소프트웨어 신뢰성 향상을 위한 정형기법
A Survey on Formal Verification Methods

주운기*, 이충호*, 김중배**
* 선문대학교 산업공학과, e-mail : ugjoo@sunmoon.ac.kr
** 한국전자통신연구원 컴퓨터소프트웨어연구소, e-mail : jjkim@etri.re.kr

Abstract

This paper considers formal verification methods for enhancing software reliability. The formal method verifies that a software is correctly implemented according to its specification by using a mathematical formalism. This paper presents a partial survey on the formal methods and discusses possible applications for the improved software implementation. Finally, some topics are remarked as further studies.

1.서론

소프트웨어 개발에 대한 구조적 분석기법, 객체지향 설계 기법, 엔티티릴레이션 모형 등 많은 기법이 제안되어왔으나, 이들 방법은 주로 도형을 이용해 시스템을 표현하며, 일치된 이해를 위해 각종 설계 지침, 지원도구들을 수반하고 있다. 하지만 이들 중 어떤 것도 소프트웨어의 검증 혹은 증명이 요구될 경우 특별한 해법을 제공해 주지 못한다. 이에 대한 대안으로 현재 많은 연구가 진행 중인 것이 정형 기법 (formal methods)이다. 정형 기법이란 정형 명세(formal specification)의 수학적 표기법을 사용하여 소프트웨어 혹은 하드웨어가 포함되는 시스템의 특성 및 기능을 서술하며 증명 가능한 사항에 대해 증명하는 기법이다. 따라서 정형 방법을 이용하면, 명세에 대한 수학적 완전성(completeness), 정확성(correctness), 일관성(consistency)을 증명할 수 있으므로, 기존의 타 방법에 비해 보다 명확한 명세가 되도록 할 수 있다([1], [3], [6], [8], [10], [13]).

소프트웨어의 검증(verification)을 위해서는 구현물이 명세를 잘 따르고있는 지의 확인이 필요하다. 반면에, 소프트웨어의 시험(testing)[3]은 에러를 발견할 목적으로 test case를 생성하여 프로그램을 시행하는 것으로, 이를 위해서는 좋은 시험사례(test case)가 필요하다. 전통적인 소프트웨어 시험은 소프트웨어를 구현 완료 후 별도의 시험담당 팀에서 개발이 완료된 소프트웨어에 대한 오류를 찾는 형태로 진행되고, 정적인(static) 시험에 한정적이다. 그러나, 시스템이 복잡해짐에 따라 검토해야 할 항목 수가 매우 많아지고, 오류가 적은 소프트웨어를 적은 비용으로 빨리 개발하고자 하는 목적을 위해서는 조사 및 walkthrough 방식의 전통적인 비정형(informal) 방법 대신 정형화기법(formal method) 및 이를 위한 자동화된 도구를 사용할 필요가 있다. 이러한 정형화 검증 기법을 활용하면 소프트웨어 개발 주기 초기부터 효과적인 검증 작업을 수행 할 수 있으므로 소프트웨어의 신뢰성을 확보할 수 있다. 본 논문에서는 소프트웨어 신뢰성 향상을 위해 사용할 수 있는 정형 검증 기법과 도구를 살펴보고, 이에 대한 활용방안 및 추후 과제를 고찰하였다.

2.정형방법

정형 방법을 위해서는 수학이론에 근거한 모델링(modeling), 명세언어(specification language) 및 검증기법(verification methodology)을 이용한다. 모델링 과정에서는 시스템 기능에 대한 데이터의 불변성질(data invariant property), 상태(state) 그리고 연산(operation)들을 정의한다. 이들 정의된 시스템 기능은 명세언어를 이용하여 표현하는데,명세언어는 순차적(sequential) 시스템에 적합한 Oxford 대학의 Z, VDM, Larch와 동시적(concurrent) 시스템에 적합한 CSP(Communicating Sequence Process), CCS, 영국 Manchester 대학의 VDM (Vienna Development Method), CCITT표준인 SDL (Specification and Description Language), LOTOS(Language of Temporal Ordering Specification) 및 temporal logic 등이 있다([4], [6], [8], [13]). 그리고, 명세의 검증을 위한 방법으로는 모델체킹(model checking) 방법과 정리증명(theorem proving)이 있다.

2.1 정형 명세 방법

명세 기술 방법으로는 잘 정의된 구문과 구문 규칙을 가지고 있고, 명세의 의미를 형식적인 방법으로 정의하는 정형명세기법(FDT : Formal Description Technique), 잘 정의된 구문을 가지고 있지만 형식적으로 정의되지 않은 의미를 가지는 준형식(semi-formal) 기술기법, 그리고, 구문과 구문 규칙 및 명세 의미가 비 형식적인 비형식 기술법이 있다. 정형명세는 어떤 시스템이 반드시 만족 시켜야 할 조건들을 입, 출력 자료 혹은 시스템의 행위를 대상으로 집합 이론, 대수 논리 혹은 술어 논리에 바탕을 둔 수학적 기호들로 표기하여, 애매성, 모호성이 제거된 표기가 가능하고, 군더더기 정보와 구현 종속적인 정보를 초기에 제거함으로서 시스템의 핵심을 이해하기 쉽게한다. 그리고, 제시된 명제에 관한 증명이 가능하므로, 요구 명세의 분석 단계에서부터 미리 심각한 오류들을 제거할 수 있고, 이 오류들이 설계 및 구현으로 파급되는 것을 막을 수 있기 때문에 보다 효과적으로 고 품질의 소프트웨

어 시스템을 개발할 수 있다.

명세언어는 용도에 따라 기술형(descriptive) 언어와 구축형(constructive)언어로 구분할 수 있는데, 기술형 언어는 모델의 성질을 기술하기 위해 주로 구현 초기 단계(분석 및 설계)에서 이용하는 로직 언어로, LTL(Linear Time temporal Logic) 및 CTL(Computation Tree Logic) 등이 있다. 구축형 언어는 모델 정의를 직접적으로 보조하기 위한 수단으로 사용되는 것으로, 유사-구현(quasi-implementation) 형태의 언어이므로 구현 직전 단계에서 주로 이용한다. 그러나, 이러한 성격 때문에 구현 방식을 미리 고려한 명세가 될 수 있다는 문제가 있다. 구축형 명세 언어는 FSM(Finite State Machine)과 같은 Communication Finite-state I/O Automata, SDL(Specification and Description Language) 및 ESTELLE과 같은 Extended communicating I/O Automata, CCS(Calculus of Communication System) 및 LOTOS(Language Of Temporal Ordering Specification)와 같은 Labeled transition communicating I/O Automata, Petri-nets, Z 언어 등이 이에 속한다. 이와 같이 기술형 언어 및 구축형 언어의 주요 용도가 서로 다르므로, 소프트웨어 개발을 위해서는 두 유형의 언어를 공용할 필요가 있고 두 가지 기능을 모두 가진 언어에 대한 개발도 필요한 상태이다. 전반적으로 명세언어를 비교하면, ESTELLE는 기능적 취급범위(functional coverage) 및 구조화(structuring)에서 우수하며, SDL은 분석력(analyzability), 구조화(structuring), 재 사용성(reusability) 및 확장성(extensibility) 면에서 우수하다. 또한 LOTOS는 형식기술(formal definition), 표현력(expressive power) 및 간결성(conciseness) 특성에서 우수하다. 그리고, 이들을 자동화하는 지원도구에 대한 것으로는 Estelle과 SDL을 위한 도구가 많이 연구되고 있다.

정형명세기법은 표현 형태에 따라 텍스트 위주의 표현 방법인 프로그래밍 언어 모델(Estelle, SDL, LOTOS 등), 프로토콜 설계자 시각 위주의 표현방법인 상태천이 모델(FSM, CFSM, 페트리네트 등), 그리고 이들 두 기법을 혼합한 혼합형 모델의 3가지로도 구분할 수 있다. 명세화를 위해 사용되는 FDT는 서비스, 프로토콜, 인터페이스 및 참조 모델에 대하여 명확하고 간결한 규칙을 제공해야 하고, 규격의 모든 조건을 만족하는지를 검증할 수 있어야 하며, 구현이 표준 규격에 일치하는지 그리고 표준들이 서로 간에 일관성을 유지하는지를 결정할 수 있어야 하며, 구현을 지원할 수 있는 도구를 이용할 수 있는 것이 좋다.

2.2 정형 검증

명세의 검증을 위한 방법으로는 모델체킹(model checking) 방법과 정리증명(theorem proving)이 있다([2], [4], [5], [9]). 모델 체킹 방법은 시험열(test sequence)을 생성하여 시스템의 모든 전이(transition)가 바르게 진행되는 지의 확인을 통해 검증하는 방법이다. 따라서, 자동화 도구의 이용이 용이하다는 장점이 있지만, 상태 및 천이의 수가 많은 복잡한 시스템의 경우에는 검토해야 할 천이의 수가 매우 많으므로 이 방법을 적용한 검증에 현실적인 제약이 있다. 반면에, 정리 증명 방법은 스펙의 문장 구성(syntactical) 및 로직에서 수학적인 형태의 정리(axiom)나 추론규칙(inference rule)을 추출하여 연역적(deduction)인 방법으로 증명하는 검증방식이다. 이 방법을 위해서는 증명 대상의 시스템에 대한 충분한 이해와 수학적인 지식이 필요하고 자동화된 검증이 어렵다는 단점이 있지만, 문제의 복잡성에 관계없이 무한 상태 문제에도 적용 가능하다는 장점을 가진다. 이와 같이 모델 체킹 및 정리 증명을 도와주는 자동화된 검증도구를 각각 모델체커(model checker) 및 정리증명기(theorem prover)라 한다.

(1) 시험열 생성 방안

모델 체킹 방법을 통한 검증을 위해서는 각 상태(state)에서 각각의 시험대상 천이에 대한 입력과 이 입력에 대한 출력이 구현에서도 올바르게 이루어졌는지를 확인하고, 구현에서 결과적으로 도착한 상태에 대해서도 검정하는 과정으로 구성된다. 이 과정에서 시험을 위한 입력/출력시퀀스가 필요한데, 출력오류와 천이오류를 걸러내기 위한 시험단계에서 관리한계(controllability limit)와 관찰한계(observability limit)의 문제가 발생할 수 있다. 일반적으로 관리한계 문제를 해결하기 위해서 초기 상태로부터 시작하여 원하는 상태까지 가장 짧은 패스(shortest path)를 이용하여 상태에 도착한 후 해당 천이를 시험하게되고, 관찰한계 문제를 해결하기 위해 시험하는 천이 후에 도착한 상태의 유일한 시험 시퀀스를 시험계열에 포함시켜 적용한 후 구현 FSM의 결과 상태를 확인하는 방법을 사용한다. 이를 위해 사용되는 시험열은 T, U, D, W의 4가지 유형이 가능하다. 이들 방법 중 T 생성방법을 제외한 나머지 3가지 방법은 각각 하나의 천이의 도착상태에 대한 확인 과정에 사용하는 상태 확인 시험열로 어떤 것을 사용하느냐에 따라 구별된다[13]. 각 시험열 생성방법에 대해서 응용성, 시험열의 길이, 시험열의 비 유일성, 시험열의 오류감지 능력을 기준으로 비교하면, T 생성방법은 간단하지만 모든 오류들을 감지하지 못한다는 단점이 있으며 U생성방법은 매우 효율적이면서도 적용범위가 넓다는 장점을 가진다. 이에 비해 D 생성방법은 U 생성방법만큼 유용하지만 제약조건이 강하며 또한 DS(Distinguishing Sequence)가 반드시 존재한다는 것을 보장하지 못한다는 단점이 있다. 마지막으로 W 생성방법은 다른 시험열 생성 방법들에 비해서 그 시험열의 길이가 긴 단점이 있다.

(2) 정형 방법의 도구

주어진 명세에 대해 구현물이적합하게 구현되었는 지를 검증하기 위해서는 명세를 특정한 형태의 명세언어로 표현한 후, 명세언어로 표현된 명세에 대해 검증을 수행할 수 있다. 이와 같은 명세 기반 설계 검증 도구([6], [7], [11], [12], [14], [15])로는 모델체커(model checker)의 경우는 logical formula을 이용하는 도구와 machine을 이용하는 도구로 나눌 수 있는데, temporal logic을 이용하는 도구로는 SMV, SPIN, HyTech, Kronos, EMC, CAESAR, Murphi, UV, Concurrency Workbench, CV, FOTMAT 등이고, 동작성능 확인도구로는 COSPAN/FormalCheck, FDR, Cuncurrency Workbench, Auto 등이다. 또한 이들의중간 단계는 버클리의 HSIS, 스텐포드의 STeP, VIS, PVS, METAFrame 등이다. 그리고, 정리증명기(theorem prover)도 도구의 자동화 정도에 따라 구분할 수 있는데, 사람이 지정해야 하는 수동 도구로는 ACL2,

Eves, LP, Nqthm(Boyer-Moore theorem prover), Reve, RRL 등이 있고, 완전자동화 도구로는 Coq, HOL, LEGO, LCF, Nuprl 등이 있다. 그리고, 반자동화 도구로는 Analytica, PVS, STeP 등이 있다. 이들 중 SMV, SPIN, DESIGN/CPN, UPPAAL, KRONOS 및 HYTECH은 인터넷에서 무료로 구할 수 있고([7], [11], [12], [14], [15]), 일부의 도구들에 대한 비교를 (표 1)에 기술하였다.

(표 1) 검증 도구 별 사용 언어

도구 이름	주관기관	사용 언어
SMV	CMU	BDD, CTL
SPIN	Bell Lab	Promela, LTL
DESIGN/CPN	미국 Meta사, 덴마크 CPN 그룹	ML
UPPAAL	스웨덴 UPPsala 대학, 덴마크AALborg 대학	timed automata
KRONOS	Verimag 대학	Timed CTL
HYTECH	Cornell 대학, UC Berkeley	Integrated CTL
PVS	SRI computer science Lab.	typed higher-order logic
LOTOS toolbox	Info. Tech. Arch	LOTOS
FDR	Formla Systems Ltd.	CSP
ProofPower	ICL secure systems	HOL, Z
Nqthm	Computational Logic Inc.	LISP

3.정형검증방법의효과및과제

정형적 방법에 대한 논란은 지금까지 진행 중이다. 이 방법의 옹호자들은 그것들이 소프트웨어 개발 방법에 대변혁을 일으킬 수 있다고 주장하지만, 비난자들은 그것들이 불가능할 정도로 어렵다고 생각한다.이는 대부분의 사람들에게 정형적방법들은 익숙하지 않아서인데, 정형적 방법들은 소프트웨어 엔지니어가 기존의 방법 또는 객체-지향 방법을 사용하였을 때 보다 명세서를 좀더 완전하게, 일관성을 가지며, 그리고 모호하지 않게 생성할 수 있도록 하여 줄 수 있다. 안전이 중대한 시스템에서의 장애는 많은 대가를 치를 수 있다. 컴퓨터 소프트웨어가 고장나면, 생명을 앗아가거나 또는 심각한 경제적 결과를 초래될 수 있으므로, 소프트웨어를 사용하기 전에 모든 오류들을 찾아내야 하는 것이 필수적이고, 정형적 방법들은 이러한 상황의 명세 오류들을 매우 효과적으로 줄여주며, 그 결과 이런 방법들이 고객이 일단 소프트웨어를 사용하면 오류를 거의 갖지 않는 소프트웨어에 대한 기초로서의 역할은 한다.

정형적 명세 기법들이 아직까지는 산업체에서 폭넓게 사용되지는 않지만, 이 기법들은 덜 정형적인 기법들에 비해 상당한 이점을 제공하는데도 불구하고, 아직 몇 가지 문제를 가지고 있다[8]. 실제 필요한 것은 추상 모델을 처리할 수 있는 것이 아니라 복잡한 대형 문제를 처리할 수 잇는 도구가 필요한데, 이를 위한 실질적인 도구는 없는 상태이고, 사용하기 힘든 정형 분석 도구의 사용법을 익히고 적용하는 시간보다 비 정형 기법을 활용하는 것이 시간 절약과 비용 면에서 유리한 경우가 많다. 따라서, 범용의 도구 개발도 필요하지만, 특수 분야에 적용이 용이한 도구의 개발, 사용하기 편리한 GUI를 갖춘 도구의 개발, 대형시스템을 처리할 수 있는 도구가 필요한데, 이를 위해서는 수학적 기호 대신 소프트웨어 공학자 들이 익숙한 기호를 사용하는 도구의 개발도 필요하다. 정형적 명세는 주로 기능과 데이터에 초점을 맞추고 있어서 타이밍, 제어 및 행위측면은 표현하기가 쉽지 않다. 그리고, 문제의 어떤 부분(예, 인간/기계 인터페이스)들은 정형기법보다는 그래픽 기법들 또는 프로토타입들을 사용하는 것이 더 좋을 수 있다. 마지막으로, 정형적 방법들을 사용하는 명세화는 구조적 분석과 같은 방법들보다 배우기가 어렵고, 일부 소프트웨어 실무자들에게는 상당한 "문화적 충격"을 나타낸다. 그러나, 사용하기 편리한 자동화 도구의 개발을 통해 이러한 문제들은 없어질 것이고, 향후 수학에 기초한 정형 방법론은 소프트웨어 개발 과정의 광범위한 부분에서 활용될 것으로 믿는다.

4.결론

소프트웨어 신뢰성 확보를 위한 적합성 검증은 어떤 구현물이 해당 명세(표준)에 적합한지 여부를 시장 유통전에 검토.확인하기 위해 일반적으로 제조업자, 이용자, 규제기관, 제3자 독립기관 등에 의해 행해진다. 이러한 활동은 관련자에게 해당 소프트웨어가 관련 표준과 부합되어 상호 운용에 문제가 없이 유통될 수 있다는 믿음을 제공하게된다. 소프트웨어 개발을 위한 구조적 분석기법, 객체지향 설계기법, 엔터티릴레이션 모형 등 많은 기법이 제안되어왔다. 이들 여러 가지 방법은 도형을 이용해 시스템을 표현하며, 일치된 이해를 위해 각종 설계지침, 지원도구들을 수반하고 있다. 하지만 이들 중 어떤 것도 소프트웨어의 검증 혹은 증명이 요구될 경우 특별한 해법을 제공해 주지 못한다. 이에 대한 대안으로 현재 많은 연구가 진행 중인 것이 정형 방법이다.

본 연구에서는 소프트웨어 신뢰성 향상를 위한 방안의 하나로 정형 방법에 대한 조사를 하였고, 그 활용 방안을 모색하였다. 정형 방법에 대한 다양한 연구 및 도구가 개발되어 있지만, 아직 해결되어야 할 문제도 많은 상태이다. 먼저, 다양한 도구 중 목적에 적합한 도구 선택을 위한 지침이 연구되어야 하는데, 이를 통해서 각 도구의 개선 및 활용 효과의 극대화가 이루어질 것이다. 둘째로, 사용하기 편이한 도구의 개발이 필요하다. 배우기 쉽고 값싼 도구의 개발 및 보급을 통해 소프트웨어 개발의 경쟁력을 확보할 수 있을 것으로 판단된다.

참고문헌

[1] B. Hailpern and P. Santhanam, "Software Debugging, Testing, and Verification", *IBM Systems Journal*, Vol.41, No.1, 2002, pp.4-12.
[2] G.J. Holzman, *Design and Validation of Computer Protocols*, Prentice-Hall, 1991.

[3] R.S. Pressman, *Software Engineering : A Practitioner's Approach*, McGraw-Hill International Editions, 1992.
[4] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie, *Systems and Software Verification : Model-Checking Techniques and Tools*, Springer, 2001.
[5] H.V. Bertine, W.B. Elsner, P.K. Verma, and K.T. Tewani, "Overview of Protocol Testing Programs, Methodologies, and Standards", *AT&T Technical Journal*, 1990, pp.7-16.
[6] E.M. Clarke and J.M. Wing, "Formal Methods : State of the Art and Future Directions", *ACM Computing Surveys*, 1996,
[7] Design/CPN-Computer Tool for Coloured Petri Nets, *http://www.daimi.au.dk/designCPN*, 2003.
[8] F. Dietrich and J.-P. Hubaux, "Formla Methods for Communication Services : Meeting the Industry Expections", *Computer Networks*, 2002, Vol.38, pp.99-120.
[9] K. Drira, P. Azema, and P.S. Sannes, "Testability Analysis in Communicating Systems", *Computer Networks*, Vol.36, 2001, pp.671-693.
[10] G.J. Holzmann, "Design and Validation of Protocols : A Tutorial", *Computer Networks and ISDN Systems*, 1993, Vol.25, pp.981-1017.
[11] G.J. Holzmann, "The Model Checker SPIN", *IEEE Transactions on Software Engineering*, 1997, Vol23, No.5, pp.279-295.
[12] HyTech: The HYbrid TECHnology Tool, *http://www-cad.eecs.berkeley.edu/~tah/HyTech*, 2003.
[13] D.P. Sidhu and T.-K. Leung, "Formal Methods for Protocol Testing : A Detailed Study", *IEEE Transactions on Software Engineering*, 1989, Vol.15, No.4, pp.413-426.
[14] The SMV System, *http://www-2.cs.cmu.edu/~modelcheck/smv.htm*, 2003.
[15] The tool Kronos, *http://www-verimag.imag.fr/ TEMPORISE/kronos*, 2003.