LTS 명세 검증을 위한 모델 검증기 개발

박 용 범† · 김 태 균†† · 김 성 운†††

요 약

본 논문은 LTS 프로세스 명세 검증을 위해 프로토콜의 행위와 상태에 대한 deadlock, livelock, reachability, liveness 검증을 위한 모델 검증기 구현에 대해 기술하였다. Modal mu-calculus를 사용하여 구현된 모델 검증기는 modal logic으로 표현된 프로토콜 특성이 LTS 명세하에서 만족하는지를 자동적으로 검증해 주는 model checking 도구이고 LTS 명세의 Safety와 Liveness 검증에 매우 강력한 성능을 보이는 것을 구현을 통해 실험적으로 증명하였다. 제시된 도구는 Windows NT 환경하에서 IBM PC로 C++ 언어를 사용하여 구현되었다.

A Study on Implementation of Model Checking Program for Verifying LTS Specification

Yong-Bum Park† · Tae-Gyun Kim†† · Sung-Un Kim†††

ABSTRACT

This paper presents an implementation of model checking tool for LTS process specification, which checks deadlock, livelock and reachability for the state and action. The implemented formal checker using modal mu-calculus is able to verify whether properties expressed in modal logic are true on specifications. We prove experimentally that it is powerful to check, safety and liveness for the state and action on LTS. The tool is implemented by C++ language and runs on IBM PC under Windows NT.

1. 서 론

최근 정보통신 관련 소프트웨어 시스템들은 기능면으로 복잡해지고, 규모면에서 광범위해짐에 따라 명세의 무결성 및 완전성을 검사하여 신뢰성 있는 소프트웨어 개발 방법으로 정형 기술 기법의 사용이 필수 불가결하게 되었다.

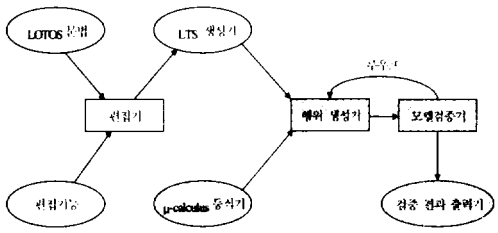
정형 기술 기법 중 가장 핵심적인 기술인 검증 기술은 주어진 프로토콜 사양과 그 하위 계층에서 제공되는 서비스 사양에 근거하여 통신 개체들의 상호 동작이 주어진 서비스 사양을 만족하는가를 시험하는 과정이며, 모든 프로토콜에 필수적인 일반적인 정확성(general correctness)특성을 만족하는가에 대한 프로토콜 사양을 분석하는 과정이다. 프로토콜 명세는 deadlock이나 잘못된 행위가 존재하지 않아야 하는 안전성(safety)과 정의된 행위가 반드시 일어나야 하는 필연성(liveness) 등과 같은 특성들을 지니고 있고, model checking은 프로토콜 명세의 이와 같은 특성을 자동적, 형식적으로 검증하는 기술이다[1][2].

본 논문에서는 명세의 표현력에 있어 조합, 이해, 수

† 정 회 원 : 한국전자통신연구원
†† 준 회 원 : 부경대학교 정보통신공학과
††† 정 회 원 : 부경대학교 정보통신공학과
논문접수 : 1997년 9월 18일, 심사완료 : 1998년 2월 6일

정 및 확장이 용이하며 구현물에 독립적으로 기술되고 프로토콜과 서비스의 문법적 분석이 가능한 LOTOS(Language Of Temporal Ordering Specification) 정형명세기법을 사용한다. LOTOS로 표현된 프로토콜 명세로부터 의미적 분석과 증명이 용이한 형식기술기법 중 레이블 천이 시스템(LTS : Labeled Transition System)을 생성하여, 개발된 LTS 명세 검증기에 입력하고 이 모델에 modal mu-calculus논리식을 적용하여 검증기능을 수행하는 LTS 행위 명세 검증 도구 구현에 대해 기술한다.

LTS 행위 명세 검증도구는 LOTOS에 의한 프로토콜 개발 환경인 LOVTE(LOtos Verification and Testing Environment)내에 구현되어 있고, (그림1)의 검은 부분에 해당한다.



(그림 1) LOVTE 개발환경
(Fig. 1) LOVTE Development Environment

(그림1)의 검은 부분에 해당하는 LTS 행위 명세 검증도구의 기능에 대해, 본 논문의 2장에서는 검증 사항 및 대상에 대해 기술하고, 3장은 프로토콜 행위 명세를 검증하기 위해 사용된 대수적 명세 기법 중 행위 특성을 가장 강력하게 나타낸 modal mu-calculus에 대해 정의한다. 4장은 검증을 위해 사용된 model checking 알고리즘에 대해 기술하고, Borland C^{++}로 구현된 검증기에 의해 실제 LTS를 사용하여 프로토콜 특성을 검증한다. 마지막으로 5장은 간단한 결론과 향후 연구 사항에 대해 기술한다.

2. 검증 사항 및 대상

통신 프로토콜이 적절한 통신을 하기 위해서는 프로토콜 상태의 deadlock, 비정상적인 도달등과 같은 잠재적 설계 에러가 없어야 하며, 사용자 요구사항과 일치하는지 결정하고, 다른 프로세서와의 원활한 통신이 이루어지는지 검사해야 한다.

프로토콜 검증은 프로토콜 명세의 정확성, 안전성과 일관성 및 필연성등을 알아보는 것으로 model checking에서 보다 구체적으로 검증하여야 할 프로토콜의 특성은 아래와 같다[3][4].

- deadlock : 한 상태에서 다른 어떤 상태로의 천이가 존재하지 않기 때문에 다음 행위를 할 수 없는 경우, 즉 그 상태에서 나가는 천이가 존재하지 않는다.
- livelock : 프로토콜(명세) 상태들의 부분집합 내에서 그 상태들만을 무한히 반복적으로 천이하는 경우로써 그 부분집합 이외의 다른 상태로의 천이가 존재하지 않는다.
- reachability : 프로토콜이 작동되기 시작할 때, 즉 프로토콜의 시작에서 정의되어지는 특별한 상태로써 초기상태가 존재하는데, 이 초기상태로부터 프로토콜은 정의된 천이의 순서에 의해 일부 또는, 모든 다른 상태에 도달하게 된다. 프로토콜이 정의된 천이 순서에 의해 정의된 상태로 도달한다면 그 천이와 상태에 대해 이 명세는 올바른 프로토콜이다.
- liveness : 프로토콜의 어떤 정당한 특성(상태 또는 행위)이 결국 만족되어지는 것으로, 결국에는 도달되어야 하는 상태와 반드시 발생해야 하는 행위를 나타낸다.

(그림1)의 구성도에서 LTS 행위 검증을 위한 각 모듈별 개발 내용은 〈표1〉과 같다.

〈표 1〉 모듈별 개발 내용
〈Table 1〉 Development Contents of the Related Modules

연구 개발 모듈	연구 개발 내용
μ-calculus 등식기	· μ-calculus 조직 구현 · deadlock 검사 등식 구현 · livelock 검사 등식 구현 · reachability 검사 등식 구현
행위 생성기	· deadlock, livelock, reachability 검사 알고리즘 구현 · LTS입력과 변수 binding기능 구현
모델 검증기	· deadlock 검증 모듈 구현 · livelock 검증 모듈 구현 · reachability 검증 모듈 구현
검정 결과 출력기	· Windows NT 환경하에서 GUI 기능에 의한 검정 결과 M/M 인터페이스 구현

3. Modal mu-calculus

3.1 LTS

본 절에서는 프로토콜을 검증하는 정형 명세 기법 중 프로토콜 정형 명세를 위한 의미 모델로써 많이 사용되는 LTS를 다음과 같이 정의한다[5].

LTS는 다음 4개의 요소로 구성된 레이블화된 천이 시스템 "LTS = ⟨S, L, T, So⟩" 으로 정의되는데, 여기서

-S : 상태들의 집합(a set of states)

-L : 관찰가능한 행위집합(a set of observable actions)

-$T \subseteq S \times (L \cup \{\tau\}) \times S$: 천이 관계 (transition relation)

-$So \in S$: 초기스테이트(initial state)

이 정의에서 심볼 "τ"는 비결정형 모델(non-determinant model)을 위해 이용되는 시스템내부 혹은 관찰가능치 않은 행위(internal 혹은 inobservable act-ion)를 나타낸다.

3.2 Modal mu-calculus

Modal mu-calculus는 external fixed point를 가진 modal logic이며, 프로토콜의 특성을 가장 다양하게 표현하는 명제적 temporal logic이다. 구문은 atomic proposition, ∧(conjunct), ∨(disjunct), [ ](necessarily), ⟨ ⟩(possibly), μ(minimum fixed point), ν(maximum fixed point)로 구성되어 있으며, 일반화된 modal mu-calculus의 논리식은 다음과 같다[6].

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

여기서 tt는 모든 상태에 대해 참인 것을 나타내고 ff는 거짓임을 나타내며 Z는 명세적 변수, 즉 프로세스를 나타낸다. k는 행위 집합의 원소이고, Φ는 프로세스 특성을 나타낸 방정식이다. νZ는 maximum fixed point operator이고, μZ는 minimum fixed point operator이다.

다음은 프로토콜 행위 분석을 위해 필연성과 안전성을 표현한 modal mu-calculus 논리식이다.

3.2.1. 안전성

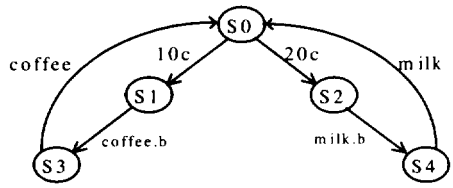
프로토콜의 부당한 상태, 즉 deadlock이나 livelock과 같은 상태에 대해 배제함을 나타내는 특성으로 modal mu-calculus의 식은 다음과 같다.

만약 정당한 상태를 나타내는 식이 Φ라면 $\nu Z.\Phi \wedge [-]Z$로 표현된다. 여기서 [-]는 모든 행위를 나타낸다.

(그림2)는 자판기 ven을 나타낸 LTS 모델인데, 이 LTS의 존재하지 않는 하나의 상태는 커피와 우유가 동시에 발생하는 것이며, 이 상태에 대한 안전성 논리식은

$$\nu Z.([coffee]ff \vee [milk]ff) \wedge [-]Z$$

로 표현되고, 위 식의 명제 값이 참이 된다는 것은 (그림2)의 LTS가 상태에 대한 안전성을 가지고 있음을 의미한다. 위 식이 참이 되기 위해서는 ([coffee]ff ∨ [milk]ff)와 [-]Z가 반드시 참이 되어야 하며, [coffee]ff와 [milk]ff는 어느 한 쪽이 참이 되거나 둘 다 참이 되어야 한다. [-]Z는 LTS에서 정의된 모든 행위에 의해 프로세스 Z로 천이함을 의미한다.



(그림 2) 자판기 ven LTS
(Fig. 2) Ven LTS for Vending Machine

이러한 LTS 모델에서 행위에 대한 안전성 표현은

$$\nu Z.([k]ff \wedge [-]Z$$

이며, 식의 명제 값이 참이 된다는 것은 검증 대상 LTS가 행위에 대한 안전성을 가짐을 의미한다. 즉, 행위k가 절대 발생하지 않음을 나타낸다. 자판기 ven에서 행위 tea는 결코 발생하지 않으며, νZ.([tea]ff ∧[-]Z)로 표현되는데, 식이 참이 되기 위해 [tea]ff와 [-]Z는 반드시 참이 되어야 한다.

3.2.2 필연성

프로토콜의 어떤 정당한 특성(상태 또는 행위)이 결

국 만족되어지는 것으로, 결국에는 도달되어져야 하는 상태와 반드시 발생해야 하는 행위를 나타낸다.

LTS 모델로 표현된 프로세스의 상태에 대한 필연성은

$$\mu Z.\Phi \vee (\langle - \rangle tt \wedge [-]Z)$$

로 표현되고 이는 식 Φ에 해당되는 상태는 항상 도달되어져야 함을 나타낸다. 또한 LTS 모델로 표현된 프로세스에서 행위에 대한 필연성의 논리식은

$$\mu Z.\langle - \rangle tt \wedge [-]Z$$

로 표현하는데, 결국 행위 k는 발생할 수 밖에 없음을 나타낸다. 상태에 대한 필연성의 식이 참이 되기 위해 $\mu Z.\Phi$와 (⟨-⟩tt ∧ [-]Z)가 동시에 거짓이 될 수 없고, 행위에 대한 식이 참이 되기 위해 [k]Z는 반드시 참이 된다. 즉, 필연성에 대한 식이 참이 된다는 것은 검증 대상 LTS가 필연성 특성을 가지고 있다는 것을 의미한다. 여기서 [k]Z는 행위 k에 의해 프로세스 Z로 천이해야 해야 함을 의미하고, ⟨-⟩tt는 어떤 행위에 대해서도 참임을 의미한다.

위에서 기술한 프로토콜의 특성(liveness, safety)을 표현하는 modal mu-calculus의 식으로부터 완전한 자판기의 두 가지 특성에 대한 표현식의 예로 행위 10c가 발생하면 반드시 coffee.b와 coffee가 연속적으로 발생하고, deadlock과 livelock이 없음을 나타내는 modal mu-calculus의 논리식은 다음과 같으며, 식이 참이 되기 위해 [-]Z, $\mu Y.\langle coffee \rangle tt$와 [coffee.b]는 동시에 참이 되어야 한다.

$$\mu Z.[10_c](\mu Y.\langle coffee \rangle tt \wedge [coffee.b]Y) \wedge [-]Z$$

또한, S2상태는 20c에 의해 milk.b를 반드시 발생하는 상태이며, 이에 대한 논리식은 vZ.(μY.A ∨ (⟨milk.b⟩tt ∧ [20c]Y)) ∧ [-]Z 단, A={S2}으로 나타내어진다.

4. Modal mu-calculus를 적용한 LTS모델 검증

4.1. 검증 알고리즘

Modal mu-calculus논리식으로부터 deadlock, livelock, reachability 그리고 liveness유무를 검사하기 위해 B.cleaveland와 B.steffen에 의해 개발된 model checking 알고리즘인 Solve를 사용한다[7]. Solve 알고리즘은 modal mu-calculus의 재귀적 특성을 이용한 것으로, 논리식 L를 다음과 같이 둔다.

(νX.A∧[a]X) ∨ (μY.B∨([a]Y∧⟨a⟩tt))

maximum fixed point νX.Φ는 논리식 Φ에 의해 프로세스 X가 항상 참임을 의미하고, minimum fixed point로 표현된 μY.Φ는 논리식 Φ에 의해 프로세스 Y가 부분적으로 참임을 의미한다. 위 논리식 Lμ는 행위 a만을 가지는 LTS에 대해 atomic proposition A는 항상 참이고, 부분적으로 atomic proposition B도 참임을 의미한다.

완전한 논리식 Lμ에 대한 변수 X1을 생성하고, maximum fixed point와 minimum fixed point로 표현된 부 논리식 νX.Φ과 μY.Φ에 대한 변수 X2와 X3를 생성한다. X1을 좌변에 두면 max{X1 = X2∨X3}인 초기 max block을 얻게되고, 변수 X2의 등식 (X4 ∧ X5)을 생성하여 max block에 첨가한다. max block에서 부 논리식 νX.Φ에 대해 위 과정을 반복적으로 적용하여 다음의 완전한 max block을 얻는다.

$$\begin{aligned} \max\{X_1 &= X_2 \vee X_3 \\ X_2 &= X_4 \wedge X_5 \\ X_4 &= A \\ X_5 &= [a]X_2\} \end{aligned}$$

변수 X3와 관련한 minimum fixed point로 표현된 부 논리식 μY.Φ에 대한 min block을 위의 과정을 적용하여 다음과 같이 생성한다.

$$\begin{aligned} \min\{X_3 &= X_6 \vee X_7 \\ X_6 &= B \\ X_7 &= X_8 \wedge X_9 \\ X_8 &= [a]X_3 \\ X_9 &= \langle a \rangle X_{10} \\ X_{10} &= tt\} \end{aligned}$$

위의 전체식 Lμ는 max와 min의 두 block으로 나누어지는데, LTS의 각 상태는 위에서 파생된 Xi와 관련된 bit-vector와 counter를 가지고 아래의 초기화

규칙을 따라 초기화한 다음 상태-변수짝의 리스트인 배열 M[1...m]을 작성하고, 파생된 변수 Xi의 천이 관계를 나타낸 edge-labeled directed graph G를 만든 후, 갱신 알고리즘에 의해 배열 M의 원소가 공집합(empty)이 될 때까지 bit-vector와 counter의 내용을 갱신한다.

Bit-vector는 갱신된 결과에 의해 LTS의 상태가 변수 Xi의 등식을 만족하는지를 나타낸다. 즉 deadlock과 도달 가능한 상태 및 발생 가능한 행위를 검출한다. 갱신된 counter의 내용은 livelock이 발생한 LTS의 상태를 검출한다. 따라서, 갱신된 bit-vector와 counter의 내용은 LTS의 상태와 관련된 정보를 나타낸다.

다음은 bit-vector, counter, 배열 M에 대한 초기화 규칙이다.

4.1.1 규칙 1

가. bit-vector를 초기화하는 규칙은 다음과 같다.

① max block좌변의 변수 Xi와 관련된 bit-vector 값을 true(1)로 초기화한다. 만일 우변이 atomic proposition이고, 상태 Xi가 그 atomic proposition을 만족하지 않으면 false(0)로 초기화한다. 우변이 ⟨a⟩Xj형태이고, 상태 Si가 a천이를 가지지 않으면 false(0)로 초기화한다.

② min block 좌변의 변수Xi와 관련된 bit-vector 값을 false(0)로 초기화한다. 만일 우변이 atomic proposition이고, 상태 Si가 atomic proposition을 만족하면 true(1)로 초기화하고, 우변이 [a]Xj형태이고 상태Si가 a천이를 가지지 않으면 true(1)로 초기화한다.

나. Counter를 초기화하는 규칙은 아래와 같다.

① max block내의 항등식이 다음과 같을 때 적용된다.

- 만약 Xi = Xj∨Xk이면 S.C[i]는 S에 Xi 우변의 disjunct들이 참인 갯수로 초기화한다.
- 만약 Xi = ⟨a⟩Xj이면 S.C[i]에 상태S에 의한 a 천이의 갯수로 초기화한다.

② min block내의 항등식이 다음과 같을 때 적용된다.

- 만약 Xi = Xj∧Xk이면 S.C[i]에 Xi 우변의 conjunct들이 거짓인 갯수로 초기화한다.
- 만약 Xi = [a]Xj이면 S.C[i]에 상태S에 의한 a 천이의 갯수로 초기화한다.

배열 M에 대한 규칙은 다음과 같다.

상태-변수 쌍의 리스트인 배열 M[1...n]은 S.X[i] 값이 변화될 때마다 ⟨S, Xi⟩를 배열 M에 첨가한다.

위의 초기화 규칙에 의해 모두 초기화가 되어졌다면, 아래의 갱신 알고리즘을 적용하여 배열M의 ⟨S, Xi⟩가 없어질 때까지 bit-vector, counter, 배열 M을 반복적으로 갱신한다.

4.1.2 규칙 2

가. Xi가 Max Block에 존재한다면 알고리즘은 다음과 같다

① Xk가 Max Block에 존재한다.

- $X_i \vee X_k$인 Xk가 max block의 좌변에 존재하면, S.C[k]는 1감소하고, S.C[k]가 0이면, S.X[k]는 거짓(0)으로 설정하고, ⟨S, Xk⟩를 M에 첨가
- $X_i \wedge X_k$인 Xk에 대해 만약, Xk 가 max block의 좌변에 존재하고, S.X[k]가 참(1)이면 S.X[k]를 거짓으로 바꾸고, ⟨S, Xk⟩를 M에 첨가
- $X_i \xrightarrow{a} X_k$인 Xk에 대해 만약, Xk가 max block의 좌변에 존재하고, $S' \xrightarrow{a} S$인 상태 S'에 대한 S'.C[k]를 1로 감소시키고, 만일 0이 된다면 S'.C[k]는 거짓(0)으로 설정하고, ⟨S', Xk⟩를 M에 첨가
- $X_i \xrightarrow{[a]} X_k$인 Xk에 대해 만약, Xk가 max block의 좌변에 존재하고, $S' \xrightarrow{a} S$인 상태 S'에 대한 S'.X[k]를 조사하여 1이면 0으로 설정하고 ⟨S', Xk⟩을 M에 첨가

② Xk가 Min Block내에 존재

- $X_i \vee X_k$인 Xk가 min block의 좌변에 존재하고, S.X[k]가 거짓(0)이면, S.X[k]는 참(1)으로 설정하고 ⟨S, Xk⟩를 M에 첨가
- $X_i \wedge X_k$인 Xk가 min block의 좌변에 있다면, S.C[k]를 1감소 시키고, 만일 S.C[k]가 0이면, S.X[k]는 참(1)으로 설정하고 ⟨S, Xk⟩를 M에 첨가

- X_i<a>X_k 인 X_k가 min block의 좌변에 존재하면, S'^{a}.S인 S'에 대한 S'.X[k]가 거짓이면 참으로 설정하고 〈S', X_k〉를 M에 첨가
- X_i[a].X_k인 X_k가 min block의 좌변에 있다면, S'^{a}.S인 S'에 대한 S'.C[k]를 1 감소 시키고, 만일 0이면 S'.X[k]는 참(1)으로 설정하고, 〈S', X_k〉를 M에 첨가

X_i는 배열 M에 저장된 요소이고, edge-labeled directed graph G에 의해 X_k로 천이된다. 이때 사용되는 연산자에 의해 위의 규칙 중 하나가 선택된다. 또한 S'는 행위 a에 의해 S로 천이되는 상태를 나타낸다.

나. X_i가 Min Block에 존재한다면 알고리즘은 다음과 같다

① X_k가 Min Block내에 존재

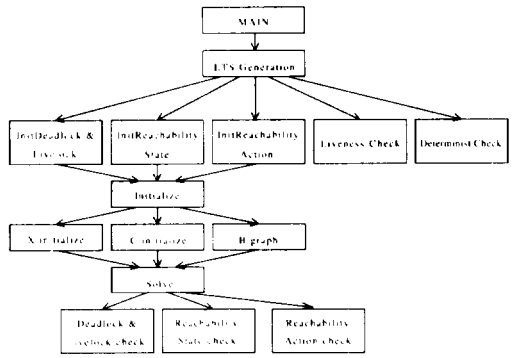
- X_i ..X_k인 X_k에 대해 S.C[k]는 1감소 하고, 만일 S.C[k]가 0이면, S.X[k]는 참(1)으로 설정하고 〈S, X_k〉를 M에 첨가
- X_i ..X_k인 X_k에 대해 S.X[k]가 거짓이면, S.X[k]는 참(1)으로 설정하고, 〈S, X_k〉를 M에 첨가
- X_i[a]X_k인 X_k에 대해 S'^{a}.S인 S'에 대한 S'.C[k]를 1감소하고, 만일 0이면 S'.X[k]는 참(1)으로 설정하고, 〈S', X_k〉를 M에 첨가
- X_i<a>X_k인 X_k에 대해 S'^{a}.S인 S'에 대한 S'.X[k]가 거짓이면 참으로 설정하고 〈S', X_k〉를 M에 첨가

② Xk가 Max Block내에 존재

- X_i ..X_k인 X_k에 대해 S.X[k]가 참(1)이면 S.X[k]를 거짓으로 설정하고, 〈S, X_k〉를 M에 첨가
- X_i ..Xk인 X_k에 대해 S.C[k]는 1감소하고, 만일 0이면, S.X[k]는 거짓(0)으로 설정하고, 〈S, X_k〉를 M에 첨가
- X_i[a]X_k인 X_k에 대해 S'^{a}.S인 상태S'에 대한 S'.C[k]를 조사하여 참(1)이면 거짓(0)으로 설정하고 〈S', X_k〉을 M에 첨가
- X_i<a>X_k인 X_k에 대해 S'^{a}.S인 S'에 대한 S'.C[k]를 1을 감소시키고, 만일 0이 된다면 S'.C[k]는 거짓(0)으로 설정하고, 〈S', Xk〉를 M에 첨가

4.2. Borland C^{++} 을 이용한 알고리즘 구현

LTS 프로토콜 명세에 대한 Borland C^{++} 언어로 구현된 모델 검증기 모듈 구성은 (그림3)과 같다.



(그림 3) 검증기 모듈 구성도
(Fig. 3) The Configuration of Model Checking Modules

모듈은 전체적으로 LTS 생성, deadlock과 livelock검증, 상태와 행위에 대한 reachability검증, liveness, deterministic 등의 기능을 수행하는 부분으로 구성되어 있다.

다음은 모델검증을 위한 각 모듈에 대한 기능 설명이다.

① LTS Generation
현재 상태(Si), 행위(Action), 다음 상태(So)로 구성된 레이블 천이시스템의 논리적 구조를 메모리상에 기억

② InitDeadlock & livelock
Deadlock & livelock을 검사하기 위한 modal mu-calculus 논리식
νZ. (μY.A ∧((-) tt∧[-]Y))∧[-]Z, A = {S0}
을 Solve알고리즘에 따라 max, min block 으로 분리

③ InitReachability State
상태에 대한 reachability를 검사하기 위한 modal mu-calculus논리식
νZ.(μY.A∨(⟨action2⟩ tt∧[action1]Y))∧[-]Z,
A = {state} 을 Solve알고리즘에 따라 max, min block으로 분리

④ InitReachability State

행위에 대한 reachability를 검사하기 위한 modal mu-calculus논리식
νZ.[action1](μY.(⟨action3⟩tt[action2]Y))∧[-]Z을 Solve알고리즘에 따라 max, min block 으로 분리

⑤ X.initialize
Bit Array를 Solve알고리즘의 초기화 규칙에 따라 초기화

⑥ C.initialize
Counter Array를 Solve알고리즘의 초기화 규칙에 따라 초기화

⑦ B.graph
Modal mu-calculus논리식에 관련된 블럭을 가지고 edge-labeled directed graph를 생성

⑧ Deadlock and livelock check
주어진 LTS의 deadlock과 livelock 상태를 검출

⑨ Reachability for the State check
두개의 행위와 한개의 상태를 입력 후 두번째 행위가 주어진 상태에 도달 가능한지 검사

⑩ Reachability for the Action check
세개의 행위를 입력 후 입력한 세개의 행위가 순서대로 발생하는지 검사

⑪ Liveness check
초기 상태에서 도달 가능한 모든 상태와 행위를 화면에 출력

⑫ Deterministic check
어떤 상태의 특정 행위에 대해 다음 상태가 두가지 이상 존재하는지 검사

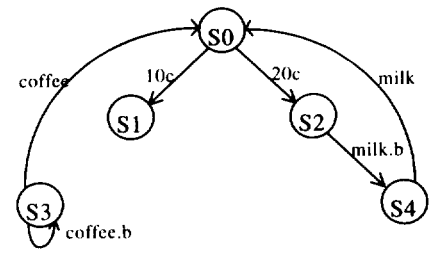
구현된 알고리즘에 의해 초기화된 Bit array와 Counter array를 갱신한 후 그 결과로써 주어진 LTS가 논리식을 만족하는지 검사한다. 제시된 알고리즘의 동작을 다음 소절에서 다양한 예로써 설명한다.

4.3. 검증 적용의 예

본 소절에서는 LTS에 대한 deadlock, livelock, reachability, liveness검증에 대해 앞 소절에서 기술한 알고리즘을 적용하여 검증 결과를 검출한다.

4.3.1. Deadlock 및 livelock 검증

(그림4)는 (그림2)의 LTS에서 상태 S1에 deadlock, 상태 S3에서 livelock이 발생한 LTS모델이고, atomic proposition은 상태 S0이다.



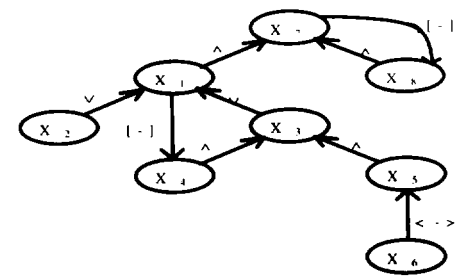
(그림 4) Deadlock 및 Livelock LTS 모델
(Fig. 4) LTS Model for Deadlock and Livelock

Deadlock 및 livelock에 대한 modal mu-calculus의 식은 다음과 같다.

$$\nu Z.\ (\mu Y.A \vee (\langle - \rangle\ tt \wedge [-]Y)) \wedge [-]Z \quad 단,\ A = \{S0\}$$

논리식은 LTS의 행위 집합 S에 포함된 어떠한 행위가 발생하더라도 그 LTS를 만족해야 하고, atomic proposition A를 만족함을 나타낸다.

앞 소절의 검증 알고리즘을 적용하면 다음과 같다. (그림5)는 변수 Xi들의 천이 관계를 나타낸 edge-labeled directed graph G를 나타낸다.



(그림 5) Edge-labeled directed graph G
(Fig. 5) Edge-labeled directed graph G

$B_1 \equiv \min\{X_1 = X_2 \vee X_3$ $X_2 = A$ $X_3 = X_4 \wedge X_5$ $X_4 = [-]X_1$ $X_5 = \langle - \rangle X_6$ $X_6 = tt\}$	$B_2 \equiv \max\{X_7 = X_1 \wedge X_8$ $X_8 = [-]X_7\}$

(그림 6) Max block, Min block
(Fig. 6) Max block, Min block

(그림6)은 maximum fixed point와 minimum

fixed point를 사용하여 생성된 max block과 min block를 나타내며, (그림7)은 초기화 규칙에 의해 초기화 된 bit-vector, counter, 배열 M[i]을 나타낸다.

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
S0	0	1	0	0	0	1	1	1
S1	0	0	0	1	0	1	1	1
S2	0	0	0	0	0	1	1	1
S3	0	0	0	0	0	1	1	1
S4	0	0	0	0	0	1	1	1

C	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
S0	0	0	2	2	0	0	0	0
S1	0	0	1	0	0	0	0	0
S2	0	0	2	1	0	0	0	0
S3	0	0	2	1	0	0	0	0
S4	0	0	2	2	0	0	0	0

M[1]=〈〈S0, X2〉, 〈S1, X4〉, 〈S0, X6〉, 〈S1, X6〉, 〈S2, X6〉, 〈S3, X6〉, 〈S4, X6〉〉
M[2]=〈 〉

(그림 7) Bit-vector, Counter and Array M[i]
(Fig. 7) Bit-vector, Counter and Array M[i]

(그림8)은 배열 M[i]가 공집합이 될 때까지 Solve 알고리즘을 적용하여, bit-vector및 counter를 갱신한 결과이다. Deadlock은 어떠한 행위에 대해서도 참임을 나타내는 변수 X5의 등식 (-)tt를 만족하지 않는 상태 〈S1, X5〉가 0으로 갱신됨으로써 검출되어진다. 또한 livelock은 관련된 상태의 counter요소에 의해 판단된다. (그림8)의 counter에서 상태 S3와 관련된 요소는 1로 갱신된다. 이는 상태 S3가 safety를 표현한 부 논리식 $\mu Y.A \vee ((-)tt \wedge [-]Y)$과 관련된 변수 X3 및 X4를 만족하지 않으므로 livelock이 발생한 상태 S3를 검출한다.

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
S0	1	1	1	1	1	1	1	1
S1	1	0	1	1	0	1	1	1
S2	1	0	1	1	1	1	1	1
S3	1	0	1	1	1	1	1	1
S4	0	0	0	0	1	1	0	0

C	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
S0	0	0	0	0	0	0	0	0
S1	0	0	0	0	0	0	0	0
S2	0	0	0	0	0	0	0	0
S3	0	0	1	1	0	0	0	0
S4	0	0	0	0	0	0	0	0

M[1]=〈 〉 M[2]=〈 〉

(그림 8) Bit-vector, Counter and Array M[i]
(Fig. 8) Bit-vector, Counter and Array M[i]

4.3.2. Reachability

reachability검증은 LTS의 초기 상태로부터 정의된 천이의 순서에 의해 일부 또는 모든 다른 상태에 도달 가능한지 검사한다.

앞에서 제시된 (그림2)의 상태 S0에서 천이 10c, coffee.b, coffee에 의해 도달되는 상태가 LTS를 만족함을 나타내는 modal mu-calculus는 다음 식과 같다.

$\nu Z.[10c](\mu Y.\langle coffee\rangle tt \wedge [coffee.b]Y) \wedge [-]Z$

입력된 modal mu-calculus논리식은 앞에서 기술한 방법으로 bit-vector와 counter를 갱신하고 그 결과로써 주어진 LTS에 대해 논리식을 만족하는지 검사할 수 있다.

4.3.3. Liveness 및 deterministic

(그림3)에서 제시된 것처럼 liveness검증과 deterministic검증은 Solve알고리즘을 사용하지 않고 독립적으로 수행된다.

Liveness검증은 초기상태에서 도달가능한 모든 상태와 행위를 출력하는 것으로, (그림2)에 대한 검증결과는 다음과 같다.

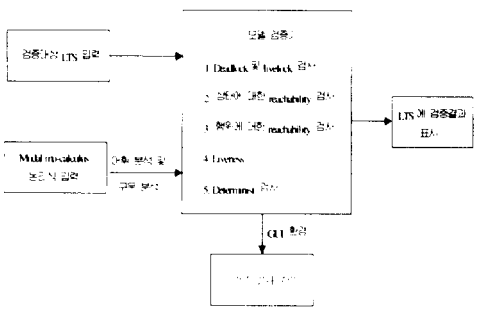
S0->10c->S1->coffee.b->S3->coffee->S0
S0->20c->S2->milk.b->S4->milk->S0

Deterministic검증은 어떤 상태의 특정행위에 대해 다음 상태가 두가지 이상 존재하는가를 검사하는 것으로, (그림2)의 경우 deterministic한 결과를 얻는다.

5. 결 론

본 논문은 LOTOS 명세의 행위 명세 부분을 중간

모델로 변환한 레이블 천이 시스템에서 교착상태 유무 조사와 주어진 천이 시스템이 임의의 상태에 대하여 초기 상태에서 도달 가능한지 검사하는 기능을 Borland C++을 이용하여 IBM PC 호환기종에서 동작하도록 구현하였다. 이러한 기능을 구현하기 위해 사용자에 의해 입력된 논리식을 어휘 분석과 구문 분석과정을 거친 후 B.cleaveland와 B.steffen에 의해 개발된 model checking 알고리즘인 Solve를 적용하여 교착상태 유무, 임의의 상태에 대하여 초기 상태에서 도달 가능한지를 판단하였다. 또한 사용자환경에서 쉽게 다룰 수 있도록 Windows NT환경 하에서 GUI기능에 의해 검증결과를 Window에 나타내었다. 구현된 검증기에 대한 전체 구성도는 (그림9)와 같다.



(그림 9) 검증기 전체 구성도
(Fig. 9) The Whole Configuration of Model Checker

향후 연구 사항으로는 두개 이상의 fixed point를 사용한 여러 가지 modal mu-calculus논리식에 대해서 검증할 수 있는 기능을 구현하는 것이다. 또한 시간 개념이 추가된 실시간 시스템에 대한 검증기능 구현도 향후 추진되어야 할 연구 사항이다.

참 고 문 헌

[1] D.schwabe, Formal Techniques for the Specification and Verification of Protocol, Ph.D Thesis, Univ. of California Los Angeles, Apr., 1981.

[2] Ashankar and S. Lam, Specification and Verification of Time-Dependent Communica-tion Protocols, Proc. of the 4rh IFIP Intl Workshop on Protocol Specification, Testing and Verification, North-Holland, Jun, 1984.

[3] A.teng and M.Liu, A Formal Model for Automatic Implementation Logical Validation of Network communication Protocol, NBS Computer Networking Symp., pp.114-123, 1978.

[4] J.M. Ayache, specification and Verification of Signalling Protocol International Switch-ing symposium, May, 1979.

[5] P.V.Koppol and K.C.Tai,Conformance Testing of Protocol specification as Labeled Transition system, International Workshop on Protocol Test System, IWPTS95, pp.143-158, Evry, France, September, 1995.

[6] R. Cleaveland, M. Dreimuller and B.steffen, Faster Model-checking for the Modal Mu-Calculus. To appear in Processings of the Workshop on Computer-Aided Verification, Lecture Notes in Computer Science.

[7] A.Arnold and B. Crubille, A Linear Algorithm To Solve Fixed-Point Equations on Transition System, Information Processing Letters 29:57-66, 30 September 1988.

박 용 범

1986년 경북대학교 전자공학과 졸업(학사)

1989년 한국과학기술원 전산학과 졸업(석사)

1989년~현재 한국전자통신연구원 선임연구원

관심분야 : 프로토콜 시험, 분산시스템

김 태 균

1998년 부경대학교 정보통신공학과 (학사)

1998년~현재 부경대학교 정보통신공학과 석사 과정

관심분야 : 프로토콜 공학 및 시험, 실시간 시스템

김 성 운

1982년 경북대학교 전자공학과 (공학사)

1990년 프랑스 국립파리 7 대학교 정보공학과(공학석사)

1993년 프랑스 국립파리 7 대학교 정보공학과(공학박사)

1982년~1985년 한국전자통신연구소 데이터통신연구실(연구원)

1986년~1995년 한국통신 연구개발원(선임연구원, 실장)

1990년~1993년 프랑스 전기통신기술연구소(초빙연구원)

1995년~현재 부경대학교 정보통신학공학과 교수

관심분야 : 프로토콜 엔지니어링, 데이터 통신, 통신프로토콜시험, 컴퓨터네트워크