KIPS Tr. Software and Data Eng.
Vol.5, No.12 pp.641~652 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2016.5.12.641

A Method to Automatically Generate Test Scripts from Checklist for Testing Embedded System

Tae Hoon Kang† · Dae Joon Kim† · Ki Hyun Chung†† · Kyung Hee Choi†††

ABSTRACT

This paper proposes a method to generate test scripts in an automatic manner, based on checklist used for testing embedded systems in the fields. The proposed method can reduce the mistakes which may be introduced during manual generation. In addition, it can generate test scripts to test various mode combinations, which is not possible to be tested by the typical checklist. The test commands in a checklist are transformed into a test script suit referencing the signal values defined in a test command dictionary. In addition, the method to generate test scripts in sequential, double permutation and random manners is proposed useful to test the inter-operations between modes, a series of operations for a specific behavior. The proposed method is implemented and the feasibility is shown through the experiments.

Keywords : Embedded System, Test, Test Script, Checklist

임베디드 시스템 테스팅을 위한 체크리스트로부터 테스트 스크립트 자동 생성 방안

강 태 훈† · 김 대 준† · 정 기 현†† · 최 경 희†††

요 약

본 논문은 임베디드 시스템 테스트를 위해 산업현장에서 많이 사용하는 체크리스트를 기반으로 테스트 스크립트를 자동으로 생성하기 위한 방법을 제안한다. 제안하는 방법은 수동 생성에서 발생할 수 있는 오류를 줄일 수 있을 뿐만 아니라, 기존의 체크리스트로는 테스트하지 못하는 다양한 모드 조합을 테스트하기 위한 테스트 스크립트도 생성할 수 있다. 체크리스트에 있는 테스트 명령어는 테스트 명령어 사전에 정의된 신호 값을 참조하여 테스트 스크립트로 변환된다. 또한, 체크리스트를 정의된 일련의 연관된 동작의 집합인 모드들 간의 동작을 확인할 수 있게 하는 순차적, Double permutation 및 무작위 방법으로 테스트 스크립트를 생성할 수 있는 방법을 제안한다. 제안된 방법은 구현되었고, 실험을 통해 그 가능성을 보여준다.

키워드 : 임베디드 시스템, 테스트, 테스트 스크립트, 체크리스트

1. 서 론

오늘날 임베디드 시스템은 모바일 기기, 가전제품, 군사용 무기 등 다양한 분야에서 사용되고 있으며 과거에 비해 점점 더 복잡한 기능을 수행하고 있다. 이에 따라 시스템의 기능을 수행하는 소프트웨어 또한 복잡해지고 있다. 임베디드 시스템이 동작하면서 발생하는 오류의 많은 부분은 소프트웨어적인 오류이며[1], 항공기 또는 군사용 무기와 같이 중요한 임무를 수행하는 시스템에서 오류 발생 시 큰 재앙을 불러오거나 주어진 임무를 올바르게 수행할 수 없을 것이다. 이렇게 복잡해진 임베디드 시스템의 소프트웨어 테스트는 선택이 아닌 필수 사항이 되고 있다. 이러한 임베디드 소프트웨어 신뢰도 향상을 위해 많은 테스트 기법들이 존재한다.

임베디드 소프트웨어 테스트 기법에는 시스템의 내부 코드를 직접적으로 테스트하는 화이트 박스(White Box)와 코드가 아닌 외부의 기능을 중점적으로 테스트 하는 블랙박스(Black Box) 기법이 있다. 화이트 박스는 소스 코드를 확보하여 코드의 흐름에 맞는 테스트 케이스를 작성하고 코드가 설계 사양에 맞게 올바르게 동작하는지 혹은 코딩 규칙을 제대로 지키고 있는지 등을 확인하는 용도로 사용한다. 반

※ 본 연구는 방위사업청(UD150042AD)의 지원으로 수행되었음.
† 준 회 원 : 아주대학교 컴퓨터공학과 석사과정
†† 정 회 원 : 아주대학교 전자공학과 교수
††† 정 회 원 : 아주대학교 컴퓨터공학과 교수
Manuscript Received : April 21, 2016
First Revision : July 1, 2016
Accepted : July 1, 2016
* Corresponding Author : Ki Hyun Chung(khchung@ajou.ac.kr)

면 블랙박스는 시스템이 요구사항에 나타난 기능을 적절히 수행하고 있는지를 중점적으로 테스트하기 때문에 다양한 동작을 확인할 수 있도록 효율적인 테스트 케이스를 작성하는 것이 중요하다. 본 논문에서는 블랙박스 테스트 케이스 생성에 대해 다룬다.

블랙박스 테스트를 위한 테스트 케이스 생성 방법은 많은 방법들이 사용되고 있다. 요구사항 모델을 이용하는 테스트 케이스 생성 방법, 시스템 입력들의 필요에 따라 조합하여 사용하는 다양한 입력조합(Combinatorial) 생성 방법 혹은 여러 형태의 무작위 조합(Random) 방법 등이 있다. 이와 같은 체계적인 생성 방법 이외에 산업 현장에서는 개발 시스템에 대한 엔지니어의 지식, 과거의 경험, 사용자 피드백 등을 활용하여 테스트하고자 하는 항목을 정리하여 테스트에 사용하는 체크리스트(Checklist)를 이용하는 방법이 흔히 많이 사용되고 있다.

체크리스트란 테스트 대상의 기능을 점검하기 위해 테스트해야 하는 내용과 조건을 나열해 놓은 목록이다. 일반적인 체크리스트는 테스트해야 하는 내용이 나열되어 있지만 테스트 대상 시스템에 실제 입력 값을 인가하는 데이터는 테스터(Tester)가 체크리스트를 참조하여 수작업으로 결정한다. 이러한 방법은 크게 세 가지 결점이 있다. 1) 테스터의 실수 또는 주관적인 생각에 따라 결과가 달라질 수 있다. 2) 체크리스트에 작성된 항목이 많다면 테스트 케이스를 만들기가 힘들다. 3) 체크리스트 항목이 실제 테스트 케이스와는 형식이 많이 다르기 때문에 체크리스트 항목을 테스트 케이스로 만들기 위해서는 시스템 입력 조건 등을 고려해야 하고 이는 사람이 분석하기엔 많은 노력과 시간이 필요하다.

본 논문에서는 이러한 단점을 보완할 수 있는 체크리스트의 체계를 구축하고, 테스트 스크립트(테스트 케이스를 실제 시스템 적용하기 위해 물리적인 값으로 변환한 형태로 본 논문에서는 테스트 케이스와 혼용해서 사용한다)로 변환하기 위해 필요한 정보가 작성된 테스트 명령어 사전(Test Command Dictionary)을 정의하여 체크리스트로부터 자동으로 테스트 케이스를 도출하는 방법을 제안한다. 또한 정의된 일련의 연관된 동작의 집합인 모드들 간의 동작을 확인할 수 있게 하는 순차적 방법, 동작 모드 조합에 따른 정확성 여부를 테스트할 수 있는 Double Permutation 및 무작위 방법으로 테스트 스크립트를 생성할 수 있는 방법을 제안한다.

본 논문의 구성은 다음과 같다. 제 2장은 블랙박스 테스트를 위한 테스트 케이스 생성 방법에 대한 관련 연구를 소개하고, 제 3장은 체크리스트 체계와 테스트 명령어 사전 구조 및 테스트 케이스 생성 방법을 제시한다. 제 4장은 논문에서 제시하는 내용을 바탕으로 실험 과정 및 결과에 대해 기술한다. 그리고 결론에서는 앞으로 진행되어야 하는 연구방향을 기술하고 논문을 마무리한다.

2. 관련 연구

테스트 케이스 생성 방법으로는 소스 코드를 분석하여 테스트 케이스를 생성하는 화이트 박스(White box) 테스트 케이스 생성과 시스템 수준의 입출력을 바탕으로 테스트 케이스를 생성하는 블랙박스(Black box) 테스트 케이스 생성 법으로 크게 나눌 수 있다.

화이트 박스 테스트 케이스는 소스 코드 수준에서 코드의 실행 오류(Run time error) 등을 파악하기 위해 설계 단계에서 많이 사용한다. 하지만 소스 코드를 획득하기 어렵거나 하드웨어에 내장된 시스템의 동작이 시스템 사양과의 일치 여부를 검증하는데 사용하기는 어렵다.

블랙박스 테스트 케이스는 소스 코드가 아니라 완성된 소프트웨어 자체를 대상으로 하여 테스트 하는데 사용된다. 특히, 임베디드 시스템과 같이 하드웨어에 내장된 소프트웨어가 요구사항 대로 동작하는 지를 테스트하는 경우에 사용된다. 목표에 부합되는 입력 생성의 어려움은 있으나, 설계 문서와 같은 시스템의 구현 상세 정보 없이 테스트 케이스를 만들 수 있는 장점도 있다.

다양한 블랙박스 테스트 케이스 생성 방법이 연구되어 왔다. 대표적으로는 무작위(Random) 생성 및 변형된 무작위 생성 방법, 시스템 입력 조합(Combinatorial)을 이용하는 생성 방법 및 모델 기반(Model-based) 생성 방법[17] 등이 있다. 무작위 생성에서는 테스트 대상 제품(이하 SUT: System Under Test)에 대한 내부 정보를 대부분 사용하지 않기 때문에, 사용자의 무작위 동작과 같은 예기치 못한 다양한 입력에 대한 SUT의 평가에 유용할 수 있으나, 순차적인 동작 흐름과 같은 기능을 테스트하기에는 부적절하다. 이러한 비효율적인 요소를 줄이기 위하여 다양한 적응형 무작위 생성(Adaptive Random Generation)방법들이 제시되었다[4]. 입력 조합에 의한 생성 방법은 시스템의 입력 정보를 테스트 목적에 맞게 조합하여 테스트 케이스로 사용하는 방법이다. 페어와이즈(Pairwise)[20] 및 클래스 분류 트리 방법(Classification Tree Method)[10] 등이 이에 속한다고 할 수 있다. 모델 기반 생성 방법은 시스템의 요구사항 혹은 기능을 모델화하고 모델로부터 테스트 케이스를 생성하는 방법이다.

요구사항을 기반으로 테스트 케이스를 생성하는 다양한 방법도 있다. 동등 분할(Equivalence partitioning)[21] 기법이나 경계 값 분석을 통한 생성 방법(Boundary value analysis) [21] 등이 이에 속한다.

이외에도 변이 값(Mutation) 주입을 통한 테스트 방법(Mutation)[21] 및 유한 상태 머신(Finite state machine)을 이용한 방법[26] 등도 있다.

위와 같은 구조적인 테스트 케이스 생성 방법과 더불어 자동차 전장과 같은 실제 산업 분야에서는 엔지니어의 경험을 기반으로 테스트 케이스를 생성하여 사용한다. 설계 및 테스트 과정에서의 경험이나, 사용자 오류 보고 등의 오류 데이터베이스를 활용하여 테스트 케이스를 생성하는 것이다. 이와 같은 경험을 기반으로 생성한 테스트 케이스는 구조적 방법으로 생성된 테스트 케이스가 찾아내지 못하는 시스템의 특성에 따른 오류를 발견할 수 있는 장점이 있다.

경험 기반으로 테스트 케이스를 생성하기 위해 많이 사용하는 것이 체크리스트이다. 체크리스트는 테스트 엔지니어가 다루었던 유사한 시스템이나, 과거의 경험, 직관, 테스트 엔지

니어의 지식, 사용자의 피드백을 활용하여 테스트 하고자 하는 내용을 포함한다.

우리가 아는 지식으로는 체크리스트를 활용하여 체계적으로 테스트 케이스를 생성하는 방법은 없다. 유사한 방법으로 유스 케이스(Use case)를 활용한 테스트 시나리오 생성법과 FT(Fault tree)를 활용한 방법이 있다.

[23]의 저자들은 유스 케이스를 활용하여 사용자 관점에서 바라본 시스템의 동작을 표현한 시나리오 생성 방법을 제시하고 있다. 이 방법에서는 유스 케이스 다이어그램(Use case diagram)을 활용하여 유스 케이스 모델을 만들고, 모델링을 통해서 결정된 각 유스 케이스에 대한 처리 내용을 구체적으로 정의한 유스 케이스 명세서를 작성한다.

유스 케이스 다이어그램은 액터(Actor), 유스 케이스, 관계(Relation)로 구성된다. 액터는 시스템 외부에서 시스템과 상호작용을 하는 모든 것을 나타내며, 주로 사용자를 나타내고 경우에 따라서는 시스템 혹은 특정 장치를 나타낼 수 있다. 유스 케이스는 시스템이 제공해야 하는 기능을 나타내고, 이러한 유스 케이스가 모여 시스템 전체의 기능을 나타낸다. 관계는 액터와 유스 케이스 사이의 연관관계를 나타내며, 관계된 요소들 사이의 연결은 선으로 표시한다.

유스 케이스 명세서 구성은 크게 시나리오 흐름과 조건으로 나누어진다. Fig. 1은 시나리오 흐름과 조건을 표현한 그림이다[25].

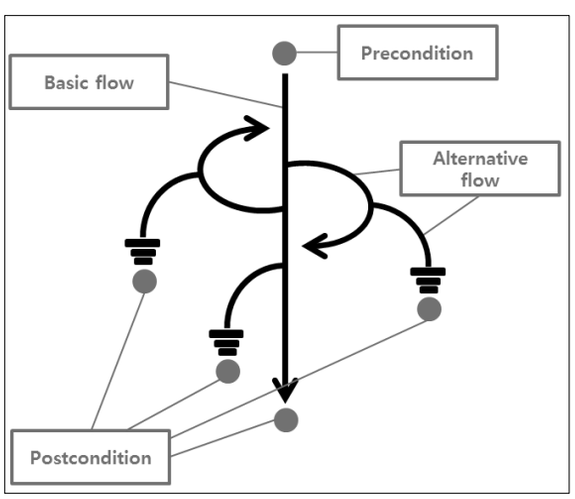


Fig. 1. Basic Flow of Events and Alternative Flows of Events for a Use Case

시나리오 흐름은 시스템이 사용자의 요구 기능을 정상적으로 수행하는 기본흐름(Basic flow)과 요구 기능의 수행도중 실패할 경우에 적절히 대체해야할 대체흐름(Alternative flow)로 구성된다.

시나리오 조건은 유스 케이스의 수행이 시작되기 위한 선행조건(Precondition)과 수행이 완료된 후에 만족되어야 하는 후행조건(Postcondition)으로 구성된다. 유스 케이스 명세서는 앞서 설명한 구조를 기반으로 Table 1과 같이 구성할 수 있다.

Table 1. Use Case Description

Section	Description
Name	An appropriate name for the use case.
Actors	A list of actors associated with the use case.
Precondition	A precondition is the state of the system and its surroundings that is required before the use case can be started.
Postcondition	A postcondition is the state the system can be in after the use case has ended.
Basic flow	The basic flow is the description of the normal, expected path through the use case.
Alternative flow	If the basic flow represents the normal route to success, the alternative flows can be considered as detours.

FT는 시스템고장을 발생시키는 원인들과 관계를 논리기호를 사용하여 트리 형태로 표현하는 방법이다[27]. FT는 원인 이벤트에 의해 결과 이벤트가 발생하는 구조를 가진다. FT를 활용하면 결과 이벤트를 오류 발생 상황이라 가정하고 이와 같은 상황이 발생할 수 있는 테스트 케이스를 생성할 수 있다.

오류 발생의 경로를 거슬러 올라가 나무 모양의 트리 형태로 전개함으로써 발생 원인을 해석한다. 상위 이벤트를 결과 이벤트(event)라 하면 하위 이벤트를 원인 이벤트로 구성한다. 각 원인 이벤트와 결과 이벤트 사이는 AND, OR, Priority AND, XOR 등의 다양한 논리기호로 표현한 논리게이트로 연결하여 트리 구조를 만든다.

3. 체크리스트 기반 테스트 스크립트 생성

앞에서 언급한 여러 가지 테스트 케이스 생성 방법은 각 방법 나름대로의 특징을 가진다. 이와 같은 체계적인 테스트 케이스 생성 방법과 더불어 엔지니어의 경험이나 오류 보고 등을 활용한 체크리스트가 산업 현장에서 많이 활용되고 있는 것이 현실이다. 앞 절에서 언급한 어떠한 테스트 케이스 생성 방법에서도 체크리스트를 활용하여 테스트 케이스를 생성하는 방법에 대해서 구체적으로 다루고 있지는 않다. 본 논문에서는 체크리스트를 활용하여 보다 효율적이고 다양한 테스트 케이스를 체계적으로 생성할 수 있는 방법에 대해 다룬다.

3.1 테스트 스크립트 자동 생성을 위한 체크리스트

체크리스트는 테스트 엔지니어가 어떤 시스템을 테스트하는데 있어서 확인하고자 하는 기능이나 주요 동작을 기술한 문서이다. 테스트 엔지니어는 체크리스트에 포함하고 있는 항목을 이용하여 시스템의 정상 동작 여부를 판단한다. 체크리스트는 일반적으로 테스터의 경험에서 얻어진 다양한 경우 및 흔히 발생할 수 있는 시스템 오류를 점검하기에 적합한 기능들의 정상 동작 여부를 점검할 수 있는 내용을 기술한다. 체크리스트를 바탕으로 실제 테스트를 수행하기 위해서는 시스템에 입력될 테스트 스크립트가 필요하다.

일반적으로 경험 기반 기법의 체크리스트는 체계적으로

동작모드	기능분류	기능설명	테스트 명령
특수모드1	진입시조건	주 에러 발생 시 특수모드1 투입되지 않음	정상
	주동작	F/C-Fan ㅣ 고속 운전	7200초
		Display All On 확인	Display_All_On
	종료조건	투입 상태에서 특수모드 투입 버튼을 1회 누르면 특수모드 2로 전환	Switch ON
		주 에러(D_Sensor) 발생 시 Test 해제 후, 해제 후 동작 이후 Test Res	D-ERR
		주 에러(F_Sensor) 발생 시 Test 해제 후, 해제 후 동작 이후 Test Res	F-ERR

Fig. 2. A Checklist Example

도출되기 보다는 테스트 경험에 의해 테스트 하고자하는 내용을 목록으로 정리하고 다음번 테스트에서 해당 내용을 누락 없이 재활용 하는 것을 목적으로 작성한다. 이러한 체크리스트 형식은 테스트 엔지니어가 이해할 수 있는 언어로 작성된다. 테스트 엔지니어는 체크리스트에 작성된 목록을 보고 무엇을 테스트해야 하는지는 쉽게 파악할 수 있지만, 이것을 시스템이 동작하기 위한 데이터로 만드는 것은 어렵다.

또한 체크리스트를 기반으로 테스트 엔지니어가 수작업으로 테스트를 진행할 때, 테스트 엔지니어의 판단 오류에 따른 부정확한 테스트, 많은 동작을 수작업으로 진행함으로써 발생하는 테스트 시간 과다, 테스트 정보의 기록 오류, 예기치 못한 오류 가능성이 있는 모든 테스트 목록에 대한 작성 등의 어려움도 생긴다.

산업현장에서 일반적으로 사용하는 체크리스트는 통일된 양식이 없고 목적에 맞게 양식을 만들어 사용한다. 테스트 엔지니어는 이러한 양식으로부터 테스트 하고자하는 내용을 참조하여 시스템을 동작시키기 위한 테스트 스크립트를 만든다. 이처럼 기존에는 테스트 스크립트를 테스트 엔지니어가 체크리스트를 바탕으로 수작업으로 작성하였지만 수작업에 따른 체크리스트 기반 테스트의 단점을 극복하기 위해 본 논문에서는 먼저 일반적으로 사용하는 체크리스트의 항목들을 정리하고 체계화하여 테스트 스크립트를 자동 생성할 수 있는 체크리스트를 제안한다.

테스트 스크립트를 자동으로 생성하기 위해서는 효율적으로 체크리스트를 체계화하는 것이 필요하다. 제안하는 체크리스트 체계화 방법은 유스 케이스 명세서 작성에 기반이 되는 시나리오 흐름 및 조건에 대한 방법을 참조하여 체계적인 구조를 만든다.

제안하는 체크리스트는 Fig. 2와 같다. 체크리스트는 “동작모드”, “기능분류”, “기능설명”, “테스트 명령”으로 이루어져 있다. 이들 중 “동작모드”, “기능설명” 항목은 실제 테스트 스크립트 생성과는 무관한 항목이며 테스트 엔지니어에게 테스트 정보를 알려주는 용도로 사용된다.

“동작모드”는 동작 모드 명을 적는 항목이다. 예를 들면, 대기 모드, 동작 모드 등 고유의 기능 전체를 하나의 독립적인 동작 그룹이 분류될 수 있는 동작 모드 이름이다. 따라서 다른 분류 조합으로 SUT의 다른 동작을 테스트할 수 있다.

“기능분류”는 각 동작 모드 내에서 수행하는 기능을 말하며 “진입시조건”, “종료조건” 그리고 “주동작” 항목으로 구성된다. “진입시조건”은 해당 모드에 진입하기 위하여 만족시켜야하는 조건이다. “진입시조건”은 일부 혹은 전부 만족시켜야 “주동작”으로 진입할 수 있다. “주동작”은 항상 실행되어야하는 일련의 작업들이다. “종료조건”은 해당 모드를 종료하기 위한 사건이나 조건들이다. “진입시조건”과 같이 일부 혹은 전부 만족할 경우 해당 모드를 종료한다. 따라서 각 모드는 진입시조건->주동작->종료조건으로 수행된다. “기능설명”은 각 기능이 수행해야 하는 내용을 설명한다. “테스트 명령”란은 체크리스트 해당 열의 기능을 테스트하기 위한 실제 테스트 명령어명 (커맨드라 부른다)을 입력하는 곳이다.

3.2 테스트 명령어 사전(Test Command Dictionary)

테스트 명령어 사전은 테스트 명령에 대한 입력 값 정보를 정의한 문서이며 텍스트(Text) 형식이나 엑셀(Excel) 파일 등으로 만들어 질 수 있다.

테스트 명령어 사전은 테스트 명령어들의 집합이다. SUT가 수행해야 할 단위 기능인 테스트 명령어에는 각 시간대별로 SUT에 인가되는 모든 입력들의 물리적 값들이 정의되어 있다. 다음 Table 2는 N개의 입력 신호(i1, i2,...,iN)을 가진 SUT에 테스트 명령어가 실행된 순간부터 5msec 마다 가해지는 입력 신호들 값으로 구성된 테스트 입력의 예이다. 이 경우 실제 SUT 테스트를 위해서는 N개의 입력이 모두 필요하다.

Table 2. A Test Input Example

Time (msec)	SUT input signal value			
	i1	i2	...	iN
0	2	1.5		T
5	3	4.1		F
10	2	4.4		T
…				
30	4	3.7		T

시간은 입력을 인가해야하는 시간 단위로 필요에 따라 정의할 수 있으며, 열 간의 시간 간격은 입력이 가해지는 상대 시간이다. 입력 신호는 사전에 정의한 대로 소수, 정수, 참/거짓 값 등이 사용될 수 있다.

하지만 실제 테스트 명령어에서는 모든 입력 값을 다 정의하는 것이 아니라 이전 값에서 변화된 값만을 정의한다. (나머지 테스트 명령어에서 정의되지 않는 입력 신호는 이전의 값을 유지한다) 모든 시스템 초기 입력 값은 사전에 정의한다.

Fig. 3에서는 어떤 SUT 테스트를 위한 세 개의 테스트 명령어 'A_To_D', 'D_To_A' 그리고 'A센서ERR'를 보여주고 있다. 해당 SUT는 5개의 입력 신호 A_sensor, B_sensor_N1, B_sensor_N2, D_sensor_N1, D_sensor_N2를 가지고 있다.

5개의 입력 신호 초기 값이 -70, 0, 0, 40, 40 으로 설정되었다면, 테스트 명령어 'A센서ERR'은 다음 Table 3과 같은 입력 신호를 테스트 명령어 수행 시작 시점(Time = 0)부터 20, 70, 80, 500sec에 SUT에 입력됨을 의미한다.

Sensor 관련 명령어

명령어	group	TIME	D_Sensor_N1
A_To_D	1	0	43
		5	43
		10	41
		11	

명령어	group	TIME	D_Sensor_N2
D_To_A	1	0	44
		10	45
		15	47

명령어	group	TIME	A_Sensor	D_Sensor_N1	B_Sensor_N1	B_Sensor_N2
A센서 ERR		0	-70			
		20	-15	45		
		70	-29	45	-5	-5
		80	-15	10	15	15
		500				

Fig. 3. A Test Command Example

'A센서ERR' 명령에 대한 테스트 스크립트는 테스트 명령어 'A센서ERR' 실행 시점(0 sec)에서 입력 된 'A_sensor'의 값 '-70', 나머지 B_sensor_N1, B_sensor_N2, D_sensor_N1, D_sensor_N2의 신호 값은 초기 신호 값 들을 20초 동안 유지한다. 20초 후에 'A_sensor'의 값은 '-15', 'D_sensor_N1'의 값 '45'로 변경시켜 나머지 신호 값은 이전 상태를 유지한 채, 다음 입력이 오는 시점인 70초까지 50초 동안 유지한다. 이후 'B_sensor_N1'와 'B_sensor_N2' 값을 '-5'로 변경 입력하고 'A_sensor'의 값을 '-29'로 변경하여 다음 입력이 오는 80초 까지 10초 동안 유지한다. 같은 방법으로 80초에 대한 입력을 500초 까지 420초 동안 유지하여 'A센서ERR' 명령어에 대한 테스트 스크립트 생성을 완료한다. Table 3은 실제 입력되는 각 변수 값들을 보여주고 있다.

Table 3. Input Signal Values of Test Command 'A센서ERR'

Time (sec)	SUT input signal value				
	A_sensor	B_sensor _N1	B_sensor _N2	D_sensor _N1	D_sensor _N2
0	-70	0	0	40	40
20	-15	0	0	45	40
70	-29	-5	-5	45	40
80	-15	15	15	10	40
500	-15	15	15	10	40

3.3 테스트 스크립트 생성 방법

테스트 케이스를 생성하기 위한 기본 구조는 Fig. 4와 같다. 체크리스트가 수행되면 체크리스트의 명령어를 위에서부터 순차적으로 수행한다. 명령어에 대한 입력은 명령어 사전(CMD Dictionary)에 정의되어 있는 명령어에 대한 입력 시퀀스에 따라 실행된다. 테스트 명령어 사이의 실행 시간 간격은 미리 설정된 시간 간격을 따른다. 즉, Fig. 4와 같이 테스트 스크립트 생성은 체크리스트에 있는 테스트 명령어(Checklist), 명령어 사전(CMD Dictionary) 및 순서지정파일(Sequence File: 용도는 다음 절에서 설명)을 참조하여 테스트 스크립트 생성기가 하게 된다.

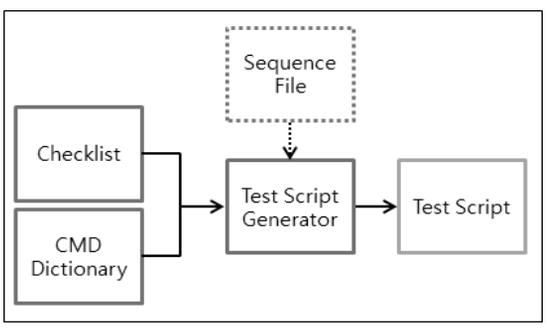


Fig. 4. Test Script Generation Flow

체크리스트의 테스트 명령은 테스트 명령어 사전에 반드시 존재해야하며, 해당하는 테스트 명령은 테스트 명령어 사전에서 검색하여 일치하는 명령어의 신호 값을 정의된 시간에 맞추어 테스트 스크립트를 생성한다.

"진입시조건"과 "종료조건"은 하나 이상의 조건으로 구성될 수 있다. 그리고 그 조건들은 동시 만족되어야 할 수도 있고 일부만 만족되어도 될 수도 있다. 어떠한 경우든지 조건들이 가능한 조합으로 조건의 만족 여부를 보아야 할 필요가 있다. 예를 들면 "진입시조건"이 "C = (C1 and C2 and C3)"라면 C가 만족되는 경우도 보아야겠지만, C가 만족되지 않을 때도 테스트 하여야 할 것이다. 즉, 여기서 C 조건 만족 여부는 소스 코드 기반의 화이트 박스 테스트에서 조건문 테스트에서 적용되는 "조건 커버리지(condition coverage)", "결정 커버리지(decision coverage)" 혹은 "변형된 조건 결정 커버리지(MCDC: Modified Condition Decision Coverage)"[21]를 만족하는 조합으로 구성될 수 있다. 테스트 스크립트 생성기는 생성 정책에 따라 커버리지(coverage) 기준을 적용하여 테스트 스크립트를 생성한다.

3.4 체크리스트 기반 테스트 조합 생성

앞에서 정의한 동작 모드를 조합하여 다양한 테스트 스크립트를 생성할 수 있다. 하나의 동작 모드는 특정 동작을 위한 일련의 기능이다. 다양한 환경 하에서 SUT의 정상 동작 여부를 테스트하기 위해서는 다양한 모드 조합으로 만들어진 테스트 스크립트가 필수적이다.

동작 모드의 순서에 조합에 따라 시스템이 도달하는 상태가 다르고 이에 따른 시스템의 동작 오류를 파악하여야 할 필요가 있다. 따라서 본 논문에서는 동작 모드의 조합을 각 모드의 고유 기능 테스트에 집중하는 순차적 방법, 기존의 Pairwise 및 Triple pair 방법[20]에서 사용하는 방법과 같은 개념의 조합인 Double Permutation과 Triple Permutation

방법 그리고 무작위 조합에 따른 동작 확인을 위해 무작위 조합을 사용하는 방법을 제시한다.

모드 동작을 이루고 있는 테스트 명령어의 다양한 조합도 가능하지만, 테스트 명령어의 조합에 따라서는 SUT가 도달할 수 없는 상태가 되기 때문에 본 논문에서는 동작 모드의 조합에 의한 테스트 생성 조합 방법 만에 대해 다룬다.

1) 순차적 방법

순차적 방법은 다양한 동작 모드를 필요에 따라 적절히 배치하여 테스트 케이스를 생성하는 방법을 말한다. 이 방법은 주로 각 모드 내의 기능 동작이 제대로 실행되는 지를 테스트하는 용도로 사용되며, 따라서 테스터의 경험 기반의 시나리오를 작성하여 생성한다. 동작 모드 내부의 조건에 따라 동작이 달라지는 시험 조건에 대하여 중점적으로 테스트 할 수 있도록 조합을 생성한다. 순차적 방법은 체크리스트에 작성된 동작 모드를 기본 단위로 하여 시나리오에 따라 동작 모드를 순차적으로 나열하여 조합을 생성한다. 각 동작 모드의 내부 로직은 “진입시조건”, “주동작” 그리고 “종료조건”의 순으로 수행되며 각 기능에 대한 테스트 명령을 생성한다.

모드를 순차적으로 생성하기 위해서 간단한 문법을 정의하여 동작모드 간 연관관계를 표현할 수 있도록 하였다. 정의된 문법을 이용하여 순서지정파일(Sequence File)을 작성하고, 모드 조합을 순서지정파일에 정해진 순서대로 테스트 케이스를 생성한다. 정의된 문법은 {‘/’, ‘_’, ‘;’}의 3 가지 기호를 통하여 의미를 지니도록 하였다. 모드 간의 연계는 ‘/’로 나열하며 ‘;’ 기호는 하나의 문법의 완료를 의미한다. 마지막으로 ‘_’ 명령어는 해당 모드 안에서 특정한 종료조건 혹은 진입조건을 지정한다.

```
“초기모드/일반모드”(*);
“동작모드A_DERR/특수모드#1”;
```

Fig. 5. A Sequence File Example

Fig. 5는 문법을 이용하여 순서지정파일을 작성한 예시이다. “초기모드/일반모드”는 초기모드와 일반모드를 순차적으로 수행되어야 함을 의미한다. 다음 순차적으로 “동작모드A”와 “특수모드#1”가 수행된다. 단 “동작모드A_DERR”는 동작모드A가 종료할 때 종료될 수 있는 여러 가지 조건 중에서, DERR 조건으로 종료되어야 함을 의미한다.

2) Double Permutation 방법

순차적 모드의 생성 방법을 사용하여 생성된 테스트 스크립트는 특정 동작 모드의 주동작 테스트에 주안점을 두고 있다. 하지만 SUT가 다양한 동작 모드를 가지고, 각 동작 모드의 “진입시조건”이나 “종료조건”을 하나 이상을 갖는 경우 하나의 모드에서 다른 모드로 진입이 잘되는지 또는 다양한 “진입시조건” 및 “종료조건”에서 정상적으로 동작하는 지를 살펴볼 필요가 있다. 이와 같은 테스트 스크립트를 만들기 위해서는 순차 방법으로 어렵다. 즉, 순차 방법으로는 각기 다른 모드 동작의 “진입시조건”과 “종료조건”의 조합에 따른 모드의 진입여부와 종료여부의 정확성을 판단할 수 있는 조합을 생성할 수 없다.

Double Permutation 방법은 두 특정 모드가 차례로 수행되었을 때의 SUT 동작을 테스트하기 위한 테스트 스크립트를 생성하는 방법이다. 즉, SUT에 N개의 모드가 존재하면 가능한 특정 두 모드의 조합은 NC2가 된다. 예를 들면, ‘모드 A’, ‘모드 B’, ‘모드 C’ 3개에 대한 Double Permutation 조합은 3C2 = 6개(AB, BA, AC, CA, BC, CB)가 된다. 이들 중 두 모드가 순차적으로 수행될 수 없는 경우를 뺀다. 그 후 생성된 조합을 이전 조합의 끝 모드가 다음 조합의 시작 모드가 되게 나열한다. 예를 들면, 가능한 나열 조합은 ‘ABBCCAACCBBA’가 된다. 나열된 조합 중 같은 모드가 연속될 경우 하나는 제거하고 ‘ABCACBA’ 모드 동작 조합을 이용하여 테스트 스크립트를 생성한다. 이렇게 함으로서 전체 테스트하는 모드의 수를 줄일 수 있기 때문이다. 두 모드 조합을 활용한 것은 Pairwise 테스트 케이스 생성에서와 같이, 여러 모드 조합의 동작 시, 두 가지의 모드 조합을 테스트하면 오류 약 75% 정도를 파악할 수 있다는데 통계에 근거한 것이다[20].

이 때 각 모드의 “진입시조건”과 “종료조건”에는 여러 가지 조건이 있을 수 있다. 이 조건들도 테스트하기 위하여 하나의 모드에 수행될 때, “진입시조건” 또는 “종료조건”을 다르게 하여 스크립트를 생성한다. 위의 ‘ABCACBA’ 모드 조합 수행 시, 첫 번째 A모드 “진입시조건”과 두 번째 A모드 “진입시조건”의 조합을 다르게 한다. 예를 들어 A 모드의 “진입시조건” [c1 and c2 and c3]인 경우 조건 조합 정책이 MCDC라면 A모드 진입 시마다 Table 4의 3개의 조건 조합이 다르게 사용되게 테스트 스크립트를 생성한다.

Table 4. Condition Combinations by MCDC

No.	C_1	C_2	C
1	False	True	False
2	True	False	False
3	True	True	True

3) Triple Permutation 방법

Triple Permutation은 세 특정 모드가 차례로 수행되었을 때 SUT의 동작을 테스트하기 위한 테스트 스크립트를 생성하는 방법이다. 생성 방법은 Double Permutation과 같다. Double Permutation 방법에서와 같이 여기서도 이들 중 세 모드가 순차적으로 수행될 수 없는 경우를 뺀다. 이 방법은 세 모드 조합을 활용한 하면 75%를 훨씬 상회하는 오류를 파악할 수 있다는 통계에 근거한 것이다[20].

4) 무작위 생성 방법

무작위 생성은 동작모드를 고려하지 않고 동작모드 내의

커맨드를 무작위로 선택하여 의도하지 않은 시나리오에 대한 SUT 동작을 테스트하기 위한 테스트 스크립트를 생성한다.

테스트 스크립트 무작위 생성 시, 테스트 스크립트 내의 총 명령어 수와 명령어 간 시간을 정할 수 있다. 커맨드 사이의 시간을 정해주는 이유는 해당 커맨드가 입력되었을 때 테스트하고자하는 제품의 특성에 따라 명령어 유지 시간이 달라질 수 있기 때문이다.

무작위 생성의 단점은 동작모드 단위의 행동을 고려하여 명령어가 선택되는 것이 아니라 동작모드 단위의 행동과는 무관한 명령어들이 선택될 수 있기 때문에 오류가 발생하였을 경우 어떠한 동작모드에 영향을 미치는지를 분석을 통해 파악해야 한다.

3.5 테스트 스크립트 자동 생성 프로그램

앞 절에서 제안한 방법으로 구현한 테스트 스크립트 생성기의 초기 화면은 Fig. 6과 같다. 테스트 스크립트 생성기는 테스트 명령어 사전, 체크리스트, 순서 지정 파일, 테스트 스크립트 생성 위치를 입력으로 받고 테스트 엔지니어가 선택한 동작모드와 생성 방법에 따라 출력으로 테스트 스크립트 파일을 생성한다. 테스트 스크립트 생성기는 체크리스트를 입력받게 되면 체크리스트 란에 생성 가능한 동작모드를 표시하며 테스트 엔지니어는 원하는 동작모드를 선택할 수 있다.

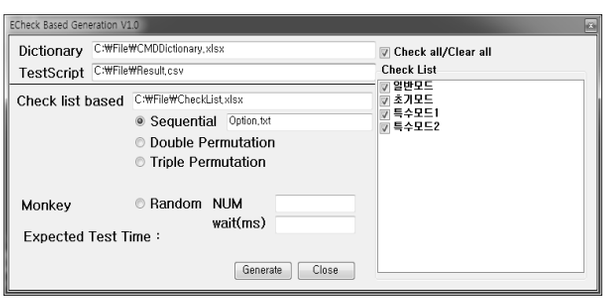


Fig. 6. Initial Screen Shot of Test Script Generator

순차적 방법은 체크리스트 란에 표기된 동작모드 순서대로 테스트 스크립트를 생성한다. 순서 지정 파일을 입력 받게 되면 순서 지정 파일에 입력된 동작모드 순서대로 테스트 스크립트를 생성하며 입력되지 않은 동작모드는 체크리스트 란에 표기된 순서대로 생성된다.

Double Permutation 방법의 경우, 동작 모드 조합을 만들고, 각 운전모드의 “진입시조건”과 “종료조건”의 테스트 스크립트의 생성 부분에 조건 조합 정책(예, MCDC)을 적용시킨 테스트 스크립트를 생성한다. Triple Permutation도 위의 Double Permutation과 같은 방식으로 테스트 스크립트를 생성하며 대신 두 개의 운전모드의 조합이 아닌 세 개의 운전모드를 조합하여 테스트 스크립트를 생성한다.

Random은 생성될 수 있는 커맨드 개수(NUM)와 커맨드 간 시간 간격(wait)에 따라서 커맨드를 무작위로 선택하여 테스트 스크립트를 생성한다.

마지막으로 테스트 예상 시간(Expected test time)은 생성된 테스트 스크립트로부터 실제 테스트를 수행하는데 걸리는 총 시간을 나타내며 테스트 엔지니어에게 테스트 수행 예상 소요 시간을 알려준다.

4. 실 험

본 실험에서는 본문에서 제시하는 이론을 바탕으로 구현한 테스트 스크립트 자동 생성 프로그램이 체크리스트와 테스트 명령어 사전 파일을 참조하여 테스트 스크립트를 올바르게 생성하는지 검증하였다.

실험에서 사용한 체크리스트 파일은 4개의 동작모드로 구분되며, 각 동작모드의 이름은 초기모드, 일반모드, 특수모드1, 특수모드2로 정의하였다. 아래 Fig. 7은 동작모드 중 초기모드에 대한 구성을 나타낸 그림이다.

실험 대상 시스템은 Fig. 7의 “D<50” 명령어 내의 “SysIn_DSensor” 등 16개의 입력 신호를 가지고 있다. 이들 신호는 “InitValue”이라는 내재된 테스트 명령어 실행에 의해 모든 입력 신호 값은 초기화 되고, 다른 테스트 명령으로 수정될 때까지 그 값을 유지한다.

동작모드	기능분류	기능설명	테스트 명령
초기모드	진입시조건	D센서가 50 미만	D<50
		주 에러 없을 시	정상
	주동작	데모모드 동작	DemoMode 진입
	종료조건	30초 작업 후 정상 종료	30초
		D센서가 50 이상	D>=50
		D_Sensor 에러 발생 시 초기모드 종료	D-ERR
		F_Sensor 에러 발생 시 초기모드 종료	F-ERR
		R1_Sensor 에러 발생 시 초기모드 종료	R1-ERR

Fig. 7. Initial Mode in the Checklist

Fig. 7의 “체크리스트 초기모드 구성”에 작성된 테스트 명령은 Fig. 8의 “테스트 명령어 사전” 파일에서 정의한 명령어를 참조한다.

명령어	group	TIME	SysIn_DSensor					
D<50	1	0	49					
명령어	group	TIME	SysIn_DSensor					
D>=50	1	0	51					
명령어	group	TIME	SysIn_DSensor	SysIn_FSensor	SysIn_R1Sensor	FB_FFan	FB_CFan	SysIn_JetFreezing
정상	8	0	49	-10	15	0	0	
		5						0
명령어	group	TIME	SysIn_DSensor	SysIn_FSensor	SysIn_R1Sensor	FB_FFan	FB_CFan	SysIn_JetFreezing
D-ERR	8	0	-50	-10	15	0	0	
		5						0
명령어	group	TIME	SysIn_DSensor	SysIn_FSensor	SysIn_R1Sensor	FB_FFan	FB_CFan	SysIn_JetFreezing
F-ERR	8	0	49	-50	15	0	0	
		5						0
명령어	group	TIME	SysIn_DSensor	SysIn_FSensor	SysIn_R1Sensor	FB_FFan	FB_CFan	SysIn_JetFreezing
R1-ERR	8	0	49	-10	-50	0	0	
		5						0
명령어	group	TIME	SysIn_DemoMode					
DemoMode 진입		0	1					
		1						
명령어	group	TIME						
30초		0						
		30						

Fig. 8. Initial Mode Commands in the Test Command Dictionary

Table 5는 실험에서 사용한 전체 동작모드(초기모드, 일반모드, 특수모드1, 특수모드2)의 “진입시조건”, “주동작”, “종료조건”에서 사용하는 명령어의 개수를 나타낸 테이블로 총 25개의 명령어로 구성되어있다.

Table 5. Number of Commands for the Operation Modes

	Precondition	Action	Postcondition
InitMode	2	1	5
NormalMode	0	5	0
SpecialMode1	1	1	1
SpecialMode2	1	1	7

4.1 테스트 스크립트 생성

테스트 스크립트 생성 프로그램이 체크리스트로부터 테스트 스크립트를 올바르게 생성하는지 확인하기 위해 로그 파일을 이용하여 생성 과정을 분석하였다. 프로그램이 체크리스트 파일에 나열된 동작모드 중, Fig. 7의 “초기모드”에 작성된 명령어를 “진입시조건”, “주동작”, “종료조건”의 처리 순서를 고려하여 정상적으로 스크립트를 생성하는지 분석하였다.

초기모드의 테스트 스크립트 생성 과정을 살펴보면 Fig. 7의 “체크리스트 초기모드” 중 가장 먼저 수행되는 기능분류는 “진입시조건”이다.

“진입시조건”에 작성된 명령어는 “D<50”과 “정상” 명령어 이며, 각 명령어의 의미는 D-Sensor 값이 50 미만이고, 오류 발생이 없는 “정상” 상태이면 “주동작”의 명령어를 수행하겠다는 의미이다. “D<50”과 “정상” 명령어는 Fig. 8에 정의 되어있으며, 프로그램은 정의된 명령어를 참조하여 실제 시스템의 입력 데이터와 다음 입력 데이터 간의 대기시간을 고려하여 하나의 테스트 케이스로 변환한다.

다음으로 수행되는 명령어는 주동작에 작성된 “DemoMode 진입” 명령어이다. “DemoMode진입” 명령의 의미는 “DemoMode” 기능에 맞는 특정한 동작을 수행하는 모드로 진입하겠다는 의미이며, 종료조건이 발생하면 모드를 빠져나오게 된다.

마지막으로 수행되는 명령어는 종료조건에 작성된 “30초” 명령어이다. 종료조건에 작성된 명령어가 하나 이상일 경우 동작 모드가 종료될 때 마다 순차적으로 하나씩 호출되도록 설정하였다. 따라서 첫 번째에 위치한 “30초” 명령이 수행된 것이다. 만약 이후에 “초기모드”가 다시 한 번 호출되고 종료된다면 “30초” 명령어 다음에 작성된 “D>=50” 명령이 수행될 것이다. “30초” 명령이 수행되면서 현재 동작모드인 “초기모드”를 종료하고 다음 동작모드로 넘어가게 된다.

동작 모드 실행 순서를 보여주는 로그 내용의 일부인 Fig. 11에서 “순차모드 테스트 스크립트 생성 순서”의 <A>부분은 동작모드 중 “초기모드” 테이블에 작성된 테스트 명령이 진입시조건, 주동작, 종료조건을 고려하여 올바른 순서대로 생성되었는지를 확인할 수 있다. 내용을 분석해보면 앞서 설명한 것과 같이 “D<50” -> “정상” -> “DemoMode진입” -> “30초” 순서대로 명령어들이 수행된 것을 확인할 수 있다.

Fig. 9는 테스트 스크립트 생성 프로그램에 의해 생성된 테스트 스크립트 결과 파일이다. 하나의 테스트 케이스인 16개의 입력 신호가 시간 (두 번째 줄 맨 우측에 나타남)에 따라 두 줄에 표현되어 있다. 스크립트의 각 테스트 케이스를 분석해보면 “TestCase#1”은 시스템의 모든 입력 초기 데이터를 정의하는 명령어인 “InitValue”가 수행되어 모든 입력 값은 초기화된 된다. 스크립트 내의 “-” 표시는 이전 값을 유지하라는 뜻이며 명령어 사전에 정의되지 않는 입력 값임을 의미한다. “TestCase#2”는 “D<50” 명령이 수행되어 만들어진 테스트 케이스이다. Fig 8의 “D<50” 명령을 살펴보면 ‘SysIn_DSensor’ 입력 신호를 “49”로 정의하였고, Fig 9의 <가> 열에서 ‘SysIn_DSensor’ 입력이 “49”로 생성된 것을 확인할 수 있다. “TestCase#3”은 “정상” 명령이 수행되어 만들어진 테스트 케이스로 Fig. 8에 작성된 데이터가 정상적으로 테스트 케이스로 생성된 모습을 확인할 수 있다. 마찬가지로 “TestCase#4”, “TestCase#5”는 각각 “DemoMode 진입”, “30초” 명령이 정상적으로 수행되어 생성된 테스트 케이스이다.

<가>

	FB_FFan	FB_CFan	SysIn_FSensor	SysIn_R1Sensor	SysIn_R2Sensor	SysIn_DSensor	SysIn_RTSensor	SysIn_FDoorSwitch	SysIn_RDoorSwitch
TestCase#1	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-
TestCase#2	-	-	-	-		49	-	-	-
TestCase#3	0	0	-10	15 -		49	-	-	-
	-	-	-	-	-	-	-	-	-
TestCase#4	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-
TestCase#5	-	-	-	-	-	-	10 -	-	-
	-	-	-	-	-	-	10 -	-	-

	SysIn_TestModeSwitch	SysIn_JetFreezing	SysIn_ExpFreezing	SysIn_DemoMode	SysIn_SmartDiagnosis	SysIn_Power	SysIn_LQCMode	TIME
TestCase#1	0	0	0	0	0	0	0	10000
	-	-	-	-	-	-		5000
TestCase#2	-	-	-	-	-	-		5000
TestCase#3	-	-	-	-	-	-		5000
	-		0 -	-	-	-		5000
TestCase#4	-	-	-	1 -	-	-		1000
	-	-	-	-	-	-		5000
TestCase#5	-	-	-	-	-	-		30000
	-	-	-	-	-	-		5000

Fig. 9. The Result of Test Script Generation

4.2 순차적 모드 생성

Table 5에 작성된 4개의 동작모드를 포함하는 파일을 사용하여 순차적 모드 테스트 스크립트를 제작하였다. 순차적 모드의 순서를 설정하는 Fig. 10의 “순서지정파일”에는 “초기모드/일반모드”는 항상 테스트 스크립트의 첫 부부분에 위치하도록 작성하였고 특수모드1 다음은 항상 초기모드가 실행되며, 특수모드2 다음은 항상 일반모드가 실행되도록 작성하였다.

```
“초기모드/일반모드”;
“특수모드1/초기모드”;
“특수모드2/일반모드”;
```

Fig. 10. The Configuration of Sequence File

Fig. 11에 작성된 <A>, <B>, <C>, <D>는 각각 초기모드, 일반모드, 특수모드1, 특수모드2를 나타낸 것이며 ①, ②, ③, ④는 동작모드에서의 “진입시조건”, “주동작”, “종료조건”을 나타낸 것이다. 생성된 동작모드의 순서는 앞서 설명한 “순서지정파일”에 설정한 것과 같이 <A>, <B>, <C>, <A>, <D>, <B> 순서로 생성되었다. 각 동작모드의 “진입시조건”, “주동작”, “종료조건”을 고려하여 테스트 스크립트로 변환하는 과정은 4.1절에서 설명한 내용과 동일하다.

Fig. 11. The Sequence of Sequential Mode Test Script Generation

4.3 Double Permutation 모드 생성

순차적 모드를 사용하여 테스트 스크립트를 생성하는 조건과 동일한 파일을 사용하여 Double Permutation 모드로 테스트 스크립트를 생성하였다. 4개의 동작모드를 Double Permutation 모드로 생성한 결과 Fig. 12와 같은 순서로 테스트 스크립트가 생성되었다.

Fig. 12의 “Double Permutation”에 의한 모드 순서를 살펴보면 알파벳으로 표시된 A, B, C, D 는 각 동작모드의 순서를 간단히 표시한 것이다. 생성된 스크립트는 “ACBCDBDCADABA” 순서로 나열되어 있고 모든 Permutation 조합이 겹쳐진 순서이다. 예를 들어 밑줄 친 동작모드 CBC는 C와 B가 조합되어 만들어진 순서로 CB 조합과 BC 조합을 모두 만족하는 순서이며, 나열된 모든 조합이 이러한 조건을 만족한다.

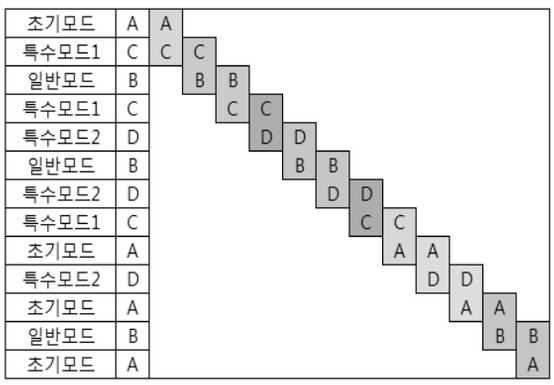


Fig. 12. Mode Sequence by the Double Permutation Policy

Double Permutation은 2개의 동작모드 간의 연관성을 테스트하기 위하여 위와 같은 순서로 테스트 스크립트를 생성한다.

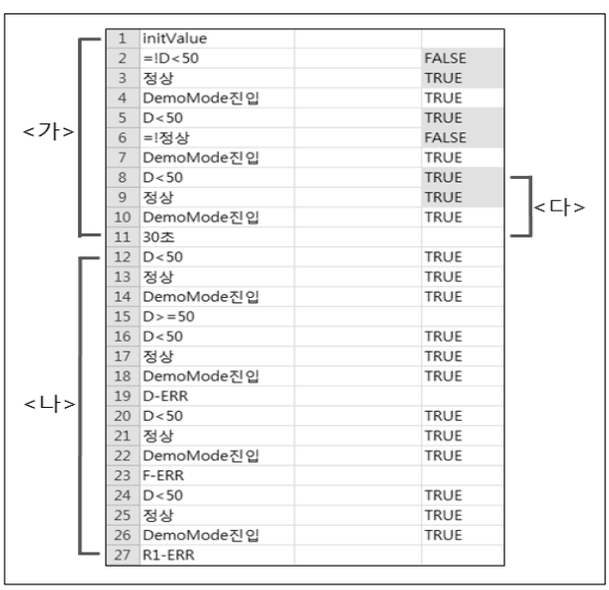


Fig. 13. Command Sequence of Initial Mode by the Double Permutation Policy

또한 Double Permutation 모드는 추가적으로 각 동작모드의 “진입시조건”과 “주동작”에 MCDC기법을 적용함으로써 더욱 체계적인 테스트 스크립트를 생성한다. Fig. 13은 Fig. 12에 나열된 동작모드 중 “초기모드”의 테스트 스크립트 생성과정을 로그파일로 기록한 내용이다. 앞서 설명한 것과 같이 <가> 부분은 “진입시조건”과 “주동작”에 MCDC 기법이 적용되었다는 것을 보여준다. “진입시조건”에 해당하는 명령어인 “D<50”과 “정상”은 각 참과 거짓에 맞는 테스

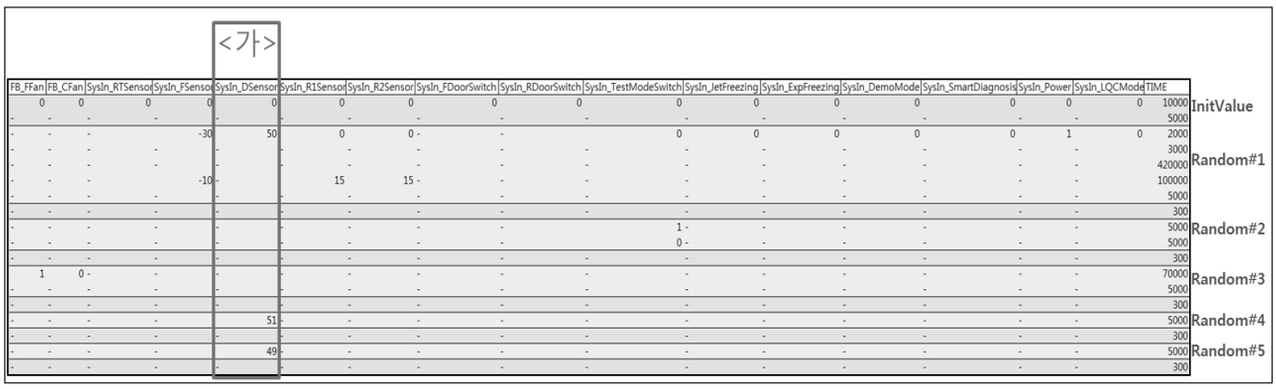


Fig. 14. Test Scripts Generated by the Random Policy

트 명령어를 생성하고, 두 조건이 모두 참인 경우에만 “주동작”에 정상 진입하여 종료되는 것을 확인할 수 있다. <나>의 조건은 5개의 “종료조건”이 정상적으로 만족하는지를 확인하는 테스트 케이스이다. 이때 동작모드의 “진입시조건”은 참 조건으로 정상적으로 진입한다. 예를 들어 <다> 부분을 살펴보면 “진입시조건”인 “D<50”와 “정상” 명령이 모두 참인 조건에서 “DemoMode진입” 명령이 수행되어 동작모드로 진입하고 “30초”라는 명령으로 종료조건을 수행하여 해당 동작모드를 종료한다. 이후 <나> 에서는 위와 같이 “진입시조건”이 모두 참인 경우를 반복하여 모든 “종료조건”을 테스트할 수 있는 명령어가 생성되는 것을 확인할 수 있다. 이러한 기법은 그림으로 설명한 “초기모드” 뿐만 아니라 “일반모드”, “특수모드1”, “특수모드2”에도 모두 적용된다.

4.4 무작위 생성

무작위 테스트 스크립트 생성 방법은 체크리스트에 작성된 명령어를 무작위로 선택하여 스크립트를 제작하는 방법이다. 테스트 스크립트 생성 프로그램은 무작위 테스트 스크립트를 제작할 때 생성을 원하는 총 명령어 수와 각 명령어 사이의 대기 시간을 입력할 수 있다. 무작위 테스트 스크립트를 제작하기 위하여 총 5개의 테스트 명령어와 300ms의 시간 간격으로 스크립트를 생성하였다.

Table 6은 무작위 생성 기능에 의해 자동으로 선택된 명령어이며 Random#1은 “일반모드동작”이라는 명령어가 선택되었다는 의미이다.

Fig. 14는 무작위로 선택된 테스트 명령어를 수행하여 생성한 테스트 스크립트 파일이다. Table 6에 작성된 내용과 같이 Random#1부터 Random#5까지 총 5개의 테스트 케이스가 생성된 것을 확인할 수 있고 각 커맨드 사이에는 300ms라는 대기시간이 추가된 것을 확인할 수 있다. 예를 들어 Fig. 14의 Random#4 테스트 케이스를 살펴보면 “D>=50” 명령어가 수행되어 테스트 케이스가 생성되었다는 것을 확인할 수 있다. “D>=50” 명령어는 Fig. 8의 테스트 명령어 사전 파일에서 SysIn_DSensor 값을 “51”로 정의하였고 Fig. 14의 <가> 열에서도 마찬가지로 SysIn_DSensor 값이 “51”로 생성된 것을 확인할 수 있다.

Table 6. Commands Selected by the Random Generation Policy

Number	Test Command
Random#1	일반모드동작
Random#2	SwitchON
Random#3	특수모드1동작
Random#4	D>=50
Random#5	D<50

5. 결 론

본 논문에서는 체크리스트를 사용하여 테스트 스크립트를 자동으로 생성하는 방법을 제시하였다. 실제 산업현장에서 일반적으로 사용하는 체크리스트의 단점을 보완하기 위해 체크리스트 체계를 구축하고 체크리스트 체계로부터 테스트 스크립트를 자동으로 도출하는 방법을 제안하였다.

테스트 스크립트를 생성하는 방법은 순차적 생성 방법, Double Permutation 생성 방법, 무작위 생성 방법을 사용하여 생성 가능하다. 순차적 생성 방법은 사용자가 원하는 동작모드의 순서를 작성하고 작성된 순서에 맞도록 테스트 스크립트를 생성하는 방법이다. 또한 각 동작모드에서 수행해야 하는 “종료조건”, “진입시조건”을 지정할 수 있다. 순차적 생성모드는 사용자의 경험적인 측면을 많이 부여할 수 있는 장점이 있다. Double Permutation 생성 방법은 각 동작모드간의 동작 순서에 의한 오류를 테스트하는데 적절한 테스트 스크립트를 생성하며, 동작모드가 “진입시조건” 및 “종료조건”은 커버리지 정책에 따라 다양한 조합으로 생성한다. 또한 “종료조건”이 여러 개인 동작모드는 모든 “종료조건”의 테스트 명령어를 수행한다. 마지막으로 무작위 생성 방법은 체크리스트에 작성된 테스트 명령어를 무작위로 선택하여 테스트 스크립트를 만드는 방법으로, 무작위 테스트 스크립트는 무작위 동작에 의한 오류를 찾아낼 수 있다.

논문에서 제안한 방법을 구현하였고, 실험을 통해 테스트 스크립트를 자동으로 생성할 수 있음을 확인하였다. 생성된 스크립트를 임베디드 시스템에 적용하여 테스트 한 결과 일반적인 체크리스트 방법을 이용하여 수작업으로 만들었던

스크립트로는 검출 하지 못했던 다양한 오류를 검출할 수 있었다. 제안한 방법에서 체크리스트를 이용하여 체계적인 테스트 케이스 생성하는 것은 다른 선행 연구에서는 다루지 못하는 내용이다. 이처럼 일반적인 체크리스트 방법은 체크리스트에 작성된 목록만을 검사하지만, 제안하는 자동생성 방법은 다양한 테스트 기법을 활용하여 목록에 작성된 내용뿐만 아니라 오류발생 가능성이 있는 동작까지도 테스트할 수 있다. 이는 수작업으로 하기 힘든 체계적이고 다양한 조합의 테스트 스크립트 생성에 대한 장점을 의미한다.

본 논문에서 제시한 테스트 스크립트 자동 생성 방법을 좀 더 효율적으로 사용하기 위해선 구조화된 시스템 스펙 및 요구사항 문서로부터 데이터를 추출하여 체크리스트 체계를 자동으로 구축하는 방법에 대한 연구가 필요하다.

References

[1] J. Y. Seo, A. Y. Sung, B. J. Choi, and S. B. Kang, "Automating Embedded software Testing on an Emulated Target Board," *Proc. of the Second International Workshop on Automation of Software Test*, 20-26 May, 2007.

[2] S. Y. Jeong, Y. W. Chang, and C. J. Yoo, "Test Case Genration Technique Based on State Transition Model for Embedded System," *Journal of Korean Institute of Information Technology*, Vol.9, No.4, pp.11-21, 2011.

[3] M. R. Keyvanpour, H. Homayouni, and Hossein Shirazee, "Automatic Software Test Case Generation: An Analytical Classification Framework," *International Journal of Software Engineering and Its Applications*, Vol.6, No.4, Oct., 2012.

[4] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Proceedigns of the 9th Asian computaing Science Conference, volum 3321 of Lecture Notes in Computer Science*, pp.320-329, 2004

[5] Kuo Chung Tai and Yu Lei, "A Test Generation Strategy for Pairwise Testing," *IEEE Transactions on Software Engineering,* Vol.28, No.1, pp.109-111, Jan., 2002.

[6] K. P. Chan, T. Y. Chen, and Dave Towey, "Restricted Random Testing," in *Proceedings of the 7th European Conference on Software Quality Helsinki, Finland, Vol.2349/2002 of Lecture Notes in Computer Science*, pp.321-330, Jun., 2002.

[7] M. Conrad, H. Dörr, I. Fey, and A. Yap, "Model-based Generation and Structured Representation of Test Scenarios," *Workshop on Software-Embedded Systems Testing (WSEST)*, Gaithersburg, USA, Nov., 1999.

[8] P. S. Loo and W. K. Tsai, "Random Testing Revisited," *Information and Software Technology*, Vol.30, Iss.7, pp.402-417, Sep., 1988

[9] T. Y. Chen, F. C. Kuo, Huai Liu, and W. E. Wong, "Does Adaptive Random Testing Deliver a Higher Confidence than Random Testing?" *The Eighth International Conference on Quality Software, QSCI'08*, pp.145-154, Aug., 2008.

[10] M. Grochtmann and K. Grimm, "Classification Trees for Partition Testing," *Software Testing, Verification & Reliability*, Vol.3, No.2, pp.63-82, Jun., 1993.

[11] J. H. Shin, K. H. Chung, and K. H. Choi, "Destructive Test of a BLDC Motor controller utilizing a Modified Classification Tree Method," *KIPS Tr. Software and Data Eng.*, Vol.3, No.6, pp.201-214, piSSN: 2287-5905, Mar., 2014.

[12] P. M. Kruse and M. Luniak, "Automated test case generation using classification trees," Software Quality Professional Magazine, 2010.

[13] The International Engineering Consortium, Technical Report, "Specification and Description Language(SDL)."

[14] AGEDIS Consortium, Technical Report, "Model Based Test Generation Tools."

[15] A. Hartman and K. Nagin, "The AGEDIS Tools for Model Based Testing," *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp.129-132.

[16] Alexander Pretschner, "Model-Based Testing," *Proceedings of the 27th Interantional Conference on Software Engineering*, pp.723-822.

[17] H. S. Park, "Generating Structural Test Cases for MATLAB Stateflow Model Using Rapidly-exploring Random Tree," Ajou Univ, Engineering doctoral dissertation, 2014.

[18] M. Utting and B. Legeard, "Practical Model-Based Testing: A Tools Approach," Morgan kaufmann, 2007.

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

[20] Yu Lei and K. C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing," *High-Assurance Systems Engineering Symposium*, *Proceedigns. 3rd IEEE International*, 1998.

[21] A. P. Mathur, "Foundations of Software Testing," Pearson Education, 2008.

[22] Junyeon Hwang, "Auto Test Script Generation Based on Checklist," Master Dissertation, Ajou University, Suwon, Korea, 2015.

[23] Joseph Schmuller, "Teach yourself UML in 24 Hours, 3/E," SAMS, 2004.

[24] Ivar Jacobson, "Object-Oriented Software Engineering: A Use-Case-Driven Approach," Addison-Wesley, 1992.

[25] Ivar Jacobson, Kurt Bittner, and lan Spence, "Use Case Modeling," Addison-Wesley, 2002.

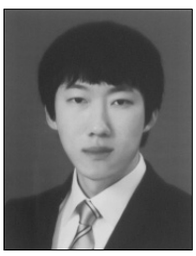
[26] Paul C. Jorgensen, "Software Testing: A Craftsman's Approach, 4/E," CRC Press, 2016.

[27] U. S. NRC, "Fault Tree Handbook (NUREG-0492)," US, 1981.

[28] Rick Kuhn, Raghu Kacker, Yu Lei, and Justin Hunter, "Combinatorial Software Testing," IEEE, 2009.

강 태 훈

e-mail : kth10@ajou.ac.kr
2014년 아주대학교 정보컴퓨터공학과 (학사)
2014년～현 재 아주대학교 컴퓨터공학과 석사과정
관심분야 : 운영체제, 임베디드 시스템, 임베디드 시스템 테스팅



김 대 준

e-mail : kdj4527@ajou.ac.kr
2015년～현 재 아주대학교 컴퓨터공학과 석사과정
관심분야 : 운영체제, 임베디드 시스템, 임베디드 시스템 테스팅

정 기 현

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

최 경 희

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