KIPS Tr. Software and Data Eng.
Vol.6, No.6 pp.293~302 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2017.6.6.293

Test Case Generation For Simulink/Stateflow Model Using Yices and Model Information

Han Gon Park† · Kihyun Chung†† · Kyunghee Choi†††

ABSTRACT

This paper proposes a method that generates test cases from Simulink/Stateflow(SL/SF) using a SMT (Satisfiability Modulo Theory) solver, Yices and information of SL/SF model. The most difficult problem to generate test cases from SL/SF model is to solve reachability problem. In the propose method, Yices and the tables built with the model information are utilized to solve the reachability problem. The method utilizes the SMT model, that is the SL/SF model transformed in Yices. The tables built from SL/SF are used for backward processing of the proposed method and increases test generation efficiency. A commercial refrigerator model and two car ECU (Electrical Control Unit) models are used to evaluate the performance of the proposed algorithm..

Keywords : Test Case Generation, Simulink/Stateflow, Model Based Test, Yices

Yices와 모델 정보를 이용한 Simulink/Stateflow 모델의 테스트 케이스 생성 기법

박 한 곤† · 정 기 현†† · 최 경 희†††

요 약

본 논문에서는 SMT (Satisfiability Modulo Theories) Solver인 Yices와 Simulink/Stateflow (SL/SF)의 모델 정보를 이용한 SL/SF 모델의 테스트 케이스 생성 기법을 제안한다. SL/SF 모델로부터 테스트 케이스 생성 시 발생하는 가장 어려운 점은 도달 가능성 문제를 해결하는 것이다. 제안하는 방법에서는 Yices와 모델 정보로부터 만들어진 테이블로 도달 가능성 문제를 해결한다. 제안하는 방법에서는 SL/SF 모델을 Yices의 입력 언어로 변환한 SMT 모델을 사용하여 테스트 케이스를 생성한다. SL/SF 모델로부터 생성된 정보들은 제안하는 테스트 케이스 생성 알고리즘의 Backward 프로세싱에 사용되어 테스트 케이스 생성 효율을 증가시킨다. 제안된 테스트 케이스 생성 기법은 상용 냉장고 제어 시스템 모델과 자동차의 ECU (Electrical Control Unit) 모델을 이용하여 성능을 평가한다.

키워드 : 테스트 케이스 생성, 시뮬링크/스테이트플로우, 모델 기반 테스트, 와이즈

1. 서 론

오늘날 사용자의 요구사항이 점점 다양해짐에 따라 임베디드 시스템의 복잡도가 증가하고 있다. 임베디드 시스템의 복잡도 증가는 곧 내장되는 소프트웨어의 복잡도 증가를 의미하고 소프트웨어 복잡도의 증가는 결함발생 가능성의 증가를 의미한다. 소프트웨어 결함은 요구사항 오류, 설계 오류, 코드 오류 등 다양한 원인에 의해 발생하지만, 일반적으로 요구사항 오류에 의해 가장 많이 발생한다. 요구사항 오류에는 요구사항 자체의 오류, 요구사항 간의 오류, 요구사항과 코드의 불일치 등 다양한 유형이 있다. 요구사항에서 발생하는 결함을 검출하기 위해 주로 블랙박스 테스트가 사용된다.

블랙박스(Black Box) 테스트는 프로그램의 소스코드를 분석하여 이를 바탕으로 시스템을 테스트하는 화이트 박스(White Box) 테스트와는 달리 시스템을 블랙박스로 간주하여 내부 구조, 소스 코드 등에 대한 상세한 정보 없이 명세를 기반으로 시스템을 테스트 한다[1]. 화이트 박스 테스트의 경우 소스코드, 내부구조 등에 대한 지식이 필요하기 때문에 많은 테스트 노력이 필요하지만, 블랙박스 테스트의 경우 소스코드나 내부구조에 대한 지식이 없어도 명세만 있으면 시스템을 테스트 할 수 있다. 블랙박스 테스트에서는 명세를 기반으로 시스템에 입력을 가하고 그에 대한 시스템의 출력

※ 본 연구는 방위사업청(UD150042AD)의 지원으로 수행되었음.
† 준 회 원 : 아주대학교 전자공학과 석사
†† 정 회 원 : 아주대학교 전자공학과 교수
††† 정 회 원 : 아주대학교 소프트웨어학과 교수
Manuscript Received : October 19, 2016
Accepted : November 9, 2016
* Corresponding Author : Kihyun Chung(khchung@ajou.ac.kr)

을 확인하는 방식으로 테스트가 이뤄지기 때문에 테스트 케이스(Test case)가 테스트의 결과에 가장 중요한 영향을 미치는 요소 중의 하나이다. 즉, 테스트 케이스의 품질이 높을수록 테스트의 신뢰성이 향상되고 적은 노력으로 임베디드 소프트웨어의 결함을 찾아낼 수 있는 가능성이 높아진다.

최근 자동차, 전자 제품, 항공 등의 산업체는 제품의 요구사항을 자연어가 아닌 정형화된 모델을 이용하여 관리하고, 이를 이용하여 블랙박스 테스트를 진행하는 모델 기반 테스트 기법을 많이 사용한다. 모델기반 테스트 기법은 여러 가지 장점을 가진다. 첫째, 모델은 시뮬레이션이 가능하므로 요구사항 검증이 가능하다. 둘째, 테스트 케이스 생성이 가능하다. 셋째, 테스트 수행 시 테스트 입력에 대한 예상 출력 생성 시스템 즉 오라클(Oracle)로도 사용이 가능하다.

다양한 모델링 도구가 존재하지만 그 중에서도 Mathwork사의 Simulink/Stateflow(SL/SF)[2]는 자동차, 항공, 중장비 분야에서 널리 사용된다. 요구사항을 Stateflow 모델로 관리할 경우 추상적인 요구사항을 시각적인 모델로 표현하기 때문에 실제 시스템과 개념적 거리감을 줄일 수 있고 재사용성을 높일 수 있다. 또한, 작성된 모델을 테스팅 시스템에서 사용할 수 있고[3], 모델로부터 자동 코드 생성(Auto code generation)을 통해 소프트웨어의 코드를 자동으로 생성할 수 있다[4].

모델 기반 블랙박스 테스트에서 가장 중요한 작업 중의 하나는 테스트 목적에 적합한 테스트 케이스를 생성해야 하는 것이다. 규모와 복잡도가 낮은 시스템의 경우 엔지니어의 경험, 직관에 의해 수동으로 테스트 케이스를 생성하여 테스트에 사용할 수 있지만 자동차, 항공기와 같이 복잡한 시스템의 경우 수동 생성에는 한계가 있다. 따라서 모델로부터 테스트 케이스를 자동으로 추출하는 기법이 필요하다.

모델 기반 테스트에서 특정 테스트 대상의 테스트 케이스를 생성하기 위해서는 먼저 시스템이 특정 테스트 대상에 도달할 수 있는 바로 이전 상태에 있어야 한다. 이것은 도달 가능성 문제의 일종으로, 도달 가능성 문제[5]란 어떤 특정한 상태에서 대상 상태에 도달할 수 있는지 결정하는 문제를 말한다. 모델 기반 테스트에서 테스트 대상(혹은 상태)을 만족시키기 위한 시스템 입력을 구하기 위해서는 시스템을 바로 테스트 대상 바로 이전 상태에 도달시킬 수 있어야, 그 상태로부터 해당 테스트 대상의 테스트 케이스를 생성할 수 있게 된다. 따라서 테스트 케이스 생성을 위해서는 도달 가능성 문제를 해결해야 한다.

SL/SF 모델 기반 테스트 케이스 생성 시 도달 가능성 문제를 해결하기 위한 많은 연구가 있었다. 이에 대한 자세한 내용은 2장에서 다룬다. 많은 선행연구에서 도달 가능성 문제를 충분히 해결하지 못했기 때문에 모델의 복잡도가 커질 경우 낮은 테스트 케이스 생성 효율(coverage)을 보였다. 본 논문에서는 도달 가능성 문제를 해결하고 높은 테스트 케이스 생성 효율을 위해 SMT (Satisfiability Modulo Theories) Solver의 한 종류 Yices[6]와 모델 정보를 사용하여 SL/SF로 작성된 모델을 대상으로 테스트 케이스를 자동으로 추출하는 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 이전에 제안됐던 모델 기반 테스트 케이스 관련 연구를 기술하고, 3장에서는 테스트 케이스 생성 기법에 대해 자세하게 기술한다. 4장에서는 제안된 테스트 생성 기법을 상업용 모델을 사용해서 평가한 결과를 기술하고, 5장에서는 연구의 결론을 기술한다.

2. 관련 연구

2.1 SImulink/Stateflow 모델

Mathwork사의 SL/SF은 가장 널리 사용되는 모델링 도구로 소프트웨어의 로직(Logic)을 모델링하기에 적합하다. SL/SF 모델은 상태(State)와 전이(Transition)로 표현되는데 상태는 어느 한 순간에서의 시스템의 상태를 의미하고 전이는 상태와 상태를 연결한다. 상태 내부에는 “en:”, “du:” 그리고 “ex:”을 기술할 수 있는데 “en:”에는 상태에 처음 진입했을 때 실행되는 동작을 기술하고 “du:”에는 상태에 머무르는 동안 실행되는 동작을 기술한다. “ex:”에는 상태에서 빠져나갈 때 실행되는 동작을 기술한다. 전이가 발생하기 위해서는 전이 조건(Transition Condition)이 만족돼야하는데 전이 조건의 대괄호“[ ]” 내에 기술되고 전이 조건이 만족했을 때 실행돼야 하는 동작(Action)은 “/” 뒤에 기술된다.

Fig. 1은 Stateflow 모델의 예를 보여준다. 예제에서 모델링한 시스템의 입력은 “b_ACC”이고 출력은 “Power”이다. Fig. 1에서 상태는 “Power_Off”, “Power_On” 그리고 “Maintain”으로 총 3개가 존재하고 전이는 총 4개로 상태를 연결하는 전이 3개와 디폴트(Default) 전이 1개가 존재한다. 디폴트 전이는 검은 점에서 상태로 연결되며, 모델이 동작하는 초기 전이이다. 하나의 상태에서 여러 개의 전이가 존재하는 경우 각 전이는 우선순위를 갖고 여러 개의 전이가 동시에 만족될 경우 우선순위대로 전이가 발생한다. Fig. 1의 모델은 다음과 같이 동작한다. 먼저, 모델이 동작하면 디폴트 전이에 의해 “Power”은 0의 값을 갖고 “Power_Off” 상태에 진입한다. “Power_Off” 상태에서 입력 “b_ACC”의 값으로 1이 인가되면 “Power”은 1의 값을 갖고 “Power_On” 상태로 한다. “Power_On”상태에 있는

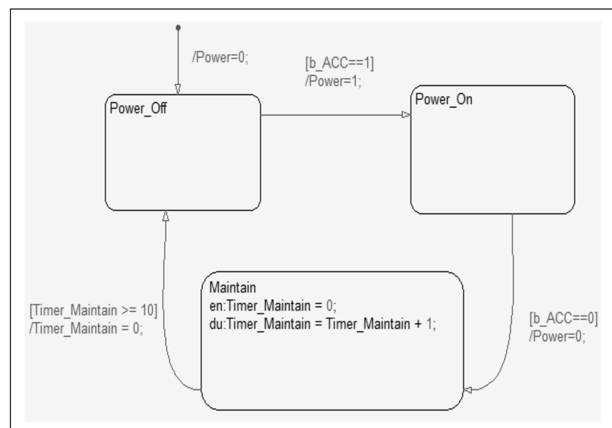


Fig. 1. An Example of Stateflow Model

동안 “Power”는 1의 값을 유지한다. “Power_On” 상태에서 입력 “b_ACC”의 값으로 0이 인가되면 “Power”는 0의 값을 갖고 “Maintain” 상태로 이동한다. “Maintain”상태에 처음 진입하면 “en:”이 실행되어 “Timer_Maintain” 변수에 0의 값이 할당되고, 계속 머무를 경우 “du:”가 실행되어 매 tick마다 “Timer_Maintain” 변수의 값이 1씩 증가한다. 여기서 tick은 SL/SF 시뮬레이션 클럭(Simulation Clock)을 의미한다. “du:”가 계속 실행되어 “Timer_Maintain” 변수의 값이 10이 되면 “Timer_Maintain” 변수에 0의 값이 할당되고, “Maintain” 상태에서 “Power_Off” 상태로 이동한다.

2.2 SL/SF 모델 기반 테스트 케이스 생성

SL/SF 모델로부터 테스트 케이스를 생성하는 방법에 관한 선행연구가 많이 있었다. [7]은 도달 가능성 문제를 해결하고 테스트 케이스를 구하기 위해 SL/SF 모델을 무작위로 시뮬레이션하고 모델의 피드백을 활용하지만, 전이 조건이 복잡하거나 모델의 복잡도가 큰 경우 테스트 케이스의 생성 효율이 떨어진다. [8]는 SL/SF 모델을 실행 가능한 모델로 변경한 후, Symbolic PathFinder [9]를 사용해서 테스트 케이스를 생성한다. 이 방법은 모델 변환 시 소요되는 비용이 매우 크다는 단점이 있다. [10]은 SL/SF 모델을 모델 분석 프로그램인 SMV 프로그램[11]으로 변환한 후 CTL [12]를 사용하여 테스트 케이스 생성한다. 필요한 변수 값은 빠르게 구할 수 있지만, 모델이 복잡할 경우 역시 도달 가능성 문제를 해결하지 못했다. [13]은 테스트 케이스 생성을 위해 로봇 공학에서 주로 사용되는 RRT 알고리즘을 적용한다. 테스트 케이스의 생성 효율은 증가했지만, 입력 값이 무작위로 변경되어 시나리오 기반으로 테스트 케이스가 생성되지 않고 테스트 케이스의 의미 파악이 어렵다는 단점이 존재한다. [14]는 SL/SF 모델을 SMT 모델로 변환한 후 Yices를 이용하여 테스트 케이스를 생성한다. 이 방법의 경우 전체 시스템을 대상으로 테스트 케이스를 생성하기 때문에 테스트 케이스 생성 시간이 오래 걸리고 특정 서브시스템의 테스트 케이스를 생성하지 못한다는 단점이 있다. [15]에서는 SL/SF 모델을 SMT 모델로 변환한 후 Yices를 이용하여 Stateflow 단위로 테스트 케이스를 생성한다. Stateflow 단위로 테스트를 생성하기 때문에 [14] 방법 보다 테스트 케이스 생성 시간이 단축되고 재사용성을 높였지만, 생성되는 테스트 케이스가 지엽적이 될 가능성이 높고 모델의 복잡도가 커지거나 하나의 상태에서 복수개의 전이가 발생 가능한 상태가 많은 경우 테스트 케이스 생성 효율이 떨어진다. [14]와 [15] 모두에서의 문제점은 모델이 피드백이 가지거나 복잡한 전이 조건을 가질 때 테스트 케이스 생성이 어렵다는 점이다. 본 논문에서는 모델의 상태에 대한 정보를 Table에 저장하고 테스트 케이스 생성 시에 사용하여 이전 방법의 문제를 완화하였다. [16]에서는 SL/SF 모델과 유사한 UML 모델을 이용하여 테스트 케이스를 생성하지만 내부 변수에 대해 고려하지 않는다.

SL/SF 모델을 대상으로 테스트 케이스를 생성하는 여러 상업용 도구가 있다. [17]은 휴리스틱 기법의 단점을 개선하기 위해 무작위 입력 생성을 기반으로 SAT Solver [18]을 사용하여 테스트 케이스를 생성하지만, 사용자 정의 함수가 모델에 포함되어 있는 경우 테스트 케이스 생성 효율이 떨어진다. [19]는 무작위 기반 휴리스틱 기법을 사용하여 테스트 케이스를 생성한다. 휴리스틱 기법을 사용하기 위해서는 사용자가 모델을 추가적으로 분석하여 정보를 입력해야 했기 때문에 많은 노력이 요구된다.

2.3 SMT Solver

SMT Solver는 SMT (Satisfiability Modulo Theories) 문제를 해결하는 엔진이다. SMT 문제란 어떠한 일차 논리식이 주어졌을 때, 주어진 논리식이 참이 되게 하는 변수의 값이 존재하는지 검사하는 문제이다. 예를 들어, SMT Solver에 일차 논리식으로 “1<x<5”, “y==8”이 주어졌다고 했을 때 SMT Solver는 주어진 논리식이 참이 되게 하는 변수 x, y의 값을 구해준다. SMT Solver에는 다양한 종류가 있고[20], SMT Solver마다 정량화된 언어 형식을 사용한다. 즉, SMT Solver를 사용하기 위해서는 일차 논리식을 사용하고자 하는 SMT Solver의 입력 언어 형식으로 변환해야 한다. SMT Solver는 일차 논리식으로 표현가능 한 선형적인 문제만 해결할 수 있기 때문에 제약을 갖지만, 테스트 케이스 생성(Test Case Generation), 모델 체킹(Model Checking) 등 공학 소프트웨어 분야에서 주로 사용된다.

본 논문에서는 도달 가능성 문제를 해결하기 위해 SMT Solver로 Yices를 사용하여 Stateflow 모델의 테스트 케이스를 생성한다. 이를 위해 Stateflow 모델을 Yices의 입력 언어로 변환하여 선형 논리식으로 표현한 모델을 사용하는데, 이 때 변환된 모델을 SMT 모델[14]이라 한다. SMT 모델을 Yices에 입력한 후 커버하고자 하는 테스트 대상의 조건을 Yices에 입력하면 현재 모델의 상태에서 조건을 만족하는 변수 값들이 존재하는 경우 Yices는 그 값들을 구해준다. 이와 같은 방식을 사용하면 만족해야하는 테스트 대상의 깊이(Depth)가 깊어도 최상위 노드에서 테스트 대상에 도달하기 필요한 조건 값들만 알면 테스트 케이스를 구할 수 있다. Yices의 입력 언어는 모든 구문이 변수로 표현되고 반드시 후위 연산으로 표현해야한다. Fig. 2와 Fig. 3은 각각 Fig. 1의 “Maintain” 상태와 “Power_Off”에서 “Power_On”로 향하는 Flowgraph의 SMT 모델의 예이다.

```
;;-------------------------------------------------
;;      Maintain        State   ID 5
;;-------------------------------------------------
(define State_5_Entry::bool
        (= Timer_Maintain_2_50 0)
)

(define State_5_During::bool
        (= Timer_Maintain_2_50 (+ Timer_Maintain_2_51 1))
)

(define State_5_Exit::bool
        (= 0 0)
)
```

Fig. 2. An SMT Model Example of State

```
(define FG_State_3::bool
        (if (= b_ACC_2 1)
                (and (= State_3_Exit true)
                (= Power_2_0 1)
                (= State_4_Entry true)
                (= ActiveState_Out_2 4)
                )
                (and (= State_3_During true)
                        (= Power_2_0 Power_2_1)
                        (= ActiveState_Out_2 3)
                        ))
        ))
```

Fig. 3. An SMT Model Example of Flowgraph

3. SL/SF 정보를 이용한 테스트 케이스 생성

테스트 케이스는 테스트 목표에 따라 다르게 생성된다. 테스트 목표란 테스트하고자 하는 대상을 정의하는 것을 말한다. 커버리지 기반 테스트 생성의 경우 테스트 목표에는 상태 커버리지(State Coverage), 전이 커버리지(Transition Coverage) 그리고 MCDC (Modified Condition and Decision Coverage) 등이 있다. 커버리지(Coverage)는 전체 테스트 대상 중에 실제 테스트 된 테스트 대상의 비율을 말한다. 예를 들어, 테스트 목표를 전이 커버리지로 설정하여 생성한 테스트 케이스 세트가 70%의 전이 커버리지를 가진다면, 그 테스트 케이스 세트로 SL/SF 모델 전체 전이의 개수의 70%의 전이를 테스트할 수 있다.

제안하는 방법에서는 테스트 케이스를 생성하기 위해 Yices를 사용한다. Stateflow 모델로부터 변환된 SMT 모델을 Yices에 입력하고 테스트 대상을 만족하기 위한 조건을 Yices 입력언어로 변환한 후 Yices를 이용하여 테스트 케이스를 구한다. 테스트 케이스 생성에 필요한 정보는 SL/SF 모델을 분석하여 생성하는 여러 가지 Table을 활용한다.

3.1 테스트 케이스 생성을 위한 정보

본 논문에서 제안하는 테스트 생성 기법을 사용하기 위해서는 SL/SF 모델의 여러 정보가 필요하다. 모델 정보의 종류는 크게 Yices 정보, Config 정보 그리고 User Define 정보로 나뉜다. Yices 정보에는 테스트 케이스 생성 시 Yices를 사용하기 위해 필요한 정보들이 저장되고 Config 정보에는 생성된 테스트 케이스와 그 때의 Stateflow의 상태 정보가 저장된다. User Define 정보는 테스트 케이스 생성 알고리즘에서 사용되는 정보들을 포함한다. Yices 정보와 User Define 정보는 테스트 케이스 생성 전에 미리 구축하지만, Config 정보는 테스트 케이스를 생성하면서 구축된다.

Yices 정보는 Symbol Table, Target Table 그리고 State Table로 구성된다. Symbol Table에는 입력 변수, 출력 변수, 로컬 변수, 상태 변수 등 Yices에서 사용되는 SL/SF 모델의 모든 변수 정보가 들어있다. Target Table에는 SL/SF 모델 내에서 테스트 대상이 될 수 있는 모든 정보가 Yices의 입력언어로 변환되어 들어있다. 테스트 대상은 테스트 목표에 따라 달라진다. 예를 들어, 테스트 목표가 전이 커버리지인 경우 SL/SF 모델에 존재하는 모든 전이가 테스트 대상이 된다. 따라서 Target Table에는 모든 테스트 대상의 정보가 들어있다. State Table에는 SL/SF 모델 내에 존재하는 모든 상태들의 정보가 들어있다.

Config 정보에는 테스트 케이스 생성 시 생성되는 정보들이 트리구조로 저장된다. 이 정보는 SL/SF 모델에서 현재 활성화된 상태, 변수, 진행 시간 정보와 현재 상태에서 생성된 테스트 케이스가 저장된다. 또한 Config 정보에 저장된 정보는 테스트 대상의 입력을 구할 때, 시스템 입력을 구할 때, 시뮬레이션 등 테스트 케이스 생성 알고리즘 전체에서 사용된다.

User Define 정보는 InterState Transition Table, Output Generation Table, State Matrix Table 그리고 Input Range Table로 구성된다.

InterState Transition Table에는 현재 상태에서 다른 상태로 전이하기 위해 사용되는 입력 정보가 들어있다. 예를 들어, Stateflow 내에서 “Off”라는 상태에 머물러 있다가 입력 “Power”에 1이 인가되어 상태가 “On”으로 이동했다고 하자. 이 때, “Off” 상태에서 다른 상태로 전이하기 위해 사용되는 입력 정보는 “Power”가 된다. InterState Transition Table은 테스트 케이스 생성 과정에서 테스트 대상을 만족하기 위해 구한 Stateflow의 입력을 유지시키기 위해 사용된다. 위의 예에 추가하여 설명하면 다음과 같다. 입력 변수가 2개인 Stateflow A와 B가 있고 A의 출력 “Power”와 “Demo”가 B의 입력으로 들어간다고 하자. 그리고 B의 “Off”에서 “On”로의 전이가 테스트 대상이고 전이 조건은 “[Power==1]”라 하자. Stateflow B에서는 테스트 대상을 만족하기 위한 입력으로 “Power”는 1의 값을 구할 것이다. “Demo”는 테스트 대상과 관련 없는 입력이므로 이전 tick의 입력 값을 유지한다. InterState Transition Table에는 Stateflow B의 “Off” 상태에서 “Power” 정보가 들어있으므로 해당 정보를 참조하여 Stateflow A에서는 “Power”가 1이 되도록 하는 입력 값을 구한다.

Output Generation Table에는 현재 상태에서의 SL/SF 모델의 출력 정보가 들어있다. 예를 들어, “Off” 상태에서 SL/SF 모델의 출력 값이 0일 경우 “Off” 상태의 출력 정보는 0이 된다. Output Generation Table은 SL/SF 모델에서 구한 테스트 케이스로부터 시스템 입력을 구하기 위해 사용된다. 이에 대한 자세한 내용은 3.2절에서 설명한다.

State Matrix Table에는 SL/SF 모델 내에 존재하는 상태들 간의 이동 정보가 들어있다. 현재 상태에서 도달할 수 있는 테스트 대상이 없거나 특정 테스트 대상을 만족하길 원할 때 경로를 알기 위해 사용된다. State Matrix Table은 모델의 존재하는 모든 상태의 계층구조를 분석하여 생성된다. Fig. 4는 State Matrix Table의 예를 보여준다. 첫 번째 행과 열은 각 상태의 ID를 의미한다. 예를 들어, 7번 상태에서 5번 상태로 이동하기 위해 Fig. 4의 State Matrix Table을 참조하면 다음과 같다. 먼저, 첫 번째 열에서 7번을 찾고 첫 번째 행에서 5번을 찾는다. 이후 해당 행과 열이 만나는 지점의 상태 ID 3을 구한다. 이것의 의미는 7번에서 5번으

로 가기 위해서는 3번으로 먼저 이동해야 한다는 뜻이다. 이후 첫 번째 열에서 3번을 찾고 첫 번째 행에서 5번을 찾아서 해당 행과 열이 만나는 지점의 상태 ID 4번을 구한다. 행과 열이 만나는 지점의 상태 ID가 5번이 될 때 까지 이와 같은 과정을 반복하면 “7 → 3 → 4 → 5”의 경로를 찾을 수 있다. 자기 자신 상태에 대한 경로, 부모에서 자식 그리고 자식에서 부모로의 상태 간의 경로를 찾아야 하는 경우 빈칸으로 처리된다.

State ID \ State ID	3	9	6	7	8	4	5
3		9	4	4	8	4	4
9	3						
6	3			4	8	4	3
7	3		4		8	4	3
8	3		6	7			
4	3		6	7			5
5	3		6	7		3	

Fig. 4. An Example of State Matrix Table

Input Range Table에는 SL/SF 모델의 입력들의 범위 정보와 SL/SF 모델의 피드백 입력 정보가 들어있다. SL/SF 모델의 입력들의 범위 정보는 Yices가 조건을 만족하기 위한 테스트 케이스를 구할 때 SL/SF 모델의 유효 입력 범위 내에서 구하도록 만들기 위해 사용되나. 예를 들어, 입력 “Power”가 0, 1의 값만을 갖을 수 있다면 “Power”의 유효 입력 값은 0과 1이 된다. SL/SF 모델의 피드백 입력 정보는 피드백을 고려하여 실제 시스템의 동작과 일치하는 테스트 케이스를 생성하기 위해 사용된다. Fig. 5는 Input Range Table의 예를 보여준다. 각 입력 값에 대한 범위가 effective value 열에 들어있고 해당 입력이 피드백 입력인 경우 feedback 열에 1이 표시된다. 예를 들어, C_Vehicle_Speed가 가질 수 있는 값은 {0,1,2,3,4,5} 중의 하나이다. 그리고 피드백 열이 1로 표시된 경우 해당 입력은 피드백 입력이므로 현재 tick의 테스트 케이스 생성 시 입력 값을 새로 구하지 않고 이전 tick의 값을 그대로 사용한다.

input	effective value	min	max	feedback
C_MemoryP1Cmd	0,1	0	1	
C_MemoryP1Cmd_ToOn	0,1	0	1	
C_MemoryP2Cmd	0,1	0	1	
C_MemoryP2Cmd_ToOn	0,1	0	1	
C_PpositiOn	0,1	0	1	
C_VehicleSpeed	0, 1, 2, 3, 4, 5	0	5	
L_MirrHorizontalSNSR_IN	0,1	0	1	
L_MirrVerticalSNSR_IN	0,1	0	1	
b_LHMIRHorizSReturnMemReq	0,1	0	1	1
b_LHMIRSManualMode	0,1,2	0	2	1
b_LHMIRVerSReturnMemReq	0,1	0	1	1
b_LHMIRHorizSReturnMemReq_ToOn	0,1	0	1	1
b_LHMIRVerSReturnMemReq_ToOn	0,1	0	1	1
b_LHIMSSReturnMemReq	0,1	0	1	1
b_LHIMSSReturnMemReq_ToOn	0,1	0	1	1

Fig. 5. An Example of Input Range Table

테스트 케이스 생성 시 사용되는 정보들은 Fig. 6과 같이 정리하여 나타낼 수 있다.

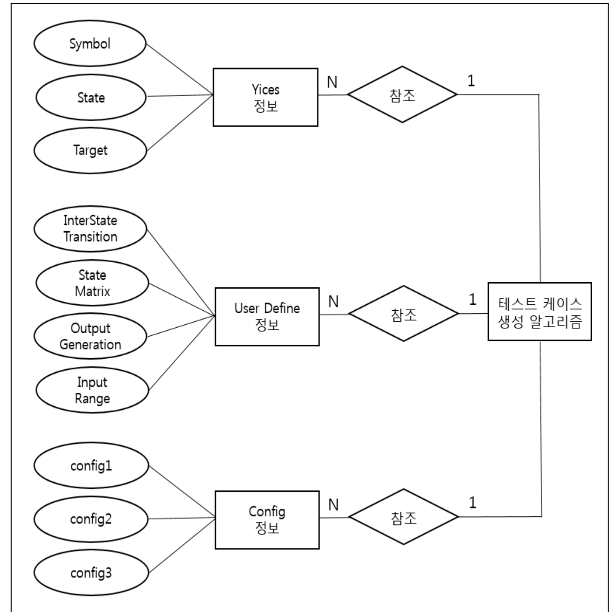


Fig. 6. A Structure of Information

3.2 Yices를 이용한 테스트 케이스 생성

Yices는 테스트 대상을 만족시키기 위한 SL/SF 모델의 입력을 구할 때 사용된다. 예를 들어, SL/SF 모델이 현재 “Unactive”라는 상태에 머물러 있고 테스트 대상은 전이 조건이 “[Input1>0&&Input2==3]”인 “Unactive”에서 “Active”로의 전이라고 하자. 그리고 Stateflow의 입력은 Input1, Input2, Input3 총 3개이고 “Unactive”라는 상태에 있을 때 Input1, Input2, Input3의 값은 각각 0, 1, 2라고 하자. 전이 조건을 만족하는 테스트 케이스를 구하기 위해서 먼저, Stateflow가 현재 “Unactive”에 머물러 있을 때의 SMT 모델을 Yices에 입력시킨다. 그리고 테스트 대상의 조건 즉, “[Input1>0&&Input2==3]”을 Yices 입력 언어로 변환한 후 변환된 조건을 Yices에 입력한다. 이 때, 실제 시스템의 동작과 유사하고 시나리오 기반 테스트 케이스를 생성하기 위해 테스트 조건을 만족시키기 위해 필요한 입력들만 값을 변경하고 나머지 입력 값들은 이전 tick의 값을 그대로 유지시킨다. 따라서 입력 Input1과 Input2의 값만 변경되고 테스트 조건과 관련이 없는 입력 Input3는 “Unactive”라는 상태에 있을 때의 입력 값 2를 그대로 유지시킨다. 이후 Input Range Table을 참조해서 Stateflow의 입력 Input1, Input2의 유효 범위를 Yices에 입력해준다. 조건이 참이 되는 입력 값이 존재할 경우 입력 값을 구해준다. Yices는 조건이 참이 되는 값을 구할 때 참이 되는 값을 무작위로 선정하여 구해주기 때문에 반드시 입력 값의 유효 범위를 설정해주어야 한다. 참이 되는 값이 없을 때는 해당 테스트 목표에 대한 테스트 케이스는 생성할 수 없고 테스트 커버리지는 그만큼 낮아지게 된다.

3.3 테스트 케이스 생성 알고리즘

본 논문에서는 전체 모델의 테스트 케이스를 생성하기 위해 모델을 구성하는 서브 Stateflow(Sub Stateflow) 별로 테스트 케이스를 생성한다. 이 때, 서브 Stateflow들은 실행 순서가 정해져있기 때문에 실행 순서대로 서브 Stateflow의 테스트 케이스를 생성한다. 예를 들어, 자동차의 ECU 모델이 A, B, …, I, J 총 10개의 서브 Stateflow로 구성되어 있고 실행 순서가 알파벳 순서라고 했을 때, ECU 모델의 테스트 케이스를 생성하기 위해 A, B, …, I, J 순서대로 테스트 케이스를 생성한다. 실행 순서대로 테스트 케이스를 생성하는 이유는 A의 출력이 B의 입력으로 인가될 수 있는 것처럼 앞단의 출력이 뒷단의 입력으로 인가될 수 있기 때문이다. 따라서 항상 실행 순서를 고려하여 테스트 케이스를 생성한다.

테스트 케이스 생성 알고리즘의 수도코드(Pseudo Code)는 Algorithm 1에 나와 있다. n개의 서브 Stateflow를 가진 SL/SF 모델의 테스트 케이스 생성 과정은 다음과 같다.

(1) 실행 순서가 앞선 서브 Stateflow부터 실행하여 n번째 Stateflow까지 실행한다. 실행 순서가 동일한 것은 임의로 순서를 정한다.

(2) 서브 Stateflow의 정보들을 읽어온다. 이 때, 정보는 Yices 정보, User Define 정보 그리고 Config 정보를 의미한다.

(3) Stateflow의 Config 정보에서 현재 활성화된 상태, 변수, 실행 시간 정보를 config 변수에 저장한다.

(4) config를 사용하여 테스트 대상을 찾고, Yices를 사용하여 찾은 테스트 대상을 만족하기 위한 테스트 케이스를 구한다. 구한 테스트 케이스 target_input에 저장한다. 이 때, 생성된 테스트 케이스는 피드백 입력을 고려해서 생성되고, target을 만족하는 시스템 입력이 아닌 Stateflow의 입력이다. 따라서 Stateflow의 입력으로부터 시스템 입력을 구하는 과정이 필요하다.

(5) target_input으로부터 시스템 입력을 구하여 system_input에 저장한다.

(6) system_input이 테스트 대상을 만족하는지 확인하기 위해 Yices를 이용해서 시뮬레이션 한다. 시뮬레이션으로 테스트 대상이 만족됐는지 확인하고 만족되었으면 새로운 config를 생성하여 Config 정보에 저장하고 현재 config를 새롭게 생성된 config로 갱신한다. 또한, feedback 신호를 갱신한다.

Algorithm 1. Test Case Generation Algorithm

```
k=1;
for(Stateflow(k) until k>n)
  Info_Load(Stateflow_sequence(k));
  while(all test targets are not covered)
    config = Config Info.current_config;
    target_input = find_target_input(config, Feedback_data);
    system_input = find_system_input(k, target_input);
     if (simulation(system_input))
       Config Info ← new_config;
       Config Info.current_config=new_config;
       update_feedback;
  k=k+1;
```

테스트 케이스 생성 알고리즘에서 핵심이 되는 부분은 (4) 테스트 대상을 만족하기 위한 Stateflow의 입력을 구하는 부분, (5) Stateflow 입력으로부터 시스템 입력을 구하는 부분 그리고 (6) 검사 부분이다. 이와 관련된 자세한 내용은 다음에 차례로 설명한다.

1) find_target_input() 함수

find_target_input()은 현재 Stateflow의 활성화 된 상태에서 발견된 테스트 대상을 만족하기 위해 필요한 Stateflow의 입력을 구한다. Algorithm 2에 find_target_input()의 알고리즘이 나와 있다. find_target_input()은 크게 3부분으로 구성된다. 먼저, config를 사용하여 현재 상태에서 전이 가능한 테스트 대상을 찾는다. 이 때, 발견되는 테스트 대상은 현재 상태에서 하나의 tick이 지난 후 도달할 수 있어야한다. 예를 들어, 현재 config의 실행 시간이 100초이고 (1 tick이 1초라고 가정) 활성화된 상태가 “Active100”일때 경우, “Active100”에서 101초가 됐을 때 전이 가능한 상태들이 테스트 대상으로 발견된다.

현재 활성화된 상태에서 전이 가능한 테스트 대상이 존재하는 경우 테스트 대상을 만족하기 위한 Stateflow의 입력을 구한다. 현재 활성화된 상태에서 전이 가능한 테스트 대상이 존재하지 않는 경우에는 남아 있는 테스트 대상을 확인한다. 전체 테스트 대상 중에 이미 만족됐거나, 만족을 시도했지만 실패한 테스트 대상을 제외한 나머지 테스트 대상들을 r_targets에 저장한다. r_targets은 아직 만족을 시도하지 않은 테스트 대상을 의미한다. r_targets 중 하나의 테스트 대상을 final_target으로 설정한 후 State Matrix Table을 참조하여 현재 상태에서 final_target의 소스 상태(Source State)로 이동하기 위한 next_target을 구한다. 현재 상태에서 next_target을 만족하기 위한 Stateflow의 입력을 구한다. 구한 Stateflow의 입력으로부터 시스템 입력을 구해 시뮬레이션 한 결과 테스트 대상(next_target)을 만족했다고 했을 때 아직 final_target이 만족된 것이 아니므로 현재 상태가 final_target의 소스 상태(Source State)로 이동할 때까지 State Matrix Table을 계속 참조하여 next_target을 구한 후 이동한다. 현재 상태가 final_target의 소스 상태에 도달한 경우 final_target을 해제한다. final_target의 소스 상태에 도달하게 되면 테스트 대상으로 해제한 final_target이 발견되기 때문에 해당 테스트 대상을 만족하는 Stateflow 입력을 구할 수 있다.

Stateflow의 입력 중에 피드백 입력이 존재할 경우 피드백을 고려하여 테스트 케이스를 생성하기 위해 이전 tick의 피드백 정보가 들어있는 feedback를 사용한다. 피드백을 고려한 테스트 케이스 생성은 3.3에서 자세히 설명한다.

Algorithm 2. Pseudo Code of 'find_target_input()' Function

```
find_target_input(config, Feedback_data)

target = find_target(config);
state = config.current_state;
feedback = Feedback_data;
target_input = null;

if(final_target!=null)
  if(state == final_target.Src)
    final_target = null;
  else
    next_target = find_next(state, final_target, State Matrix Table);
   target_input = find_Stateflow_input(next_target,feedback);
    return target_input;

else if(target.count==0)
  r_targets = find_remained_targets(Target Table);
  i=1;
  while (i <= number of remained_targets)
   final_target = r_targets(i);
   next_target = find_next(state, final_target, State Matrix Table);
  target_input = find_Stateflow_input(next_target,feedback);
   i=i+1;
  return target_input;

if(target.count>0)
  j=1;
  while (j <= target.count)
    target_input = find_Stateflow_input(target(j),feedback);
    j=j+1;
  return target_input;
```

2) find_system_input() 함수

find_system_input()은 find_target_input() 함수에서 구한 Stateflow의 입력으로부터 시스템 입력을 구한다. 뒷단의 입력으로부터 가장 앞단의 시스템 입력을 역으로 구하는 Backward 방식이다. Algorithm 3은 find_system_input()의 알고리즘의 pseudo code 이다. 먼저 target_input 중에 앞단의 Stateflow의 입력이나 출력이 있는지 확인해서 target_variable에 저장한다. target_variable이 존재하지 않는 경우 앞단의 Stateflow의 입출력과 target_input이 전혀 관련 없으므로 앞단의 현재 입출력 값을 그대로 target_input에 추가한다. target_variable이 존재할 때는 3가지 경우의 수가 존재한다.

첫째, target_input 중에 앞단의 Stateflow의 입력이 있는 경우 run_yices 함수를 통해 앞단 Stateflow의 현재 활성화된 상태에서 해당 입력을 Yices에 입력하여 시뮬레이션 한다. 시뮬레이션으로 구한 앞단 Stateflow의 입력을 target_input에 추가한다. 이 때, 뒷단의 입력이 앞단에서도 유효하면 Yices는 앞단에서도 동일한 입력을 구해준다. 하지만, 해당 입력이 입력 범위를 초과하는 등 유효하지 않는 경우 해당 target를 만족하는 테스트 케이스는 찾아낼 수 없다.

둘째, target_input에 앞단 Stateflow의 출력이 있는 경우 run_yices 함수를 통해 앞단 Stateflow의 현재 활성화된 상태에서 해당 출력을 발생시켜야 한다. 현재 상태에서 해당 출력을 발생시킬 수 있거나 1 tick 내에 전이할 수 있는 다른 상태에서 해당 출력을 발생시킬 수 있는 경우 Yices 시뮬레이션에 성공한다. 시뮬레이션에 성공하면 시뮬레이션 후 앞단 Stateflow의 입력을 target_input에 추가한다. 시뮬레이션에 성공하지 못한 경우 현재 상태에서 해당 출력을 발생시킬 수 있는 상태로 이동해야한다.

셋째, target_input에 앞단 Stateflow의 입력과 출력이 모두 있는 경우 run_yices 함수를 통해 앞단 Stateflow의 현재 활성화된 상태에서 해당 입력을 Yices에 입력하여 해당 출력을 발생시켜야 한다. 시뮬레이션에 성공하면 앞단 Stateflow의 입력을 target_input에 추가한다. 시뮬레이션에 성공하지 못한 경우 현재 상태에서 해당 출력을 발생시킬 수 있는 상태로 이동해야한다.

run_yices 함수를 통해 시뮬레이션에 실패한 경우 Output Generantion Table을 참조해서 해당 출력을 발생시킬 수 있는 후보 상태 candidate_state를 찾는다. candidate_state가 존재하면 앞단 Stateflow의 현재 상태에서 candidate_state까지 이동한다. 이동에 성공하여 target_variable을 만족할 경우 그 때의 앞단 Stateflow의 입력을 target_input에 추가한다. 이동에 실패할 경우 현재 상태에서 target_varible을 만족할 수 없으므로 테스트 케이스 생성에 실패한다.

위의 과정을 반복적으로 수행하면 target_input에 모든 Stateflow의 입력이 추가되어 시스템 입력을 구할 수 있다.

Algorithm 3. Pseudo Code of 'find_system_input()' Function

```
k=n+1;
find_system_input(k, target_input)
 if(k==0)
  return target_input;

 Stateflow = Stateflow(k-1);
 config = config Info of Stateflow.current_config;
 state = config.current_state;
 target_variable = find_target_variable(target_input);

if(target_variable==0)
 target_input(config.input_output);
 return find_system_input(k-1, target_input);

if(target_variable>0)
 target_input = run_yices(config, target_input);
 if(target_input .count>0)
   return find_system_input(k-1, target_input);

candidate_state = find_candidate_state(target_variable,
                                  Output Generantion Table);

for c=1:candidate_state.count
 target_input = move_candidate_state
        (state, candidate_state[c], State Matrix Table);
 if(target_input .count>0)
   return find_system_input(k-1, target_input);
```

3) simulation() 함수

simulation()은 시스템 입력이 실제로 테스트 대상을 만족하는지 알기 위해 Yices를 이용해 실행 순서대로 Stateflow들를 Yices형식으로 변환한 SMT 모델을 시뮬레이션 한다. 시뮬레이션 과정은 다음과 같다. 먼저, Stateflow의 Config 정보로부터 현재 config를 얻고 현재 config의 정보와 시스템 입력 system_input을 Yices에 입력하여 run_yices 함수를 사용해 시뮬레이션 한다. 시뮬레이션 후 생성된 변수들은 variable_list에 저장된다. variable_list에는 시뮬레이션 후 Stateflow의 모든 입력, 출력, 내부 변수, 상태 그리고 실행 시간이 들어있다. Create_config 함수는 variable_list를 이용하여 새로운 config를 생성하고 Config 정보에 추가한다. 그리고 Config 정보의 현재 config를 새롭게 생성한 config로 갱신한다. Update_feedback 함수는 variable_list에서 피드백 신호의 정보를 찾아 Feedback_data에 저장한다. Feedback_data에 저장된 정보는 find_target_input()에서 피드백 입력을 고려한 테스트 케이스 생성 시 사용된다. Update_Coverage 함수는 variable_list의 상태 정보를 이용하여 Stateflow의 커버리지를 확인한다. 테스트 대상이 만족된 경우 Stateflow의 커버리지가 증가한다. Algorithm 4는 simuliation()의 흐름을 수도코드로 나타낸 것이다.

Algorithm 4. Pseudo Code of 'simulation()' Function

```
simulation(system_input)
k=1;
for(all Stateflows)
    Stateflow = Stateflow(k);
    config = config Info of Stateflow(k).current_config;
    variable_list = run_yices(config, system_input);
     create_config(config Info of Stateflow(k), variable_list);
    update_feedback(variable_list);
    update_coverage(variable_list);
    k=k+1;
```

3.4 피드백 입력을 고려한 테스트 케이스 생성

Stateflow 모델에서 테스트 케이스 생성 시 피드백 입력이 존재할 경우 반드시 피드백 입력을 고려하여 테스트 케이스를 생성하여야 한다. 피드백 입력을 고려하지 않고 테스트 케이스를 생성할 경우 실제 시스템의 동작을 제대로 반영하지 못하므로 잘못된 테스트 케이스가 생성된다. 예를 들어, 현재 실행 시간이 10 tick이고 현재 상태에서 테스트 대상의 전이 조건이 "[Input1==1&&Input2>2]"라 하자. 이 때, "Input2"는 피드백 입력 신호이고 입력 값의 범위는 0~10이다. 현재 Stateflow에서 테스트 대상을 만족하는 입력은 Yices를 이용하면 "Input1"은 1, "Input2"는 "[Input2>2]"를 만족하는 입력 범위 내의 값을 구할 것이다. 하지만, "Input2"의 경우 피드백 입력이므로 9 tick 때 이미 10 tick에서의 "Input2값이 결정된다. 따라서 "[Input1==1&&Input2>2]"의 조건을 만족하는 테스트 케이스를 구할 때 "Input2"의 값은 이전 tick의 값으로 고정시켜야 한다. 이를 위해 본 논문에서는 Input Range Table의 피드백 정보를 참조하여 피드백 입력에 해당하는 값들을 테스트 케이스를 생성 할 때마다 Feedback_data에 저장하고 이를 테스트 케이스 생성 시 사용하여 잘못된 테스트 케이스가 생성되지 않도록 한다.

4. 실 험

본 논문에서 제안한 테스트 케이스 생성 기법은 1개의 상용 냉장고 제어시스템 모델, 2개의 상용 자동차 ECU 모델을 대상으로 상태, 전이 커버리지 측면에서 평가했다. 평가에 사용된 상용 냉장고 제어시스템 모델, 자동차 Mirror 제어 모델 그리고 IMS 모델의 복잡도는 각각 상, 중 ,하이다. 냉장고 제어 시스템은 크게 입력부, 제어부, 구동부로 나뉘는데 그 중 Main Control Logic 모델은 냉장고 제어시스템의 핵심 제어부의 모델이다. 제어부는 입력부로부터 각종 신호를 입력 받아 구동부로 제어 신호를 보낸다. Mirror는 차량의 사이드 미러(Side Mirror)를 제어하는 ECU의 모델이고 IMS는 차량의 문, 미러 그리고 와이퍼를 관리하는 ECU의 모델이다. 모델에 대한 통계 자료는 Table 1에 나와 있다.

Table 1. Model Statistics

Model	State 수	Transition 수
Main Control Logic	109	295
Mirror	49	135
IMS	34	51

Table 2에는 테스트 생성 기법을 사용하여 생성한 테스트 케이스의 커버리지 결과가 나와 있다. 커버리지 는 상태 커버리지, 전이 커버리지를 기준으로 측정했다. 결과를 확인해보면 생성된 테스트 케이스는 모든 모델에 대하여 상태 커버리지를 100% 만족했고, 전이 커버리지의 경우 Mirror 모델을 제외한 나머지 모델에서 100%의 커버리지를 만족했다. 복잡도가 낮은 모델뿐만 아니라 복잡도가 가장 큰 모델에서도 상태, 전이 커버리지를 100% 만족한 결과는 제안된 테스트 케이스 생성 기법이 모델이 복잡해질수록 심화되는 도달 가능성 문제를 효과적으로 해결했음을 보여준다. [15]의 방법을 사용한 결과보다 성능이 좋음을 확인할 수 있다. ([15]에서는 [15]에서 제안하는 방법이 [14]의 방법보다 성능이 좋음을 주장하고 있다.)

Table 2. Coverage Statistics

Model	State Coverage (%)		Transition Coverage (%)	
	[제안]	[15]	[제안]	[15]
Main Control Logic	100	90	100	87
Mirror	100	82	79	60
IMS	100	85	100	73

Mirror 모델의 경우 Main Control Logic 모델보다 복잡도가 더 떨어졌지만, 전이 커버리지는 79%로 오히려 더 낮게 측정되었는데 그 이유는 피드백 입력 신호 때문이다. Mirror 모델에는 20개가 넘는 피드백 입력이 존재했는데 피드백 신호가 많을 경우 테스트 케이스 생성의 어려움이 존재한다. 예를 들어 설명하면 다음과 같다.

Fig. 5는 피드백 신호가 존재하는 시스템의 예이다. A, B, C는 각각의 Stateflow를 의미하고 C의 출력 'Co_1'이 피드백 되어 B의 입력 "Bi_1"으로 들어간다. 현재 B 내의 전이들을 대상으로 테스트 케이스를 생성하고 있고 B에서 현재 상태는 "BS_1"이라 하자. 그리고 현재 실행 시간은 5 tick이고 4 tick에서의 "Co_1"은 0라고 가정하자. "BS_1"에서 테스트 대상은 "BS_2"으로 가는 전이가 될 것이다. 해당 전이의 전이 조건을 보면 "[Bi_1==1&&Bi_2==1]"이다. 전이 조건을 만족하기 위해서는 현재 5tick에서 "Bi_1"이 1의 값을 가져야하지만 5 tick에서의 "Bi_1"의 값은 이미 4 tick에서의 "Co_1"의 값 0로 결정되기 때문에 5 tick에서 테스트 케이스 생성 시 제어할 수 없는 입력 변수가 된다. 따라서 5 tick에서 해당 전이를 만족하는 테스트 케이스는 구할 수 없게 된다. 본 논문에서 제안한 테스트 생성 기법은 피드백 입력의 값을 매 실행 시간마다 따로 저장하여 테스트 케이스 생성 시 이용하지만 Fig. 7과 같은 경우에는 테스트 케이스를 생성할 수 없다. 테스트 케이스 생성에 실패한 경우 모델 분석을 통한 휴리스틱 기법을 사용해서 테스트 케이스를 생성해야 한다.

상용 도구인 TCG [21]를 사용하여 실험에 사용한 모델에서 테스트 케이스를 추출한 후 제안된 기법을 사용하여 생성한 테스트 케이스와 커버리지를 비교하였다. 각 모델에서 상태 커버리지는 약 90%(IMS)~83%(Mirror)를 보였고, 전이 커버리지는 약 81%(IMS)~74%(Mirror)를 보여, 제안하는 방법이 우수함을 알 수 있었다. 이러한 결과는 TCG에서도 피드백 값을 적절히 처리하지 못하기 때문에 발생하는 문제라고 생각된다. 일반적으로 모델 기반 테스트 케이스 생성 도구들은 무작위로 다양한 테스트 케이스를 생성하고 커버리지를 측정하기 때문에 피드백을 효과적으로 처리하지 못한다. 물론 이 커버리지 값은 모델에 따라 다른 결과를 낼 수 있다.

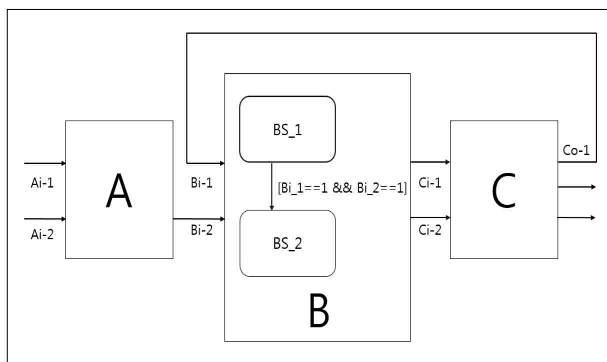


Fig. 7. An Example Model with a Feedback

5. 결 론

본 논문에서는 Stateflow 모델 기반 테스트 케이스 생성 기법을 제안한다. 모델 기반 테스트 케이스 생성에서 발생하는 도달 가능성 문제를 해결하기 위해 SMT Sovler의 한 종류인 Yices와 모델 정보를 사용했다. 테스트 케이스 생성에 사용되는 정보들은 SL/SF 모델을 분석하여 생성하였다. 모델 정보는 Yices 정보, Config 정보 그리고 User Define 정보로 나눠고 테스트 케이스 생성 알고리즘에 사용된다.

제안된 테스트 케이스 생성 기법의 성능은 테스트 목표를 상태 커버리지와 전이 커버리지로 설정하여 3가지 상용 모델을 대상으로 평가하였다. 모델의 복잡도가 커져도 높은 테스트 케이스 생성 효율을 보였지만, 피드백 입력 신호가 많은 모델의 경우 테스트 케이스 생성 효율이 떨어지는 결과를 보였다. 향후 피드백 문제를 해결하기 위한 휴리스틱 기법 연구가 필요할 것이다.

제안하는 방법은 상용도구와 비교하여 커버리지가 더 높은 테스트 케이스를 생성하였다. 이 결과는 모델에 따라 달라질 수 있고 일반화할 수는 없지만, 제안하는 방법의 효용성을 충분히 보여주는 것이라 하겠다.

References

[1] L. H. Tahat, B. Vaysburg, B. Korel, and A. J. Bader, "Requirement-based automated black-box test generation," in *Proceeding of the 25th Annual International Computer Software and Applications Conference COMPSAC*, pp.489-495, 2001.

[2] MATLAB Simulink Stateflow [Internet], http://www.mathworks.com/products/stateflow. (The MathWorks, Inc.)

[3] Lutz Köster, Thomas Thomsen, and Ralf Stracke, "Connecting Simulink to OSEK: Automatic Code Generation for Real-Time Operating Systems with TargetLink," *Society of Automotive Engineering,* pp.2001-01-0024, 2001.

[4] Auto Code Generation [Internet], http://www.reactive-systems.com/papers/bcsf.pdf. (October 19, 2013.)

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

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

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

[9] 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 NASA softwrae," in *Proceedings of the ISSTA*, pp. 15-25, 2008.

[10] H. S. Hong, I. S. Lee, O. Sokolsky, and S. D. Cha, "Automatic Test Generation From Statecharts Using Model Checking," in *Proceedings of FATES'01, Workshop on Formal Approaches to Testing of Software, BRICS Notes Series*, Vol.NS-01-4, pp.15-30.

[11] K. L. McMillan, "Symbolic Model Checking - an Approach to the State Explosion Problem," Ph.D. dissertation, Carnegie Mellon University Pittsburgh, PA, USA, 1992.

[12] 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, Issue 1, pp.94-124, 1998.

[13] H. S. Park, "Generating Structural Test Cases of Simulink/Stateflow Model Based on RRT Algorithm Using Heuristic Input Analysis," *Korea Information Processing Society*, Vol.2, No.12, pp.829-840, 2012.

[14] S. M. Seo, "Test Case Generation for Simulink Stateflow Model using SMT Solver," M.S. dissertation, Ajou University, Suwon, Korea, 2014.

[15] J. W. Kim, "Simulink/Stateflow Model Based Test Case Generation using a Decomposition Approach," M.S. dissertation, Ajou University, Suwon, Korea, 2015.

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

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

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

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

[20] L. de Moura, "SMT Sovlers," 2006.

[21] R-Bench TE [Internet], http://www.btstech.co.kr/page_vaHk97.

박 한 곤

e-mail : wkfksla@ajou.ac.kr

2009년 아주대학교 전자공학부(학사)

2017년 아주대학교 전자공학과 석사

관심분야 : 임베디드 시스템 테스팅, 실시간 시스템

정 기 현

e-mail : khchung@ajou.ac.kr

1984년 서강대학교 전자공학과(학사)

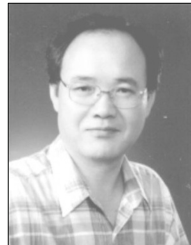
1988년 미국 Ilinois주립대 EECS(석사)

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 소프트웨어학과 교수

관심분야 : 운영체제, 분산시스템, 실시간/멀티미디어 시스템