Journal of Korea Multimedia Society Vol. 8, No. 8, August 2005(pp.1119-1133)

통신 프로토콜 검정기 및 적합성시험 도구 개발

서미선[†], 황종규[††], 이재호[†††], 김성운[††††]

요 약

프로토콜 명세의 검정과 적합성시험은 프로토콜 개발 과정에서 가장 중요한 부분으로, 명세에 규정되어진 시스템 기능의 정확성을 향상시키는데 사용되는 상호 보완 기술이다. 본 논문에서는 LTS(Labeled Transition System)로 명세화된 프로토콜 모델의 안전성 및 필연성 특성을 모형검사 기법에 의해 검정하였고, 실제적으로 교착상태의 유무나 초기 상태에서 임의의 상태로 도달 가능한지에 대한 검사를 실험적으로 증명하는 도구를 구현하였다. 구현된 프로토콜 검정기는 modal mu-calculus를 사용하여 modal 논리로 표현된 특성이 명세에 대해 올바른지 아닌지를 검정할 수 있다. 또한 검정되어진 프로토콜 명세로부터 UIO(Unique Input Output) 방법에 의해 유일한 입출력열을 이용하여 이끌어낸 결과 시퀀스가 구현에 대해 올바른지를 검사하는 적합성시험 계열 생성도구를 개발하였으며, 개발된 도구는 Windows NT 환경하에서 C++언어를 이용하여 구현되었다.

Development of Verification and Conformance Testing Tools for Communication Protocol

Mi-Seon Seo[†], Jong-Gyu Hwang[††], Jae-Ho Lee[†††], Sung-Un Kim[††††]

ABSTRACT

As a very important part in development of the protocol, verification and conformance test for protocol specification are complementary techniques that are used to increase the level of confidence in the system functions as prescribed by their specifications. In this paper, we verify the safety and liveness properties of rail signal control protocol type 1 specified in LTS with model checking method, and experimentally prove that it is possible to check for the deadlock, livelock and rechability of the states and actions on LTS. The implemented formal checker is able to verify whether properties expressed in modal logic are true in specifications using modal mu-calculus. We also propose a formal method on generation of conformance test cases using the concept of UIO sequences from verified protocol specification. The suggested tools are implemented by C++ language under Windows NT.

Key words: Verification(검정), Conformance Testing(적합성 시험), Modal Mu-Calculus, Conformance Test Case(적합성 시험 계열)

1. 서 론

통신 소프트웨어에 대한 사용자의 요구사항이 복잡화, 다양화, 대형화되어짐에 따라 개발에 따르는 어려움은 더욱 증대되었고, 과거에 사용된 비정형적 방법(informal method)에 의한 개발은 많은 오류와

※ 교신저자(Corresponding Author) : 김성운, 주소 : 부산광역시 남구 대연3동 599-1번지(608-737), 전화 : 051)620-6476, FAX : 051)620-6450, E-mail : kimsu@pknu.ac.kr
접수일 : 2004년 8월 16일, 완료일 : 2005년 1월 18일
[†] 준회원, 부경대학교 정보통신공학과
(E-mail : jljpms@mail1.pknu.ac.kr)
[††] 정회원, 한국철도기술연구원 전기신호연구본부 선임연구원
(E-mail : jghwang@krri.re.kr)
[†††] 정회원, 한국철도기술연구원 전기신호연구본부 책임연구원
(E-mail : jhlee1@krri.re.kr)
[††††] 종신회원, 부경대학교 정보통신공학과
※ 이 논문은 2003년도 두뇌한국 21사업에 의해 지원되었음.

비효율성을 내포하고 있다. 따라서 개발에 소요되는 비용 및 시간을 절약하고 심각한 오류를 형식적 방법에 의해 검출하는 기술인 정형 기법(formal method)의 적용이 증가하는 추세이며, 특히 정형 기법은 검정, 구현 및 시험 시에 전통적 자연어에 근거한 프로토콜 개발에 비하여 더 체계적인 방법을 제공하는 장점이 있다[1].

정보통신망을 위한 프로토콜들이 상호 적절한 기능을 하기 위해서는 잠재적 설계 에러가 없어야 하며, 사용자 요구사항과 일치하고 다른 프로세서와의 통신이 원활하게 이루어져야 한다. 따라서, 하나의 새로운 정보통신 시스템을 구현하기 위해서는 그림 1과 같이 사용자 요구사항 분석, 프로토콜의 구조적 설계 및 명세화, 검정, 구현, 적합성시험 등의 단계를 거쳐야 하고, 그 중 검정 및 적합성시험은 가장 핵심이 되는 부분이기 때문에 더욱 정확하고 안정적인 수행이 필수적이다.

사용자 요구사항과 명세와의 일치성을 구현 전에 확인하는 검정 단계는 모든 프로토콜에 필수적인 정확성(correctness) 특성을 만족하는가에 대해 프로토콜 명세를 분석하는 과정이다. 모형검사(model checking)는 이러한 과정을 자동적, 형식적으로 검정하는 기술이며, 일반적으로 유한 상태 레이블 천이 시스템(LTS)으로 모델링된 concurrent 시스템 명세의 올바름을 검정하기 위한 모델 검정에 시제논리에 기반을 둔 CTL(Computation Tree Logic)이 많이 사용되어 왔으나 해당 검정 알고리즘의 구현 및 적용 과정에서 시스템 내부 병렬 프로세스간의 상호작용

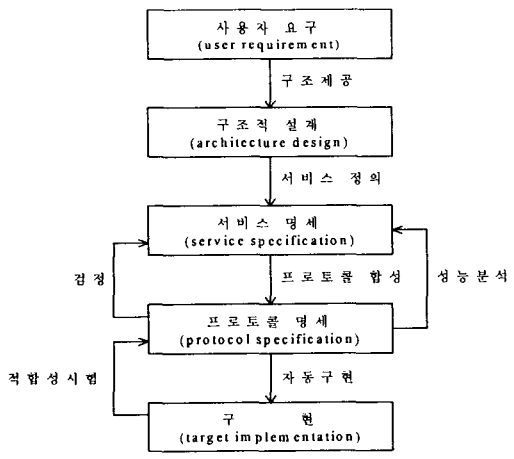


그림 1. 프로토콜 개발 단계

으로 인한 시스템 요소의 증가에 따라 상태가 기하급수적으로 증가하는 상태폭발 문제가 제기되어 왔다. 그러나 modal mu-calculus 논리를 시스템 안전성 및 필연성 특성 명세에 사용하면, 행위에 의한 순환적 정의가 가능하므로 상태폭발 문제가 해결될 수 있다. 프로토콜 표준으로부터 관련 제품 구현에 있어 적합성시험은 구현 제품의 프로토콜 표준에 대한 구현의 올바름을 검정하는 시험으로, 프로토콜 명세로부터 시험 계열을 생성하여 구현에 적용함으로써 적합성 여부를 판단하는 일련의 과정이다[2,3].

본 논문에서는 LTS로 명세화된 프로토콜의 안전성 및 필연성 특성을 모형 검사 기법에 의해 검정하기 위해 modal mu-calculus를 사용하는 검정기와 검정되어진 프로토콜 명세로부터 UIO 시퀀스 기법에 의해 적합성시험 계열을 자동으로 생성하는 도구를 구현한다. 구현된 도구의 동작 성능의 정확성 측정을 위해 철도청에서 개발한 철도 신호제어 프로토콜 type1을 사용한다.

2장에서는 검정 및 시험 대상인 철도 신호제어 프로토콜을 기술, 그 동작과정을 LTS와 I/O FSM (Input/Output Finite State Machine)으로 모델링한다. 3장에서는 대상 프로토콜의 특성 만족여부를 판단하기 위해 modal mu-calculus와 모형검사 알고리즘인 solve 알고리즘을 적용하여 검정하는 방법 및 실제 구현된 프로토콜 검정기의 프로시져를 기술한다. 4장에서는 적합성시험의 정의 및 생성된 I/O FSM 중간 모델을 대상으로 UIO 시퀀스 생성 및 시험 계열 생성 방법, 그리고 구현된 시험기의 프로시져를 기술하고, 마지막으로 5장에서 본 논문의 결론 및 향후 추진사항에 대해 기술한다.

2. 철도 프로토콜 정의 및 시스템 명세 모델

본 장에서는 철도 신호제어 프로토콜 type 1을 기술하고 해당 프로토콜의 동작과정을 중간모델인 LTS와 I/O FSM으로 모델링한다.

2.1 정의

본 논문의 검정 및 시험 대상은 철도 신호제어 프로토콜 type 1으로, 역정보전송장치(LDTS:Local Data Transmission System)와 전자연동장치(EIS : Electronic Interlocking System) 간의 정보 전송 방

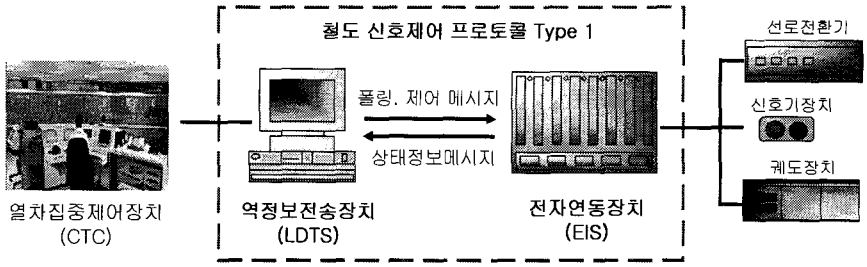


그림 2. 철도 신호 제어 프로토콜 Type 1

식이다. LDTS는 열차집중제어장치(CTC : Centralized Traffic Control)로부터의 제어 명령을 EIS에게 전송하고, EIS는 현장신호설비를 제어 및 감시하여 상태정보 메시지를 LDTS에게 전송함으로써 상호간 통신한다.

2.2 Type 1의 LTS 모델링

1) LTS의 정의

LTS는 프로토콜을 검정하기 위해 프로토콜 정형명세를 위한 의미 모델로서 많이 사용되며 다음과 같이 정의한다[4].

<정의 1> 레이블 천이 시스템(LTS : Labeled Transition System)

LTS는 다음 4개의 요소로 구성된 레이블화 된 천이시스템 LTS = <S, L, T, s_o>으로 정의된다.

- S : 상태들의 집합(a set of states)
- L : 관찰 가능한 행위집합(a set of observable actions)
- $T \subseteq S \times (L+\tau) \times S$: 천이 관계(transition relation)
- $s_o \in S$: 초기 스테이트(initial state)

이 정의에서 심볼 "τ"는 비결정형 모델(non-determinant model)을 위해 이용되는 시스템 내부 혹은 관찰이 가능하지 않은 행위(internal 혹은 in-observable action)를 나타낸다.

2) LTS 모델링

철도 프로토콜 Type 1을 LTS로 모델링한 것은 그림 3과 같고, 표 1과 2는 각각 그림 3의 LTS 모델의 상태와 행위에 대한 설명이다.

2.3 Type 1의 I/O FSM 모델링과 동작분석

1) I/O FSM의 정의

일반적으로 I/O FSM은 프로토콜의 송수신간 제어 특성을 명세화 하는데 많이 사용되어져 왔으며

표 1. LTS 모델의 상태 설명

상태	상태설명
S_0	LDTS와 EIS사이의 통신이 시작되기 전 상태
S_1	Master Clock이 런타임에 이른 상태
S_2	응답을 기다리는 상태
S_3	열차가 이동한 상태
S_4	Polling timer가 런타임에 이른 상태
S_5	응답을 기다리는 상태

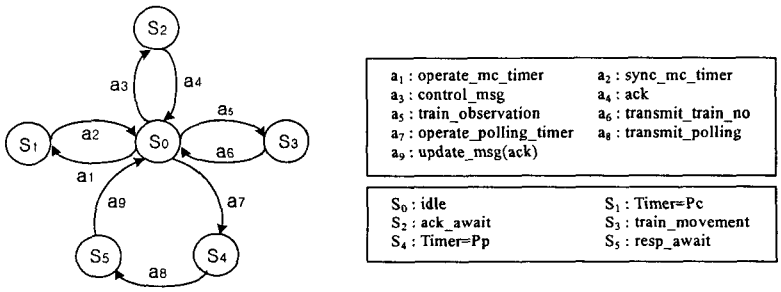


그림 3. LDTS에 대한 LTS 모델링

표 2. LTS 모델의 행위 설명

행위	행위 설명
a_1	Master Clock Timer 작동
a_2	Master Clock을 전송하여 정보기록시각을 맞춤
a_3	제어 메시지 전송
a_4	Ack 수신
a_5	열차 이동을 관찰
a_6	열차 번호 전송
a_7	Polling Timer 작동
a_8	Polling 메시지 전송
a_9	Ack 또는 update 메시지 수신

그 정의는 다음과 같다[5,6].

<정의 2> 입출력 유한상태기계
(I/O FSM : Input/Output Finite State Machine)
I/O FSM은 다음의 다섯 가지 요소로 구성된다.
$M = <S, s_o, I, O, t_r>$ 여기서

- S : 한정된 스테이트들의 집합(a set of finite states)
- s_o : 초기 스테이트(initial state)
- I : 한정된 입력 알파벳(finite input alphabet)
- O: 한정된 출력 알파벳(finite output alphabet)
- t_r : 천이함수(transition function),
  $t_r \subseteq \{s\text{-}i/o \rightarrow s' | s, s' \in S \wedge i \in I \wedge o \in O\}$

다음의 4개 요소(p, a, b, q)는 I/O FSM M의 천이(transition)로 불려지는데 아래와 같이 정의된다.

$(p, a, b, q) \in S \times I \times O \times t_r$

이상의 정의에서 각 천이는 하나의 입력과 하나의 출력에 의해 일어난다.

2) I/O FSM 모델링 및 동작분석

Type 1의 프로토콜 흐름 특성을 I/O FSM으로 모델링한 것은 그림 4와 같고, 표 3은 그림 4의 상태에 대한 설명이다.

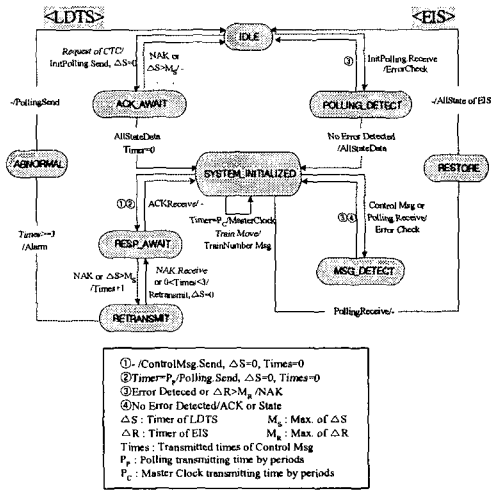


그림 4. Type 1의 I/O FSM 모델링

3. 모형검사방법에 의한 프로토콜 검정

본 장에서는 대수적 명세기법 중 프로토콜의 행위 특성을 가장 강력하게 표현하는 modal mu-calculus

표 3. I/O FSM 모델의 상태 설명

상 태	상 태 설 명
IDLE	LDTS와 EIS간 통신을 시작하기 전 상태
ACK_AWAIT	LDTS에서 polling을 전송하고 응답을 기다리는 상태
POLLING_DETECT	Polling을 수신한 EIS가 메시지의 에러를 체크하는 상태
SYSTEM_INITIALIZED	LDTS가 응답을 받음으로 송수신간의 시스템이 초기화된 상태
RESP_AWAIT	LDTS가 제어 메시지 전송 후, 응답을 기다리는 상태
RETRANSMIT	LDTS가 NAK 수신, 또는 타이머가 expire된 후 메시지 재전송
MSG_DETECT	제어 메시지를 수신한 EIS가 메시지의 에러를 체크하는 상태
ABNORMAL	재전송 3회 이후에도 NAK 또는 어떤 응답도 받지 못할 경우 발생하는 통신 이상 상태
RESTORE	통신 이상 상태에서 정상상태로 복구되는 상태

를 소개하고 2장에서 제시한 LTS로 명세화된 철도 신호제어 프로토콜(그림 3)의 안전성(deadlock 이나 livelock이 없음)과 필연성(정당한 프로토콜의 특성 즉, 상태 또는 행위가 결국 만족되어짐) 특성을 모형검사방법을 사용해서 검정하는 완전성 검정 과정을 일예로 기술한다. 그리고 실제적으로 구현된 프로토콜 검정기의 동작을 보인다.

3.1 Modal mu-calculus

시스템 특성 명세 언어인 modal mu-calculus는 시스템 상태의 부분 집합이 공통적으로 만족하는 논리식인 고정점을 가진 modal 논리이다. 연산자는 원자명제(atomic proposition), ∧(논리곱: conjunction), ∨(논리합: disjunction), [ ](필연성: necessity), < >(가능성: possibility), ν(최대고정점: greatest fixed point), μ(최소고정점: least fixed point)로 구성되어 있으며, 일반화된 modal mu-calculus의 논리식은 다음과 같다.

$$\Phi ::= tt \mid ff \mid Z \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [k]\Phi \mid \langle k \rangle \Phi \mid \nu Z.\Phi \mid \mu Z.\Phi \quad (1)$$

(1)에서 tt는 모든 상태에 대해 참인 것을 나타내고 ff는 거짓임을 나타내며 Z는 명제적 변수, k는 천이 집합의 원소, Φ는 프로세스 특성을 나타낸 방정식이다. νZ는 시스템의 상태 집합들이 일반적으로 만족하는 특성인 최대고정점연산자이며, μZ는 시스템의 상태 집합들이 공통적으로 만족하는 최소고정점연산자이다[7].

1) 안전성(Safety)

안전성 특성은 프로토콜의 부당한 상태 즉, deadlock이나 livelock과 같은 잘못된 상태를 배제하는 특성이다.

Deadlock은 한 상태에서 다른 어떤 상태로의 천이가 존재하지 않기 때문에 다음 행위를 할 수 없는 경우를 말하며, livelock은 특정 상태들의 부분집합 내에서 그 상태들만을 무한히 반복적으로 천이하는 경우로서 그 부분집합 이외의 다른 상태로의 천이가 존재하지 않는 경우를 말한다.

만약 정당한 상태를 나타내는 식이 Φ라면 상태에 대한 안전성은 νZ.Φ∧[-]Z로 표현되고 여기서 [-]는 모든 행위를 나타낸다. 이 식의 명제 값이 참이 된다는 것은 해당 LTS가 상태에 대한 안전성 특성을 가지고 있음을 의미한다. 또한 행위에 대한 안전성 표현은 νZ.[k]ff∧[-]Z이며, 식의 명제 값이 참이 된다는 것은 행위 k가 절대 발생하지 않음과 LTS가 행위에 대한 안전성 특성을 가짐을 의미한다.

2) 필연성(Liveness)

Deadlock 및 livelock이 존재하지 않는다는 조건하에서 프로토콜이 초기 상태로부터 정의된 천이의 순서에 의해 결국에는 도달되어져야 하는 상태와 반드시 발생해야 하는 행위 즉, reachability와 liveness를 만족하는 특성이다.

LTS 모델로 표현된 프로세스의 상태에 대한 필연성은 μZ.Φ∨(<->tt∧[-]Z)로 표현되고, 이 식이 참이 된다는 것은 μZ.Φ와 (<->tt∧[-]Z)이 동시에 거짓이 될 수 없고 식에 해당되는 상태는 항상 도달되어져야 함을 나타낸다. 또한 LTS 모델로 표현된 프로세스에서 행위에 대한 필연성은 μZ.Φ<->tt∧[k]Z로 표현하는데, 이 식이 참이 되기 위해 [k]Z는 반드시 참이 된다. 즉, 결국 행위 k는 발생할 수밖에 없음을 나타낸다. 필연성에 대한 식이 참이 된다는 것은 검정 대상 LTS가 필연성 특성을 가지고 있다는 것을 의미한다.

시스템의 특성을 표현하는 대수적 명세 언어에는 modal mu-calculus외에도 Process calculi, Hennessy-Milner logic과 같은 언어가 있다. Process calculi는 프로토콜의 필연성 특성을 표현할 수 없다는 단점이 있고, Hennessy- Milner logic은 단지 한 번 발생하는 프로토콜의 안전성 및 필연성을 표현할 수는 있으나 지속적인 특성을 표현할 수 없다는 단점이 존재한다[8,9]. 따라서 본 논문에서는 지속적이고 반복적인 프로세스의 안전성과 필연성을 표현 가능한 modal mu-calculus를 사용한다.

3.2 모형검사 알고리즘과 검정적용 예

Modal mu-calculus 논리식으로부터 deadlock, livelock, 그리고 reachability 특성을 만족하는지를 검사하기 위해 R. cleaveland와 B. steffen이 개발한 Solve 알고리즘을 적용하면 자동적이고 효율적인 모형검사가 가능하다[10]. Solve 알고리즘은 modal mu-calculus의 재귀적 특성을 이용한 것으로, 결국에는 원자 명제인 상태 A에 도달되어야 하며 deadlock과 livelock이 없음을 나타내는 modal mu-calculus의 논리식은 (2)와 같다. 이 식은 LTS의 행

위 집합 S에 포함된 어떠한 행위가 발생하더라도 그 LTS를 만족함을 나타낸다.

$$\nu Z.(\mu Y.A \vee (<->tt \wedge [-]Y)) \wedge [-]Z,\ A=\{S_0\} \quad (2)$$

먼저 최소고정점으로 표현된 부 논리식 μY.Φ에 대한 변수 X_1을 생성한다. 변수 X_1을 X_2와 X_3로 나타내어 X_1의 우변에 두면, min{X_1=$X_2 \vee X_3$}인 초기 min block을 얻고, 이러한 과정을 반복 적용하여 min block에 대한 전체 등식을 얻게 된다. 마찬가지로 최대고정점으로 표현된 부 논리식 νZ.Φ를 X_7로 두고 max{X_7=$X_1 \wedge X_8$}인 등식을 생성하여 max block에 첨가한다. 그림 5는 위의 과정을 통해 얻어진 (2)의 min block과 max block을 나타내고, 거기에서 파생된 변수 X_i들의 천이 관계를 나타낸 edge-labeled directed graph G는 그림 6과 같다.

Solve 알고리즘의 초기화 알고리즘을 따라, 철도 신호제어 프로토콜 type 1 LTS의 각 상태와 변수 X_i의 관계를 나타내는 bit-vector, counter, 배열 M을 초기화한다. 여기에서 bit-vector는 min, max block에서 생성된 변수와 입력 LTS의 상태들의 관계를 0와 1로써 표현한 것이다. Counter는 min, max block의 연산자에 대한 LTS의 천이수를 나타낸 것이고 배열 M은 bit-vector의 값이 변경될 때마다 변화된 <S, X_i>를 첨가하여 나타낸다. 초기화 규칙은 그림 7과 같고 알고리즘에서 사용되는 기호는 다음과 같다.

- R_v : Max, min block에서 변수 X_i의 우측 연산자
- R_c : Bit-vector 내에서 변수 X_i의 우측 요소의 값

$B_1 \equiv \min\{X_1 = X_2 \ X_3$ $X_2 = A$ $X_3 = X_4 \ X_5$ $X_4 = [-]X_1$ $X_5 = <->X_6$ $X_6 = tt\}$	$B_2 \equiv \max\{X_7 = X_1 \ X_8$ $X_8 = [-]X_7\}$

그림 5. Max block, min block

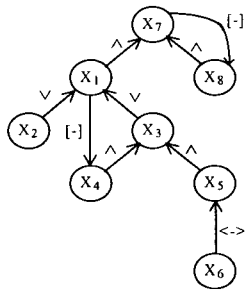


그림 6. Edge-labeled directed graph G

```
// Bit-vector와 배열 M에 대한 초기화 알고리즘 프로시져
procedure bit-vector(L, B)                          //L : LTS, B : max-min block
    for Xi ∈ min block (X1, X2, X3, X4, X5, X6) do
        if (Rv 가 α이고 S가 α를 만족) || (Rv 가 [a]Xi이고 S가 a천이를 갖지 않음) then
            S.X[i] := 1
            <S, Xi>를 배열 M에 추가
        else  S.X[i] := 0
        end if
    end for
    for Xi ∈ max block (X7, X8) do
        if (Rv 가 α이고 S가 α를 만족하지 않음) || (Rv 가 <a>Xi이고 S가 a천이를 갖지 않음) then
            S.X[i] := 0
            <S, Xi>를 배열 M에 추가
        else  S.X[i] := 1
        end if
    end for
end bit-vector

// Counter에 대한 초기화 알고리즘 프로시져
procedure counter(L, B)
    for Xi ∈ min block (X1, X2, X3, X4, X5, X6) do
        if Rv 의 연산자가 '∧' then S.C[i]에 Rc 이 0인 개수를 입력
        else if Rv 가 [a]Xi then S.C[i]에 상태 S에 의한 a천이의 개수 입력
    end for
    for Xi ∈ max block (X7, X8) do
        if Rv 의 연산자가 '∨' then S.C[i]에 Rc 이 1인 개수를 입력
        else if Rv 가 <a>Xi then S.C[i]에 상태 S에 의한 a천이의 개수 입력
    end for
end counter
```

그림 7. 초기화 알고리즘

- S.X[i] : Bit-vector 내에서 관련 상태와 변수가 교차하는 지점의 요소의 값
- S.C[i] : Counter 내에서 관련 상태와 변수가 교차하는 지점의 요소의 값
- α : 원자 명제
- $S' \xrightarrow{[a]} S$: S′는 행위 [a]에 의해 S로 천이되는 상태
- $S' \xrightarrow{\langle a \rangle} S$: S′는 행위 <a>에 의해 S로 천이되는 상태

X_1 - X_6으로 이루어진 min block은 0로, X_7과 X_8로 이루어진 max block은 1로 bit-vector의 값이 셋팅되어지고, 초기화 규칙에 의해 값이 변한 요소들의 상태와 변수쌍은 배열 M에 더해진다. 이렇게 초기화 알고리즘을 적용한 bit-vector, counter, 배열 M의 결과는 그림 8과 같이 얻어진다.

초기화가 되어졌다면 그림 10의 갱신 알고리즘을 적용하여 배열 M의 <S, X_i>가 공집합이 될 때까지 bit-vector, counter, 배열 M을 반복적으로 갱신한다. 갱신된 bit-vector와 counter의 내용은 LTS의 상태 및 변수 X_i와 관련된 정보를 나타내는데, bit-vector의 변수 X_5의 결과에 의해서는 deadlock을, counter의 결과에 의해서는 livelock이 발생한 LTS의 상태를 판단할 수 있다.

Modal mu-calculus 식으로부터 철도 신호제어 프로토콜 type 1 LTS의 완전성을 검정한 결과는 그림 9와 같다.

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
S_0	0	1	0	0	0	1	1	1
S_1	0	0	0	0	0	1	1	1
S_2	0	0	0	0	0	1	1	1
S_3	0	0	0	0	0	1	1	1
S_3	0	0	0	0	0	1	1	1
S_5	0	0	0	0	0	1	1	1

C	X_3	X_4
S_0	2	3
S_1	2	1
S_2	2	1
S_3	2	1
S_4	2	1
S_5	2	1

M[1]=<<S_0,X_2>, <S_0,X_6>, <S_1,X_6>, <S_2,X_6>, <S_3,X_6>, <S_4,X_6>, <S_5,X_6>>
M[2]=< >

그림 8. Bit-vector, counter, 배열 M의 초기화 결과

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
S_0	1	1	1	1	1	1	1	1
S_1	1	0	1	1	1	1	1	1
S_2	1	0	1	1	1	1	1	1
S_3	1	0	1	1	1	1	1	1
S_4	1	0	1	1	1	1	1	1
S_5	1	0	1	1	1	1	1	1

C	X_3	X_4
S_0	0	0
S_1	0	0
S_2	0	0
S_3	0	0
S_4	0	0
S_5	0	0

M[1]=< >
M[2]=< >

그림 9. Bit-vector, counter, 배열 M의 갱신결과

```
// Max block과 min block에 대한 갱신 알고리즘의 프로시저
// L : LTS, B : max-min block, X : 초기화된 bit-vector, C : 초기화된 counter, M : 배열 M, G: edge-labeled directed graph

procedure update(L, B, X, C, M, G)
  while M ≠ NULL do
    M에서 <S, X>를 추출
    for Xi ∈ min block (X1, X2, X3, X4, X5, X6) do
                  // G에서 Xi --α--> Xj인 관계
        if Xj ∈ max block then
            jmaxblock(X, C, M, G)
            //Xj는 max block에 속함
        else
            jminblock(X, C, M, G)
            //Xj는 min block에 속함
    end for

    for Xi ∈ max block (X7, X8) do
        if Xj ∈ max block then
            jmaxblock(X, C, M, G)
            //Xj는 max block에 속함
        else
            jminblock(X, C, M, G)
            //Xj는 min block에 속함
    end for
  end while
end update

procedure jmaxblock(X, C, M, G)
    if 연산자가 '∧' then
        if S.X[j]의 값이 1 then
            S.X[j]를 0로 설정
            M에 <S, Xj>를 추가
        end if
    else if 연산자가 '∨' then
        S.C[j]의 값을 1 감소
        if S.C[j]의 값이 0 then
            S.X[j]를 0로 설정
            M에 <S, Xj>를 추가
        end if
    else if 연산자가 '[a]' then
        for S' --[a]--> S do
            if S'.X[j]의 값이 1 then
                S'.X[j]를 0로 설정
                M에 <S', Xj>를 추가
            end if
        end for
    else if 연산자가 '<a>' then
        for S' --<a>--> S do
            S'.C[j]의 값을 1 감소
            if S'.C[j]의 값이 0 then
                S'.X[j]를 0로 설정
                M에 <S', Xj>를 추가
            end if
        end for
    end if
end jmaxblock

procedure jminblock(X, C, M, G)
  if 연산자가 '∧' then
      S.C[j]의 값을 1 감소
      if S.C[j]의 값이 0 then
          S.X[j]를 1로 설정
          M에 <S, Xj>를 추가
      end if
  else if 연산자가 '∨' then
      if S.X[j]의 값이 0 then
          S.X[j]를 1로 설정
          M에 <S, Xj>를 추가
      end if
  else if 연산자가 '[a]' then
      for S' --[a]--> S do
          S'.C[j]의 값을 1 감소
          if S'.C[j]의 값이 0 then
              S'.X[j]를 1로 설정
              M에 <S', Xj>를 추가
          end if
      end for
  else if 연산자가 '<a>' then
      for S' --<a>--> S do
          if S'.X[j]의 값이 0 then
              S'.X[j]를 1로 설정
              M에 <S', Xj>를 추가
          end if
      end for
  end if
end jminblock
```

그림 10. 갱신 알고리즘

위의 결과에서 bit-vector의 X_5의 모든 요소가 1로, counter의 X_3와 X_4의 모든 요소가 0로 갱신되었음을 알 수 있고, 이는 모든 요소가 max block과 min block을 만족함을 의미한다. 따라서 그림 3의 철도 신호 제어 프로토콜 type 1의 LTS 모델은 deadlock과 livelock이 발생하지 않고 논리식 $\nu Z.(\mu Y.A \vee (<->tt \wedge [-]Y)) \wedge [-]Z$, $A=\{S_0\}$를 만족하는 완전한 프로토콜 모델임을 판단한다.

3.3 알고리즘 구현

1) 검정기의 동작

기술된 검정 방법을 기반으로 개발된 프로토콜 검정기는 deadlock, livelock, reachability, liveness 그리고 determinist와 같은 프로토콜의 특성을 보다 구체적으로 검정해 준다. 구현된 프로토콜 검정기의 구성은 그림 11과 같다.

본 프로토콜 검정기는 LTS 모델의 각 시퀀스를 현재 상태(S_i), 행위(Action), 다음 상태(S_o)로 구성한 LTS 파일을 입력으로 한다. 선택항목 중 사용자가 'Deadlock & Livelock', 'Reachability State', 'Reachability Action' 중에서 한 항목을 선택하면 'InitDeadlock & Livelock', 'InitReachability State', 'InitReachability Action' 부모듈을 호출하여, 정의된 modal mu-calculus 논리식에 따라 각각 max block과 min block을 생성하고, max block과 min block에서 생성된 변수와 입력 LTS의 state를 이용해 bit-vector와 counter를 만든다(Initialize). Initialize 모듈은 bit-vector를 초기화하는 X.initialize, counter를 초기화하는 C.initialize, 그리고 edged-labeled directed graph를 생성하는 B.graph 부모듈을 호출하게 되고, 다음으로 X.initialize와 C.initialize, B.graph에 배열 M이 공집합(empty)이 될 때까지 Solve의 갱신 알고리즘을 적용시켜 Deadlock & Livelock, Reachability State, 그리고 Reachability Action의 검정 결과를 판단한다.

그러나 만약 사용자가 'Liveness' 또는 'Determinist'를 선택할 경우, 모형검사 알고리즘과는 상관없이 'Liveness check'는 초기상태에서 도달가능한 모든 상태와 행위를 출력하고, 'Determinist check'는 어떤 특정한 상태에서 동일한 행위에 의해 다음 상태가 두 가지 이상 존재하는지를 검사한다.

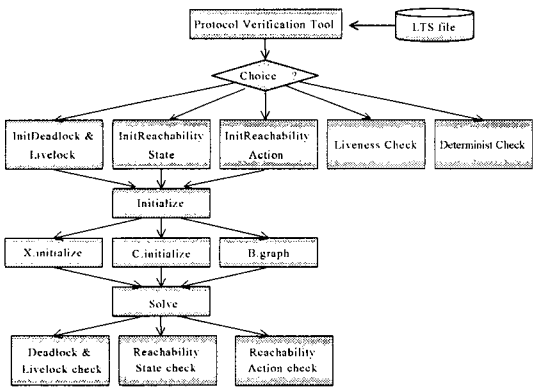


그림 11. 프로토콜 검정기의 동작절차

2) 검정기 결과 분석

본 소절에서는 각 검정항목의 검정 결과를 판단하는 절차에 대해 간략히 기술한다.

(가) Deadlock & Livelock Check

Solve 알고리즘을 통해서 배열 M이 공집합이 될 때까지 BitArray X와 CounterArray C를 갱신한 결과에서 검정 결과를 판단하는데, deadlock 판정 procedure는 그림 12와 같고, BitArray의 항상 참이 되어야하는 요소 X[i][X_5]가 0인 경우의 상태를 검출하면 deadlock 발생여부를 판단할 수 있다. 또한 Livelock Check는 검출 논리식에서 안정성(safety)을 표현한 부 논리식과 관련된 변수 X_3와 X_4가 CounterArray에서 1로 set 되어 있는 경우 검출된다.

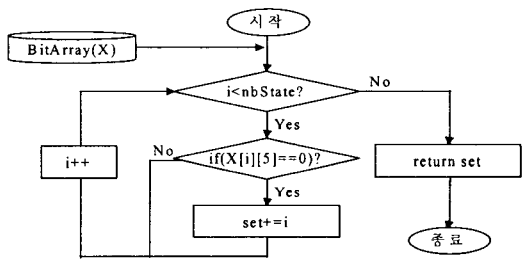


그림 12. Deadlock check 부모듈의 procedure

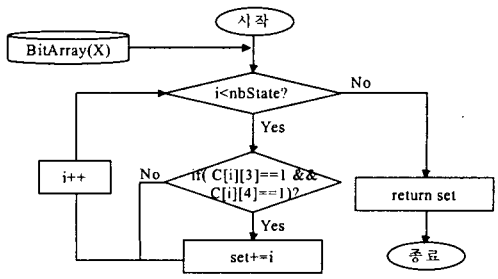


그림 13. Livelock check 부모듈의 procedure

(나) Reachability State Check

Reachability State는 먼저 검사 상태인 s와 관련된 BitArray의 요소 X[i][X2]가 1인 경우와 이 상태에서 act_1과 act_2와 각각 관련된 BitArray의 요소 X[i][X4]와 X[i][X5]가 모두 1인지를 검사하고, 또한 CounterArray C의 요소 C[i][3]의 요소가 1이 되는 값을 검출함으로써 행위 발생여부를 검사한다. 이에 대한 프로시져는 그림 14에 나타나 있다.

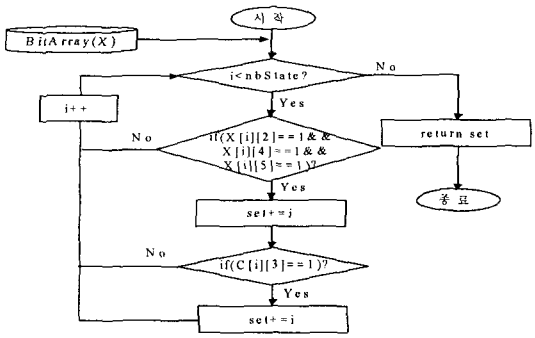


그림 14. Reachability state check 부모듈의 procedure

(다) Reachability Action Check

Reachability Action은 우선 $act_2 \rightarrow act_3$가 발생하는 상태와 관련된 BitArray의 요소 X[i][X3]가 1인 상태 's_2'를 검출하여 set2에 저장하고, 이러한 조건을 만족하는 상태가 존재하면 $act_1 \rightarrow act_2$가 발생하는 상태와 관련된 CounterArray의 요소 C[i][1]가 1이고 act_2를 거쳐 's_2'를 발생하는 상태인 's_1'을 검출하여 set1에 저장한다. Action reachability를 만족하는 이 상태를 return하였을 때, main 함수에서는 set1과 set2안에 값이 있는지 없는지를 판단하여, 둘 중 하나라도 값이 없거나 둘 다 없다면 입력된 3개의 행위에

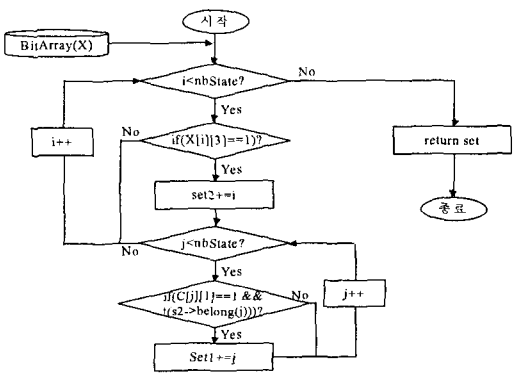


그림 15. Reachability action check 부모듈의 procedure

대해 reachability action을 만족한 것으로 판단한다.

(라) Liveness Check

Liveness Check는 초기 상태와 거기에서 파생되는 행위를 출력한다. 그리고 만약 다음 상태가 초기 상태와 같지 않을 경우, 시작 상태를 다음 상태로 설정하여 이 과정을 계속 반복하다가 다음상태가 초기 상태와 같아질 때, 부모듈을 종료하며 이에 대한 절차는 그림 16과 같다.

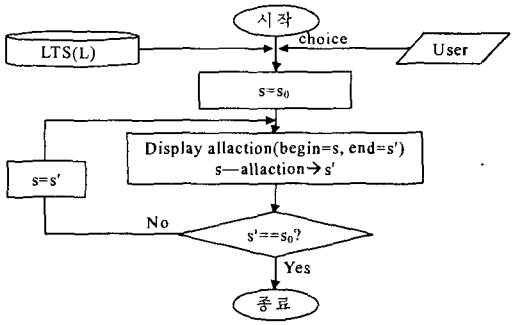


그림 16. Liveness check 부모듈의 procedure

(마) Determinist Check

Determinist Check는 각 상태에서 동일한 행위에 대해 서로 다른 다음 상태가 있는 경우에는 0을 리턴하고, 그렇지 않은 경우에는 1을 리턴한다. 이에 대한 절차는 그림 17과 같다.

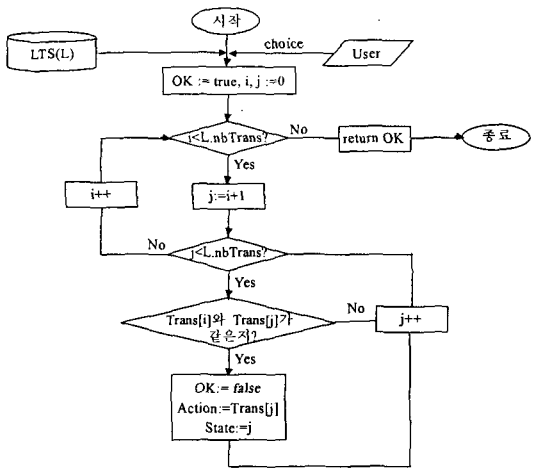
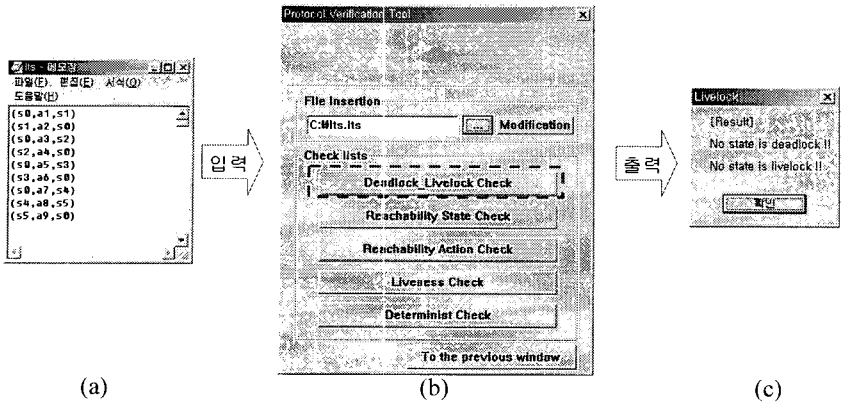


그림 17. Determinist check 부모듈의 procedure

3.4 구현 도구의 동작 확인

기술된 알고리즘을 사용하여 구현된 프로토콜 검



(a) (b) (c)

그림 18. 프로토콜 검정기 Deadlock 체크 결과

정기의 실행화면은 그림 18의 (b)와 같다.

위 그림은 그림 3의 LTS 모델을 입력하여 그 결과를 출력한 예로서, (a)와 같이 확장자가 lts인 완전한 LTS 모델을 입력으로 하고, 'Deadlock_Livelock Check'를 실행하면 내부의 모형검사 알고리즘을 수행하고, 결과로써 deadlock과 livelock이 발생하지 않았음을 출력한다. 그림 19는 상태 'S_4'에서 deadlock이 발생한 모델을 입력한 예로, 잘못된 입력으로 인한 deadlock의 발생과 발생한 장소를 출력하고 'Modification' 버튼을 사용해서 모델을 수정할 수 있도록 하였다.

4. 적합성시험

본 장에서는 적합성시험의 정의를 기술하고 그림 4의 I/O FSM 프로토콜 명세로부터 구현물에 대한 적합성시험 계열을 생성하는 방법을 제시한다. 그리고 구현된 적합성시험 생성기의 동작 절차를 보인다.

4.1 적합성시험의 정의

적합성시험의 목적은 어떤 프로토콜의 구현 I(Implementation)가 원래 명세(혹은 표준) S(Specification)에 합당하게 구현되었는지를 시험하는 것으로 통신 프로토콜 제품 구현 과정에서 중요한 역할을 한다. 일반적으로 적합성시험의 정의는 주어진 명세 S를 기초로 하여 생성된 시험 계열(test cases)로서 구현 I가 명세 S에 대해 프로토콜 행위(behaviour)와 능력(capacity)이 일치하는지를 시험하는 것이다 [11-14].

시험 계열 생성을 위한 일반적인 방법들은 명세 S가 입출력 유한상태기계(I/O FSM) 형태로 표현된 프로토콜로부터 출발하는데, 프로토콜의 명세를 나타내는 I/O FSM 모델이 그림 20과 같이 주어질 때 적합성시험은 아래의 3단계로 이루어진다.

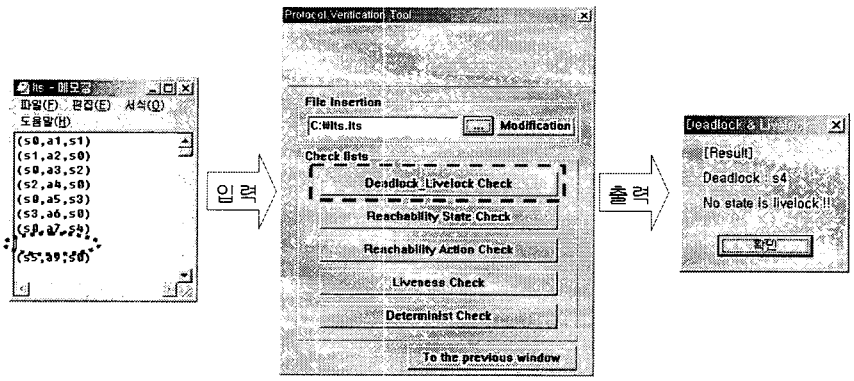


그림 19. Deadlock 발생 모델에 대한 검정기 출력

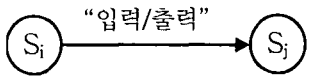


그림 20. 명세 I/O FSM

① 명세 I/O FSM의 상태 S_i에 해당하는 상태로 구현 I/O FSM을 위치시킨다.
② 명세 I/O FSM에서 얻어진 시험 계열 "입력"을 구현 I/O FSM에 적용시킨 후 생성되는 "출력"을 판단한다.
③ 구현 I/O FSM에서 생성된 출력이 명세 I/O FSM에서 기술된 "출력"과 같은지를 확인하고 도착한 상태가 명세 I/O FSM의 그것과 같은지를 검정한다.

이러한 I/O FSM모델로 표현된 프로토콜들로부터의 시험계열 생성을 위해 일반적으로 UIO(Unique Input/Output) 시퀀스, DS(Distinguish Sequence) 시퀀스, CS (Characterizing Set) 시퀀스 등을 사용한다[15]. 여기에서 DS 시퀀스나 CS 시퀀스 등은 존재를 위한 기본 조건이 모델 I/O FSM 자체가 완전 명세(complete specification) 되어야 하나, UIO 시퀀스는 이러한 요구 조건이 없으며 또, 일반적으로 통신 프로토콜을 모델링하는 I/O FSM은 부분적으로 명세 된다. 또한 DS 시퀀스나 CS 시퀀스는 명세내에서 찾기가 어렵고 존재 여부도 일반적으로 판단하기가 매우 어렵다. UIO 시퀀스는 I/O FSM의 모든 상태에 대해 존재하며 길이도 DS 시퀀스나 CS 시퀀스에 비해 짧기 때문에 결과적으로 짧은 시험 계열 생성을 보장한다. 따라서 본 논문에서는 UIO 시퀀스를 사용하여 시험계열을 생성하는 방법을 사용한다.

4.2 UIO방법에 의한 시험 계열 생성 및 생성적용 예

UIO(Unique Input Output) 시퀀스는 시험하는 천이 후에 도착한 상태의 유일한 입력/출력 시퀀스를 시험 계열에 포함시켜 적용한 후, 구현 I/O FSM의 결과 상태를 확인하는 방법이다. I/O FSM으로 표현된 프로토콜 명세로부터 적합성시험 계열 생성은 명세에 나타나 있는 각 천이에 도착 상태의 UIO 시퀀스를 concatenation하여 생성하는데 다음과 같은 식으로 나타낼 수 있다.

$$R_i \cdot \text{shortest-path}(S_0\text{-}S_i) \cdot T_{ij}@UIO(S_j) \qquad (3)$$

위 식에서 R_i는 시험대상을 초기화 상태로 보내는 심볼이며, shortest-path(S_0-S_i)는 초기상태 S_0에서 해당 시험천이에 대한 시작 상태까지의 shortest path, T_{ij}는 시험되어야 할, 명세 I/O FSM의 상태 S_i에서 S_j로 보내는 천이를 나타내고, UIO(S_j)는 도착 상태의 UIO 시퀀스를 나타낸다. 또 @는 concatenation 심볼이다. 여기서 UIO 시퀀스는 시험되는 천이에 의해 도착된 상태가 명세에서 원하는 올바른 S_j인가를 시험하는데 사용된다.

위에서 제시된 UIO 시퀀스 생성 알고리즘은 다음과 같다.

간결성을 위해 철도 프로토콜 type 1의 I/O FSM 모델 그림 4의 각 천이를 표 4와 같은 약어로 대체하고 그림 4에 위의 알고리즘을 적용하여 각 상태별 UIO 시퀀스를 구한 것은 표 5와 같다.

위의 표 5를 사용하여 (3)을 적용하면 표 6과 같은 철도 프로토콜 type 1에 대한 적합성시험 계열이 생성된다. ()내에 표현된 상태는 하나 이상의 천이에 대한 구별로서 출발상태와 도착상태를 나타낸다.

결과적으로 위에 나타낸 적합성시험 계열의 입력 부분을 type 1에 대한 구현이 받아들이고 출력 부분을 생성하여 명세의 출력과 같은지를 판단하면 type 1의 구현이 type 1의 명세에 대해 정확하게 구현되었다는 적합성시험 결과를 확인할 수 있다.

입력 : 프로토콜 명세 I/O FSM
출력 : 모든 상태의 UIO시퀀스

(1) 길이 l 의 모든 천이가 어느 상태 s_i에만 홀로 존재하는지 검사 (초기에는 l =1).
(2) 만약 스텝 1에서 길이 l 의 UIO와 s_i를 위해 존재하지 않으면 모든 천이에 대해 검사한 후, 길이 l + 1로써 반복한다.
(3) 주어진 I/O FSM의 다른 모든 상태에 대해서 (1)과 (2)를 반복한다.

그림 21. UIO 시퀀스 생성 알고리즘

표 4. 천이에 대한 약어

천이	약어	천이	약어
Request of CTC/InitPolling.send, △S=0	A	Control Msg or Polling.Receive/Error Check	K
NAK or △S>M_S/ -	B	Timer=P_C/Master Clock	L
Error deteced or △R>M_R/NAK	C	Train Move/Train NumberMsg	M
InitPolling.Receive / ErrorCheck	D	NAK or △S>M_S/Times+1	N
AllStateData / Timer=0	E	NAK.receive or 0<Times<3/Retransmit, △S=0	O
No error detected/AllStateData	F	Times>=3/ALARM	P
-/ControlMsgSend, △S=0, Times=0	G	PollingReceive/-	Q
Timer=P_P/Polling.Send, △S=0, Times=0	H	- /PollingSend	R
ACK Receive/ -	I	- /AllState of EIS	S
No error detected/Ack or State	J		

표 5. Type 1 상태별 UIO 시퀀스

상태	상태별 UIO 시퀀스
IDLE(S_0)	Request of CTC / Initpolling.Send, △S=0 (A)
ACK_AWAIT(S_1)	NAK or △S>M_S/ - (B)
POLLING_DETECT(S_2)	No error detected / AllStateData (F)
SYSTEM_INITIALIZED(S_3)	-/ControlMsgSend, △S=0, Times=0 (G)
RESP_AWAIT(S_4)	ACK Receive/ - (I)
RETRANSMIT(S_5)	NAKReceive or 0<Times<3/Retransmit, △S=0 (O)
MSG_DETECT(S_6)	No error detected/Ack or State (J)
ABNORMAL(S_7)	- /PollingSend (R)
RESTORE(S_8)	- /AllState of EIS (S)

표 6. Type 1에 대한 적합성시험 계열 생성

천이		적합성시험 계열	천이	적합성시험 계열
A		R_i A B	J	R_i A E K J G
B		R_i A B A	K	R_i A E K J
C	(S_2S_0)	R_i D C A	L	R_i A E L G
	(S_6S_3)	R_i A E K C G	M	R_i A E M G
D		R_i D F	N	R_i A E H N O
E		R_i A E G	O	R_i A E H N O I
F		R_i D F G	P	R_i A E H N P R
G		R_i A E G I	Q	R_i A E Q S
H		R_i A E H I	R	R_i A E H N P R A
I		R_i A E H I G	S	R_i A E Q S A

4.3 알고리즘 구현

I/O FSM을 기반으로 적합성시험 계열을 생성하는 도구의 전체적인 절차는 그림 22와 같다.

먼저 I/O FSM 모델을 입력으로 하여 I/O FSM 각 상태별로 그림 23의 절차를 거쳐 UIO 시퀀스를 생성한 다음, 초기 상태에서 각각의 시험 상태에 이

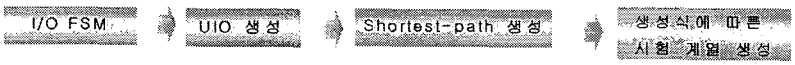


그림 22. 적합성시험 계열 생성 절차

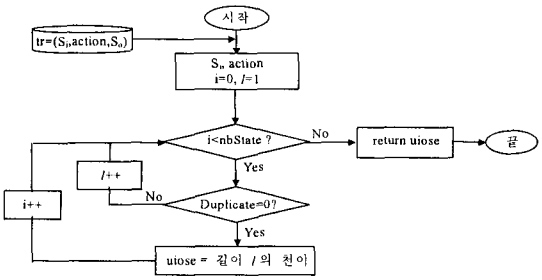


그림 23. UIO 시퀀스 생성 부모듈의 procedure

르는 가장 짧은 경로를 구하고 적합성시험 계열 생성식에 따라 각 천이에 대한 시험 계열을 생성한다. 적합성시험 계열을 생성하는 부모듈은 그림 24의 절차와 같이 실행된다. 부모듈은 초기화 입력 R_i를 적용하여 모든 시험대상이 초기화 상태에서 시작하도록 한다. 그리고 최단경로와 시험대상 천이인 T_{ij}, 도착상태의 UIO 시퀀스를 concatenation함으로써 모든 천이에 대한 적합성시험 계열을 생성한다.

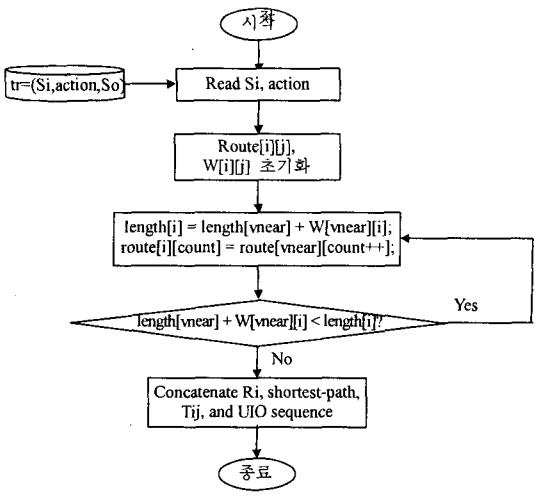


그림 24. 적합성시험 계열 생성 부모듈의 procedure

4.4 구현 도구의 동작 확인

기술된 알고리즘을 사용하여 구현된 적합성 시험 계열 생성기의 실행화면은 그림 25와 같다.

확장자가 fsm인 파일을 검색하여 입력으로 하고 'Conformance Test Start' 버튼을 누르면, 각 상태에 대한 UIO 시퀀스와 각 천이에 대한 적합성 시험 계열이 출력된다.

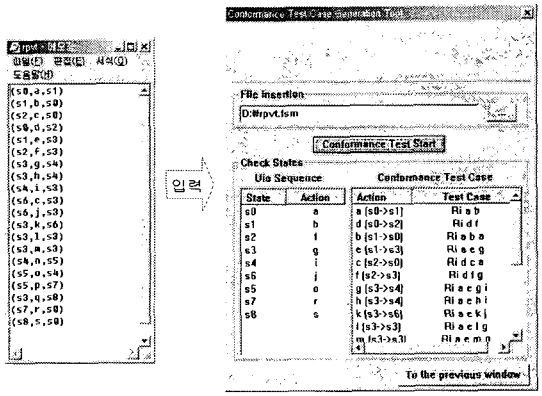


그림 25. 적합성 시험 계열 생성기 실행화면

5. 결론 및 향후 연구 추진사항

본 논문에서는 LTS로 명세화된 통신 프로토콜을 정형기법으로 명세화하고 안전성 및 필연성 특성을 대수적 명세 기법인 modal mu-calculus를 사용하여 검정하는 프로토콜 검정기와 검정되어진 프로토콜 명세로부터 UIO 방법에 의해 적합성시험 계열을 생성하는 도구를 구현하였다. 구현시 사용자가 쉽게 다룰 수 있도록 윈도우 NT 환경 하의 GUI (Graphic User Interface)기능을 위해 visual C++을 사용하였다.

구현된 도구를 이용하여 철도 신호제어 프로토콜 type1을 검정 및 시험하는 방법을 보였으며, 구현된 도구가 올바로 동작함도 확인하였다.

본 논문에서 제시된 검정 및 시험 계열 생성 방법은 자연어에 근거한 수동적 검정과 시험 계열 생성에 비해 보다 신뢰성 있는 프로토콜을 개발하고, 제품의 개발생산력 및 품질의 안정성을 유지하기 위해 사용되어지며 프로토콜 개발에 소요되는 비용과 시간을 최소화시킬 수 있다. 또한 개발 및 표준의 부정확한 부분과 미진한 부분을 발견 보완함으로써, 개발 제품의 상업화 및 폭넓은 현장적용으로 국내 표준화 기술 경쟁력의 확보가 기대된다.

향후 연구 사항으로는 프로토콜 검정기의 입력을 LTS뿐만 아니라 I/O FSM도 가능하도록 기능을 확장하고, 또한 적합성시험 생성기에 최적화 기법에 의한 적합성시험 계열 생성방법을 적용함으로써 개발된 도구의 성능을 더욱 향상시키고 기능을 확장하는데 있다.

참 고 문 헌

[1] D. Schwabe, "Formal Techniques for the Specification and Verification of Protocols," *Report No. CSD-810401, UCLA, (PhD Thesis)*, Apr. 1981.

[2] ISO/IEC 9646-1, "Information Technology - open Systems Interconnection - Conformance testing methodology and framework - Part1 : General concepts," 1994.

[3] R. Cleaveland, "Tableau-Based Model-checking in the Propositional Mu-calculus," *Acta Informatica*, Vol. 27, No. 3, pp. 725-727, 1990.

[4] P. V. Koppol and K. C. Tai, "Conformance Testing of Protocols specified as Labeled Transition systems," *International Workshop on Protocol Test System, IWPTS95*, pp. 143-158, Evry, France, 1995.

[5] R. J. Linn, "Conformance Testing for OSI Protocols," *Computer Networks and ISDN Systems*, Vol. 18, No. 3, pp. 203-219, 1989.

[6] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications," *ACM TOPLAS*, Vol. 8, No. 2, pp. 244-263, Apr. 1986.

[7] Kenneth L. McMillan, *Symbolic model checking*, Kluwer Acacenic Publishers,. Model checking, 1996.

[8] J. Bradfield and C. Stirling, "Local model checking for infinite state spaces," *Theoretical Computer Science 96*, pp. 157-174, 1992.

[9] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[10] R. Cleaveland and B. Steffen, "A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus," *Formal Methods in System Design*, Vol. 2, No. 2, pp. 121-147, 1993.

[11] B. Sarikaya, *Principles of Protocol Engineering and Conformance Testing*, Ellis Horwood, New York, NY, 1993.

[12] K. G. Knightson, *OSI Protocol Conformance Testing : IS 9646 Explained*, McGraw-Hill, Inc., New York, NY, 1993.

[13] S. U. Kim, "INAP protocol comformance Testing," *IEEE-IN97, Corolado Springs, USA*, May, 1997.

[14] 김상기, 김성운, 정재윤, "형식기술기법에 의한 INAP프로토콜 적합성 시험계열 생성," *정보처리학회 논문집*, 제4권 제2호, pp. 552-562, 1997년 1월.

[15] A. R. Cavalli, S. U. Kim, and P. Maigron, "Improving Conformance Testing for LOTOS," *FORTE'93, Boston, USA*, pp. 367-381, Oct. 1993.

서 미 선

2003년 2월 부경대학교 정보통신공학과 졸업(학사)
2005년 2월 부경대학교 정보통신공학과 졸업(석사)

관심분야 : 프로토콜 엔지니어링, 데이터통신, 컴퓨터 네트워크

황 종 규

1994년 2월 건국대학교 전기공학과 졸업(학사)
1996년 2월 건국대학교 전기공학과 졸업(석사)
2005년 2월 한양대학교 전자통신전파공학과 졸업(박사)
1995년~현재 한국철도기술연구원 전기신호연구본부 선임연구원

관심분야 : 철도신호통신 기술, 프로토콜 엔지니어링, 컴퓨터 네트워크

이 재 호

1987년 2월 광운대학교 전자공학과 졸업(학사)
1989년 2월 광운대학교 전자공학과 졸업(석사)
2005년 2월 고려대학교 메카트로닉스공학과 졸업(박사)
1995년~현재 한국철도기술연구원 전기신호연구본부 책임연구원
관심분야 : 철도신호통신기술, 프로토콜 엔지니어링

김 성 운

1982년 12월~1985년 9월 한국전자통신연구원, 연구원
1985년 10월~1995년 8월 한국통신 연구개발본부, 연구실장
1990년 8월 프랑스 국립 파리 7 대학교 정보공학과 졸업 (석사)
1993년 8월 프랑스 국립 파리 7 대학교 정보공학과 졸업 (박사)
2000년 8월~2001년 7월 미국 NIST 초빙 연구원, DARPA 과제 수행
관심분야 : 프로토콜 엔지니어링, DWDM optical network, RWA, QoS, GMPLS