http://dx.doi.org/10.3745/KTSDE.2013.2.7.471

Test-case Generation for Simulink/Stateflow Model using a Separated RRT Space

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

ABSTRACT

This paper proposes a black-box based test case generation method for Simulink/Stateflow model utilizing the RRT algorithm which is a method to efficiently solve the path planning for complicated systems. The proposed method in the paper tries to solve the reachability problem with the RRT algorithm, which has to be solved for black-box based test case generations. A major problem of the RRT based test case generation algorithms is that the cost such as running time and required memory size is too much for complicated Stateflow model. The typical RRT algorithm expands rapidly-exploring random tree (RRT) in a single state space. But the proposed method expands it in dynamic state space based on the state of Simulink model, consequently reducing the cost. In the paper, a new definition of RRT state space, a distance measure and a test case generation algorithm are proposed. The performance of proposed method is verified through the experiment against Stateflow model.

Keywords : Black-box Testing, Model Based Testing, RRT, Simulink/Stateflow, Test Case Generation

분할된 RRT 공간을 이용한 Simulink/Stateflow모델 테스트케이스 생성

박 현 상† · 최 경 희†† · 정 기 현†††

요 약

본 논문은 Rapidly-exploring Random Tree(RRT) 알고리즘을 이용한 Simulink/Stateflow 모델 기반의 블랙박스 테스트 케이스 자동 생성 기법을 제안한다. RRT는 복잡한 시스템의 경로 계획을 효율적으로 해결하는 좋은 방법으로 널리 사용되고 있다. 본 논문에서 제안하는 기법은 블랙박스 테스트 케이스 생성 시 해결해야 되는 도달 가능 문제를 RRT를 통해 해결하고자 한다. RRT를 이용하여 테스트 케이스를 생성 할 때의 가장 큰 단점은 Stateflow 모델의 내부 상태가 복잡한 시스템을 위한 RRT 확장 시 시간과 메모리 측면에서 많은 비용이 발생하게 된다는 점이다. 일반적인 RRT 기법이 대상 시스템을 단일한 RRT 공간으로 구성 하는 반면 제안된 기법에서는 대상 시스템을 Stateflow의 상태를 기준으로 동적 분할하여 RRT 공간을 모델링 구성 함으로써 RRT 확장 시 필요한 비용을 감소시켰다. 본 논문에서는 분할 RRT 공간을 위한 RRT 공간의 정의와, 거리 측정 기법, 테스트 케이스 생성 알고리즘을 제시한다. 또한, 예제 Stateflow 모델을 기반으로 한 테스트 케이스 생성 실험을 통해 제안된 알고리즘의 성능을 보인다.

키워드 : 블랙박스 테스팅, 모델 기반 테스팅, RRT, Simulink/Stateflow, 테스트 케이스 생성

1. 서 론

모델 기반 블랙박스 테스트에서 주로 사용되는 시스템 모델링 기법으로 Mathworks사에서 제공하는 Stateflow [1]가 있다. Stateflow는 상태 다이어그램의 일종으로 시스템의 변화를 이산적인 상태와 연속적인 상태, 두 측면에서 동시에 나타낼 수 있기 때문에, 널리 사용되고 있는 시스템 모델링 기법이다. Stateflow는 시스템의 상태와 그에 따른 동작을 정형화된 방법으로 기술할 수 있다는 장점이있으나, 시스템의 이산적인 상태를 모델의 상태로 정확하게 매핑하기 어렵기 때문에 내부 변수나, 전이 조건과 같은 개념을 추가로 사용한다는 단점이 있다. 이는 테스트 대상 모 델의 복잡성을 높여 시스템 검증을 어렵게 하는 원인이 된다.

Stateflow 모델 기반 블랙박스 테스팅을 수행하기 위해서는 먼저 Stateflow모델을 기반으로 테스트 케이스를 생성한다. 이 후 테스트 케이스를 테스트 대상 시스템과 모델에

† 준 회 원 : 아주대학교 정보통신전문대학원 정보통신공학과 박사과정
†† 정 회 원 : 아주대학교 정보통신전문대학원 교수
††† 정 회 원 : 아주대학교 전자공학부 교수
논문접수 : 2013년 2월 26일
수 정 일 : 1차 2013년 4월 4일
심사완료 : 2013년 4월 8일
* Corresponding Author : Hyeon Sang Park(elvaimay@ajou.ac.kr)

입력하여 실행한 결과를 비교, 분석하여 테스트 결과를 도출한다. Stateflow 모델 기반 테스트 케이스 생성 전략은 많은 선행 연구가 이루어졌다. 상업적인 도구로는 [2-4]가 있으며, 다른 연구 결과들로는 [5-9]가 있다.

모델 기반 블랙박스 테스팅을 수행할 때에는 대상 시스템의 내부 상태나, 변수를 직접 제어할 수 없고 시스템의 말단 입력만을 제어할 수 있다. 따라서 시스템을 테스트하고자 하는 목표 상태로 제어하기 위해서는 시스템의 초기 상태로부터 목표 상태까지 도달하기 위한 경로와 그에 따른 시스템의 말단 입력 순열을 구해야 한다. 이를 도달 가능 문제(reachability problem)[10]라고 한다. 블랙박스 테스트를 수행하기 위해서는 반드시 도달 가능 문제를 해결해야, 시스템에 적용할 수 있는 테스트 케이스를 생성할 수 있다. 하지만 Stateflow 모델은 대부분 비선형적인 하이브리드 오토마타에[11] 속하기 때문에 도달 가능 문제를 푸는 것은 [12,13]에 따르면 결정 불가능(undecidable) 하고, 문제를 푸는데 시간이 매우 오래 걸린다. 근사값을 구하는 알고리즘들도 많은 비용이 필요하거나 결정 불가능하기 때문에 적용하기 힘들다[14].

도달 가능 문제를 해결하기 위한 방법 중 하나로 동작 계획에서 사용하는 RRT 알고리즘[15]을 테스트 케이스 생성에 이용하는 연구가 진행되었다. RRT 알고리즘은 시스템의 초기 상태에서 테스트 목표 상태로 도달하는 경로를 빠르게 찾을 수 있기 때문에 도달 가능 문제를 해결 할 수 있다. RRT를 사용한 테스트 케이스 생성 연구 결과로는[16-19]들이 있다. 선행 연구에서는 테스트 대상 시스템의 상태 변화를 단일 RRT공간에서 모델링 하여 테스트 케이스 생성 시 시스템의 상태와는 관련 없이 시스템의 모든 변수를 참조하게 된다. 그러나 Stateflow 모델에서는, Stateflow의 활성화된 상태에서 다음 상태로의 전이를 결정하는 변수는 시스템의 전체 변수 중 일부분이다. 따라서 시스템을 Stateflow의 각 상태에 따라 참조되는 변수로 구성된 다중 상태 공간으로 모델링 한다면, 대상 시스템의 활성화 된 상태에 따라 RRT 알고리즘에서 참조하는 변수의 수가 감소하게 된다. 이는 RRT 알고리즘을 이용한 테스트 케이스 생성 기법의 성능을 향상 시킬 수 있는 요인이다.

본 논문의 구성은 다음과 같다. 2장에서는 논문의 배경인 Stateflow 모델과 RRT 알고리즘, 테스트 케이스 생성 알고리즘에 대한 선행 연구에 대해 각각 기술하며, 3장에서는 분할된 상태 공간에 따른 테스트 케이스 알고리즘. 4장에서는 예제 Stateflow을 기반으로 한 실험을 통해 선행 연구에서 제시된 알고리즘과 비교하여 제안된 기법의 유용성을 보였다. 마지막으로 5장에서 연구의 결론과 향후 연구 계획을 기술한다.

2. 연구 배경 및 관련 연구

2.1 Rapidly Exploring Random Trees(RRT)

시스템이 변수 집합 $V = \{v_1, v_2, \cdots, v_m\}$로 구성된다고 할 때 특정 시점에서의 시스템의 상태는 V 의 원소의 값 조합으로 나타낼 수 있다. RRT 알고리즘은 시스템의 상태를 트리의 노드로 보고 한 노드 nd_x에서 다른 노드 nd_y로 이동할 수 있는 경로, 즉 시스템의 입력 순열을 찾는 알고리즘이다.

많은 종류의 RRT 알고리즘들이 존재하지만[20], [17]의, Table 1이 가장 기본적인 형태이다. 시스템의 초기 상태 nd_{init}, 트리 확장 횟수 K, 확장 시 시스템이 진행될 시간 Δt가 주어진 상태에서 트리 확장이 시작된다. 각 반복에서 시스템의 무작위 상태 nd_{rand} 가 생성된다. 그리고 *nearest_node()*에서 RRT 노드 중 nd_{rand} 와 가장 가까운 노드 nd_{near} 가 찾아진다. 그 후 시스템 상태 에서 적용하기 위한 입력 조합 v을 임의로 선택한다. 이 입력이 시스템에 시간 Δt 동안 적용되어 새롭게 만들어진 시스템 상태 nd_{new} 가 트리에 추가된다. 결과적으로 반복이 K 번 수행 되면서 트리에 K 개 추가되어 트리가 구축된다.

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
```

RRT 알고리즘에서 가장 중요한 요소는 RRT 노드 사이의 거리를 측정하는 함수이다. 이 함수의 성능에 따라서 목표 RRT 노드 까지의 경로를 탐색하는 비용이 좌우된다[20]. RRT 관련 선행 연구들에서는 각 상황에 맞는 거리 측정 함수를 정의하여 사용하고 있다.

2.2 Stateflow 모델 및 테스트 케이스 생성 기법

Stateflow는 시스템을 개념적인 이산 상태로 보고, 상태의 변화는 전이로 나타내며, 각 전이는 전이가 발생하기 위한 조건과, 조건을 만족하였을 때 일어나는 행동을 포함한다. 한 상태에서 발생하는 전이가 여러 개일 때를 고려하여, 전이들은 우선순위를 가지게 된다. Stateflow가 포함하는 변수는 크게 시스템 입력 변수, 시스템 출력 변수, 시스템 내부 변수로 나뉘어질 수 있으며, 이는 실제 시스템의 입, 출력과 내부 상태와 동일하게 표현된다.

Fig. 1은 Mathworks 에서 제공하는 Stateflow의 예제의 변형이다. 자동차의 속력과, 상한 임계치, 하한 임계치를 입력으로 하여 기어의 올림, 내림을 결정짓는 자동 변속 기어의 간단한 모델이다. 점에서 나아가는 화살표가 가리키는 *"steady_state"*상태가 시스템의 초기 상태이며, 각 전이의

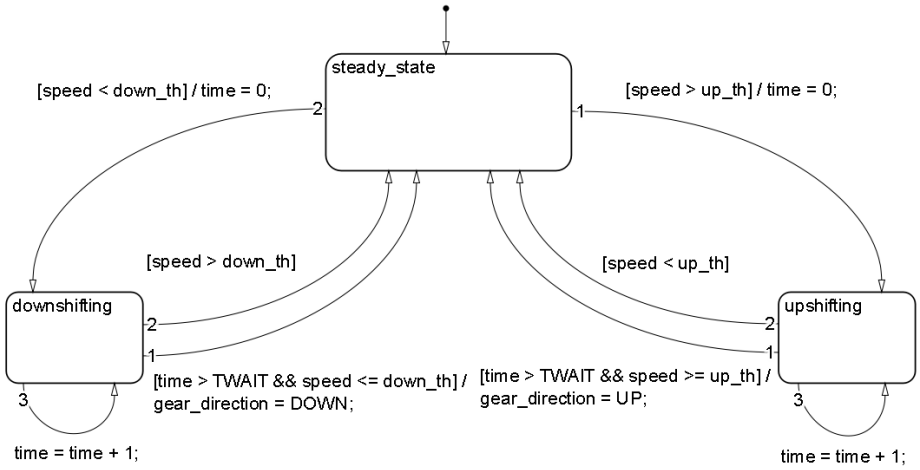

Fig. 1. Mathworks Simulink Stateflow example[21]

발생 조건은 '[', ']' 사이에 기술되어 있고, 전이가 발생하였을 때 행해지는 행동들은 '/' 뒤에 기술되어 있다. 모델은 세 개의 입력 변수 *"speed"*, *"down_th"*, *"up_th"*, 하나의 내부 변수 *"time"*, 하나의 출력 변수 *"gear_direction"*을 가지며, 초기 값은 모두 0이다. UP, DOWN, NONE, TWAIT는 미리 값이 정해진 상수로 각각 0, 1, 2, 1의 값을 갖는다.

Simulink/Stateflow 기반 테스트 케이스 생성 기법은 많은 연구 결과가 있다. 먼저 상업적 도구를 살펴보면 Reactis[2]는 테스트 케이스를 생성하기 위해서 무작위 기반 휴리스틱 기법으로 시스템 입력을 생성하기 때문에 테스트 케이스 생성 결과 테스트 커버리지 수치의 변동이 크며, 테스트 케이스 생성 시간이 길다. T-VEC Tester[3]는 constraint solving 기술, Design Verifier[4]는 무작위 기반에 추가로 SAT solver[22]와 같은 증명 엔진을 추가적으로 사용함으로써 이와 같은 단점을 보완하였지만 내장 MATLAB 함수, 재귀 함수 등 모든 Stateflow 구성 요소를 처리하지 못한다는 단점이 있다. [5]에서는 Stateflow를 SMV 프로그램[23]으로 바꾼 후 CTL[24] 공식을 통해 정적 분석으로 테스트 케이스를 생성하였다. 이 기법은 Stateflow 모델이 커질 경우 요구 메모리가 크게 증가하는 단점이 있다. [6]에서는 Stateflow 모델을 시뮬레이션 하여 나온 결과 피드백을 이용하여 무작위 입력을 조정함으로써 테스트 커버리지 수치가 떨어지는 단점을 보완하였다. [7]에서는 Stateflow 모델을 언어 비종속적인 실행가능 모델로 변경한 후 Symbolic Path Finder[25]를 이용하여 테스트 케이스를 생성함으로써 원하는 커버리지의 테스트 케이스를 생성할 수 있다. [8]는 유전자 알고리즘 기반 무작위 기법을 이용하여 테스트케이스를 생성함으로써 단순 무작위 기법의 단점을 보완하였다.

RRT를 이용한 테스트 케이스 생성 알고리즘은 주로 하이브리드 오토마타[26]를 대상으로 진행되었다. 하이브리드 오토마타는 Stateflow와 같이 복잡한 시스템을 모델링 하기 위해 사용되는 언어이며, Stateflow는 하이브리드 오토마타로 변형이 가능하다는 선행 연구가 이루어졌다[11]. RRT를 이용한 선행 테스트 케이스 생성 연구는 테스트 커버리지 분야에서 본 연구와 차이를 보인다. [14]에서는 시스템의 변수 값의 조합을 가지고 시스템이 요구사항에 만족하는 정도를 수치화 시킨다. 요구사항 만족도를 일정 수준 이하로 떨어뜨리는 입력을 판별하는 것을 테스트 타당성이라고 하며 이에 따라 테스트 케이스를 생성한다. [19]에서는 star discrepancy[27]를 이용하여 시스템의 상태들을 최고로 커버할 수 있는 테스트 케이스를 찾는 것을 테스트 커버리지로 삼고 있다. 본 연구에서는 Stateflow 모델의 내부 상태에 초점을 맞춰서 테스트 하고자 하는 시스템의 내부 상태들을 정의하고 시스템의 초기 상태에서 목표 상태들까지 도달하는 테스트 케이스 생성을 목표로 한다.

테스트 커버리지 분야 외에도 RRT의 거리 측정 함수에서 차이를 보인다. 선행 연구[14-19]에서는 RRT 노드 간의 거리를 측정하기 위해서 시스템의 모든 변수를 사용하지만, 본 연구에서는 RRT 공간을 Stateflow의 상태 별로 분할하여, 거리 측정 시, 각 상태에서 참조되는 변수만을 사용함으로써 RRT 공간의 차원을 감소시켰다.

3. RRT 확장을 통한 테스트 케이스 생성 기법

테스트 케이스 생성을 위한 기본적인 프레임워크는 Fig. 2와 같다. 테스트 목표 생성기는 테스트 수행을 통해 만족하기 위한 테스트 대상을 생성한다. 생성된 테스트 대상은 RRT 확장의 목표가 된다. 테스트 케이스 생성기는 SL/SF 시뮬레이터를 이용하여 RRT 확장을 하며, RRT가 테스트 목표에 도달하였을 경우 테스트 케이스를 생성한다.

RRT 확장을 이용하여 테스트 케이스 생성을 하기 위해서는 먼저 SL/SF 모델에 대한 RRT 노드와, RRT 공간, 거리 함수가 정의 되야 한다. 그 후 테스트 케이스 생성을 위한 RRT 확장 알고리즘을 제안한다.

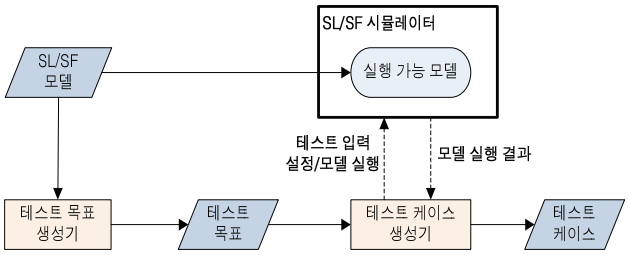

Fig. 2. Test case generation framework

3.1 RRT 공간의 정의

Stateflow 모델의 상태 s에서 다른 상태로의 이동 가능 여부는 상태 s에서 나가는 전이에 사용된 변수에 의해서 결정된다. 따라서 RRT 공간을 모델에 포함된 모든 변수를 이용하기보다는 상태 s에서 나가는 전이에 사용된 변수만 을 이용하여 구축하면 RRT 공간의 차원을 줄이면서 모델을 동작시킬 수 있다. Stateflow 모델이 n개의 상태 $S=\{s_1, s_2, \cdots, s_n\}$로 이루어져 있고 m개의 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 활성화 되어 있으면 a_i는 1, 그렇지 않으면 0으로 정의된다.

정의 1. $C(s_i)=\{c_1, c_2, \cdots, c_p\}$는 상태 s_i를 시작으로 하는 전이들의 조건에 포함되어 있는 변수 집합이다.
정의 2. 상태 s_i의 RRT 노드는 $C(s_i)$와 A 값의 조합이다.
정의 3. 상태 s_i의 RRT 공간 RSs_i는 상태 s_i의 존재할 수 있는 모든 RRT 노드의 집합이다.

$C(S_i)$의 원소의 개수가 p라면 S_i에 대한 RRT 공간의 차원은 모델의 변수의 개수 m보다 작은 p가 된다. Fig. 1의 모델을 이용하여 정의의 각 요소를 설명하면

$S=\{$"steady_state","upshifting","downshifting"$\}$

$V=\{$"speed","up_th","down_th",time","gear_direction"$\}$,

$C(S_{\text{"steady\_state"}})=\{$"speed","up_th","down_th"$\}$,

$C(S_{\text{"upshifting"}})=\{$"speed","up_th","time"$\}$,

$C(S_{\text{"downshifting"}})=\{$"speed","down_th","time"$\}$이 된다.

Stateflow 모델의 변수는 불리언, 정수, 실수 형을 가지며, 상태의 활성화 정보는 불리언 형이다. RRT 노드 간의 거리를 구하기 위해서는 다른 타입의 값 복수 개로 구성된 노드의 거리를 구할 수 있어야 한다. 이를 위해 제안된 기법에서는 Gower's General Similarity Coefficient[28]를 거리 측정 방법으로 이용한다. [28]에 따르면 유사도는 0에서 1사이의 값으로 이루어진다. 유사도가 1이라는 것은 두 RRT 상태가 동일하다는 것을 의미하고 0은 완전히 다르다는 것을 의미한다.

상태 s_i의 RRT 공간 RSs_i의 거리 함수는 정의 4이다. c_{kx}와 c_{ky}는 각각 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.

정의 4. 상태 S_i의 RRT 공간 RSs_i 안에 존재하는 두 RRT노드 nd_x, nd_y 사이의 거리 $d_{si}(nd_x, nd_y)$는 다음과 같다.

$$d_{S_i}(nd_x, nd_y) = \frac{\sum_{K=1}^{n(C(s_i))} ds_k(c_{kx}, c_{ky})}{n}$$

$$\begin{cases} ds_k(c_{kx}, c_{ky}) = \begin{cases} 0 & if \quad c_{kx} = c_{ky} \\ 1 & if \quad c_{kx} \neq c_{ky} \end{cases} & c_k\text{가 불리언 형인 경우} \\ ds_k(c_{kx}, c_{ky}) = \dfrac{|c_{kx} - c_{ky}|}{|c_{k\max} - c_{k\min}|} & c_k\text{가 정수나 실수인 경우} \end{cases}$$

3.2 RRT 확장 알고리즘

제안하는 Table 2의 알고리즘은 2.1절의 RRT 확장 알고리즘을 테스트 케이스 생성을 위해 변형한 알고리즘이다. Table 2의 알고리즘은 Table 1의 알고리즘의 입력 파라메터에서 추가로 TG를 사용한다. TG는 시스템이 임의의 테스트 목적을 가지고 테스트 되기 위해서 도달해야 하는 시스템의 내부 상태 RRT 노드들의 집합이다. 3.1절에서 RRT 공간은 Stateflow의 각 상태 별로 존재하기 때문에 각 상태 별 트리가 따로 존재하게 된다. T_{init}은 Stateflow의 초기 상태의 트리이며, *MultipleRRT*는 각 상태의 트리를 저장하는 리스트이다. 먼저 *SelectTree()*에서 *MultipleRRT*에 존재하는 상태 별 트리 중, 확장을 할 상태 S의 트리 T_S를 임의로 선택한다. 다음 T_S에 대해 RRT 확장을 1회 한 후, 추가된 RRT 노드 nd_{new}가 목표 RRT노드일 경우, 목표까지의 경로가 찾아진 것이므로, TG에서 제거한다. 정의 2에 따라, RRT 노드는 Stateflow의 활성화 된 상태 정보를 가지고 있다. 활성화 된 각 상태 *activeState*에 대해서, 트리가 존재하

Table 2. Modified RRT extension algorithm for test case generation

```
ModfiedExtends (nd_init, K, Δt, TG)
T_init.init(nd_init)
MultipleRRT.add(T_init);
for k=1 to K do
  if(TG != ∅) return MultipleRRT;
  (S, T_s)=SelectTree(MultipleRRT)
  nd_rand ← random_node(S);
  nd_near ← nearest_ node(nd_rand, T_s,S);
  v ← select_input(nd_rand, nd_near);
  nd_new ← new_ node(nd_near, v, Δt);
  if(nd_new ∈ TG )
     TG = TG − nd_new
  for activeState in nd_new.A
   if(!(T_activeState ∈ MultipleRRT))
     T_activeState.init(nd_new)
     MultipleRRT.add(T_activeState)
   else
     T_activeState.add_vertex(nd_new);
   T_s.add_edge(nd_near, nd_new, v);
return MultipleRRT
```

지 않으면 트리의 첫 RRT노드로써 nd_{new}를 추가하여 트리를 만들어 *MultipleRRT*에 추가한다. 트리가 존재할 경우 해당 트리에 nd_{new}가 추가 된다. 이 반복이 K회 수행되거나, *TG*의 RRT 노드들에 도달하는 모든 경로가 찾아지면 알고리즘이 종료된다.

트리를 확장하기 위해 입력을 모델에 적용시켜서 모델의 활성화 상태가 바뀐다면, 즉 *activeState*와 S가 다른 상태라면, T_s.*add_edge*(nd_{near}, *ndnew*, *v*)에 의해, 다른 상태 트리 간의 연결이 만들어지게 된다. *MultipleRRT*에 존재하는 각 상태의 트리들은 서로 분리되어 있는 것이 아니라 관련이 있는 상태의 전이에 따라 트리 끼리 연결이 된다. 모델의 모든 상태는 모델의 오류가 없다면 초기상태 *init*에서부터 출발하여 일련의 상태를 거쳐 도달 할 수 있으므로, 모델의 각 상태의 트리가 하위 트리가 되어 합쳐져서 모델 전체의 트리가 만들어질 수 있다. 만약 초기 상태 *init*에서 S_1을 거쳐 S_2로 전이되는 모델이라면, 트리는 $T_{init} \rightarrow T_{S1} \rightarrow T_{S2}$ 순서로 연결 되어 전체 모델의 트리가 구축된다.

RRT의 확장은 트리 T_{init}의 RRT 노드 nd_{init}에서 시작되며 각 상태의 트리는 부모를 거슬러 올라가면 T_{init}에 도달할 수 밖에 없으므로, nd_{init}에서 각 트리의 모든 노드로 도달할 수 있는 경로가 반드시 존재한다. *TG*의 원소 RRT 노드 *tg*에 대해서 *tg*가 트리에 포함이 되어있다면 *tg*의 테스트 케이스는 nd_{init}으로부터 *tg*까지의 경로를 구성하는 입력 *v* 의 순열이 된다.

3.3 예제

3.2절에서 제시된 테스트 케이스 생성 알고리즘을 그림 1의 예제 모델을 이용하여 설명한다. 도달하고자 하는 테스트 목표가 상태 *"upshifting"*이 활성화 되고, 변수 *"speed"*의 값이 30 이상이고, *"up_th"*의 값이 20 이상 *"time"*의 값이 1 이상인 RRT 노드라고 할 때, 이 RRT 노드를 찾아가는 과정을 알고리즘 2를 이용하여 설명하겠다. 예시 모델의 실행 주기가 10ms이기 때문에 알고리즘의 Δt 는 10ms로 한다.

먼저 모델의 초기 상태, *"steady_state"*가 활성화 된 상태에서 모든 변수가 초기값 0을 가지는 nd_{init}이 $T_{\text{"steady_state"}}$의 첫 RRT 노드로 추가된다. *"steady_state"*의 RRT 공간은 *"speed"*, *"up_th"*, *"down_th"*로 구성된다는 것을 염두 해두자. 확장을 위해 선택할 수 있는 모델의 상태가 *"steady_state"* 밖에 없으므로 $T_{\text{"steady_state"}}$를 확장한다. 입력을 *"speed"*=23, *"up_th"*=20, *"down_th"*=0으로 선택하여 시스템을 Δt 만큼 진행시키면 모델의 상태가 *"steady_state"*에서 *"upshifting"*으로 변경된다. 따라서 RRT 공간도 *"upshifting"* 상태에 따라 *"speed"*, *"up_th"*, *"time"*의 구성으로 바뀐다. 변수가 *"speed"*=23, *"up_th"*=20, *"time"*=0 이고, 상태 *"upshifting"*이 활성화 된 RRT 노드 $nd_{new,\ 1}$이 생성되어 새로운 트리 $T_{\text{"upshifting"}}$ 의 루트 노드가 된다. 트리의 확장 후 상태가 바뀌었으므로, nd_{near}는 nd_{init}과 연결되어, $T_{\text{"steady_state"}}$와 $T_{\text{"upshifting"}}$사이의 연결을 제공한다.

다음 확장을 위해 *SelectTree()*에 의해 선택된 트리가 $T_{\text{"upshifting"}}$라 하자. 입력을 *"speed"*=33, *"up_th"*=20, *"down_th"*=0으로 선택하여 시스템을 Δt 만큼 진행시키면 변수가 *"speed"*=33, *"up_th"*=20, *"time"*=1 이고, 상태 *"upshifting"*이 활성화 된 RRT 노드 $nd_{new,\ 2}$가 생성되어 $T_{\text{"upshifting"}}$에 추가된다. 이 때 $nd_{new,\ 2}$는 테스트 목표이므로 테스트 목표에 도달하는 경로 $nd_{init} \rightarrow nd_{new,\ 1} \rightarrow nd_{new,\ 2}$가 얻어진다. 시스템이 노드 간 이동하는데 소요되는 시간은 Δt 즉, 10ms이며 테스트 목표로 이동하기 위한 테스트 케이스는 Table 3과 같다.

Table 3. Test case table for example model

Time(ms)	*Speed*	*up_th*	*down_th*
0	23	20	0
10	33	20	0

4. 실 험

제안된 알고리즘을 평가하기 위해서 Fig. 1의 모델과, 추가로 자동차에 사용되는 다른 시스템들의 Stateflow 모델을 대상으로 실험을 수행한다. 실험은 제안된 알고리즘과 [17]의 일반적인 RRT 확장 알고리즘으로 RRT 확장에 소요되는 비용을 시간과 메모리 측면에서 비교한다. RRT 확장을 각 1000, 5000, 10000번 실행하면서 두 알고리즘의 성능을 비교하였으며, 실험 대상 컴퓨터의 CPU는 Intel Core i5 2.67GHz이며 RAM은 4.00GB이다.

Table 5는 Fig. 1의 모델을 대상으로 실험을 수행한 결과이다. 트리 확장에 소요된 메모리는 RRT 노드들이 점유하고 있는 메모리이며, 메모리가 증가한다는 것은 트리가 RRT 공간에서 확장되어서 RRT 노드의 개수가 증가했다는 것을 의미한다. 트리 확장에 소요된 시간은 주로 트리의 확장을 위해 RRT 공간에 존재하는 각 RRT 노드들의 거리를 계산하고 가장 가까운 RRT 노드들을 판단하는 과정과 Stateflow 모델을 시뮬레이션 하기 위해 소비된다. 트리 확장의 횟수에 따라 메모리는 거의 선형적으로 증가하는 추세를 보인다. 정확히 선형적으로 증가하지 않는 이유는 새롭게 확장된 RRT 노드가 이미 RRT 공간에 존재하는 노드일 경우 중복해서 RRT 공간에 추가하지 않기 때문이다. 반면 매 확장할 때마다 수행해야 되는 RRT 노드 간의 거리 계산의 횟수는 RRT 공간의 노드의 개수가 증가함에 따라 비례하여 증가하기 때문에 트리 확장에 소요 되는 시간은 트리 확장 횟수의 제승에 비례하여 증가하는 추세를 보인다. 트리 확장의 횟수가 늘어남에 따라 기존 알고리즘과 제안된 알고리즘의 성능 차이가 크게 벌어지는 것을 확인 할 수 있다. 기존 알고리즘은 단일 RRT 공간을 사용함으로써, *"speed"*,*"up_th"*,*"down_th"*, *"time"*,*"gear_direction"*, 다섯 개의 모델 변수로 RRT 공간이 형성된다. 제안된 알고리즘은 3.1절에서 설명하였듯이 각 상태에 따라서 세 개의 모델 변수로 RRT 공간들이 형성된다. 이 차이가 알고리즘의 성능 차이를 보이게 된다. 트리 확장 비용의 메모리 측면에서 살펴보면 변수 개수의 감소는 RRT 공간에 생성되는 RRT

노드 개수의 감소를 이끈다. 또한 RRT 공간에 포함되는 각 노드의 메모리 크기가 작아진다. 종합적으로 메모리 소모가 개선된 알고리즘이 기존 알고리즘보다 낮아진다. 성능 차이를 시간 측면에서 살펴보면 생성되는 RRT 노드 개수가 감소하는 것은 매번 트리를 확장 할 때마다 각 노드 간의 거리를 비교 해야 되는 횟수가 감소하는 것을 의미한다. 또한 노드의 변수 개수가 감소함으로써, 두 노드 간의 거리를 계산 할 때 각 변수 별 계산해야 되는 횟수가 감소된다. 두 알고리즘의 소요 시간 차이는 이와 같은 원인으로 발생한다. 두 알고리즘의 테스트 케이스 생성 커버리지 율을 상태 커버리지, 전이 커버리지, MC/DC(Modified Condition/Decision Coverage)[29] 측면에서 비교해봤을 때 Table 5와 같이 거의 비슷한 수치를 보인다. 결과적으로 개선된 알고리즘은 기존 알고리즘에 비해 비슷한 커버리지 율을 유지하면서 높은 성능을 보인다.

예제 Stateflow 모델 외에 좀 더 복잡한 Stateflow 모델에 대하여 알고리즘 평가 실험을 수행한다. 서리 제거 장치는 차량의 시동 신호와 서리 제거 장치 작동 스위치를 입력받아 서리 제가 장치를 동작시킬 지 결정하는 모델이며, 차량 경보 장치는 차량 시동 상태, 차량 도어 개폐 상태, 차속 등을 참고하여 현재 차량 상태에 이상이 있음을 알려주는 모델이다. 실험의 대상이 되는 모델의 기본적인 통계정보는 Table 4와 같다.

두 Stateflow 모델에 대하여 앞에서 수행하였던 실험과 같은 방식으로 실험을 수행하였다. 각 모델의 실험 결과는 Table 6-7에 나와있다. 실험 대상 모델의 규모가 클수록 테스트 케이스 생성에 필요한 RRT 확장에 소요되는 메모리와 시간의 격차가 더 크게 벌어지는 것을 확인 할 수 있다. 실험 결과들이 대부분 같은 추세를 보이는 반면 Table 6의 서리 제거 장치 모델에 대한 메모리 소요 비교는 다른 추세를 보이는데, 이는 서리 제거 장치 모델의 RRT 공간이 다른 모델에 비해 작기 때문에 RRT 확장에 따른 RRT 노드의 포화 상태가 되었기 때문이다. RRT 공간이 포화 된 상태에서도 기존 알고리즘이 개선된 알고리즘에 비해 2.5배 정도 많은 메모리를 소비하는 것을 확인 할 수 있다.

Table 4. Statistics of more coplex models for expements

Model	# of states	# of transitions	# of variables
Defroster	9	9	6
Car alarm	20	71	55

Table 5. Experimental results of gear selection model

# of RRT extensions		1000	5000	10000
Time (sec)	Typical	8.10	174.23	1641.2
	Proposed	2.53	24.66	1878.87
Memory (byte)	Typical	48672	242592	461688
	Proposed	8752	43232	87232
Coverage Ratio	Typical	95.65%	95.65%	95.65%
	Proposed	91.30%	95.65%	95.65%

Table 6. Experimental results of defroster model

# of RRT extensions		1000	5000	10000
Time(sec)	Typical	18	107.33	400.09
	Proposed	13.29	71.92	173.58
Memory (byte)	Typical	2496	2496	2556
	Proposed	960	1008	1072
Coverage Ratio	Typical	96.77%	96.77%	96.77%
	Proposed	93.54%	96.77%	96.77%

Table 7. Experimental results of car alarm model

# of RRT extensions		1000	5000	10000
Time(sec)	Typical	102.33	4361.94	11699.15
	Proposed	30.32	371.18	1052.45
Memory (byte)	Typical	326400	1619200	3235200
	Proposed	17800	74920	182520
Coverage Ratio	Typical	67.60%	67.60%	67.60%
	Proposed	67.60%	67.60%	67.60%

5. 결 론

본 논문에서는 Stateflow 기반 모델의 테스트 케이스 생성 기법을 제안하였다. 제안된 기법에서는 동작 계획 수립에 사용되는 RRT 알고리즘을 테스트 케이스 생성 알고리즘에 접목시켜 테스트 케이스를 생성하였다. 이는 모델 기반 블랙박스 테스팅에서 발생 하는 도달 가능 문제를 해결 해 줄 수 있다. 제안된 기법은 일반적인 RRT를 접목한 테스트 케이스 생성 알고리즘과는 다른 변형된 RRT 기법을 사용하였다. 첫 번째로 단일 RRT 공간이 아닌 Stateflow의 각 상태 별로 분할된 RRT 공간을 이용하였다. 두 번째로, 각 상태의 RRT 공간은 시스템의 모든 변수로 구성되는 것이 아닌 Stateflow의 각 상태와 관련된 변수만으로 구성된다. 이 기법은 시스템의 단일 공간이 Stateflow 상태를 중심으로 분할되기 때문에 테스트 케이스 생성 시 기존의 RRT 기반 테스트 케이스 생성 알고리즘 보다 실제로 적용되는 RRT 공간의 차원이 낮아진다. 이에 따라 우리는 두 가지 측면에서의 장점을 얻을 수 있었다. 먼저 각 RRT 공간의 노드가 저장 해야 되는 변수 값의 개수가 작아진다. 이는 테스트 케이스 생성시 점유되는 메모리가 감소하는 것을 의미한다. 그리고 RRT 확장 시 각 RRT 노드 사이의 거리를 계산 하기 위한 시간이 감소된다. 이는 테스트 케이스 생성시 소요되는 시간이 감소하는 것을 의미한다. 우리는 실험을 통해 제안된 알고리즘이 기존의 RRT 기반 테스트 케이스 생성 알고리즘보다 메모리 점유율, 생성 소요 시간 측면에서 좋은 결과를 보여주는 것을 확인 할 수 있었으며, 특히 실험 대상 Stateflow 모델의 규모가 클수록 그 차이는 더욱 커지는 것을 알 수 있다.

참 고 문 헌

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

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

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

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

[5] 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.

[6] 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.

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

[8] J. S. Oh, M. Harman, S. Yoo, "Transition coverage Testing for Simulink/Stateflow Models Using Messy Genetic Algorithms" in Proceedings of the 13th annual conference on Genetic and evolutionary computation pp.1851-1858, 2011.

[9] P. Peranandam, S. Raviram, M. Satpathy, "An Integrated Test Generation Tool for Enhanced Coverage of Simulink/ Stateflow Models", Design, Automation & Test in Europe Conference & Exhibition, pp.308-311, 2012.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] T. Nahhal, T. Dang, "Test Generation For Analog And Mixed-Signal Circuits using Hybrid system models" in International Journal of VLSI Design & Communication Systems Vol.2, Issue.3, pp.21-38, 2011.

[20] 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.

[21] MATLAB Simulink, Modeling an Automatic Transmission Controller, http://www.mathworks.co.kr/products/simulink/examples.html?file=/products/demos/shipping/simulink/sldemo_autotrans.html

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

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

[24] 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

[25] 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.

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

[27] "Star Discrepancy." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/StarDiscrepancy.html

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

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

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 정보 공학부 (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 시스템등