절차적 프로그램으로부터의 객체 추출 방법론

진 윤 숙† · 마 평 수†† · 신 규 상††

요 약

기존 시스템의 유지보수와 구조의 변화 요구를 충족시키기 위해 객체지향 시스템으로의 재공학이 요구된다. 객체지향 시스템으로의 재공학이 요구되는 대상 시스템은 활용 가치는 높지만 설계문서가 없거나 설계 문서가 현재의 코드와 일치하지 않아 변경하기 힘든 절차적 시스템이 될 수 있다. 이런 시스템일 경우 설계 문서를 이용하여 객체를 추출하는 방법은 사용하기 곤란하다. 그러므로 절차적 소스 분석을 통해서 객체를 추출하는 연구가 진행되어 왔다. 본 논문에서는 변수, 자료형, 함수간의 관계를 나타내는 VTFG(Variable-Type-Function Graph)를 이용한 클러스터링 기반의 자동 객체 추출 방법을 제시하였다. VTFG는 객체의 근간이 될 수 있는 변수, 자료형, 함수간의 관계와 그의 중요성을 나타내는 가중치를 표현한다. VTFG의 가중치를 이용하여 서로 연관된 변수, 자료형, 함수를 클러스터링함으로써 객체의 애트리뷰트와 오퍼레이션을 추출한다. 본 논문에서 제시한 객체 추출 방법은 너무 큰 객체를 추출하거나 객체로 추출하지 못하는 기존 연구의 단점을 해소하고 자동 객체 추출을 통해 사용자의 개입을 최소화함으로써 절차적 시스템을 객체지향 시스템으로 재공학하는 과정을 용이하게 한다.

A Method of Object Identification from Procedural Programs

Yun-Sook Jin† · Pyeong-Soo Mah†† · Gyu-Sang Shin††

ABSTRACT

Reengineering to object-oriented system is needed to maintain the system and satisfy requirements of structure change. Target systems which should be reengineered to object-oriented system are difficult to change because these systems have no design document or their design document is inconsistent of source code. Using design document to identifying objects for these systems is improper. There are several researches which identify objects through procedural source code analysis. In this paper, we propose automatic object identification method based on clustering of VTFG(Variable-Type-Function Graph) which represents relations among variables, types, and functions. VTFG includes relations among variables, types, and functions that may be basis of objects, and weights of these relations. By clustering related variables, types, and functions using their weights, our method overcomes limit of existing researches which identify too big objects or objects excluding many functions. The method proposed in this paper minimizes user's interaction through automatic object identification and make it easy to reengineer procedural system to object-oriented system.

1. 서 론

절차적 언어와 방법으로 개발된 기존 시스템은 유지보수하기 힘들고 구조의 변화 요구를 충족시키지 못한다. 그러나 기존 시스템은 오랫동안 사용되었고 응용 영역에 대해 많은 정보를 포함하고 있기 때문에 시스템의 대체나 재개발에 드는 비용은 매우 크다. 이처럼 사업적 가치는 높으나 변경하기 힘든 기존 시스템은 재공학되어야 한다[1].

소프트웨어의 재공학의 한 방법으로 객체지향 개념

† 정 회 원 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 연구원
†† 정 회 원 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 책임연구원
논문접수 : 1998년 10월 7일, 심사완료 : 1999년 9월 10일

들이 사용될 수 있다. 객체지향 개념은 문제 이해, 개발자간의 의사 전달, 기획, 문서 작성, 프로그램과 데이터베이스 설계에 용이하므로 소프트웨어 개발 단계의 분석, 설계, 구현에 이르기까지 사용되고 있다. 객체지향 개발에서의 구조 단위는 추상화(abstraction), 캡슐화(encapsulation), 정보 은닉성(information hiding)과 같은 개념을 가지는 객체이다. 이러한 개념들은 개발자가 실세계에서의 응용 영역 객체 개념으로 일관되게 생각할 수 있게 하므로, 소프트웨어를 쉽게 이해할 수 있게 하고 시스템의 요구가 변화되었을 때 적은 비용으로도 변화를 반영하여 유지보수성을 높일 수 있다. 반면, 기능 중심 시스템은 시스템의 요구가 변화될 경우 대단위의 재구조화가 필요하다. 따라서 소프트웨어의 유지보수성 및 변경용이성을 높이기 위해 기존의 기능중심의 절차적 시스템을 객체지향 시스템으로 변환하려는 소프트웨어 재공학이 많이 연구되고 있다[1].

객체 추출을 자동화하는 것은 매우 어렵다. 재공학이 필요한 절차적 소프트웨어는 적절한 설계문서가 없거나, 있다 하더라도 요구사항이나 명세가 바뀔 때마다 수정을 하지 않아 현재의 코드와 일치하지 않는 경우가 대부분이다[2]. 그러므로 재공학을 위한 객체 추출은 소스 코드 분석을 통해서만 가능하다. 소스코드 분석을 통한 객체 추출은 절차적 언어로 작성된 소스 프로그램을 전체적으로 분석하여 언어에 독립적인 의미를 파악함으로써 실세계의 응용 영역 객체(application-domain object)를 표현하는 컴퓨터 영역 객체(computer-domain object)를 추출, 복원하는 과정이다.

본 논문에서는 C 프로그램으로부터 객체를 자동으로 추출하기 위한 방법을 제시한다. 본 논문에 제시된 방법은 객체 추출을 위하여 변수-자료형-함수 그래프(VTFG : Variable-Type-Function Graph)를 사용한다. VTFG는 세가지 구문요소, 즉 변수, 자료형, 함수간의 관계와 관계의 중요도를 나타내는 가중치를 포함한다. 객체 추출에 필수적인 관계만을 나타낸 초기 VTFG가 구축되고 나면 내부 연결 정도에 기반하여 초기 VTFG를 서브그래프로 분할하는 클러스터링이 수행된다. 클러스터링된 VTFG에 또다른 공유관계를 추가하여 재클러스터링한 후 클러스터내의 변수와 함수들로 클래스 애트리뷰트와 오퍼레이션을 추출한다. 본 논문의 객체 추출 방법은 절차적 소프트웨어를 객체지향 소프트웨어로 변환하는 소프트웨어 재공학 도구인 RESORT (REsearch on object-oriented SOftware Reengineering Technology) 시스템에서 구현되고 있다[13, 14].

본 논문에서 제 2장은 관련 연구를 살펴보고, 제 3장은 RESORT 시스템에 대해 간략히 설명한다. 제 4장은 객체를 추출하기 위해 사용되는 VTFG를 정의하고, 제 5장은 객체의 후보가 되는 클러스터를 찾는 클러스터링 방법을 소개한다. 제 6장에서는 클러스터링을 이용한 자동 객체 추출 방법을 제시하고, 제 7장은 실행 예를 통해 기존 연구와 비교하고, 제 8장은 결론을 포함한다.

2. 관련연구

COBOL이나 FORTRAN을 대상으로 객체지향 시스템으로의 재공학에 대한 연구는 매우 많다[3-6]. Fantechi는 COBOL의 자료 구조를 클래스로 변환하려고 하였으며[3], Subramaniam은 FORTRAN 코드로부터 객체 모델을 추출하여 이를 다시 구현함으로써 객체지향 시스템으로 재공학하려 했다[4]. [5-6]은 FORTRAN을 대상으로 하였지만 전역변수와 인자를 분석함으로써 객체를 추출하였다는 관점에서 본 논문과 유사하다.

Gall et al.의 연구에서는 추상화 수준이 각기 다른 시스템 중간 표현을 사용하여 절차적 코드를 객체지향 구조로 변환한다[7]. 이를 위하여 구조도(structure chart), 자료흐름도(dataflow diagram), 개체관계도(entity-relationship diagram), 객체지향 응용 모델을 생성한다. 시스템의 중간 표현들 중 구조도와 자료흐름도는 소스코드로부터 자동으로 추출되며, 개체관계도와 객체지향 응용 모델은 영역지식을 사용하여 전문가에 의해 생성된다. 이 접근 방식은 시스템을 위한 다양한 설계문서의 복구가 가능하고, 기존 시스템의 이해를 도울 수 있다. 그러나 이것은 개체관계도 생성 단계에서부터 전문가의 참여가 요구되며 따라서 대규모 시스템의 재공학에는 적합하지 않다.

위의 방법과는 달리 소스코드로부터 자동으로 객체를 인식하려는 노력이 있다[8-12]. Livadas와 Johnson의 연구에서는 전역변수를 기반으로 하는 방법과 함수의 형식인자와 반환값의 자료형을 기반으로 하는 방법을 이용하여 자동으로 객체를 추출한 다음, 관계형 데이터베이스 질의와 유사한 질의를 통해 객체를 정제한다[8]. 그러나 객체 정제를 위해 기존 시스템을 이해하고 자동 추출된 객체 후보에서 최종적인 객체를 결정할 수 있도록 하는 도구 지원이 부족하다.

Canfora et al.의 연구에서 제시된 방법은 전역변수와 함수간의 관계를 표현하는 bipartite graph를 사용하며 대화적 과정을 통해 전역변수와 함수를 병합, 분할, 삭제함으로써 연관된 서브그래프를 인식하여 재사용가능한 요소를 찾는다[9]. 그러나 대규모의 시스템일 경우 전역변수와 함수의 수가 많고 그에 대한 병합, 분할, 삭제가 상당히 많이 일어나게 되므로 그때마다의 병합, 분할, 삭제 결정이 사용자에게 부담이 될 수 있다.

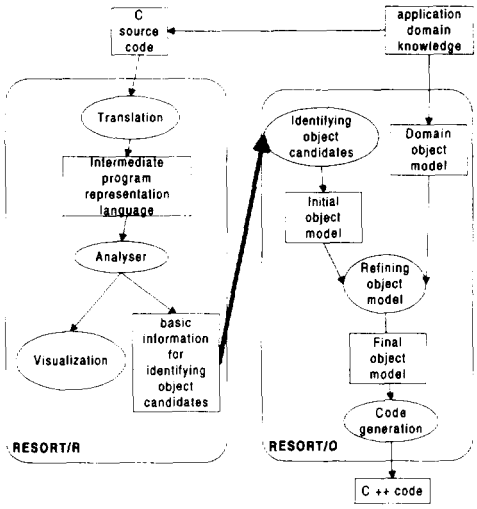
3. RESORT(REsearch on object-oriented SOftware Reengineering Technology) 시스템

RESORT 시스템은 절차 중심으로 개발된 기존의 소프트웨어를 객체지향 소프트웨어로 변환하여 유지보수성과 재사용성을 높여주기 위한 통합적인 소프트웨어 재공학 환경이다[13,14]. 현 단계에서는 C언어로 작성된 기존의 시스템을 재공학하여 객체지향 구조로 변환한 다음 C++언어로 소스코드를 생성하는 데 초점을 두고 있다.

RESORT시스템은 크게 두 부분으로 구성되는데 기존 시스템에서 정보를 추출하는 역공학 도구(RESORT/R)와 추출된 정보를 바탕으로 객체지향 시스템으로 재구축하는 도구(RESORT/O)로 이루어진다.

(그림 1)은 RESORT시스템 전체적인 기능흐름도를 나타낸다. 먼저, C 소스코드를 매개언어 형식인 중간단계의 프로그램 표현(IPR, Internal Program Representation)으로 변환한다. 이 IPR은 프로그래밍 언어 독립적인 특성을 지니므로 본 시스템의 대상을 C 언어 이외의 언어로 쉽게 확장할 수 있다. 변환된 IPR은 시각화와 객체추출을 위한 기본정보의 추출을 위해 분석된다. 분석된 자료는 사용자의 기존 프로그램 이해를 위한 시각화를 위해 사용되고 객체지향 시스템 재구축 도구에서 객체 추출을 위해 사용된다. 지금까지의 작업은 RESORT/R도구에서 이루어진다.

RESORT/O에서는 RESORT/R에서 분석된 기본 정보를 이용하여 객체 후보들을 자동으로 추출하고 이들을 초기 객체모형으로 구성한다. 또 한편으로는 사용자는 응용영역에 대한 지식을 이용하여 응용 영역 객체모형을 생성한다. 그리고 나서 사용자는 편집기와 정제기를 이용하여 응용 영역 객체모델을 바탕으로 초기 객체모형을 최종 객체모형으로 편집 및 정제한다. 이 최종객체모형을 코드 생성 처리를 통해 C++ 코드로 생성된다.

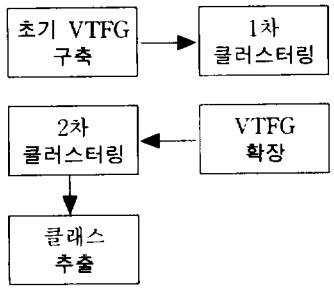


(그림 1) RESORT 시스템의 기능

RESORT시스템은 5만 라인 이하의 중 규모인 C언어로 된 소프트웨어를 C++ 언어로 된 소프트웨어로 변환을 목표로 하고 있고 기존의 절차 중심 소프트웨어 구조를 객체지향 소프트웨어 구조로 변환한다. 뿐만 아니라 기존의 소프트웨어와 새로 생성되는 소프트웨어의 프로그램 구조를 그래픽하게 시각화하여 정보를 제공한다.

본 논문에 언급된 객체 추출 방법은 (그림 1)의 Identifying object candidates단계에서 초기 객체 모델을 생성하기 위해 RESORT/R의 분석 정보를 이용하여 자동으로 객체를 추출하기 위한 것이다. (그림 2)는 객체를 추출하기 위한 Identifying object candidates 단계의 세부적인 단계를 나타낸 것이다. 앞에서 언급하였듯이 VTFG(Variable-Type-Function Graph)와 클러스터링을 이용하여 같은 객체에 속할 가능성이 있는 변수, 자료형, 함수들을 하나의 클러스터로 묶기 위한 것이다. VTFG와 클러스터링에 대한 세부설명은 4장과 5장에 각각 언급되어 있다. 먼저 C 프로그램의 공유기반 분석에 의해 얻어지는 변수, 자료형, 함수간의 관계 중 좀 더 중요한 세부유형의 관계를 적용한 초기 VTFG를 구축한다. 초기 VTFG에 대해 클러스터링하여 내부 연결 정도에 따라 밀접한 관계를 가지는 클러스터로 분할한다. 분할된 VTFG에 초기 VTFG 구축에

적용되지 않은 나머지 관계들을 적용하여 다시 한번 클러스터링하여 객체 후보가 되는 최종적인 클러스터를 생성한다. 마지막으로 클래스 추출 단계에서는 최종 클러스터로부터 클래스의 애트리뷰트와 오퍼레이션을 결정한다.



(그림 2) Identifying object candidates 단계의 세부적 단계

4. 변수-자료형-함수 그래프(Variable-Type-Function Graph : VTFG)

객체는 데이타와 행위를 표현하는 것으로서 절차적 프로그램의 변수, 자료형, 함수로 구성될 수 있다. 절차적 프로그램에서 구조와 기능은 인자를 가진 함수와 전역변수 간의 관계에 의해 표현되므로 함수는 객체에서의 오퍼레이션이 되고 그 함수에 의해 다루어지는 전역변수와 인자는 객체의 애트리뷰트가 될 수 있다. 또한 복합 자료 구조(aggregated data structure)는 연관된 변수의 그룹이므로 복합 자료 구조를 표현하는 자료형은 객체의 내부 상태를 나타낼 수 있고 이 자료형을 다루는 함수는 객체의 상태를 바꾸는 행위를 나타낼 수 있다. 따라서 절차적 프로그램의 변수, 자료형, 함수는 객체지향 프로그램의 객체를 이루는 근간이 된다.

본 논문에서 제시하는 방법은 객체를 추출하기 위해 변수, 자료형, 함수간의 관계를 나타내는 *Variable-Type-Function Graph(VTFG)*를 사용한다. VTFG는 방향성이 없으며 가중치(weight)를 가진 그래프 G = (V, E, w)로 표현되며 V(G)는 정점(vertex) 즉, 변수, 자료형, 또는 함수들의 집합이며, E(G)는 그들 간의 관계를 나타내는 에지(edge)들의 집합이다. 각 에지들의 가중치 w(e)는 두 정점간의 관계 정도를 나타내며 0과 1사이의 값을 가진다.

VTFG G = (V, E, w)

V(G) = { v | v is a variable, a type, or a function }

E(G) = { (v1, v2) | v1 and v2 is in V(G) and there are more than one relation between v1 and v2 }

w(e) = the degree of relation which exists between two vertices connected by e

하나의 에지를 이루는 두 정점 사이에는 세부유형이 다른 여러 개의 관계가 존재할 수 있다. 변수, 자료형, 함수간에는 변수-함수 관계, 자료형-함수 관계, 함수-함수 관계가 존재하며 각 관계는 여러 개의 세부유형을 가진다. 세부적인 내용은 4.2절에서 설명하겠다.

에지의 가중치는 에지를 이루는 두 정점 사이에 존재하는 모든 관계들의 가중치 중 최대치로 정의된다. 각 관계의 가중치는 그 세부유형에 따라 다르게 정의된다.

4.1 공유 기반 분석

VTFG는 프로그램 내의 자료 공유(data sharing)와 자료형 공유(type sharing)를 기반으로 구축된다. 객체지향 프로그래밍에서 객체는 자료 중심이다. 따라서 객체를 추출하기 위해 우선 절차적 프로그램에서 중심이 되는 자료를 추출하고 그 자료에 대한 행위를 나타내는 함수를 추출해야 한다. C 프로그램에서 자료는 변수의 형태로 나타난다. 또한 서로 관련있는 여러 변수들을 하나의 묶음으로 표현하는 자료형 또한 자료의 의미를 가지고 있다. 따라서 객체를 추출하기 위해서는 C 프로그램에서의 변수와 자료형을 고려하는 것이 중요하다. 자료 공유와 자료형 공유를 분석하는 것은 객체의 자료를 다루는 오퍼레이션이 될 수 있는 함수를 추출하기 위한 수단이 된다. 같은 자료와 자료형을 공유하는 두 함수는 공유되는 자료와 자료형과 함께 같은 객체로 포함될 가능성이 높다. 자료 공유는 두 함수가 같은 변수를 공유하여 그 값을 사용 또는 수정하는 경우 일어나며 자료형 공유는 두 함수가 다루는 변수들의 자료형이 같을 경우 일어난다.

두 함수가 같은 전역변수(global variable)를 사용 또는 수정할 경우나 한 함수가 다른 함수를 호출하면서 인자(parameter)를 전달할 경우에 자료 공유가 이루어진다. 따라서 전역변수 분석이나 인자 분석을 통하여 자료 공유를 판명할 수 있다. 자료 공유를 파악하기 위해 전역변수와 인자가 사용되었는지 또는 수정되었

는지를 알아야 한다. 절차적 프로그램에서 자료의 사용과 수정은 객체의 행위를 구현하는 수단이 될 수 있기 때문에 자료의 사용과 수정분석은 객체 추출의 중요한 요인이 된다. 자료 x의 사용은 함수 f가 수행될 동안 x의 값을 이용하는 것을 의미하며, x의 수정이라는 것은 함수 f의 실행 후 x의 값이 정의되거나 변화되었다는 것을 의미한다.

특히 인자의 수정 여부를 고려할 경우 인자 전달 방식에 주의해야 한다. 수정이라는 것은 함수 호출 후에 그 값이 변화되어야 하므로 참조호출(call-by-reference) 방식일 때만 적용된다. C 언어는 인자 전달 방식으로서 값호출(call-by-value) 방식을 사용하는데, 참조호출(call-by-reference) 방식의 효과를 내기 위해서 변수의 주소(address)를 인자로 전달한다. 이때 전달된 형식인자는 포인터 형태를 가지며, 포인터가 가리키는 값이 수정될 경우 x가 수정되었다고 한다. x가 구조체인 경우 호출되는 함수의 실인자로서 x를 이루는 내부 변수의 주소가 전달되어 위의 예처럼 수정된다면 x는 수정된다고 할 수 있다.

인자 분석을 통해 얻어질 수 있는 관계의 가중치는 함수의 호출 수와 연관되어 있다. 함수가 호출될 때마다 전달되는 실인자는 각기 다르다. 그러므로 관계의 가중치가 관계가 성립되는 호출의 비율에 의해 결정된다. 매 호출시마다 실인자와 함수간의 관계가 성립된다면 그 관계의 가중치는 1이 될 것이다. 예를 들어 전체 프로그램에서 함수 func1과 func2의 호출 수가 똑같이 10번이고 변수 x가 함수 func1과 func2의 실인자로 전달하는 경우는 각각 10번, 2번이라고 할 때 x와 func1와의 관계는 x와 func2과의 관계보다 더 밀접하다고 할 수 있다.

이 외에 x의 값을 사용 또는 수정하는 또 다른 형태가 있는데 이를 잠재적 사용(implicit use) 또는 잠재적 수정(implicit modification)이라 한다. 잠재적 사용과 잠재적 수정의 형태는 같은 메모리 주소를 가지는 앨리어스(alias)와 관련이 있다. 예를 들어, x를 가리키는 포인터 p의 경우 x와 *p는 앨리어스이며 *p = y는 실제로 x의 값을 변화시킨다. 포인터가 실제로 무엇을 가리키는지를 결정하는 앨리어스 계산은 매우 복잡하고 어렵다. 본 논문에서는 앨리어스에 속하는 변수들이 사용 또는 수정되었을 경우를 제외한다.

자료형 공유는 두 함수에서 다루는 인자나 전역변수의 자료형이 같을 경우 이루어진다. 본 논문에서 언급하는 자료형은 C 프로그램에서 struct나 배열과 같은 복합 자료 구조의 자료형에 제한한다. 두 함수가 다른 변수를 다룰지라도 그 변수들이 같은 자료형으로 선언되었다면 같은 종류의 객체가 될 수 있고 두 함수는 그 객체의 오퍼레이션이 될 수 있다.

4.2 변수, 자료형, 함수간의 관계

VTFG에서 에지는 변수-함수 관계(variable-function relation), 자료형-함수 관계(type-function relation), 함수-함수 관계(function-function relation) 중 하나를 나타낸다. 두 정점간에 세 관계 중 하나의 관계가 존재하면 두 정점은 같은 객체의 멤버가 될 가능성이 있다. 변수-함수 관계, 자료형-함수 관계, 함수-함수 관계는 각각 R_i (v, f), R_i (t, f), R_i (f, f)로 표현되며, i는 각 관계에 속하는 세부유형을 식별하며 v는 변수, t는 자료형, f는 함수를 가리킨다. 각 관계에 속하는 세부유형의 가중치는 각각 $w(R_i\ (v, f))$, $w(R_i\ (t, f))$, $w(R_i\ (f, f))$로 나타낸다. 관계의 가중치는 0에서 1사이의 값을 가지며 두 정점간의 관계가 어느 정도 밀접한지를 상대적으로 측정하는 수단이 된다. 가중치는 4.1절에서도 언급하였듯이 함수의 호출과 밀접한 관련이 있다. 함수 내부에서 사용 또는 수정되는 전역 변수의 경우 모든 함수 호출시마다 사용 또는 수정되므로 가중치는 1이다. 함수의 인자로 전달되는 변수에 의해 고려되는 관계의 가중치는 그 인자가 전달되는 함수 호출시에만 적용되므로 이 관계는 그 함수의 전체 호출에 대한 비율만큼 줄어들게 된다.

변수-함수 관계는 절차적 프로그램에서의 변수와 함수가 객체지향 프로그래밍에서 자료 추상화(data abstraction) 개념을 구현하고 있기 때문에 중요하다. 이 관계는 변수와 함수 사이에 존재하며 각기 다른 가중치를 가지는 다섯 가지 세부유형이 있다. 이러한 세부유형들은 전역변수 분석과 인자 분석을 통해 얻을 수 있다.

(1) R_1 (v, f)는 전역변수 v가 함수 f의 수행 후 수정되는 것을 의미한다. v는 f의 수행시마다 항상 수정되므로 가중치는1이다.

(2) R_2 (v, f)는 전역변수 v가 함수 f내에서 사용되지만 f의 수행 후 수정되지 않는 것을 의미한다. f의 수행시마다 항상 이런 유형의 관계가 존재하므로 가

중치는1이다.

(3) R_3 (v, f)는 v가 f의 호출시 실인자로 전달되면서 f의 수행 후 수정되는 것을 의미한다. 이 유형의 변수-함수 관계는 인자 분석을 통해 얻어지므로 가중치는 f의 호출 수와 관련이 있다. 가중치는 R_3 (v, f)의 관계가 성립되는 호출 수를 f의 전체 호출 수로 나눈 값이다.

(4) R_4 (v, f)는 v가 f의 호출시 실인자로 전달되면서 f 수행 후 수정되지 않고 f내에서 사용되는 것을 의미한다. 가중치는 R_4 (v, f)의 관계가 성립되는 호출 수를 f의 전체 호출 수로 나눈 값이다.

(5) R_5 (v, f)는 v가 f의 반환값에 의해 수정되는 것을 의미한다. v가 구조체이고 v를 이루는 내부변수가 f의 반환값으로 수정된다면 이 관계가 존재한다. 또한 v가 포인터이고 v가 가리킬 수 있는 값이 f의 반환값으로 수정될 경우에도 이 관계가 존재한다. f의 모든 호출시마다 R_5 (v, f)가 성립되지 않을 수 있으므로 가중치는 R_5 (v, f)의 관계가 성립되는 호출 수를 f의 전체 호출 수로 나눈 값으로 정의된다.

자료형-함수 관계는 함수와 연관된 자료형 사이에 존재하는 관계로서 전역변수 분석, 인자 분석, 자료형 공유 분석에 의해 얻어질 수 있다. 여기에서 언급되는 자료형은 정수, 문자, 부동소수 등과 같은 기본 자료형이 아니라 여러 종류의 자료형을 가지는 변수들의 모임인 복합 자료 구조의 자료형을 말한다. 이런 자료형은 관련이 있는 여러 데이타의 모임이라는 점에서 객체에서의 데이타의 성격을 지닌다. 그러므로 이러한 자료형을 공유하는 함수는 객체에서의 행위를 나타낼 가능성이 있다. VTFG는 자료형-함수 관계를 고려함으로써 객체의 데이타와 행위의 후보들을 나타낼 수 있다. 이 관계에는 여섯 가지 세부유형이 있다.

(1) R_1 (t, f)는 f 수행 중 수정되는 형식인자의 자료형과 f 사이의 관계를 의미한다. 형식인자를 고려하기 때문에 f의 모든 호출에 적용될 수 있으므로 이 관계의 가중치는 1이다.

(2) R_2 (t, f)는 f의 형식인자가 f 내에서 사용되지만 수정되지 않는 경우 f의 형식인자의 자료형과 f 사이의 관계이며 가중치는 1이다.

(3) R_3 (t, f)는 f 수행 중 수정되는 전역변수의 자료형과 f 사이의 관계를 의미하며 가중치는 1이다.

(4) R_4 (t, f)는 f 내에서 수정되지 않지만 사용되는 전역변수의 자료형과 f 사이의 관계를 의미하며 가중치는 1이다.

(5) R_5 (t, f)는 f 의 자료형 즉, f가 반환하는 값의 자료형 t과 f 사이의 관계이다. 이 관계의 가중치는 f의 전체 호출 수에 대해 t와 f사이에 R_5 (t, f)가 존재하는 호출 수의 비율로 정의한다.

(6) R_6 (t, f)는 f가 반환하는 값에 의해 할당되는 변수의 자료형과 f 사이의 관계를 의미하며 가중치는 t와 f사이에 R_5 (t, f)가 존재하는 호출 수를 f의 전체 호출 수로 나눈 값이다. 만약 복합 자료 구조의 자료형을 가진 변수의 내부변수가 f의 반환값으로 할당된다면 그 내부변수의 자료형이 기본 자료형일지라도 f와 그 복합 자료 구조의 자료형 사이에는 이 관계가 존재할 수 있다.

함수-함수 관계는 인자를 통하여 같은 데이타를 공유하는 함수 사이에 존재한다. 그런 함수들은 같은 데이타를 다루므로 데이타 중심의 객체지향 개념으로 볼 때 같은 객체에 속할 가능성이 있다. 또한 절차적 프로그램에서 호출 관계는 공유하는 데이타가 없을 때 같은 기능을 구현하기 위한 수단이 될 수 있다. 그러므로 공유 데이타가 없고 호출 관계에 있는 두 함수는 같은 객체에 속할 수 있다. 함수-함수 관계는 두 가지 세부유형을 가진다.

(1) R_1 (f, f)는 함수 f1과 f2의 수행 결과 같은 변수를 수정할 경우 즉, 함수 f1 수행 후 수정되는 변수와 함수 f2 수행 후 수정되는 변수가 같을 경우를 의미한다. 이 관계는 mod[f]를 이용하여 얻어질 수 있다. 함수에 의해 수정되는 변수가 실인자이므로 특정 호출시에만 관련이 있기 때문에 가중치는 f1과 f2의 전체 호출 수에다 변수를 수정하는 f1호출 수와 f2 호출 수를 나눈 값으로 한다.

(2) R_2 (f, f)는 함수 f1이 다른 함수 f2를 호출할 경우를 의미하며 호출 관계는 상대적으로 우선 순위가 낮으므로 가중치는 0.2이다.

〈표 1〉 관계의 세부유형과 가중치

관계	세부유형	정 의	가중치
변수-함수 관계	R_1 (v, f)	전역변수 v가 f의 수행후 수정된다.	1
	R_2 (v, f)	전역변수 v가 f 내에서 사용된다.	1

〈표 1〉 계속

관계	세부유형	정 의	가중치
	R_3 (v, f)	전역변수가 f의 실인자로 전달되어 f 수행후 수정된다.	R_3 (v, f) 해당 호출수 / f의 호출수
	R_4 (v, f)	전역변수가 f의 실인자로 전달되어 f 수행중 사용만 된다.	R_4 (v, f) 해당 호출수 / f의 호출수
	R_5 (v, f)	전역변수가 f의 반환값에 의해 할당된다.	R_5 (v, f) 해당 호출수 / f의 호출수
자료형-함수 관계	R_1 (t, f)	f 수행후 수정되는 형식인자의 자료형과 f사이에 존재한다.	1
	R_2 (t, f)	f 수행중 사용만 되는 형식인자의 자료형과 f 사이에 존재한다.	1
	R_3 (t, f)	f 수행후 수정되는 전역변수의 자료형과 f 사이에 존재한다.	1
	R_4 (t, f)	f 수행중 사용만 되는 전역변수의 자료형과 f 사이에 존재한다.	1
	R_5 (t, f)	f의 반환값의 자료형과 f 사이에 존재한다.	R_5 (t, f) 해당 호출수 / f의 호출수
	R_6 (t, f)	f의 반환값에 의해 할당되는 변수의 자료형과 f 사이에 존재한다.	R_6 (t, f) 해당 호출수 / f의 호출수
함수-함수 관계	R_1 (f, f)	함수의 출력이 같은 경우에 존재한다.	R_6 (f, f) 해당 호출수 / f1 호출수 + f2 호출수
	R_2 (f, f)	호출관계에 있는 함수 사이에 존재한다.	0.2

<표 1>은 각 관계의 세부유형과 가중치를 정리한 것이다.

5. 클러스터링

클러스터링 단계에서는 VTFG를 클러스터(*cluster*)라고 하는 여러 개의 서브그래프로 분할하며 최종적인 클러스터는 객체의 후보가 된다. 클러스터는 적어도 하나의 정점을 가진다. VTFG G가 클러스터 C_1, C_2, ..., C_n으로 클러스터링된다고 하자. 각 클러스터의 정점들의 합집합 $V(C_1) \cup V(C_2) \cup \ldots \cup V(C_n)$은 원래의 G의 정점들의 집합 $V(G)$과 같고 각 클러스터는 서로 상이하여 $V(C_1) \cap V(C_2) \cap \ldots \cap V(C_n) = \emptyset$이다. 또한 각 클러스터의 에지들의 집합과 클러스터간을 연결하는 에지들의 집합의 합집합은 G의 에지들의 집합 V(E)와 같다.

클러스터링 방법을 설명하기 위해 필요한 용어를 정의한다. 내부 에지(*internal edge*)는 같은 클러스터에 속한 두 정점을 연결한 에지이며 외부 에지(*external edge*)는 서로 다른 클러스터에 속한 정점을 연결하는 에지이다. 예를 들어 C_i와 C_j를 VTFG G의 클러스터라 하자. C_i에 속한 두 정점을 연결하는 에지는 내부 에지이며 C_i에 속한 정점과 C_j에 속한 정점을 연결한 에지는 외부 에지이다.

VTFG G의 클러스터 C_i는 다음과 같이 정의되는 내부연결도(*internal connectivity*: IC)를 가진다.

<정의 1> 클러스터 C_i의 내부연결도 $IC_G(C_i)$는 클러스터 C_i에 포함되는 정점들의 상호 연관 정도를 의미하며 내부에지의 가중치의 합에다 내부에지와 외부에지의 가중치의 합을 나눈 값으로 얻어진다.

$$IC_G(C_i) = \frac{\text{내부에지의가중치의합}}{\text{내부에지와외부에지의가중치의합}}$$

IC의 값은 0과 1사이이며, 하나의 정점을 가진 클러스터의 IC는 0, 외부 에지가 없는 격리된 클러스터의 IC는 1이다. 클러스터의 IC값이 크다는 것은 가중치가 큰 관계를 가진 정점들이 클러스터 내에 밀접하게 모여있다는 것을 의미한다.

<정의 2> 클러스터 C_i와 C_j간의 외부연결도(external connectivity : EC)는 C_i와 연결된 클러스터 C_j와의 연결 정도를 의미하며 다음과 같이 정의한다.

$$\forall j \neq i,\ EC(C_i, C_j) = \frac{C_i\text{와}C_j\text{를연결하는외부에지의가중치의합}}{\text{내부에지와외부에지의가중치의합}}$$

EC는 C_i와 연결된 클러스터 C_j에 대해 각각의 값을 가진다. EC가 크면 C_i와 C_j를 연결된 에지의 가중치가 크므로 C_i와 C_j가 상당히 밀접한 관계를 지니고 있음을 의미한다. 하나의 정점을 가진 클러스터 C_i의 외부에지가 하나의 다른 클러스터 C_j와만 연결되어있다면 C_j에 대한 C_i의 $EC(C_i, C_j)$ 값은 1이다. 만약 C_i와 C_j를 연결하는 외부에지가 없다면 $EC(C_i, C_j)$는 0의 값을 가진다.

VTFG G의 최종 클러스터의 집합 C는 IC와 EC를 이용하여 클러스터들을 반복적으로 병합함으로써 얻을 수 있다. 클러스터링 방법은 다음과 같다.

Step 1 : 각 클러스터 C_i에 대해 IC값보다 EC값이 더 큰 인접 클러스터 중 EC값이 가장 큰 C_j들을 찾

는다. IC ≤ EC는 내부적으로 연결된 정도보다 다른 클러스터와의 관계 정도가 더 큰 것을 의미한다.

Step 2 : 찾아진 클러스터의 쌍에 대해 병합을 하게 되는데 두 가지 방법이 있다. 만약 IC(C_i) ≤ EC(C_i, C_j)이고 IC(C_j) ≤ EC(C_j, C_i)인 C_i와 C_j가 있다면 클러스터 C_i와 C_j를 병합한다.

Step 3 : 그렇지 않은 나머지 C_i와 C_j의 쌍, 즉 IC(C_i) ≤ EC(C_i, C_j)이지만 IC(C_j) > EC(C_j, C_i)인 C_i와 C_j에 대해서는 C_i와 C_j를 연결하는 외부 에지의 가중치(w(C_i, C_j))와 EC(C_i, C_j)를 곱한 값이 가장 큰 C_i와 C_j를 병합한다. 만약 하나의 C_i에 대해 w(C_i, C_j)*EC(C_i, C_j) 값 중 최대값을 가지는 C_j가 하나 이상일 경우는 C_i와 C_j를 병합했을 때의 내부연결도 증가분이 큰 C_j를 선택한다.

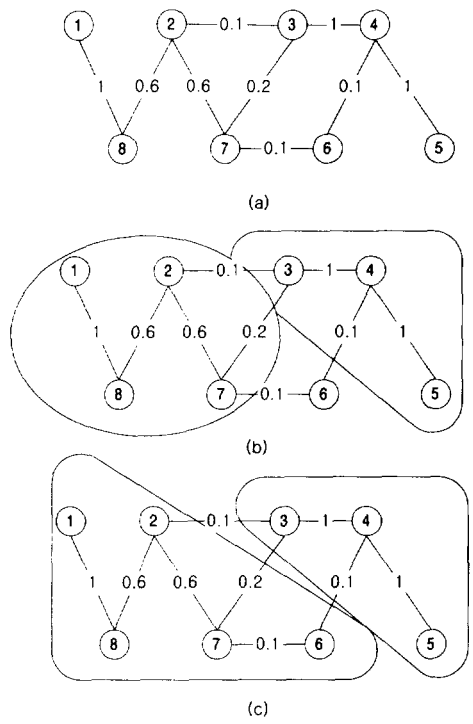
Step 4 : 전체 클러스터의 집합이 변하지 않을 때까지 Step 1에서 Step 3까지의 과정을 반복한다.

다음은 클러스터링 알고리즘을 pseudo 코드로 표현한 것이다. Cluster() 함수는 클러스터 결과값이 더 이상 변하지 않을 때까지 Step1에서 Step 3을 반복한다. ClusterBasic() 함수는 Step 1에서 Step 3까지의 과정을 구현한다. M은 IC(Ci) < EC(Ci, Cj) 이고 IC(Cj) < EC(Cj, Ci)인 클러스터의 쌍 (Ci, Cj)를 찾기 위한 배열로서 이러한 관계를 가지는 Ci, Cj에 대해 M[i][j]는 true 값을 가진다. P는 Step 2에서 같이 병합될 수 있는 클러스터들의 리스트이다.

```
void cluster()
        {
                do {
                        clusterBasic();
                } while(Cn-1 != Cn);        // 클러스터리스트가 변함이 없을 때까지
        }

        void clusterBasic()
        {
                // Step 1 IC(Ci) < EC(Ci, Cj) 인 cluster pair (Ci, Cj)찾기
                for ( i = 0; i < clusterNum; i++ )
                        for ( j = 0; j < clusterNum; j++ )
                                if ( EC (Ci, Cj) != 0 && IC(Ci) <= EC(Ci, Cj) )
                                        M[i][j] = true;

                // Step 2 IC(Ci) < EC(Ci, Cj) 이고 IC(Cj) < EC(Cj, Ci)인 (Ci, Cj) 찾기
                for ( i = 0; i < clusterNum; i++ )
                        for ( j = 0; j < i; j++ )
                                if ( M[i][j] && M[j][i] ) {
                                        Pi = Partition including Ci;
                                        Pj = Partition including Cj;
                                        if ( Pi == NULL && Pj == NULL ) {
                                                create a new Partition P;
                                                add Ci and Cj to P;
                                                add P to partition list;
                                        }
                                        else if ( Pi == NULL && Pj != NULL )
                                                add Ci to Pj;
                                        else if ( Pi != NULL && Pj == NULL )
                                                add Cj to Pi;
                                        else if ( Pi != NULL && Pj != NULL )
                                                merge Pi and Pj;
                                }

                // merge clusters
                for ( i = 0; i < partitionNum; i++ ) {
                        merge all clusters in Pi to one cluster;
                }

                // Step 3 IC(Ci) < EC(Ci, Cj) 이고 IC(Cj) >= EC(Cj, Ci)인 (Ci, Cj) 찾기
                weightSum = -1;
                for ( i = 0; i < clusterNum; i++ )
                        for ( j = 0; j < i; j++ )
                                if ( M[i][j] && !M[j][i] ) {
                                        weightSum_ij = w(edge connecting Ci and Cj)
                                        if weightSum_ij > weightSum;
                                                mergeCluster = (Ci, Cj);
                                }
                merge two cluster in mergeCluster;
        }
```

(그림 3) 클러스터링을 위한 알고리즘

(그림 4)는 간단한 VTFG의 예와 클러스터링의 단계별 결과를 보여준다. (그림 4)의 VTFG에서는 표현의 편의를 위해 정점을 구분하지 않았으며 정점 사이를 연결하는 에지 위의 숫자는 에지의 가중치를 나타낸다. 위의 클러스터링 단계를 k번 반복한 후의 클러스터 집합을 C^k라고 표현하며 C_i^k는 클러스터링 단계를 k번 반복한 후의 i번째 클러스터를 의미한다. 클러스터링을 하기 위해 VTFG에 포함되는 정점을 하나의 클러스터로 초기화하면 초기 클러스터 집합은 C^0 = { {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8} }이다. 각 클러스터 C_i^0에 대해 EC 값을 계산하여 Step 1의 과정을 수행하면 (C_1^0, C_8^0), (C_2^0, C_7^0), (C_2^0, C_8^0), (C_3^0, C_4^0), (C_4^0, C_3^0), (C_4^0, C_5^0), (C_5^0, C_4^0), (C_6^0, C_4^0), (C_6^0, C_7^0), (C_7^0, C_2^0), (C_8^0, C_1^0)이 IC값보다 크면서 최대의 EC값을 가지는 클러스터들의 쌍이다. 여기서 Step 2의 병합 대상에 해당되는 클러스터들은 C_1^0와 C_8^0, C_2^0와 C_7^0, C_3^0과 C_4^0, C_4^0와 C_5^0이며 이에 따라 병합이 일어난다. C_3^0, C_4^0, C_5^0는 하나의 클러스터가 되므로 Step 2의 결과적인 클러스터는 { {1, 8}, {2, 7}, {3, 4, 5}, {6} }로서 4개이다. Step 3의 과정에서 나머지 클러스터들의 쌍 중 C_i와 C_j를 연결하는 외부 에지의 가중치(w(C_i, C_j))와 EC(C_i, C_j)를 곱한 값이 가장 큰 C_i와 C_j는 (C_2^0, C_8^0)이다. 그러므로 이에 대해 클러스터를 병합하게 되면 C_1^0, C_2^0, C_7^0, C_8^0이 하나의 클러스터가 된다. 따라서 클러스터링을 한번 반복한 후의 결과는 (그림 4)의 (b)와 같이 C^1 = {{1, 2, 7, 8}, {3, 4, 5}, {6}}이다. C^1에 속한 클러스터에 대해 다시 Step 1를 수행하면 (C_6^1, C_{1278}^1)과 (C_6^1, C_{345}^1)를 찾을 수 있다. 둘 다 Step 2에 해당되지 않으므로 Step 3을 바로 수행한다. (C_6^1, C_{1278}^1)과 (C_6^1, C_{345}^1)는 외부 에지의 가중치(w(C_i, C_j))와 EC(C_i, C_j)를 곱한 값이 0.05로서 최대값을 가진다. 하지만 C_6^1과 C_{1278}^1의 병합 결과 IC의 증가분이 더 크므로 (그림 4)의 (c)와 같이 C_6^1은 C_{1278}^1과 병합되며 C^2 = {{1, 2, 6, 7, 8}, {3, 4, 5}}가 된다. C^2에 대한 클러스터링은 IC값보다 EC 값이 큰 클러스터의 쌍이 없으므로 그만둔다. 최종 클러스터 집합의 결과는 C^2로서 (그림 4)의 (c)에 나타나 있듯이 두 개의 클러스터 {1, 2, 6, 7, 8}과 {3, 4, 5}이 추출된다.



(그림 4) VTFG의 예와 클러스터링 결과 (a) VTFG (b) C^1 (c) C^2

본 논문에서 제시한 클러스터링 방법에서는 서로 다른 클러스터를 약하게 연결하는 관계가 무시되므로 너무 큰 객체를 추출하는 기존 방법의 한계를 해결한다. 기존의 방법에서는 그래프의 strongly connected component를 찾고 그를 구성하는 변수와 루틴을 객체의 애트리뷰트와 오퍼레이션으로 결정한다. 이러한 방법은 어떤 에지에 의해 연결된 모든 정점이 같은 strongly connected component에 포함되므로 너무 큰 객체가 추출될 수 있다.

6. 자동 객체 추출

이 장에서는 (그림 2)의 자동으로 객체를 추출하기 위한 방법을 설명하기 위해 실제 예를 들어 설명한다. 먼저 C 프로그램의 공유 기반 분석에 의해 초기 VTFG를 구축한다. 초기 VTFG에서는 변수, 자료형, 함수간의 관계 중 좀 더 중요한 세부유형의 관계를 적용한다. 초기 VTFG에 대해 클러스터링하여 내부 연결 정도에 따라 밀접한 관계를 가지는 클러스터로 분할한

다. 분할된 VTFG에 초기 VTFG 구축에 적용되지 않은 나머지 관계들을 적용하여 다시 한번 클러스터링하여 객체 후보가 되는 최종적인 클러스터를 생성한다. 마지막으로 클래스 추출 단계에서는 최종 클러스터로부터 클래스의 애트리뷰트와 오퍼레이션을 결정한다.

6.1 VTFG 구축 및 클러스터링

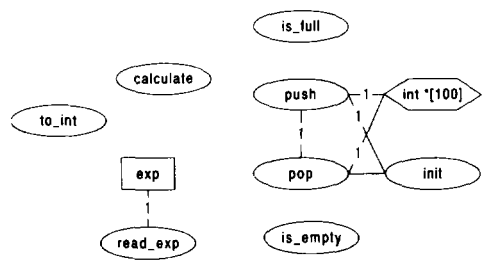
초기 VTFG는 4.2절에서 언급한 세가지 관계 중 공유되는 변수가 수정되는 관계만을 적용한 VTFG이다. 변수를 수정하는 함수는 객체지향 개념에서 객체의 상태를 변화시키는 행위를 의미하므로 이 관계를 우선적으로 적용한다. 변수-함수 관계에서는 전역변수가 함수 내에서 수정되는 R_1 (v, f), 함수의 인자로 전달된 전역변수가 수정되는 R_3 (v, f), 전역변수가 함수의 반환값으로 할당되는 R_5 (v, f)이 적용된다. 자료형-함수 관계에서는 함수내에서 수정되는 인자의 자료형에서 나타나는 R_1 (t, f), 함수의 인자로 전달되어 수정되는 전역변수의 자료형에서 나타나는 R_3 (t, f), 함수의 반환값으로 전달되는 자료형에서 나타나는 R_5 (t, f)이 적용되고 두 함수 사이에 공유되는 인자가 수정될 경우의 R_1 (f, f)인 함수-함수 관계만을 적용한다.

```
char exp[100];
main() {
   read_exp();
   printf(%s = %d\n,exp, calculate());
}

void read_exp() { scanf(expression : %s\n, exp); }
int calculate() {
   int data[100], top, i = 0;
   init(&top);
   while ( i < strlen(exp) )
      switch (exp[i++] ) {
        case + : push(&data, &top, pop(&data, &top)+pop
                 (&data, &top)); break;
        case - : push(&data, &top, pop(&data, &top)-pop
                 (&data, &top)); break;
        case * : push(&data, &top, pop(&data, &top)*pop
                 (&data, &top)); break;
        case / : push(&data, &top, pop(&data, &top)/pop
                 (&data, &top)); break;
        default : push(&data, &top, to_int(&i)); break;
      )
    return pop(&data, &top);
}
int to_int(i)
int *i; {
   int n = 0;
   (*i)--;
   while ( exp[*i] <= 0 || exp[*i] >= 9 ) { n = n *10 +
exp[(*i)++]; }
   return n;
}

void init(t) int *t; { *t = 0; }
int is_empty(t) int t; { if t == 0 return 1; else return 0; }
int is_full(t) int t; { if t == 100 return 1; else return 0; }
void push(st, t, val) int *st[100], t, val; { if !is_full(t)
     *st[t++] = val; }
int pop(st, t) int *st[100], t; { if !is_empty(t) return
     *st[--t]; }
```

(그림 5) 산술식을 계산하는 C프로그램의 예

(그림 5)는 산술식을 계산하는 C 프로그램의 예이다. 실제로 이와 같이 아주 간단한 프로그램은 전역변수를 사용해도 상관없으나 기존 방법의 단점을 보여주기 위해 의도적으로 인자 전달을 통해 구현하였다. 그러나 이런 방법은 일반적인 C 프로그램에서 보편적으로 쓰이는 방법이다.

(그림 6)는 (그림 5)의 C 프로그램에 위에 언급한 관계만을 적용한 VTFG이다. 타원으로 표현된 정점은 함수를 나타내고 사각형은 전역변수를, 육각형은 자료형을 각각 나타낸다. 두 정점을 연결하는 에지 위의 숫자는 에지의 가중치를 나타낸다. 함수 push와 함수 pop사이에는 R_1 (f, f) 관계가 성립되고 각 함수의 모든 호출에 대해 같은 관계가 성립되므로 가중치는 1이다. 또한 push와 pop은 자료형 int *[100], 즉 크기가 100인 정수 배열을 가리키는 포인터형과 R_1 (t, f) 관계가 있다. 함수 read_exp는 전역변수 exp를 수정하므로 read_exp와 exp사이에는 R_1 (v, f)관계가 존재한다. 함수 to_int, calculate, is_empty, is full은 이 단계에서 적용되는 관계를 가지지 않으므로 다른 에지와 연결되어 있지 않다.

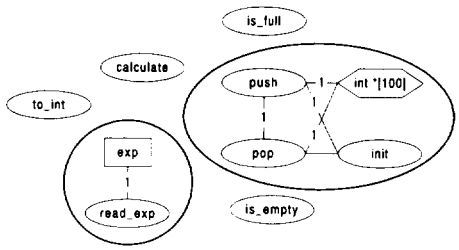


(그림 6) (그림 5)의 C프로그램에 대한 초기 VTFG

우선순위가 높은 관계들만을 적용한 초기 VTFG에 대

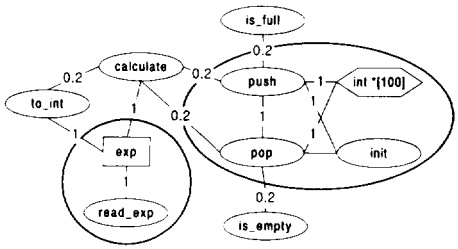
해 제5장에서 서술된 클러스터링을 행한다. 클러스터링을 통해 얻어진 클러스터는 객체가 될 가능성이 있는 객체 후보들이다.

(그림 7)는 (그림 6)의 VTFG를 클러스터링한 결과이다. (그림 6)의 VTFG의 에지들이 독립적으로 완전히 두 그룹으로 분리되어 있어서 클러스터링 결과를 정점들의 집합으로 표현하면 {{int*[100], init, push, pop}, {exp, read_exp}, {to_init}, {calculate}, {is_empty}, {is_full}}이다.



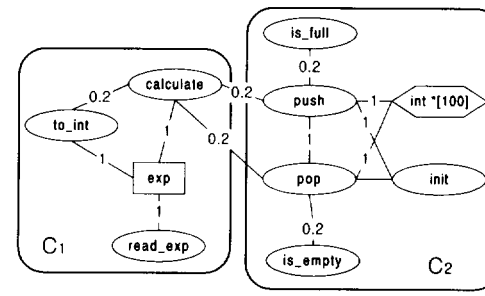
(그림 7) (그림 6)의 VTFG를 클러스터링한 결과

클러스터링이 이루어진 VTFG에 초기 VTFG에서 적용되지 않는 관계들을 적용시킨다. 이때 추가되는 관계는 클러스터의 정점의 갯수가 1인 클러스터를 구성하는 정점들에 대해서만 적용시킨다. 변수, 자료형, 함수간의 관계를 분리하여 적용함으로써 객체를 생성하는데 의미적인 관계를 우선적으로 고려하게 되어 좀 더 정확한 객체를 추출할 수 있다. (그림 8)는 (그림 7)의 확장된 VTFG이다. calculate와 to_int는 exp를 사용하므로 가중치가 1인 R_2 (v, f) 관계가 존재한다. 호출 관계를 이루는 calculate와 to_int, push, pop 사이에는 각각 가중치가 0.2인 R_2 (f, f)가 존재한다. 마찬가지로 is_full과 push, is_empty와 pop 사이에도 같은 관계가 존재한다.



(그림 8) (그림 7)의 클러스터링된VTFG의 추가

확장된 VTFG에 대해 다시 클러스터링을 하여 최종적인 객체의 후보를 얻는다. 1차 클러스터링의 결과인 {int*[100], init, push, pop}, {exp, read_exp}, {to_init}, {calculate}, {is_empty}, {is_full}를 각 클러스터로 초기화하고 2차 클러스터링을 수행한다. (그림 9)은 최종 클러스터링의 결과를 보여준다. calculate와 exp를 연결하는 에지의 가중치와 to_int와 exp를 연결하는 에지의 가중치가 큰 반면 calculate와 push, pop를 각각 연결하는 에지들의 가중치는 작기 때문에 calculate와 to_int는 {exp, read_exp} 클러스터와 병합된다. 함수 is_empty와 함수 is_full은 EC의 값이 각각 pop과 push에 대해 가장 크므로 {int*[100], init, push, pop} 클러스터에 병합된다.



(그림 9) (그림 5)의 프로그램의 최종 클러스터

제 2장에서 언급한 [8-12]의 결과에 비해 본 논문의 객체 추출 방법은 (그림 5)의 프로그램에 있는 모든 함수를 객체에 포함시킨다. 이는 변수-함수 관계와 자료형-함수 관계외에 함수-함수 관계를 포함하는 VTFG를 사용함으로써 인자 전달에 의한 자료 공유를 반영할 수 있었고 가중치를 가지는 VTFG를 두 번 클러스터링함으로써 각 관계의 우선 순위를 반영하여 하나의 객체로 포함될 가능성이 높은 변수, 자료형, 함수를 그룹화였기 때문이다.

6.2 클래스 추출

클러스터링으로 얻어진 클러스터로부터 애트리뷰트와 오퍼레이션을 결정하여 클래스를 추출해야 한다. 클러스터는 하나의 클래스로 매핑된다. 클러스터내의 변수들과 자료형을 이루는 내부 변수들은 객체의 애트리뷰트가 되고 함수는 클래스의 오퍼레이션이 된다. (그림 9)에서 클러스터 C_1에 해당되는 부분은 산술식을 계산하는 클래스를 구현하며 전역변수 exp는 이 클래

스의 애트리뷰트가 될 수 있고 read_exp, calculate, to_int는 클래스의 오퍼레이션이 될 수 있다. 클러스터 C_2는 스택의 여러 행위를 구현하는 함수 init, is_empty, is_full, push, pop들로 구성되어 있다.

클래스의 지역화를 위해 추출된 애트리뷰트와 오퍼레이션의 가시성(visibility)를 결정해야 한다. 클래스의 멤버의 가시성은 클래스 내부에서만 사용되는 멤버는 private로, 클래스 외부에서도 사용되는 멤버는 public으로 결정한다. (그림 9)의 push와 pop은 다른 클래스에 속하는 calculate의 엑세스되므로 스택을 구현하는 클래스의 public 오퍼레이션이 된다.

7. 실행 예를 통한 기존 방법과의 비교

<표 2>는 (그림 5)의 프로그램을 기존 방법을 이용하여 객체 추출한 결과와 본 논문에서 제시한 객체 추출 방법의 결과를 비교한 것이다. (그림 5)의 프로그램은 전역 변수 1개와 함수 9개를 포함하고 있고 AT&T의 Indian Hill C Style로 작성되었다고 가정했을 경우 총 라인수는 72라인이다. 각 함수별로 살펴보면 main()와 init()은 각각 5라인, read_exp()는 4라인, calculate()는 19라인, to_int()는 10라인, is_empty()와 is_full()은 각각 8라인, push()와 pop()은 각각 6라인씩이다. 객체 추출은 main()함수를 제외하므로 (표2)에서는 main()함수를 제외하고 계산되었다.

객체 추출 방법을 비교하기 위해 (그림 5)의 프로그램을 객체 추출한 이상적인 경우를 수동으로 객체 추출하는 경우로 가정하였다. 이는 매우 주관적일 수도 있으나 아주 간단한 예제이므로 그 결과가 직관적이라고 판단된다.

자동 추출율은 (그림 5)의 C 프로그램의 라인 수와 변수/함수의 수, 그리고 이상적인 결과의 클래스 수에 따라 결정한다. 객체로 추출되지 못한 변수 또는 함수를 [7]에서는 remainder라고 한다. 자동추출율은 전체 부분에서 이러한 remainder가 차지하는 비율을 뺀 비율이 될 수 있다. 즉, (그림 5)의 프로그램에서 객체로 추출되기 전의 전체 라인수에 대해 객체로 추출된 라인 수의 퍼센트와 전체 변수/함수 수에 대해 객체로 추출된 변수/함수 수의 퍼센트로 결정한다. 또한 자동 추출율은 가장 이상적인 객체 추출 결과와 비교하여 산정할 수 있다. 이상적인 결과의 클래스 수에 대해 각 방법의 결과로 추출된 클래스 수의 퍼센트로 결정할 수 있다.

객체 추출을 위한 대부분의 기존 방법은 전역변수와 함수간의 관계를 바탕으로 하는 전역변수 기반 방법을 사용한다[8-12]. 일반적으로 C프로그래밍에서 전역변수 사용은 프로그램의 결합도(coupling)를 높이게 되므로 피하는 경향이 있다. 대신 인자 전달을 통해 변수의 값을 사용하고 모듈간의 결합도를 줄이게 된다. 이럴 경우 기존의 전역변수 기반의 객체 추출 방법은 많은 부분의 함수를 객체에 포함시키지 못한다. (그림 5)의 프로그램에서 전역변수는 exp뿐이다. [8-12]에서 사용하는 전역변수 기반의 객체 추출 방법은 전역변수 exp와 함수 read_exp, calculate, to_int를 객체로 추출한다. 그러나 함수 init, is_empty, is_full, push, pop은 지역변수 data, top과 함께 stack을 구현하는 객체로서 추출될 수 있다. stack을 구현하는 함수들이 전역변수를 쓰지 않기 때문에 기존의 연구에서는 이들을 객체로 추출할 수 없다.

또한 자료형 기반의 객체 추출 방법을 사용하는 [8, 11, 12]에서도 모든 함수에 대해 객체로 추출하지 못한다. 형식인자의 자료형이나 반환값의 자료형을 고려하므로 (그림 5)의 예에서는 함수 push와 pop의 형식인자인 st의 자료형에 의해 함수 push와 pop은 하나의

〈표 2〉 기존 연구와의 객체 추출 결과 비교표

		Ideal	전역변수 기반	자료형 기반	본 논문의 방법
클래스		{exp, read_exp, calculate, to_int} {init, push, pop, is_full, is_empty}	{exp, read_exp, calculate, to_int}	{push, pop}	{exp, read_exp, calculate, to_int} {init, push, pop, is_full, is_empty}
Remainder		-	init, push, pop, is_full, is_empty	exp, read_exp, calculate, to_int, init, is_full, is_empty	
자동추출율	line수	100%(67/67)	50.7%(34/67)	17.9%(12/67)	100%(67/67)
	변수/함수 수	100%(9/9)	44.4%(4/9)	22.2%(2/9)	100%(9/9)
	클래스수	(2)	50%(1/2)	50%(1/2)	100%(2/2)

객체로 추출할 것이다. 그러나 여전히 여러 개의 나머지 함수들은 객체로 추출되지 못한다.

위의 두 경우는 C 프로그램에서 많이 사용되는 인자 전달에 의한 함수간의 자료 공유를 고려하지 않았기 때문에 일어난다.

그러나 본 논문에서 제시된 VTFG의 함수-함수 관계는 함수 push과 함수 pop, 함수 init과 push, init과 pop간의 관계와 함수 calculate와 함수 to_int, calculate와 push, calculate와 pop, push와 함수 is_full, pop과 함수 is_empty간의 관계를 추출하게 함으로써 이들이 객체를 구성할 수 있게 한다. 또한 이 세 가지 관계를 하나의 VTFG에 한꺼번에 표현함으로써 사용자에 의한 관계 조합을 피할 수 있다. VTFG에는 기존의 연구에서도 고려하는 전역변수-함수 관계와 자료형-함수 관계이외에도 함수와 함수간의 관계를 고려한다. 함수와 함수간의 관계는 인자 전달에 의한 두 함수 간의 자료 공유를 반영할 수 있다. 또한 이 세 가지 관계를 하나의 VTFG에 한꺼번에 표현함으로써 사용자에 의한 관계 조합을 피할 수 있다.

8. 결 론

본 논문에서는 변수, 자료형, 함수간의 관계를 나타내는 VTFG를 이용한 자동 객체 추출 방법을 제시하였다. 본 논문의 객체 추출 방법은 사용자의 개입을 최소화하여 자동으로 객체를 추출함으로써 절차적 프로그램을 객체지향 프로그램으로 재공학하는 과정을 용이하게 한다.

VTFG의 클러스터링을 이용한 객체 추출 방법은 기존 연구에 비해 몇가지 장점을 지닌다.

우선, VTFG는 기존의 연구[8-12]에 비해 자료 공유와 자료형 공유를 기반으로 다양한 관계를 고려한다. 본 논문의 VTFG는 변수-함수 관계와 자료형-함수 관계 뿐만 아니라 인자 전달에 의한 두 함수 간의 자료 공유를 반영하는 함수-함수 관계를 고려한다.

둘째, 객체 추출에 고려되는 관계를 순차적으로 분리하여 클러스터링함으로써 관계간의 우선 순위를 적용하였다. 변수의 값은 객체의 상태를 나타내고 변수를 수정하는 함수는 객체지향 개념에서 객체의 상태를 변화시키는 행위를 의미하므로 객체의 은닉성(encapsulation)을 위해서 변수 수정을 먼저 고려하였다. 변수, 자료형, 함수간의 관계의 중요도를 가중치뿐만 아니라 적용되는 순서로서 차별화함으로써 좀 더 정확한 객체의 후보를 찾을 수 있다.

셋째, 클러스터링 방법은 VTFG의 가중치를 이용하여 내부적으로 연결 정도가 큰 그룹을 찾는다. VTFG에서 하나의 객체에 포함될 수 있는 변수, 자료형, 함수간의 관계는 상대적으로 큰 가중치를 가진다. 클러스터링은 서로 다른 클러스터를 약하게 연결하는 관계를 무시하므로 너무 큰 객체를 추출하는 기존 방법의 한계를 해결한다.

그러나 좀 더 의미적으로 정확한 객체를 추출하기 위해서는 몇가지 추가적인 연구가 요구된다. 예를 들면, 서로 다른 클래스의 행위가 하나의 함수에 양분되어 구현된 경우 함수를 분할(slicing)하여 각기 해당 클래스의 오퍼레이션으로 추출하여야 한다. 또한 4.1절에서 언급한 잠재적 사용 또는 잠재적 수정을 일으키는 앨리어스를 분석하여 좀 더 정확한 변수-함수 관계를 얻어야 할 것이다.

참 고 문 헌

[1] Ivar Jacobson, "Re-engineering of old systems to an object-oriented architecture," OOPSLA'91, pp.340-350, 1991

[2] Rene R. Klosch, "Reverse Engineering : Why and How to Reverse Engineer Software," Proc. of the California Software Symposium, pp. 92-99, April, 1996

[3] A. fantechi , P. Nesi, and E. Somma, "Object-Oriented Analysis of COBOL," EUROMICRO Proc. Software Maintenance and Reengineering, pp.157-164, 1997

[4] Gokul V. Subramaniam and Eric J. Byrne, "Deriving an Object Model from Legacy Fortran Code," Proc. Int'l Conf. Software Maintenance, pp.3-12, November, 1996

[5] Doris L. Carver, "Reverse Engineering Procedural Code for Object Recovery," Proc. Software Engineering and Knowledge Engineering, pp.442-449, 1996

[6] B. L. Achee and Doris L. Carver, "Creating Object-

Oriented Designs Form Legacy FORTRAN Code," Journal of Systems Software, Vol.39, pp.179-194, 1997

[7] Harald Gall, Rene Klosch, and Roland Mittermeir, "Object-Oriented Re-Architecturing," Proc. European Software Engineering Conference, pp.499-519, September, 1995

[8] Panos E. Livadas and Theodore Johnson, "A New Approach to Finding Objects in Programs," Journal of Software Maintenance : Research and Practice, Vol. 6, pp. 249-260, September,1994

[9] G. Canfora, A. Cimitile, and M. Munro, "An Improved Algorithm for Identifying Object in Code," Software-Practice and Experience, Vol.26(1), pp.25-48, January, 1996

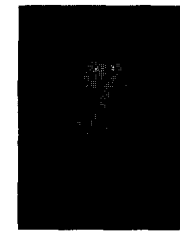
[10] C. L. Ong and W. T. Tsai, "Class and object extraction from imperative code," Journal of Object-Oriented Programming, pp.58-68, March-April, 1993

[11] Sying-Syang Liu and Norman Wilde, "Identifying Objects in a Conventional Language : An Example of Data Design Recovery," Proc. Int'l Conf. Software Maintenance, pp.266-271, November, 1990

[12] Roger M. Ogando, Stephen S. Yau, Sying S. Liu, and Norman Wilde, "An Object Finder for Program Structure Understanding in Software Maintenance," Journal of Software Maintenance: Research and Practice, Vol. 6, pp. 261-283, 1994

[13] 박성희, 신규상, 마평수, 이헌기, 윤석진, 진윤숙, "절차적 소프트웨어를 객체지향 소프트웨어로 변환하는 재공학 도구의 설계", 한국정보처리학회 추계 논문집, pp. 733-737, 10월, 1997

[14] Yunsook Jin, Pyeong S. Mah, and Gyusang Shin, "Deriving an Object Model from Procedural Programs," TOOLS Pacific, pp. 233-242, November, 1997



진 윤 숙

e-mail : ysjin@etri.re.kr

1993년 2월 경북대학교 전자계산학과(학사)

1993년 2월~1993년 12월 삼성데이타시스템 근무

1994년 3월~1996년 2월 경북대학교 컴퓨터과학과(석사)

1995년 12월~현재 한국전자통신연구원 컴퓨터소프트웨어기술연구소 연구원

관심분야 : 객체지향 기술, 소프트웨어 재공학, 멀티미디어 기술 등

마 평 수

e-mail : pmah@etri.re.kr

1985년 서울대학교 식물병리학과 (학사)

1992년 City University of New York, USA 전산학과 (석사)

1995년 Wright State University, USA 전산학과 (박사)

1885년~1989년 시스템공학연구소 연구원

1989년~1990년 ㈜태양금속 정보산업연구실 대리

1996년~현재 한국전자통신연구원 컴퓨터소프트웨어기술연구소 책임연구원

관심분야 : 멀티미디어 저장서버, 멀티미디어 검색, 소프트웨어 재공학 등

신 규 상

e-mail : gsshin@etri.re.kr

1981년 성균관대학교 통계학과 (학사)

1983년 서울대학교 계산통계학과 (석사)

1983년~현재 한국전자통신연구원 컴퓨터소프트웨어기술연구소 책임연구원

관심분야 : 소프트웨어공학, 객체지향 기술, 멀티미디어 기술 등