http://dx.doi.org/10.3745/KTSDE.2013.2.11.765

Generating Test Cases of Stateflow Model Using Extended RRT Method Based on Test Goal

Hyeon Sang Park† · Kyung Hee Choi†† · Ki Hyun Chung†††

ABSTRACT

This paper proposes a test case generation method for Stateflow model using the extended RRT method. The RRT method which has been popularly used for planning paths for complex systems also shows a good performance for test case generation. However, it does not consider the test coverage which is important for test case generation. The proposed extension method hires the concept of test goal achievement to increase test coverage and drives RRT extension in the direction that increases the goal achievement. Considering the concept, a RRT distance metric, random node generation method and modified RRT extension algorithm are proposed. The effectiveness of proposed algorithm is compared with that of the typical RRT algorithm through the experiment using the practical automotive ECUs.

Keywords : Black-Box Testing, Model Based Testing, RRT, Stateflow, Test Case Generation

테스트 목표 기반의 향상된 RRT 확장 기법을 이용한 Stateflow 모델 테스트 케이스 생성

박 현 상† · 최 경 희†† · 정 기 현†††

요 약

본 논문은 Rapidly-exploring Random Tree(RRT) 확장 기법을 이용하여 Stateflow 모델 기반의 블랙박스 테스트 케이스 자동 생성 방법을 제안한다. 복잡한 시스템의 경로 계획 문제를 효율적으로 해결하는 방법으로 널리 사용되고 있는RRT기법은 테스트 케이스 생성에서도 좋은 성능을 보이고 있으나, 테스트 케이스 생성에 있어서 중요한 부분을 차지하는 테스트 커버리지를 고려하고 있지 않다. 제안하는 확장 기법은 테스트 커버리지를 향상시키기 위하여 테스트 목표 달성 율의 개념을 RRT 확장에 도입하여 테스트 목표를 더 달성할 수 있는 방향으로 RRT 확장을 유도 한다. 이를 위해서 테스트 목표 달성 율을 고려한 RRT 거리 함수와 RRT 무작위 노드 생성 방법, 그리고 변형된 RRT 확장 알고리즘을 제안한다. 제안된 기법의 유용성은 실제 자동차에서 사용되는 제어 ECU들의 Stateflow 모델을 이용한 실험을 통해 기존 RRT를 이용한 테스트 케이스 생성 기법과의 성능을 비교를 통해 보인다.

키워드 : 블랙박스 테스트, 모델 기반 테스트, RRT, Stateflow, 테스트 케이스 생성

1. 서 론

최근 내장 시스템 소프트웨어를 탑재한 전자 제품을 개발하는 많은 산업체에서는 제품의 명세서를 자연어 형식의 문서로 관리하는 것이 아닌 정형화 된 형태의 모델로 작성하고, 이를 바탕으로 제품의 시험을 진행하는 모델 기반 블랙박스 자동 테스트 기법을 많이 도입하고 있다. 제품 사양 모델링 기법 중 Mathworks사의 Stateflow[1]는 가전, 자동차, 항공, 선박 등 많은 산업 분야에서 실용적으로 쓰이고 있는 모델링 기법이다. 많은 산업체들이 생산하는 제품의 명세서를 Stateflow 모델과 같은 정형화된 형식으로 관리하고, 이를 바탕으로 제품 개발과 제품 검증까지의 과정을 자동화 시키는 기법을 연구하고 있다[2].

Stateflow는 시스템을 이산적인 상태로 보고 시스템의 변수로 이루어진 조건에 따라 상태의 변화를 전이로 모델링하는 상태 다이어그램의 한 종류이다. Stateflow는 복잡한 명세를 이해하기 편한 기법으로 모델링 할 수 있다는 이점이 있으나, 모델이 복잡해질수록, Statwflow모델을 이용하여 블랙박스 테스트를 위한 자동 테스트 케이스 생성을 하는 것은 매우 어렵다는 문제가 있다.

블랙박스 테스트 시 테스트 케이스를 생성하는 방법이 매우 어려운 것은 블랙 박스 테스트 환경에서는 시스템의 말

† 준 회 원 : 아주대학교 정보통신전문대학원 정보통신공학과 박사과정
†† 정 회 원 : 아주대학교 정보통신전문대학원 정보통신공학과 교수
††† 정 회 원 : 아주대학교 일반대학원 전자공학과 교수
논문접수 : 2013년 5월 13일
수 정 일 : 1차 2013년 8월 14일
심사완료 : 2013년 9월 5일
* Corresponding Author : Hyeon Sang Park(elvaimay@ajou.ac.kr)

단 입력의 값 만을 제어할 수 있으며, 시스템의 내부 변수나 말단 출력의 값을 직접 제어 할 수 없기 때문이다. 시스템이 특정 내부 변수나 말단 출력의 값을 갖게 하는 말단 입력의 값을 구하는 문제는 도달 가능성 문제[3]의 일종이다. Stateflow모델은 하이브리드 오토마타[4]의 일종이며[5], 하이브리드 오토마타의 도달 가능성 문제는 결정 불가능한 문제라고 알려져 있다[6, 7]. 이러한 어려운 문제를 배경으로 블랙박스 환경에서 Stateflow 모델 기반으로 테스트 케이스를 생성하기 위한 연구들이 많이 진행되어 왔다[8-10].

도달 가능성 문제를 풀기 위해서 많은 방법이 시도되었다. 대표적인 한 가지 방법은 제약 해결사(constraint solver)를 이용하는 방법[8, 11]으로, 이 방법은 필요한 변수의 값을 빠르게 구할 수 있다는 장점이 있으나, 단점 또한 존재한다. 첫째로, 풀어야 하는 시스템의 조건이 비선형적일 때는 변수의 값을 구하는데 들어가는 비용이 매우 크다는 것이다. 더불어, 풀어야 할 시스템의 조건에 동적 연결 라이브러리 함수와 같은 내부 동작 방식을 알 수 없는 요소가 포함되어 있을 경우에는, 변수의 값을 구하는 것은 불가능하다. 둘째로, 조건을 풀어서 조건을 만족하는 각 변수의 값을 구했다 하더라도, 변수가 내부 변수거나, 말단 출력일 경우, 블랙 박스 테스트 환경에서는 구한 값을 내부 변수나, 말단 출력에 적용을 하지 못하기 때문에, 해당 변수가 제약 해결사로부터 구해진 값을 갖도록 하기 위한 조건 식을 다시 구해야 하며, Stateflow 모델이 복잡할수록 조건 식을 구하기 어려워진다.

도달 가능성 문제를 풀기 위한 또 다른 방법으로, Stateflow 모델을 분석하기 쉬운 다른 모델로 변환을 하는 방법이 있다[9, 10]. 모델 변환 작업은 많은 비용이 들어가며, 원본 Stateflow 모델이 복잡해질수록 그 비용은 더 커진다. 또한 이 방법도, 제약 해결사를 이용한 방법과 마찬가지로, Stateflow 모델 내부에, 그 내용을 파악하기 어려운 요소가 포함되어 있을 경우, 변환된 모델로도 분석이 어려운 문제가 있다.

블랙박스 테스트를 위한 테스트 케이스 생성 기법에서는 도달 가능성 문제는 반드시 해결해야 하므로, 본 논문에서는 무작위 말단 입력 선택을 기반으로, 도달 가능성 문제 해결을 시도한다. 무작위 기법은, 모델의 특성에 관계 없이 범용적으로 적용할 수 있는 장점이 있으나, 그 성능의 편차가 크다는 단점이 있다. 무작위 기법의 성능을 보완 하기 위해서 본 논문에서는 로봇 공학의 동작 계획에서 사용되는 기법인 RRT(Rapidly-exploring Random Tree)[12]를 이용한다. 로봇의 시작 위치에서 목표 위치까지 장애물을 피해가며 경로를 찾아가는 과정이, 블랙박스 테스트에서 시스템의 시작 상태에서, 입력을 제어하여 제약 조건을 피하면서, 테스트 목표 상태까지 도달하게끔 만드는 것과 유사하다는 측면이 RRT를 테스트 케이스 생성에 이용 할 수 있는 근거가 된다[13]. 그러나 동작계획은 일반적으로 찾아야 하는 목표가 한 개인 반면 테스트 케이스 생성은 다수개의 목표를 찾아야 한다는 차이점이 있다. 이러한 배경으로 RRT를 이용하여 테스트 케이스를 생성하는 선행 연구들이 이루어졌다[13-17].

RRT는 무작위 기반으로 테스트 대상 시스템의 상태들을 빠르고 조밀하게 탐색해 나가지만, RRT를 이용한 관련 연구는 테스트 케이스 생성 성능의 기준이 되는 테스트 커버리지를 고려하지 않아 테스트 케이스 생성을 위한 효율적인 RRT 확장이 이루어지지 않는 단점이 있다. 테스트 커버리지를 높이기 위해서는 많은 테스트 목표를 달성하는 테스트 케이스들을 생성 할 수 있어야 한다. 본 논문에서는 RRT를 이용하여 Stateflow 모델 기반으로 테스트 케이스를 생성할 때의 테스트 커버리지를 높이기 위해 테스트 목표의 달성 율을 반영한 변형된 RRT 확장 방법을 제안한다. 제안된 RRT 확장 방법은 기존의 RRT 확장 방법과 세 가지의 다른 요소가 존재한다.

1. 테스트 목표 달성 율을 반영한 RRT 무작위 노드 생성 기법
2. 테스트 목표 달성 율을 반영한 RRT 노드 간의 거리함수
3. 테스트 목표 달성 율을 반영한 RRT 확장 기법

본 논문의 구성은 다음과 같다. 2장에서는 논문의 배경인 RRT와 RRT기반 테스트 케이스 생성에 대한 선행 연구, Stateflow 모델 기반 테스트 케이스 생성에 대한 선행 연구를 각각 기술하며, 3장에서는 제안하는 변형된 RRT 확장 기법, 4장에서는 예제 Stateflow모델을 이용하여 기존 RRT 기법과 제안된 기법의 성능 비교 실험을 통해 제안된 기법의 효용성을 보인다. 마지막으로 5장에서 연구의 결론을 기술한다.

2. 연구 배경 및 관련 연구

2.1 Rapidly Exploring Random Trees(RRT)

시스템 S가 내부 상태 변수 $V=\{v_1, v_2, v_3, \cdots, v_n\}$으로 이루어 질 때, 시스템의 상태는 $v_1, v_2, v_3, \cdots, v_n$값의 조합으로 나타낼 수 있다. RRT 알고리즘은 특정 시점에서의 시스템의 상태를 트리의 노드로 정의한다. 시스템의 시작 상태를 트리의 루트로 보고, 시스템이 도달해야 하는 목표 상태 노드까지의 경로를 무작위 기반으로 찾아가는 것이 RRT 확장의 목표이다.

RRT 확장 알고리즘은 많은 변형들이 존재하지만[18] Table 1의 알고리즘이 그 원형이다[12].

RRT 확장 알고리즘을 이용하려면 먼저 RRT 노드 간의 거리를 측정하는 거리 함수를 정의해야 한다. RRT 노드 간의 거리가 가까울수록, 두 RRT 노드는 서로 비슷한 내부 상태 변수의 값들을 가진다고 말할 수 있다.

RRT 확장은 시스템의 초기 상태 nd_{init}, 확장 횟수 K, 1회 확장 시 시스템의 시간 변화량 Δt를 입력 인자로 받는다. 확장이 K회 반복 되면서 각 확장 때 마다 다음과 같은 단계를 통해서 RRT의 확장이 이루어진다. 1)무작위로 내부 상태 변수의 값을 결정하여 RRT 무작위 노드 nd_{rand}를 생성한다. 2)RRT에 포함된 노드들 중 nd_{rand}에 가장 가까운 노드

Table 1. Typical RRT extension algorithm

```
Extends (nd_init, K, Δt)
T.init(nd_init)
for k=1 to K do
  nd_rand ← random_node();
  nd_near ← nearest_ node(nd_rand, T);
  v ← select_input(nd_rand, nd_near);
  nd_new ← new_ node(nd_near, v, Δt);
  T.add_vertex(nd_new);
  T.add_edge(nd_near, nd_new, v);
return T
```

nd_{near}를 찾는다. 3)시스템 S를 nd_{near}에서, nd_{rand}에 가장 가깝게 변화 시키기 위한 시스템의 입력 v를 결정한다. 4)입력 v를 Δt시간 동안 시스템 S에 적용하여 새롭게 생성된 노드 nd_{new}를 RRT에 추가한다.

RRT의 장점은 대상 시스템의 말단 입력만을 제어하여 트리를 확장하기 때문에, 시스템의 특성에 제약을 받지 않고 적용 가능하다는 점이다. 두 번째 장점으로는, 목표 상태가 도달 할 수 있는 시스템의 상태라면, RRT 확장의 횟수가 증가함에 따라, 도달 확률이 1에 가까워진다는 점이다[15].

2.2 RRT를 이용한 테스트 케이스 생성 기법

RRT를 이용한 테스트 케이스 생성기법의 선행 연구들은 하이브리드 오토마타 모델 기반으로 많이 이루어졌다. 각 선행 연구들은 테스트 케이스 생성에 RRT를 적용하기 위해서 여러 활용 기법을 보여주고 있다[14-17]. 이 연구들은 공통적으로 RRT 공간을 이산 변수들과 연속 변수들의 조합으로 구성하고 있다. 각 연구들의 거리 함수, 테스트 커버리지, RRT 무작위 노드 생성 방법, RRT 확장 방법을 본 연구와 비교한다.

1) RRT 거리 함수

각 연구에서 사용된 RRT 노드 거리 함수는 기본적으로 RRT 공간을 구성하는 변수들의 유클리드 거리(Euclidean distance)를 사용하고 있으나[16,17], 일부 연구에서 RRT 확장의 효율을 높이기 위해 다른 거리 함수들을 제안하고 있다. [15]에서는 하이브리드 오토마타에서 사용할 수 있는 공간 기반, 시간 기반, 명세 기반의 거리 함수를 제안하고 있다. 이 함수들을 실제로 사용하기 위해서는 한 이산 상태에서 다른 이산 상태로 이동하기 위한 조건들과 소요되는 시간, 명세 달성 율 등을 분석할 수 있어야 하는데, 이는 모델에 따라서 변하는 요인이므로 다양한 모델에 적용하기엔 부적합하다. [14]에서는 두 RRT 노드 간의 거리를 이산 상태의 거리와, 연속 변수 값의 유클리드 거리의 조합으로 정의하고 있으나, 이산 상태의 거리를 계산하기 위한 비용이 크다는 단점이 있다.

본 논문에서 제안하는 RRT 거리 함수는 RRT 노드의 이산 변수와 연속 변수의 유사 도를 계산함과 동시에 RRT 노드의 테스트 목표 달성 율을 추가로 반영함으로써, 테스트 목표를 더 많이 달성 할 수 있는 RRT 노드가 RRT 확장 지점으로 선택되도록 한다. RRT 거리 함수에 대한 정의는 3.3절에 기술되어 있다.

2) 테스트 커버리지

테스트 커버리지 측면에서 기존 연구들을 살펴보면, [17]에서는 시뮬레이션을 수 차례 실행 한 후 결과 시스템의 상태가 가장 밀집되어 있는 시스템 상태 공간을 RRT 확장의 목표지역으로 삼고 이 목표 지역을 커버할 수 있도록 RRT를 확장시킨다. [16]에서는 RRT 공간을 균등한 작은 공간으로 나눈 후, 각 RRT 노드와 작은 RRT 공간의 경계 사이의 거리의 평균 값을 이용하였다. 즉 RRT의 노드들이 RRT 공간을 조밀하게 채운 정도가 테스트 커버리지가 되었다. [14]에서는 시스템의 이산 상태에 따라 공간을 나눈 후, 이 공간을 RRT가 얼마나 커버하는지를, Star discrepancy의 근사값을 이용하여 계산 한 결과를 테스트 커버리지로 삼았다.

본 논문에서는 Stateflow 모델 기반으로 테스트 케이스를 생성하기 때문에 Stateflow 모델에서 이미 사용하고 있는 구조적 테스트 커버리지인 MC/DC[19] 기준을 따른다. 테스트 커버리지는 3.2절에 정의되어 있다. 관련 연구와는 다르게 테스트 커버리지를 산출하기 위한 추가 시뮬레이션이나 계산에 소요 되는 비용이 상대적으로 매우 작다.

3) RRT 무작위 노드 생성 방법

RRT 무작위 노드가 RRT 공간에 생성되는 위치에 따라서 RRT의 확장 방향이 결정된다. 관련 연구들은 RRT 확장이 테스트 커버리지를 높일 수 있는 방향으로 이루어 지기 위해 다양한 무작위 노드 생성 방법을 취하고 있다. [14]에서는 RRT 노드가 존재하지 않는 빈 RRT 공간 범위를 판별한 후 그 안에서 균등한 확률로 RRT 무작위 노드가 생성되도록 방향성을 부여하였다. [17]에서는 시뮬레이션을 통해 판별된 테스트 목표 공간에서 RRT 무작위 노드를 균등 또는 가우시안 분포를 이용해 생성하여 목표 공간으로 RRT 확장을 유도하였다. [16]에서는 RRT를 확장할 때 RRT 무작위 노드에 가깝게 RRT가 확장이 되었는지를 관찰하였다. RRT가 무작위 노드에 가깝게 확장이 될수록, RRT 무작위 노드의 값을 생성 하는 확률 분포의 표준편차를 줄여 RRT 노드 확장의 방향성을 더하며, RRT가 무작위 노드에 멀게 확장이 되면, 표준편차가 늘어나, RRT 확장의 방향이 분산되어 올바른 RRT 확장의 방향을 찾게 된다.

본 논문에서는 테스트 목표 달성율이 높지 않은 Stateflow의 상태, 즉 추가로 테스트 목표를 달성할 가능성이 높은 Stateflow의 상태에서 RRT가 확장 될 수 있도록, RRT 무작위 노드의 상태 활성화 변수 값 결정에 테스트 목표 달성 율을 반영하였다. RRT 무작위 노드 생성 기법은 3.4절에 기술되어 있다.

4) RRT 확장 방법

관련 연구들[14-17]의 RRT 확장 방법은 [12]의 RRT 확장 방법을 이용하였으나, 본 논문에서는 [12]의 RRT 확장 방법에 더불어, 노드를 RRT에 추가하는 기준을 변형하였다. 기존 방법에서는, RRT 무작위 노드에 가장 가깝게 근접한 RRT 노드 만을 트리에 추가하지만. 제안된 기법에서는 RRT 무작위 노드에 가장 가까운 노드 외에도, 새로운 테스트 목표를 찾은 노드 또한 트리에 추가한다. 새로운 테스트 목표를 찾은 RRT 노드는 시스템 내부 상태를 변화 시킨 중요한 노드로 취급하여, 이 후 해당 노드에서 RRT 확장이 이루어 질 경우 테스트 목표를 추가로 달성 할 가능성이 높다. RRT 확장에 대한 내용은 3.5절에 기술되어 있다.

2.3 Stateflow 모델 기반 테스트 케이스 생성 기법

Stateflow 모델 기반 테스트 케이스 생성 기법은 상업적 도구나, 연구 결과로 많은 성과가 있었다. Reactis[20]는 가장 널리 사용되는 상업적인 테스트 케이스 생성 도구로, 테스트 케이스를 생성하기 위해, 무작위 기반 휴리스틱 기법으로 시스템 입력을 생성한다. 이 도구는 휴리스틱 기법을 적용하기 위해, 도구의 사용자가 추가로 분석에 필요한 정보를 기술해줘야 하는 번거로움이 있다. Design Verifier[21]는 무작위 입력 생성 기반에 추가로 제약 해결사를 이용하여, 테스트 케이스 생성을 빠르게 하지만, 모델에 내장 MATLAB 함수나, 재귀 함수, 동적 연결 라이브러리 함수와 같은 사용자 정의 기능이 포함되어 있을 경우에는, 매우 낮은 성능을 보인다.

[8]에서는 Stateflow를 SMV[22]라는 모델 분석 프로그램으로 변환한다. 그리고 CTL[23]을 이용하여 정적 분석을 통해 테스트 케이스를 생성한다. [9]에서는 Stateflow 모델을 언어 비 종속적인 실행가능 모델로 변경한 후 Symbolic Path Finder[24]를 이용하여 테스트 케이스를 생성한다. 이 기법은 모델을 변환 할 때 요구 되는 비용이 매우 크다는 단점이 있으며, 분석이 불가능한 모델에 대해서는 테스트 케이스 생성의 효율이 떨어진다. [10]에서는 Stateflow 모델을 무작위 기반으로 시뮬레이션 한 후 Stateflow 모델 전이의 조건 분석을 통해 SMT(Satisfiability Modulo Theory) 엔진인 yices[25]를 사용하여 테스트 케이스를 생성한다. 그러나 일반적인 무작위 기반 입력 생성은 RRT 보다 그 효율이 떨어지며, yices는 비선형적인 전이 조건에는 적용할 수 없다.

2.4 Stateflow 모델

Mathworks사의 Stateflow 모델은 시스템을 이산적인 상태와, 연속적인 변수, 그리고 변수의 조건을 통한 상태의 이동인 전이로 모델링 한다. 한 상태는 여러 개의 전이를 통해 다른 상태로 이동 할 수 있으며 각 전이는 우선 순위를 가진다. 각 상태는 상태에 진입하였을 때, 상태에 머물러 있을 때, 상태를 빠져나갈 때의 실행해야 하는 행동을 정의할 수 있으며, 각 전이는 전이가 발생하였을 때 실행해야 하는 행동을 정의할 수 있다. 각 전이의 발생 조건은 '[', ']' 사이에 기술되어 있고, 전이가 발생하였을 때 행해지는 행동은 '/' 뒤에 기술되어 있다. Stateflow 변수는 Stateflow 입력 변수, Stateflow 출력 변수, Stateflow 내부 변수로 나뉘어진다.

Fig. 1은 자동차 운전석의 안전벨트가 착용되지 않았을 경우 운전자에게 알려주는 안전벨트 알림 모듈의 예제 모델이다. 모델은 *"IGN_OFF", "Belt", "Unbelt", "SBR_FastDrv", "SBR_SlowDrv", "SBR_Parking"*의 여섯 개의 상태를 가지고 있으며 전이는 총 25개 이다.

모델은 시동이 걸렸는지를 알려주는 변수 *"IGN"*, 벨트가 착용되었는지를 알려주는 변수 *"Belt"*, 자동차의 속력 *"Speed"*를 입력 변수로 받는다. 내부 변수는 시동이 걸린 후 벨트가 착용되지 않은 시간을 기록하는 *"UnbeltTimer"*가 있으며, 출력 변수는 *"Light", "Blinking", "Buzzing"*이 존재한다. 출력 변수는, 벨트를 착용하지 않는 각 상황에 대해서 점등, 깜빡임, 소리 울림을 작동시킬 것인지 결정하는 역할을 한다. *"ON", "OFF", "HIGHSPEED", "LOWSPEED", "UNBELTTIME"*은 상수이며, 각 1, 0, 40, 20, 2의 값을 가진다. *"Speed"*는 0에서 200 까지의 값을 가지며 *"UnbeltTimer"*는 0에서 *"UNBELTTIME"*+1까지의 값을 가진다.

모델의 초기 상태는 검은 점에서 시작하는 전이가 가리키는 상태 *"IGN_OFF"*가 된다. 이 상태에서 시동이 걸리게 되면 *"Unbelt"* 상태로 이동하게 된다. *"Unbelt"* 상태에서 운전자가 안전 벨트를 착용하였다면, *"Belt"* 상태로 이동하게 되지만, 벨트를 착용하지 않은 상태라면, *"Unbelt"* 상태에 머무르면서 *"UnbeltTimer"*의 값이 증가하게 된다. 이 후로 계속 벨트가 착용 되지 않은 상태라면, *"UnbeltTimer"*의 값이 계속 증가하게 되어 *"UNBELTTIME"*의 값보다 크게 되면 *"SBR_Parking"* 상태로 이동하여 안전벨트 미착용을 알리는 등이 켜지게 된다. 이 때 운전자가 운전을 시작하여, 차의 속도가 *"LOWSPEED"*를 넘어가게 되면 *"SBR_SlowDrv"* 상태로 이동하여 등이 깜박이기 시작하며, 차의 속도가 더 빨라져 *"HIGHSPEED"*를 넘어가게 되면, 경고음이 울리기 시작한다. *"SBR_Parking", "SBR_SlowDrv", "SBR_FastDrv"* 상태에서 벨트를 착용하게 되면 *"Belt"* 상태로 이동하게 되어 경고등과 경고음이 꺼지게 되며, 시동이 꺼지게 되면 *"IGN_OFF"* 상태로 이동하여 경고등과 경고음이 꺼지게 된다.

3. 테스트 목표를 반영한 RRT 확장 기법

3.1 정의

Stateflow 모델 M이 상태 $S=\{S_1, S_2, \cdots, S_n\}$, 변수 $V=\{v_1, v_2, \cdots, v_m\}$로 이루어져 있다고 하자. 상태 S_i의 활성화 정보 $A=\{a_1, a_2, \cdots, a_n\}$는 상태 S_i가 활성화되어 있으면 a_i는 1, 그렇지 않으면 0으로 정의된다. 상태 S_i에서 밖으로 나가는 전이의 집합을 T_{si}라고 하고 원소의 개수가 no_trans_{si}개라고 할 때 각각의 전이는 $trans_{si,u}(1 \le u \le no_trans_{si})$로 정

의한다. $trans_{si,u}$는 $no_conds_{si,u}$개의 단일 전이 조건을 가지고 있다. 단일 전이 조건들이 논리 *'and'*나 *'or'*로 연결되어 전이 조건 식의 결과가 참이 될 경우 전이가 발생하게 된다. 단일 전이 조건의 값은 *'true'* 또는 *'false'*를 가질 수 있으므로 $trans_{si,u}$는 $2^{no\_conds si,u}$ 개 원소를 가지는 전이 조건 값의 조합 집합 $comb_cond_{si,\ u}$을 가진다. $trans_{si,u}$가 두 개의 단일 전이 조건을 가지고 있다면, $comb_cond_{si,\ u}$은 {(T, T), (T, F), (F, T), (F, F)}가 된다. 이 때 $comb_cond_{si,\ u}\ (w)$는 $comb_cond_{si,\ u}$의 w번째 원소를 나타낸다. 전이 조건이 없는 전이는 조건 값 조합으로 {(T)}만을 가진다.

모델 M이 실행되면 현재 활성화 된 상태에서 어떤 전이로 어떤 조건 조합을 통해 전이가 이루어졌는지, 또는 이루어지지 않았는지에 대한 정보를 획득할 수 있다. 이 정보를 실행 정보(Excuted Information)이라고 한다. 모델이 $comb\_cond_{si,\ u}\ (w)$ 전이 조건 조합 통해 상태 전이를 시도하였을 경우를 나타내는 정보를 $EI_{Si,u,w}$라고 한다.

Fig. 1을 예로 들어 설명하면, 상태 *"Unbelt"*에서 나가는 전이 집합 $T_{\text{"Unbelt"}}$는 {$trans_{\text{"Unbelt",1}}$, $trans_{\text{"Unbelt",2}}$, $trans_{\text{"Unbelt",3}}$, $trans_{\text{"Unbelt",4}}$}의 원소를 가지는 집합이며, $no_trans_{\text{"Unbelt"}}$는 4가 된다. *"Unbelt"*에서 나가는 우선 순위 3의 전이 $trans_{\text{"Unbelt",3}}$은 두 개의 단일 조건 *"IGN==ON"*과 *"Belt==ON"*을 가지고 있으므로 $no_conds_{\text{"Unbelt",3}}$의 값은 2가 되며 $comb_cond_{\text{"Unbelt",3}}$은 2^2=4개의 원소를 가지며 그 원소는 {(T, T), (T, F), (F, T), (F, F)}가 된다. 이 때 $comb_cond_{\text{"Unbelt",3}}(2)$는 집합의 두 번째 원소이므로 (T, F)가 된다.

Fig. 1의 모델의 상태 *"Unbelt"*가 활성화 된 상태에서 내부 변수 *"UnbeltTimer"*의 값이 0, 출력 변수 *"Light"*, *"Blinking"*, *"Buzzing"*의 값이 모두 *"OFF"*일 때, 입력 값으로 *"IGN"*이 *"ON"*, *"Belt"*가 *"OFF"*, *"Speed"*가 0을 선택하고, 모델을 실행 하면, 먼저 상태 *"Unbelt"*의 첫 번째 우선 순위를 가지는 전이가 시도 된다. 전이의 조건이 *"IGN==ON && Belt==OFF && UnbeltTimer> UNBELTTIME"*이므로, 변수 값에 따라 단일 전이 조건의 결과 값 조합은 (T, T, F)이 된다. 따라서 첫 번째 우선 순위의 전이를 통해 실행 정보 $EI_{\text{"Unbelt", 1, 2}}$가 생성된다. 첫 번째 우선 순위의 전이 조건이 만족이 되지 않아 전이가 발생하지 않았기 때문에 두 번째 우선 순위의 전이가 시도 된다. 두 번째 전이의 조건은 *"IGN==OFF"*이므로, 입력 값에 따라 전이 조건의 결과는 (F)가 된다. 따라서 두 번째 우선 순위의 전이를 통해 실행 정보 $EI_{\text{"Unbelt", 2, 2}}$가 생성된다. 두 번째 우선 순위의 전이도 조건을 만족하지 못하여 발생하지 않았기 때문에, 세 번째 우선 순위의 전이가 시도 된다. 세 번째 우선 순위의 전이는 조건으로 *"IGN==ON && Belt==ON"*을 가지고 있으며, 입력 값에 따라, 전이 조건의 결과는 (T, F)가 된다. 따라서 세 번째 우선 순위의 전이를 통해 실행 정보 $EI_{\text{"Unbelt", 3, 2}}$가 생성된다. 세 번째 전이도 조건을 만족하지 못하여 발생되지 않았으므로, 마지막 네 번째 전이가 시도 된다. 네 번째 전이는 전이 조건을 가지고 있지 않으므로, 전이가 발생하며, 실행 정보 $EI_{\text{"Unbelt", 4, 1}}$가 생성되고, *"UnbeltTimer"*의 값이 1 증가한다. 결국 예제의 주어진 입력에서, 모델을 한번 실행하면, 총 네 개의 실

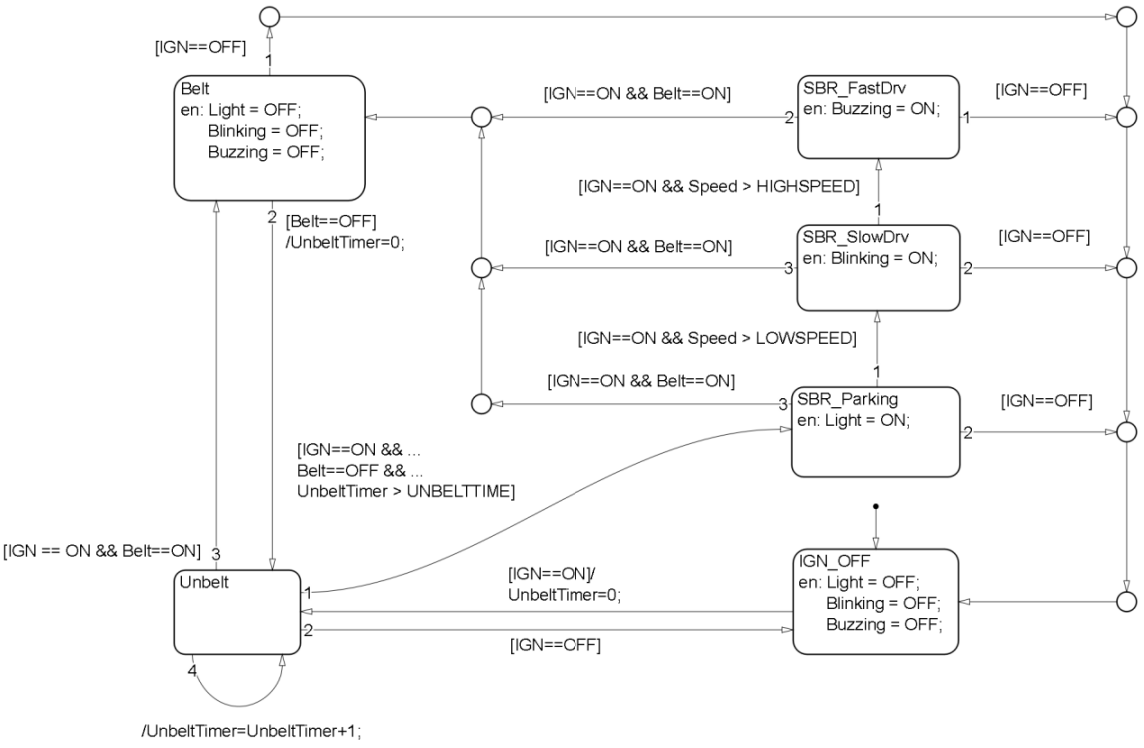

Fig. 1. Example model of Stateflow (Seat belt reminder)

행 정보 $EI_{"Unbelt",\ 1,\ 2}$, $EI_{"Unbelt",\ 2,\ 2}$, $EI_{"Unbelt",\ 3,\ 2}$, $EI_{"Unbelt",\ 4,\ 1}$가 생성된다.

RRT 노드 *nd*는 (*V, A*)로 정의되며, RRT 공간은 모든 RRT 노드의 집합으로 정의된다. nd에서 발생 될 수 있는 전이 집합 TN_{nd}는 식 (1)과 같이 정의된다. 이 때 SA_{nd}는 노드 *nd*에서 활성화 된 상태의 집합이다.

$$TN_{nd} = \bigcup_{s \in SA_{nd}} T_s \tag{1}$$

앞의 예제에서 모델의 실행이 끝난 다음에 생성되는 RRT 노드 $nd_{example}$의 내용을 살펴보면, *V={"IGN"="ON", "Belt"="OFF", "Speed"=0, "UnbeltTimer"=1, "Light"="OFF", "Blinking"="OFF", "Buzzing"="OFF"}*가 되며, *"Unbelt"* 상태의 네 번째 우선순위의 전이가 발생하였으므로, $A=\{a_{"IGN_OFF"}=0,\ a_{"Unbelt"}=1,\ a_{"Belt"}=0,\ a_{"SBR_Parking"}=0,\ a_{"SBR_SlowDrv"}=0,\ a_{"SBR_FastDrv"}=0\}$이 된다. 따라서 SA_{nd}는 *{"Unbelt"}*가 되며 $TN_{nd}=T_{"Unbelt"}=\{trans_{"Unbelt",1},\ trans_{"Unbelt",2},\ trans_{"Unbelt",3},\ trans_{"Unbelt",4}\}$가 된다.

3.2 테스트 커버리지와 테스트 목표

테스트 목표는 테스트를 수행하였을 때, 테스트 되어야 하는 항목이다. 테스트 목표는 테스트를 수행하는 시험자의 의도나, 테스트의 기준에 따라 다양하게 설정될 수 있다.

테스트 커버리지는 생성된 테스트 케이스를 이용하여 테스트를 수행하였을 때, 특정 기준에 따라 테스트가 어느 정도 수행되었는지를 수치화하는 개념이며, 식 (2)와 같이 나타낸다.

$$Test\ coverage = \frac{Number\ of\ test\ goals\ exercised}{Total\ number\ of\ test\ goals} \times 100 \tag{2}$$

이 때, 테스트 되어야 하는 목표는 테스트 커버리지의 기준에 따라 다르게 된다. Stateflow 모델은 상태 다이어그램 기반의 모델링 기법이므로, 일반적으로 상태 다이어그램에 적용될 수 있는 테스트 커버리지인 상태 커버리지, 전이 커버리지, MC/DC(Modified Condition/Decision Coverage)를 테스트 커버리지로 삼을 수 있다.

상태 커버리지는 테스트 수행 시 Stateflow 모델에 존재하는 모든 상태를 한 번 이상 활성화 시키면 커버리지가 100%가 된다. 따라서 n개의 상태를 가지고 있는 모델 M의 상태 커버리지는 n개의 각 상태를 활성화 시키는 테스트 목표를 가지고, 이 중 테스트 수행 시 몇 개의 목표가 테스트 되었는지에 따라 커버리지의 수치가 결정된다. 전이 커버리지는 모델에 존재하는 모든 전이를 한 번 이상 발생시키면 커버리지가 100%가 되며 MC/DC는 모델의 전이에 존재하는 조건 식에 대해, 기준이 되는 조건 조합을 각 한 번 이상 발생시키면 100%가 된다.

Stateflow 모델의 오류가 존재하지 않을 경우 MC/DC를 100% 만족시키면 전이 커버리지와 상태 커버리지 또한 100%를 만족하게 된다[2]. 본 논문에서는 MC/DC를 기준으로 테스트 케이스 생성을 수행한다.

MC/DC에 따라 테스트 케이스를 생성 할 때에는 조건 식을 보유한 전이에 대해서 MC/DC의 정의에 따른 조건 조합을 실행 정보로 생성해 내는 것을 테스트 목표로 한다. Fig 1의 전이 $trans_{"Unbelt",\ 3}$을 예로 들어 설명하면 해당 전이는 두 개의 단일 조건 식 *"IGN==ON", "Belt==ON"*의 *'and'* 조합으로 조건식이 이루어져 있다. 해당 전이의 MC/DC테스트 목표는 MC/DC의 정의에 따라 Table 2와 같은 전이 조건 조합이 실행 정보로 획득 되어야 한다. 만약 전이 조건 조합이 (T, F)인 테스트 목표를 달성하기 위해서는 실행 정보의 정의에 따라 Stateflow 모델이 실행 되는 중에 $EI_{"Unbelt",\ 3,\ 2}$가 획득 되야 한다.

Table 2. MC/DC test goals for transition with condition "IGN==ON && Belt==ON"

IGN==ON	Belt==ON	Excuted information
T	T	$EI_{"Unbelt",\ 3,\ 1}$
T	F	$EI_{"Unbelt",\ 3,\ 2}$
F	T	$EI_{"Unbelt",\ 3,\ 3}$

테스트 목표 달성 함수 *CS(t)*는 전이 *t*의 MC/DC테스트 목표가 얼마나 달성되었는지를 수치화하는 함수이며 식 (3)과 같이 정의된다.

$$CS(t) = \begin{cases} \frac{SatisfiedMCDC(t)}{TotalMCDC(t)} & (TotalMCDC(t) \neq 0) \\ 1 & (TotalMCDC(t) = 0) \end{cases} \tag{3}$$

*TotalMCDC(t)*는 전이 t에서 달성해야 되는 총 MC/DC 테스트 목표의 개수를 나타내며 *SatisfiedMCDC(t)*는 달성해야 되는 MC/DC 테스트 목표 중 달성된 MC/DC 테스트 목표를 나타낸다. 테스트 목표 달성율 함수를 전이의 집합에 대해 확장하면 식 (4)와 같이 나타낼 수 있다.

$$CS(T) = \begin{cases} \frac{\sum_{t \in T} SatisfiedMCDC(t)}{\sum_{t \in T} TotalMCDC(t)} & (\sum_{t \in T} TotalMCDC(t) \neq 0) \\ 1 & (\sum_{t \in T} TotalMCDC(t) = 0) \end{cases} \tag{4}$$

Fig. 1의 전이 $trans_{"Unbelt",\ 3}$을 예로 설명하면 Table 2에 따라, MC/DC를 위해 달성해야 되는 목표가 3개이므로, $TotalMCDC(trans_{"Unbelt",\ 3})$은 3이 된다. $trans_{"Unbelt",\ 2}$를 살펴보면 전이의 조건 식이 1개의 단일 조건 식으로 구성되어 있으므로, 단일 조건 식의 T/F가 테스트 목표가 된다. 따라서 $TotalMCDC(trans_{"Unbelt",\ 2})$는 2가 된다. $trans_{"Unbelt",\ 4}$에 대해서는, 해당 전이가 조건 식을 갖지 않으므로, MC/DC 목표가 정의되지 않는다.

$trans_{\text{"Unbelt", 3}}$의 MC/DC를 위해 달성해야 하는 Table 2의 목표 중 (T, T)와 (T, F)가 달성되고 (F, T)가 아직 달성되지 않은 상태라면, SatisfiedMCDC($trans_{\text{"Unbelt", 3}}$)은 2가 된다. 이 때 CS(trans"Unbelt", 3)는 2/3=0.67이 된다.

Stateflow의 상태 S_i의 목표 달성율 $CS(S_i)$는 식 (5)와 같다.

$$\mathrm{CS}(S_i) = CS(T_{S_i}) \tag{5}$$

T_{Si}는 3.1절의 정의에 따라 상태 S_i에서 나아가는 전이 집합이다. 따라서 $CS(S_i)$는 식 (4)에 따라 상태 S_i에서 발생할 수 있는 전이들의 총 달성 해야 되는 목표 개수 대비 총 달성된 목표 개수를 의미한다.

상태 *"Unbelt"*에서 나가는 전이 집합 $T_{\text{"Unbelt"}}$는 {$trans_{\text{"Unbelt",1}}$, $trans_{\text{"Unbelt",2}}$, $trans_{\text{"Unbelt",3}}$, $trans_{\text{"Unbelt",4}}$}이다. 각각의 전이에 대해서 달성해야 되는 총 테스트 목표의 개수와, 달성된 테스트 목표 개수가 Table 3과 같을 때, $CS(\text{"Unbelt"})$의 값은 (2+2+2+0)/(4+2+3+0)=0.67이 된다.

Table 3. TotalMCDC values and SatisfiedMCDC values of State "Unbelt"

$TotalMCDC(trans_{\text{"Unbelt",1}})=4$ $TotalMCDC(trans_{\text{"Unbelt",2}})=2$ $TotalMCDC(trans_{\text{"Unbelt",3}})=3$ $TotalMCDC(trans_{\text{"Unbelt",4}})=0$
SatisfiedMCDC($trans_{\text{"Unbelt",1}}$)=2 SatisfiedMCDC($trans_{\text{"Unbelt",2}}$)=2 SatisfiedMCDC($trans_{\text{"Unbelt",3}}$)=2 SatisfiedMCDC($trans_{\text{"Unbelt",4}}$)=0

RRT 노드 nd_x의 테스트 목표 달성율 $CS(nd_x)$는 식 (6)과 같이 정의된다. TN_{ndx}는 식 (1)에 따라 시스템이 RRT 노드 nd_x의 상태일 때 발생 될 수 있는 전이 집합이다.

$$CS(nd_x) = CS(TN_{nd_x}) \tag{6}$$

TN_{ndx}은 전이의 집합이므로 $CS(nd_x)$는 식 (4)에 따라 RRT 노드에서 발생 할 수 있는 전이들의 총 달성 해야 되는 목표 개수 대비 총 달성된 목표 개수를 의미한다.

3.1절의 RRT 노드 예제 $nd_{example}$에 대한 테스트 목표 달성율을 살펴보면, $nd_{example}$에서 활성화 된 Stateflow 상태는 *"Unbelt"*이므로 식 (1)에 따라 $TN_{nd_{example}}=T_{\text{"Unbelt"}}$가 되어 $CS(nd_{example})=CS(\text{"Unbelt"})$가 된다.

모든 테스트 목표 달성 함수는 0에서 1 사이의 값을 갖는다.

3.3 거리 함수 정의

두 RRT 노드 nd_x, nd_y의 거리 함수 $d(nd_x, nd_y)$는 식 (7)과 같이 정의 된다.

$$d(nd_x, nd_y) = dv(nd_x, nd_y) + CS(nd_y) \tag{7}$$

RRT에서 거리 함수를 사용할 때에는 무작위 노드에서 가장 가까운 노드를 찾을 때이다. 따라서 거리함수의 인자 중 nd_x는 반드시 고정이 된 상태에서 nd_y가 변화하면서 함수를 사용하게 된다. $dv(nd_x, nd_y)$는 두 노드의 변수 값의 거리를 나타내고, $CS(nd_y)$는 식 (6)의 노드 nd_y의 테스트 목표 달성율을 나타낸다.

Stateflow 모델의 변수들은 각기 다른 데이터 형을 가지며 불리언, 정수, 실수 형이 그 종류이다. 각 변수들은 그 최대값과 최소값 등 성격이 모두 틀리기 때문에 이를 반영해서 거리를 구할 수 있는 기법을 이용해야 한다. 본 연구에서는 Gower's General Similarity Coefficient[26]를 이용하여 변수 값들의 거리를 구한다. 두 RRT 노드 ndx, ndy의 변수 값의 거리 함수 dv(ndx, ndy)는 식 (8)과 같이 정의 된다.

$$dv(nd_x, nd_y) = \frac{\sum_{k=1}^{n(V)} ds_k(c_{kx}, c_{ky})}{n}$$

$$\begin{cases} ds_k(c_{kx}, c_{ky}) = \begin{cases} 0 \; if \; c_{kx} = c_{ky} \\ 1 \; if \; c_{kx} \neq c_{ky} \end{cases} & (c_k \; is \; boolean.) \\ ds_k(c_{kx}, c_{ky}) = \dfrac{|c_{kx} - c_{ky}|}{|c_{kmax} - c_{kmin}|} & (c_k \; is \; integer \; or \; real.) \end{cases} \tag{8}$$

c_{kx}와 c_{ky}는 RRT 노드 nd_x, nd_y의 변수 c_k에 해당하는 값이며, c_{kmax}와 c_{kmin}은 시스템 명세에 기술된 c_k의 최대값과 최소값을 의미한다. $dv(nd_x, nd_y)$은 0에서 1 사이의 값이며 노드 nd_x, nd_y의 변수 값이 같을수록 $dv(nd_x, nd_y)$의 값은 0에 가까워 지며 다를수록 1에 가까워진다.

$CS(nd_y)$의 값은 0에서 1 사이가 되므로 최종적으로 $d(nd_x, nd_y)$은 0에서 2 사이의 값을 가지게 된다.

Fig. 1의 모델을 예로 들면, 모델의 각 변수 *"IGN", "Belt", "Speed", "UnbeltTimer", "Light", "Blinking", "Buzzing"*이 각각 식 (8)의 변수 $c_1, c_2, \cdots, c_7$에 해당 된다고 하자. 그리고 상태 활성화 변수 $a_{\text{"IGN\_OFF"}}$, $a_{\text{"Unbelt"}}$, $a_{\text{"Belt"}}$, $a_{\text{"SBR\_Parking"}}$, $a_{\text{"SBR\_SlowDrv"}}$, $a_{\text{"SBR_FastDrv"}}$가 각각 식 (8)의 변수 $c_8, c_9, \cdots, c_{13}$이라고 하자. 또한 상태 *"Unbelt"*의 테스트 목표 달성율이 Table 3과 같다고 하자. 모델의 초기 상태 RRT 노드인 nd_{init}은 모델의 정의에 따라 *V={"IGN"="OFF", "Belt"="OFF", "Speed"=0, "Unbelt Timer"=0, "Light"="OFF", "Blinking"="OFF", "Buzzing"="OFF"}*, $A=\{a_{\text{"IGN_OFF"}}=1, a_{\text{"Unbelt"}}=0, a_{\text{"Belt"}}=0, a_{\text{"SBR_Parking"}}=0, a_{\text{"SBR_SlowDrv"}}=0, a_{\text{"SBR_FastDrv"}}=0\}$이 된다.

이제 nd_{init}과 앞에서의 RRT 노드 예제인 $nd_{example}$과의 거리 $d(nd_{init}, nd_{example})$을 살펴보자. 먼저 $dv(nd_{init}, nd_{example})$을 보면 (1+0+|0−0|/|200−0|+|0−1|/|3−0|+0+0+0+1+1+0+0+0+0)/13=0.26이 된다. 그리고 $CS(nd_{example})$의 값을 살펴보면 $CS(nd_{example})=CS(\text{"Unbelt"})$= (2+2+2+0)/(4+2+3+0)=0.67이 된다. 최종적으로 $d(nd_{init}, nd_{example})$는 $dv(nd_{init}, nd_{example})$+ $CS(nd_{example})$=0.26+0.67=0.93이 된다.

3.4 RRT 무작위 노드 생성 기법

RRT 무작위 노드는 V의 변수 값과, A의 변수 값을 무작위로 생성하여 만들어진다. 테스트 목표 달성 율을 높이기 위해서는 테스트 목표 달성 율이 낮은 Stateflow 상태가 활성화된 RRT 노드에서 RRT 확장을 하는 것이 유리하다.

무작위 노드 생성 시 A의 변수 값을 무작위로 생성할 때 테스트 목표 달성 율을 고려하여 생성한다. 상태 S_i에 대한 활성화 변수 a_i의 값이 1로 생성될 확률 P_{Si}은 식 (9)와 같다. 값이 1로 생성되지 않을 경우는 0으로 생성된다. 상태 S_i에 대한 활성화 값이 1이 될수록 S_i가 활성화된 RRT 노드가 선택되어 RRT 확장이 이루어질 가능성이 높아진다.

$$P_{Si} = \left(1 - CS(S_i)\right) \times \alpha + \beta \quad (\alpha \geq 0, \beta \geq 0, \alpha + \beta \leq 1) \quad (9)$$

상태 S_i에 대에서 모든 테스트 목표가 달성되더라도, 다른 상태의 테스트 목표를 달성하기 위해서는 이미 달성된 상태가 활성화 될 필요가 있기 때문에 활성화 값이 1이 될 확률은 최저 β 아래로 내려가지 않도록 한다. 또한 테스트 목표가 하나도 달성되지 않은 상태라고 하더라도 활성화가 1이 될 확률은 최고 $\alpha+\beta$ 가 된다. 본 논문에서는 효율적인 α와 β의 값을 정하는 방법에 대해서는 기술하지 않지만 대략적인 가이드라인을 제시하면, α의 값이 커지고 β의 값이 작아질수록, 테스트 목표 달성 율이 낮은 Stateflow 상태에 대한 가중치가 높아지고, 반대로 α의 값이 작아지고 β의 값이 커질수록, RRT 무작위 노드를 생성하는 데 있어서 테스트 목표 달성 율의 영향이 작아지게 된다. 테스트 목표 달성 율의 영향이 지나치게 커지게 되면, 테스트 목표 달성 율이 낮은 Stateflow 상태가 활성화 된 RRT 노드에서만 RRT가 확장되는 기아 현상이 발생 할 수 있으며, 달성 율의 영향이 매우 작아지게 되면, 달성 율이 낮은 Stateflow 상태가 활성화 된RRT 노드에서의 확장 횟수가 증가하지 않게 된다. 본 논문에서는 α의 값을 0.5, β의 값을 0.25로 하여 P_{Si}의 값을 최소 0.25, 최대 0.75까지의 범위를 가지게 하였다.

Fig. 1의 Stateflow 상태 *"Unbelt"*의 예제에서, 상태 *"Unbelt"*의 테스트 목표가 하나도 달성되지 않았다면, $CS(\text{"Unbelt"})=0$이므로, $P_{\text{"Unbelt"}}$의 값은 최대 값인 0.75가 된다. 이 후 *"Unbelt"*의 테스트 목표가 Table 3과 같이 만족이 되어, $CS(\text{"Unbelt"})$의 값이 0.67이 되었다면, $P_{\text{"Unbelt"}}$= 0.59로 확률이 감소하게 된다. 이는 테스트 목표를 달성하여 추가로 해당 Stateflow 상태와 관련된 테스트 목표를 달성할 확률이 줄었다고 판단해서, 해당 Stateflow 상태가 활성화 된 RRT 노드에서의 확장의 필요성을 감소시킨 것이다. *"Unbelt"*의 테스트 목표가 모두 달성되어 $CS(\text{"Unbelt"})$=1이 되었다면, 추가로 달성할 수 있는 테스트 목표가 존재하지 않으므로, *"Unbelt"*가 활성화 된 상태에서의 RRT 확장의 필요성이 매우 감소하게 되어, $P_{\text{"Unbelt"}}$의 값은 0.25로 최소로 감소하게 된다.

3.5 RRT 확장 기법

Table 4는 Table 1의 RRT 알고리즘의 *select_input()*과 *new_node()* 함수를 합쳐 변형한 알고리즘이다. Δt는 RRT를 한번 확장 할 때의 시스템의 시간 변화량 이며, TG는 테스트 목표로 설정된 실행 정보의 집합이다. *select_input()*에서는 nd_{near}에서부터 nd_{rand}로 되도록 가까운 상태로 변화하기 위한 입력 v를 생성하는데 nd_{rand}에 가장 가깝게 접근하게 만드는 입력을 찾는 것은 매우 비용이 많이 드는 작업이므로, 제시된 알고리즘에서는 근사치를 찾기 위해 L번의 시도를 통해 무작위 입력을 생성하고 상태를 변화 시켜서 그 중 가장 가까운 상태의 RRT 노드를 택한다. 먼저 make_random_input()에서 모델을 실행시키기 위한 입력 변수 값의 집합 I를 무작위로 생성한다. 그리고 *run_model()*에서는 RRT가 확장될 RRT 노드 nd_{near}로 모델의 상태를 변화 시킨 후, 앞에서 생성한 입력 I를 모델에 적용하여 Δt 시간 만큼 모델을 실행시킨다. 모델을 실행 시킨 후의 모델의 전체 상태가 RRT노드 nd_{cand}로 생성되며 모델이 실행되면서 발생한 실행 정보들은 실행 정보 집합 EI로 반환된다. 새로 생성된 RRT노드와 이 때 사용된 입력 집합 I는 *NewNodeList*에 삽입된다. RRT노드 생성 시도를 L번 하기 때문에 *NewNodeList*의 항목 개수도 L개가 된다.

Table 4. Modified new__node( ) for test case generation

```
mnew_node (nd_rand, nd_near, L, Δt, TG)
 NewNodeList = ∅
 NewCovList = ∅
 for l=1 to L do //Try L times for the nearest node
     I = make_random_input();
     (nd_cand , EI)= run_model(nd_near, I, Δt);
     NewNodeList.add((I, nd_cand));
     for ei in EI //Add also newly covered nodes
       if( ei∈ TG) //New test goal is covered
         TG = TG - ei;
         NewCovList.add((I, nd_cand));
   (v, nd_new) = FindNearestNode(nd_rand, NewNodeList);
return (v, nd_new, NewCovList);
```

변형된 알고리즘이 기존 알고리즘과 다른 점은 RRT 확장 시 nd_{rand}에 가장 가까운 하나의 노드 만을 추가하는 것이 아니라, L번의 시도에서 생성되는 다른 RRT 노드들 중 테스트 목표를 새롭게 달성시키는 것이 있을 경우 이 또한 RRT의 확장에 추가한다는 것이다. 새롭게 테스트 목표를 달성시킨 노드는 그 기점으로 다른 테스트 목표를 달성시킬 가능성을 가지기 때문에 다른 RRT노드보다 중요하다.

만약 RRT 확장 중에 TG에 포함된 실행 정보가 생성이 되었으면, 새로운 테스트 목표가 달성 된 것이므로, TG에서 달성된 목표를 제거하고, *NewCovList*에 해당 RRT 노드와 입력 집합을 추가한다.

L번의 RRT 노드 생성 시도가 끝나면, *FindNearestNode()*는 *NewNodeList*를 검색하여 새롭게 생성된 RRT 노드들 중, nd_{rand}와 가장 가까운 노드와 그 입력 (v, nd_{new})를 반환한다. 최종적으로 (v, nd_{new})와 함께 새롭게 테스트 목

표를 만족한 RRT 노드와 그 입력의 목록 *NewCovList*가 반환된다.

Table 5는 Table 4의 *mnew_node()*를 이용한 변형된 RRT 확장 알고리즘이다. Table 1의 기존 알고리즘에서 *select_input()*과 *new_node()* 함수가 합쳐져 *mnew_node()*으로 변경되었다.

Table 5. Modified RRT extension algorithm for test case generation

```
ModfiedExtends (nd_init, K, L, Δt, TG)
T.init(nd_init)
for k=1 to K do
  if(TG == ∅) return T;
  nd_rand ← random_node();
  nd_near ← nearest_ node(nd_rand, T);
  (v, nd_new, NewCovList)
        ← mnew_node(nd_rand, nd_near, L, Δt, TG);
  T.add_vertex(nd_new); //Add nearest node
  T.add_edge(nd_near, nd_new, v);
  for (I, nd) in NewCovList  //Newly covered nodes
    T.add_vertex(nd);
    T.add_edge(nd_near, nd, I);
return T
```

알고리즘은 모델의 초기 상태를 표현하는 RRT 노드인 nd_{init}과 RRT 확장 횟수 K, 새로운 RRT 노드를 찾기 위한 시도 횟수 L, 모델 실행 시간 Δt, 달성해야할 테스트 목표 집합 TG를 파라메터로 입력 받아 동작한다.

먼저 nd_{init}를 RRT의 루트 노드로 트리를 초기화 한다. 그리고 K번 RRT를 확장하게 된다. 각 확장 마다 먼저 *random_node()*에서 3.4절의 무작위 노드 생성 기법을 이용하여 RRT 무작위 노드 nd_{rand}를 생성한다. 그 후 *nearest_node()*에서 RRT의 노드들 중 nd_{rand}와 가장 가까운 노드 nd_{near}를 찾는다. 이 때 3.3절에서 정의된 거리함수가 사용된다. 그리고 Table 4의 알고리즘 *mnew_node()*를 통해, 새롭게 생성된 입력과, RRT 노드들 중, nd_{rand}에 가장 가까운 RRT 노드와 해당 입력인 (v, nd_{new}), 새롭게 테스트 목표를 만족시킨 입력과 RRT 노드 들의 목록 *NewCovList*가 획득된다. (v, nd_{new})는 트리의 새로운 노드로 추가되며, *NewCovList*에 포함된 각 입력과 RRT 노드 (I, nd)도 트리에 추가된다.

3.6 알고리즘 예제

Fig. 1의 예제 모델에 대하여 3.5절에서 제시한 RRT 확장 기법을 이용하여 테스트 목표를 달성하는 과정을 예를 들어 설명한다. 예제 모델의 실행 주기는 10ms이기 때문에 Table 5의 알고리즘의 파라메터 Δt는 10ms이다. 파라메터 K, L은 일반적으로 테스트 케이스 생성에 필요한 충분하고 적절한 값을 선택하나, 예제에서는 설명의 용이를 위해 각 2로 선택한다.

시스템의 초기 상태 RRT 노드 nd_{init}의 변수 값은 2.4절의 모델 설명과 같이 0 또는 "OFF"의 값을 가진다. 상태의 활성화 변수 값은 모델의 초기 상태가 *"IGN_OFF"*이기 때문에 $a_{"IGN_OFF"}$의 값은 1이며, 그 밖의 상태에 대해서는 0의 값을 갖는다.

TG는 상태 *"IGN_OFF"*와 *"Unbelt"*와 관련된 MC/DC를 만족하기 위한, 테스트 목표들의 집합으로 한다. 즉 TG는 Table 6의 테스트 목표들을 원소로 갖는다. Table 6은 상태 *"IGN_OFF"*와 *"Unbelt"*와 관련된 MC/DC 테스트 목표를 기술하고 있다.

Table 6. Test goals for algorithm example

State	Test goals
"IGN_OFF"	$EI_{"IGN\_OFF",\ 1,\ 1}$ $EI_{"IGN\_OFF",\ 1,\ 2}$
"Unbelt"	$EI_{"Unbelt",\ 1,\ 1}$ $EI_{"Unbelt",\ 1,\ 2}$ $EI_{"Unbelt",\ 1,\ 3}$ $EI_{"Unbelt",\ 1,\ 5}$ $EI_{"Unbelt",\ 2,\ 1}$ $EI_{"Unbelt",\ 2,\ 2}$ $EI_{"Unbelt",\ 3,\ 1}$ $EI_{"Unbelt",\ 3,\ 2}$ $EI_{"Unbelt",\ 3,\ 3}$

RRT 확장을 위해서 먼저 RRT 무작위 노드 nd_{rand}를 생성한다. 현재 Stateflow의 모든 상태에 대해서 달성된 테스트 목표가 하나도 없기 때문에 각 상태에 대해서 테스트 목표 달성 율은 $CS("IGN\_OFF") = 0$, $CS(""Unbelt")=0$, 나머지 상태에 대해서는 관련된 테스트 목표가 존재하지 않으므로 달성 율이 1이 된다. 모델의 상태 활성화 변수 값이 1이 될 확률은 식 (9)에 따라 *"IGN_OFF"*, *"Unbelt"*는 0.75, 나머지 상태는 0.25가 된다. nd_{rand}의 V는 *{"IGN"="OFF", "Belt"="ON", "Speed"=77, "UnbeltTimer"=3, "Light"="OFF", "Blinking"="ON", "Buzzing"="ON"}*이며, A는 $\{a_{"IGN_OFF"}=1,\ a_{"Unbelt"}=1,\ a_{"Belt"}=0,\ a_{"SBR_Parking"}=1,\ a_{"SBR_SlowDrv"}=1,\ a_{"SBR_FastDrv"}=0\}$으로 생성되었다고 하자.

다음 과정은 생성된 nd_{rand}에서 가장 가까운 노드 nd_{near}를 찾는 것이다. RRT에 속한 노드가 nd_{init}밖에 없으므로 nd_{init}이 nd_{near}로 선택 된다. 이 때 nd_{rand}와 nd_{near}r의 거리를 구하기 위해 사용한 식 (7)을 살펴보자. 거리 함수 d를 이루는 두 부분 중 테스트 목표 달성 율 부분을 살펴보면 아직까지 달성된 테스트 목표가 없기 때문에 $CS(nd_{near})$=0이 된다. 따라서 두 노드 사이의 거리는 식(8)의 두 노드의 변수 값들의 거리 dv로 결정된다. 앞에서 생성된 nd_{rand}와 nd_{near}로 선택된 nd_{init}의 dv를 계산해 보면 (0+1+|77-0|/|200-0|+|3-0|/|3-0|+0+1+1+0+1+0+1+1+0)/13=0.57이 된다.

이제 Table 4의 *mnew_node()*에서 nd_{near}에서의 새로운 RRT 노드를 생성한다. 파라메터 L의 값이 2이므로 새로운 RRT 노드 생성은 2회 시도된다. 첫번째 시도에서 무작위

입력으로 *("IGN"="OFF","Speed"=20, "Belt"="ON")*이 선택되었다고 하자. 위의 입력으로 모델을 실행하면 상태 *"IGN_OFF"*에서 조건 *"IGN==ON"*이 만족되지 못하여 전이가 발생하지 않고 *"IGN_OFF"*에 머문다. 이 때 생성된 실행 정보를 살펴보면 $EI_{"IGN_OFF",\ 1,\ 2}$가 생성 되었음을 알 수 있다. 이 결과로 생성된 RRT 노드를 nd_1이라고 하자. 생성된 실행 정보는 *TG*에 포함된 새로운 테스트 목표를 달성하였으므로, 입력 *("IGN"="OFF","Speed"=20, "Belt"="ON")*과 RRT 노드 nd_1는 *NewCovList*에 추가된다. 이 때 nd_1의 *V*는 *{"IGN"="OFF", "Belt"="ON", "Speed"=20, "UnbeltTimer"=0, "Light"="OFF", "Blinking"="OFF", "Buzzing"="OFF"}*이며, *A*는 $\{a_{"IGN_OFF"}=1,\ a_{"Unbelt"}=0,\ a_{"Belt"}=0,\ a_{"SBR_Parking"}=0,\ a_{"SBR_SlowDrv"}=0,\ a_{"SBR_FastDrv"}=0\}$이 된다.

두번째 시도에서 무작위 입력으로 *("IGN"="ON", "Speed"=49, "Belt"="OFF")*가 선택 되었다면, 상태 *"IGN_OFF"*에서 조건 *"IGN==ON"*이 만족되어 전이가 발생 한다. 그 결과로, 모델의 상태는 *"IGN_OFF"*에서 *"Unbelt"*로 이동하게 된다. 이 때 생성된 실행 정보를 보면 $EI_{"IGN_OFF",\ 1,\ 1}$이 있다. 이 결과로 생성된 RRT 노드를 nd_2라고 하자. 생성된 실행 정보는 *TG*에 포함된 새로운 테스트 목표를 달성하였으므로, 입력 *("IGN"="ON", "Speed"=49, "Belt"="OFF")*과 RRT 노드 nd_2는 *NewCovList*에 추가된다. 이 때 nd_2의 *V*는 *{"IGN"="ON", "Belt"="OFF", "Speed"=49, "UnbeltTimer"=0, "Light"="OFF", "Blinking"="OFF", "Buzzing"="OFF"}*이며, *A*는 $\{a_{"IGN_OFF"}=0,\ a_{"Unbelt"}=1,\ a_{"Belt"}=0,\ a_{"SBR_Parking"}=0,\ a_{"SBR_SlowDrv"}=0,\ a_{"SBR_FastDrv"}=0\}$이 된다.

입력 생성 시도가 모두 끝났으므로, *FindNearestNode()*에서, 생성된 노드 nd_1, nd_2 중 nd_{rand}에 더 가까운 노드를 판별한다. 노드 nd_1은 상태 *"IGN_OFF"*가 활성화 되어 있고, Table 6의 *"IGN_OFF"*와 관련된 테스트 목표는 앞의 모델 실행을 통해 모두 달성되었으므로, $CS(nd_1)=1$이 된다. 그리고 nd_2는 상태 *"Unbelt"*가 활성화 되어 있고, *"Unbelt"*와 관련된 테스트 목표는 달성된 것이 없으므로 $CS(nd_2)=0$이 된다. nd_{rand}와 nd_1의 *dv*값을 계산하면, (0+0+|77-20|/|200-0|+|3-0|/|3-0|+0+1+1+0+1+0+1+1+0)/13=0.48이 된다. 최종 nd_{rand}와 nd_1의 거리는 $dv(nd_{rand},\ nd_1) + CS(nd_1)$= 0.48+1=1.48이 된다. nd_{rand}와 nd_2의 *dv*값을 계산하면, (1+1+|77-49|/|200-0|+|3-0|/|3-0|+0+1+1+1+0+0+1+1+0)/13=0.63이 된다. 따라서 nd_{rand}와 nd_2의 거리는 $dv(nd_{rand},\ nd_2) + CS(nd_2)$=0.63+0=0.63이 된다. 최종 거리가 nd_2가 nd_1보다 작으므로 nd_2가 nd_{rand}에서 가장 가까운 노드로 선택된다. *mnew_node()*에서 생성한 노드들 중 nd_{rand}와 가장 가까운 노드와 입력들, 그리고 새로운 테스트 목표를 만족한 노드들, 즉 nd_1, nd_2와 그 입력을 RRT에 추가함으로써 첫 확장이 끝났다. 달성된 테스트 목표 $EI_{"IGN\_OFF",\ 1,\ 1}$, $EI_{"IGN_OFF",\ 1,\ 2}$는 *TG*에서 제거된다.

두 번째 확장에서 RRT 무작위 노드를 생성할 때에는 각 상태의 활성화 변수 값이 1이 될 확률이 변경된다. *"IGN_OFF"*의 경우에는 이전 RRT 확장 과정에서 해당하는 모든 테스트 목표를 달성했기 때문에 $CS("IGN_OFF")=1$이 된다. *"Unbelt"*은 여전히 달성된 테스트 목표가 없기 때문에 $CS("Unbelt")=0$이 된다. 이에 따라 활성화 변수 값이 1이 될 확률은 *"Unbelt"*가 0.75, 나머지 상태가 각 0.25가 되었다. 따라서 *"Unbelt"*가 활성화 된 RRT 무작위 노드가 만들어질 확률이 높아지며, 무작위 노드에서 가까운 RRT 노드들도 *"Unbelt"*가 활성화 된 RRT 노드가 선택될 확률이 높아진다. 테스트 목표 집합 *TG*에서 남은 테스트 목표들은 전부 상태 *"Unbelt"*가 활성화 된 상태에서 달성할 수 있음을 유의하자. 생성된 무작위 노드 nd_{rand}의 *V*는 *{"IGN"="ON", "Belt"="ON", "Speed"=196, "UnbeltTimer"=1, "Light"="ON", "Blinking"="OFF", "Buzzing"="ON"}*이며, *A*는 $\{a_{"IGN_OFF"}=0,\ a_{"Unbelt"}=1,\ a_{"Belt"}=1,\ a_{"SBR_Parking"}=0,\ a_{"SBR_SlowDrv"}=0,\ a_{"SBR_FastDrv"}=0\}$으로 생성되었다고 하자.

이제 RRT 무작위 노드에서 가까운 노드를 선택해야 한다. 현재 RRT에는 노드 3개 nd_{init}, nd_1, nd_2가 존재한다. 각 노드의 목표 달성율을 보면 nd_{init}, nd_1은 상태 *"IGN_OFF"*가 활성화 되었으므로, 식 (6)에 따라, $CS(nd_{init})=CS(nd_1)=1$이 된다. nd_2는 *"Unbelt"*가 활성화 된 상태이므로, $CS(nd_2)=CS("Unbelt")=0$이 된다. 이제 RRT 무작위 노드 nd_{rand}와 RRT의 각 노드 사이의 거리를 살펴보자. nd_{rand}의 값이 어떻게 생성되는가와 관계 없이 nd_{init}, nd_1은 테스트 목표 달성율에 의해 최소 1의 거리 값을 갖는다. nd_2는 $CS(nd_2)=0$이므로, $dv(nd_{rand},\ nd_2)$에 의해 거리가 결정이 되며, 이 값은 최대 1의 값을 갖는다. 즉, nd_{rand}에서 nd_2사이의 거리가 다른 노드 간의 거리보다 가까울 확률이 매우 높게 된다. 따라서 RRT 확장은 테스트 목표 달성 율이 떨어지는 노드 nd_2에서 이루어지도록 유도된다.

이제 nd_2에서 무작위 입력을 2회 생성하여, 새로운 RRT 노드를 생성한다. 무작위 입력의 첫 번째 시도로, *("IGN"="OFF","Speed"=100, "Belt"="ON")*가 생성되었다면, nd_2는 *"Unbelt"* 상태가 활성화 되어 있으므로, 해당 상태에서 먼저 우선 순위 1의 전이의 조건이 검사 된다. 모델의 변수 값에 따라 전이 조건 조합의 결과가 (F, F, F)가 되며 실행 정보 $EI_{"Unbelt",\ 1,\ 8}$이 생성 된다. 이 실행정보는 *TG*에 포함되지 않기 때문에 무시된다. 그 다음으로 조건이 만족된 우선 순위 2의 전이가 발생하고 모델의 상태가 *"IGN_OFF"*로 이동한다. 이 때 실행 정보 $EI_{"Unbelt",\ 2,\ 1}$이 생성 되며, 새로운 테스트 목표를 만족하여 *TG*에서 해당 테스트 목표는 삭제된다. 이 결과로 생성된 RRT 노드를 nd_3이라 하자.

무작위 입력의 두 번째 시도로, *("IGN"="ON","Speed"=134, "Belt"="OFF")*가 생성되었다고 하자. 먼저 *"Unbelt"*의 우선 순위 1의 전이의 조건이 검사되어 결과가 (T, T, F)가 나온다. 이 결과로 실행 정보 $EI_{"Unbelt",\ 1,\ 2}$가 생성되며, 새로운 테스트 목표를 달성하여 *TG*에서 제거 된다. 전이의 조건을 만족하지 못하였으므로 계속해서 우선 순위 2의 전이 조건이 검사 된다. 결과로 (F)가 생성된다. 이에 따라 실행정보 $EI_{"Unbelt",\ 2,\ 2}$가 생성되었다. $EI_{"Unbelt",\ 2,\ 2}$ 또한 새롭게 달성된 테스트 목표가 되며, *TG*에서 제거 된다. 우선 순위 2의 전이 또한 발생하지 못하였으므로 우선 순위 3의 전이의

Table 7. Test case samples

Path	Test cases	Achieved test goals
$nd_{init} \rightarrow nd_1$	("IGN"="OFF","Speed"=20, "Belt"="ON")	$EI_{"IGN\_OFF",\ 1,\ 2}$
$nd_{init} \rightarrow nd_2$	("IGN"="ON", "Speed"=49, "Belt"="OFF")	$EI_{"IGN\_OFF",\ 1,\ 1}$
$nd_{init} \rightarrow nd_2 \rightarrow nd_3$	("IGN"="ON", "Speed"=49, "Belt"="OFF") ("IGN"="OFF","Speed"=100, "Belt"="ON")	$EI_{"Unbelt",\ 2,\ 1}$
$nd_{init} \rightarrow nd_2 \rightarrow nd_4$	("IGN"="ON", "Speed"=49, "Belt"="OFF") ("IGN"="ON","Speed"=134, "Belt"="OFF")	$EI_{"Unbelt",\ 1,\ 2}$ $EI_{"Unbelt",\ 2,\ 2}$ $EI_{"Unbelt",\ 3,\ 2}$

조건이 검사 된다. 조건 조합의 결과는 (T, F)가 되어 새로운 테스트 목표 $EI_{"Unbelt",\ 3,\ 2}$가 달성되어 TG에서 제거 된다. 우선 순위 3의 전이도 발생하지 못하였으므로 마지막으로 우선 순위 4의 전이가 발생하여, 모델의 상태는 *"Unbelt"*에 머물게 되며 *"UnbeltTimer"*값이 1 증가하게 된다. 이 결과로 생성된 RRT 노드를 nd_4라 하자.

두 번째 RRT 확장 시도로 생성된 노드 nd_3과 nd_4도 앞의 확장 과정에서와 같이 nd_{rand}로부터의 거리가 측정되어 거리가 더 짧은 노드와 그 입력이 RRT에 추가되며, 새롭게 테스트 목표를 달성한 노드와 입력 또한 RRT에 추가된다. 노드 nd_3, nd_4는 새롭게 테스트 목표를 달성하였으므로 모두 RRT에 추가된다.

알고리즘 예제에서 생성된 테스트 케이스 샘플은 Table 7과 같다.

4. 실 험

제시된 알고리즘을 평가하기 위해서 Fig. 1의 모델과 더불어 Mathworks사에서 제공하는 기어 변속 장치의 변형모델과, 추가로 자동차에 사용되는 서리 제거 장치, 파워 윈도우 제어 장치, 실내등 제어 장치의 Stateflow 모델을 대상으로 실험을 수행한다. 실험은 제안된 알고리즘(Proposed)과 [12]의 일반적인 RRT 확장 알고리즘(Typical), [16]의 적응 샘플링 기법(Adaptive)으로 각각 RRT 확장을 수행하였을 시, 확보되는 테스트 커버리지를 상태 커버리지, 전이 커버리지, MC/DC 측면에서 비교한다. 알고리즘의 RRT 확장 횟수 파라메터 K는 100000, 새로운 RRT 노드 확장을 위해 후보를 뽑는 파라메터 L은 5로 설정하였으며, 실험 대상 컴퓨터의 CPU는 Intel Core i5 2.67GHz, RAM은 4.00GB이다.

Table 8은 Fig. 1의 예제 모델 외의 Mathworks 예제 모델과, 실제 자동차에서 사용되는 모델들의 통계 정보이며, Table 9는 각 모델에 대한 실험 수행 결과이다.

모델 Gear shift logic에 대한 실험 결과를 살펴보면 기존 알고리즘과 제시된 알고리즘의 결과가 동일한 것을 확인할 수 있는데, 이는 모델이 간단하여 찾을 수 있는 모든 테스트 목표에 대한 테스트 케이스 생성이 완료되었기 때문이다. Gear shift logic 모델 외에 실제 자동차에 사용되는 세 모델과 예제 모델에 대해서는 실험 결과의 차이를 보인다. 제안한 알고리즘의 성능이 전체적으로 다른 두 알고리즘의 성능보다 높은 것을 확인 할 수 있다. 이는 테스트 목표를 달성 시킬 가능성이 높은 RRT 노드로부터 트리를 확장 하는 것이 테스트 커버리지를 높이는데 기여를 한다는 논문의 아이디어가 효과가 있음을 나타낸다.

Table 8. Statistics of example models

Model	States	Transitions	Variables
Gear shift logic	9	14	6
Defroster	9	9	6
Power window	7	7	8
Interior lamp	59	99	89

각 테스트 커버리지 별로 수치를 살펴보면 상태 커버리지가 평균적으로 가장 수치가 높으며, 그 다음이 전이 커버리지, 마지막으로 MC/DC가 가장 낮은 수치를 보인다. 이는 커버리지의 달성 난이도 상 당연한 결과이다. 제안된 알고리즘은 기존 알고리즘과 비교했을 때, 상태 커버리지 달성율에 비해, 전이 커버리지와 MC/DC 달성율에서는 큰 차이를 보인다. 이는 전이 조건이 특정 조합을 달성하는 실행이 이루어져야 달성 되는 전이 커버리지와 MC/DC 특성에 따라 다양한 변수 값의 조합을 시도하기 위해, RRT 알고리즘의 거리 함수와 RRT 무작위 노드 생성 시, RRT 노드와 Stateflow의 상태에 대한 테스트 목표 달성율 수치를 반영함으로써, 테스트 목표 달성율이 낮은 RRT 노드에 대한 집중적인 RRT 확장 시도가 이루어졌기 때문이다. 또한 새로운 테스트 목표를 달성한 RRT 노드를 RRT 확장에 포함시켰기 때문에, 이 노드를 기반으로 테스트 목표가 달성될 확률이 증가하게 된다.

모델의 명세 입장에서 실험 결과를 분석해보면, 모델의 상태와 전이는 모델 작성자가 대상 시스템에서 반드시 실행되어야 한다는 의도로 작성한 것이기 때문에, 이론상 모델의 상태와 전이는 적어도 한번씩은 방문할 수 있어야 한다. 즉 모델의 상태 커버리지와 전이 커버리지의 달성율은 100%가 나와야 한다. 그러나 이는 서론에서도 언급하였지만

Table 9. Coverage ratio of example models

Model		State	Transition	MC/DC	Total
Seat Belt Reminder	Typical	100%	28%	52.78%	48.53%
	Adaptive	100%	60%	58.33%	63.24%
	Proposed	100%	88%	77.78%	83.82%
Gear shift logic	Typical	100%	100%	90.91%	95.65%
	Adaptive	100%	100%	90.91%	95.65%
	Proposed	100%	100%	90.91%	95.65%
Defroster	Typical	88.89%	77.78%	38.46%	64.52%
	Adaptive	88.89%	77.78%	46.15%	77.42%
	Proposed	100%	100%	84.62%	93.55%
Power window	Typical	71.43%	57.14%	9.09%%	40%
	Adaptive	100%	100%	27.27%	68%
	Proposed	100%	100%	54.55%	80%
Interior lamp	Typical	93.22%	70.71%	59.62%	66.99%
	Adaptive	96.61%	90.91%	70.38%	78.95%
	Proposed	100%	95.96%	80%	86.36%

도달 가능성 문제를 100% 해결 할 수 있어야 달성할 수 있는 어려운 문제이다. 달성을 못한 목표에 대해서는 모델의 정적, 동적 분석을 통해 달성할 수 있는 확률을 높일 수 있는 휴리스틱 기법이 연구 되야 한다.

MC/DC는 앞의 두 커버리지와는 다르게 테스트 케이스생성 기법의 한계로 인해 달성 못하는 테스트 목표 외에, 모델의 실행 논리에 따라서 변수 사이의 의존 관계가 생기기 때문에 달성 불가능한(infeasible) 테스트 목표가 생기게 된다. 예를 들어 한 전이의 조건식이 (A || B)일 경우에, A, B 둘 중 하나의 값이 반드시 참이 되는 구조라면, (F, F) 조합이 나오는 MC/DC 목표는 절대 달성 할 수 없게 된다. 이러한 달성 불가능한 테스트 목표가 실험 결과에 다수 포함되어 있다. 향후 연구를 통하여 모델에 따른 달성 불가능한 테스트 목표를 판별 할 수 있다면, 이를 목표 달성 율에서 제외 시킴으로써 MC/DC의 수치를 높일 수 있다.

5. 결 론

본 논문에서는 Stateflow 모델 기반의 테스트 케이스 생성에 사용하기 위한 변형된 RRT 알고리즘을 제안하였다. RRT 알고리즘은 그 특성으로 인해 테스트 케이스 생성 알고리즘에 사용하면 도달 가능성 문제를 매우 효과적으로 해결 할 수 있는 기법이다. 제안된 기법에서는 Stateflow 모델에 맞는 테스트 커버리지 수치를 높일 수 있는 RRT 알고리즘을 제시하였다. RRT 거리 함수와 RRT 무작위 노드 생성 방법에 테스트 목표 달성 율을 반영함으로써, 테스트 목표 달성 율이 낮은 RRT 노드를 먼저 확장하는 방향으로 RRT 확장 기법을 변경하였으며, RRT 확장 시 새로운 노드를 만들 때 새롭게 달성된 테스트 목표가 있을 경우 이 또한 RRT 확장에 추가함으로써, 새로운 노드를 기반으로 테스트 목표를 달성시킬 수 있는 확률을 높였다. 테스트 제안된 알고리즘의 성능은 실험을 통해 기존의 RRT 기반 테스트 케이스 생성 알고리즘보다 테스트 커버리지 수치가 더 향상된 결과를 보여줌으로써 입증되었다.

참 고 문 헌

[1] MATLAB Simulink Stateflow http://www.mathworks.com/products/stateflow/ 1994-2012 The MathWorks, Inc.

[2] M.Utting, B. Legeard, “Practical Model-Based Testing: A Tools Approach”, Morgan Kaufmann, 2007.

[3] R. Alur, “Model checking of hierarchical state machines” in Proceedings of the 6th ACM SIGSOFT FSE, Vol.23, Issue 6, pp.175-188, 1998.

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

[5] A. Agrawal, G. Simun, 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.

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

[7] T. Brihaye, "A note on the undecidability of the reachability problem for o-minimal dynamical systems" in Mathematical Logic Quarterly, Vol.52, Issue 2, pp.165-170, 2006.

[8] H. S. Hong, I. S. Lee, O. Sokolsky, 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.

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

[10] M. Satpathy, A. Yeolekar, 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.

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

[12] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning" TR 98-11 Computer Science Dept., Iowa State Univ.

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

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

[15] 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.

[16] J. Kim, "Adaptive Sample Bias for Rapidly-exploring Random Trees with Applications to Test Generation" in American Control Conference, Proceedings, Vol.2, pp.1166-1172, 2005.

[17] S. N. Ahmadyan, "Goal-oriented stimulus generation for analog circuits", in Design Automation Conference, pp.1018-1023, 2012.

[18] S. M. LaValle, J. J. Kuffner. In B. R. Donald, K. M. Lynch, D. Rus, "Rapidly-exploring random trees: Progress and prospects.", Algorithmic and Computational Robotics: New Directions, pp.293-308. A K Peters, Wellesley, MA, 2001.

[19] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, L. K. Rierson, "A Practical Tutorial on Modified Condition/ Decision Coverage", NASA, 2001.

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

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

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

[23] E.M. Clarke, E.A. Emerson, 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.

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

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

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

박 현 상

e-mail : elvaimay@ajou.ac.kr

2005년 아주대학교 정보 및 컴퓨터 공학부 (공학사)

2007년 아주대학교 정보통신전문대학원 정보통신공학과(공학석사)

2007년~현 재 아주대학교 정보통신전문대학원 정보통신공학과 박사과정

관심분야 : 소프트웨어 공학, 임베디드 시스템, 운영 체제, 실시간 고속 네트워크 시스템 등

최 경 희

e-mail : khchoi@ajou.ac.kr

1976년 서울대학교 수학교육과(학사)

1979년 프랑스 그랑데꼴 Enseeiht 대학 (석사)

1982년 프랑스 Paul Sabatier 대학 정보공학부(박사)

1982년~현 재 아주대학교 정보통신전문대학원 교수

관심분야 : 운영체제, 분산시스템, 실시간/멀티미디어 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 전자공학부 교수

관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어/실시간 시스템 등