論文 97-22-8-24

통신 프로토콜 시험항목의 오류 발견 능력 분석을 위한 시뮬레이터의 설계 및 구현

正會員 김 광 현*, 허 기 택**, 이 동 호***

Design and Implementation of simulator for fault coverage analysis of communication protocol test case

Gwang Hyun Kim*, Gi Taek Hur**, Dong Ho Lee*** *Regular Members*

요 약

본 논문은 유한 상태 기계 모델로 표현된 통신 프로토콜의 시험항목에 대한 오류 발견 능력 분석 방법을 제시한다. 시험항목에 대한 오류 발견 능력의 평가는 생성된 시험항목으로 어느 정도까지 오류를 발견해 낼 수 있는지를 측정하는 것이다. 시험항목의 오류 발견 능력 평가 방법은 주로 수학적 분석과 시뮬레이션을 이용한 방법이 사용되고 있다. 본 논문에서는 통신 프로토콜 시험항목의 오류 발견 능력 분석을 위해 시뮬레이터를 설계, 구현하였다.

Inres 프로토콜을 시뮬레이터에 적용한 결과, 출력 오류와 상태 병합·분리 오류는 100%의 높은 오류 발견율을 보였다. 구현된 오류 발견 능력 분석 시뮬레이터는 다양한 프로토콜에 적용 가능함으로써 새로운 오류 발견 능력 분석 도구로 사용될 수 있다.

ABSTRACT

In this paper, fault coverage analysis of a conformance test case for communication protocols, specified as a deterministic finite state machine(DFSM) is presented. The fault coverage analysis of a test case is defined by measuring the extent of the faults detected using a generated test case. The method that evaluates fault coverage analysis for a test case, has been researched by arithmetic analysis and simulation. In this paper, we designed and implemented a simulator for fault coverage analysis of a communication protocol test case.

With this result for Inres protocol, output fault and state merge and split fault have a high fault coverage of 100%. This simulator can be widely used with new fault coverage analysis tools by applying it to various protocols.

* 광주대학교 전자계산학과
** 동신대학교 컴퓨터학과
*** 광운대학교 전자계산학과
論文番號:97138-0428
接受日字:1997年 4月 29日

Ⅰ. 서 론

적합성 시험은 주어진 구현된 통신 프로토콜이 표준에 부합되게 만들어졌는지의 여부를 확인하는 시험으로써 정보통신 제품간의 상호 운용성을 확보하기 위해 사용되는 기본적인 시험이다. 정보통신 제품들간의 원활한 서비스 제공을 위해서는 정보통신 제품의 표준화, 적합성 시험 및 상호 운용성 보장이 필수적이다. 적합성 시험을 효율적으로 수행하기 위해서는 시험 스위트의 자동 생성과 시험 시스템에 적용하는 과정이 중요하다. 적합성 시험을 실제 적용하기 위해서는 시험기 및 시험 소프트웨어로 구성된 적합성 시험 시스템을 구축하여야 한다. 실제 시험기에 적용되는 시험 스위트는 각각의 행위를 규정하는 시험항목으로 구성된다. 시험항목의 생성에 관한 방법은 결정적 상태 기계 모델을 이용한 다양한 방법이 존재하고 있다[1, 2]. 지금까지 시험항목 생성에 관한 연구는 주로 시험항목의 크기를 최소화하는데 집중되어 왔다. 최근에는 최대의 오류 발견 능력을 갖는 시험항목 생성에 대한 많은 연구가 진행되고 있다[3, 4, 5].

시험항목 생성과 함께 생성된 시험항목을 구현 프로토콜에 적용하여 얼마나 많은 오류를 찾아낼 수 있는가를 평가하는 오류 발견 능력 분석이 필요하다. 시험항목의 오류 발견 방법은 표준 사양으로부터 생성된 시험 항목을 구현된 프로토콜에 입력 순서열을 적용하여 나타난 출력을 관찰하여 표준 사양으로부터 생성된 시험 항목의 출력과 비교함으로써 오류를 찾아내게 된다. 시험항목에 대한 오류 발견 능력의 평가 방법은 수학적 평가 방법과 시뮬레이션 방법이 주로 사용되고 있다[6, 7]. 본 논문에서는 시뮬레이션을 이용한 방법으로써, 네가지로 분류한 오류 모델에 적용할 수 있는 시뮬레이터를 설계, 구현하였다. 적용예로써 Inres 프로토콜에 대한 시험항목 생성과 오류 발견 능력 분석 결과를 제시한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 시험항목 생성 과정과 생성된 시험항목에 대한 오류 발견 능력 분석 방법을 살펴본다. 3장에서는 본 논문의 연구 결과로써 오류 발견 능력 분석을 위해 시뮬레이터를 설계하고 구현한다. 그리고 Inres 프로토콜에 대한 적용 결과를 제시함으로써 시뮬레이터의 이용 가능성을 보인다. 4장에서는 결론과 앞으로의 해결과제를 제시한다.

Ⅱ. 시험항목의 오류 검출 과정 및 오류 발견 능력 평가 방법

표준 프로토콜로부터 추상 시험 스위트를 생성하는 과정은 적합성 시험에서 매우 중요하다. 시험 스위트 생성을 위해 형식 사양으로부터 시험 목적에 부응하는 시험 스위트가 추출되며, 시험 목적은 적합성 시험 요구 사항의 일부라고 규정하고 있다. 프로토콜의 시험항목 생성시 구현에서 잘 발생할 수 있는 오류를 고려하여 시험항목을 생성하면 오류 검출 능력을 높일 수 있는 방법이 된다. 여기서는 적합성 시험의 기본이 되는 시험항목의 오류 검출 방법과 오류 발견 능력 평가 방법을 살펴보도록 한다.

2.1 시험항목의 오류 검출 과정

시험 사양으로부터 구현된 시험 스위트를 시험대상에 적용하고 그 결과를 분석하여 적합성 판정을 내리는 과정을 형식 기법의 관점으로 정의하고 있다[8, 9]. 가능한 구현들의 유형(IMPS : Implementations)은 적합성 시험에 통과한 I_{pass}와 적합성 시험에 실패한 I_{fail}로 나누어진다. 여기서 I_{pass}는 IMPS중에서 적합성 시험을 통과한 구현물을 말하고, I_{fail}는 IMPS에서 적합성 시험을 통과하지 못한 구현물을 의미한다. 시험항목 t(test case)는 실제로 시험기의 일부로서 구현되며, 시험기는 주어진 시험환경 C에서 시험대상 I(Implementation)와 함께 수행되므로 시험항목의 수행함수 exec은 아래와 같이 정의된다.

- exec(t, C[I])→{pass, fail, inconclusive}

여기서 수행함수 exec의 인수 t는 시험항목이고, C[I]는 적합성 시험환경에서 구현물을 적용하는 것을 의미한다. 시험항목의 수행과 관련한 함수는 시험항목을 시험대상에 적용하여 그 행위를 관찰하는 apply 함수와 관찰 결과를 바탕으로 판정을 내리는 verd 함수로 나눌 수 있다. 먼저 적용함수 apply는 아래와 같이 정의된다.

- apply(t, C[I])→OBS

여기서, OBS(OBServation)는 시험 환경에서 구현물의 관찰 상태를 나타낸다. verd 함수는 시험기에 시험항목을 적용하여 구현물을 통과하는지를 판단하는

함수이다. verd 함수는 다음과 같이 정의된다.

• verd(t, apply(t, C[I]))

그리고 적용함수 apply와 시험항목 수행함수 exec와의 관계는 다음과 같다.

• exec(t, C[I]) = verd(t, apply(t, C[I]))

시험 스위트의 수행은 그 시험 스위트에 속한 모든 시험항목들을 수행하는 과정이다. 따라서 시험 스위트의 수행에서는 개별적인 시험항목 수행 결과를 바탕으로 전체 시험 스위트에 대한 적합성 판정을 하게 된다. 즉, 시험 스위트 T(Test suite)가 시험대상 I에 시험환경 C에서 적용되었다면 시험 스위트에 대한 판정 부여 함수 verd_ts(T, C[I])는 다음과 같다.

$$\text{verd\_ts}(T, C[I]) = \begin{cases} \text{pass} & \text{if } \forall_t \in T : \text{exec}(t, C[I]) = \text{pass} \\ \text{fail} & \text{if } \exists_t \in T : \text{exec}(t, C[I]) = \text{fail} \end{cases}$$

주어진 사양에 적합한 구현 I는 시험환경 C에서 시험 스위트 T에 의해 fail 판정을 받을 수 없으며, 부적합한 구현이 pass 판정을 받을 수 없다. 이러한 관계는 다음과 같이 나타낼 수 있다.

• $C[I]$ passes $T \Leftrightarrow \text{verd\_ts}(T, C[I]) \neq \text{fail}$

2.2 오류 발견 능력 평가 방법

표준으로 정의한 프로토콜 사양의 구현 오류에 대한 오류 발견 능력의 정확한 측정은 불가능하지만 오류 발견 능력을 평가하기 위한 다양한 방법이 사용되고 있다. 시험

항목에 대한 오류 발견 능력이란 오류를 갖고 구현된 프로토콜이 생성된 시험항목으로 어느 정도까지 오류를 발견해 낼 수 있는가로 정의한다. 시험항목의 오류 발견 능력 평가 방법은 주로 수학적 평가 방법과 시뮬레이션을 사용한 연구가 이루어져 왔다. 시뮬레이션 기법을 사용하기 위해 오류의 형태를 적절하게 제한한 오류 모델과 함께 오류 발견 능력 평가 방법을 알아본다.

2.2.1 오류모델

실제 프로토콜을 시험할 때 구현된 프로토콜의 상태 수는 정확하게 알려져 있지 않다. 이러한 문제를 해결하기 위해서는 발생할 수 있는 오류의 형태를 적절하게 제한하는 방법을 사용한다[10, 11]. 이러한 오류는 구현과정에서 발생할 수 있다는 것으로 가정한다. 따라서 다양하게 발생할 수 있는 오류를 적절한 형태로 제한함으로써 효율적인 시험항목 생성과 오류 발견 능력 분석에 이용될 수 있게 된다. 오류 모델의 종류는 다음과 같이 분류할 수 있다.

(가) 출력 오류(Output faults)

출력 오류는 구현 프로토콜의 한 상태에 입력을 적용하여 발생되는 출력이 표준 프로토콜과 다른 출력을 발생시키는 오류이다. 즉, 결정적 유한 상태 기계(DFSM : deterministic finite state machine)로 표현된 에지 E가 사양과 다른 출력을 발생시키는 오류의 형태를 오류 모델 1로 정의한다.

(나) 천이 오류(Transfer faults)

구현 프로토콜의 어떤 상태의 입력에 대한 천이가 표준에서 정의하고 있는 것과 다른 상태로 천이가 발생하는 오류이다. DFSM으로 표현된 상태 i가 i'로 변경되면 상태 천이 오류가 발생했다고 하며 오류 모델 2로 정의한다.

(다) 상태 병합·분리 오류(State merge and split faults)

어떤 특정 상태와 입력에 대하여 하나 이상의 상태가 추가되거나 제거되는 경우가 발생하는 오류를 말한다. 상태가 두 개 이상으로 나누어지는 상태 분류 오류와 두 개 이상의 상태에서 하나로 합쳐지는 상태 병합 오류로 나눌 수 있다. 이러한 경우의 오류 형태를 오류 모델 3으로 정의한다.

(라) 혼합 오류(Hybrid faults)

혼합 오류는 오류가 발생할 때 상태나 에지의 한 부분에서만 발생하지 않고 이 두가지 형태의 오류가 동시에 발생하는 오류이다. 즉 출력 오류와 상태 천이 오류가 동시에 발생하는 오류이며 이를 오류 모델 4로 정의한다.

2.2.2 오류 발견 능력 평가 방법

시험항목의 오류 발견 능력을 평가함으로써 적합성 시험의 효율성을 평가하는 하나의 기준으로 사용될 수 있도록 한다. 평가 방법은 수학적 분석 방법과 시뮬레이션을 이용한 연구가 이루어져 왔다[12, 13].

(가) 수학적인 평가 방법

실질적으로 프로토콜을 시험할 때 구현된 프로토콜의 상태 수는 정확하게 알려져 있지 않기 때문에

생성된 시험 항목의 오류 발견 능력을 평가한다는 것은 매우 어려운 작업 중의 하나이다. 이는 기본적으로 시험항목의 수가 너무 많고, 프로토콜의 시험 상태 수가 많아질수록 시험 시간은 기하 급수적으로 증가하게 된다. 실질적으로 시험되어야 할 FSM(finite state machine)의 수가 매우 많기 때문에 모든 상태를 시험한다는 것은 거의 불가능하다. 실제로 n개의 상태, m개의 입력, p개의 출력 상태를 갖는 프로토콜이 있으면 $(np)^{mn}$ 개의 시험 상태가 존재하게 된다. 현실적으로 이렇게 많은 FSM의 상태를 시험한다는 것은 불가능하므로, 시험항목의 오류 발견 능력을 평가하기 위한 방법으로써 여러 조건들을 고려하여 수학적으로 분석하고 있다[14]. MS을 사양 머신이라 하고, TS는 테스트 스위트 그리고 **Impl**(m, Ms)는 사양을 구현한 것이라고 정의하면, 이것을 기본으로 하여 오류 발견 능력을 평가하기 위한 식은 다음과 같다.

$$FC(m, M_S, TS) = \frac{N_t(m, M_S) - N_p(m, M_S, TS)}{N_t(m, M_S) - N_c(m, M_S)}$$

- $N_t(m, M_S)$: **Impl**(m, MS)에 있는 머신의 총 수
- $N_c(m, M_S)$: M_S와 일치하는(conforming) **Impl**(m, M_S)에 있는 머신의 수
- $N_p(m, M_S, TS)$: 주어진 테스트 스위트를 통과한 **Impl**(m, M_S)에 있는 머신의 수
- $N_t(m, M_S) - N_c(m, M_S)$: M_S와 일치하지 않는(not-conforming) **Impl**(m, M_S)에 있는 머신의 수
- $N_t(m, M_S) - N_p(m, M_S, TS)$: 주어진 테스트 스위트를 통과하지 못하는 **Impl**(m, M_S)에 있는 머신의 수

위의 평가식에서 오류 발견 능력(FC: Fault Coverage)의 의미는 구현물 중에서 시험 스위트를 통과하지 못한 수와 원래의 머신(MS)과 일치하지 않은 머신 사이의 비율을 의미한다.

(나) 시뮬레이션에 의한 방법

일반적으로 프로토콜 적합성 시험은 가능한 상태 천이 모두를 시험하여야 완전한 테스트가 이루어졌다고 말할 수 있다. 그러나 현실적으로 모든 상태를 시험 항목으로 평가하는 것은 불가능함에 따라 수학적인 평가 방법과 함께 Monte Carlo 시뮬레이션 기법을 사용하여 오류 발견 능력을 평가하는데 사용되고 있다[13, 14]. 이 방법은 발생가능한 오류의 형태를 10개의 클래스로 한정하여, 임의의 오류를 갖는 FSM을 생성하여 시험항목이 오류를 발견해 내는 능력을 측정하는 방법이다. 오류 클래스의 분류는 시험하고자 하는 FSM에 대해 출력과 상태 천이가 변경되는 상황에 따라 분류한다. 첫째, 출력 오류는 상태 천이가 발생할 때 나타나는 출력을 변경하여 오류 기계를 생성한다. 두 번째의 상태 천이 오류는 상태 천이가 발생할 때 정상적인 천이가 아닌 다른 상태로 천이가 일어나도록 하여 오류 기계를 생성한다. 10개의 클래스로 분류하는 기준은 출력과 상태 천이 오류가 결합되어 하나 이상 발생하는 경우를 순서대로 분류한 것이다. 오류 모델에서 정의하고 있는 오류 형태를 알기 쉽게 표현하기 위해 화학적 표기법을 도입하면 오류의 종류를 이해하는데 상당히 편리하다. 즉 "O"는 출력이 변경된 것이고, "T"는 tail 상태의 변경, 그리고 "(OT)"는 출력과 tail 상태가 동시에 변경되었음을 나타낸다. 출력과 천이 오류를 적절하게 조합하여 10개의 클래스로 분류하여 순서대로 표기하면 다음과 같다.

클래스1(O_1): FSM으로부터 임의의 에지의 출력을 변경한다.

클래스2(T_1): 임의의 한 에지의 tail 상태를 변경한 것을 제외하고는 클래스 1과 동일하다.

클래스3(O_2): FSM에서 임의의 두 에지의 출력을 변경한다.

클래스4(T_2): 임의의 두 에지의 tail 상태가 변경을 제외하고는 클래스 3과 동일하다.

클래스5(O_1T_1): FSM에서 임의의 한 에지의 tail 상태와 다른 에지의 출력만 변경한다.

클래스6($(O_1T_1)_2$): FSM과 다른 임의의 에지와 tail 상태만을 변경한다.

클래스7($(OT)_2$): FSM 임의의 두 에지의 tail 상태와 출력만 변경한다.

클래스8(O_2T_1): 임의의 한 에지의 tail 상태와 다른 두 에지의 출력을 변경한다.

클래스9(O_2T_2): 임의의 두 에지의 tail 상태와 다른 두 에지의 출력을 변경한다.

클래스10(O_2T_3): 임의의 세 에지의 tail 상태와 다른 두 에지의 출력을 변경한다.

이렇게 오류의 형태를 적절하게 제한하는 이유는 생성 가능한 모든 구현 머신을 시험하는 것은 불가능하기 때문이다[15]. 앞에서 정의한 오류를 갖는 임의의 FSM을 사용하여 생성된 시험항목으로 오류를 찾아내기 위한 절차는 다음과 같다.

1) FSM 사양을 읽어 들인다.

2) 사용될 시험항목을 읽어 들인다.

3) 위에서 정의한 10개의 클래스에 대하여 오류 FSM을 생성한다.

4) 단계 3에서 생성된 FSM에 시험항목을 적용한다.

5) 단계 4에서 테스트를 통과한 FSM은 오류가 없는 것으로 분류되고, 통과하지 못한 FSM은 오류가 있는 것으로 판정한다.

마지막 단계에서 시험 스위트를 통과한 FSM과 통과하지 못한 FSM을 분류하여 시험항목의 오류 발견 능력을 분석하게 된다. 오류 생성기에 의해 임의로 생성된 FSM에 표준 프로토콜 사양으로부터 생성된 시험항목을 적용하여 출력이 동일하게 되면 오류가 존재하지 않는다고 가정하고, 출력이 서로 다르면 FSM에 오류가 존재하고 있다고 정의한다. 또한 임의로 생성된 FSM에서 상당수는 오류 머신이 아닐 수 있으므로 오류 발견 능력의 평가는 실제 오류에 대하여 발견된 오류 비율로 표시할 수 있다.

본 논문에서는 시험항목의 오류 발견 능력 분석 방법으로 시뮬레이션 방법을 사용하여 시뮬레이터를 설계하고 구현하였다. 시뮬레이션 방법을 사용한 이유는 임의의 오류를 갖는 FSM을 수학적으로 분석하는 방법은 쉽지 않기 때문이다. 따라서 시험 목적을 반영한 시험항목으로 임의의 FSM에 대한 오류 발견 능력 분석 방법은 시뮬레이션 방법이 적당하다.

Ⅲ. 오류 발견 능력 분석 시뮬레이터의 설계 및 구현

지금까지 구현된 도구들은 주로 적합성 시험의 전 과정에 이용될 수 있는 도구의 개발에 치중하였다[16]. 따라서 시험항목의 오류 발견 능력을 평가하는 측면에서는 상당히 부족하였다. 본 시뮬레이터는 이러한 점을 고려하여 시험항목의 생성과 함께 시험항목의 오류 발견 능력을 평가하는 도구로 사용될 수 있도록 하였다. 본 논문에서 구현한 시뮬레이터는 오류 모델에 따라 시험항목의 오류 발견 능력 분석이 가능하도록 구현하였다. 따라서 오류 발견 능력 분석 시뮬레이터는 다양한 프로토콜에 적용함으로써 새로운 오류 발견 능력 분석 도구로 사용될 수 있다. 이 장에서는 시뮬레이터의 구현 환경과 구현 모듈을 살펴보고, 구현한 실행 화면을 보여 주고, 실제 프로토콜에 대한 적용 사례를 통해 시뮬레이터의 유용성을 보인다.

3.1 구현 환경 및 시뮬레이터의 구조

본 논문의 오류 발견 능력 분석 시뮬레이터는 SunOS 5.5에서 C언어를 사용하여 구현하였고, 사용자 인터페이스는 Tcl/Tk(tool command language/tool kit)를 이용하여 구현하였다. 시뮬레이터의 기본 구성은 FDT (formal description technique)를 사용하여 프로토콜 사양을 FSM 모델로 표현하도록 한다. 시뮬레이터는 기본적으로 FSM으로 표현된 프로토콜을 이용하여 시험항목을 생성하는 부분과 이를 오류 FSM에 적용하여 오류 발견 능력을 분석하는 부분으로 구성하였다. 본 시뮬레이터의 수행 과정은 형식 기술 기법을

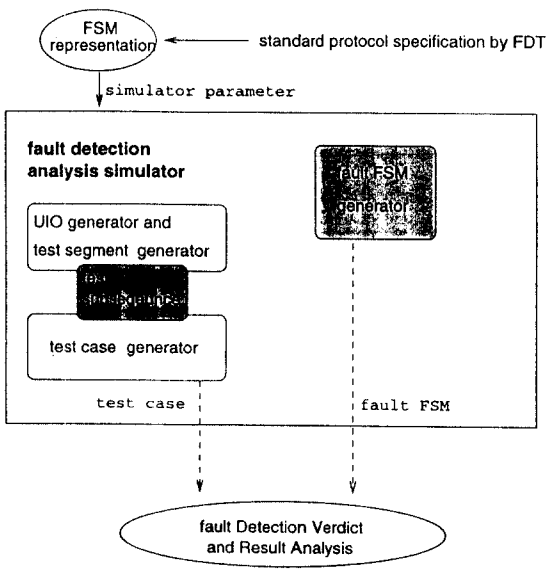

그림 3.1 오류 발견 능력 분석 시뮬레이터

사용하여 기술된 프로토콜을 FSM 형태로 변환하고 UIO 기법을 사용하여 시험항목을 생성한다. 여러 개의 UIO가 존재할 때 최소 길이의 UIO를 선택하여 시험항목을 생성하도록 하였다. 그리고 최종적으로 오류 모델에 따라 생성된 오류 FSM에 적용하여 어느 정도의 오류를 발견하는지를 평가한다. 본 논문에서 구현한 시뮬레이터의 개괄적인 구조는 그림 3.1과 같다.

3.2 시뮬레이터의 구현 모듈 및 실행 과정

일반적으로 프로토콜의 한 계층은 프로세스의 모듈로 구현되며, 계층간의 통신은 FIFO 큐 또는 메일박스 등을 통하여 이루어진다. 구현된 시뮬레이터에서는 시험항목이 오류 FSM에 모델별로 적용 가능하도록 하였다. 시뮬레이터에서 구현에서 프로그램의 주요 기능과 알고리즘을 간략하게 살펴봄으로써 어떻게 시험항목을 생성하고 오류를 발견하는지를 알 수 있다. 그림 3.2는 본 논문에서 구현한 시뮬레이터의 초기 실행 화면이다. 사용자 인터페이스는 Tcl/Tk를 이용하여 구현하였다. 시뮬레이터 초기 화면 구성을 보면 FDT를 이용하여 프로토콜을 FSM 모델로 표현한다. 이렇게 FSM으로 표현된 프로토콜을 연결된 리스트 구조로 입력하여 UIO와 시험항목을 생성한다. 생성된 시험항목을 오류 FSM에 적용하여 시험항목의 오류 발견율을 분석하는 단계로 구성되어 있다.

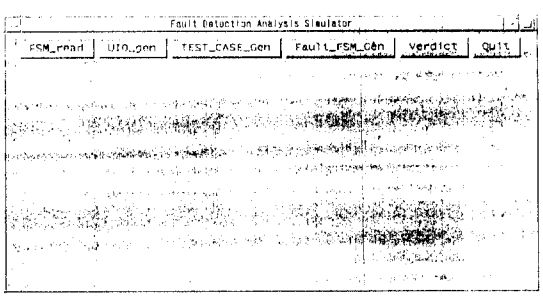

그림 3.2 시뮬레이터의 초기 실행 화면

시험항목 생성을 위한 다음 과정은 생성된 UIO를 사용하여 각 천이에 대한 테스트 세그먼트를 구하고 천이의 시험을 위한 test subsequence를 구한다. 각 상태에 대해 생성된 UIO를 사용하여 test segment를 생성하고 이를 이용하여 시험항목을 생성하는 과정이다. 알고리즘 3.1는 시험항목 생성을 위해 최단거리 알고리즘을 적용하여 subsequence를 구한 다음, 이들을 서로 결합하여 시험항목을 생성하는 과정이다. 시험항목의 길이를 최소화하는 방법은 overlapping을 사용하였다.

알고리즘 3.1(test case generation)

```
makepath(source s, destination d, s에서 d로의 shortest
path 저장, 배열 path, path tree 내에서의 각 상태
들의 level)
{
if(source와 destination이 같으면) return 1;
  {/* root를 s로 하여 구축된 shortest path tree를
  이용한다. */
  for(shortest path tree에서 s로부터 나갈 수 있는
  모든 노드를 조사)
  {
  나갈 수 있는 길이 있으면 level이 증가한 값으로
  makepath를 recursive 호출;호출된 makepath에
  서 d를 찾았으면 현재 노드에 해당하는 path 위
  치에 s 저장;상위 프로그램에 path를 찾았음을
  알린다(return 1);
    }
  여기에 도달한 것을 찾지 못했을 경우이므로 re-
  turn 0;
  }
source와 destination이 같으면 d의 level에 해당하
는 path 내 위치에 d 삽입;
return 1;
  }
char *spath(source s, destination d)
  {
  tree 구조를 clear 한다;
  root를 s로 하는 shortest path tree를 구성한다. /
  * 모든 edge는 동등 */
  s에서 d로의 path를 저장할 path 배열을 할당;
  /* 노드의 level은 shortest path tree에서의 level
  이다. */
 makepath(s, d, path, 각 node의 level이 저장된 배열);
  }
```

그림 3.3은 프로토콜이 FSM으로 표현되었을 때 시

뮬레이터가 자동적으로 FSM을 읽어 들이는 화면이다. 입력은 상태에 해당되는 노드와 각 상태에서 천이되는 입출력 레이블을 배열로 정의하여 입력 데이타로 사용하였다.

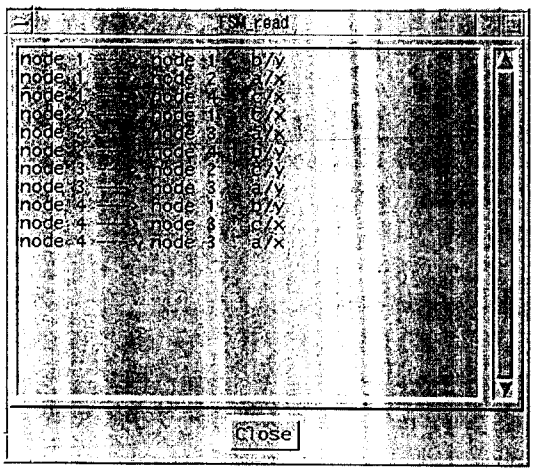

그림 3.3 시뮬레이터의 FSM Reader 화면

오류 FSM의 생성은 오류 모델에 따라 다르게 생성된다. 각 오류 모델별로 100,000개의 오류 FSM을 생성하였다. 알고리즘 3.2는 시험항목을 오류 모델별로 생성된 오류 FSM에 적용하여 시험항목이 오류를 발견하는지를 판단하도록 한다.

알고리즘 3.2(fault detection and verdict)
```
float TestFaultModel(조사할 fault model)
 {for(test 횟수){
   시험항목에 적합한 오류 FSM 구성;
   오류 FSM에 대한 uio 생성;
   for(오류 FSM에 내의 모든 천이(s→d)에 대하여)
     {초기 상태에서 s까지의 shortest path 조사;
       s→d의 입출력 조사;
       d의 UIO 검사;
       }
     }
   }
```

그림 3.4는 시뮬레이터 실행의 최종 단계로 오류 FSM에 대한 오류 발견 과정을 보여준다. 본 논문에서는 시뮬레이터의 오류 FSM 생성 부분에서 각 오류 모델 별로 100,000개의 오류 FSM을 생성하였다. 이 단계에서는 시험항목의 출력과 생성된 FSM의 출력과 비교하여 동일하면 오류가 없다고 판정하고 동일하지 않으면 오류가를 발견했다고 판정한다.

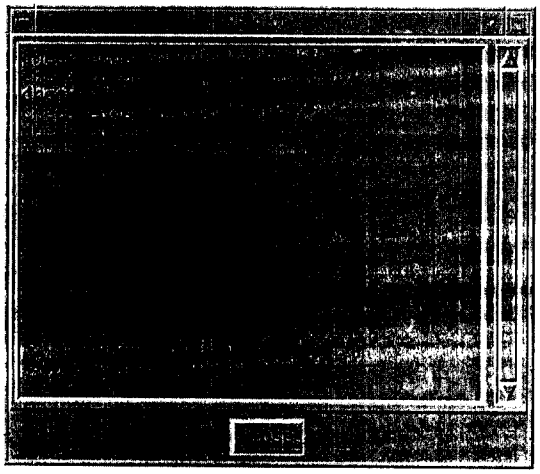

그림 3.4 시뮬레이터의 오류 판정 화면

3.3 적용 사례 및 결과 분석

시뮬레이터를 이용하여 시험항목이 어느 정도 오류 발견 능력을 갖는지를 실질적으로 평가하기 위해 실제 프로토콜에 적용하도록 한다. 기본적으로 적용될 프로토콜은 FSM으로 표현 가능한 형태로 변환한다. 원래의 프로토콜 규격을 완전하게 FSM으로 변환하는 것은 불가능하지만 현실적으로 시뮬레이터에 적용 가능성을 고려하면 FSM으로 표현하는 것이 하나의 방법이 된다. 본 논문에서도 프로토콜을 FSM으로 표현하여 시뮬레이터에 대한 입력으로 사용한다. 오류 발견 능력 분석 시뮬레이터를 이용하여 오류 모델 4가지에 대한 오류 발견 능력을 분석한다. 분석 대상 프로토콜은 OSI(Open System Interconnection) 프로토콜과 비슷한 Inres 프로토콜에 대한 적용 예를 보인다.

3.3.1 Inres 프로토콜

Inres 프로토콜은 OSI의 기본 개념을 다수 포함하고 있는 가상의 프로토콜로 이해하기 쉽고 크기가 크

지 않으므로 여러 연구에서 참조용으로 널리 쓰이고 있는 프로토콜이다. Inres 프로토콜의 구조는 그림 3.5와 같다.

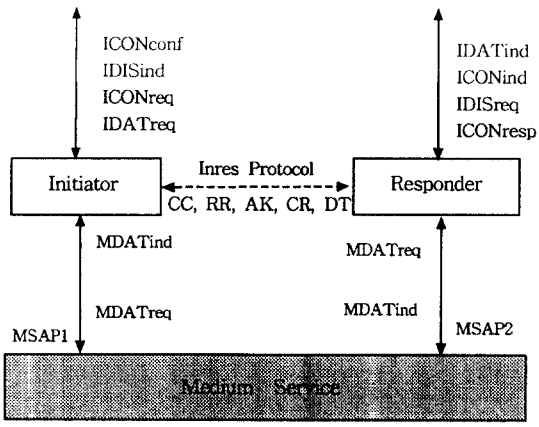

그림 3.5 Inres 프로토콜 구조

Inres 시스템은 두 서비스와 하나의 프로토콜로 구성된다.

- Medium service : data unit의 신뢰성 없는 전송에 사용된다.
- Inres Protocol : medium service를 이용하여 사용자에게 연결 위주의 서비스를 제공한다.
- Inres Service : Inres 프로토콜과 Medium Inres 서비스에 의해 제공된다.

Inres 시스템은 세 부분으로 구성되어 있으며, 이 중에서 Inres 프로토콜은 Medium 서비스를 이용하여 상위 계층 가상의 사용자에게 Inres 서비스를 제공한다. Inres 프로토콜은 OSI 참조 모델의 어떤 특정 계층에도 해당되지 않으며 일반적 프로토콜의 한 계층을 모델링한 것이다. Inres 프로토콜은 Initiator와 Reponder, 두 개체간에 동작되는 연결 위주의 프로토콜이다. 이 두 개체는 CR, CC, DT, AK, DR의 5가지 PDU를 교환하면서 통신한다. Inres 프로토콜은 다음 3단계 과정으로 구성된다.

(1)연결 설정 단계(Connection Establishment Phase) :연결 설정은 Initiator 사용자가 Initiator 개체에게 ICONreq를 보냄으로써 시작된다. Initiator 개체는 Responder 개체에게 CR을 보낸다. Responder는 CC나 DR로 응답한다. CC의 경우 Initiator가 사용자에게 ICONconf를 보내고 데이터 전송 단계로 진입한다.

(2)데이타 전송 단계 (Data Transmission Phase): Initiator 사용자로부터 IDATreq를 받으면, Initiator는 Responder에게 DT를 보낸다. 그리고 다시 IDATreq를 받을 준비를 한다. Responder는 CR을 받으면 연결 설정 단계로 들어가며, IDISreq를 받으면 연결 해제 단계로 들어간다.

(3)연결 해제 단계(Disconnection Phase): Responder 사용자로부터 IDISreq를 받으면 Responder는 DR를 보내고 새로운 Initiator로부터 CR을 받을 준비를 한다. Initiator 측에서는 DR을 받으면 사용자에게 IDISind를 보낸다. IDISind는 DT나 CR을 성공적으로 전송하지 못한 경우에도 사용자에게 보내진다. 이 때부터 새로운 연결이 시도될 수 있다. 앞의 3단계 중 제어부에 해당되는 연결 설정 단계와 연결 해제 단계에 대한 부분을 FSM으로 표현하여 시험하도록 한다. 그림 3.6은 Inres 프로토콜의 제어 부분을 표시하고 있다.

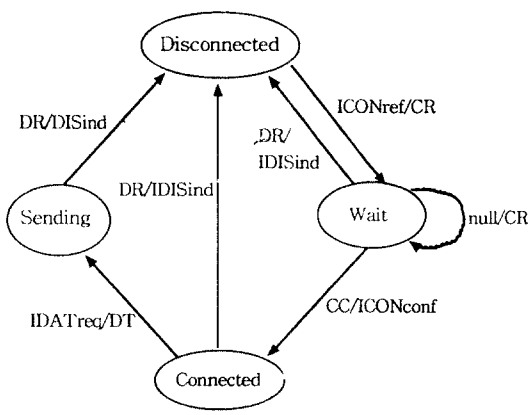

그림 3.6 Inres 프로토콜의 제어 부분

Inres 프로토콜에 대한 시험항목을 생성하기 위해 먼저 시뮬레이터의 UIO 생성기에 의해 UIO 순서를 생성한다. 천이 오류와 출력 오류를 확인하기 위해 테스트 세그먼트가 필요하다. 테스트 세그먼트는 UIO순서와 천이를 결합함으로써 생성된다. 최종적으로 시험항목을 생성하기 위해서는 테스트 세그먼트를 이용한다. 초기 상태에서 시험하고자 하는 상태까지의 최단거리를 구해 테스트 세그먼트와 결합한

다. 이렇게 결합된 테스트 세그먼트를 test subsequence라고 한다. 생성된 test subsequence를 적절하게 결합하여 중복된 부분을 제거하게 되면, 최종적으로 시험항목이 생성된다.

3.3.2 결과 분석

오류 모델 4가지에 대한 오류 발견 능력을 평가하기 위해 시뮬레이터에 의해 생성된 시험항목을 각각의 오류 모델에 적용하였다. 표 3.1은 Inres 프로토콜의 오류 발견 능력 분석 결과를 보여 준다. 적용 과정을 살펴보면 표준 프로토콜 규격으로부터 시험항목을 생성한다. 그리고 시뮬레이터를 이용하여 생성된 시험항목을 오류 모델에 따라 생성된 오류 FSM에 적용하여 오류 발견 여부를 판정한다. 시험항목을 오류 FSM에 끝까지 적용할 때까지 출력 값이 동일하면 미발견 오류에 포함시킨다. 오류 FSM 중간에 오류가 발견되면 시험을 중단하여 시험 수행 시간을 절약하는 효과를 갖도록 하였다. 각 오류 모델별로 오류 FSM은 여러 가능성을 고려하여 100,000개를 임의적으로 생성하였다. 이 정도의 오류 FSM은 발생 가능한 오류를 충분하게 포함한다. 실제로 Inres는 상태와 입출력이 많지 않은 간단한 프로토콜이다. 따라서 100,000개의 오류 FSM중에는 중복되어 생성된 다수의 오류 FSM을 포함하고 있다. 중복된 오류 FSM에 대해 시험항목을 적용해도 오류 발견율에는 차이가 없다. 시뮬레이션 결과를 보면 오류 모델 1과 3은 100%의 오류 발견율을 보이고 있다. 그러나 오류 모델 4는 오류 모델 2에 비해 오류 발견율이 다소 떨어지고 있음을 알 수 있다. 이는 출력 오류와 천이 오류가 상호 결합되어 미발견 오류가 증가되기 때문이다.

표 3.1 Inres 프로토콜의 오류 발견 능력 분석 결과

Fault model	Fault FSM	Detected FSM	Fault Coverage(%)
1	100,000	100,000	100.00
2	100,000	99,960	99.96
3	100,000	100,000	100.00
4	100,000	98,661	98.66

Ⅳ. 결 론

적합성 시험이 효율적으로 이루어지기 위해서는 시험항목이 효율적으로 생성되어야 하며 시험항목의 오류 발견 능력이 뛰어나야 한다. 본 논문에서는 발생 가능한 오류를 네 가지 형태의 오류 모델을 정의하였다. 그리고 생성된 시험항목이 어느 정도 오류 발견 능력을 갖는지를 평가하기 위해 시뮬레이터를 설계, 구현하였다. 또한 본 논문에서 구현한 오류 발견 능력 시뮬레이터는 다양한 프로토콜에 적용함으로써 새로운 오류 발견 능력 분석 도구로 사용될 수 있을 것이다. 본 시뮬레이터를 통합 환경에 적용할 수 있도록 확장하고, 실제 응용 프로토콜의 적합성 시험에 대한 효율성을 평가하는 새로운 기준을 제시하는 연구가 필요하다.

참 고 문 헌

1. D. Lee, K. Sabnani, D. M. Kristol, S. Paul, "Conformance testing of protocols specified as communicating FSMs", IEEE INFOCOM'93, pp. 115-127, 1993.
2. W. J. Chun, "Test case generation for protocols specified in estelle", *Ph. D Thesis, Computer and Information Science*, University of Delaware, 1992.
3. J. Zhu and S. T. Chanson, "Fault coverage evaluation of protocol test sequence", *Proceeding of the 12th IFIP Symposium on Protocol Specification, Testing and Verification*, Canada, 1994.
4. G. v. Bochmann, A. Petrenko and M. Yao, "Fault coverage of tests based on finite state models", *IWPTS VII*, pp. 55-74, 1994.
5. M. Yao, A. Petrenko and G. v. Bochmann, "A structural analysis approach to the evaluation of fault coverage for protocol conformance testing", *FORTE'94*, pp. 389-404, 1994.
6. H. Mottler, A. Chung and D. Sidhu, "Fault coverage of UIO-based methods for protocol testing", *Protocol Test System, IFIP*, 1994.
7. F. Lombardi and Y. U. Shen, "Evaluation and improvement of fault coverage of conformance testing by UIO sequences," *IEEE Trans. Communications*, vol. 40, no. 8, pp. 1288-1293, August 1992.

8. K. Naik and B. Sarikaya, "Testing Communication Protocols", IEEE Software pp. 27-37, January 1992.
9. A. R. Cavalli, J. P. Favreau and M. Phalippou, "Standardization of formal methods in conformance testing of communication protocols", *Computer Networks and ISDN Systems*, vol. 29, no. 1, pp. 3-14, 1996.
10. D. P. Sidhu, H. Motteler and R. Vallurupalli, "On testing hierarchies for protocols", *IEEE/ACM Trans. Networking*, vol. 1, no. 5, pp. October 1993.
11. R. E. Miller and S. Paul, "Structural analysis of protocol specifications and generation of maximal fault coverage conformance test sequences", *IEEE/ACM Trans. Networking*, vol. 2, no. 5, pp. 457-470, 1994.
12. T. Ramalingam, A. Das and K. Thulasiraman, "Fault detection and diagnosis capabilities of test sequence selection methods based on the FSM model", 24
Computer Communications, vol. 18, no. 2, pp. 113-122, February 1995.
13. T. Ramalingam, A. Das and K. Thulasiraman, "On testing and diagnosis of communication protocols based on the FSM model", *Computer Communications*, vol. 18, no. 5, pp. 329-337, May 1995.
14. A. Petrenko, G. v. Bochmann and M. Yao, "On fault coverage of tests for finite state specifications", *Computer Networks and ISDN Systems*, vol. 29, no. 1, pp. 81-106, 1996.
15. 김광현, 허기택, 이동호 "통신 프로토콜 시험항목의 오류 발견 능력 평가 방법", *한국 통신학회 논문지*, 21권, 8호, 1948-1957(쪽), 1996.
16. 김재철, 최양희, "프로토콜 시험을 위한 통합환경의 설계와 구현", *한국정보과학회 논문지*, 22권, 12호, 1695-1704(쪽), 1995.

김 광 현(Gwang Hyun Kim) 정회원
1989년 2월 : 광운대학교 전자계산학과 졸업(이학사)
1991년 2월 : 광운대학교 대학원 전자계산학과 졸업(이학석사)
1997년 2월 : 광운대학교 대학원 전자계산학과 졸업(이학박사)
1997년 3월~현재 : 광주대학교 전자계산학과 전임강사
※주관심 분야 : 프로토콜 공학, 컴퓨터 네트워크, 초고속 통신망

허 기 택(Gi Taek Hur) 정회원
동신대학교 컴퓨터학과 부교수, 한국통신학회 논문지 제20권 3호 참조

이 동 호(Dong Ho Lee) 정회원
광운대학교 전자계산학과 교수, 한국통신학회 논문지 제20권 3호 참조