변수-변수 관련성을 이용한 동적 프로그램 조각 추출 알고리즘

김 태 희† · 김 병 기††

요 약

프로그램 조각화 기법은 프로그램을 이해하기 쉬운 조각 단위로 분해하여 소프트웨어 개발자나 유지보수자가 프로그램을 쉽게 이해할 수 있도록 지원하는 방법이다. 본 논문에서는 변수-변수 관련성을 이용하여 정확하고 수행 가능한 프로그램 조각을 추출하는 동적 프로그램 조각 추출 알고리즘을 제안한다. 각 문장에서 변경되는 변수와 참조되는 변수로 나누어서 변수 집합을 계산하고, 선언부에 있는 문장에 대해 변수-변수 관련성을 계산한다. 변수-변수 관련성을 계산할 때는 선언부의 변수가 다른 문장에서 변경되는 변수로 사용된 경우와 참조되는 변수로 사용된 경우를 별도로 조사하여 변경되는 변수 집합은 무조건 관련 집합에 포함시키고, 문장에서 참조되는 변수들은 문장들을 다시 비교하여 기준 변수와 관련된 문장만을 추출하여 관련 집합에 포함시킨다. 제안한 알고리즘을 C 언어를 대상으로 실험한 결과 정확하고 수행 가능한 동적 조각을 추출하였고, 기존의 방법들보다 관련 문장을 찾기 위한 문장의 비교횟수를 평균 42%까지 감소시켰다. 기준 변수가 많을수록 기준 변수와 관련이 없는 변수가 많을수록 문장의 비교 횟수가 현저하게 감소하였다.

An Extraction Algorithm of Dynamic Program Slice Using Variable-Variable Relationships

Tae-Hee Kim† · Byung-Ki Kim††

ABSTRACT

Program slicing is a method that supports software developer or maintainer to readily understand the program by decomposing it into several slices. In this paper we propose a dynamic program slice extraction algorithm that extracts the smallest and accurate program slice using variable-variable relationships. In this algorithm, we first define a set of variables from the left variable and the right variables in each sentence. Then, the variable-variable relationships is defined for the sentence in the declaration part. When we define the variable-variable relationships in the declaration part, the sentences with the left variable defined in the set are always included in the related set. On the other hand, if the variable in the set is a right variable, only the sentences that are related with the criterion variable are included in the related set after the sentences are compared and evaluated against other sentences. Proposed algorithm is found to extract accurate and executable dynamic slice when applied to C language. When searching for the related sentences, it also reduced the number of sentence comparisons up to 42% compared to existing methods. Reduction of the number of comparisons is marked as the number of criterion variables is increased and as the number of comparison variables that are not related with the criterion variable is increased.

† 정 회 원 : 동신대학교 컴퓨터학과 교수
†† 종신회원 : 전남대학교 컴퓨터정보학부 교수
논문접수 : 1998년 7월 14일, 심사완료 : 1998년 9월 9일

1. 서 론

소프트웨어 유지보수는 소프트웨어 개발 생명주기에서 필요한 노력과 비용 중 가장 많은 부분을 차지하고 있으며, 현실적으로 많은 문제점과 이 문제점들의 해결을 위한 노력과 비용이 표면에 노출되지 못하고 있다. 현재 사용하고 있는 소프트웨어의 유지보수 비용은 소프트웨어 개발 비용과 노력의 70% 이상을 차지하며, 이 비율은 더 많은 소프트웨어가 생산됨에 따라 소프트웨어 유지보수에 사용되는 노력과 자원의 양이 증가하는 심각한 현상이 유발되고 있다. 궁극적으로 일부 소프트웨어 개발 조직은 기존의 오래된 소프트웨어를 유지보수 하는데 사용 가능한 모든 자원을 소비해야 하기 때문에 앞으로는 새로운 소프트웨어를 개발하지 않는 "유지보수 전담(maintenance-bound)" 소프트웨어 개발 조직이 탄생할 수 있다[17].

프로그램을 이해하는 작업은 소프트웨어와 관련된 분석, 설계, 구현 등의 모든 작업 가운데 중요한 역할을 수행하고, 유지보수와 재사용 활동에 깊이 관련되어 있다. 이해 활동은 유지보수되거나 재사용되는 부품의 정확한 이해 없이 만들어질 수 없다. 프로그램을 대상으로 이해하는 작업은 코드 개선, 디버깅, 시험 방법과 같은 소프트웨어 개발 과정의 품질을 증진시키는데 반드시 필요한 작업이다. 이 소프트웨어를 개발하는 모든 과정에서 프로그래머는 프로그램을 읽고 이해해야만 한다. 이와 같이 프로그램을 이해하는 작업이 매우 중요하게 인식되고 있기 때문에 컴퓨터 프로그램을 분석하고 이해하도록 지원해주는 기법과 도구에 대한 연구가 지속적으로 이루어지고 있다[18].

한편, 프로그램 전체를 이해하는 작업은 결코 쉬운 일은 아니기 때문에 보다 용이하게 프로그램을 효율적으로 이해하는 방법이 요구되고 있다. 프로그램 조각화 기법(program slicing technique)은 프로그램을 이해하기 쉬운 조각(slice) 단위로 분해하여 개발자나 유지보수자가 프로그램을 쉽게 이해할 수 있도록 지원하는 방법이다. 즉, 프로그램 전체를 한번에 이해하는 것이 아니라 프로그램을 이해하기 용이한 조각으로 분해하여 부분적으로 이해할 수 있도록 지원해주는 기법으로 프로그램의 임의의 지점에서 기준 변수의 값에 영향을 미치는 문장을 추출하여 조각화한다. 프로그램을 조각화하는 기준에 따라 프로그램 조각화 기법을 여러 가지 종류로 분류할 수 있고, 유지보수 분야뿐만 아니라 디버깅, 시험, 재공학, 재구조화, 재사용 등에서 다양하게 응용되고 있다. 일반적으로 프로그램 조각은 정적 정보나 동적 정보에 의해 계산되어지고, 동적 조각화는 정적 조각화보다 정확한 조각을 추출할 수 있다는 장점이 있다[1,15].

본 논문에서는 동적 프로그램 조각화를 이용하여 정확한 조각을 추출할 수 있는 알고리즘을 제안한다. 프로그램이 실행되는 과정을 추적하고, 수집된 추적 수행 정보(trace execution)에 문장번호, 수행되는 문장의 순서에 대한 정보를 삽입함으로서 기준 변수와 관련 있는 문장뿐만 아니라 관련 없는 문장까지도 추출한다. 추출된 프로그램 조각은 정확하면서도 크기가 작아서 프로그램에 대한 효율적인 이해를 지원할 수 있고, 각 문장의 비교 횟수를 줄일 수 있다. 본 논문의 구성은 2장에서 동적 조각화에 대한 기존의 연구에 대해 살펴보고, 3장에서는 동적 조각화에 대한 정의와 프로그램을 실행하는 과정을 추적한 수행 정보에 대해서 설명한다. 4장에서는 문장에서 사용되는 변수의 집합과 선언부에 있는 문장에 대한 변수-변수 관련성을 이용하여 동적 조각을 추출하는 알고리즘을 제시하고, C언어로 작성된 프로그램을 대상으로 제안된 동적 조각 추출 알고리즘에 대한 성능평가를 실시하고, 5장에서는 결론 및 향후 연구방향을 기술한다.

2. 관련 연구

프로그램을 이해하기 용이한 조각으로 분해하는 동적 조각화 기법에 대한 기존의 연구들을 살펴보면 다음과 같다. Gallgher는 흐름 그래프(flow graph)를 이용하여 문장의 변수사이에 제어 의존성과 데이터 의존성을 조사하여 프로그램 조각을 추출하였다. 프로그램의 각 문장들을 기본 블록(basic block)으로 간주하고, 프로그램을 그래프로 변환하여 노드 n에 대해 $def(n)$와 $used(n)$를 계산하여 기준 문장과 관련이 있는 문장들을 파악하였다. 그래프에서 노드는 한 문장을 나타내는 것으로 그래프를 순회할 때 기본 단위가 된다. $def(n)$는 노드 n에서 변화되는 값을 갖게되는 변수의 집합을 가리키고, $used(n)$는 노드 n에서 사용되는 변수의 집합을 가리킨다. 한 문장에 대해서 $def(n)$와 $used(n)$를 계산하여 기준 문장과 관련성을 조사하게 된다. 프로그램을 조각화하는 기준 C는 $<i, v>$로 표기하고, 문장 i에서 변수 v의 값에 직접적으로, 혹은 간접적으

로 영향을 미치는 모든 문장을 나타내므로 하나의 기준 문장에 여러 문장들이 관련되어 있다. 흐름 그래프와 *def(n), used(n)* 함수를 이용하여 동적 프로그램 조각을 추출하는 방법은 제어 의존성을 사용함으로써 비구조적 프로그램에 대해서 부정확한 조각이 추출될 수 있다. 그 이유는 Jump문(goto, continues, break, etc.)을 만나지 않으면 연속적으로 수행이 되지만, Jump문을 만나게되면 원하는 문장으로 Jump하여 수행되므로 중간 부분이 수행되지 않을 수 있기 때문이다. 이와 같은 경우를 고려하여 Jump문이 동적 조각에 포함되어야 하는데 포함이 되지 않을 수 있기 때문에 부정확한 조각을 추출할 수 있다. 이해하려는 프로그램에 대해 흐름 그래프를 작성해야 하므로 소요시간이 많은 편이다[7].

Ball은 자유로운 제어 흐름을 가진 비구조적 프로그램을 대상으로 정확하게 이해하기 위한 방법 즉, 확장된 제어 흐름 그래프(Control Flow Graph: CFG)를 기반으로 하는 프로그램 조각화 알고리즘을 제안하였다. 전체 프로그램과 비교할 때 동일한 입력에 대해 같은 결과를 산출하는 동적 조각화에 대한 정의를 최초로 제안하였다. 조각을 추출하기 위해서는 이해할 프로그램을 제어 흐름 그래프로 변환해야 하는데, 이 과정에서 자유로운 흐름을 나타내는 Jump문(goto, break, continue, label, etc.)들이 프로그램 조각화에 필요한 문장임에도 불구하고 포함되지 않는 경우가 발생한다. 이를 해결하고자 Jump문들에 대해 문맥자유문법(context free grammar)을 사용하여 추상화 구문(abstract syntax)으로 표기한 후, 이를 제어 흐름 그래프에 추가하여 확장된 제어 흐름 그래프를 만들었다. 반드시 필요한 Jump문이 포함되지 않는 문제점은 해결할 수 있었으나, 이해하려는 프로그램에 대해 제어 흐름 그래프와 추상화 문법으로 변환하는 작업에 많은 노력과 시간이 소요되는 경향이 있다. 또한, 제어 흐름을 이용하여 조각화를 수행할 경우 부정확한 프로그램 조각을 추출할 가능성이 있다. 제어 흐름을 갖고 있으면서 문장이 조각화에 포함되지 않을 수도 있기 때문이다[3].

Choi는 goto문이 들어있는 비구조적 프로그램을 이해하기 용이한 프로그램 조각으로 분해하는 정적 프로그램 조각을 추출하는 두 가지 알고리즘을 제안하였다. 첫 번째 방법은 확장된 제어 흐름 그래프(extended control flow graph)와 확장된 프로그램 의존성 그래프(extended program dependence graph)를 이용하여 프로그램 조각을 추출하는 것으로 goto문에 대해 가장 가까운 후진-지배자(postdominator)를 계산해야 한다. 프로그램을 제어 흐름 그래프로 변환한 후 goto문에 대해서는 가장 가까운 후진-지배자를 추가함으로써 확장된 제어 흐름 그래프를 만들었다. 확장된 프로그램 의존성 그래프를 만드는 과정도 동일하다. 확장된 그래프를 이용하는 방법에서의 문제점은 이해하고자 하는 대상 프로그램에 대해 두 개의 그래프를 생성해야 되므로 프로그램 조각을 추출하기 위해 요구되는 선행 작업에 시간과 노력이 많이 소요될 수 있고, goto문이 들어있는 노드에 대한 가장 가까운 후진-지배자를 무조건적으로 추가하게 되면 원래 프로그램을 제어 흐름 그래프로 변환할 때는 존재하지 않은 가지(edges)가 확장된 제어 흐름 그래프에 나타나게되는 경우가 발생할 수 있다. 그렇게되면, 원래 프로그램과 동일한 입력을 가지고 비교해볼 때 결과 값이 다를 수 있어 실행 가능한 프로그램 조각이 추출되지 않는다. 두 번째 방법은 제어 흐름 그래프와 프로그램 의존성 그래프를 이용하여 추출한 프로그램 조각에 goto문에 대한 가장 가까운 후진-지배자와 계승자(successor)를 추가하는 것으로, 첫 번째 방법과 비교해보면, 그래프를 확장하지 않고 가장 가까운 후진-지배자와 계승자를 찾아내는 알고리즘을 이용한다는 차이가 있다. 이 두 번째 방법도 후진-지배자와 계승자를 구하는 과정이 간단하지는 않으므로 조각화하기 위한 선수 작업에 시간 소요가 많다[4].

Korel은 문장을 블록화하는 방법을 사용하여 기준 변수와 관련 있는 문장뿐만 아니라 관련 없는 문장을 추출하는 동적 프로그램 조각화 알고리즘을 제안하였다. Gallgher의 연구와 같이 문장을 블록화하여 비공헌 문장(non-contributing action)을 찾아내는 방법을 사용하였다. 조각을 추출하는 알고리즘은 공헌 문장(contributing action)과 비공헌 문장을 찾아내는 부분으로 나누어지고, 공헌 문장을 찾기 위한 루틴을 먼저 수행한 후, 비공헌 문장을 찾는 루틴을 수행한다. 공헌 문장에 포함된 문장에 대해서 비공헌 문장을 찾는 부분에서는 비교도 하지 않으므로 공헌 문장에 포함된 문장은 비공헌 문장에 포함될 수 없다. 비공헌 문장을 추출하는 과정에서 임의의 문장이 r-entry / r-exit 인 정보를 가지고 있으면 포함시켰고, r-entry / j-exit, j-entry / r-exit, j-entry / j-exit 인 정보를 갖는 문장은 포함시키지 않았다. 평서문인 경우에는 r-entry /

r-exit, goto문, break문, continue문은 r-entry / j-exit, goto문에 따른 레이블은 j-entry / r-exit인 정보를 갖는다. r-entry / r-exit 인 정보는 비공헌 문장 집합에 포함시켰고, 이 문장들은 공헌 문장 집합에 포함될 수 있다. 비공헌 문장에 포함된 문장은 언제라도 공헌 문장에 포함될 수 있다. 프로그램에서 비공헌 문장과 공헌 문장을 적절하게 이용하여 보다 정확한 프로그램 조각을 생성할 수 있었으나, 기준 변수가 많은 경우에는 매번 문장을 비교하는데 소요시간이 많은 경향이 있다.

이상에서 살펴본 동적 프로그램 조각화 기법들이 갖는 문제점은 제어 흐름 그래프를 이용함으로써 부정확한 동적 조각이 추출되는 것과 조각된 프로그램과 원래 프로그램을 비교해 볼 때 동일한 입력에 대해 같은 결과를 산출하지 못하는 것이다. 프로그램을 프로그램 의존성 그래프, 제어 흐름 그래프, 확장된 프로그램 의존성 그래프 등으로 변환해야 하는 작업 과정이 간단하지 않는 단점도 지적할 수 있다. 이러한 문제점들을 해결하고, 정확하면서 수행 가능한 동적 프로그램 조각을 추출하기 위해서는 관련 문장뿐만 아니라 비관련 문장을 추출해야 한다. 각 문장의 비교 횟수를 최소한으로 줄여야 한다.

3. 동적 프로그램 조각화

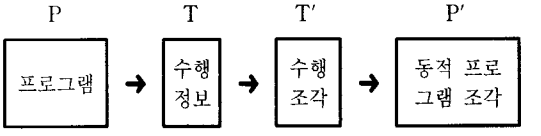
이 장에서는 동적 프로그램 조각화 기법의 정의와 동적 조각을 추출하기 위해서 요구되어지는 수행 정보(trace execution)에 대해 설명한다.

3.1 동적 프로그램 조각화 정의

프로그램 조각화 기법에서 정적 조각화(static slicing)는 기준 문장과 기준 변수를 고려하여 특정 입력이 아닌 가능성이 있는 모든 입력에 대해 정적 조각(static slices)을 추출하고, 동적 조각화(dynamic slicing)는 기준 문장과 기준 변수에 대해 특정한 입력을 고려하여 동적 조각(dynamic slices)을 추출한다. 정적 조각화보다 정확한 프로그램 조각을 추출하는 동적 조각화는 선택된 기준 변수에 대하여 특별한 입력 값이 주어지면 기준 문장과 관련이 있는 프로그램의 일부분을 추출하는 것으로 수행 가능한 조각을 추출한다.

동적 프로그램 조각화는 프로그램을 대상으로 임의의 수행 위치 q에서 기준 변수에 대해 관련 있는 문장을 찾아 프로그램의 일부분을 식별하는 것이다. 전체 프로그램의 일부분인 프로그램 조각은 같은 입력에 대해 결과 값이 동일하게 산출되어야 한다. 프로그램 P에 대한 조각화의 기준은 여러 형태가 있지만, 공통적으로 내포하고 있는 정보는 특정 입력값, 기준 문장, 기준 변수에 대한 것이다. 가장 기본적인 기준 C는 (x, $y^{(q)}$)으로 표기하고, x는 입력 값, y^{q}은 수행 위치 q에서의 기준 문장을 나타낸다. 조각화 기준 C에서 프로그램 P에 대한 동적 조각은 몇 개의 문장을 제거함으로써 만들어지는 문법적으로 정확하고 논리적으로 수행 가능한 프로그램 조각 P′를 추출하는 것이다[7,11].

동적 프로그램 조각을 추출하기 위해서는 선행 작업이 필요하고, 이 선행 작업에서 특정 입력 값에 의해 프로그램(P)을 실행하는 과정을 추적한 수행 정보(T: tracc execution)가 만들어진다. 수행 정보에는 문장 번호, 문장들이 수행되는 순서가 기록되어 있고, 필요에 따라서 블록번호, 변수집합 등의 다양한 정보가 포함되어 질 수 있다. 이러한 과정을 세부적으로 분류하면 네 단계로 구성된다[11]. 첫 번째 단계에서는 원래 프로그램에 대해 실행하는 과정을 추적한 수행 정보(T)를 만드는 것으로 수행 순서를 기록하고, 두 번째 단계에서는 수행 정보를 이용하여 문장과 문장 사이, 변수와 변수 사이에 상호 의존성을 조사하여 적시 적소에 필요한 정보를 삽입한다. 세 번째 단계에서는 기준 변수와 기준 문장에 대해 수행 정보(T)를 동적 조각화하는데, 두 번째 단계에서 만들어진 정보를 활용함으로써 최적의 수행 추적 조각(T′)을 추출할 수 있다. 마지막 단계에서는 조각화된 수행 추적 정보를 원래 프로그램에서의 문장과 사상시켜서 프로그램 조각(P′)으로 나타낸다. 동적 프로그램 조각을 추출하는 일반적인 과정을 다음과 같이 요약할 수 있다.



(그림 1) 프로그램을 수행 조각으로 변형
(Fig. 1) Change program execution slice

【단계 1】 프로그램 P의 수행 추적 정보 T를 생성한다. T의 각 요소는 프로그램의 수행 지점을 부분적

으로 표기하고, 그 지점에서 수행된 프로그램 명령어를 부분적으로 설명한다.

【단계 2】 T에서 각 요소간에 상호 의존성을 조사한다.

【단계 3】 수행 지점에서 특별한 변수의 값에 영향을 미치는 T의 요소들만을 추출하여 T′를정의한다. T′를 수행 추적 조각이라고 명명한다.

【단계 4】 T′를 프로그램 P에 사상시키는 작업을 수행하여 P′를 생성한다.

위에서 언급된 마지막 단계에서는 수행 조각(T′)에 포함되어 있는 문장 번호와 프로그램의 문장 번호를 사상시키는 작업으로 동적 프로그램 조각을 추출하게 된다. 프로그램을 분해하여 부분적으로 동등한 프로그램 조각을 추출하기 위해서는 기준 변수에 영향을 끼치지 않더라도 기준 변수의 값을 변화시킬 수 있는 변수라면, 프로그램 조각에 포함시켜야 한다. 결국 추출된 프로그램 조각은 기준 문장과 관련 있는 문장만을 포함하는 조각화의 특성을 배제하고, 확장된 프로그램 조각을 추출하게 된다. 즉 기준 변수와 직접적인 관련은 없어도 간접적으로 필요한 문장에 대해 프로그램 조각에 포함시켜야 한다.

이러한 문제점을 해결하기 위해서 본 논문에서는 각 문장들에서 사용되는 변수를 좌측 변수(LV : Left Variable)와 우측 변수(RV : Right Variable)로 나누어서 계산한 변수 집합과 선언부에 있는 문장에 대한 변수-변수 관련성(Variable-Variable Relationships)에 대한 정보를 이용하였다.

3.2 프로그램 수행 정보(Program Trace Execution)

프로그램을 조각화하기 위한 선행 작업에서 만들어지는 수행 정보는 동적 조각화 기법에서 매우 중요하다. 수행 정보에는 문장 번호와 문장이 수행되는 순서에 대한 정보가 들어있다. 특정 입력 값이 결정되면, 그 값에 따른 프로그램의 수행되는 순서를 y^q으로 표시하는데, y는 문장의 번호를, q는 문장의 수행되는 순서를 가리킨다. Korel[12]과 Korel & Laski[16]에서는 수행 정보에 기준 변수, 기준 문장, 입력 변수의 값을 삽입하였고, Duesterwald[5]에서는 특정한 입력 값, 기준 변수와 기준 문장을 삽입하였다. 수행 정보에 포함되는 내용을 살펴보면, 여러 형태가 있지만, 대부분 세 가지 요소 즉 입력 값, 기준 문장, 기준 변수에 대한 내용을 포함한다. 이 때 기준 변수와 입력 값이 여러 개인 경우도 고려할 수 있다. 본 논문에서는 수행 정보에 변수 집합 $U(X_n)$과 변수-변수 관련성 $VV(s_i(v))$에 대한 정보를 추가하였다.

4. 변수-변수 관련성을 이용한 동적 프로그램 조각화 알고리즘

프로그램을 대상으로 기준 문장과 기준 변수에 대해 관련 문장을 추출하는 경우에 직접적인 관련 변수는 아니지만, 프로그램 조각에 반드시 필요한 임의의 변수가 들어가지 않는 경우가 발생할 수 있다. 이런 부정확한 조각을 추출하는 경우를 방지하기 위해서 문장에서 사용하는 변수에 대한 정보를 좌측 변수와 우측 변수로 나누어서 저장하였다. 이 장에서는 문장에서 사용되는 변수 집합과 변수-변수 관련성에 대한 정의를 설명하고, 수행 정보를 만들어 가는 과정을 살펴보고, 프로그램 조각을 추출하는 알고리즘에 대해서 설명한다.

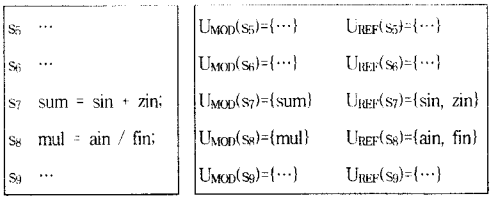
4.1 문장에 대한 변수의 집합

이해하고자 하는 대상 프로그램에 대해서 모든 문장에 대한 변수의 집합을 구한다. 각 문장 s_i는 할당 연산자를 기준으로 좌측에서 사용되는 즉, 값이 변경되는 변수의 집합 $U_{LV}(s_i)$과 우측에서 사용되는 즉, 값이 참조되는 변수의 집합 $U_{RV}(s_i)$으로 나누어 구한다. 변수의 집합을 나누어 계산하는 이유는 기준 변수가 문장에서 참조만 되는 경우와 값이 변경되는 경우를 별도로 고려해야 하기 때문이다. 기준 변수가 변경되는 변수의 집합의 원소인 경우에는 그 문장은 반드시 관련 문장 집합에 포함되어야 하고, 기준 변수가 참조되는 변수의 집합의 원소인 경우에는 관련 문장 집합에 포함되지 않을 수도 있기 때문이다.

【정의 4.1】 문장의 변수 집합

s_i는 프로그램 P에서 문장 번호 순서상 i번째 문장을 나타낸다고 하자. 문장(s_i)에서 사용되는 변수의 집합을 $U(s_i)$로 표기하고, $U(s_i)$는 문장(s_i)에 서 참조되는 변수의 집합 $U_{RV}(s_i)$과 변경되는 변수의 집합 $U_{LV}(s_i)$으로 구성한다.

$$U(s_i) = U_{LV}(s_i) + U_{RV}(s_i)$$



(그림 2) 문장의 변수 집합
(Fig. 2) Variables set of sentences

각 문장에 대한 변수의 집합은 변수-변수 관련성(variable-variable relationships)을 계산할 때 활용할 수 있다. 변수 집합을 계산하는 과정은 사용자나 유지보수자가 직접 해야하는 작업이다. 각 문장에서 사용되는 변수의 집합은 어렵지 않게 계산할 수 있으므로 번거로운 작업은 아니다. (그림 2)는 문장에 대한 변수의 집합을 계산한 결과를 보여준다. 할당 연산자를 기준으로 좌측의 변수의 집합과 우측의 변수의 집합을 조사하여 작성한다.

4.2 변수-변수 관련성

각 문장에 대해 조사한 변수의 집합을 활용하여 변수 선언 부분의 문장에 대해서 변수-변수 관련성(variable-variable relationships)을 조사하여 관련된 문장의 집합을 계산한다. 【정의 4.2】에서 설명된 바와 같이 변수-변수 관련성은 변수 선언 부분의 문장에 대해서만 계산한다. 본 논문에서 대상으로 하는 C언어에서는 프로그램에서 사용하는 변수에 대해서는 변수 선언 부분에 반드시 선언해야 하므로 프로그램에서 사용되는 모든 변수가 변수 선언 부분에 포함되어 있다. 본 논문에서는 모든 변수에 대해 변수-변수 관련성을 계산하였다. 변수-변수 관련성이란 임의의 변수가 문장에 대한 변경되는 변수의 집합에 포함되는 문장의 번호로 구성된 집합을 일컫는다. 변수 선언 부분의 모든 문장들은 변수 선언 부분을 제외한 나머지 문장과 비교를 시작한다. 변수 선언 문장과 나머지 문장을 비교할 때 실질적으로는 선행된 작업으로 작성된 변수의 집합과 비교를 하는데, 변수의 집합 중에서도 변경되는 변수의 집합 $U_{LV}(s_i)$과 비교를 한다. 변수-변수 관련성을 조사하는 작업은 자동으로 이루어진다. 【정의 4.2】는 변수-변수 관련성에 대한 정의를 보여준다.

【정의 4.2】 변수-변수 관련성(Variable-Variable Relationships)

s_i는 프로그램 P에서 문장 번호 순서상 i번째 문장을 나타낸다고 하자. s_{cl}는 프로그램에서 변수 선언 부분의 마지막 문장(cl번째)이라고 하고, s_{ls}은 프로그램에서 마지막 문장(ls번째:last sentence)을 나타낸다고 하자. v는 프로그램에서 사용되는 변수를 나타내고, $VV(s_i(v))$는 변수 v를 선언한 문장(s_i)에 대한 변수-변수 관련성을 나타낸다.

$$VV(s_i(v)) = \{\ s \mid s \in v = U_{MOD}(s_t),\ 0 \le i \le cl,\ cl+1 \le t < ls\ \}$$

4.3 동적 프로그램 조각화 알고리즘

본 논문에서는 조각화 기준 C를 (x, y^q, v)로 표현하는데, x는 기준 변수와 관련이 있는 문장을 추출하기 위해서 필요한 변수의 입력 값이고, y^q은 기준 문장을 가리키며, v는 기준 변수를 가리킨다. 주어진 기준에 따라 이해하기 용이한 동적 프로그램 조각을 추출하기 위해서 해야할 일은 문장에서 사용되는 변수 집합 $U(X_n)$에 대한 정보를 좌측 변수(LV)와 우측 변수(RV)로 나누어 계산하고, 이 변수 집합을 이용하여 문장과 문장사이에 관련성을 조사하는 변수-변수 관련성에 대해 계산하는 것이다. 전체적인 동적 프로그램 조각화 알고리즘은 (그림 3)과 같다. 제안한 알고리즘에 적용시킨 프로그램 수행 정보는 (그림 4)와 같이 나타낼 수 있으며, (a)는 전체 프로그램을 나타내고, (b)는 프로그램을 실행하여 추적한 수행 정보로 나타내는 것으로 문장 번호 및 수행 순서를 보여준다. 제안한 알고리즘의 세부적인 단계는 다음과 같다.

입력 : 조각화 기준 C = (x, y^q, v)
출력 : C에 대한 동적 조각
R_C : a set of related actions

1. Create PTE(program trace execution)
   : Attach NUMBLOCK(numbering for the sentences)
2. Compute $U(X_n)$(the used variables set in the sentence X_n)
3. Compute Variable-Variable relationships for the sentences in the declaration part
   $VV(s_i(v)) = \{\ s \mid s \in v = U_{MOD}(s_t),\ 0 \le i \le cl,\ cl+1 \le t < ls\ \}$
4. Compute $IRS(X^n)$ the immediate related sentence for criterion variable(v)
   : the compared sentence
5. Find the compared sentences as marked

6. $R_C = \emptyset$
7. repeat
8. **Find related actions for the criterion C**
9. until there no exist marked sentence
10. Show a dynamic slice that is constructed from P by removing that belong to NR_C

procedure Find related actions for the C

11. p = 1
12. for all sentences s ∈ $IRS_{LV}(C, X^n)$ do Find and mark Z^t of v
13. IRS에 포함된 문장에 표시(mark)를 하여 표시된 문장을 모두 검색하면 종료한다.
14. while p < t
    if ($U_{LV}(X^p) \cap U(Z^t)$) then $R_C = R_C \cup \{ X^p \}$
    else p = p + 1
15. endwhile
16. compute $R_C = R_C \cup \{ VV_{LV}(Z^t) \}$ for all sentences s ∈ $VV_{LV}(Z^t)$

end Find related actions for the C

(그림 3) 동적 프로그램 조각화 알고리즘
(Fig. 3) Dynamic program slicing algorithm

4.3.1 변수-변수 관련성 추출

동적 조각을 추출하는 첫 번째 작업은 각 문장에 대한 변수 집합과 변수-변수 관련성을 계산하는 것이다. 변수 집합은 각 문장에서 사용되는 변수를 좌측 변수(LV)와 우측 변수(RV)로 나누어 계산한다. 기준 문장과 관련 있는 문장을 찾을 때 좌측 변수만이 영향을 끼치기 때문이다. 그 이유는 좌측 변수가 우측의 식을 이용하여 계산된 값을 할당하므로 값의 변화가 있을 수 있고, 반면에 우측 변수는 자신의 값은 변화하지 않으면서 계산에 참조만 되므로 현재 문장의 변화에 참여하지 않기 때문이다. 따라서 변수-변수 관련성은 선언부에 있는 문장들과 다른 문장의 좌측 변수 사이의 교집합을 계산하는 것이다. 이렇게 비교가 이루어지면, 불필요한 우측 변수에 대한 비교 횟수를 줄일 수 있다.

두 가지의 정보를 작성한 뒤, 기준 문장과 다른 문장사이에 관련성을 파악하기 위해서 비교가 이루어지는데, 이 때 비교되는 문장은 선언부에 있는 자료형을 선언한 문장들이다. 선언부에 있는 문장들만 비교가

```
#include<iostream.h>
#include<conio.h>
void main( )
{
01 int start_num;
02 int end_num;
03 int add_num;
04 int sum;
05 int by3;
06 int temp;
07 sum = 0;
08 by3 = 0;
09 scanf("%d",&start_num);
10 scanf("%d",&end_num);
11 scanf("%d",&add_num);
12 while(start_num<=end_num){
13  sum += start_num;
14  temp = start_num%3;
15  if(temp == 0)
16    by3++;
17  start_num += add_num;
}
18 start_num -= add_num;
19 printf("d%,", sum);
 }
```

(a) 전체 프로그램(P)

```
문장번호/수행순서
01[01]  int start_num;
02[02]  int end_num;
03[03]  int add_num;
04[04]  int sum;
05[05]  int by3;
06[06]  int temp;
07[07]  sum = 0;
08[08]  by3=0;
09[09]  scanf("%d",&start_num);
10[10]  scanf("%d",&end_num);
11[11]  scanf("%d",&add_num);
12[12]  start_num <= end_num;
13[13]  sum += start_num;(+temp)
14[14]  temp = start_num%3;(+by3)
15[15]  temp = 0;
16[16]  by3++;
17[17]  start_num += add_num;
12[18]  start_num <= end_num
13[19]  sum += start_num;(+temp)
14[20]  temp = start_num%3;(+by3)
15[21]  temp == 0;
16[22]  by3++;
17[23]  start_num += add_num
12[24]  start_num <= end_num
13[25]  sum += start_num;(+temp)
14[26]  temp = start_num%3;(+by3)
15[27]  temp = 0;
16[28]  by3++;
17[29]  start_num += add_num;
12[30]  start_num <= end_num
18[31]  start_num -= add_num;
19[32]  printf("d%", sum);
```

(b) 프로그램에 대한 수행 정보(T)

(그림 4) 프로그램에 대한 추적 수행 정보(T)
(Fig. 4) The trace execution for program(T)

되어서 변수-변수 관련성을 계산하면 다른 문장들은 이 변수-변수 관련성만을 이용하여 관련 문장을 추출할 수 있다. 그 이유는 프로그램에서 사용되는 모든 변수들은 선언부에 반드시 선언이 되어야 하므로 프로그램의 모든 변수가 선언부에 있다는 것이다. 그 변수와 관련이 있는 문장을 찾아서 그 문장의 변수 집합과 선언부에 있는 문장의 변수-변수 관련성 집합의 합집합을 계산하면 임의의 문장의 변수-변수 관련성 집합이 추출된다. 변수-변수 관련성을 추출하는 과정은 (그림 5)와 같다. 이 방법으로 조각을 추출하면 모든 문장과 비교하지 않아도 되기 때문에 각 문장의 비교 횟수를 줄일 수 있다. 또한 원래 프로그램과 비교하여 동일한 입력 값에 적용시키면 같은 결과를 산출하고, 정확하면서 최소의 크기의 동적 조각을 추출한다.

VV(1^1) = { LV : 9^9, 17^{17}, 17^{23}, 17^{25}, 18^{31}, 19^{32} }
VV(2^2) = { LV : 10^{10}, 19^{32} }
VV(3^3) = { LV : 11^{11}, 19^{32} }
VV(4^4) = { LV : 7^7, 13^{13}, 13^{19}, 13^{25}, 19^{32} }
VV(5^5) = { LV : 8^8, 16^{16}, 16^{22}, 16^{28}, 19^{32} }
VV(6^6) = { LV : 14^{14}, 14^{20}, 14^{26}, 19^{32} }

(그림 5) 수행정보에 대한 변수-변수 관련성
(Fig. 5) Variables-variables relationships for trace execution

4.3.2. 관련 문장 추출

동적 조각을 추출하는 두 번째 작업은 기준이 정해지면 기준 문장과 직접적으로 관련이 깊은 문장 IRS (Immedated Relationship Sentence)를 추출하는 것이다. 각 문장에서 사용되는 변수에 대한 집합은 계산된 후이므로 기준 변수와 각 문장의 변수 집합을 비교함으로써 계산되어진다. 이 때 교집합이 존재하면 그 문장은 관련이 있는 것이므로 IRS에 포함시키고, 표시(mark)를 한다. 이 표시된 문장들에 대해 관련 문장을 찾을 때까지 반복하게 된다.

IRS에 포함된 문장들은 처음으로 추출된 관련 문장들로 R_C에 포함시키고 다른 관련 문장을 찾기 위해 비교를 계속한다. IRS에 포함된 문장의 좌측 변수와 우측 변수에 대해 모두 비교를 해야하는데, 다른 문장의 좌측 변수사이의 교집합이 존재하는지를 조사하면 관련 있는 문장을 모두 찾을 수 있다. (그림 6)는 입력값 (start_num = 3, end_num = 10, add_num = 3), 기준 변수(sum), 기준 문장(printf("%d", sum))에 대한 변수 집합과 변수-변수 관련성을 이용하여 추출한 수행 조각(T')를 나타내는데, 기준 문장과 관련된 문장의 집합 R_C를 계산하는 과정을 단계별로 보여준다. (그림 7)은 (그림 6)에서 보여준 수행 조각을 프로그램에 사상시킨 결과를 보여준다.

준비 작업, criterion sentence = { 19^{32} }
criterion variable = { sum }
IRS = { 7^7, 13^{13}, 13^{19}, 13^{25} }
( )는 수행 정보에서 문장이 수행된 순서 번호
첫 번째, related variable of criterion variable = { sum(7) }
R_C = { 4^4, 7^7, 13^{13}, 13^{19}, 13^{25} }
두 번째, related variable of criterion variable = { sum(7), start_num(25), end_num(25) }
R_C = { 4^4, 1^1, 2^2, 7^7, 9^9, 13^{13}, 12^{12}, 17^{17}, 13^{19}, 17^{23}, 13^{25}, }
세 번째, related variable of criterion variable = { sum(7), start_num(25), end_num(25), add_num(23) }
R_C = { 4^4, 1^1, 3^3, 2^2, 7^7, 9^9, 11^{11}, 13^{13}, 12^{12}, 17^{17}, 13^{19}, 17^{23}, 13^{25}, 10^{10} }
최종결과(T') = { 4^4, 1^1, 3^3, 2^2, 7^7, 9^9, 11^{11}, 13^{13}, 12^{12}, 17^{17}, 13^{19}, 17^{23}, 13^{25}, 10^{10} }

(그림 6) 기준 C={(3, 10, 3), 19, sum}에 대한 수행 조각(T')
(Fig. 6) Execution slice for criterion C={(3, 10, 3), 19, sum}(T')

```
#include<iostream.h>
#include<conio.h>
void main( )
{
01 int start_num;
02 int end_num;
03 int add_num;
04 int sum;
07 sum = 0;
09 scanf("%d", &start_num);
10 scanf("%d", &end_num);
11 scanf("%d", &add_num);
12 while(start_num<=end_num) {
13  sum += start_num;
17  start_num += add_num;
  }
19 printf("%d", sum);
}
```

(그림 7) 수행 조각에 프로그램을 사상시킨 동적 프로그램 조각(P')
(Fig. 7) Dynamic program slices that mapping execution slice to program(P')

4.4 다른 알고리즘과 비교 분석

이 절에서는 가장 최근에 발표된 블록화 방법[12]과 본 논문에서 제안한 방법을 비교 분석한 결과를 정리한다. 첫째, 한 개의 프로그램에 대해서 기준 변수를 다양하게 제공함으로써 블록화 방법과 본 논문에서 제안한 방법으로 조각화 하는 경우에 문장을 비교한 횟수를 분석해 보고, 둘째, 두 개의 다른 프로그램에 대해서 문장의 비교 횟수를 분석해 본다. 분석 결과를 정리해보면, 블록화 방법인 경우에는 기준변수가 바뀔 때마다 매번 모든 문장을 비교해야 하는 반면, 제안한 방법은 한번 작성된 변수-변수 관련성을 이용하여 모든 문장을 비교하지 않고 손쉽게 관련 문장을 추출할

수 있다는 장점이 있다. 또한, 기준 변수가 여러 개인 경우에는 블록화 방법보다 변수-변수 관련성을 이용하는 방법이 문장의 횟수를 감소시켜줌으로써 프로그램을 이해하는데 더 효율적임을 알 수 있었다. <표 1>은 블록화 방법과 제안한 방법으로 조각을 추출하는 경우에 각 문장들을 비교한 횟수를 사용된 변수별로 나타내었다. <표 2>에서와 같이 블록화 방법은 기준 변수마다 각기 문장의 비교 횟수가 다르게 분석되었으나, 제안한 방법에서는 한번 작성된 변수-변수 관련성을 사용하므로 문장의 비교 횟수가 감소함을 알 수 있었다. 기준 변수가 두 개 또는 세 개인 프로그램에 적용하여 블록화 방법과 비교 분석한 결과는 <표 3>과 같다.

〈표 1〉 문장 비교의 결과
〈Table 1〉 The result of sentence comparison

(단위:횟수)

변수 이름	sum	by3	start_num	add_num	temp	end_num	vv 관련성	합계
블록화 방법	106	113	94	31	107	31		482
제안한 방법	+20	+30	+13	+4	+22	+3	156	248

〈표 2〉 기준 변수에 따른 문장의 비교 결과
〈Table 2〉 The result of sentence comparison according to criterion variables

기준변수 / 방법	기준 변수		
	2개	3개	4개
블록화 방법	219회	313회	420회
제안한 방법	206회	219회	241회
비교횟수 감소율	5%	30%	42%

〈표 3〉 블록화 방법과 변수-변수 관련성 방법의 비교
〈Table 3〉 Comparing blocking method with variable-variable relationships

코드 라인수	사용된 변수	기준 변수	비교 횟수		
			블록화 방법	제안한 방법	감소율
21라인	6개	3개	313회	219회	30%
9라인	4개	2개	47회	39회	17%

5. 결 론

본 논문에서는 프로그램에 대한 정확하고 신속한 이해를 증진시키는 방안으로, 문장에서 사용되는 변수를 좌측 변수와 우측 변수로 분류하여 작성한 변수 집합과 프로그램의 선언부에 있는 문장들에 대한 변수-변수 관련성을 이용하는 동적 프로그램 조각화 알고리즘을 제안하였다. 변수-변수 관련성은 모든 문장에 대해 산출하지 않고, 대부분 자료형을 선언한 선언부에 있는 문장에 대해서만 계산하였다. 문장에서 사용되는 변수를 좌측 변수와 우측 변수로 분류한 이유는 좌측 변수는 값의 변화가 있어 기준 변수에 직접적인 영향을 미치고, 우측 변수는 값의 변화가 없어서 기준 변수에 직접적인 영향을 미치지 못하지만, 간접적인 영향으로 관련성을 가질 수 있기 때문에 별도로 계산되어야 한다.

기준 변수가 6개인 프로그램을 대상으로 제안한 방법을 실험한 결과는 정확하면서 최소 크기의 동적 조각을 추출하였고, 문장의 비교 횟수가 감소됨을 보였다. 또한 기준 변수가 여러 개이고, 기준 변수와 관련된 임의의 변수가 많은 경우에 문장의 비교 횟수가 현저하게 감소됨을 보였다.

현재는 절차적 언어에 국한되어 적용되지만, 앞으로 객체 지향 언어나 병렬 언어에 적용시킨다면, 프로그램을 이해하는데 큰 도움될 것으로 생각한다. 또한, 조각화된 프로그램을 재사용 가능한 부품으로 부품화시키는 방법에 대한 연구를 진행하고 있다.

참 고 문 헌

[1] Agrawal, H. and Horgan, J.R., "Dynamic Program Slicing," *Proceedings of ACM SIGPLAN '90 Conference Programming Language Design and Implementation*, pp.246-256, June 1990.

[2] Agrawal, H., "On Slicing Programs with Jump Statement," *Proceedings of ACM SIGPLAN '94 Conference Programming Language Design and Implementation*, pp.302-313, June 1994.

[3] Ball, T., Horwitz, S., "Slicing Programming with Arbitrary Control Flow," *Proceedings of the First International Workshop Automated and Algo-*

rithmic Debugging, Letures Notes in Computer Science, Springer Verlag, 1993.

[4] Choi, J.D., Ferrante, J., "Static Slicing in the Presence of Goto Statement," *ACM Transactions on Programming Languages and Systems 16(4)*, pp.1097-1113, July 1994.

[5] Duestterwald, E., Gupta, R., and Soffa, M.L., "Rigorous Data Flow Testing through Output Influences," *Proceedings of the Second Irvine Software Symposium: ISS '92*, pp.135-142, March 1992.

[6] Ferrante, J., Ottenstein, K., and Warren, J., "The Program Dependence Graph and its Use in Optimization," *ACM Transaction on Programming Languages and Systems 9(3)*, pp.319-349, July 1987.

[7] Gallgher, K.B., "Using Slicing in Software Maintenance," Ph.D. Thesis, Dept. of Computer Science, University of Maryland dat College Park, Maryland, 1990.

[8] Horowitz, S., Reps, T., and Binkley, D., "Interprocedural Slicing Using Dependence Graphs," *ACM Transactions on Programming Languages and Systems 12(1)*, pp.26-61, January 1990.

[9] Hsieh, C.S., Unger, E.A., "On the Control Structure of a Program Slice," *Journal of System Software 34*, pp.123-126, 1996.

[10] Kamkar, M., Shahmehri, N., and Fritzson, P., "Affect-Chaining and Dependency Oriented Flow Analysis Applied to Queries of Programs," *Proceedings of the ACM Symposium on Personal and Small Computers*, 1988.

[11] Kamkar, M., "An Overview and Comparative Classification of Program Slicing Techniques," Journal of System Software, pp.197-214, 1995.

[12] Korel, B., "Computation of Dynamic Program Slices for Unstructured Programs," *IEEE Transactions On Software Engineering 23(1)*, Jan. 1997.

[13] Lyle, J.R., and Weiser, M., "Automatic Program Bug Location by Program Slicing," *Proceedings of Second IEEE Symposium on Computers and Applications*, pp.877-883, June 1987.

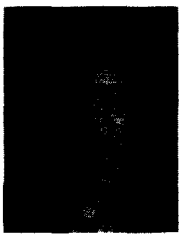
[14] Schneidewind, N.F., "The State of Software Maintenance," *IEEE CSP Tutorial : Software Maintenance and Computers*, 1996.

[15] Weiser, M., "Program Slicing," *IEEE Transactions on Software Engineering 10(4)*, pp.352-357, July 1984.

[16] B. Korel and J. Laski. "Dynamic Slicing of Computer Programs," *Journal of System and Software*, pp.178-198, 1990.

[17] 김태희, 강문설, "소프트웨어 역공학", *정보처리 3(5)*, 한국정보처리학회 pp.31-42, September 1996.

[18] 김태희, 김병기, "프로그램 이해를 지원하는 프로그램 조각화 기법의 비교 분석", *한국정보처리학회 소프트웨어공학연구회지 1(1)*, pp.65-74, September 1998.



김 태 희

thkim@www.dongshinu.ac.kr

1991년 동신대학교 전자계산학과 (공학사)

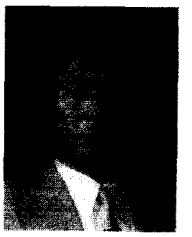
1993년 전남대학교 전산통계학과 (이학석사)

1996년 전남대학교 전산통계학과 (박사수료)

1993년 3월~1997년 2월 동신대학교 컴퓨터학과 강사

1997년 3월~현재 동신대학교 컴퓨터학과 전임강사

관심분야 : 소프트웨어공학(소프트웨어 유지보수, 재공학, 역공학), 객체지향시스템



김 병 기

bgkim@chonnam.chonnam.ac.kr

1978년 전남대학교 수학과(이학사)

1980년 전남대학교 수학과(이학석사)

1996년 전북대학교 수학과(박사수료)

1981년 3월~현재 전남대학교 컴퓨터정보학부 교수

1994년 1월~현재 한국정보처리학회 이사

관심분야 : 소프트웨어공학, 소프트웨어 에이전트, 객체지향시스템