KIPS Tr. Software and Data Eng.
Vol.5, No.12 pp.653~662 pISSN: 2287-5905



https://doi.org/10.3745/KTSDE.2016.5.12.653

# Test Case Generation for Simulink/Stateflow Model Based on a Modified Rapidly Exploring Random Tree Algorithm

Han Gon Park† · Ki Hyun Chung†† · Kyung Hee Choi†††

## ABSTRACT

This paper describes a test case generation algorithm for Simulink/Stateflow models based on the Rapidly exploring Random Tree (RRT) algorithm that has been successfully applied to path finding. An important factor influencing the performance of the RRT algorithm is the metric used for calculating the distance between the nodes in the RRT space. Since a test case for a Simulink/Stateflow (SL/SF) model is an input sequence to check a specific condition (called a test target in this paper) at a specific status of the model, it is necessary to drive the model to the status before checking the condition. A status maps to a node of the RRT. It is usually necessary to check various conditions at a specific status. For example, when the specific status represents an SL/SF model state from which multiple transitions are made, we must check multiple conditions to measure the transition coverage. We propose a unique distance calculation metric, based on the observation that the test targets are gathered around some specific status such as an SL/SF state, named key nodes in this paper. The proposed metric increases the probability that an RRT is extended from key nodes by imposing penalties to non-key nodes. A test case generation algorithm utilizing the proposed metric is proposed. Three models of Electrical Control Units (ECUs) embedded in a commercial vehicle are used for the performance evaluation. The performances are evaluated in terms of penalties and compared with those of the algorithm using a typical RRT algorithm.

Keywords : Test Case Generation, Simulink/Stateflow, Rapidly Exploring Random Tree, ECU

# 변형된 RRT 알고리즘 기반 Simulink/Stateflow 모델 테스트 케이스 생성

박 한 곤† · 정 기 현†† · 최 경 희†††

## 요 약

본 논문에서는 경로탐색 분야에서 많이 사용되는 RRT 알고리즘을 기반으로 한 테스트 케이스 생성 알고리즘을 제안한다. RRT 알고리즘 성능에 영향을 주는 가장 중요한 요소는 RRT 공간 내 노드 사이의 거리를 계산하는 거리 함수이다. Simulink/Stateflow (SL/SF) 모델의 테스트 케이스는 모델의 특정 상태에서 특정한 조건(본 논문에서 테스트 타겟이라 명명함)을 검사하기 위해 필요한 입력 시퀀스이기 때문에, 특정 조건을 검사하기 위해서는 먼저 모델을 특정 상태로 이끌어가는 것이 필요하다. 여기서 모델의 상태는 RRT의 노드로 표현된다. 일반적으로 어느 한 상태의 경우 다수의 조건을 검사할 필요가 있다. 예를 들어, 모델의 특정 상태가 다수의 전이가 발생 가능한 SL/SF model의 한 상태로 표현될 때, 전이 커버리지를 측정하기 위해서는 반드시 다수의 조건을 모두 검사해야 한다. 본 논문에서는 테스트 타겟들이 키 노드라 불리는 SL/SF 상태로 표현되는 특정 상태에서 다수 발견되는 점에 착안해서 만든 거리 계산 함수를 제안한다. 제안된 거리 함수는 키 노드가 아닌 노드에 페널티를 부과해서 RRT가 키 노드로부터 확장될 확률을 증가시킨다. 본 논문에서는 제안된 거리 함수를 이용한 테스트 케이스 생성 알고리즘을 제안한다. 성능 평가를 위해 상업용 자동차에 들어가는 3가지 전자제어장치 모델이 사용된다. 제안된 테스트 케이스 생성 알고리즘의 성능은 페널티 측면에서 평가되고 기존의 RRT 알고리즘을 사용한 테스트 케이스 생성 알고리즘의 성능과 비교한다.

키워드 : 테스트 케이스 생성, 시뮬링크/스테이트플로우, Rapidly Exploring Random Tree, 전자제어장치

## 1. 서 론

Mathworks사의 Simulink/Stateflow(SL/SF)는 널리 사용되는 모델링 도구로 이산 시스템과연속 시스템을 모두 효율적으로 표현할 수 있는 데이터플로우 그래프(Dataflow Graph) 방식을 사용한다. 하지만, SL/SF에는 몇 가지 결점이 존재한

※ 본 연구는 방위사업청(UD150042AD)의 지원으로 수행되었음.
† 준 회 원 : 아주대학교 전자공학과 석사과정
†† 정 회 원 : 아주대학교 전자공학과 교수
††† 정 회 원 : 아주대학교 컴퓨터공학과 교수
Manuscript Received : June 27, 2016
First Revision : August 23, 2016
Accepted : August 24, 2016
* Corresponding Author : Ki Hyun Chung(khchung@ajou.ac.kr)

다. Simulink는 시스템을 모델링 하기 위한 다양한 블록들을 제공하는데 그중 하나가 시스템을 상태(State)와 전이(Transition)로 모델링 하는 Stateflow(SL/SF)이다. SL/SF 모델에서 상태들을 연결하는 전이는 전이 조건과 전이 조건이 만족할 경우 발생하는 동작(Action)으로 표현된다. 하나의 상태에서 한 개 이상의 전이가 발생 가능한 경우, 전이는 우선순위를 갖는다. 여러 개의 전이 조건이 동시에 만족될 경우 우선순위가 가장 높은 순위의 전이가 발생한다. 모델 입력, 출력 그리고 내부 변수 값들은 전이 조건과 동작을 표현하기 위해 사용된다.

SL/SF의 단점 중 하나는 상태의 개수가 많아지면 모델이 매우 복잡해진다는 점이다. 모델의 복잡도가 증가할수록 모델을 이해하는 것이 매우 어려워지고 모델링 시 결함을 유입시킬 가능성이 증가한다.

복잡도가 증가함에 따라 발생하는 또 다른 단점은 SL/SF 모델로부터 테스트 케이스(Test Case)를 자동으로 생성할 때 발생한다. 모델의 특정한 상태에서 테스팅 요구 사항을 검증하기 위해 모델은 먼저 그 특정 상태에 도달해 있어야 한다. 초기 상태에서 특정 상태에 도달하기 위해서는(테스팅은 보통 초기 상태에서 시작된다.) 모델을 특정 상태로 이동시키기 위한 입력 값의 시퀀스를 찾아야 한다. 모델에 따라서 입력 값의 시퀀스를 찾는 것은 가능할 수도 있고 불가능할 수도 있다. 시퀀스를 찾는 것은 도달 가능성 문제와 유사하다. 상당한 노력에도 불구하고 일반적으로 도달 가능성의 문제를 완전히 해결하는 것은 불가능하다.

SL/SF 모델로부터 테스트 케이스를 생성하는 방법에 관한 선행연구가 많이 있었는데 이에 대한 자세한 내용은 다음 장에서 다룰 것이다. 그중 하나가 경로 탐색(Path Planning) 응용 분야에서 큰 성과를 얻은 Rapidly exploring Random Tree(RRT) 알고리즘을 이용한 방법이다. 경로 탐색에서 시스템의 상태는 노드(Node)로 표현되고 노드는 시스템의 현재 상태와 시스템을 표현하는데 사용되는 변수 값들로 정의된다. 경로 탐색의 목표는 시스템의 상태를 어느 한 노드에서 타겟 노드로 이동시키는 입력 값의 시퀀스를 결정하는 것이다.

RRT 알고리즘은 트리(Tree)에서 확장될 노드를 찾고 찾은 노드로부터 트리를 확장시켜 나가는 과정을 반복적으로 수행하면서 RRT를 만든다. 충분한 개수의 노드가 만들어진다고 가정하면 RRT 알고리즘은 RRT에 들어있는 노드 사이의 최대 거리가 brute-force random 알고리즘에 의한 최대 거리 보다 더 작을 경우 효율적이다. RRT에 들어있는 노드의 개수가 충분하다면 brute-force random 알고리즘을 사용할 때보다 타겟(RRT의 노드로 표현된)이 트리에 존재할 확률이 더 증가하고 트리의 크기가 증가함에 따라 RRT의 노드들이 테스트 타겟(Test Target)과 일치하는 확률도 또한 증가한다.

경로 탐색 문제는 모델로부터 테스트 케이스를 생성하는 문제와 유사하다. 우리가 RRT 노드에 모델의 상태를 입력시킬 수만 있다면 2가지 모두 초기 노드에서 타겟 노드에 도달하도록 하는 입력 값의 시퀀스를 찾아내기 위해 문제로 귀결된다. 이러한 유사점으로부터 본 논문에서는 테스트 케이스 생성 과정에서 발생하는 도달 가능성 문제를 해결하기 위해 RRT 알고리즘을 이용할 것을 제안한다. RRT 알고리즘은 시스템의 특성이나 동작에 독립적이라는 장점이 있다.

무작위 알고리즘에 속하는 RRT 알고리즘은 RRT를 확장시키기 위한 노드를 무작위로 선택한다. 트리가 확장되면 알고리즘은 무작위로 선택된 노드와 트리에 있는 노드 사이의 거리를 측정한다. 트리에 있는 노드 중 무작위로 선택된 노드와 가장 짧은 거리를 갖는 노드가 가장 근접한 노드로 선택되고 트리는 그 노드로부터 다시 확장한다. 가장 근접한 노드를 결정하는 지표는 노드 사이의 거리이다.

RRT 알고리즘에 기반을 둔 테스트 케이스 생성 알고리즘은 RRT 확장 과정 동안 시스템의 테스트 케이스를 만든다. 테스트 케이스는 테스트 타겟 생성기(Test Target Generator)에 의해 결정된 테스트 타겟을 검증하기 위해 사용되는 입력 값의 집합으로 정의된다. 테스트 타겟은 요구사항으로 특정 조건일 수도 있고 테스트 되어야 하는 시스템의 상태일 수도 있다. 예를 들어, 테스트 타겟은 2개의 상태 사이의 전이 조건일 수 있다. 테스트 타겟들은 테스트 케이스 생성 정책(예, 상태, 전이, MC/DC)에 따라 다르게 구성된다.

테스트 케이스 생성의 경우 보통 실용적인 SL/SF 모델에 대해 다수의 테스트 타겟들이 존재하지만, 경로 탐색의 경우 오직 1개의 타겟만이 존재한다는 점이 2가지의 차이점이다. 테스트 케이스 생성의 경우 다수의 타겟이 존재하기 때문에 다수의 타겟을 찾을 수 있도록 RRT 알고리즘은 수정되어야 한다. RRT 알고리즘은 더 이상의 테스트 타겟이 존재하지 않거나 미리 정의된 횟수만큼 RRT를 확장할 때까지 RRT를 계속해서 확장해나간다.

특정 상태에서 전이 조건을 확인하고 싶은 경우 전이 조건을 검증하기 전에 모델이 특정 상태에 도달해 있도록 이끌어가는 작업이 필요하다. 시스템의 상태는 RRT의 노드로 표현된다. 일반적으로 특별한 노드들의 경우 다수의 조건을 확인할 필요가 있다. 예들 들면, 특별한 상태(또는 노드)가 SL/SF 모델에서 다수의 전이가 발생 가능한 1개 상태 표현될 때, 전이 커버리지(Transition Coverage)를 측정하기 위해서는 다수의 조건을 모두 확인해야 한다. 우리는 이러한 특별한 노드를 키 노드(Key Node)라 부른다. RRT를 키 노드로부터 확장시키는 경우 키 노드가 아닌 노드로(Non-Key Node)부터 확장시킬 때보다 테스트 타겟을 찾을 확률이 더 높기 때문에 키 노드로부터 RRT를 확장하는 것이 더 효율적이다.

우리는 키 노드에 우선권을 주기 위해 키 노드가 아닌 노드에 페널티(Penalty)를 부과하는 변형된 RRT 알고리즘을 제안한다. 키 노드가 아닌 노드에 페널티를 부과함으로써 무작위로 선택된 노드와 키 노드가 아닌 노드 사이의 거리가 페널티에 의해 증가한다. 따라서 키 노드가 아닌 노드가 가장 근접한 노드로 선택되는 확률이 감소하기 때문에 트리

가 키 노드가 아닌 노드로부터 확장될 확률 또한 감소하게 된다. 결과적으로, 키 노드가 아닌 노드로부터 테스트 타겟 노드를 탐색하는 기회가 감소하고 대신 키 노드로부터 테스트 타겟 노드를 탐색하는 기회가 증가한다.

키 노드에서 찾은 테스트 타겟의 수가 증가함에 따라 키 노드가 될 확률은 감소할 것이고 테스트 타겟의 수가 줄어들면 테스트 타겟을 찾을 확률이 또한 감소한다. 제안된 알고리즘은 이러한 직관적인 사실을 포함한다.

본 논문의 요지는 다음과 같다.

1) SL/SF 모델에서 다수의 테스트 타겟들을 찾기 위해 변형된 RRT 알고리즘을 제안한다.
2) 제안된 알고리즘은 키 노드가 아닌 노드에는 페널티를 부과하고 키 노드에는 우선권을 부여함으로써 테스트 타겟을 찾을 확률을 증가시킨다.
3) 제안된 알고리즘은 상업 모델을 사용해서 평가한다.

2장에서는 본 논문의 배경이 되는 기존의 RRT 알고리즘에 대해 설명하고 이전에 제안됐던 테스트 케이스 생성 알고리즘들에 대해 다룬다. 3장에서는 기존의 RRT 알고리즘에 페널티 부과 개념을 적용한 변형된 RRT 알고리즘에 대해 설명하고 제안된 RRT 알고리즘을 이용한 새로운 테스트 케이스 생성 알고리즘에 대해 소개한다. 4장에서는 제안된 알고리즘의 성능을 차량용 ECU 모델을 사용해서 평가하고 그 결과를 기존의 알고리즘과 비교한다. 5장에서는 연구의 결론을 기술한다.

## 2. 관련 연구

### 2.1 RRT 알고리즘

변수 $V = \{v_1, v_2, \cdots v_p, v_{p+1}, ..., v_m\}$로 표현되는 어느 시스템에서 모델의 한 노드 $ND_x$($ND_x$는 어느 한순간 시스템의 상태)에서 타겟 노드 $ND_y$로 가는 경로를 찾길 원한다고 가정해보자. 시스템의 상태는 모델에 따라 다르게 정의된다. 예를 들어, 시스템의 상태는 변수 값의 집합과 SL/SF의 현재 활성화된 상태로 정의될 수 있다. RRT 알고리즘에서 노드는 RRT 노드라 부른다. RRT 알고리즘은 시스템의 노드 공간(모든 가능한 노드들이 들어있는 공간)에 있는 2개 노드 사이의 경로를 탐색한다. RRT 알고리즘은 시스템에 의해 주어진 제한 요소들을 고려해서 경로를 탐색하고 탐색한 경로에 맞는 변수 시퀀스를 구한다.

RRT 알고리즘은 여러 가지 버전이 존재한다. 기존의 RRT 알고리즘은 Algorithm 1에 나와 있다. 이 알고리즘은 시스템의 시작 노드 $ND_{in}$에서부터 RRT를 만든다. RRT는 RRT를 만드는데 걸리는 시간과 RRT의 노드 개수를 고려해서 미리 정해진 확장 횟수 K번 만큼 확장된 후 완성된다. 트리의 첫 번째 노드는 $ND_{in}$이다. 매 반복마다 *random_node()*에 의해 $ND_{rd}$가 선택되고 *nearest_node*($ND_{rd}$,*T*)를 실행해 트리에 있는 노드 중 $ND_{rd}$와 가장 근접한 거리에 존재하는 $ND_{nr}$(Nearest Node)이 선택된다. 이후 시스템의 상태를 $ND_{nr}$에서 $ND_{rd}$로 이동시키기 위한 입력 값 $I_{nr}$을 구하는 것이 필요하다. 내부 변수 값은 시스템을 입력 값 $I_{nr}$으로 미리 정해진 시뮬레이션 시간 $\Delta T$만큼 시뮬레이션해서 구한다. 새로운 노드 $ND_{nw}$는 $ND_{nr}$에 $V_{nr}$($I_{nr}$와 내부 변수 값들을 포함하는)을 적용시켜 생성한다. 따라서 $ND_{nw}$는 $ND_{rd}$이거나 또 다른 $ND_{rd}$에 가장 가까운 노드(Closest Node)일 것이다. 시스템의 상태를 $ND_{nr}$에서 $ND_{rd}$로 이동시키는 정확한 $I_{nr}$ 값을 구하지 못하는 경우 가장 가까운 노드에서 $ND_{rd}$로 이동하기 위한 변수 값들이 구해진다. 가장 가까운 노드는 몇 번의 시도를 거쳐 트리에 있는 노드들로부터 선택된다. RRT 알고리즘에서는 $ND_{rd}$를 무작위로 선택하기 때문에 모델을 $ND_{nr}$에서 $ND_{rd}$로 이동시키는 입력 값 $I_{nr}$을 찾는 것은 매우 어렵고 이에 따라 $ND_{nw}$가 $ND_{rd}$이 되도록 만드는 것은 매우 힘들다. $ND_{nw}$, $ND_{nr}$과 $ND_{nw}$, 사이의 간선(Edge) 그리고 입력 값 $I_{nr}$은 트리에 각각 노드, 간선, 그리고 변수 값(모델을 $ND_{nr}$에서 $ND_{nw}$로 이동시키는)으로 추가된다. RRT의 확장 횟수가 $K$를 초과하거나 모든 타겟 노드를 찾은 경우 알고리즘은 종료된다.

```
RRT (ND_in,K)
T.init(ND_in)
for k=1 to K do
ND_rd ← random_node( );
ND_nr ← nearest_ node(ND_rd,T);
  I_nr ← select_input(ND_nr,ND_rd);
  V_nr ← Simulating(ND_nr,I_nr,ΔT);
ND_nw ← new_ node(V_nr);
T.add_vertex(ND_nw);
T.add_edge(ND_nr,ND_nw,I_r);
return T
```

Algorithm 1. A Typical RRT Algorithm

### 2.2 SL/SF 테스트 케이스 생성 알고리즘

SL/SF 모델 기반 테스트 케이스 생성에 대한 기존 연구들을 살펴보면 [7]에서는 Yices를 사용해서 테스트 커버리지를 높이고자 했다. [9]에서는 SL/SF 모델을 언어에 독립적이고 실행 가능한 모델로 변환한 후 Symbolic PathFinder[10]을 사용해서 테스트 케이스를 생성했다. [11]에서는 SL/SF 모델을 SMV 프로그램[12]으로 변환한 후 CTL[3]을 사용해서 테스트 케이스를 생성했다. 이 방법은 SL/SF 모델이 복잡할 경우 상당한 노력이 필요하다는 문제점이 존재한다. [14-17]에서는 SL/SF 모델과 유사한 UML 스테이트 다이어그램에서의 테스트 케이스 생성 방법에 대해 다룬다. 그러나 이들 모두 내부 변수에 대해 고려하고 있지 않다. [18]에서는 Satisfiability Modulo Theories (SMT) 모델로부터 테스트 케이스를 추출하는

알고리즘을 제안했다. 그러나 이 알고리즘도 비선형적인 기능이 포함된 범용 시스템에서 사용하기에는 적합하지 않았다.

SL/SF 모델에서 테스트 케이스를 생성하는 상업용 툴들이 있다. [19]에서는 무작위 기반 휴리스틱 기법을 사용한다. 하지만 휴리스틱 무작위 기법이 적절한 입력 시퀀스를 찾을 수 없을 경우 높은 커버리지를 얻기 위한 테스트 케이스를 생성하는데 매우 긴 시간이 소요될 수 있다. 이러한 단점을 개선하기 위해서는 더 많은 정보가 필요하다. T-VEC Tester[20]와 Design Verifier[21]는 제약 해결사와 무작위 기반 기법의 단점을 개선하기 위해 사용하는 엔진인 SAT Solver[22]를 이용한다. 그러나 이것들은 재귀 함수와 같은 사용자 정의 기능이 포함되어 있을 경우 단점을 갖는다.

RRT 알고리즘을 사용하는 많은 테스트 케이스 생성 알고리즘들은 자동차, 항공기 그리고 가전제품과 같이 복잡한 시스템을 모델링 할 때 사용하는 하이브리드 오토마타[23]를 기반으로 개발되어왔다. 몇 가지 선행 연구들은 SL/SF 모델을 하이브리드 오토마타로 변환한 후 하이브리드 오토마타에서 사용되는 기법[24]을 사용해서 테스트했다.

[25]에서는 공간 기반, 시간 기반 그리고 명세 기반 거리 함수를 제안했다. 이 함수를 실제로 사용하기 위해서는 한 이산 상태에서 다른 이산 상태로 시스템을 이동시키기 위한 조건들과 소요되는 시간 그리고 명세 달성률이 요구되었는데 이는 모델에 따라서 변하는 요인이므로 다양한 모델에 적용하기엔 부적합하다. [26, 27]에서는 노드의 개수가 적은 곳으로 RRT를 확장하는 기법을 제안했는데 두 RRT 노드 간의 거리를 이산 상태 거리와 연속 변수 값의 유클리드 거리의 조합으로 정의했다.

## 3. 변형된 RRT 알고리즘을 이용한 테스트 케이스 생성

3장에서는 변형된 RRT 알고리즘을 사용해서 SL/SF 모델의 테스트 케이스 생성 알고리즘에 대해 다룬다.

SL/SF 모델 $M$이 n개의 상태를 갖는 상태 집합 $S=\{S_1, S_2, \cdots, S_n\}$와 m개의 변수를 갖는 변수의 집합 $V=\{v_1, v_2, \cdots v_p, v_{p+1}, ..., v_m\}$ ($v_1 \sim v_p$는 모델의 입력, $v_{p+1} \sim v_m$는 내부 변수와 모델의 출력)를 갖는다고 하자. 상태 $S_i$의 활성화 변수가 $a_i$로 표현될 때 활성화 변수의 집합 $A$는 $A=\{a_1, a_2,\cdots, a_n\}$로 정의된다. 상태 $S_i$가 활성화되면 $a_i$는 1, 그렇지 않으면 0의 값으로 정의된다. 내부 변수들과 활성화 변수들은 SL/SF 모델 $M$을 모델 입력 값으로 시뮬레이션 한 후 결정된다. 일반성을 배제하지 않으면서 우리는 내부 변수와 모델 입력 값들이 적절하게 정렬되었다고 가정할 수 있다.

1개의 RRT 노드 $ND_x=\{V_x, A_x\}$는 $M$이 변수 집합 $V_x$와 활성화 변수 집합 $A_x$을 갖고 있는 상태를 나타낸다. $ND_x$와 $ND_y$을 연결하는 간선은 $e_{x,y}$로 정의된다. $ND_x$에서 $ND_y$로의 전이를 발생시키는 변수 집합은 $V_{x,y}$로 정의되고 시스템 입력 집합은 $I_{x,y}$로 정의된다. $ND_x$에서 $ND_y$에 도달하기까지의 경로는 $(e_{x,1},e_{1,2},\cdots,e_{i-1,i},e_{i,i+1},\cdots,e_{z-1,z})$로 정의되는데 이 때 $e_{i-1,i}$와 $e_{i,i+1}$는 $ND_i$를 통해 연결된다. 입력 시퀀스 $IS_{x,z}$는 $(I_{x,1},I_{1,2},\cdots,I_{i-1,i},I_{i,i+1},\cdots,I_{z-1,z})$로 정의된다.

2개의 SL/SF 상태 사이의 전이 조건은 “x<0”와 같은 단일 조건이나 “((x<0 and y<0) or (p≥q))”와 같은 복합 조건으로 구성된다. *and*, *or*, 그리고 *xor*와 같은 논리 연산자들은 여러 개의 단일 조건을 연결해준다. 각 단일 조건들은 “True” 나 “False” 값을 갖는다.

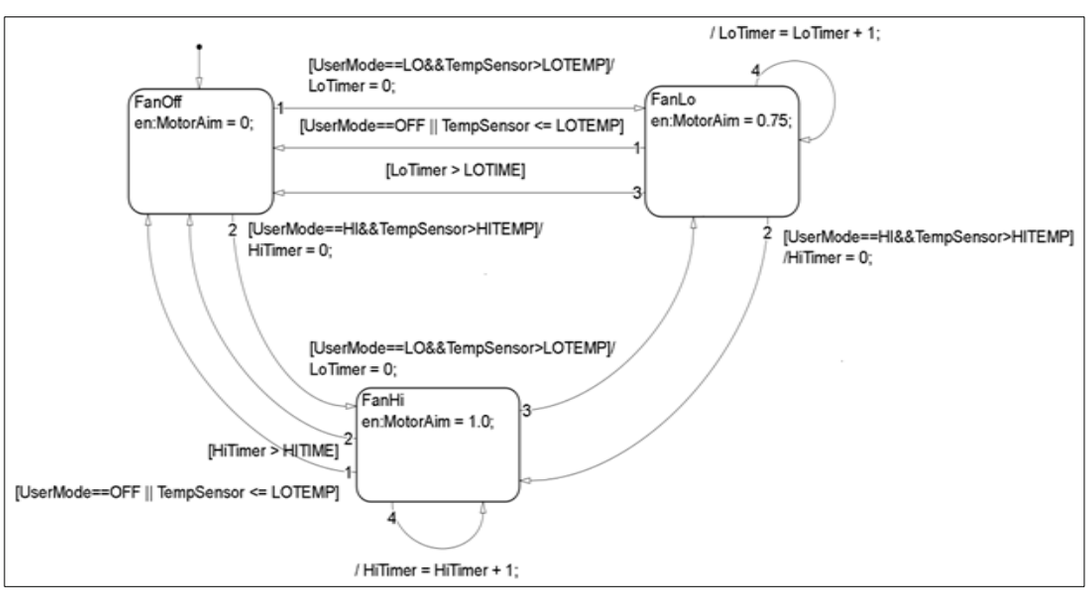



Fig. 1. An Example of an SL/SF Model

예를 들어, Fig. 1에는 “FanLo”에서 “FanOff”로 가는 2개의 전이가 존재한다. 더 높은 우선순위의 전이(Fig 1에서 우선순위 “1”) 조건은 2개의 단일 조건으로 구성된 복합 조건(“UserMode == OFF” and “TempSensor <= LOTEMP”)이다. “UserMode”가 “OFF”이고 “TempSensor”가 “LOTEMP”보다 더 크면 복합 조건의 값은 (True, False)가 된다.

테스트 케이스는 테스트 타겟을 만족시키기 위한 입력 시퀀스로 정의된다. 여기서 테스트 타겟이란 우리가 확인하고 싶은 것을 말한다. 예를 들어, 2개의 상태 사이의 전이가 발생했는지 확인하고 싶은 경우 테스트 타겟은 (전이, 전이 조건의 값)이 된다. Fig. 1을 예로 들어 설명하면, “FanOff”에서 “FanLo”로 전이가 발생했는지 확인하고 싶은 경우 테스트 타겟은 {“UserMode==LO&&Temp Sensor>LOTEMP,” (True, True)}가 된다.

각 테스트 타겟은 $M$의 초기 상태에서부터 테스트 되어야 하기 때문에 테스트 케이스는 $M$을 초기 상태에서 타겟 상태로 이동시키기 위한 입력 시퀀스가 된다. 타겟 상태에 도달하면 테스트 타겟을 만족시키는 입력 시퀀스에 의해 전이가 발생하게 된다. 테스트 케이스는 일반적으로 유일하지 않다. 테스트 케이스 생성의 예는 차후에 다룰 것이다.

### 3.1 테스트 케이스 생성을 위한 RRT 알고리즘 수정

제안된 알고리즘은 상태, 전이 또는 Modified Condition/Decision Coverage (MC/DC) 커버리지와 같은 테스트 커버리지 목표에 기반을 둬서 SL/SF 모델로부터 추출된 테스트 타겟을 만족하기 위한 테스트 케이스를 생성한다. 테스트 타겟 생성은 이 논문의 범위를 벗어나기 때문에 다루지 않는다. RRT는 타겟 목록(TG)에 있는 모든 타겟들이 만족되거나 RRT 확장 횟수가 미리 정해진 반복 횟수 K를 초과할 때까지 확장된다. Algorithm 2는 제안된 테스트 케이스 생성 알고리즘의 수도 코드(Pseudo Code)로 키 노드가 아닌 노드에 페널티를 부과하는 점을 제외하고는 기존의 RRT 알고리즘과 유사하다.

SL/SF 모델 $M$의 초기 상태는 RRT의 시작 노드 $ND_{in}$이 되고 기존의 RRT 알고리즘처럼 RRT를 확장시킨다. 변형된 알고리즘은 새로운 노드 $ND_{nw}$가 트리에 추가될 때마다 $M$의 동작이 타겟 목록 TG에 있는 타겟들 중에서 어느 테스트 타겟들을 만족하는지 검사한다. 만약 $M$의 동작이 TG에 있는 어느 타겟과 일치한다면, 그 타겟은 $ND_{nw}$를 추가해 RRT를 확장시키면서 만족된다. 트리가 확장되고 $ND_{nw}$에서 가장 근접한 노드 $ND_{nr}$가 키 노드가 된다. 이때, $ND_{nr}$의 페널티 값 $ND_{nr}.p$은 증가하지 않는다. 만약 RRT에 $ND_{nw}$를 추가해서 어떤 테스트 타겟도 만족되지 않으면 $ND_{nr}$은 키 노드가 될 수 없기 때문에 $ND_{nr}.p$는 페널티 계수 $a$만큼 증가한다.

예를 들어, Fig. 1에서 모델이 “*FanHi*”에서 “*FanLo*”로 이동할 수 있는지 테스트하기 위해 먼저 모델은 “*FanHi*” 상태에 도달해 있어야 한다. “*FanHi*” 상태에는 전이 커버리지를 위한 3개의 테스트 타겟이 존재한다. 만약 RRT에 새로운 노드를 추가시켜 $M$의 상태가 “*FanHi*”에서 “*FanLo*”로 전이되면, “*FanHi*”에서 3개의 테스트 타겟들 중 하나가 만족된다. 따라서 “*FanHi*”에서의 노드는 키 노드가 돼서 페널티 값은 증가하지 않고 그대로 1의 값을 갖는다. 반대로, 노드가 키 노드가 될 수 없는 경우 “*FanHi*”에서의 노드는 $a$만큼 페널티 값을 부과 받는다.

$I_{nr}$은 모델을 $ND_{nr}$ 에서 $ND_{rd}$나 $ND_{rd}$에 가장 가까운 노드로 전이시키기 위한 입력 값의 집합이다. 일단 $I_{nr}$을 구하면, $V_{nr}$은 SL/SF 모델을 $I_{nr}$로 시뮬레이션 시켜서 구한다. $V_{nr}$은 $I_{nr}$와 $I_{nr}$에 의해 결정되는 내부 변수 집합으로 구성된다. 새로운 노드 $ND_{nw}$는 $ND_{nr}$에 $V_{nr}$을 적용시켜서 구한다. RRT에 $ND_{nw}$를 추가해서 TG에 있는 어느 타겟을 만족하면 $I_{nr}$는 테스트 케이스에 추가된다. 이 알고리즘은 TG에 있는 모든 타겟들이 만족되거나 RRT 확장 횟수가 K를 초과할 때까지 수행된다.

```
TestCaseGeneration (NDin, K, TG)
T= Φ
c=0
T.init(NDin)
while (TG != Φ && c <= K)
NDrd ← random_node( );
NDnr ← nearest_ node(NDrd, T);
Inr ← select_input(NDnr, NDrd);
Vnr ← Simulating(NDnr, Inr, ΔT);
NDnw ← new_ node(Vnr);
T.add_vertex(NDnw);
c=c+1;
TestTarget = FindNewTestTarget (TG, NDnw);
if  TestTarget is not null
    testcases.add (ISNDin, NDnw);
    NDnr.p= 1;
    remove TestTarget  from TG;
else
    NDnr.p= a * NDnr.p;
end while
```

Algorithm 2. The Proposed Test Case Generation Algorithm

### 3.2 노드 사이의 거리 측정

Algorithm 1과 2에서 보여준 알고리즘들은 가장 근접한 노드를 찾기 위해 *nearest_ node*( ) 단계에서 RRT 노드와 $ND_{nd}$ 사이의 거리를 측정해야 한다. [25-28]에 이 지표에 대한 많은 제안들이 존재한다. 그중 본 논문에서는 *Gower's General Similarity Coefficient* [29]를 이용하고 수정한다. 2개 노드 사이의 *Similarity Coefficient* 값이 증가할수록 노드 사이의 거리는 더 가까워진다. 가장 큰 *Similarity Coefficient*를 갖는 노드가 가장 가까운 노드로 선택된다. 거리는 *Similarity Coefficient*에 반비례하므로 2개 노드 사이의

거리는 "1 - *Similarity coefficient*."로 표현될 수 있다. 거리 값이 점점 작아짐에 따라 노드 사이의 거리는 가까워진다. 2개 노드 $ND_x$와 $ND_y$ 사이의 거리 $Dist_{x,y}$는 아래와 같이 정의될 수 있다.

$$Dist_{x,y} = D_{x,y} + ND_y.p \times Cov_y$$

$$D_{x,y} = \frac{1}{m+n}\left(\sum_{k=1}^{n} da_{x,y,k} + \sum_{h=1}^{m} dv_{x,y,h} \mid Range_h\right)$$

for $dv_{x,y,h}$

when $v_h$ *is Boolean* or *Enumeration*

$$\left\{\begin{array}{l} dv_{x,y,h} = \begin{cases} 0 \text{ if } v_h(ND_x) = v_h(ND_y) \\ 1 \text{ if } v_h(ND_x) \neq v_h(ND_y) \end{cases} \\ Range_h = 1 \end{array}\right\}$$

when $v_h$ *is not Boolean*

$$\begin{cases} dv_{x,y,h} = |v_h(ND_x) - v_y(ND_y)| \\ Range_h = |\max(v_h) - \min(v_h)| \end{cases}$$

for $da_{x,y,k}$

$$da_{x,y,k} = \begin{cases} 0 \text{ if } a_k(ND_x) = a_k(ND_y) \\ 1 \text{ if } a_k(ND_x) \neq a_k(ND_x) \end{cases}$$

여기서 m은 SL/SF 모델 $M$을 모델링 하는데 사용된 변수의 개수이고 n은 모델의 상태 활성 변수의 개수이다. $d_{x,y,h}$은 $ND_x$와 $ND_y$의 h번째($1 \le h \le m+n$) 변수 사이의 거리로 $|ND_x(v_h) - ND_y(v_h)|$로 표현되고 여기서 $v_h(ND_x)$는 $ND_x$에서의 $v_h$ 값을 의미한다. $Range_h$는 $|max(v_h) - min(v_h)|$에 속하는 $v_h$의 범위이고 $Dist_{x,y}$를 정규화(Normalize)하는데 사용된다. Boolean 변수나 열거형 변수의 경우 $ND_x$가 $ND_y$와 같으면 $d_{x,y,h}$는 0의 값을 갖고 다르면 $d_{x,y,h}$는 1의 값을 갖는다. 이 경우에 $Range_h$는 1이다. $a_k(ND_x)$는 $ND_x$에서 $a_k$의 값을 의미한다. $a_k(ND_x)$는 0 또는 1의 값을 갖는다. $ND_y.p$는 $ND_y$의 페널티 값을 의미한다.

$ND_x$와 $ND_y$ 사이의 거리 $Dist_{x,y}$는 $D_{x,y}$와 페널티 값과 $ND_y$의 타겟의 수를 고려한 페널티 거리 양의 합이다. $Cov_y$는 총 테스트 타겟 수 중에 만족된 테스트 타겟 수의 비율을 의미한다. 만족된 테스트 타겟의 수가 증가함에 따라 $ND_y$가 키 노드로 남아있을 확률은 $Cov_y$ 만큼 감소한다. 확률 감소는 유사한 양만큼 거리 증가로 알고리즘에 반영된다. 거리 페널티는 페널티 값과 $ND_y$의 커버리지 값의 곱으로 표현된다. 만약 어느 노드에서 테스트 타겟이 존재하지 않으면, 커버리지는 $Cov_y$의 최댓값인 1을 갖는다.

### 3.3 제안된 알고리즘을 이용한 테스트 케이스 생성 예

제안된 알고리즘을 사용해서 테스트 케이스를 생성하는 과정은 Fig. 1의 모델을 사용해서 상세하게 설명된다. Fig. 1의 모델은 fan motor를 제어하는 간단한 SL/SF 모델로 2개의 입력 "*TempSensor*"와 "*UserMode*"를 갖는다. "*TempSensor*"은 Fan 제어기의 온도 값을 의미하고 "*UserMode*"는 사용자에 의해 설정된 제어 모드를 의미한다. "*UserMode*"는 3가지 모드 값 "*OFF*", "*LO*" 그리고 "*HI*"를 가질 수 있고 내부 변수에 의해 결정되는 출력 "*MotorAim*"은 적절한 모드에서 모터를 활성화시킨다.

모델에는 3가지 스테이트 "*FanOff*", "*FanLo*" 그리고 "*FanHi*"가 존재한다. 초기 스테이트는 "*FanOff*"이다. 사용자가 "*Usermode*" 입력으로 "*LO*"를 선택하고 "*TempSensor*"의 값이 "*LOTEMP*"(미리 정해진 상수값) 보다 크면 모델은 "*FanOff*"에서 "*FanLo*"로 이동한다. "*FanLo*" 스테이트에 있는 동안 출력 "*MotorAim*"은 최댓값의 75%를 의미하는 0.75 값을 유지한다. "*FanLo*"에서 머물러있는 시간이 "*LOTIME*"(미리 정해진 상수값)보다 크거나 사용자가 "*Usermode*" 입력으로 "*OFF*"를 선택하고 "*TempSensor*"의 값이 "*LOTEMP*"보다 작거나 같으면 모델은 "*FanOff*"로 이동한다. "*FanLo*"에 머물러있는 시간은 내부 변수 "*LoTimer*"에 기록된다.

사용자가 "*Usermode*" 입력으로 "*HI*"를 선택하고 "*TempSensor*"의 값이 "*HITEMP*"보다 크면, 모델은 "*FanHi*"로 이동하고 출력 "*MotorAim*"은 최댓값 100%을 의미하는 1값을 유지한다. "*FanHi*"와 "*FanLo*"사이의 모드 전이도 유사하게 발생한다. 타이머에 관한 2가지 내부 변수 "*LoTimer*"와 "*HiTimer*"가 있고 미리 정의된 4개의 상수 "*LOTEMP*", "*HITEMP*", "*LOTIME*" 그리고 "*HITIME*"가 있다. 이 예제에서 상수들은 각각 20, 30, 80 그리고 120의 값을 갖는다.

테스트 타겟들은 처음에 테스트 목표에 따라 달라져야 한다. Table 1에는 "*FanOff*"와 "*FanLo*" 사이의 전이 조건들을 검사하기 위해 만들어진 테스트 타겟 집합이 나와 있다. 전이 커버리지에 대한 테스트 타겟은 전이와 전이 조건의 값의 쌍으로 정의된다. "$trans_{i,j}$"는 상태 i에서의 j번째 우선순위를 갖는 전이를 의미한다. 예를 들어, "$trans_{FanOff,1}$"는 "*FanOff*"에서의 전이 중 첫 번째 우선순위 전이인 "[*UserMode > Lo && TempSensor > LOTEMP]/LoTimer=0*"를 의미한다.

Table 1. A Possible Test Target Set, TG

| Test Target Group | Test targets |
|---|---|
| $TG_{FanOff}$ | {$trans_{FanOff,1}$, *True*} |
| $TG_{FanLo}$ | {$trans_{FanLo,1}$, (*False, True*)}, {$trans_{FanLo,3}$, *True*} |

모델이 활성화되면 M은 "*FanOff*"로 이동하므로 $a_{\text{"FanOff"}}$의 값은 1이고 $a_{\text{"FanOff"}}$를 제외한 모든 활성 변수들은 0의 값을 갖는다. "*UserMode*"를 제외한 모든 입력, 내부, 외부 변수들은 초기 값으로 0의 값을 갖는다. "*UserMode*"의 초기 값은 OFF이다. 모든 노드들의 페널티 값도 초기에 1로 설정된다. RRT는 모델의 초기 스테이트 "*FanOff*"의 상태를 나타내는 초기 노드 $ND_{in}$에서부터 시작되고 $ND_{in}$의 모든 다

른 변수들은 초기 값으로 0의 값을 갖는다. *random_node( )* 함수에 의해 노드 $ND_{rd}$가 생성되면 *nearest_ node($ND_{rd}$,T)* 단계에서 트리에 있는 노드 중 $ND_{rd}$에서 가장 근접한 거리에 있는 노드가 $ND_{nr}$로 선택된다. $ND_{in}$은 트리에 있는 유일한 노드이므로 $ND_{in}$이 $ND_{nr}$이 된다. 새로운 노드 $ND_{new}$를 확장시키기 위해 먼저 *select_input($ND_{nr}$,$ND_{rd}$)* 함수를 사용해서 $I_{nr}$을 구한다. 여기서 $I_{nr}$은 모델의 상태를 $ND_{nr}$에서 $ND_{rd}$ 또는 $ND_{rd}$에 가장 가까운 노드로 이동시키기 위한 입력 값의 집합을 말한다. 구한 $I_{nr}$이 {"*TempSensor*" = 35, "*UserMode*" = LO}라 할 때, 모델 $M$을 $I_{nr}$로 시뮬레이션해서 내부 변수 "*LoTimer*"와 "*HiTimer*", 출력 변수 "*MotorAim*" 그리고 활성 변수들을 구하고 $V_{nr}$($I_{nr}$, 내부변수, 출력변수, 활성변수로 구성)을 만든다. 그리고 현재 활성 상태에서 $V_{nr}$로 시스템을 구동시켜 새로운 노드 $ND_1$를 생성한다. $ND_1$의 변수 집합을 {("*TempSensor*" = 35, "*UserMode*" = LO, "*LoTimer*"= 0, "*HiTimer*"= 0, "*MotorAim*"= 0.75), ($a_{\text{"FanOff"}}$= 0, $a_{\text{"FanLo"}}$= 1, $a_{\text{"FanHi"}}$= 0)}라 하자. 트리에 있는 노드 중 $ND_{rd}$에서 가장 근접한 노드를 발견하기 위해 노드 사이의 거리를 측정해야 한다. $ND_{rd}$와 $ND_{in}$사이의 거리를 $D_{rd,in}$라 할 때 $D_{rd,in}$ = (0+0+1+|20-0|/100 + 0/1 + |40-0/200| + |60-0|/200)/8=0.21의 값을 갖는다. "*TempSensor*", "*LoTimer*" 그리고 "*HiTimer*"의 범위를 각각 100, 200, 200라 하면 변수의 개수 m과 스테이트의 개수 n은 각각 5와 3의 값을 갖는다. "*MoterAim*"의 범위는 1이다. 결론적으로, $Dist_{rd,in}$ = 0.21 + 1 × 1 = 1.21이 된다. $ND_{rd}$과 $ND_1$에 대해서도 같은 과정을 반복하면 $D_{rd,1}$ = (1+1+1+|20-35|/100 + 1/1 + |40-0|/200 + |60-0|/200)/8=0.58, $Dist_{rd,1}$ = 0.58 + 1 × 0 = 0.58이 된다. $Dist_{rd,1}$이 $Dist_{rd,in}$보다 더 작기 때문에 $ND_1$이 가장 근접한 노드로 선택된다. $ND_1$에서 $ND_{rd}$에 도달하기 위한 입력 값의 집합 $I_{nr}$을 {"*TempSensor*" = 20, "*UserMode*" =OFF}라 하자. 모델을 $I_{nr}$로 시뮬레이션해서 내부 변수, 출력 변수 그리고 활성 변수들을 구한 후 $ND_2$를 생성한다. 시뮬레이션 된 모델은 전이에 의해 "*FanOff*" 상태에 도달해 있게 되고 테스트 타겟 {$trans_{FanLO,1}$, (*False, True*)}은 만족된다. 테스트 타겟이 만족되었기 때문에 $ND_1$의 페널티 값은 증가하지 않는다. 만약 어떠한 테스트 타겟도 만족시키지 못하면 $ND_1$의 페널티 값은 증가할 것이다. Table 2에는 2개의 타겟에 대한 입력 시퀀스가 나와 있다. 모든 테스트 타겟이 만족되거나 반복 횟수 K를 초과할 때까지 위 과정은 반복된다.

Table 2. Test Case Samples

| Path | Test case | Test targets |
|---|---|---|
| NDinit→ND1 | {TempSensor=35, UserMode=LO} | {transFanOff,1,True} |
| NDinit→ND1 →ND2 | {TempSensor=35, UserMode=LO} {TempSensor=20, UserMode=OFF} | {transFanLo,1,(True, False)} |

## 4. 실 험

제안된 알고리즘의 성능은 기존의 RRT 알고리즘을 사용한 [5]의 알고리즘과 비교해서 상업용 자동차의 3개의 전자제어장치(ECU) 모델을 대상으로 상태, 전이 커버리지 측면에서 평가했다. 페널티 계수 $a$의 범위는 1~10으로 설정했다. 제안된 알고리즘에서 요구되는 계산 시간과 추가적인 메모리 크기는 크지 않기 때문에 비교하지 않았다. RRT 최대 확장 횟수 K는 테스트 케이스를 생성하기에 충분히 큰 값 10,000으로 설정했고 평가를 위해 인텔 코어 i5 2.67 GHz와 4 GB RAM이 장착된 PC를 사용했다.

평가에 사용된 ECU 모델은 driver door controller, air suspension controller 그리고 body/chassis로 각각의 복잡도는 하, 중, 상이다. 모델에 대한 통계 자료는 Table 3에 나와 있다.

Table 3. Model Statistics of the Three Evaluated ECUs

| ECU Model | No. of state | No. of transitions | No. of variables |
|---|---|---|---|
| Driver door controller | 20 | 55 | 59 |
| Air suspension controller | 83 | 272 | 150 |
| Body /chassis | 230 | 477 | 282 |

Fig. 2 ~ Fig. 4에 시뮬레이션 결과가 나와 있다. 제안된 알고리즘은 α가 1보다 큰 모든 경우에 대해 기존의 RRT 알고리즘을 사용했을 때 보다 상태 커버리지와 전이 커버리지 모두에서 더 높은 커버리지 비율을 보여줬고 이는 키 노드가 아닌 노드에 페널티를 부과함으로써 키 노드에 우선권을 부여하는 전략이 효과적임을 의미한다.

제안된 방법의 커버리지 비율은 α가 10에 가까워질 때까지 높아졌고 이후 약간 감소했다(데이터는 Fig에 나와 있지 않다.) 이러한 사실은 키 노드가 아닌 노드에 극도의 페널티를 부과하는 것이 커버리지 비율에 부정적인 영향을 끼침을 의미한다.

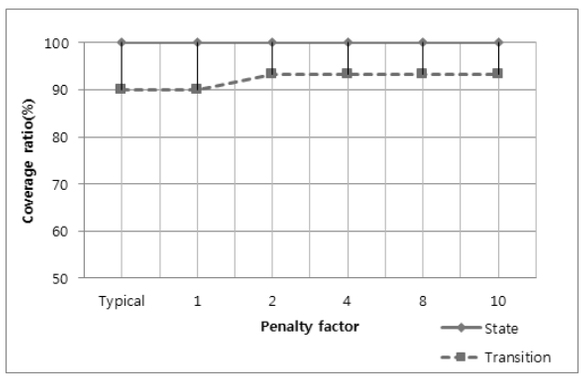



Fig. 2. Coverage Comparison for the Driver Door Controller

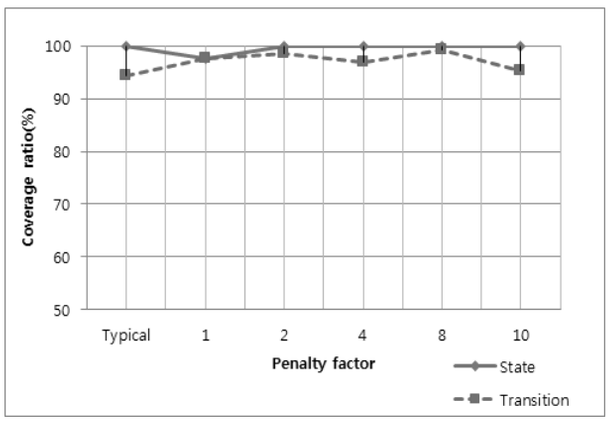



Fig. 3. Coverage Comparison for the Air Suspension Controller

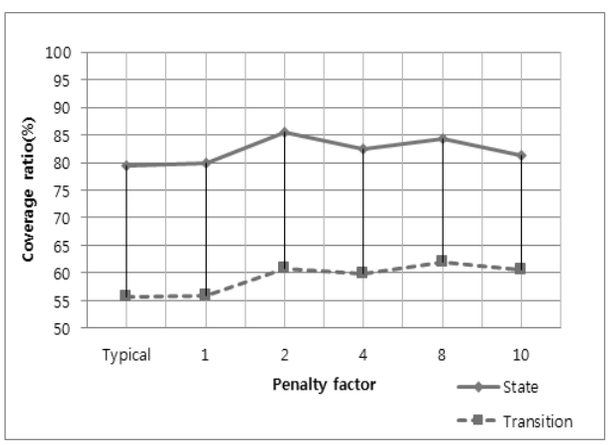



Fig. 4. Coverage Comparison for the Body/Chassis Controller

복잡도가 높은 body/chassis 모델의 경우 상태 커버리지 비율이 상대적으로 낮았다. 커버리지 비율이 100%가 될 수 없는 많은 이유들이 존재하는데 그중 하나로 모델이 도달 불가능 경우를 포함하고 있을 수 있다. 예들 들어, 모델에 고립된 상태가 있는 경우 그 상태를 만족하기 위한 테스트 케이스를 찾는 것은 불가능하다. 또 다른 이유로 차량에 들어가는 모델들은 전 세계 국가, 지역에서 요구되는 모든 기능들을 포함한다는 점을 들 수 있다. 일반적으로 같은 ECU라 할지라도 국가나 지역에 따라 다른 기능이 필요하다. 이 모든 기능들을 테스트하기 위해서 SL/SF 모델은 다른 상태와 전이들로 구성되어야 한다. 국가과 지역에 따라 파라미터 값을 다르게 설정하면 모델은 다르게 동작한다. 다른 파라미터를 갖는 기능들은 절대로 같이 동작되면 안 되기 때문에 모델은 관련이 없는 상태과 전이를 포함할 수밖에 없고 따라서 1번의 시뮬레이션으로 모든 기능들을 완전히 커버할 수 없다. 이러한 관련이 없거나 실행 불가능한 기능들은 커버리지 비율을 낮춘다.

커버리지 비율을 낮추는 다른 이유로 타이머를 포함하는 기능들에 대한 테스트 케이스를 찾아내는 것이 어렵다는 점을 들 수 있다. 예를 들어, Fig. 5에서 입력 "*Button*"이 1이고 "*LampOff*"에서 활성화된 타이머의 값이 3000ms보다 크면, 모델은 "*LampOff*"에서 "*LampOn*"로 전이한다. 이 테스트 타겟을 만족시키기 위해서 RRT는 입력 "*Button*"이 1의 값을 갖는 노드를 적어도 30번 생성해야 한다(모델의 1tick이 100ms 일 경우). RRT 알고리즘이 트리를 확장하기 위해 무작위로 노드와 입력 값들을 선택한다는 점을 고려하면 입력 "Button"의 변경없이 타이머 값을 증가시키는 것은 매우 어렵다.

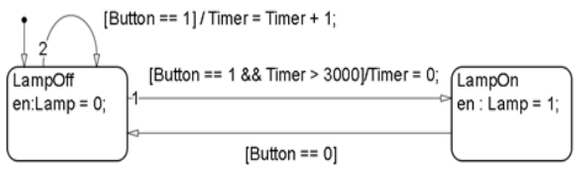



Fig. 5. An Example of an SL/SF Model with a Timer Logic

## 5. 결 론

본 논문은 RRT 기반의 Simulink/Stateflow 모델 테스트 케이스 자동 생성 알고리즘을 제안했다. 제안된 알고리즘은 고유 거리 함수를 사용해서 변형된 RRT 알고리즘을 이용한다. 어느 한 노드에서 한 개 이상의 테스트 타겟이 만족되면, 그 노드는 키 노드가 되어 노드의 페널티 값이 증가하지 않는다. 반대로 어떤 테스트 타겟도 만족되지 못하면, 노드의 페널티 값은 증가한다. 키 노드에 우선권을 부여함으로써, RRT가 키 노드로부터 확장될 확률이 증가하고 테스트 타겟이 만족될 확률도 증가한다. 제안된 알고리즘의 성능 평가를 위해 상업용 차량의 3가지 전자제어장치(ECU)들이 사용되었다. 제안된 방법은 테스트 케이스 생성 시간이나 요구되는 메모리 량의 큰 증가 없이 기존의 RRT 알고리즘을 사용한 방법보다 상태 커버리지와 전이 커버리지에서 모두 더 높은 커버리지 비율을 보여줬다. 페널티 계수를 상당히 큰 값으로 설정했을 때, 키 노드에 너무 큰 우선권이 부여되어 RRT가 원하지 않는 방향으로 확장됐기 때문에 성능이 오히려 나빠지는 결과를 보였다.

## References

[1] MATLAB Simulink Stateflow, The MathWorks, Inc. [Internet], http://www.mathworks.com/products/stateflow/ 1994-2013.

[2] R. Alur, "Model checking of hierarchical state machines," in *Proceedings of the 6th ACM SIGSOFT FSE*, Vol.23, Iss.6, pp.175-188, 1998.

[3] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's Decidable About Hybrid Automata?" *Journal of Computer and System Sciences*, Vol.57, Iss.1. pp.94-124, 1998.

[4] T. Brihaye, "A note on the undecidability of the reach-ability problem for o-minimal dynamical systems," in *Mathematical Logic Quarterly*, Vol.52, Iss.2 pp.165-170, 2006.

[5] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," TR 98-11 Computer Science Dept., Iowa State Univ. <http://janowiec.cs.iastate.edu/papers/rrt.ps>, 1998.

[6] J. M. Esposito, J. Kim, and V. Kumar, "A Probabilistic Approach to Automated Test Case Generation for Hybrid Systems," in Hybrid Systems: Computation and Control, 2004.

[7] M. Satpathy, A. Yeolekar, and S. Ramesh, "Randomized Directed Testing (REDIRECT) for Simulink/Stateflow Models," in *Proceedings of the 8th ACM International Conference on Embedded Software*, pp.217-226, 2008.

[8] The Yices SMT Solver [Internet], http://www.csl.sri.com.

[9] C. S. Pasareanu, "Model Based Analysis and Test Gen-eration for Flight Software," in *SMC-IT 2009, Third IEEE International Conference*, pp.83-90, 2009.

[10] C. S. Pasareanu, P. C. Mehlitz, D. H. Bushnell, K. G. Burlet, M. Lowry, S. Person, and M. Pape, "Combining unit level symbolic execution and system level concrete execution for testing," in *Proc. ISSTA'08 (to appear), NASA Softwrae*, 2008.

[11] H. S. Hong, I. S. Lee, O. Sokolsky, and S. D. Cha, "Automatic Test Generation From Statecharts Using Model Checking," in Technical Report, Department of Computer & Information Science University of Pennsylvania, MS-CIS-01-07, 2001

[12] K. L. McMillan, "Symbolic Model Checking - an Ap-proach to the State Explosion Problem," Kluwer Academic Publishers, 1993.

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

[14] S. Kansomkeat and W. Rivepiboon, "Automated-Generating Test Case Using UML Statechart Diagrams," *Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology*, pp.296-300, 2003.

[15] P. Samuel, R. Mall, and A. K. Bothra, "Automatic test case generation using unified model language (UML) state diagrams," *IET Software*, Vol.2, Iss.2, pp.79-93, 2008.

[16] C. Dougsa-ard, K. Dahal, A. Hossain, and T. Suwannasart, "GA-based Automatic Test Data Generation for UML State Diagrams with Parallel Paths," Advanced design and manufacture to gain a competitive edge, pp.147-156, 2008.

[17] S. K. Swain, D. P. Mohapatra, and R. Mall, "Test case Generation Based on State and Activity Models," *Journal of Object Technology*, Vol.9, Iss.5, pp.1-27, 2010.

[18] P. Roy and N. Shankar, "SimCheck: a contract type system for Simulink," *Journal of Innovations in Systems and Software Engineering,* Vol.7, Iss.2, pp.73-83, 2011.

[19] Reactis, Reactive Systems, Inc. [Internet], http://www.reactive-systems.com/products.msp.

[20] T-VEC tester, T-VEC Technologies, Inc. [Internet], http://www.t-vec.com/solutions/products.php.

[21] Design Verifier, Mathworks, Inc. [Internet], http://www.mathworks.com/products/sldesignverifier/.

[22] SAV Solving In General [Internet], http://www.satisfiability.org/.

[23] T. A. Henzinger, "The Theory of Hybrid Automata," in *Proceedings of Logic in Computer Science, Eleventh Annual IEEE Symposium*, pp.278-292, 1996.

[24] A. Agrawal, G. Simun, and G. Karsai, "Semantic Translation of Simulink/Stateflow models to Hybrid Automata using Graph Transformations," in *Proceedings of the Workshop on Graph Transformation and Visual Modelling Techniques*, Vol.109, pp.43-56, 2004.

[25] J. M. Esposito, "Randomized Test Case Generation for Hybrid Systems: metric selection," in *System Theory, Proceedings of the Thirty-Sixth Southeastern Symposium*, pp.236-240, 2004.

[26] T. Dang and T. Nahhal, "Coverage-guided test generation for continuous and hybrid systems," *Formal Methods in System Design*, Vol.34, No.2, pp.183-213, 2009.

[27] S. N. Ahmadyan, J. A. Kumar, and S. Vasudevan, "Goal-oriented stimulus generation for analog circuits," *Design Automation Conference*, pp.1018-1023, 2012.

[28] M. S. Branicky, M. M. Curtiss, and J. A. Levine, "RRTs for nonlinear, discrete, and hybrid planning and control," *Proceedings 42nd IEEE conference Decision and Control*, Vol.1, pp.657-663, 2003.

[29] J. C. Gower "A general coefficient of similarity and some of its properties," *Biometrics*, Vol.27, pp.857-871, 1971.

박 한 곤

e-mail : wkfksla@ajou.ac.kr

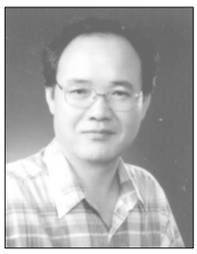
2009년 아주대학교 전자공학부(학사)

2015년~현 재 아주대학교 전자공학과 석사과정

관심분야 : 임베디드 시스템 테스팅, 실시간 시스템 등

정 기 현

e-mail : khchung@ajou.ac.kr
1984년 서강대학교 전자공학과(학사)
1988년 미국 Ilinois주립대 EECS(석사)
1990년 미국 Purdue대학 전기전자공학부 (박사)
1991년~1992년 현대반도체 연구소
1993년~현 재 아주대학교 전자공학과 교수
관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어/실시간 시스템



최 경 희

e-mail : khchoi@ajou.ac.kr
1976년 서울대학교 수학교육과(학사)
1979년 프랑스 그랑데꼴 Enseeiht대학 (석사)
1982년 프랑스 Paul Sabatier대학 정보공학부(박사)
1982년~현 재 아주대학교 컴퓨터공학과 교수
관심분야 : 운영체제, 분산시스템, 실시간/멀티미디어 시스템