KIPS Trans. Softw. and Data Eng.
Vol.8, No.6 pp.235~242 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2019.8.6.235

Extracting Reusable Test Cases from Modified MATLAB Simulink Model

Park Geon Gu[†] · Han Hye Jin[††] · Chung Ki Hyun[†††] · Choi Kyung Hee[††††]

ABSTRACT

This paper proposes a reusable test case extraction technique for modified MATLAB Simulink/Stateflow (SL/SF) model. Creating test cases for complicated SL/SF model like ECU(Electrical Control Unit) of automotive, requires a lot of time and effort. An intuitive way to reduce to create new test cases whenever the model changes, is to reuse some test cases which have been generated for the original model. In this paper, we propose a method to define reusable test cases in SL/SF after defining model behavior and judging model equality by test cases. The proposed technique is evaluated using a commercial automotive controller model.

Keywords : Test Case Reuse, Simulink/Stateflow, Model Based Test

변경된 MATLAB Simulink 모델로부터 재사용 가능 테스트 케이스 도출

박 건 구[†] · 한 혜 진[††] · 정 기 현[†††] · 최 경 희[††††]

요 약

본 논문에서는 제어기 기능이 표현 된 변경된 MATLAB Simulink/Stateflow(SL/SF) 모델의 재사용 가능한 테스트 케이스 도출 기법을 제안한다. 자동차의 ECU(Electrical Control Unit)와 같이 복잡한 SL/SF 모델의 테스트 케이스를 작성하는데 많은 시간과 노력이 필요하다. 모델이 수정 될 때마다 새로 만들어낼 테스트 케이스를 줄이기 위한 직관적인 방법은 수정 전 모델에서 생성한 테스트 케이스 중 일부를 재사용하는 것이다. 본 논문에서는 모델 행동을 정의하고 테스트 케이스 별 모델 동등성을 판단하여 수정 후 SL/SF에 재사용 가능한 테스트 케이스를 도출하는 방법을 제안한다. 제안된 테스트 케이스 재사용 기법은 상용 자동차 제어기 모델을 이용하여 성능을 평가한다.

키워드 : 테스트 케이스 재사용, 시뮬링크/스테이트플로우, 모델 기반 테스트

1. 서 론

임베디드 시스템과 같이 소프트웨어가 포함된 다양한 시스템의 기능 발전은 소프트웨어를 통해서 이뤄지고, 소프트웨어가 복잡해지며 많은 오류를 포함할 가능성이 높아졌다. 이에 따라 소프트웨어 테스트에 더 많은 노력이 필요하다.

소프트웨어 테스트 방법은 크게 소스 코드를 중심으로 하는 화이트 박스 테스트와 요구사항 중심으로 하는 블랙박스 테스트로 나눌 수 있다[1]. 화이트 박스 테스트는 소프트웨어 개발 단계에서 소프트웨어 소스코드 오류를 분석하고 적절한 테스트 케이스를 만들어 소프트웨어를 실행시키며 수행하는 동적 테스트와 소스 코드에 오류가 발생할 수 있는 가능성이 높은 코드를 점검하는 정적 테스트로 나뉜다[2].

블랙박스 테스트는 개발 마지막 단계나 완제품 상태에서 개발된 소프트웨어가 요구사항을 정확히 수행하는 지를 점검하는 테스트이다. 임베디드 소프트웨어는 개발된 소프트웨어가 소프트웨어의 하드웨어를 적절히 동작 시키는 지를 점검하게 된다.

블랙박스 테스트는 combinatorial, 오류 주입(fault injection) 등 여러 가지 방법이 있다[3-5]. 모델 기반(model based) 테스트 방법도 그 중의 하나이다. 모델 기반 테스트를 위해 테스트 대상 시스템의 요구사항을 모델화하는 것이 선행된다. 모델 기반 테스트는 작성된 모델을 다양한 방법으로 사용한다. 작성된 모델을 입출력 장치처럼 사용하거나, 작성된 모델과 실제 시스템에 테스트 케이스를 같이 주입하여 그 결과를 비교하는 방법, 그리고 작성된 모델로부터 테스트 케이스를 생성하여 테스트에 사용하는 방법 등이 있다.

※ 본 연구는 국방과학연구소의 지원(계약번호:UD170016DD)으로 수행되었음.
† 준 회 원 : 아주대학교 전자공학과 석사과정
†† 준 회 원 : 아주대학교 컴퓨터공학과 석사과정
††† 정 회 원 : 아주대학교 전자공학과 교수
†††† 정 회 원 : 아주대학교 컴퓨터공학과 교수
Manuscript Received : November 30, 2018
Accepted : December 28, 2018
* Corresponding Author : Chung Ki Hyun(khchung@ajou.ac.kr)

모델의 또 다른 이용 방법은 개발하고자 하는 시스템의 기능을 사전에 파악하는 것이다. 구현하고자하는 기능을 모델화하고 모델이 정확히 동작하는지 확인하기 위해 테스트를 수행한다. 이 경우 모델의 정확한 테스트를 위해 양질의 테스트 케이스 생성이 필수적이다. 하지만 복잡한 모델의 경우 테스트 케이스 생성은 상당히 많은 시간과 노력이 소모되며 좋은 테스트 케이스를 생성하기 어렵다.

뿐만 아니라 소프트웨어 개발 과정이나 기능이 변경될 때, 모델의 수정은 빈번히 발생하며, 이러한 수정된 소프트웨어를 테스트하기 위한 테스트 케이스 생성에 추가의 시간과 노력이 필요하다. 이 때, 모델의 수정 전 구축한 테스트케이스 일부를 재사용 할 수 있다면 새로운 테스트 케이스를 구축하는 시간과 노력을 절감할 수 있다.

모델 작성 도구는 MATLAB Simulink/Stateflow, UML, SysML 등이 있다. 이 중 자동차, 항공, 전자 시스템 개발을 위한 요구사항은 대부분 MATLAB Simulink/Stateflow (SL/SF) 모델링을 통해 작성하고 있다. 본 논문에서는 시스템의 기능이 표현된 SL/SF 모델 변경 시 이전 테스트 케이스 중 재사용 할 수 있는 것들을 효과적으로 추출하여 사용하는 기법을 제안한다. 제안된 방법은 실제 자동차 제어기 모델에 적용하여 그 유효성을 검증한다.

본 논문의 구성은 다음과 같다. 2장은 모델 기반 테스트 및 관련 연구 현황을 설명하고 3장에서 Model equivalence라는 핵심 개념을 정의 한다. 4장은 시스템의 기능이 표현된 모델 수정 시 재사용 가능 테스트 케이스 추출 기법에 대해 설명한다. 5장에서는 제안된 기법의 타당성을 자동차 제어 모델을 통해 평가하며 6장에서는 결론을 기술한다.

2. 관련 연구

여러 테스트 환경에서 테스트 케이스 생성 부하를 줄이기 위해 많은 테스트 케이스 재사용 연구와 모델 equivalence 연구들이 진행되어 왔다. [6]의 저자는 수많은 Test Case를 Web Ontology Language로 표현하고 재사용 가능한 테스트 케이스의 공통적인 특성을 정의하고 도출했다. 이후 Ontology 구축, 테스트 케이스 재사용 과정을 설명한다. [7]의 저자는 테스트 사례의 재사용 가능성, 재 작성 패턴 및 테스트 커버리지 간의 관계에 중점을 두고 테스트 커버리지 기준으로 테스트 재사용성의 일반적인 평가 방법을 제시한다. 제시한 재 작성 패턴을 적용한 응용 프로그램 3가지로 실험을 진행했으며 프로그램 수정 방법과 테스트 커버리지가 테스트 케이스 재사용성에 미치는 영향을 검증했다. [8]의 저자는 수정 전, 후 버전 프로그램의 입력 도메인을 분석하여 수정 전 테스트 케이스를 최대한 재사용 할 수 있는 회귀 테스트 방법론을 연구했다. 제안된 방법론은 전통적인 도메인 기반 테스트 전략과 달리 프로그램의 사양과 테스트 기준을 고려한다. 이들 연구는 코드 및 프로그램 수정 시 테스트 케이스 재사용 연구이므로 모델 기반 개발 방법에 적용하기 어렵다. 모델을 테스트하기 위한 테스트 케이스 재사용 기법에 대한 연구는 찾아보기 힘들다.

수정된 모델의 테스트 케이스을 위해서는 두 모델의 차이를 찾아내고 이 부분에 대한 테스트 케이스 생성이 필요하다. 두 모델의 차이를 찾아내기 위해서는 두 모델이 같은 지를 알아낼 수 있어야 한다. 이와 관련된 모델 equivalence 연구가 진행되었다. 수정된 모델과 원래의 모델이 동등한 기능을 하는지 판별하는 것이 검증의 중요한 과정이다. [9]의 저자는 Simulink 모델 기반 개발 시 모델의 구성 요소 간 equivalence를 증명하기 위한 검증 방법을 연구했다. SL/SF 모델 equivalence에 대한 학계 연구는 찾아보기 힘들고 두 SL/SF 모델의 차이점을 비교하는 상용 도구인 Simdiff라는 도구가 많이 활용되고 있다[10].

Simdiff는 서로 다른 두 SL/SF 모델의 차이를 추출해주는 도구다. SL내 다양한 Block, In/Output, 연결선의 추가, 삭제, 변경을 추출하고 SF내 Transition, State, Junction의 추가, 삭제, 변경을 도출한다. Simdiff로부터 얻은 정보를 사용하기엔 몇 가지 문제점이 있다. 첫째, 얻은 정보를 사용하기 힘들다. 도출 결과가 HTML파일로 생성되어 데이터를 사용하기 위해선 HTML을 분석한 후 원하는 정보를 얻어오는 과정이 필요하다. 둘째, 추출해주는 정보 이외에 원하는 정보를 얻을 수 없다. 이 문제를 해결하기 위해 모델 구조 비교기를 구성하여 두 SL/SF 모델을 분석하고 원하는 정보들을 얻어 사용할 수 있는 방법을 고안했다.

3. Model Equivalence

변경된 모델에 원래 모델의 테스트 케이스를 재사용할 수 있는지 파악하기 위해 변경된 모델과 원래 모델의 차이점을 알아야한다. 그리고 그 차이가 특정 테스트 케이스 관점에서 변경된 모델에 영향을 미치는지 파악해야 한다.

변경 전후의 동일성에 관한 연구는 소프트웨어 분야에서 많이 진행되었다. 임베디드 소프트웨어에서 메모리 계층 접근은 실행 시간과 전력소모 증가에 영향을 준다. 소프트웨어 개발 과정에서 이를 줄이기 위해 global loop transformations 및 global data-flow transformations 두 가지 방법을 코드 최적화에 많이 사용한다[11]. 이러한 코드 최적화 과정에서 코드의 변형이 발생한다. 기능이 변경되지 않는 소프트웨어의 변경은 변경된 소프트웨어의 기능이 기존 프로그램의 기능과 같다는 것을 전제로 하여야 한다. 이때 변경 전, 후 소프트웨어가 동등한 행동을 보이면 두 소프트웨어는 Code equivalence를 가진다고 말한다[12]. Code equivalence의 행동이란 두 프로그램에 임의의 입력을 주었을 때 나오는 출력이 같음을 말한다. 즉, 변경 전, 후 소프트웨어에 같은 입력을 주었을 때 같은 출력이 나오면, Code equivalence의 행동이 같다고 말하고, 두 소프트웨어는 해당 테스트 입력에 대해 equivalence한 상태이다.

본 논문에서는 해당 개념을 통해 SL/SF로 작성된 모델의 Model equivalence와 Model equivalence의 행동을 정의하고 변경 된 모델에 대한 테스트 케이스 재사용성을 평가한다. Model equivalence는 수정 전, 후 모델이 동일한 입력에 대해 같은 행동을 수행하는지 확인하는 것이다. 소프트웨어 코드 수

정 전, 후 행동이 같은지 확인하는 code equivalence 개념을 응용한 것이지만 Model equivalence의 행동은 code equivalence의 행동과는 몇 가지 측면에서 다르다. Code equivalence의 행동인 입력에 대한 출력과 달리 모델의 특성을 고려하여 몇 가지 정보를 더해 모델 행동 정보를 정의한다. 모델 행동이란 입력을 적용한 모델 시뮬레이션 과정 중 관찰되는 모델의 상태 변화이다. 모델 행동은 크게 단순 행동과 복합 행동으로 구분한다. 단순 행동의 요소는 1) Stateflow 모델의 active state와 2) 모델 출력으로 정의하며, 복합 행동은 단순 행동에 더해 3) active transition, 4) active junction 및 5) 내부 변수로 구성된다. 수정 전, 후 모델에 같은 입력을 인가하여 행동에 정의된 요소들의 결과가 동등하다면 같은 행동을 보인다고 판단하고 해당 입력에 대해 Model equivalence가 성립한다고 평가한다. 만약 각 요소들의 결과가 다르면 해당 입력에 대해 같은 행동을 수행하지 못한다고 판단하고 해당 테스트케이스에 대해 Model equivalence가 성립하지 않는다고 평가한다.

4. SL/SF 모델 수정 시 재사용 가능 테스트 케이스 추출

자동차의 다양한 제어기 개발 시 SL/SF 모델을 많이 사용한다. 개발된 모델은 개발자의 설계 사양을 만족하는 지 검사할 때[13]나, 개발된 제어기의 테스트 케이스 작성[14] 또는 자동 코딩[15] 등에 사용된다. 이 때 개발된 모델이 정확한지를 검사하는 것이 선행되어야 한다. 개발된 모델이 설계자가 원하는 사양을 제대로 반영하고 있는 지 확인하기 위해 테스트 케이스를 이용하여 검사한다. SL/SF 모델의 테스트 케이스 생성 방법으로 다양한 자동 생성 방법[16-18]이 소개되고 있다. 하지만 자동 생성 방법은 시간과 노력은 적게 소모되지만 테스트 케이스의 질이 떨어진다는 단점이 있다. 따라서 수동 생성 방법은 엔지니어의 시간과 노력이 많이 소모되나 테스트 케이스의 질을 높일 수 있다는 점에서, 자동차 제어기 모델과 같이 높은 수준의 신뢰도를 요구하는 경우에는 수동 테스트케이스를 많이 사용한다.

그런데 제어 모델에 약간의 수정이 발생해도 제어 모델에 대한 테스트케이스를 다시 생성해야하는 문제가 발생한다. 개발 단계에서 제어기의 요구사항이 빈번하게 발생한다는 점을 고려하면 테스트 케이스 재생성에 필요한 엔지니어의 시간과 노력이 상당히 소모된다. 만약 이전에 만들어 놓은 테스트 케이스의 일부를 수정 후 모델에 재사용 할 수 있다면 시간과 노력의 소모를 줄일 수 있다. 본 논문에서는 수정 전 모델을 위해 생성한 테스트 케이스 중에서 수정된 모델 검증에 재사용 가능한 테스트 케이스 추출할 수 있는 방법을 제안한다.

4.1 모델 테스트 케이스 재사용

SL/SF 모델이 변경 되었을 때 수정 전 모델의 테스트 케이스를 수정 후 모델에 재사용할 수 있다면 그 테스트 케이스는 재사용 가능하다고 말한다. 이때 재사용 가능성을 판단하는 것을 테스트케이스 재사용성 평가라고 한다. 테스트케이스 재사용성 평가는 Model equivalence 성립 여부와 모델 구조 비교를 통해 진행한다. Model equivalence 성립 여부는 명확하게 특정 테스트 케이스에 대한 모델 행동이 같으면 해당 테스트 케이스는 재사용할 수 있다고 판단하기 위함이다. 모델 구조를 비교를 하는 이유는 Model equivalence가 다르더라도 테스트케이스 재사용 가능한 경우를 파악하기 위함이다. 예를 들어 Fig. 1의 모델 수정 전(M1), 후(M2) 예를 살펴보면 State A에서 State B로 향하는 두 개의 Transition 조건 CT1과 CT2가 하나의 Transition 조건으로 합쳐지며 [CT1 || CT2]로 통합된 경우가 있다. 떨어진 두 Transition을 OR연산자로 묶었기 때문에 두 모델은 같은 모델이라고 할 수 있다. 그런데 Transition 두 개가 하나로 합쳐졌으므로 두 모델의 행동이 다르다고 평가되어 Model equivalence가 성립하지 않는다고 평가된다. 하지만 구조적으로 다를 뿐 실제 같은 내용이기 때문에 모델 구조 비교를 통해 해당되는 부분을 통과하는 테스트케이스를 재사용 할 수 있다고 평가할 수 있다.

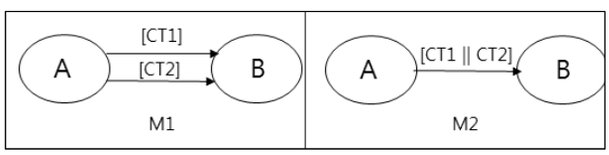

Fig. 1. Example of Model before Modification(M1), Modified Model(M2)

테스트케이스 재사용 평가 기준은 재사용 가능, 재사용 불가로 나뉜다. 재사용 가능은 Model equivalence가 동일하거나 Model equivalence가 동일하지 않더라도 해당 부분이 모델 구조 비교 상 동등해서 재사용 가능한 경우를 의미한다. 재사용 불가는 모델 equivalence가 동일하지 않고 해당 부분이 모델 구조 비교 상 동등하지 않은 경우를 의미한다.

4.2 테스트 케이스 재사용 절차

테스트 케이스 재사용 절차는 1) 모델 행동 생성, 2) 구조 비교, 3) 모델 행동 비교 그리고 4) 테스트케이스 재사용성 평가의 4단계로 진행된다.

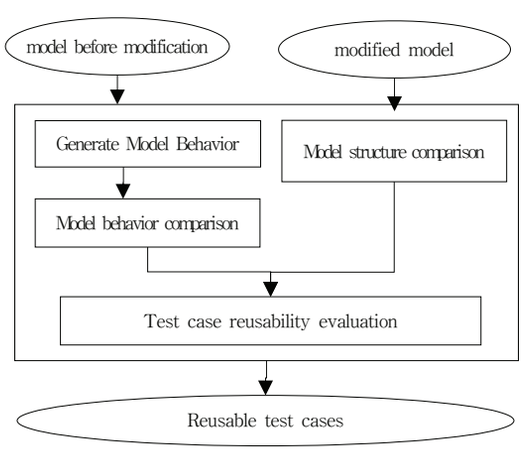

Fig. 2. Test Case Reuse Procedure

모델 행동 생성 단계에서는 같은 테스트 케이스로 수정 전, 후 모델의 시뮬레이션을 수행하여 모델 행동을 추출한다. 모델 행동 비교 단계에서는 모델 행동 생성 단계에서 추출된 결과들의 비교를 수행한다. 구조 비교 단계에서는 수정 전, 후 모델을 구조적으로 비교한 결과를 추출하며 테스트케이스 재사용성 평가 단계에서 앞선 두 결과를 토대로 재사용 가능성을 평가한다.

다음은 재사용성 평가 과정을 예시를 통해 설명한다. 테스트 케이스 재사용성 평가 과정을 설명하기 위해 수정 전, 후 예시 모델을 사용한다.

1) 예시 모델

수정 전 모델은 다음 Fig. 3과 같다. 최 상위 단계에 Subsystem이 존재하며 그 안에 Chart가 존재한다. Chart의 구성은 State안에 Sub State 3개가 존재하며 각 Sub State는 Transition으로 연결되어 있다.

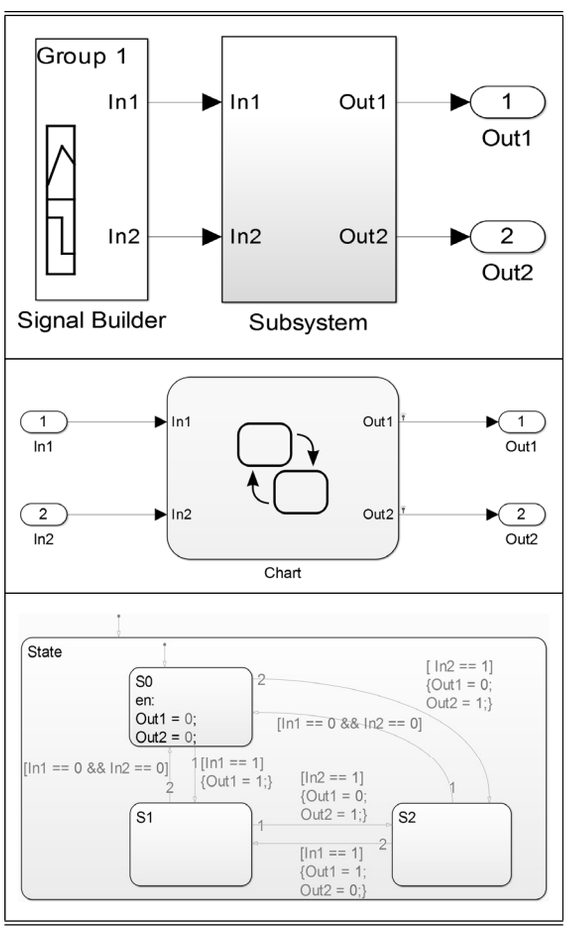

Fig. 3. Model before Modification

수정 후 모델은 다음 Fig. 4와 같다. Stateflow 내부의 S0에서 S1으로 가는 Transiton과 S0에서 S2로 가는 Transiton의 조건이 수정 되었고 S2에서 S1으로 가는 Transition이 삭제되었다.

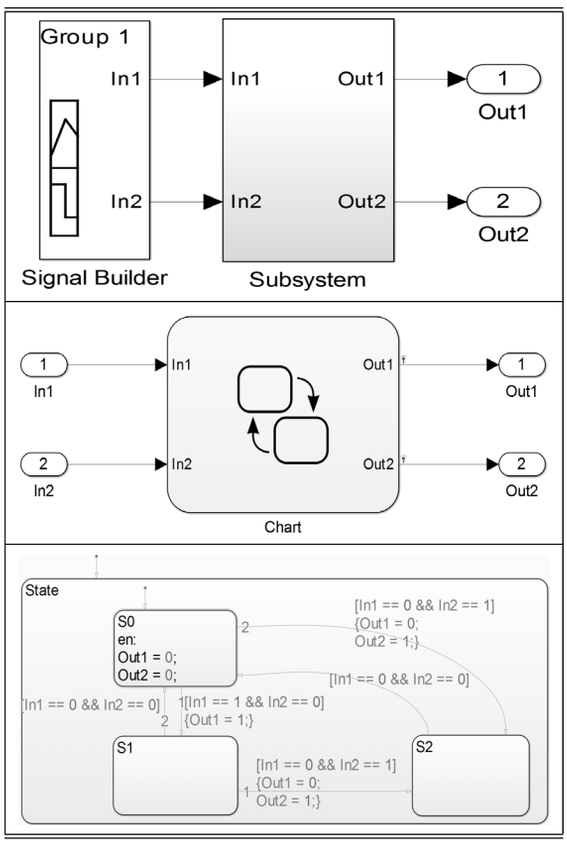

Fig. 4. Modified Model

2) 행동 생성

행동 생성 단계는 테스트 케이스 별 모델 행동을 생성하는 과정이며 테스트 목적에 따라 State 정보 및 Transition 정보를 생성한다. Fig. 5에 나타난 모델 행동은 테스트케이스를 적용해서 수정 전, 후 모델을 시뮬레이션 하여 단위 시간 별 활성화 되는 stateflow의 State와 출력 신호이다. 수정 전 모델에 대해 다음과 같은 모델 행동을 얻을 수 있다. 맨 처음 두 줄은 Stateflow의 이름과 경로다. 그 다음 내용은 단위 시간 별 Active 되는 State의 ID와 Output Signal 값이다.

Chart
Origin/Subsystem/Chart

Time	ActiveState	Output1	Output2
0	63,64	0	0
0.01	65	1	0
0.02	64	0	0

Fig. 5. Simple Model Behavior of Model before Modification

위의 결과를 보면 시뮬레이션의 단위 시간은 0.01초라는 것을 알 수 있다. 0초일 때는 초기 상태로 63번, 64번 State에 도달하였으며 Output1과 Output2의 출력은 0이다. 0.01초일 때는 65번 State로 전이했으며 Output1의 출력이 1로 바뀌었다. 0.02초에는 64번 State로 전이했으며 Output1의 출력이 다시 0으로 바뀌었음을 알 수 있다.

Chart
Origin/Subsystem/Chart

Time	ActiveState	Output1	Output2
0	83,84	0	0
0.01	86	1	0
0.02	84	0	0

Fig. 6. Simple Model Behavior of Modified Model

수정 후 모델에 대해서도 Fig. 6과 같은 결과를 얻을 수 있다. 이때 수정 전 모델의 행동과 수정 후 모델의 Simple 행동의 Active State를 비교해보면 ID가 변화했음을 알 수 있다. 이는 다른 State로 전이해서 그런 것이 아니라 모델 행동 추출을 위해 새로운 테스트 입력을 적용하여 시뮬레이션 할 때마다 State의 ID가 변경되기 때문이다. 이러한 문제를 보완하기 위해 State 정보가 필요하다. Table 1의 State정보는 State ID와 State Path로 구성되며 State ID는 시뮬레이션 시 확정되는 State의 고유 숫자이다. State Path는 모델 내 State의 계층적 위치를 나타낸다. 다른 Stateflow 내의 여러 State가 이름이 같을 경우 구분을 위해 필요하다.

Table 1. State DB of Model before Modification and Modified Model

Model before modification	
State ID	State Path
63	Subsystem/Chart/State
64	Subsystem/Chart/State/S0
65	Subsystem/Chart/State/S1
66	Subsystem/Chart/State/S2

Modified model	
State ID	State Path
83	Subsystem/Chart/State
84	Subsystem/Chart/State/S0
85	Subsystem/Chart/State/S2
86	Subsystem/Chart/State/S1

3) 구조 비교

구조 비교 단계는 수정 전, 후 모델의 구조적 차이 및 문법적 차이를 비교하는 과정이다. 이를 위해 먼저 Simulink 구조 DB와 Stateflow 정보 DB를 생성한다. Simulink 구조는 Sub system들의 계층 구조를 파악하기 위한 DB로 각 계층별 Input, Output, Simulink Block, Sub System 정보, 각 요소 간 연결 관계를 추출한 정보이다. Stateflow 정보 DB는 모델 내 모든 Stateflow들의 정보를 추출한 것으로 Stateflow 별 포함된 State, Transition, Function 정보를 포함한다. 이 정보를 통해 수정 전, 후 모델의 그래프를 구축하여 그래프 비교를 통해 모델 수정사항을 파악할 수 있다. 비교 요소는 Simulink의 경우 Block의 삭제, 추가, 교체 Block간 연결된 Signal의 삭제, 추가, 수정으로 구성되고 Stateflow의 경우 State 삭제, 추가, 변경과 Transition의 변경, Function의 추가, 삭제, 변경으로 구성된다. 예를 들어 Origin 모델과 Modified 모델의 구조를 비교하면 다음 Fig. 7과 같은 결과를 얻을 수 있다.

모델 구조 비교 결과 Stateflow 내부 Transition이 변경 및 삭제된 것을 알 수 있다.

```
Before  /Origin.mdl
After   /Modified.mdl

START
StateFlow
Path : Origin/Subsystem/Chart
Name : State

[CHANGE_TRANSITION]
S0 -> S1 [order = 1]
S0 -> S2 [order = 2]
S1 -> S2 [order = 1]

[DELETE_TRANSITION]
S2 -> S1 [order = 2]
```

Fig. 7. Comparison of Model Structure before and after Modification

4) 행동 비교

행동 비교기는 앞서 살펴 본 행동 생성 단계의 결과물인 모델 수정 전, 후 행동 결과를 비교하는 것이다. 이를 통해 모델 equivalence 평가를 진행한다. Simple 행동이면 시간별 Active State와 Output signal을 비교하여 일치 여부를 판별하고 Full 행동이면 시간별 Active State, Active Transition, Junction, Output signal을 비교하여 일치 여부를 판별한다. 같은 테스트 케이스를 앞의 수정 전, 후 모델에 적용한 simple 행동과 행동 비교 결과는 다음과 같다.

수정 전, 후 모델에 대한 State DB와 Transition DB를 참고하여 State ID 별 동치 관계를 파악 할 수 있고 이에 따라 해당 테스트 케이스에 대해 수정 전, 후 모델 Simple 행동이 같다는 것을 알 수 있다.

Chart
Origin/Subsystem/Chart

Time	ActiveState	Output1	Output2
0	63,64	0	0
0.01	65	1	0
0.02	64	0	0

Fig. 8. Simple Behavior of Model before Modification

Chart
Origin/Subsystem/Chart

Time	ActiveState	Output1	Output2
0	83,84	0	0
0.01	86	1	0
0.02	84	0	0

Fig 9. Simple Behavior of Modified Model

Test Case 1_Behavior

Chart
Origin/Subsystem/Chart

Time	ActiveState	Output1	Output2
0	Same	Same	Same
0.01	Same	Same	Same
0.02	Same	Same	Same

Fig. 10. Comparison of Model of before and after Modification

5) 테스트 케이스 재사용성 평가

테스트 케이스 재사용성 평가는 수정 전, 후 모델 행동 비교 결과, 모델 구조 비교 결과를 통해 진행한다. 단위 시간 별 모델 행동 비교 결과를 모델 구조 비교 결과를 토대로 비교한 후 일치하면 재사용 가능, 불일치하면 재사용 불가 판정을 한다.

앞서 살펴본 예의 결과를 토대로 재사용성 평가를 하면 수정 전, 후 모델 행동이 같으므로 해당 테스트 케이스를 State/Transition 목적으로 재사용 가능하고 평가할 수 있다.

6) 모델 행동 상세 평가

테스트 케이스 재사용성 평가 시 실제 이동한 Transition 등 좀 더 자세한 정보를 얻고 싶거나 MC/DC로 평가하고 싶을 경우 단위 시간 별 Active Transition, Active Junction도 확인할 필요가 있다. 같은 테스트 케이스를 앞의 수정 전, 후 모델에 적용한 상세 행동은 다음 Fig. 11, Fig. 12와 같다.

Chart
Origin/Subsystem/Chart

Time	ActiveTransition	ActiveState	ActiveJunction	Output1	Output2
0	48, 59		44, 45		0
	0				
0.01	52	46		1	0
0.02	51	45		0	0

Fig. 11. Full Behavior of Model before Modification

Chart
Origin/Subsystem/Chart

Time	ActiveTransition	ActiveState	ActiveJunction	Output1	Output2
0	68,78		64,65		0
	0				
0.01	72	67		1	0
0.02	71	65		0	0

Fig. 12. Full Behavior of Model after Modification

수정 전, 후 모델 Full 행동의 Active Transition을 비교해 보면 ID가 변화했음을 알 수 있다. 이는 다른 Transition으로 전이해서 그런 것이 아니라 모델 행동 추출을 위해 새로운 테스트 입력을 적용하여 시뮬레이션 할 때 마다 Transition의 ID가 변경되기 때문이다. 이러한 문제를 보완하기 위해 Transition DB가 필요하다. Transition DB는 Transition ID와 Source State ID, Destination State ID, Label String으로 구성된다. 이때 Source State ID가 없다면 Default Transition 임을 알 수 있다. 수정 전, 후 모델에 대한 Transition DB는 다음 Table 2와 같다. T ID는 Transition ID, SS ID는 Source State ID, DS ID는 Destination ID를 의미한다.

Table 2. Transition DB before and after Modification

Model before modification			
T ID	SS ID	DS ID	Label String
59		44	
48		45	
49	45	47	[In2 == 1] {Out1 = 0; Out2 = 1;}
50	47	45	[In1 == 0 && In2 == 0]
51	46	45	[In1 == 0 && In2 == 0]
52	45	46	[In1 == 1] {Out1 = 1;}
53	46	47	[In2 == 1] {Out1 = 0; Out2 = 1;}
54	47	46	[In1 == 1] {Out1 = 1; Out2 = 0;}

Modified model			
T ID	SS ID	DS ID	Label String
78		64	
68		65	
69	65	66	[In1 == 0 && In2 == 1] {Out1 = 0; Out2 = 1;}
70	66	65	[In1 == 0 && In2 == 0]
71	67	65	[In1 == 0 && In2 == 0]
72	65	67	[In1 == 1 && In2 == 0] {Out1 = 1;}
73	67	66	[In1 == 0 && In2 == 1] {Out1 = 0; Out2 = 1;}

5. 실 험

본 연구에서 제안하는 테스트케이스 재사용 기법을 상용 자동차 제어기 모델을 통해 평가하였다. 실험에 사용된 제어 모델은 3개로 각 제어 모델의 사양은 다음 <Table 3>과 같다.

제어모델1은 Stateflow 하나만 존재하며 비교적 적은 수의 State와 Transition을 가진다. 제어모델2는 3개의 Stateflow가 존재하여 앞단의 수정에 의한 뒷단의 영향을 알아볼 수 있다. 제어모델3 역시 여러 개의 Stateflow들이 서로 연결되어 있으며 이전 모델들과 비교하여 상당히 많은 수의 State와 Transition을 가진다. 각 제어 모델의 테스트 케이스는 State, Transition Coverage 목표이므로 Model equivalence 비교 시 Simple 모델 행동을 사용한다.

Table 3. Specification of Control Model used in the Experiment

	Model1	Model2	Model3
Stateflow	1	3	7
State	8	16	42
Transition	17	29	115
Test Case	9	40	37

각 제어 모델의 수정 사항은 Transition 삭제, Transition 조건 수정, State 삭제, 추가 등이 있다. 제어모델3이 가장 많은 변경 사항을 가지며 제어모델2, 제어모델1 순으로 단순한 변경 사항을 가진다.

각 제어 모델에 대한 테스트케이스 재사용성 평가 결과는 Table 4와 같다.

Table 4. Evaluation of Test Case Reusability by Control Model

	Model1	Model2	Model3
Test Case	9	40	37
Reusable	5	22	0
Non-reusable	4	18	37
Reuse rate(%)	56%	55%	0%

Table 4는 제어 모델 별 수정 전, 후 모델에 대한 테스트 케이스 재사용성 평가 결과이다. 제어모델1은 총 9개의 테스트 케이스 중 5개가 재사용 가능했고 제어모델2는 40개의 테스트케이스 중 22개가 재사용 가능했다. 하지만 제어모델3은 37개의 테스트 케이스 모두 재사용 할 수 없었다. 그 이유는 제어모델3은 여러 서브시스템으로 구성되어 서로 영향을 줬기 때문이며 수정 사항도 가장 많았기 때문이다.

이때 재사용 불가능 테스트 케이스도 일부분은 재사용이 가능하다. Model equivalence가 서로 달라지는 시점을 알 수 있기 때문에 그 이후의 테스트 케이스만 다시 작성하면 되기 때문이다.

6. 결 론

본 논문에서는 임베디드 시스템의 기능 변경으로 시스템의 기능이 표현된 모델 수정 시 수정 전 테스트 케이스 중 재사용 할 수 있는 테스트 케이스를 효과적으로 추출하여 사용하는 방안을 제시했다. 기존의 방법은 모델이 수정되면 처음부터 다시 테스트 케이스를 생성하여 시스템 개발 기간이 증가하는 문제가 존재했다. 제안하는 방법은 재사용 가능한 테스트 케이스는 그대로 사용하고 재사용 불가능한 부분의 테스트 케이스만 생성하면 되므로 테스트 시간을 줄일 수 있고 개발 기간을 획기적으로 단축할 수 있다. 제안한 방법의 유용성은 실제 자동차 제어 모델을 사용하여 확인하였다. 향후, 더 복잡한 모델에 대하여도 검증을 실시하여 유용성을 더 검증할 필요가 있다.

References

[1] H. Liu and H. B. Kuan Tan, “Covering Code Behavior on Input Validation in Functional Testing,” *Information and Software Technology*, Vol.51, No.2, pp.546-553, 2009.

[2] S. Nidhra and J. Dondeti, “Blackbox and Whitebox Testing Techniques-A Literature Review,” *Int. J. Embed. Syst. Appl. (IJESA)*, Vol.2, No.2, 2012.

[3] C. D. Nguyen, A. Marchetto, and P. Tonella. “Combining Model-based and Combinatorial Testing for Effective Test Case Generation,” In ISSTA, pp.100-110, 2012.

[4] R. Svenningsson, H. Eriksson, J. Vinter, and M. Törngren, “Model-Implemented Fault Injection for Hardware Fault Simulation,” *Model-Driven Engineering, Verification, and Validation (MoDeVVa)*, pp.31-36, 2010.

[5] J. Yang, J. Bauman, and A. Beydoun, “A Systems Engineering Approach to Verification of Distributed Body Control Applications Development,” *SAE Technical Paper 2010-01-2328*, 2010.

[6] S. Guo, J. Zhang, W. Tong, and Z. Liu, “An Application of Ontology to Test Case Reuse,” *International Conference on Mechatronic Science, Electric Engineering and Computer*, pp.19-22, 2011.

[7] Y. Dong, Y. Wang, M. F. Lau, and S. Y. Lin, “Experiments on Test Case Reuse of Test Coverage Criteria,” *in Ubiquitous Intelligence Computing and 7th International Conference on Autonomic Trusted Computing (UIC/ATC)*, 7th International Conference on, pp.277-281. 2010,

[8] Y. Dong, M. F. Lau, and S. Y. Lin, “On Partitioning the Domain for Test Case Reusability,” *The Eighth International Conference on Quality Software*, pp.264-269, 2008.

[9] A. B. Hocking, J. Knight, M. Aiello, and S. Shiraishi, “Proving model equivalence in model based design,” *in Software Reliability Engineering Workshops (ISSREW)*, 2014 IEEE International Symposium on, pp.18-21, 2014.

[10] SimDiff [Internet], http://www.ensoftcorp.com/KO/simdiff/ Inc.

[11] K. C. Shashidhar, M. Bruynooghe, F. Catthoor, and G. Janssens, “Functional Equivalence Checking for Verification of Algebraic Transformations on Array-intensive Source Code,” *Proc. DATE'05*, pp.1310-1315, 2005.

[12] K. C. Shashidhar, M. Bruynooghe, F. Catthoor, and G. Janssens, “Verification of Source Code Transformations by Program Equivalence Checking,” *International Conference on Compiler Construction*, pp.221-236, 2005.

[13] Mahapatra, S., Egel, T., Hassan, R., Shenoy, R. et al., “Model-Based Design for Hybrid Electric Vehicle Systems,” *SAE Technical Paper 2008-01-0085*, 2008.

[14] S. Mohalik, A. A. Gadkari, A. Yeolekar, K. Shashidhar, and

S. Ramesh, “Automatic Test Case Generation from Simulink/Stateflow Models using Model Checking,” *Software Testing Verification and Reliability*, Vol.24, No.2, pp.155-180, 2014.

[15] MATLAB Simulink Coder [Internet], https://kr.mathworks.com/products/simulink-coder.html, Inc.

[16] MATLAB Design Verifier [Internet], https://kr.mathworks.com/products/sldesignverifier.html, Inc.

[17] A. Sridhar, D. Srinivasulu, and D. P. Mohapatra. “Model-based Test-case Generation for Simulink/Stateflow using Dependency Graph Approach,” *In 2013 3rd IEEE International Advance Computing Conference (IACC)*, pp.1414-1419. 2013.

[18] H. G. Park, K. H. Chung, and K. H. Choi, “Test Case Generation For Simulink/Stateflow Model Using Yices and Model Information,” *KIPS Transactions on Software and Data Engineering*, Vol.6, pp.293-302, 2017.

박 건 구

http://orcid.org/0000-0003-4413-0088
e-mail : parkgungu@ajou.ac.kr
2017년 아주대학교 전자공학과(학사)
2017년～현 재 아주대학교 전자공학과 석사과정
관심분야 : 실시간 운영체제, 임베디드 시스템 테스팅, 모델 기반 테스트

한 혜 진

https://orcid.org/0000-0002-1878-6270
e-mail : chloeehan@ajou.ac.kr
2017년～현 재 아주대학교 컴퓨터공학과 석사과정
관심분야 : 임베디드 시스템, 임베디드 시스템 테스팅

정 기 현

https://orcid.org/0000-0002-3745-9101
e-mail : khchung@ajou.ac.kr
1984 서강대학교 전자공학과(학사)
1988 미국 Ilinois주립대 EECS(석사)
1990 미국 Purdue대학 전기전자공학부 (박사)
현 재 아주대학교 전자공학과 교수
관심분야 : 컴퓨터구조, VLSI설계, 실시간 시스템, 테스팅 등

최 경 희

https://orcid.org/0000-0002-3918-4471
e-mail : khchoi@ajou.ac.kr
1976 서울대학교 수학교육과(학사)
1979 프랑스 그랑데꼴 Enseeiht 대학(석사)
1982 프랑스 Paul Sabatier대학 정보공학부(박사)
현 재 아주대학교 컴퓨터공학과 교수
관심분야 : 컴퓨터공학, 운영체제, 실시간 시스템, 테스팅 등