두 프로그램의 동일성에 대한 연구

박성옥○ 이문근
전북대학교 컴퓨터과학과
{sopark, mklee}@cs.chonbuk.ac.kr

A Study on the Equivalence in two Different Programs

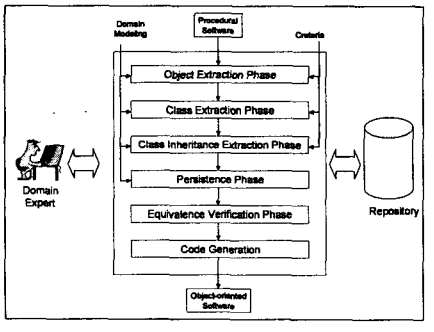
Sung-Og Park○ Mun-Kun Lee
Dept. of Computer Science, Chonbuk University

요 약

절차지향 프로그램으로부터 객체지향 프로그램으로 재공학 과정은 여러 단계를 거친다. 재공학은 객체 추출, 클래스 추출, 상속성 추출, 지속성 추출, 코드생성, 테스트등의 과정을 거친다. 변환 과정을 정당화하기 위해 테스트 단계에서 두 프로그램의 의미적 동일성이 검증되어야 한다. 이러한 검증과정이 없이는 변환이 무의미하다. 본 논문에서는 두 프로그램의 의미적 동일성의 의미와 검증에 대한 방법론을 제안한다.

1. 서론

소프트웨어는 제작된 후 지속적인 유지·보수가 필요하다. 이러한 흐름에 맞추어 시스템을 재공학할 때 유지·보수 및 재사용 측면에서 기존의 방식들보다 유리한 객체지향 파라다임을 적용하여 기존의 시스템으로 재개발한다면, 소프트웨어 생산성을 향상시킬 수 있고, 소프트웨어의 유지·보수 비용을 절감할 수 있으며, 시스템에 새로운 요구를 수용할 수 있게 되는 등 많은 장점을 가지게 된다[1].



<그림 1> 절차중심 SW의 객체중심 SW로의 재공학 절차 흐름도

일반적으로 재공학 방법은 <그림 1>과 같은 절차로 진행되는 것이 바람직하다. 객체 추출 단계에서는 전역 변수, 사용자 자료형 함수와 파라미터를 기반으로 관련성 정도를 기준으로 클러스터링한다. 클래스 추출 단계에서는 클러스터링된 객체 후보들의 공통적인 특성을 추출하여 클래스를 추출한다. 클래스 추출 단계의 결과는 클래스들간에 대등한 평면 관계를 이루고 있다. 상속관계 추출 단계는 전 단계에서 추출된 평면화된 클래스들의 공통적 특성을 추출하여 *part-of* 관계나 *is-a* 관계를 추출하여 계층 구조를 만든다. 지속성 단계에서는 절차 지향 프로그램에는 존재하지 않는 객체 지향 특성들과 동적인 부분을 추가한다. 이러한 것들은 클래스의 생성자, 소멸자, 동적 메모리 할당/해제 등이 있다. 동일성 검증 단계에서는 생성된 객체지향 프로그램이 정상적으로 작동하는지와 원래의 절차지향 프로그램과 변환된 객체지향 프로그램이 의미상으로 동등한지에 대한 검사를 한다. 코드 생성 단계에서는 전단계에서 생성된 뼈대 코드(Skeleton code)를 기반으로 실행 가능한 객체 지향 프로그램을 생성한다.

본 논문에서는 객체지향으로 재공학 과정 중 다섯 번째 단계인 동일성 검증에 대하여 기술한다. 패러다임이 다른 두 프로그램의 동일성의 의미와 구체적인 방법에 대하여 논의한다. 동일성 비교는 비교 기준에 대한 절대적/상대적 측면의 비교와 비교 범위에 대한 시스템/행위적 측면의 비교가 있다. 객체지향 프로그램으로 재공학 과정의 패러다임이 상이한 두 프로그램의 비교는 상대적이며 행위적인 동일성을 의미한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대하여 기술, 3장에서는 패러다임이 상이한 두 프로그램의 동일성의 의미에 대한 기술, 4장에서는 결론 및 향후 연구과제에 대하여 기술하였다.

2. 관련연구

행위적 측면에서의 동치 관계는 강한(Strong) bisimulation, 약한(Weak) bisimulation과 관찰적 합동(Oberservational

본 연구는 한국 과학재단 특정기초연구(과제번호 1999-2-2-303-003-3) 지원으로 수행되었음

Congruence)이 있다.

2.1 강한 bisimulation

강한 bisimulation은 일반적으로 강한 동일성(Strong Equivalence)과 제한적인 동일성(Constrainted Equivalence)으로 불리기도 한다[3]. 이것은 이진 관계로서, 어떤 프로세스(혹은 프로그램)의 집합 S에 대하여 원소 P와 Q가 존재할 때, 모든 액션 a에 대하여, P에서 a에 의해 P'으로 전이될 때 Q에서도 a에 의하여 Q'으로 전이되고 이렇게 전이된 P'과 Q'가 역시 집합 S에 포함되며, 역으로도 성립할 때를 가리켜 강한 bisimulation이라고 한다[3].

2.2 약한 bisimulation

약한 bisimulation과 동일한 의미로 쓰이는 말들로는 관찰적 동일성(Observational Equivalence), 문맥상의 동일성(Contextual Equivalence), 연산적 동일성(Operational Equivalence)과 bisimulation이 있다. 약한 bisimulation은 두 개의 구(Phrase)가 문맥상으로 동일하다는 것은 첫 번째 구의 완전한 프로그램이 두 번째 구에 의하여 프로그램 실행의 결과의 영향없이 대치 가능하다는 것이다.

일반적으로 두 프로그램이 동일하지 않다는 것은 증명하기 쉽다. 이 것은 두 프로그램중에서 수식(Expression)을 비교하여 서로 다른 결과를 발생한다는 것을 조사함으로써 이루어진다. 반대로 두 개의 프로그램이 동일하다는 것을 증명한다는 것은 어렵다. 이 것은 프로그램에 함수/프로시져등이 존재할 경우 각각은 지역 상태를 가질 수 있기 때문에 두 개의동일성 비교는 미묘한 문제를 발생시킬 수 있기 때문이다.

결론적으로 보면, 약한 bisimulation은 프로그램의 시작과 결과만을 비교하지 중간 과정에서 어떤 action이 발생하는지에 대해서는 관여하지 않는다.

2.3 관찰적 합동

프로그램 P와Q가 동일 또는 관찰적 합동이라는 것은, 모든 액션 a에 대하여, P에서 a에 의하여 P'으로 전이될 때, Q에서도 a에 의해 Q'으로 전이하고 이때의 P'과 Q'이 관찰적 동일(Observation Equivalence) 하면, 또한 역으로의 관계인 Q에서 a에 의하여 Q'으로 전이될 때, P에서도 a에 의해 P'으로 전이하고 이때의 P'과 Q'이 관찰적 동일하면 이를 가리켜 관찰적 합동이라 한다[3].

3. 두 프로그램의 동일성 및 적용 방법론

3.1 동일성의 정의

두 프로그램은 아래와 같은 두 가지 관점에서 동일하다고 할 수 있다. 두 프로그램의 동일성을 비교하기 위한 기준(C)은 다음과 같다.

$$C=\langle E,\ R,\ S_0,\ S_t,\ E_{sequence}\rangle$$

이곳에서 E는 환경, R은 사용자 요구(Requirement), S_0는 초기 상태, S_t는 최종 상태, $E_{sequence}$는 이벤트의 순서를 의미한다. R에서 사용되는 것은 프로그램의 행위적인 부분이 아닌 프로그램의 안전성, 속도 등과 같은 것들이다.

두 프로그램의 동일성은 아래와 같은 3가지 튜플로 정의할 수 있다.

$$\langle C^1,\ C^2, D_{C^1C^2}\rangle$$

이곳에서, C^1은 첫 번째 프로그램의 기준, C^2는 두 번째 프로그램에 대한 기준, $D_{C^1C^2}$는 두 프로그램이 실행할 경우 각 성질에 대한 수치화된 차이점이다. $D_{C^1C^2}$는 요구 사항에 명시된 것으로 예를들면 응답시간, 실행시간 등이 있다.

정의 1) 절대적/상대적 측면의 동일성

1) 절대적 측면의 동일성

완전한 의미의 동일성으로서 두 프로그램이 동일한 환경(Environment), 동일한 요구조건, 동일한 상태로부터 동일한 최종 상태에 동일한 이벤트의 순서대로 발생하는 것이다. 이러한 경우 두 프로그램은 완전한 동등 관계에 있다.

$$C^1\equiv C^2,\ D_{C^1C^2}=0$$

2) 상대적 측면의 동일성

상대적 의미의 동일성으로서 두 프로그램이 동일한 환경, 동일한 상태로부터 동일한 최종상태를 요구 조건 이내에 발생할 경우이다. 이러한 경우는 한 개의 패러다임(또는 연산)이 다른 패러다임(또는 연산)을 포함하는 경우이다. 이를 수식으로 표현하면 다음과 같다 :

$$E^1=E^2, R^1\neq R^2, S_0^1=S_0^2,\ S_t^1=S_t^2,$$
$$(E^1_{sequence}\neq E^2_{sequence})\vee(E^1_{sequence}=E^2_{sequence})$$

또는

$$C^1\equiv C^2,\ D_{C^1C^2}\rangle \text{한계값}$$

상대적 측면에서 바라본다면 두 프로그램의 동일성 비교는 내부 상태(동작)의 비교가 아닌 두 프로그램이 동일한 인터페이스를 제공(동일한 순서의 입력에 대한 동일한 순서의 출력)하는가에 달려 있다.

만일 프로그램 A가 프로그램 B를 상대적 측면에서 포함한다면 A는 B와 동일하다고 말할 수 있으나 반대의 경우는 성립하지 않는다.

절대적 측면의 동일성의 경우는 동일한 언어로 작성되어 동일한 순서를 가진 두 프로그램일 경우이다. 가령 두 프로그램이 변수 이름만 가지고 동일한 순서를 가지고 있을 때이다. 상대적 측면의 동일성의 경우는 서로 다른 개발 환경을 가지고 서로 다른 알고리즘을 적용한 프로그램일 경우이다. 서로 다른 개발환경과 알고리즘을 가지고 있기 때문에 행위적 측면에서는 동일할지 몰라도 두 프로그램 내부는 상이하다.

정의 2) 시스템/행위적 측면의 동일성

1) 시스템 측면의 동일성

시스템은 프로그램으로만 구성된 것은 아니다. 시스템은 프

로그램만이 아니라 요구사항 등과 같은 여러 가지 요소로 구성된다. 이러한 요소중 환경과 프로그램의 요구사항은 동일성 검증에 중요한 역할을 한다. 시스템 측면의 동일성은 이러한 전체적인 요소들에 대한 것들을 포함한다.

2) 행위적 측면의 동일성

원시 프로그램이 가지고 있는 행위적 측면에 대한 동일성을 의미한다.

동일성에 대한 정의를 보면 동일성도 상대적이라는 것을 알 수 있다. 요구 조건을 매우 엄격하게 할 경우 두 프로그램은 서로 다를 것이다. 그러나 요구 조건을 매우 느슨하게 한다면 두 프로그램이 동일할 수 있다. 가령 프로그램 A와 B가 존재할 경우 입력 자료 D_1에 대하여 P_1 작업을 A가 5초, B가 11초에 처리하고 동일한 결과를 발생한다고 가정하자. 만일 P_1 작업을 20초 이내에 처리하도록 명세되어 있다면 A와 B는 동일한 의미의 프로그램이라 말할 수 있다. 그러나 P_1 작업을 10초 이내에 처리하도록 명세되어 있다면 두 프로그램은 동일하지 않다.

정의 3) 서로 다른 패러다임을 가진 언어의 동일성

서로 다른 패러다임을 가진 언어의 동일성은 언어가 지원하는 특성을 비교하여 동일성을 결정하여야 한다. 두 언어가 서로 동일한 특성을 가진다면 어떠한 경우에도 동일한 기능을 할 수 있는 프로그램을 작성할 수 있다는 의미이다. 만일 두 언어가 서로 다른 특성을 가진다면 사용자의 요구명세에서 필요로 하는 특성에 따라 프로그램의 동일성이 결정된다.

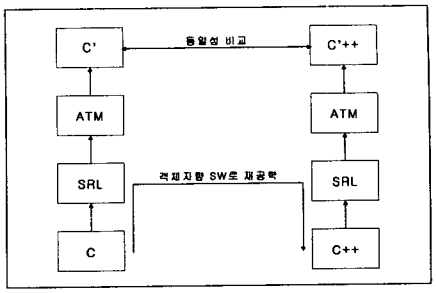
언어의 패러다임 측면에서 관찰할 때 객체 지향 패러다임은 절차지향 패러다임을 포함한다. 따라서 객체지향 언어로 작성된 프로그램이 사용자의 요구사항(Requirement)과 프로그램의 행위적 측면을 만족시킨다면 상대적/시스템 측면에서 프로그램은 포함관계가 성립되어 동일하다고 할 수 있다.

논문에서 적용하는 절차지향 프로그램으로부터 객체지향 프로그램으로 재공학하는 과정은 서로 다른 언어의 파라다임을 가지고 있기 때문에 상대적 측면의 동일성이 적용된다. 또한 사용자의 요구명세의 적용 유무에 따라 시스템/행위적 측면의 동일성이 결정된다.

3.2 동일성 비교 방법론

패러다임이 다른 두 개의 프로그램을 비교하는 방법으로 사용하는 동일성은 상대적이며 행위적인 동일성을 기준으로 한다. 실시간 시스템과 같은 안전성이 중시되는 시스템은 시간이 매우 중요한 기능을 담당하기 때문에 상대적인 시스템 동일성을 비교하여야 하지만 일반 시스템은 시간적 측면보다는 행위적 측면을 강조하기 때문이다. 따라서 비교 대상은 원시 코드만으로 한정한다.

두 프로그램의 동일성을 검증하기 위한 구조는 <그림 2>와 같다.



<그림 2> 동일성 비교

동일성 비교 순서는 다음과 같다 :

1) SRL로 변형 : 프로그램의 정보를 표현할 수 있는 언어 중립적 메타 언어인 SRL(System Representation Language)로 변경하고,

2) ATM[5]으로 변형 : SRL만으로는 프로그램의 상태를 알 수 없다. 따라서 프로그램을 정형적으로 명세하는 ATM으로 변경하고,

3) C'/C'++로 변형 : 두 프로그램을 비교할 수 있는 형태로 변경한다.

4. 결론 및 향후 연구과제

객체지향 프로그램으로부터 객체지향 프로그램으로 재공학 과정은 여러 단계를 거친다. 재공학은 객체 추출, 클래스 추출, 상속성 추출, 지속성 추출, 코드생성, 테스트등의 과정을 거친다. 변환 과정을 정당화하기 위해 테스트 단계에서 두 프로그램의 의미적 동일성이 검증되어야 한다. 이러한 검증과정이 없이는 변환이 무의미하다. 본 논문에서는 두 프로그램의 의미적 동일성의 의미와 검증에 대한 방법론을 제안하였다.

두 프로그램의 동일성은 비교 기준에 따라 차이가 난다. 논문에서는 절대적/상대적 측면의 동일성과 시스템/행위적 측면의 동일성에 대하여 기술하였다.

향후 연구과제로는 제시한 방법론에 대한 구체적인 적용과이를 구현하는 것이다.

[참고문헌]

[1] Andrew M. Pitts, "Lecture Notes on Semantics of Programming Languages," 1998

[2] Andrew M. Pitts, "Operational-Based Theories of Program Equivalence," pp. 241-298

[3] Milner, "Communication and Concurrency," Prentice Hall, 1991

[4] Wuu Yang, susan Horwita and Thomas Reps, "Detecting Program Components with Equivalent Behaviors," 1989

[5] 노경주, 박지연, 이문근, "순환 공학을 위한 정형 기법 : 추상시간 기계," 2000년 정형기법 워크샵, pp. 137-148, 2000

[6] 박성욱, 최정란, 이문근, "절차지향 SW를 객체지향 SW로 재공학하기 위한 클래스와 상속성 추출에 관한 연구," 한국소프트웨어공학 학술대회 논문집, 제2권, 제1호, pp. 51-60, 2000