외부환경을 고려한 ESTEREL 프로그램 검증

안윤경[O1], 전진성[2], 한태숙[1]
[1]한국과학기술원 전자전산학과 프로그래밍언어 연구실
yunkyung.ahn@pllab.kaist.ac.kr, han@cs.kaist.ac.kr
[2]국방과학연구소
jinseong.jeon@add.re.kr

Taking Environments into account for ESTEREL Program Verification

Yunkyung Ahn[O1], Jinseong Jeon[2], Taisook Han[1]
[1]Programming Language Laboratory, Dept. of Electronic Engineering and Computer Science, KAIST
[2]Agency for Defense Development

요 약

프로그램 검증은 가능한 모든 경우에 대하여 안정성을 확인하는 작업이다. reactive 프로그램을 외부환경에 대한 정보 없이 검증하면 실제로 가능하지 않은 경우를 오류로 찾을 수도 있다. 본 논문에서는 observer를 이용한 기존의 ESTEREL 프로그램 검증 기법에 외부환경에 대한 가정을 추가하는 방법을 소개하고, 사례연구를 통해 외부환경을 고려하여 검증하는 과정을 보인다.

1. 연구 배경

ESTEREL[1]은 임베디드 시스템, 통신 프로토콜 등의 반응형 시스템(reactive system)을 프로그래밍하기 위해 개발된 동기적(synchronous) 언어이다. ESTEREL로 작성된 반응형 프로그램은 외부환경(environment)과 끊임없이 상호 작용한다. 따라서 ESTEREL 프로그램이 안전한지 검증하기 위해서는 프로그램뿐만 아니라 외부환경의 행동(behavior)에 대한 고려가 필요하다.

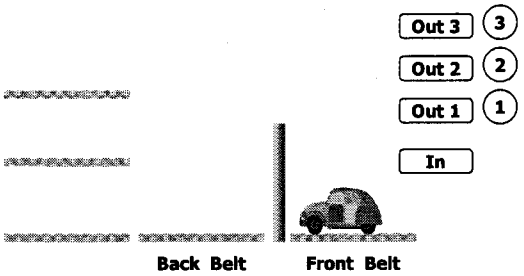
기존의 observer를 이용한 ESTEREL 프로그램 검증방법[2]은 외부환경에 대한 모델링을 필요로 하지 않는다. 따라서 외부환경에 대한 제약 없이 가능한 모든 입력에 대해 프로그램이 안전한지 검증하게 된다. 이 기법을 이용하면 프로그램은 올바르게 동작하지만, 외부환경이 예상대로 동작하지 않아 오류가 발생할 수도 있다.

본 논문에서는 observer를 이용한 검증 방법을 확장하여 외부환경을 고려하는 방법을 제안한다. 모터, 센서, 상호작용하는 다른 프로그램 등의 외부환경을 고려함으로써, 외부환경이 기대한 행동을 할 때 ESTEREL 프로그램이 안전한지 검증할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 예제 시스템인 자동주차엘리베이터 시스템을 설명한다. 3장에서 observer를 이용하는 기존의 검증기법을 살펴본 후 이 기법으로 예제 시스템을 검증한 결과를 보이며 이때의 문제점을 살펴본다. 4장에서는 해결방법으로 기존의 observer를 확장하여 외부환경을 고려하도록 하는 방법을 소개하고, 제안한 방법으로 예제 시스템을 검증한 결과를 보인다.

2. 예제 시스템

본 논문을 통하여 사용할 예제로 [그림 1]과 같은 자동주차엘리베이터 시스템을 개발하였다.

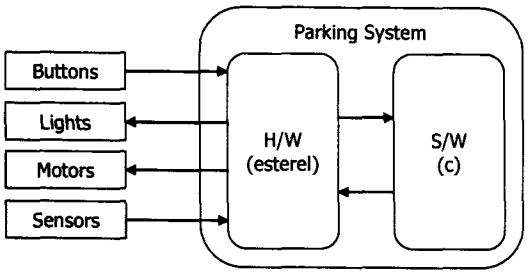


[그림 1] 자동주차엘리베이터 시스템

개발한 주차건물은 총 3층으로 이루어져 있고 각 층에는 한 대의 차를 주차할 수 있다. 문이 하나 있고 문 앞에 전방 벨트(Front Belt)와 문 바로 뒤에 차를 층간으로 이동시키는 후방 벨트(Back Belt)가 있다. 모든 층과 전방 벨트, 후방 벨트는 컨베이어 벨트로 만들어져 차를 앞뒤로 이동시킬 수 있다. 주차건물에는 4개의 버튼과 3개의 표시등이 있어 사용자와의 인터페이스(interface)를 제공한다. 차를 전방 벨트에 두고 In 버튼을 누르면 자동으로 차가 주차건물 안의 가장 가까운 층에 주차되고 해당되는 층의 표시등이 켜진다. Out 버튼을 누르면 해당 층에 주차된 차가 주차건물 밖의 전방 벨트로 옮겨지며 표시등이 꺼진다. 주차 시스템에는 총 7개의 모터와 10개의 센서가 있다. 각 벨트를 앞

뒤로 움직이는 모터와 후방 벨트를 위아래로 움직이는 모터, 그리고 문을 열고 닫는 모터가 필요하다. 또한 각 벨트 위에 차가 완전히 올라왔는지 알려주는 센서와 후방 벨트가 몇 층에 있는지 알려주는 센서, 그리고 문이 완전히 닫혔음을 알려주는 센서와 완전히 열렸음을 알려주는 센서가 필요하다.

자동주차시스템의 하드웨어 부분은 ESTEREL로 작성하고 소프트웨어 부분은 C언어로 작성하였다. 하드웨어는 사용자 요청을 소프트웨어에게 알려주고 소프트웨어의 신호에 따라 센서를 확인하며 각 모터를 동작시키는 일을 한다. 시스템에서 하드웨어와 소프트웨어의 분할 모습은 [그림2]와 같다. ESTEREL 프로그램은 C 프로그램, 모터, 센서등과 끊임없이 반응한다.

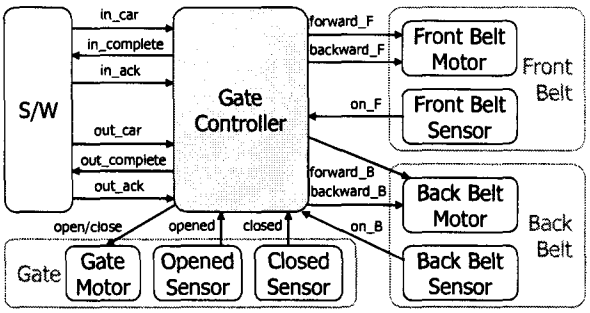


[그림 2] ESTEREL 프로그램과 외부환경

주차 시스템에서 하드웨어는 총 5개의 모듈로 이루어져 있다. 버튼을 통하여 들어오는 사용자 요청을 소프트웨어에 전달하는 모듈, 문을 제어하는 모듈, 후방 벨트를 위아래로 움직이는 모듈, 후방 벨트에서 해당 층으로 차를 넣거나 해당 층의 차를 후방 벨트로 빼는 모듈, 주차된 층에 해당하는 표시등을 켜는 모듈이다.

본 논문에서는 설명을 간략하고 명확하게 하기위해 문 제어기(Gate Controller)만을 살펴보도록 하겠다. [그림 3]은 문 제어기가 외부환경과 주고받는 신호(signal)들을 나타낸다. 문 제어기는 소프트웨어로부터 차를 주차장 안으로 넣어/빼 달라는 요청(*in_car/out_car*)을 받으면 안전하게 차를 주차장 안/밖의 후방/전방 벨트에 올려둔 다음 소프트웨어에게 차를 넣었음/뺐음을 알린다(*in_complete/out_complete*).

문 제어기의 동작 중 만약 문이 완전히 열리지 않은 상태에서 차를 주차건물 안으로 넣으려고 하면 차에 손상이 생길 수 있다. 그러므로 차를 주차 건물로 넣을 때에는 문이 완전히 열려있어야 하며 이것은 시스템이 만족해야하는 안전성 속성(safety property)들 중에 하나이다. 우리는 앞으로 주차장 시스템이 이 속성을 만족하는지 검증할 것이다.



[그림 3] 문 제어기와 외부환경과의 인터페이스

3. 기존 연구 및 문제점

3.1 Observer를 이용한 검증방법

ESTEREL 프로그램이 안전성 속성을 만족하는지 observer를 이용하여 검증하는 방법이 있다. 여기서 observer는 검증할 ESTEREL 프로그램과 동시에(parallel) 수행되면서 프로그램이 속성을 만족하지 않는 순간 violated 신호를 발생시키는 ESTEREL 프로그램이다.

프로그램이 만족해야하는 안전성 속성은 LTL(Linear time Temporal Logic)[3]로 표현한다. LTL은 논리연산자(boolean operator)에 선형시제 연산자(linear temporal operator)를 추가한 것으로 시스템과 관련된 신호들이 어떤 순서로 일어나야 하는지 기술할 수 있다. 시제 연산자는 미래시제(future-tense) 연산자와 과거시제(past-tense) 연산자로 구별된다. ESTEREL 프로그램의 안전성 속성을 표현하기 위해서 언어의 특성을 고려하여 과거시제 연산자를 선택한다. 과거시제 연산자를 이용한 식은 모델의 현재 상태와 그 이전의 상태들이 만족해야하는 속성을 기술한다. 안전성 속성을 표현한 안전성 식 s의 정의는 다음과 같다. 이때 n은 임의의 자연수이고, a는 프로그램에 존재하는 신호 이름이거나 상수 *TRUE, FIRST*이다.

$$
\begin{aligned}
s &:= \Box\, p \\
p &:= a \mid (p) \mid \neg p \mid p \wedge p \mid p \vee p \\
&\quad \mid p \rightarrow p \mid p \rightarrow n\, p \mid \ominus\, p \mid \ominus : n\, p \\
&\quad \mid \boxminus\, p \mid \diamondsuit\, p \mid p\, S\, p \mid p\, B\, p
\end{aligned}
$$

정의된 안전성 속성에 대한 observer를 손으로 직접 작성할 수도 있으나 이는 매우 번거로운 작업이며 실수하기 쉽다. 또한 속성이 복잡할수록 직접 observer를 작성하는 것이 어려우므로 자동화 과정이 필요하다. 안전성 식으로부터 observer를 자동으로 생성하는 기존의 연구[3]를 기반으로 observer를 생성하고 검증할 프로그램과 observer가 동시에 수행되는 새로운 프로그램을 생성하는 도구를 구현하였다.

안전성 식 s는 □ p (always p)형태이며 모든 상태(state)에서 p가 만족함을 의미한다. 여기서 p는 과거식이다. 과거식 p에 대해서 p가 만족되는 순간마다 *sat_p* 신호를 발생하는 프로그램 부분(fragment)으로의 변환을 SAT_p라 하자. 변환은 구조적(structural), 재귀적(recursive)으로 정의된다. 우선 p가 단순히 신호이름인 경우에는 신호 p가 존재(present)할 때 마다 *sat_p*를 발생하는 프로그램으로 변환한다. 상수 *TRUE*의 변환은 매 순간 *sat_true*를 발생시키며, *FIRST*의 경우는 최초의 순간에만 한번 *sat_first*를 발생시키는 프로그램을 생성한다.

논리 연산자인 ¬ ,∧,∨,→ 에 대한 변환 역시 연산자의 의미에 의해 쉽게 정의될 수 있다. 이 중 ∨에 대한 변환은 아래와 같다. SAT_p는 p가 만족되는 순간 *sat_p*를 발생하고, SAT_q는 q가 만족되는 순간 *sat_q*를 발생한다. 이때 이 둘을 동시에 수행시키면서 *sat_p*나 *sat_q*가 발생하는 순간 $p \vee q$가 만족됨을 뜻하는 *sat_p_or_q*신호를 발생시킨다.

$SAT_{p \vee q}$ =

```
SATp
||
SATq
||
[
  every immediate tick do
    present [ sat_p or sat_q ] then
      emit sat_p_or_q ;
    end present;
  end every
]
```

과거시제 연산자인 ⊖, ⊟, ◇, S, B (previously, has-always-been, once, since, back-to) 에 대한 변환 중 ⊖에 대한 변환의 정의는 아래와 같다. 현재 상태에서 ⊖ p가 만족한다는 것은 바로 이전 상태에서 p가 만족함을 뜻한다. 따라서 $SAT_{\ominus p}$는 p가 만족하는 순간을 확인하여 바로 다음 순간 *sat_P_p*신호를 발생한다.

$SAT_{\ominus p}$ =

```
SATp
||
[
  loop
    present [ sat_p ] then
      pause;
      emit sat_P_p;
    else
      pause;
    end present;
  end loop
]
```

SAT_p를 이용하여 안전성 식 s가 만족하지 않는 순간 *violated* 신호를 발생시키는 Safety 모듈로의 변환을 $SAFETY_s$라 하자. $SAFETY_s$에 대한 정의는 아래와 같다. 입력신호 *I_p*는 상수 *FIRST*와 *TRUE*를 제외한 p에 있는 신호들의 이름이다. *SAT_p*에 의해 발생되는 신호들 중 입력 신호에 없는 신호들인 *L_p*는 내부 신호(local signal)이다.

$SAFETY_s$ =

```
module Safety:
  input I_p;
  output violated;

  signal L_p in
    SATp
    ||
    [
      await immediate [not sat_p];
      emit violated;
    ]
  end signal
end module
```

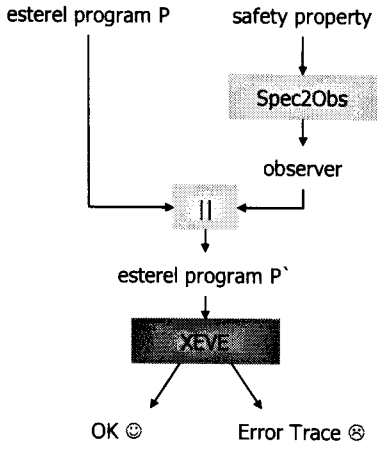
여러 개의 안전성 식이 주어진 경우 각각의 식들은 해당 Safety 모듈로 변환되고 이 모듈들을 동시에 수행시키는 상위 모듈인 Checker가 생성한다. 안전성 식이 n개 주어진 경우 생성된 Checker의 모습은 아래와 같다. 여기서 *I_p*는 Safety 모듈들의 입력 신호들을 모은 것이다.

```
module Checker:
  input I_p ;
  output violated_1, ···, violated_n ;

  run Safety_1
  ||
  ···
  ||
  run Safety_n
end module
```

끝으로 검증할 ESTEREL 프로그램과 Checker 모듈이 동시에 수행하는 새로운 프로그램을 생성한다. Verify 모듈은 생성되는 프로그램의 최상위 모듈이다. 아래는 n개의 안전성 속성들이 주어진 경우 생성된 Verify 모듈이다. E는 검증할 ESTEREL 프로그램이고 *I_E*는 E의 입력 신호들, *O_E*는 E의 출력 신호들이다.

```
module Verify:
  input I_E;
  output O_E;
  output violated_1, ···, violated_n ;

  run E
  ||
  run Checker
end module
```

[그림4]는 observer를 이용하여 ESTEREL 프로그램이 안전성 속성을 만족하는지 검증하는 전체과정을 보여준다. 검증할 ESTEREL 프로그램 *P*와 만족해야 하는 속성들이 주어지면 자동으로 observer를 생성한다. 그 다음 observer와 *P*가 동시에 수행되는 새로운 프로그램 *P`*을 생성한다. *P`*은 프로그램 *P*가 속성을 만족하지 않는 순간 *violated* 신호를 발생시킨다. 이제 프로그램이 수행 중 *violated* 신호가 발생할 가능성이 있는지 검사해야 한다. 이를 위해 ESTEREL 프로그램을 위한 모델체커인 XEVE[4]를 이용한다. XEVE는 선택된 특정 출력 신호의 발생 가능성을 검사하는 기능을 가지고 있다. 만약 *violated* 신호가 절대 발생하지 않는다면(never emitted), ESTEREL 프로그램이 속성을 만족하는 것이다. 그렇지 않은 경우에는 XEVE가 *violated* 신호가 발생하는 상태에 도달하는 입력신호들의 시퀀스를 생성해 주어 에러의 원인을 찾도록 돕는다.



[그림 4] observer를 이용한 검증방법

3.2 실험 및 문제점

2장에서 설명한 자동주차시스템에서 차를 주차 건물로 넣는 도중에는 항상 문이 완전히 열려있는 상태임을 검증하고자 한다. 이 안전성 속성은 LTL로 다음과 같이 표현된다. 이 식은 전방 벨트를 앞쪽으로 움직일 때(*forward_F*)마다 문이 완전히 열린 상태(*opened*)임을 의미한다.

$$\square\ (forward\_F \rightarrow opened)$$

기존의 observer를 이용한 방법으로 검증한 결과 주차시스템이 속성을 만족하지 않는 경우가 발생했다. 이는 문이 완전히 열린 상태(*opened*)에서 문을 닫는 행동을 하지 않았음에도 열린 상태가 계속 유지되지 않기 때문이었다. 주차시스템의 문 제어기는 문이 완전히 열린 상태임을 확인한 후 차를 주차 건물로 넣으며 그 도중에는 문을 닫으라는 신호를 모터에 보내지 않는다.

기존 방법은 ESTEREL 프로그램의 외부환경에 대한 모델링을 필요로 하지 않으므로 프로그램을 모든 가능한 외부환경에 대해 검증한다. 우리의 예에서 문 센서의 행동은 문 모터가 어떻게 움직이는지와 관련이 있다. 즉, 프로그램이 문을 열도록(닫도록) 모터를 움직이면 문 센서는 언젠가는 문이 완전히 열렸음(닫혔음)을 알릴 것이다. 또한 문이 완전히 닫히거나 열린 상태일 때 문 모터를 동작시키지 않는다면 문의 상태는 그대로 유지될 것이다. 그러나 기존 방법은 문 센서에 대한 어떤 가정도 하지 않기 때문에 문 센서의 출력신호는 어떻게도 발생할 수 있다. 즉, 문 센서인 opened(closed) 센서는 문이 완전히 열렸을(닫혔을) 때 *opened(closed)* 신호를 발생하는데, 이 경우 매 순간 센서는 *opened* 신호나 *closed* 신호 또는 두 신호를 동시에, 혹은 어떤 신호도 발생시키지 않을 수 있다. 기존 검증 방법은 이 모든 경우에 대해 프로그램이 속성을 만족하는지 검증하게 된다. 따라서 외부환경이 예상대로 동작할 때 ESTEREL 프로그램이 안전한지 검증하기 위해서는 외부환경에 대한 모델링을 해주어야 한다.

4. 외부환경을 고려하는 검증 방법

외부환경이 가정대로 동작할 때 ESTEREL 프로그램이 안전성 속성을 만족하는지 검증하고자 한다. 이를 위해 기존의 observer를 이용한 방법에서 *violated* 신호가 발생되는 순간, 처음부터 그 순간까지 외부환경이 가정대로 동작해왔는지 검사하도록 하였다. 우리는 외부환경이 원하는 대로 동작하는 경우에 안전성 속성을 만족하지 않는 것만을 고려할 것이다.

우리는 외부환경의 행동 역시 안전성 식으로 표현할 수 있음을 착안하였다. 외부환경의 행동이 안전성 식 $\square\ e$ 으로 표현이 된다고 하자. 이전 장에서 소개한 방법의 $SAT_{\square e}$를 이용하여 매 순간 처음부터 그 순간까지 계속 외부환경이 가정대로 행동해 왔다면 *sat* 신호를 발생시키는 Assume 모듈을 생성할 수 있다. 생성되는 Assume모듈의 형태는 다음과 같다.

$ASSUME_e$ =
module Assume:
input I_e;
output sat_H_e;

signal L_e in
$SAT_{\square e}$
end signal
end module

생성된 Assume 모듈은 매 순간 처음부터 그 순간까지

외부환경이 만족되어왔는지 여부를 알려준다. 이를 이용하여 검증할 프로그램이 가정된 외부환경의 행동을 만족하면서 안전성 속성 s을 만족하지 않는 순간 *violated* 신호를 발생시키는 *SAFETYs* 모듈을 생성할 수 있다. 안전성 속성 s는 $\Box\, p$ 으로 표현이 된다고 하자. 안전성 속성을 만족하지 않는 순간 *violated* 신호를 발생시키게 되는데 이때 present 구문을 이용하여 *SAT_ENV*가 모두 존재(present)할 때만 *violated* 신호를 발생시킨다. *SAFETYs*의 형태는 다음과 같다. *SAT_ENV*는 Assume 모듈들의 출력신호들을 and 연산자로 연결한 것이다. 따라서 *violated* 신호는 안전성 속성을 만족하지 않는 순간에, 또한 처음부터 그 순간까지 외부환경이 가정된 행동을 만족하고 있었을 경우에만 발생한다. 여기서 입력신호들의 집합인 I_p에는 Assume 모듈들의 출력 신호들이 추가된다.

```
SAFETY_s  =  module Safety:
               input I_p;
               output violated;

               signal L_p in
                 SAT_p
                 ||
                 [
                   await immediate [not sat_p];
                   present SAT_ENV then
                     emit violated;
                   end present
                 ]
               end signal
             end module
```

외부환경에 대한 가정이 m개, 검증해야하는 속성이 n개 주어진 경우, Observer의 상위 모듈인 Checker 모듈의 모습은 다음과 같다. Safety 모듈들뿐만 아니라 외부환경에 대한 정보를 제공하는 Assume 모듈들도 함께 수행된다.

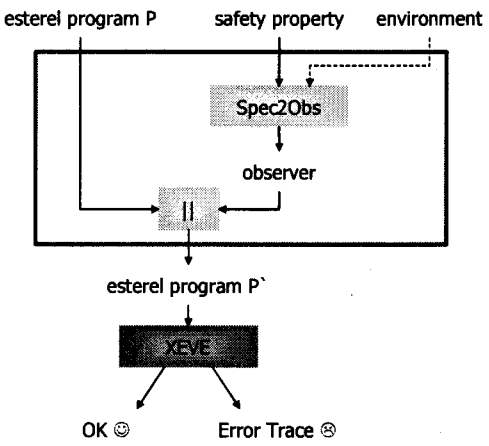
```
module Checker:
  input I_p ;
  output violated_1, ···, violated_n ;

  signal SAT_ENV in
    run Assume_1
    || ···
    run Assume_m
    ||
    run Safety_1
    || ···
    run Safety_n
  end signal
end module
```

[그림 5]는 외부환경을 고려하여 프로그램이 안전성 속성을 만족하는지 검증하는 전체 과정을 보여준다. 검증할 ESTEREL 프로그램 P와 만족해야하는 속성, 그리고 외부환경의 행동에 대한 가정이 주어지면 외부환경이 가정대로 동작하면서 안전성 속성이 만족하지 않는 경우 *violated* 신호를 발생시키는 observer를 생성한다. 그 다음 P와 생성된 observer가 동시에 수행하는 새로운 프로그램 P'을 생성한 후 XEVE를 이용하여 *violated* 신호가 발생될 가능성이 있는지 검사한다. 외부환경에 대한 가정을 주지 않는다면 기존 방법처럼 가능한 모든 외부환경에 대해 검증이 이루어진다. 검증 결과 *violated* 신호가 발생할 가능성이 없다면 외부환경이 가정대로 행동할 때 ESTEREL 프로그램이 안전성 속성을 만족함을 의미한다. *violated* 신호가 발생할 수 있다면 외부환경이 가정대로 동작함에도 ESTEREL 프로그램이 안전성 속성을 만족하지 않을 수 있음을 의미한다.

새로 제안한 방법으로 주차시스템예제를 검증하였다. 이전 검증의 경우 문 센서가 예상대로 행동하지 않아 안정성 속성을 어기게 되었다. 이제 아래와 같이 문 센서의 행동을 안전성 식으로 표현하여 검증하였다. 이 식은 현재 문이 완전히 열린 상태가 아니라면 바로 이전에도 문이 완전히 열린 상태가 아니었거나 문이 완전히 열렸다면 문 모터에 문을 닫으라는 신호(*close*)가 있었음을 의미한다.

$$\Box\,(\neg\ opened \rightarrow \ominus\,(\neg\ opened\ \vee\ close))$$

외부환경을 고려하여 생성된 observer와 XEVE를 이용하여 문 센서가 주어진 가정대로 행동하는 한 ESTEREL 프로그램은 안전성 속성을 만족함을 검증하였다.



[그림 5] 외부환경을 고려하는 검증방법

5. 결론

본 논문에서는 ESTEREL 프로그램을 검증할 때 외부환경을 고려하는 방법을 제안하였다. 기존 observer를 사용하는 검증방법이 외부환경에 대한 모델링을 하지 않아 모든 가능한 외부환경에 대해 검증이 이루어지는 반면, 우리의 방법은 외부환경의 행동을 LTL로 표현해 주면 외부환경이 주어진 가정대로 행동할 때 ESTEREL 프로그램이 안전한지 검증한다. 또한 예제로 자동주차시스템을 개발하여 기존의 방법과 제안한 방법으로 검증한 결과를 보였다.

참고문헌

[1] G. Berry. The Constructive Semantics of Pure Esterel. 1999.

[2] Lalita Jategaonkar Jagadeesan, Carlos Puchol, James E. Von Olnhausen. Safety Property Verification of ESTEREL Programs and Applications to Telecommunications Software. In Computer-Aided Verification. 1995

[3] Z.Manna and A.Pnueli. The Temporal Logic of Reactive and Concurrent Systems, Specification. Springer-Verlag, 1992

[4] Amar Bouali. XEVE: an ESTEREL Verification Environment(Version v1_3). 1997