자료흐름을 고려한 테스트 스위트 생성기법

김 용 승[†] · 우 성 회[†] · 오 병 호[††] · 이 상 호[†††]

요 약

프로토콜이 방대화되고 복잡해짐에 따라 기존의 제어흐름 중심의 테스트는 프로토콜이 갖는 자료흐름, 전이조건, 지연 등의 복합요소를 테스트하는데 문제점을 갖는다. 따라서 본 연구에서는 자료흐름을 제어흐름에 통합하여 테스트 스위트를 생성함으로써 제어흐름 분석시 발생하는 비결정성 문제를 해결하고, 도달가능 트리를 이용하여 실행 불가능 경로를 제거한 후 실행가능 경로에 대한 테스트 스위트만을 생성하는 기법을 제안하고 이를 설계 및 구현하였다. 설계 및 구현된 도구는 기존 방법과 비교하여 UIO 길이를 줄이며 오류 가능성이 높은 특정 경로에 대한 부분적인 테스트가 필요한 경우에 효율적으로 사용할 수 있다. 또한 구현된 자동 테스트 스위트 생성도구의 활용으로 프로토콜 구현의 생산성을 높일 수 있으며 프로토콜 테스팅 환경의 기반을 제공한다.

Test Suites Generation Method in Consideration of Data Flow

Yong Seung Kim[†] · Sung Hee Woo[†] · Byeong Ho Oh[††] · Sang Ho Lee[†††]

ABSTRACT

Recent evolution of communication networks has led toward increasingly a complex, large-scale protocol. Protocol conformance tests therefore, which consider only control flow, have many problems on testing data flow, transition condition, delay and so on. We propose, design, and implement the tool to solve these problems. The tool, which solves nondeterminism, generates test suites from an integrated flow graph and excludes infeasible path with reachable tree. The presented tool reduces the length of UIO sequence and is efficient partially to test the path that error rate is high. Our automatic test suite generator provides basis of protocol testing environment and high production.

1. 서 론

90년대를 정보화 시대의 시작이라 한다면, 이 정보화 시대는 모든 사람들이 통신망이나 기타 관련되는 정보기술을 이용하여 어느 곳에서나 어느 누구에게로 정보를 전달할 수 있어야 한다. 이를 위하여 서로 정보를 교환하고 교환된 정보가 처리될 수 있도록 하는 자유로운 통신환경을 구축하기 위하여 많은 정보통신 제품 혹은 서비스 시스템 등이 개발되고 있다. 이에 ISO에서는 표준안인 OSI를 개발하여 모든 구현제품이나 실제 시스템이 이 표준안에 맞게 개발되도록 권고하고 있다. 따라서 개발된 시스템이 OSI 표준 즉, 프로토콜 표준을 따르는지에 대한 여부를 시험하는 것이 적합성 시험[2]이다. 이 시험은 구현제품인 IUT에 입력/출력인 데이터 시퀀스를 적용하는 것이다. 적합성 시험을 위한 테스트 스위트 생성에 관한 연구

※이 연구는 95년도 학술진흥재단의 자유융모 과제에 의해 수행된 결과임
† 정 회 원:청주전문대 전자계산과
†† 정 회 원:충남전문대 전자계산과
††† 정 회 원:충북대학교 컴퓨터과학과
논문접수:1997년 1월 21일, 심사완료:1997년 7월 4일

는 주로 제어흐름을 중심으로 전이의 출력 및 도달 상태를 확인하는 목적으로 이루어졌으나 프로토콜의 제어흐름만[1][4]을 고려하는 경우 실제 프로토콜이 갖는 자료흐름, 전이조건, 지연 등의 복합요소에 대한 테스트상의 문제가 발생한다. 프로토콜이 복잡한 경우 실제 완전한 모든 자료흐름을 고려한다는 것은 불가능하지만 부분적인 자료흐름 요소를 고려함으로써, 제어흐름만을 고려할 경우에 발생하는 문제점[6]을 해결할 수 있으므로 자료흐름[3][5]이 부분적으로 고려된 연구[11][14][15][16]가 진행 중에 있다.

또한 정보통신의 고속화 멀티미디어화 등에 따라 프로토콜의 규모가 커지고 복잡해짐에 따라 프로토콜의 개발 싸이클 즉, 설계, 검증, 구현, 시험 등의 작업을 수작업으로 하는 것은 불가능하게 되었다. 따라서 긴 개발 시간과 수작업에 의한 에러를 줄이며, 수작업으로는 처리하기 어려운 과정을 자동화하는 도구는 더 향상된 품질과 높은 생산성을 지닌 결과를 얻을 수 있다. 즉, 프로토콜 공학의 궁극적인 목표는 개발 싸이클[14]에 따른 과정을 수행함에 있어 생산성을 향상시키는 것이라 할 수 있다.

본 연구에서는 기존 연구[12]의 결과를 토대로 프로토콜이 Estelle로 기술된다는 가정 하에 분기조건을 기술한 PROVIDED절의 변수를 기준으로 자료흐름을 제어흐름과 통합하여 프리디케이트(Predicate)가 포함된 제어흐름도를 생성한다. 그리고 이 제어흐름도로부터 제어흐름 분석시 발생하는 비결정성 문제를 해결하고 실행 불가능 경로를 제거한 실행가능 경로에 대한 테스트 스위트만을 얻는 방법을 제안한다. 또한 [12]의 연구에서 제안된 방법 중 문맥변수와 흐름도 생성을 간소화하는 방법을 제안하고 이를 수행하는 도구를 설계 및 구현하였다. 본 연구의 내용은 2절에서 기존에 연구된 테스트 스위트 생성 방법 및 테스팅 환경을 기술하고 3절에서는 생성기의 구성 및 핵심 모듈의 기능과 입출력을 기술하며 마지막 4, 5절에서는 사례로서 TP0를 이용하여 테스트 스위트의 생성과정을 보이고 결과를 분석하였다.

2. 프로토콜의 테스트 스위트 생성 방법 및 테스팅 환경

적합성 시험에 필요한 테스트 스위트에 관한 대부분의 연구는 FSM을 기본으로 명세서상의 제어흐름만을 분석하였다. 이것은 구현제품이 프로토콜 명세에 있는 어느 한 상태에서 다음 상태로 정확히 이동하는지 그리고 그 과정에서 정확한 출력을 내는지를 시험하는 것이다. 이러한 시험을 위한 테스트 스위트 생성방법으로 T method, D method, UIO method, W method 등이 있으며 이 방법들은 FSM이 minimal하고 strongly connected 되었으며 완전히 기술된 Mealy machine 이어야 한다는 제약사항[3]을 가지고 있다. 이중 UIO method는 상태별로 유일한 입출력 유형을 갖는 입력 순차열의 집합으로 일반적인 DS 보다 짧고 마지막(tail) 상태를 테스트하므로 오류검색 범위가 T method 보다 크다. UIO method에 의한 상태 S_i에서 S_j로의 테스트 시퀀스를 생성하는 방법을 간단히 살펴보면 다음과 같다.

1단계 : Machine이 초기 상태로 가도록 reset를 입력한다.
2단계 : 초기 상태에서 상태 S_i까지의 가장 짧은 경로를 찾아 추가한다.
3단계 : 상태 S_i에서 상태 S_j로의 트랜지션 입력을 추가한다.
4단계 : 상태 S_j를 위한 UIO를 추가한다.

그러나 자료흐름을 고려하지 않을 경우 UIO가 모든 상태에 다 존재하지 않을 수도 있고 비결정성 전이가 발생하여 테스트 스위트의 신뢰성이 낮아질 수 있다. 따라서 FSM을 기반으로 얻어진 테스트 스위트들은 제어부분만을 시험할 수 있고 자료 부분의 구현상 오류는 시험 불가능하다. 그리고 전체적인 제어흐름만을 고려할 경우 시간적, 공간적 복잡성 및 제약성 등 다음과 같은 문제점들을 갖는다.

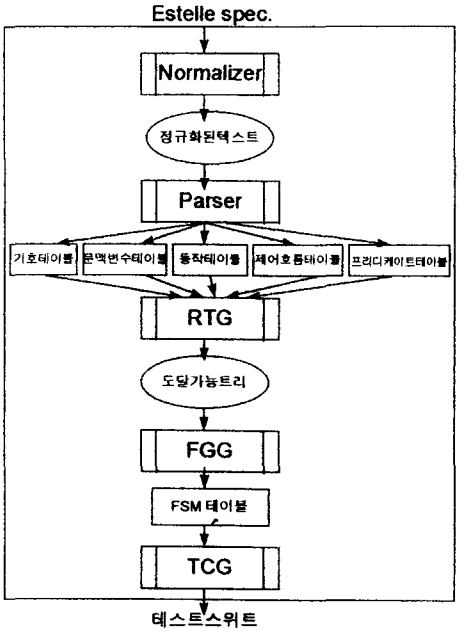
- 출력된 자료의 내용 값에 대한 시험결과를 표시할 수 없으므로 완전한 시험의견(Pass/Fail)의 표현이 불가능하다.
- 생성된 테스트 스위트에 부분적으로 실행 불가능한 경우가 포함될 수 있으며 전체 테스트 스위트의 신뢰성을 저하시킨다.
- 비결정성을 내포하는 프로토콜인 경우 제어흐름 분석만으로는 근본적인 처리가 불가능하다.

이러한 문제를 해결하기 위하여 기본적인 제어흐름과 일부 자료흐름을 통합한 EFSM 모델을 기본으로 테스트 스위트를 생성하는 방법이 연구되고 있다. EFSM을 기반으로 한 테스트 스위트 생성은 전체 자료흐름을 고려할 경우 너무 복잡해지므로 정규화 과정(normalization)을 거쳐 단일 모듈로 변환하기도 하고 문법적으로 정확한 변환을 위한 시뮬레이션을 이용하는 방법이 제안되기도 하였다. 이외의 자료흐름에 관한 연구로 Sarikaya[15]는 노드를 연결하여 DFG를 만들고 기능적으로 이를 통합하였으며 Ural[5]은 변수의 정의, 사용 과정을 I-O-Define chain으로 연결하고 자료흐름의 변칙성을 검색하였다. Luo[16]는 명세서에서 CFG를 생성하는 알고리즘을 제시하였고 Chan[7]은 제어흐름도에 의한 순환경로에 자료흐름을 통합하여 자료흐름 측면에서 실행가능 여부를 판단하였다. 따라서 모든 자료흐름을 프로토콜 시험에 적용시키는 것은 현실적으로 어려운 작업이지만 자료흐름 테스트의 종류 및 기준, 자료흐름 분석과정에서 실행불가능 경로의 존재 및 분석의 필요성[8], Driver Program을 이용한 실행성[9], 테스트 불능 프로그램의 개념과 사례[10]등이 정의되었으며 평가기준에 대한 제안과 실험이 계속적으로 이루어지고 있다. 그러나 프로토콜의 자료흐름에 대한 그 동안의 연구는 여전히 불완전한 측면을 가지고 있어 기존 문제점을 보완하고 실행성과 자동화를 고려하는 새로운 연구의 필요성이 제기되고 있다. EFSM을 기반으로한 테스트 스위트 생성 방법으로는 FSM을 기반으로 하는 테스트 스위트 생성방법을 확장한 UIOv method, Multiple UIO method, Partial W method 등이 있다.

프로토콜 구조의 복잡화, 네트워크 이용확산으로 인한 정보통신의 원활한 소통의 필요, 편리하고 신뢰성 있는 통신 프로토콜에 대한 요구 등으로 인해 더 효율적이고 정확한 프로토콜의 설계 및 구현 기법이 요구되고 있다. ISO에서는 적합성 시험의 일반적인 사항을 표준으로 기술하며 테스팅 환경은 프로토콜 개발 싸이클에 따라 단계별로 지원도구 및 시스템들이 연구되고 있다. 캐나다 Concordia 대학의 CONTEST-ESTL[14], NIST의 integrated tool set[14], 프랑스의 EDB, 캐나다 UBC의 테스팅 환경 등[13]이 연구되고 있는 실정이며 각 단계를 지원하는 도구들은 하나의 통합된 환경을 구축하고 있는 추세이다.

3. 테스트 스위트 생성기의 설계 및 구현

자료흐름이 고려된 테스트 스위트 생성기의 구조는 다음 (그림 1)과 같다. 본 연구에서 설계 및 구현한 부분은 Parser 모듈과 FGG 모듈로 SUN spark10 시스템에서 C언어로 구현하였으며 Normalizer 모듈, TCG 모듈과 RTG 모듈은 이미 연구된 [5][7][12]의 방법을 사용하였다. 테스트 스위트 생성기는 (그림 1)과 같이 4개의 서브모듈에 의해 실제 실행 가능한 테스트 케이스들을 생성한다. 정규화 모듈인 Normalizer는 Estelle spec.을 단일 모듈로 만들기 위한 단순화 과정을 수행하는 모듈로 정규화된 Estelle 텍스트를 생성 한다. Parser 모듈은 정규화된 Estelle 텍스트로부터 기호 테이블, 문맥변수 테이블, 제어흐름 테이블, 동작 테이블, 프리디케이트 테이블을 생성한다. 도달가능 트리 생성모듈[12] (RTG : Reachable Tree Generator)은 전 단계에서 생성된 테이블로부터 도달가능 트리를 생성하고 생성된 트리는 흐름도 생성모듈의 (FGG : Flow Graph Generator) 입력이 된다. 흐름도 생성모듈은 테스트 케이스를 생성하기 위한 기본



(그림 1) 테스트 스위트 생성기의 구조
(Fig. 1)Architecture of Test Suite Generator

자료가 되는 FSM 테이블을 생성한다. 테스트 케이스 생성모듈은 이 FSM 테이블에 테스트 스위트 생성 알고리즘을 적용하여 시험에 쓰일 테스트 스위트들을 생성하게 된다. 각 모듈별 기능 및 입출력을 [12]의 연구에서 개선된 것들을 중심으로 살펴보면 다음과 같다.

3.1 Parser 모듈

Parser 모듈은 정규화된 Estelle 텍스트를 입력받아 기호 테이블, 동작 테이블, 제어흐름 테이블, 문맥변수 테이블, 프리디케이트 테이블을 생성한다. Parser 모듈의 전체 처리 알고리즘과 Parser 모듈을 구성하는 4가지 서브모듈 중 핵심 모듈의 기능 및 구성요소를 살펴보면 다음과 같다.

〈알고리즘〉

1단계 : 명세서에 기술된 변수를 분석하여 기호 테이블을 생성한다.

2단계 : 각 트랜지션 블록내의 동작 블록을 분해하여 동작 테이블을 생성한다.

3단계 : 내부변수가 전이조건에 사용된 경우 문맥변수 테이블을 생성한다.

4단계 : 통제 가능한 변수가 전이조건으로 사용된 경우 프리디케이트 테이블을 생성한다.

5단계 : 제어흐름 테이블을 생성한다.

기호 테이블 생성 모듈[12]은 정규화된 Estelle 텍스트를 입력받아 변수를 등록하며 수행 시작에서 끝까지 변수값의 변화 과정을 트랜지션별로 기록하여 변수 값을 추적할 수 있는 테이블을 생성한다. 동작 테이블 생성모듈[12]은 정규화된 Estelle 텍스트의 트랜지션 부분중 BEGIN END 사이의 구문을 입력받아 output절을 제외한 나머지 구문을 트랜지션별로 등록시킨다. 이때 수행문의 등록은 기호 테이블에 있는 변수값의 호출 및 갱신을 수반한다.

(1)제어흐름 테이블(CFT:Control Flow Table) 생성모듈

이 모듈은 정규화된 Estelle 텍스트를 입력받아 트랜지션별 현 상태와 입력/출력, 다음 상태 등을 생성하는 모듈로 텍스트의 FROM, TO, WHEN, output 절을 찾아 각 상태와 입/출력을 등록시키며 문맥번호, 동작번호, 프리디케이트번호, 기호 테이블을 참조하여 시스템의 제어흐름 그래프를 생성하기 위한 기본 자료들로 구성된 테이블을 생성한다. 알고리즘은 다음과 같다.

〈알고리즘〉

1단계 : Estelle로 표현된 텍스트의 트랜지션 부분을 읽어들인다.

2단계 : 트랜지션별 FROM, TO, WHEN, output절의 내용을 저장하고 WHEN절의 입력이 외부 통제 가능한 변수이면 프리디케이트 번호와 결합시킨다.

3단계 : 동작번호는 동작 테이블의 동작번호를, 조건 문맥번호는 문맥변수 테이블의 문맥번호를 참조하도록 한다.

4단계 : 마지막 트랜지션까지 1단계에서 3단계의 과정을 반복한다.

(2)문맥변수 및 프리디케이트 테이블(CVPT:Context Variable and Predicate Table) 생성 모듈이 모듈은 정규화된 Estelle 텍스트의 PROVIDED절을 WHEN절과 비교하여 중복되지 않는 변수일 때 이것을 한 상태에서 다음 상태로의 전이조건으로 간주하여 문맥변수 테이블에 트랜지션별로 PROVIDED절의 부울변수 및 부울수식을 찾아 등록시킨다. 부울변수는 두 가지의 값을 가지므로 하나의 부울변수 등록은 결국 두 가지 상태의 조건으로 등록된다. 또한 위 비교 결과가 같으면 부울변수 및 부울수식을 프리디케이트 테이블에 등록시키고 번호를 부여하여 입력에 통합시킨다. 이것은 한 상태에서 같은 입력에 의하여 서로 다른 상태로 전이되는 비결정성 문제를 해결하기 위하여 또 하나의 전이조건으로 입력에 통합시킨 것으로 제어흐름 테이블에서 참조된다.

프로토콜에서 상태의 전이는 자료의 입력과 전이조건에 의해 발생되며 전이조건은 변수 및 상수가 논리 연산자로 결합된 상태로 사용된다. 문맥변수는 프로토콜의 제어흐름을 결정하는 제어변수를 총칭하며 문맥변수는 전이조건의 기술에 사용되는 좁은 의미의 제어변수이다. IUT를 Black Box로 볼 때, 문맥변수가 입력 변수인 경우 테스터는 PDU의 입력 값을 조정하여 제어흐름을 통제할 수 있다. 그러나 내부변수가 문맥변수로 사용된 경우에는 흐름의 외부 통제

가 불가능하므로 문맥변수는 통제 가능여부로 구분할 수 있다. 따라서 통제 가능한 입력이면 전이조건으로 입력에 통합시키고 통제 불가능한 입력이면 문맥변수로 간주한다. 전이조건은 논리 값을 가지는 부울변수 및 부울수식으로 PROVIDED절의 부울변수 및 부울수식이 된다. 본 연구는 문맥변수들의 상태를 동시에 고려할 것을 제안하는 [12]와는 달리 전이에 직접 필요한 문맥변수만을 고려하고 이 변수의 결과값을 등록하여 테이블을 간소화하였다. 생성 알고리즘은 다음과 같다.

〈알고리즘〉

1단계 : Estelle로 표현된 텍스트의 트랜지션 부분을 읽어들인다.

2단계 : PROVIDED절의 변수와 WHEN절의 변수를 비교한다.

3단계 : 2단계에서의 비교 결과가 같으면 5단계로 분기하고 같지 않으면 4단계로 분기한다.

4단계 : PROVIDED절의 부울수식 또는 부울변수에 문맥번호를 부여하여 하나의 조건으로 문맥변수 테이블에 등록시킨 후 6단계로 분기한다.

5단계 : 부울변수 및 부울수식에 번호를 부여하여 프리디케이트 테이블에 하나의 조건으로 등록시킨 후 6단계로 분기한다.

6단계 : 마지막 트랜지션까지 1단계에서 5단계의 과정을 반복한다.

3.2 흐름도 생성 (FGG : Flow Graph Generator) 모듈

이 생성 모듈은 도달가능 트리를 입력받아 상태 집합과 입출력을 구하여 FSM 테이블을 생성한다. 이 테이블은 실행성이 보장되고 확장된 상태공간 중 미사용된 상태를 제거하여 축소시킨 형태로, 문맥변수를 통제 가능 여부로 구분한 후 입력과 상태에 결합하여 자료흐름과 제어흐름이 통합되어 표시된다. 본 연구에서는 이 모듈의 출력이 [12]에서 제안된 FSM 형태의 흐름도 대신 구현을 용이하도록 테이블 형태로 출력하도록 하였다. 도달가능 트리에서 한 상태에서 다음 상태로 전이될 때 출발 상태는 현 상태에, 목적지 상태는 다음 상태에, 입력은 레이블인 전이조건번호에 해당되는 입력에 대응된다. 생성 알고리즘은 다음과 같다.

〈알고리즘〉

1단계 : 도달가능 트리를 입력받아 노드는 "상태"로 레이블은 "입력"으로 집합을 생성한다.

2단계 : 레이블에 의해 한 상태에서 다음 상태로 전이될 때 출발 상태를 현 상태로, 도달 상태를 다음 상태로, 레이블을 입력으로 저장한다.

3단계 : 레이블에 해당되는 전이조건번호를 제어흐름 테이블에서 찾아 입력에 해당되는 메시지로 대치시킨다.

4단계 : 마지막 노드까지 1단계에서 3단계의 과정을 반복한다.

3.3 주요 테이블의 구조

본 연구에서 설계 및 구현된 생성기의 주요 모듈별 생성 테이블의 레코드 형식을 보면 다음과 같다.

(1) 제어흐름 테이블의 레코드 구조

제어흐름 테이블 생성 모듈에 의해 생성되는 테이블의 레코드 구조는 다음과 같다.

```
struct control_table {
    char t_num[3];/* 트랜지션번호 */
    char c_state[5];/* 현 상태 */
    char n_state[5];/* 다음 상태 */
    char c_in[5];/* 입력 */
    char c_out[5];/* 출력 */
    char act_num[3];/* 동작번호 */
    char cv_num[3];/* 문맥번호 */
}
```

(2) 문맥변수 테이블의 레코드 구조

문맥변수 테이블 및 프리디케이트 테이블 생성 모듈에 의해 생성되는 문맥 변수 테이블의 레코드 구조는 다음과 같다.

```
struct context_table {
    char cv_num[3];/* 문맥번호 */
    char tran_con[1];/* 전이조건 */
    char b_exp[30];/* 부울수식 및 부울변수 */
}
```

(3) 프리디케이트 테이블의 레코드 구조

문맥변수 테이블 및 프리디케이트 테이블 생성 모

듈에 의해 생성되는 프리디케이트 테이블의 레코드 구조는 다음과 같다.

```
struct predicate_table {
    char p_num[3];/* 프리디케이트번호 */
    char b_exp[30];/* 부울수식 및 부울변수 */
}
```

(4) FSM 테이블의 레코드 구조

흐름도 생성 모듈에 의해 생성되는 테이블의 레코드 구조는 다음과 같다.

```
struct FSM_table {
    char c_state[5];/* 현 상태 */
    char fsm_in[5];/* 입력 */
    char n_state[5];/* 다음 상태 */
}
```

4. 사례연구

본 연구에서는 적용 사례로 Estelle로 기술된 TP0를 사용하였다[5]. TP0는 트랜스포트 프로토콜로 OSI 7 layer 모델의 4계층에 해당되는 것으로 종점간의 데이터 전송을 쉽게 할 수 있도록 한다. TP0는 2개의 인터페이스를 가지며 상위 및 동급 계층간에 자료를 주고받는 5개의 서브 경로를 가지고 있다. 트랜스포트 프로토콜에 의해 제공되는 서비스는 다음과 같다.

- connection establishment: 'T-Connection Request', 'T-Connection Indication', 'T-Connection Response', 'T-Connection Confirm' Primitive를 사용하여 정해진 순서에 의해 설정하는 역할을 수행한다.
- data transfer: connection이 설정되면 두 사용자는 데이터를 양방향으로 전송할 수 있다. 송신측에서 자료 전송 요청 메시지를 반송하면 수신자는 그 자료와 함께 자료가 도착했다는 indication을 받는 것으로 전송이 끝나게 된다.
- connection release: 'T-Disconnection Request', 'T-Disconnection Indication' 두 개의 primitive를 사용하여 설정된 연결을 종료한다.

TP0의 정규화된 Estelle 텍스트로부터 도달가능 트리를 얻기까지의 과정을 보이기 위하여 [5]에 있는 Estelle 텍스트의 다음 트랜지션 t8과 t14를 선택하였다. t8은 비결정성 문제, t14는 실행 불가능 경로를 설명하는데 적합한 트랜지션이다.

```
WHEN dr(disconnect.reason, add.clear.reason)
FROM wfcc
PROVIDED dr.in.disconnect.reason="user.init"
TO idle
t8:BEGIN
    ndreq.disc_reason:=dr.disconnect_reason;
    tdind.ts_disc_reason:=dr.disconnect_reason;
    tdinds.ts_user_reason:=dr.add_clear_reason;
    output nderq(disc.reason);
    output tdind(ts.user.reason., ts.disc.reason);
  END;

FROM data
PROVIDED out_buffer<>nil
TO data
t14:BEGIN
     remove(out_buffer, dt.out.user.data);
     output dt(user.data);
   END;
```

(그림 2) Estelle spec. 의 트랜지션 t8과 t14
(Fig. 2) Transition t8 and t14 of Estelle spec.

트랜지션 t8은 상태 "wfcc"에서 메시지 dr을 수신하고 dr.in.disconnect.reason = "user. init" 조건을 만족하면 다음 상태 "idle"로 전이되면서 변수 ndreq. disc_reason, tdind.ts_disc_reason, tdinds.ts_user_reason에 dr.disconnect_reason, dr.disconnect_reason, dr.add_clear_reason값을 할당하고 메시지 ndreq와 tdind를 출력하는 동작을 수행한다. 트랜지션 t8에서 dr은 상태 "wfcc"에서 수신되는 메시지로 제어흐름 테이블의 입력부분에 기록되고 FROM절의 "wfcc"는 현 상태로, TO절의 "idle"은 다음 상태로 기록된다. PROVIDED절의 부울수식의 변수와 WHEN절의 변수는 일치하므로 이 수식에 번호를 부여하여 입력 메시지 dr에 통합시키고 프리디케이트 테이블에 기록한다. dr은 t8과 t9에서 입력 메시지로 사용되므로 비결정성 문제의 원인이 된다. 그러나 PROVIDED절의 부울수식을 하나의 전이조건으로 만들어 입력에 통합시킴으로써 이러한 비결정성 문제를 해결하였다. 따라서 입력에 통합된 프리디케이트는 제어흐름만을 고려할 경우 한 상태에서 같은 입력으로 다른 상태로 전이되는 비결정성 문제를 해결하는 전이조건문이다. 또한 BEGIN END 사이의 output문을 제외한 수행문은 트랜지션 번호와 같은 번호를 부여하여 동작

테이블에 등록시킨다. 그리고 output 문은 제어흐름 테이블의 출력으로 기록된다. 이때 테이블의 입력과 출력에 결합된 U와 L은 상위층과 하위계층간의 인터액션 포인트를 나타낸다. 트랜지션 t8은 외부에서 통제 가능한 변수가 존재하는 경우이며 또 하나의 트랜지션으로 통제불능인 변수 즉, 문맥변수가 존재하는 트랜지션 t14의 예를 보면 현 상태 "data"에서 수신되는 메시지 없이 out_buffer 〈〉 nil 조건만을 만족하면 다음 상태 "data"로 전이되면서 출력버퍼를 비우고 메시지 dt를 출력하는 동작을 수행한다. t8과 같이 FROM절의 "data"는 현 상태, TO절의 "data"는 다음 상태, output문의 dt는 출력으로 제어흐름 테이블에 기록되고 BEGIN END 사이의 수행문인 remove 함수는 출력버퍼를 비우는 수행문으로 동작 테이블에 기록된다. 그리고 PROVIDED절의 부울변수인 out_buffer는 외부 통제가 불가능한 내부변수이므로 문맥번호를 부여하여 2개의 조건으로 테이블에 기록된다. 이 사례에서는 통제불가능 변수 in_buffer와 out_buffer의 상태를 고려하여 4개의 조건으로 만들고 입출력 버퍼의 용량을 1로 간주하였다. 기호 테이블 〈표 1〉, 제어흐름 테이블 〈표 3〉, 동작 테이블 〈표 2〉, 문맥변수 테이블 〈표 4〉, 프리디케이트 테이블 〈표 5〉의 참조관계를 보면 기호 테이블의 변수는 제어흐름 테이블의 입출력, 문맥변수 테이블의 변수와 동작 테이블의 변수를 참조하고, 제어흐름 테이블의 문맥번호, 동작번호, 입력 부분의 프리디케이트번호는 각각 해당 테이블의 번호를 참조한다. 위와 같이 Parser 모듈은 [5]의 트랜지션 t1~t19 까지의 트랜지션 블록을 분석하여 기호 테이블 〈표 1〉, 동작 테이블 〈표 2〉, 제어흐름 테이블 〈표 3〉, 프리디케이트 테이블 〈표 5〉, 문맥변수 테이블 〈표 4〉를 생성한다. 기호 테이블과 동작 테이블은 선택된 트랜지션 중심으로 변수와 동작문을 기술하고 나머지 테이블은 전체 내용을 기술하였다. 제어흐름 테이블로부터 도달가능 트리의 생성 과정을 구체적으로 살펴보면 현 상태를 출발 노드로, 다음 상태를 도달 노드로, 에지를 입력으로 트리를 작성한다. 이때 트랜지션 t14는 출력버퍼 비우기, 트랜지션 t16은 입력버퍼 비우기를 수행하는 트랜지션으로 버퍼의 사이즈를 1로 지정하였기 때문에 출력버퍼 넣기를 수행하는 트랜지션 t13과 입력버퍼 넣기를 수행하는 트랜지션 t15가 선행되지 않으면 전이가 불가능하다. 제어흐름만을 고려할 경우 트랜지션 t14와 t16은 전이조건을 고려하지 않기 때문에 이론적으로 전이가 가능하다. 따라서 제어흐름 테이블로부터 도달가능 트리를 유출할 때 자동적으로 이러한 경로는 표현되지 않는다. FGG 모듈은 〈표 3〉으로부터 위와 같은 방법[12]의 생성 규칙을 적용하여 제어흐름 테이블로부터 실행성이 보장되고 확장된 상태 공간중 미사용 상태가 제거된 트리(그림 3)를 생성한다. TCG 모듈[12]은 1단계에서 트리의 트랜지션 번호를 에지로 하여 각 에지에 연결된 시작 상태와 도달 상태를 기록한 후, 해당 트랜지션의 입력으로 트랜지션 번호를 대치하여 〈표 7〉을 생성한다. 2단계에서는 에지 리스트 테이블에 기술된 상태를 체크하여 각 상태에 대한 UIO 시퀀스 테이블 〈표 8〉을 생성한다. 마지막 단계에서 〈표 8〉를 이용하여 실제 IUT에 적용될 테스트 스위트 〈표 9〉를 생성한다.

예를 들어 (그림 3)의 상태 "wfcc"에서 상태 "data"로의 트랜지션 t6에 대한 테스트 시퀀스는 2절에서 기술한 알고리즘에 의하여 다음 시퀀스 1과 같이 구할 수 있다.

rU.tcreq.p1/L.cc.p6/U. tdreq－－시퀀스 1

시퀀스 1에서 r은 제어를 초기 상태로 되돌리기 위한 reset이며 U.tcreq.p1은 초기 상태 "idle"에서 상태 "wfcc"로 가기 위한 preamble이고 L.cc.p6은 상태 "wfcc"에서 상태 "data"로의 트랜지션을 테스트하기 위한 test body이고 U.tdreq는 목적 상태를 확인하기 위한 상태 "data"의 UIO인 postamble이다.

〈표 1〉 기호 테이블
〈Table 1〉 Symbol Table

트랜지션번호 / 변수	t8	t14
ndreq.disc_reason	dr.disconnect_reason	
tdind.ts_disc_reason	dr.disconnect_reason	
tdinds.ts_user_reason	dr.add_clear_reason	
out_buffer		empty

〈표 2〉 동작 테이블
〈Table 2〉 Action Table

동작번호	수행문
a8	ndreq.disc_reason : = dr.disconnect_reason tdind.ts_disc_reason : = dr.disconnect_reason tdinds.ts_user_reason : = dr.add_clear_reason
a14	out_buffer : = empty

〈표 3〉 제어흐름 테이블
〈Table 3〉 Control Flow Table

트랜지션 번호	현상태	다음상태	입력	출력	동작번호	문맥번호
t1	idle	wfcc	U.tcreq.p1	L.cr	a1	
t2	idle	idle	U.tcreq.p2	U.tdind	a2	
t3	idle	wftr	L.cr.p3	U.tcind	a3	
t4	idle	wftr	L.cr.p4	U.tcind	a4	
t5	idle	idle	L.cr.p5	L.dr	a5	
t6	wfcc	data	L.cc.p6	U.tccon	a6	
t7	wfcc	data	L.cc.p7	U.tccon	a7	
t8	wfcc	data	L.dr.p8	L.ndreq,U.tdind	a8	
t9	wfcc	idle	L.dr.p9	L.ndreq,U.tdind	a9	
t10	wftr	data	U.tcres.p10	L.cc	a10	
t11	wftr	idle	U.tcres.p11	L.dr, U.tdind	a11	
t12	wftr	idle	U.tdreq	L.dr	a12	
t13	data	data	U.tdatr	^	a13	
t14	data	data	^	L.dt	a14	c2
t15	data	data	L.dt	^	a15	
t16	data	data	^	U.tdati	a16	c3
t17	data	idle	U.tdreq	L.ndreq	a17	
t18	data	idle	L.ndind	U.tdind	a18	
t19	data	idle	L.nrind	U.tdind	a19	

〈표 4〉 문맥변수 테이블
〈Table 4〉 Context Variable Table

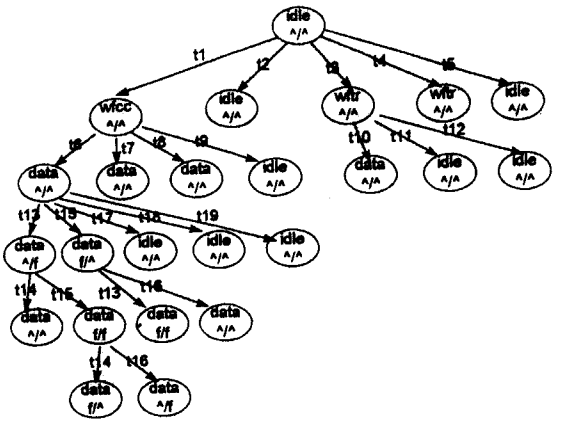
문맥 번호	전이 조건	부울변수 및 부울수식
c1	T	outul_buffer = empty
c2	F	out_buffer〈 〉empty
c3	T	in_buffer = empty
c4	F	in_buffer〈〉empty

〈표 5〉 프리디케이트 테이블
〈Table 5〉 Predicate Table

프리디케이트 번호	부울수식 및 변수
p1	tcreq.in.qts.req = ok
p2	tcreq.in.qts.req 〈 〉ok
p3	cr.in.max.tpdu.size= nil and cr.in.option = ok
p4	cr.in.max.tpdu.size 〈 〉niland cr.in.option = ok
p5	cr.in.option 〈 〉ok
p6	cr.in.max.tpdu.size 〈 〉nil
p7	cr.in.max.tpdu.size = nil
p8	dr.in.disconnect.reason = 'user.init'
p9	dr.in.disconnect.reason 〈 〉'user.init'
p10	tcreq.in.qts.req 〈= qts.estimate
p11	tcres.in.qts.req〉qts.estimate

〈표 6〉 FSM 테이블
〈Table 6〉 FSM 테이블

현 상태	입력	다음상태
idle(/)	U.tcreq.p1 U.tcreq.p2 L.cr.p3 L.cr.p4 L.cr.p5	wfcc(/) idle(/) wftr(/) wftr(/) idle(/)
wfcc(/)	L.cc.p6 L.cc.p7 L.dr.p8 L.dr.p9	data(/) data(/) idle(/) idle(/)
wftr(/)	U.tcres.p10 U.tcres.p11 U.tdreq	data(/) idle(/) idle(/)
data(/)	U.tdatr L.dt U.tdreq L.ndind Linrind	data(f/) data(/) idle(/) idle(/) idle(/)
data(f/)	U.tdatr ^	data(f/f) data(/)
data(f/f)	^ ^	data(f/) data(/f)
data(/f)	^ L.dt	data(f/f) data(/)



(그림 3) TP0의 도달가능 트리
(Fig. 3) Reachable tree of TP0

〈표 7〉 에지 리스트
〈Table 7〉 Edge lists

입력	시작상태와 도달상태
U.tcreq.p1	(idle(ˆ/ˆ), wfcc(ˆ/ˆ))
U.tcreq.p2	(idle(ˆ/ˆ), idle(ˆ/ˆ))
L.cr.p3	(idle(ˆ/ˆ), wftr(ˆ/ˆ))
L.cr.p4	(idle(ˆ/ˆ), wftr(ˆ/ˆ))
L.cr.p5	(idle(ˆ/ˆ), idle(ˆ/ˆ))
L.cc.p6	(wfcc(ˆ/ˆ), data(ˆ/ˆ))
L.cc.p7	(wfcc(ˆ/ˆ), data(ˆ/ˆ))
L.dr.p8	(wfcc(ˆ/ˆ), idle(ˆ/ˆ))
L.dr.p9	(wfcc(ˆ/ˆ), idle(ˆ/ˆ))
U.tcres.p10	(wftr(ˆ/ˆ), data(ˆ/ˆ))
U.tcres.p11	(wftr(ˆ/ˆ), idle(ˆ/ˆ))
U.tdreq	(wftr(ˆ/ˆ), idle(ˆ/ˆ))
U.tdatr	(data(ˆ/ˆ), data(ˆ/ˆ)) (data(f/ˆ), data(f/f))
ˆ	(data(ˆ/f), data(ˆ/ˆ)) (data(f/f), data(f/ˆ)
L.dt	(data(ˆ/ˆ), data(f/ˆ)) (data(ˆ/f), data(f/f)
ˆ	(data(f/ˆ), data(ˆ/ˆ)) (data(f/f), data(ˆ/f)))
U.tdreq	(data(ˆ/ˆ), idle(ˆ/ˆ))
L.ndind	(data(ˆ/ˆ), idle(ˆ/ˆ)
L.nrind	(data(ˆ/ˆ), idle(ˆ/ˆ))

〈표 8〉 시퀀스
〈Table 8〉 UIO sequence

상태	UIO sequence
idle(ˆ/ˆ)	U.tcreq.p1, U.tcreq.p2, L.cr.p3, L.cr.p4, L.cr.p5
wfcc(ˆ/ˆ)	L.cc.p6, L.cc.p7, L.dr.p8, L.dr.p9
wftr(ˆ/ˆ)	U.tcres.p10, U.tcres.p11, U.tdreq
data(ˆ/ˆ)	U.tdreq, L.ndind, L.nrind
data(ˆ/f)	(ˆL.dt) (t14, t15) (ˆU.tdreq) (t14, t17) (ˆL.ndind) (t14, t18) (ˆL.nrind) (t14, t19) (L.dt ˆ) (t14, t15) (L.dt ˆ) (t15, t16)
data(f/ˆ)	(U.tdatr ˆ) (t13, t14) (U.tdatr ˆ) (t13, t16) (ˆU.tdreq) (t16, t17) (ˆL.ndind) (t16, t84) (ˆL.nrind) (t16, t19)
data(f/f)	(ˆ/ˆ) (t14, t16) (ˆ/ˆ) (t16, t14) (ˆL.dt) (t16, t15)

〈표 9〉 테스트 스위트
〈Table 9〉 Test Suite

테스트 스위트
rtreq.p2/treq.p1
rtreq.p1/dr.p9/treq.p1
rtreq.p1/cc.p6/L.ndind
rcr.p3/tcres.p11/treq.p1
rcr.p3/tcres.p10/L.ndind
rtreq.p1/cc.p6/U.tdatr/null/L.ndind
rtreq.p1/cc.p6/U.tdatr/L.dt/null/null
rtreq.p1/cc.p6/U.tdatr/L.dt/null/null
rtreq.p1/cc.p6/L.dt/null/L.ndind
rtreq.p1/cc.p6/L.dt/U.tdatr/null/null
rtreq.p1/cc.p6/U.tdatr/L.dt/null/null/U.tdreq
rtreq.p1/cc.p6/U.tdatr/L.dt/null/U.tdatr/null
rtcreq.p1/cc.p6/U.tdreq/U.tdatr/null/U.tdreq/tcreq.p1

5. 결과 분석

본 논문의 제안 방법에 의하여 생성된 테스트 스위트를 기존의 제어흐름 지향의 생성기법[1]과 비교하면 다음과 같다. 예를 들어 상태 "idle"에서 t1, t6, t14, t18을 통하여 상태 "idle"로 되돌아오는 패스에 대한 테스트 스위트는 제어흐름만을 고려할 경우 시퀀스 2와 같다.

rtcreq/cc/null/ndind－－시퀀스 2

위 시퀀스 2에서 tcreq는 t1, t2에서 cc는 t6, t7에서 입력으로 사용되므로 비결정성 문제가 발생한다. 그리고 t14의 트랜지션은 출력 버퍼 비우기 동작을 수행하는 부분으로 버퍼의 크기를 1로 지정하였기 때문에 출력 버퍼 넣기를 수행하는 t13이 선행되지 않으면 실행 불가능한 경로(infeasible path)[12]이다. 이러한 실행 불가능한 테스트 시퀀스의 적용은 테스팅 자체의 신뢰도를 저하시킨다. 또한 TP0에서 입력 버퍼 비우기 동작을 수행하는 t16도 입력 버퍼 넣기를 수행하는 t15가 선행되지 않으면 실행 불가능한 경로이다.

본 연구에서 생성된 시퀀스 3은 t1, t2 그리고 t6, t7과 같은 트랜지션에서 발생하는 비결정성 문제를 해결하기 위하여 PROVIDED절의 부울값(p1, p6)을 입력변수에 통합한다. 또한 단독으로는 실행 불가능한 경로인 t14, t16을 문맥변수를 사용하여 제거한 후 실행 가능한 패스만으로 표현된 도달가능 트리로부터 생성된 테스트 시퀀스는 다음 시퀀스 3과 같다.

rtcreq.p1/cc.p6/U.tdatr/null/ndind/tcreq.p1－－시퀀스 3

t14에 해당되는 입력인 null은 t13의 입력에 해당되는 U.tdatr을 선행 입력으로 하고 있다. 따라서 제어흐름에 전이조건을 기술한 문맥변수와 프리디케이트를 결합시킨 시퀀스 3은 시퀀스 2에서 존재하는 실행 불가능 경로 t14가 없고, 같은 입력인 cc에 의해 다음 상태로 전이될 때의 비결정성 문제가 존재하지 않는다. 또한 도달가능 트리를 사용하여 실행 가능한 테스트 시퀀스만을 얻을 수 있어 시험대상 경로의 수가 축소된다. 그리고 입력에 제어조건을 추가하고 입력을 차별화 하여 UIO 길이가 짧아지며 각 상태에 여러 종류의 UIO가 존재하므로 동일한 길이를 갖는 여러 종류의 테스트 스위트를 생성할 수 있다. 이것은 사용율이나, 오류가능성이 높은 특정 경로에 대한 부분적인 테스트가 필요한 경우에 효율적으로 사용할 수 있다.

6. 결 론

기존의 테스트 스위트 생성에 관한 연구는 FSM을 기준으로 명세서상의 자료흐름을 무시하고 제어흐름만을 분석하였다. 설계대상 프로토콜의 규모가 방대하고 복잡해지는 오늘날의 추세에 비추어 볼 때 자료흐름을 프로토콜 시험에 적용함은 매우 어려운 작업이나 자료흐름을 고려하지 않을 경우 완전한 시험 의견 표현이 불가능하고 생성된 테스트 스위트에 부분적으로 실행 불가능한 경우가 표시되는 등 문제점을 갖게 된다.

본 연구에서는 이러한 문제점을 해결하기 위하여 전이조건의 기술에 사용된 문맥변수를 제어흐름에 통합하여 부분적으로 자료흐름이 고려된 실행 가능 흐름도를 정의하여 비결정성 문제를 근본적으로 해결하고, 도달가능 트리를 활용하여 실행 불가능한 경로가 제거된 테스트 스위트만을 추출하는 방법을 설계하고 이를 구현하였다. 제안된 생성방법은 도달가능 트리를 사용하여 실행 가능한 테스트 스위트만을 얻을 수 있어 시험대상 경로의 수를 축소시키고 입력에 전이조건을 추가 및 입력을 차별화 하여 UIO 길이를 줄일 수 있다. 또한 각 상태에 여러 종류의 UIO가 존재하므로 동일한 길이를 갖는 여러 종류의 테스트 스위트를 생성할 수 있어 사용율이나, 오류가능성이 높은 특정 경로를 효율적으로 테스트 할 수 있다. 구현된 생성기는 자동화된 테스트 스위트 생성도구로 생산성을 높일 수 있으며 프로토콜 테스팅 환경의 기반을 제공한다.

앞으로의 연구방향은 테스트 스위트 생성기에 입력되는 프로토콜 명세서가 Estelle 이외의 다른 프로토콜 명세언어를 처리할 수 있는 테스트 스위트 생성기를 연구 및 설계하는 것이며 이를 기반으로 다른 테스팅 환경요소들을 연구 및 추가 설계하여 개발 싸이클에 따른 하나의 완전한 프로토콜 테스팅 환경을 구축하는 것이다.

참 고 문 헌

[1] B.S.Bosik, M.U.Uyer, "Finite State machine based formal methods in protocol conformance testing," Computer Networks and ISDN System, vol. 22, pp.7-33, 1991.

[2] B.Sarikaya, "Conformance Testing: Architectures and test sequences," Computer Networks and ISDN System, vol. 17, pp.111-126, 1989.

[3] K.Sabnani, A.Dahbura, "A protocol test Generation Procedure," Computer Networks and ISDN System, vol. 15, pp.285-297, 1988.

[4] S.Fujiwara, G.V.Bochmann, F.Khendek, M. Amalou, A.Ghedamsi, "Test Selection based on Finite State Models," IEEE Transaction on Software Engineering, Vol 17, No. 6, pp.591-603, 1991.

[5] H.Ural, "Test sequence selection based on static data flow analysis," Computer Communications, pp.234-242, 1987.

[6] H.Kloosterman, "Test derivation from nondeterministic finite state machines," Proceedings of the 5th International Workshop on Protocol Test Systems, pp.254-265, 1992.

[7] W.Chan and P.D.Amer, "Test Case Generation for Protocols Specified in Estelle," FORTE'90, Marrid, No. 11, pp.197-210, 1990.

[8] M. Chellappa, "Nontraversible Paths in a Program," IEEE Transaction on Software Engineering, Vol. SE-13, NO. 6, pp.751-756, 1987.

[9] P.G.Frankl, S.N.Weiss, "An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing," IEEE Transactions on Software Engineering, Vol. 19, NO. 8, pp.774-787, 1993.

[10] E.J.Weyuker, "On Testing Non-testable Programs," The Computer Journal, vol. 25, No. 4, pp.465-470, 1982.

[11] 우성희, 오병호, 이상호, "Estelle로 표현된 프로토콜의 테스트 스위트 생성에 관한 연구," 정보과학회 춘계 학술 발표 논문집, 21권 1호, pp. 463-466, 1994.

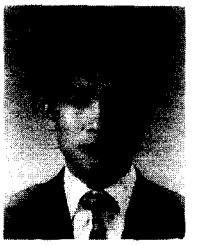
[12] 오병호, 우성희, 이상호, "조건 문맥 변수를 고려한 제어흐름도 생성 기법," 한국통신학회 논문지 20권 12호, pp.328-337, 1995.

[13] 우성희, 오병호, 이상호, "프로토콜 적합성 시험을 위한 테스트 스위트 검증 도구의 설계," 한국통신학회 논문지 20권 6호, pp.1619-1630, 1995.

[14] A.A.A.Loureiro, S.T.Chanson, S.T.Vuong, "FDT Tools for Protocol Development," Dept. of Computer Science University of British Columbia, Technical Report 91-5.

[15] B.Sarikaya, G.V.Bochman and E.Cerny, "A Test Design Methodology for Protocol Testing," IEEE Transaction on Software Engineering, Vol. SE-13, NO. 5, pp.518-531,1987.

[16] G.Luo, G.V.Bochman and A.Petrenko, "Test Selection Based on Communicating Nondeterministic Finite-State Machines using a Generalized Wp-Method," IEEE Transaction on Software Engineering, Vol. 20, pp. 149-162, 1994.



김 용 승

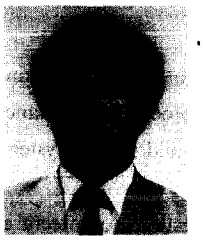
1980년 숭실대학교 전자계산학과 졸업(공학사)
1983년 숭실대학교 대학원 전자계산학과(공학석사)
1991년 숭실대학교 대학원 전자계산학과 박사과정 수료
1983년~1991년 8월 충주공업전문대 전자계산과 교수
1991년 9월~현재 청주전문대 교수
관심분야: 정보통신, 프로토콜 공학, 통신망 관리

우 성 희

1990년 청주대학교 전자계산학과 졸업(공학사)
1993년 충북대학교 대학원 전자계산학과(이학석사)
1995년 충북대학교 대학원 전자계산학과 박사과정 수료
1995년~현재 청주전문대 전자계산과 전임강사
관심분야: 프로토콜 공학, 네트워크, 소프트웨어 공학 등

오 병 호

1974년 공주교육대학 졸업(학사)
1991년 청주대학교 산업경영대학원 전자계산학과(이학석사)
1996년 충북대학교 대학원 전자계산학과(이학박사)
1985년~현재 충남전문대 전자계산과 교수
관심분야: 프로토콜 공학, 정보보안, 데이터 압축



이 상 호

1976년 숭실대학교 전자계산학과 졸업(공학사)
1981년 숭실대학교 대학원 전자계산학과(공학석사)
1989년 숭실대학교 대학원 전자계산학과(공학박사)
1981년~현재 충북대학교 컴퓨터과학과 교수
1994년~현재 충북대학교 전자계산소 소장
관심분야: 프로토콜 공학, 시뮬레이션, 소프트웨어 공학 등