번호를 부여한다. if나 case인 경우 하나의 조건식에 두 개 이상의 문장이 수행되기 때문에 조각화 기준과 관련이 있는 문장만을 추출하기 위해서는 (그림 1)에서 보여주는 것처럼 일정한 형식을 갖추어야 한다. 셋째 변수선언문과 배정문을 제외한 나머지 문장 가운데 조건식이 포함된 문장과 함수 호출하는 문장과 호출되는 문장에는 체크 기호(√)를 사용하여 표시한다. 동적 조각 추출 마지막 단계에서 조건에 관련되어 실행되어야 하는 문장이 하나도 없는 경우를 찾아서 조건문을 동적 조각에서 제외시킬 때 활용한다.

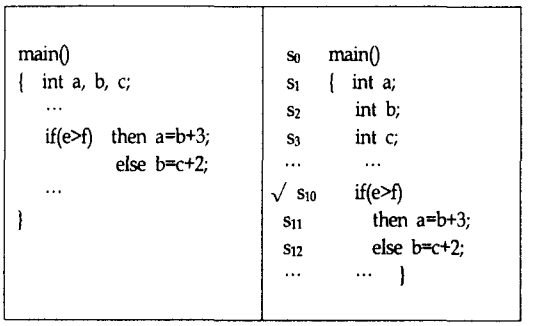


그림 1. 프로그램의 일정한 형식
Fig 1. The unique form of program

프로그램의 기준 변수와 기준 문장을 선정한 후 조각화를 실시하여 프로그램에 대한 이해를 향상시킬 수 있다. 호출된 함수가 조각에 포함되는 경우에는 함수 전체를 대상으로 하여 관련된 모든 문장을 추출한다. 메인 함수에 기준 변수와 기준 문장이 있는 경우와 호출되는 함수에 기준 변수와 기준 문장이 있는 경우를 분리하여 고려한다.

3.2 제안한 알고리즘 적용을 위한 선행 작업

(1) 각 문장에 대해 관련 변수집합 산출

대상 프로그램의 모든 문장에 대해 배정연산자를 기준으로 좌측에서 사용되는 변수들, 즉 값이 변경되는 변수집합과 우측에서 사용되는 변수들, 즉 값이 참조되는 변수집합으로 나눌 수 있다. 변수집합을 나누어 계산하는 이유는 기준 변수가 변경되는 변수집합의 원소인 경우에는 그 문장은 반드시 관련 문장집합에 포함되어야 하고, 기준 변수가 참조되는 변수집합의 원소인 경우에는 관련 문장집합에 포함되지 않을 수도 있기 때문이다.

[정의 1] 문장의 변수집합

프로그램 P에서 문장번호 순서상 i번째 문장을 s_i로 나타내고, 문장 s_i에서 사용되는 변수집합을 $U(s_i)$로 나타낸다. 문장의 변수집합 $U(s_i)$는 문장 s_i에서 참조되는 변수의 집합 $U_{REF}(s_i)$과 변경되는 변수집합 $U_{MOD}(s_i)$으로 구성되며, 다음과 같은 수식으로 정의한다.

$$U(s_i) = U_{MOD}(s_i) + U_{REF}(s_i)$$

제어 구조에 포함된 문장들의 참조되는 변수집합에는 제어 조건식에 사용된 변수와 제어 조건식의 문장번호를 포함시킨다. 이유는 제어 구조에서 사용되지 않는 변수가 제어 조건식에 사용되었을 때 제어 조건식이 정적 조각에 포함되지 않을 수 있기 때문이다.

① 각 문장에 대한 변수집합은 변수간 관련성 (inter-variable relationships : VV)을 식별할 때 활용할 수 있다. 다음과 같은 코드가 있다고 하자.

```
s5    ...
s6    sum = sin + zin;
s7    mul = ain / fin;
s8    while(i < n)
s9    { sum = ain + 2;
```

② 이 때 문장에 대한 변수의 집합을 배정연산자를 기준으로 변경되는 변수의 집합과 참조되는 변수의 집합으로 분류하여 나타내보면 다음과 같다.

$U_{MOD}(s_5)=\{\cdots\}$	$U_{REF}(s_5)=\{\cdots\}$
$U_{MOD}(s_6)=\{sum\}$	$U_{REF}(s_6)=\{sin, zin\}$
$U_{MOD}(s_7)=\{mul\}$	$U_{REF}(s_7)=\{ain, fin\}$
$U_{MOD}(s_8)=\{i, n\}$	$U_{REF}(s_8)=\{s_9\}$
$U_{MOD}(s_9)=\{sum\}$	$U_{REF}(s_9)=\{ain, i,n/s_8\}$

[정의 2] 변수간 관련성 (Inter-Variable Relationships)

프로그램 P에서 문장 번호 순서상 i번째 문장을 s_i로 나타내고, 프로그램에서 변수 선언 부분의 마지막 문장(cl번째)을 s_{cl}이라고 나타내며, 프로그램에서 마지막 문장(ls : *last sentence*)을 s_{ls}라 나타낸다. 그리고 프로그램에서 사용되는 변수를 v

로 나타내고, 변수 v를 선언한 문장 s_i에 대한 변수간 관련성을 $VV(s_i(v))$로 나타냈을 때, 변수간 관련성 $VV(s_i(v))$를 다음과 같이 정의한다.

$$VV(s_i(v)) = \{s \mid s \in v = U_{MOD}(s_t),\ 1 \le i \le cl,\ cl+1 \le t \le ls\}$$

[정의 2]에서 설명된 바와 같이 변수간 관련성은 임의의 변수 i와 [정의 1]에서 산출된 $U_{MOD}(s_t)$를 비교하여 동일한 경우 그 때의 (s_t)에 해당되는 문장집합을 의미한다. s_t에서 t의 범위는 선언 부분 다음문장$(cl+1)$부터 프로그램 마지막 문장(ls)까지를 포함한다. 변수 i와 관련있는 문장을 추출하기 위해서는 프로그램의 모든 문장의 모든 변수와 비교해야 하는데, [정의 2]에 따르면 $U_{MOD}(s_t)$ 집합의 변수만 비교하므로 비교되는 횟수를 줄일 수 있는 장점이 있다. 변수간 관련성은 변수 선언 부분의 문장만을 대상으로 산출한다. 본 논문에서 연구 및 실험 대상으로 선정한 C 프로그램은 문법상 변수가 변수 선언 부분에 반드시 선언되어야 하므로 프로그램에서 사용되는 모든 변수가 변수 선언 부분에 포함되어 있다. 변수간 관련성을 조사하는 작업도 자동으로 이루어진다. [정의 2]에서 정의한 변수간 관련성을 식별해보자. 원시 프로그램과 변수 선언 부분에 대한 변수간 관련성 $VV(s_i(v))$를 식별하면 다음과 같다.

(2) 원시 프로그램에서 조건문장에 체크

원시 프로그램에서 조건이 기술된 배정연산자의 기호가 없는 문장에 대해서 문장번호에 체크기호 "√"를 표시하고, 함수호출 문장에도 체크기호 "√"를 표시한다. 동적 조각을 추출한 후 마지막 단계에서 블록으로 실행되어야 하는 문장을 체크할 때 사용한다. 조건과 관련되어 실행되는 문장이 한 개도 없음에도 불구하고 조건이 명시된 문장만 동적 조각에 포함되는 경우를 제외시키는데 활용한다. 이 과정은 (그림 2)에서 보여준다.

```
 s1   int i;
 s2   int j;                        VV(s1(i))={s1,s5,s15}
 s3   int R2;                       VV(s2(j))={s2,s5}
 s4   int R3;                       VV(s3(R2))={s3,s6,s13}
 s5   scanf("%d%d",&i, &j);         VV(s4(R3))={s4,s7,s10}
 s6   R2 = 0;
 s7   R3 = 0;
√s8   do {
√s9     if(i%3) {
 s10        R3++;
 s11        printf("%d",i); }
√s12    if(i%2) {
 s13        R2++;
 s14        printf("%d",i); }
 s15    i++;
√s16   }while(i != j);
```

3.3 제안한 동적 조각화 알고리즘

(1) 평서문, 제어·반복구조 조각화

프로그램의 대부분을 구성하고 있는 평서문에는 초기값 설정, 계산식, 입력문, 출력문 등이 있고, 평서문을 조각화하는 방법은 문장에서 변경변수(U_{MOD})와 참조변수(U_{REF})를 활용하는 것이다. 조각 추출의 대상이 되는 문장의 변수는 변경변수로 값의 변화에 영향을 미칠 수 있는 특성을 갖는다. 입력문과 출력문의 경우는 배정 연산자가 없기 때문에 포함된 변수들 모두를 문장의 변경변수에 넣는다. 입력 값이 주어진 후 프로그램이 실행되면 실행 추적결과가 산출된다. 이 산출된 추적결과를 대상으로 기준변수와 기준문장이 결정되면, 선언부분에서 기준변수가 포함된 문장의 VV을 참조하여 동적 조각을 추출할 수 있다. 문장에서 영향을 미치지 않는 참조변수를 분리시켰기 때문에 불필요하게 발생되는 변수 비교횟수를 줄일 수 있다.

조건문, 제어문, 반복문의 경우 조건이 포함된 문장에서 조건변수는 변경 변수집합에 포함시키고, 참조 변수집합에는 이 조건문장과 관련 되어 함께 수행해야하는 문장번호를 모두 포함시킨다.

이 참조변수집합은 동적 조각을 추출할 때 함께 수행되는 문장이 포함되지 않았음에도 불구하고 조건을 나타내는 문장만이 포함되는 경우를 제외시키기 위한 방법에 사용된다. 값의 변화에는 영향을 미치지 않겠지만, 조건 체크는 조건이 참일 동안 매번 실행되어야 하므로 변경 변수집합에 포함시켜서 조각추출 대상이 되어야 한다. 또한 조건문, 제어문, 반복문 범위에 속하는 문장의 경우 참조 변수집합에 조건을 체크하는 문장번호를 함께 기술함으로써 기준변수와는 직접적인 관련은 없지만 조건을 체크하는 문장이 포함되어야

함을 표시하도록 한다. C 프로그램에서는 반복문의 경우 시작과 끝 부분에 조건이나 변수가 포함되지 않은 키워드로만 구성된 문장을 포함하는데, 이 키워드로만 구성된 문장은 항상 조각에 포함시킨다. 조건 체크는 반드시 한 번은 해야 하기 때문이다. 제어문의 경우 then, else와 같이 다음 문장에 걸리는 키워드는 연결되는 다음 문장과 함께 문장 번호를 주게 되면 조각 추출이 성공적으로 이루어질 수 있다.

```
s1
s2
s3   int n;
s4   int a;
s5   int i;
s6   int s;
s7   scanf("%d%d",&n,&a);
√s8  i = 1;
 √   s = 1;
s9   if (a > 0)
√s1     s = 0;
 0   while( i <= n ) {
√s1      if (a > 0)
 1           s += 2;
s12      else  s *= 2;
s13      i++; }
s14      printf("%d", s); }
√s1
 5
```

```
s1
s2
s3
s4   int n;
s5   int a;
s6   int i;
s7   int s;
√s8  scanf("%d%d",&n,&a);
√s1  i = 1;
 0   s = 1;
√s1  if (a > 0)
 1   while( i <= n ) {
s13      if (a > 0)
s14      else  s *= 2;
√s1   i++; }
 0   while( i <= n ) {
√s1      if (a > 0)
 1       else  s *= 2;
s13   i++; }
s14   while( i <= n ) {
√s1   printf("%d", s); }
 0
s15
```

(a) 원시 프로그램 (b) 프로그램 실행추적결과

그림 2. 원시 프로그램과 프로그램 실행추적결과

Fig. 2 Source program and Excuse trace

(그림 2)는 원시 프로그램과 a = 0, n = 2인 입력값에 의해서 실행되는 프로그램 실행추적결과를 보여준다. 원시 프로그램을 대상으로 산출해 놓은 변경 변수집합, 참조 변수집합, VV를 프로그램 실행추적결과에 대해서 기술하면 (그림 3)과 같다.

실행	변경변수 집합(U_{MOD})	참조변수 집합(U_{REF})	VV
1	n		s_1, s_5, s_{10}
2	a		s_2, s_5, s_8, s_{11}
3	i		s_3, s_6, s_{10}, s_{14}
4	s		s_4, s_7, s_{13}, s_{15}
5	n, a		
6	i		
7	s		
8	a	s_9	
10	i, n	s_{11},s_{12},s_{13},s_{14}	
11	a	i,n/s_{10},s_{12},s_{13}	
13	s	a/s_{11},i,n/s_{10}	
14	i	i,n/s_{10}	
10	i, n	s_{11},s_{12},s_{13},s_{14}	
11	a	i,n/s_{10}	
13	s	a/s_{11},i,n/s_{10}	
14	i	i,n/s_{10}	
10	i, n	s_{11},s_{12},s_{13},s_{14}	
15	s		

그림 3. 문장의 변경 변수집합, 참조 변수집합, 변수선언 문장의 VV

Fig. 3 Modify-variable set, reference-variable set, VV of declaration sentence

(그림 3)의 내용을 활용하여 동적 조각을 추출하는 과정을 단계별로 표현하면 다음과 같다.

(단계 1) 기준 문장 : s_{15}, 기준 변수 : s
프로그램 실행 추적 문장에 대해 변수간 관련성 집합을 산출한다. s를 변수 선언 부분의 문장에서 찾아 변수간 관련성 집합을 관련 문장 집합에 포함시킨다. $DS(s)$ = { s_4, s_7, s_{13}, s_{15} }

(단계 2) 5개 문장의 변수집합을 찾아서 집합에 포함된 변수의 변수간 관련성 집합을 찾는다.
: $U_{REF}(s_4)$={ }, $U_{REF}(s_7)$= {}, $U_{REF}(s_{13})$={ a/s_{11},i,n/s_{10}}
: 변수 i, n, a 에 대한 변수간 관련성 찾는다.
VV 산출 i={s_3, s_6, s_{10}, s_{14}}, n={s_1, s_5, s_{10}},
a={s_2, s_5, s_8, s_{11}}
$DS(s)$={s_1, s_2,, s_3, s_5, s_6, s_8, s_{10}, s_{11}, s_{14}}

(단계 3) 단계 1과 단계 2의 합집합을 산출한다.
$DS(s)$ = {s_1, s_2,, s_3, s_4,, s_5, s_6, s_7, s_8, s_{10}, s_{11}, s_{13}, s_{14}, s_{15} }

(단계 4) 실행추적문장의 조건문을 대상으로 비교한다. s_8, s_{10}, s_{11} 문장을 대상으로 함께 실행해야할 문장이 한 개라도 발견되면 그 조건문장은 반드시 포함시켜야 하고, 발견되지 않으면

관련된 그 조건문을 단계 3의 결과에서 제외시킨다 .

제외문장 집합 = { s_8 }

∴ $DS(s)$={s_1, s_2,, s_3, s_4,, s_5, s_6, s_7, s_{10}, s_{11}, s_{13}, s_{14}, s_{15}}

(2) 호출함수의 조각화

```
s1   int a;
s2   int b;
s3   int f(int x, int y)
s4   { a = a + x;
s5     b = b + y;
s6     return x+2; }
s7   int g(int y)
s8   { a = a + y;
s9     return  y+1;  }
     void main()
s10  { int s;
s11    int *p;
s12    s=0;
s13    scanf("%d", &a);
s14    scanf("%d", &b);
s15    p = &b;
s16    while (*p < 10){
s17      s = s + f(3, 4);
s18      s = s + g(3); }
s19    printf("%d", *p);
s20    printf("%d", s); }
```

```
s1   int a;
s2   int b;
     void main()
s10  { int s;
s11    int *p;
s12    s=0;
s13    scanf("%d", &a);
s14    scanf("%d", &b);
s15    p = &b;
s16    while (*p < 10){
s17      s = s + f(3, 4);
s3   int f(int x, int y)
s4   { a = a + x;
s5     b = b + y;
s6     return x+2; }
s17      s = s + f(3, 4);
s18      s = s + g(3);}
s7   int g(int y)
s8   { a = a + y;
s9     return  y+1;  }
s18      s = s + g(3);}
s16    while (*p < 10){
s19    printf("%d", *p);
s20    printf("%d", s); }
```

그림 4. 원시 프로그램과 실행추적결과(a=2, b=6)

Fig. 4 Program and Excuse trace

본 논문에서는 함수 호출의 경우 값에 의한 호출(call-by-value)과 포인터 사용 호출을 고려하여 동적 조각을 추출한다. 값에 의한 호출인 경우에는 인수를 문장(s_i)의 참조 변수집합 $U_{REF}(s_i)$에 포함시키고, 포인터 인수인 경우에는 인수를 문장(s_i)의 변경 변수집합 $U_{MOD}(s_i)$에 포함시킨다. 메인함수에서 호출한 문장의 변경 변수집합에 호출되는 함수 문장들의 번호를 기술하여 관련이 있음을 표시한다. (그림 4)에서는 예제 프로그램을 보여주고, (그림 5)에서는 입력 값에 따른 프로그램 실행 추적결과와 변수간 관련성, 문장에 대한 변수집합을 보여준다.

실행	변경변수 집합(U_{MOD})	참조변수 집합(U_{REF})	V V
1	a		s_{13}, s_4, s_8
2	b		s_{14}, s_5
10	s		s_{12}, s_{17}, s_{18}, s_{20}
11	*p		s_{15}, s_{16}, s_{19}
12	s		
13	a		
14	b		
15	p	b	
16	*p	s_{17}, s_{18}	
17	s	x, y/s_3, s_4, s_5, s_6, *p/s_{16}	
3		x, y, s_4, s_5, s_6	
4	a	x, s_3	
5	b	y, s_3	
6	x	s_3	
17	s	x, y/s_3, s_4, s_5, s_6, *p/s_{16}	
18	s	y/s_7, s_8, s_9, *p/s_{16}	
7		y, s_8, s_9	
8	a	y, s_7	
9	y	s_7	
18	s	y/s_7, s_8, s_9, *p/s_{16}	
16	*p	s_{17}, s_{18}	
19	*p		
20	s		

그림5. 문장에 대한 변수집합

Fig. 5 The variable set of a sentence

```
s1   int a;
s2   int b;
s3   int f(int x, int y)
s4   { a = a + x;
s5     b = b + y;
s6     return x+2; }
s7   int g(int y)
s8   { a = a + y;
s9     return  y+1;  }
     void main()
s10  { int s;
s11    int *p;
s12    s=0;
s13    scanf("%d", &a);
s14    scanf("%d", &b);
s15    p = &b;
s16    while (*p < 10)     {
s17      s = s + f(3, 4);
s18      s = s + g(3);     }
s19    printf("%d", *p);
s20    printf("%d", s); }
```

그림 6 입력값 a=2, b=6과 조각화 기준 C = <s_{20}, s>에 대한 동적 조각

Fig. 6 Input a=2, b=6 and C = <s_{20}, s>

(3) 포인터변수의 조각화

포인터변수인 경우는 기호 "*"의 유무에 상관없이 변수명이 일치하면 변수간 관련성, 참조 변수집합, 변경 변수집합을 산출한다. 입력함수와 포인터변수를 표현하는 기호 "&"를 찾게 되면 함께 사용한 변수명을 변경변수집합에 포함시키도록 한다. 입력함수의 입력을 받아들이는 변수와 포인터변수에 주소를 할당하는 변수는 값의 변화를 가져올 수 있기 때문에 변경 변수집합에 넣는다. (그림 6)은 추출된 동적 조각을 보여준다.

입력 : 조각화 기준 C = (cs, cv)
출력 : 조각화 기준 C에 대한 동적 프로그램 조각
$DS(cv)$: 조각화 기준과 관련된 문장의 집합
s_{ds} : 프로그램에서 변수 선언 부분의 마지막 문장
s_{ls} : 프로그램에서 마지막 문장
cs : 조각화하는 기준 문장
cv : 조각화하는 기준 변수

program DynamicSlice
begin
initialize s_{ls}
construct $U_{MOD}(s_i)$, $U_{REF}(s_i)$
check the predicate sentence and the functional call sentence and the called function sentence
construct EH, $VV(s_i(v))$ in EH
for k=1 to number of elements in EH
define cs, cv
$DS(cv) = Us_{k \in U}\ (VV(s_k(v)) \cup U_{REF}(s_k))$ $VV \in s_k$
if ($DS(cv) \supseteq U_{REF}(s_k)$) $DS(cv) = DS(cv)$ - (s_k *in* $U_{REF}(s_k)$)
endfor
output s_{ls} and DS sets for the last definition of all variables
end

그림 7. 동적 조각화 알고리즘
Fig. 7 Dynamic Slicing Algorithm

4. 적용 사례 및 결과분석

본 논문에서 제안한 동적 조각화 알고리즘의 성능을 평가하기 위해서 C 언어로 작성된 프로그램을 선정하였다. 한 개 이상의 함수로 구성된 30~50라인 정도의 200개 프로그램을 대상으로 동적조각을 추출하는데 문장의 비교횟수를 식별하여 분석한 결과를 제시한다. 문장의 비교횟수는 동적조각을 추출하는 알고리즘의 성능을 평가하는 데 매우 중요한 요소이다. 기존의 그래프를 사용했던 알고리즘과 비교해보면 동적조각 추출과정에서 사용된 메모리크기도 감소됨을 알 수 있다.

```
s1  int start;
s2  int end;
s3  int add;
s4  int sum;
s5  int by;
s6  int temp;
s7  sum=0;
s8  by=0;
s9  scanf("%d", &start);
s10 scanf("%d", &end);
s11 scanf("%d", &add);
s12 while (start <=end) {
s13     sum = sum + start;
s14     temp = start % 3;
s15     if(temp=0)
s16       by = by + 1;
s17     start = start + add; }
s18 start = start - add;
s19 printf("%d", sum); }
```

```
s1  int start;
s2  int end;
s3  int add;
s4  int sum;
s5  int by;
s6  int temp;
s7  sum=0;
s8  by=0;
s9  scanf("%d", &start);
s10 scanf("%d", &end);
s11 scanf("%d", &add);
s12 while (start <=end) {
s13     sum = sum+start;
s14     temp = start % 3;
s15     if(temp=0)
s16       by = by + 1;
s17     start = start+add;}
s12 while (start <=end) {
s18  start = start - add ;
s19  printf("%d", sum); }
```

그림 8. 원시 프로그램과 입력값 start=3, end=3, add=5인 실행추적결과
Fig. 8 Source program and Excuse trace about start=3, end=3, add=5

본 논문에서 제안한 알고리즘과 Korel의 블록화 방법에서 비교횟수를 조사하는데 (그림 8)의 예제를 대상으로 한다. 첫째, Korel방법은 관련된 문장이 제어구조에 포함되어 제어구조의 나머지 문장까지 비교대상이 되는 반면 제안한 알고리즘은 관련된 문장만을 비교하게 되므로 문장을 비교하는 횟수를 줄일 수 있다. 둘째, Korel방법은 기준변수가 바뀔 때마다 매번 모든 문장을 비교해야 하는 반면 제안한 알고리즘은 한번 작성된 VV을 이용하기 때문에 모든 분장을 비교하지 않고 쉽게 관련 문장을 추출할 수 있다. 또한 기준변수가 여러 개인 경우에는 Korel방법보다 제안한 알고리즘이 문장의 횟수를 감소시킴으로써 실행시간을 단축할 수 있다. (표 1)은 두 방법의 기준변수에 따른 문장의 비교횟수를 조사한 테이블로 문장의 비교횟수가 감소함을 알 수 있다. 기준변수가 2개 또는 3개인 프로그램을 대상으로 문장의 비교횟수를 분석한 결과를 (표 2)에서 보여준다. (표 3)은 200개 프로그램을 대상으로 기준변수를 3개~6개를 선정하여 동적조각을 추출하는 경우 문장의 비교 횟수를 산출하고 평균을 보여준다.

표 1. 원시 프로그램을 대상으로 기준변수에 따른 문장의 비교횟수테이블
Table. 1 The comparison table of a sentence

방법 \ 변수명	sum	by	start	add	temp	end	VV	합계
Korel	106	113	94	31	107	31	-	482
제안 방법	+20	+30	+13	+4	+22	+3	156	248

표 2. 기준변수 개수에 따른 문장의 비교횟수테이블
Table. 2 The comparison table of a sentence t according to criterion variables

방법 \ 기준변수개수	2개	3개	4개
Korel	219회	313회	420회
제안한방법	206회	219회	241회
감소율	5%	30%	42%

표 3. 문장의 평균 비교횟수 테이블
Table. 3 The average the comparison table of a sentence

프로그램	프로그램 line	사용된 변수	기준 변수	평균 비교횟수		비교횟수 감소율
				Korel 방법	제안 방법	
200개	30~50 line	2개 ~8개	3개 ~6개	123회	92회	35%

프로그램 200여개를 대상으로 문장의 비교횟수를 비교하여 분석한 결과 평균적으로 35%의 감소율을 나타내었다. 동적조각을 추출하는 시간을 단축할 수 있고 추출과정에서 사용되는 메모리 크기도 줄일 수 있다. 단 기준변수가 2개 이하인 경우까지는 제안한 방법이 비교횟수가 많이 나왔다.

5. 결 론

본 논문에서는 C 언어로 작성된 프로그램을 이해하기 위한 방법으로 동적조각화 알고리즘을 제안한다. 제안한 방법은 선행 작업으로 원시프로그램의 모든 문장에 대해 변경변수집합, 참조변수집합, VV을 산출하고 이를 활용하여 프로그램실행추적 결과를 대상으로 동적조각을 추출한다. 기준변수가 3개 이상일 경우 Korel방법보다 문장의 평균 비교횟수를 35% 감소시켰고, 정확한 동적조각을 추출하는 시간을 단축시켰다. 향후 제안한 동적 조각화 알고리즘을 객체지향 프로그래밍언어를 대상으로 적용시킬 수 있도록 연구를 진행하는 중이다.

참고문헌

[1] Xiangyu Zhang, Rajiv Gupta, Youtao Zhang, "Precise Dynamic Slicing Algorithms", Proceedings of the 25th International Conference on Software Engineering(ICSE'03), 2003.

[2] Arpad Beszedes, Tamas Gergely, Zsolt Mihaly Szabo, Janos Csirik and Tibor Gyimothy, "Dynamic Slicing Method for Maintenance of Large C Programs", Proceedings of the Fifth European Conference on Software Maintenance and Reengineering(CSMR'01), 2001.

[3] H.Agrawal, R. DeMillo, and E. Spafford, "Debugging with Dynamic Slicing and Backtracking" Software Practice and Experience(SP&E), Vol. 23, No. 6, pp589-616, 1993.

[4] D.C.Atkinson, M.Mock, C.Chambers, and S.J.Eggers, "Program Slicing Using Dynamic Point-to Data", ACM SIGSOFT 10th Symposium on the Foundations of Software Engineering(FSE), pp29-40, November, 2002.

[5] F. Tip, "A Survey of Program Slicing tools", Technical Report, Dept. of Computer and Information Science, Linkoping University, 1993.

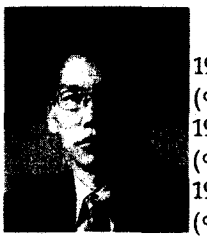
[6] B.Korel, "Computation of Dynamic Program Slices for Unstructured Programs" IEEE Transaction on Software Engineering, 23(1), pp17-34, January 1997.

[7] B. Korel, S. Yalamanchili, "Forward computation of dynamic program slices" International Symposium on Software Testing and Analysis(ISSTA), August 1994.

저자 소개

김태희(Tae-Hee Kim)

1991년 동신대학교 전산계산학과 (공학사)
1993년 전남대학교 전산통계학과 (이학석사)
1999년 전남대학교 전산통계학과 (이학박사)
1997년~현재 동신대학교 정보생활과학대학 디지털콘텐츠학과 조교수
※관심분야 : 소프트웨어 공학, 객체지향 모델링



강문설(Moon-Seol Kang)

1986년 전남대학교 전산통계학과 (이학사)
1989년 전남대학교 전산통계학과 (이학석사)
1994년 전남대학교 전산통계학과 (이학박사)
1994년~현재 광주대학교 공과대학 컴퓨터학과 부교수
※관심분야 : 소프트웨어 공학, 컴포넌트 기술, 객체지향 모델링