KIPS Trans. Softw. and Data Eng.
Vol.8, No.8 pp.331~342 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2019.8.8.331

Generating Test Cases and Scripts from Requirements in Controlled Language

Hye Jin Han† · Kihyun Chung†† · Kyunghee Choi†††

ABSTRACT

This paper proposes a method to generate test cases and test scripts from software requirements written in a controlled natural language, which helps develop reliable embedded software. In the proposed method, natural language requirements are written in a controlled language, the requirements are parsed and then inputs, outputs and operator data are extracted from the requirements. Test cases are generated from the extracted data following test case generation strategies such as decision coverage, condition coverage or modified condition/decision coverage. And then the test scripts, physical inputs of the test cases are generated with help of the test command dictionary. With the proposed method, it becomes possible to directly check whether software properly satisfies the requirements. Effectiveness of the proposed method is verified empirically with an requirement set.

Keywords : Test Case Generation, Test Script, Controlled Requirement, Test Coverage, Test Case Auto Generation

구조화된 자연어 요구사항으로부터 테스트 케이스 및 스크립트 생성

한 혜 진† · 정 기 현†† · 최 경 희†††

요 약

본 논문은 신뢰성 있는 임베디드 시스템 소프트웨어 개발을 위해 제한된 자연어 형식으로 작성된 소프트웨어 요구사항으로부터 테스트 케이스 및 테스트 스크립트를 생성하는 방안을 제안한다. 제안하는 방법에서는 자연어로 기술된 요구사항을 제한된 자연어 형식으로 작성하고, 이를 파싱하여 테스트에 사용되는 입력, 출력 및 연산자를 추출한다. 추출된 정보를 이용하여 Decision Coverage, Condition Coverage, Modified Condition/ Decision Coverage와 같은 테스트 케이스 생성 전략을 적용하여 테스트 케이스를 생성한다. 또한 테스트 명령어 사전을 이용하여 임베디드 시스템의 물리적인 입력 값인 테스트 스크립트를 생성한다. 제안한 방법을 이용하면, 개발된 소프트웨어가 요구사항에 적합하게 개발되었는지를 직접적으로 테스트하는 것이 가능하다. 제안한 방법의 효과는 요구사항 세트에 적용하여 실험적으로 보인다.

키워드 : 테스트 케이스 생성, 테스트 스크립트, 정형화된 요구사항, 테스트 커버리지, 테스트 케이스 자동 생성

1. 서 론

임베디드 시스템 소프트웨어의 복잡도가 지속적으로 증가함에 따라 소프트웨어의 잠재적 위험성도 함께 증가하고 있다. 특히 임베디드 시스템이 적용되는 산업에 따라 소프트웨어의 단순한 오류도 치명적인 문제를 초래할 수 있다.

일반적으로 임베디드 시스템을 테스트하기 위해 사용하는 테스트 기법에는 시스템 내부 코드를 기반으로 하는 화이트 박스(White Box)와 시스템의 외부 기능을 중점적으로 테스트 하는 블랙박스(Black Box) 기법이 있다.

현재 시스템의 소프트웨어의 품질 및 신뢰성 확보를 위해 이루어지고 있는 화이트 박스 기반 정적/동적 테스트는 소스 코드 오류 등을 확인하기에 좋은 방법이지만 시스템의 동작이 시스템 요구사항을 만족하도록 구현되어있는지 검증하기에 어려움이 있다. 만약 코드 자체가 요구사항과 다르게 해석되어 작성될 경우, 이 차이로 인해 발생한 오류가 코드 기반 정적/동적 테스트를 통해서 검출되기란 매우 어렵다.

반면 블랙박스 테스트는 소프트웨어 자체의 기능을 중점

※ 본 연구는 국방과학연구소의 지원(계약번호: UD170016DD)으로 수행되었음.
† 준 회 원 : 아주대학교 컴퓨터공학과 석사과정
†† 정 회 원 : 아주대학교 전자공학과 교수
††† 정 회 원 : 아주대학교 컴퓨터공학과 교수
Manuscript Received : December 14, 2018
First Revision : April 22, 2019
Second Revision : June 11, 2019
Accepted : June 19, 2019
* Corresponding Author : Kihyun Chung(khchung@ajou.ac.kr)

적으로 테스트하여 임베디드 시스템과 같이 하드웨어가 내장된 소프트웨어가 적절히 동작하는 지 테스트할 때 사용한다. 이러한 블랙박스 테스트는 소스 코드가 필요 없지만 테스트 엔지니어가 자연어로 작성된 체크리스트나 요구사항으로 부터 테스트 케이스를 생성할 때 자연어로 기술된 요구사항이 가지는 모호성으로 인해 테스트 케이스가 요구사항과 차이가 날 수 있다는 단점이 있다.

이와 같은 문제를 해결하는 방법 중의 하나로 무기 체계 등에서 요구사항을 정형화하는 방안을 생각할 수 있다. 요구사항을 구현한 소스 코드와 요구사항 간의 격차를 줄일 수 있을 뿐만 아니라, 요구사항을 기반으로 테스트 케이스를 정확히 생성하기 위해서 자연어로 작성된 요구사항을 정형화하고 그를 이용하여 테스트 케이스 및 테스트 스크립트를 생성하는 방안이 있다.

일반적으로 사용되는 자연어로 작성되어 있는 요구사항으로부터 테스트 케이스를 만드는 것은 다음과 같은 문제가 있다.

1) 자연어 요구사항의 단점인 모호함으로 인해 엔지니어의 경험과 상황에 따라 요구사항을 원래 의도와는 다르게 해석할 수 있다.

2) 자연어로 작성된 요구사항을 테스터 엔지니어가 분석하여 수작업으로 테스트 케이스를 만드는 작업은 매우 많은 시간이 소비되며, 테스트 항목이 바뀌거나 추가될 때마다 추가된 항목의 조건들을 고려하여 테스트 케이스를 다시 생성해야 하는 불편함이 있다.

3) 요구사항을 이용한 테스트 케이스는 요구사항에 작성된 내용만을 중점적으로 테스트하는데, 이는 요구사항 입력의 다양한 변형된 조건에 의해 발생할 수 있는 오류 동작은 테스트하기가 어렵다.

4) 테스트 스크립트는 시스템의 물리적인 입력 값이다. 테스트 케이스를 통해 테스트 스크립트를 만들 때 시스템 하드웨어 정보가 필요하다. 이 정보가 복잡하거나 잘 모르는 테스트 엔지니어가 테스트 스크립트를 만들 때에는 어려움에 직면하게 되고 많은 노력이 요구된다.

본 논문에서는 위와 같은 문제를 해결하기 위해 자연어로 기술된 요구사항 해석의 오류를 줄일 수 있도록 정해진 단어와 문법을 적용하여 요구사항을 정형화하고, 그를 통해 시스템에 맞는 적절한 테스트 전략이 적용된 테스트 케이스를 자동으로 생성하는 방법을 제안한다. 또한, 제안하는 방법에서는 테스트 스크립트를 쉽게 생성하기 위하여 계층 구조를 사용하여 스크립트에 사용되는 명령어의 재사용성을 높여 쉽고 빠르게 테스트 스크립트를 생성할 수 있도록 한다.

본 논문의 구성은 다음과 같다. 제 2장은 관련 연구로 자연어 문서의 모호성 제거를 위해 활용된 다양한 CNL(Controlled Natural Language, 이하 CNL) 적용 사례와 블랙박스 테스트를 위한 테스트 케이스 생성 방법에 대한 관련 연구를 소개하고, 제 3장은 STE(Simplified Technical English, 이하 STE)의 철학을 적용한 한국어 CNL을 기반으로 작성된 요구사항으로부터 테스트 케이스 및 스크립트를 생성 할 수 있는 방법을 제시한다. 제 4장은 논문에서 제시하는 방안을 실험하고 및 그 결과에 대해 기술한다. 그리고 결론에서는 연구 결과를 정리하고 앞으로 확장을 위한 연구방향을 제안하고 논문을 마무리한다.

2. 관련 연구

자연어로 작성된 요구사항은 자연어의 단점인 모호함으로 해석의 오류가 발생할 수 있다. 해석의 오류는 소프트웨어의 오류와 소프트웨어를 탑재한 시스템의 오류로 이어지게 된다. 이러한 문제를 해결하기 위해 다양한 방법이 시도되고 있는데, 그 중 하나가 CNL[1]이다. CNL은 문장을 작성하는 방법과 사용할 수 있는 단어 등의 제약 사항을 가진 자연어로서 요구사항의 표현에 약간의 제약을 둠으로써 모호성을 줄이고, 자연어 상태로 프로그램 처리를 가능하게 한다. CNL의 일종인 STE[18]는 어휘나 문장구조를 제한하여 문서의 가독성과 번역 수월성을 높일 수 있어, 영어, 프랑스어, 독일어, 일본어 등 다양한 언어로 정의되어 사용되고 있다. CNL은 어휘나 문장구조를 목적에 따라 제한함으로써 문서의 가독성과 번역 수월성을 높일 수 있는 가능성으로 인해 학술연구, 국방 무기체계, 항공우주 산업, 비즈니스, 자연과학 등에서 사용하고 있으며 점차 사용범위가 증가하고 있는 추세이다.

일반적으로 소프트웨어를 테스트 방법에는 화이트박스(White Box)와 블랙박스(Black Box) 테스트 방법을 사용한다.

화이트박스 테스트 방법은 시스템 내부 소스코드를 분석하여 코드의 문법, 코드의 규칙 및 코드의 실행 오류(Run Time Error) 등을 검출하는 방법이다. 이 방법은 생성된 시스템 내부 소스코드와 시스템의 요구사항과 일치 여부를 검증하는데 사용하기가 어렵다는 단점이 있다.

블랙박스 테스트 방법은 시스템 내부 소스코드를 고려하지 않고 완성된 시스템이 요구사항이나 시스템 사양 등을 분석하여 시스템이 요구사항에 맞게 동작하는지를 검출하는 방법이다[2]. 많은 다양한 블랙박스 테스트 케이스 생성 방법이 연구되어 왔다. 블랙박스 테스트 케이스 생성 방법의 종류로 무작위(Random) 생성 방법[3], 시스템 입력 조합(Combinatorial)을 이용한 생성 방법[4] 및 모델 기반(Model-based) 생성 방법[5-7] 등이 있다.

일반적으로 산업에서 자연어로 작성된 요구사항을 분석하여 UML(Unified Modeling Language, UML)이나 Simulink와 같은 모델링 도구를 이용하여 요구사항을 표현하고 모델로부터 직접 테스트 케이스를 도출하는 방법을 사용하기도 한다[8]. 모델링 도구를 통해 요구사항을 모델로 표현하고 테스트 케이스를 자동 생성 또는 시뮬레이션을 사용할 수 있는 모델 기반 생성 방법은 모델링에 요구사항을 완벽하게 반영할 수 없어 요구사항과 모델 간 차이가 발생할 수 있다는 단점이 있다. 또한 요구사항으로부터 직접 테스트 케이스를 도출할 경우, 같은 시스템 요구사항을 사용하더라도 테스트 엔지니어마다 요구사항을 다르게 해석할 수 있어, 시스템의 정확한 테스트가 이루어지지 않을 수 있다는 단점이 있다.

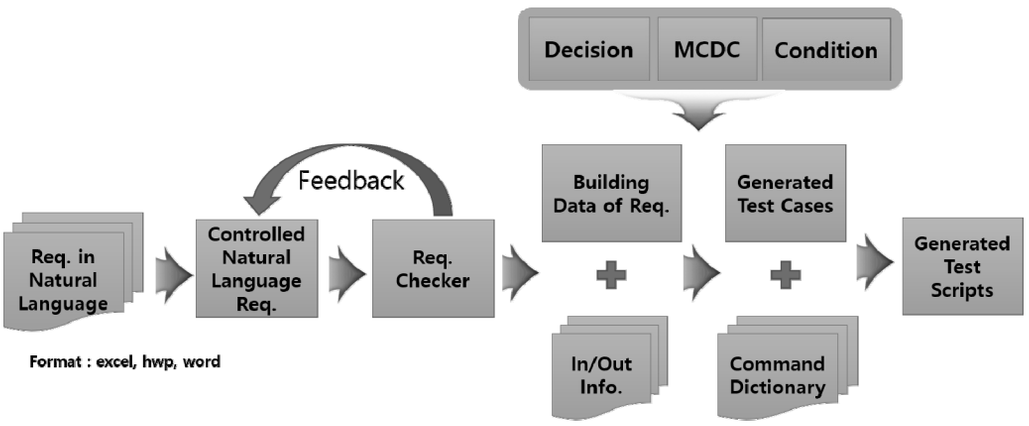

Fig. 1. Flow Diagram for Generating Test Cases and Test Scripts

구현된 소프트웨어가 해당 요구사항을 만족하는 지를 테스트하기 위해서 많이 사용하는 것 중 하나가 체크리스트이다. 체크리스트는 테스트 엔지니어가 다루었던 시스템 지식이나, 과거의 경험, 사용자의 피드백을 활용하여 테스트하고자 하는 항목을 적어놓은 리스트이다. 체크리스트는 테스트 엔지니어가 수동으로 체크 항목인 테스트 케이스를 작성하여 사용된다. 사용되는 분야에 따라 다른 형태의 체크리스트를 사용하고 있지만, 그 형태와 무관하게 모든 체크리스트는 각 테스트 항목에 어떤 입력 조건 따라 어떤 출력이 나와야 한다는 것이 명시되어 있다.

Kim et al.[9]의 연구에서는 체크리스트 기반으로 테스트 케이스를 생성하는 방법에 대해 다룬다. 이 연구는 효율적으로 체크 리스트 항목을 작성할 수 있도록 체크리스트의 체계를 제안하고, 이를 기반으로 체크리스트를 생성한다. “진입 시 조건”과 “종료 조건”의 조합에 따라 조건 커버리지(Condition Coverage), 결정 커버리지(Decision Coverage), 수정된 조건 결정 커버리지(Modified Condition/Decision Coverage, 이하 MC/DC)[11, 13]와 같은 커버리지 기반 테스트 케이스를 생성하여 다양한 조합에 따른 시스템 오동작 여부를 확인할 수 있다. 하지만 이 방법 또한 자연어로 작성되어 있는 요구사항으로부터 테스트 엔지니어가 해석한 후 체크리스트를 생성하여 테스트 케이스를 생성해야 한다는 한계가 있다.

자연어 요구사항으로 테스트 케이스를 직접 도출하는 연구는 국내외에서 매우 드물다.

연구 [19]에서는 영어로 작성된 요구사항으로부터 테스트 시나리오를 생성하는 방안을 제시하였다. 제공하는 템플릿에 맞게 작성된 요구사항으로부터 등록된 단어를 중심으로 요구사항을 파싱하고, 이 정보를 기반으로 그리고 이를 기반으로 테스트 시나리오를 작성한다. 하나의 시나리오는 어떤 ‘입력(input)’으로 어떤 ‘동작(behavior)’를 하면 어떤 ‘결과(action)’가 나와야 한다는 것을 나타낸다. 이 방법은 요구사항이 수행해야 하는 동작을 테스트하는 시나리오를 쉽게 생성하는 데는 사용할 수 있지만, 입력 조합에 의한 다양한 상황에 대한 테스트 케이스를 생성하지 못하는 단점이 있다.

연구 [20]에서는 CE(Controlled English)를 미군의 무인 항공기 경로 결정을 위한 시나리오 작성에 사용된다. 먼저 작성된 시나리오에서 ‘platform type’과 ‘system type’로 나누어진 hard source 요소를 찾아낸다. 그리고 hard source 요소의 동작을 표현하는 task들을 이용하여 사용자가 작성한 비구조적인 문장으로부터 유의미한 정보를 추출하여 무인 항공기의 경로를 결정하는데 사용한다. 이 방법은 일반적인 요구사항을 표현하는 데는 제약이 너무 많아 사용하기 힘들다.

3. 요구사항 정형화 및 테스트 스크립트 자동 생성 방안

요구사항 기반으로 테스트 케이스를 생성하기 위해 다양한 방법들이 활용되고 있다. 자연어로 작성된 요구사항을 분석하여 UML이나 Simulink와 같은 모델링 도구를 통해 모델을 만들고 모델로 부터 테스트 케이스를 도출하는 방법과 자연어로 작성된 요구사항으로부터 엔지니어의 경험 등을 활용하여 테스트 케이스를 도출하는 방법 등이 있다[7].

하지만, 이 방법들은 자연어 요구사항의 모호성 및 부정확성 등의 문제를 해결하지 못한다. 이 문제를 해결하기 위한 하나의 방법으로 본 연구는 자연어로 작성된 요구사항으로부터 직접 테스트 케이스를 도출하고, 생성된 테스트 케이스를 임베디드 시스템의 물리적인 값인 테스트 스크립트로 변환하는 방법을 제안한다.

본 연구에서 다루는 소프트웨어 블랙박스 테스트 기법에서 가장 중요한 문제 중의 하나는 자연어로 작성되어 있는 요구사항을 정확하고 일률적인 해석이 가능해야 한다는 것이다.

따라서 본 연구는 해석의 오류를 줄일 수 있도록 요구사항을 정형화하고, 정형화된 요구사항으로부터 테스트 케이스 및 스크립트를 생성하는 방안을 제안한다. 제안하는 방안의 흐름도는 Fig. 1과 같다.

테스트 대상 소프트웨어의 최종 목표는 그 소프트웨어가 주어진 요구사항을 만족하는 지를 테스트하는 것이므로, 테스트 기준은 요구사항이다. 일반적으로 요구사항은 자연어로

작성되어 있고, 자연어의 특성 상 모호함과 복잡성이 존재하여 해석의 오류를 가져온다. Fig. 1의 흐름에 따른 테스트 케이스 생성 과정은 다음과 같다.

1) 자연어 요구사항으로 부터 생길 수 있는 모호성을 최소화하고 요구사항을 체계적으로 작성하기 위한 방안으로 요구사항 작성에 사용되는 단어나 문장 구조를 제한하여 정형화된 요구사항을 작성한다.

2) 요구사항 규칙과 등록된 단어를 기반으로 자연어로 작성된 요구사항이 정확히 작성되었는지를 요구사항 검사기로 확인한다.

3) 정형화된 요구사항으로부터 테스트 케이스를 생성에 필요한 정보를 추출한다. 이 때, 데이터에 사용된 입/출력 정보가 필요하다. 입/출력 정보에는 데이터에 작성된 입/출력 명, 신호 명, 유효 범위, 데이터 타입 등이 정의되어 있다. 이 정보를 통해 테스트 케이스로 사용될 수 있는 데이터를 추출한다.

4) 요구사항으로부터 추출된 데이터와 데이터에 사용된 입/출력 정보를 이용하여 원하는 테스트 전략을 적용하여 테스트 케이스를 생성한다. 테스트 전략의 예로는 결정 커버리지, 조건 커버리지, 수정된 조건 결정 커버리지 등이 있다.

5) 생성한 테스트 케이스를 테스트 대상 시스템에 입력으로 들어가는 물리적인 입력 값인 테스트 스크립트로 변환한다. 본 논문은 [14]에서 제안하는 계층 구조를 활용하고, 요구사항에 작성된 입/출력, 테스트 케이스 값, 명령어 사전을 이용하여 해당 테스트 케이스의 시간 별 입/출력인 테스트 스크립트를 생성한다.

3.1 요구사항 정형화

Kwon et al.[10]의 연구는 STE의 철학을 기반으로 한국어 요구사항 작성 규칙을 제안하고 이를 무기체계 요구사항에 시범 적용하여 그 가능성을 보인바 있다. 본 논문에서는 [10]에서 생성한 한국어 요구사항 규칙을 이용하여 요구사항을 정형화하는 방안을 사용한다. [10]에서 제안한 한국어 요구사항 규칙은 요구사항 작성 규칙과 문장 형식을 구조화한 요구사항 템플릿으로 구성되어 있다.

일상생활에서 사용하는 모든 일반 문장에 적용되어 처리 가능한 자연어 요구사항 규칙을 만든다는 것은 매우 어렵다. 따라서 본 논문에서 제안하는 요구사항 규칙은 제한된 분야에 사용하는 요구사항을 처리할 수 있는 규칙을 정하였다. 제안하는 요구사항 규칙은 19개로 구성되어 있다. 규칙은 가능한 단순화하여 현재 사용하는 요구사항을 크게 변경하지 않고도 사용할 수 있도록 하였다. Table 1은 [10]에서 소개한 요구사항 체제 규칙의 일부이고, Table 2는 작성 규칙의 일부이다.

요구사항 템플릿은 6개의 템플릿과 관련 템플릿 요소로 구성되어 있다. 템플릿은 한 문장의 구조를 BNF(Backus-Naur Form) 다이어그램[12]으로 정의하고, 템플릿 요소는 템플릿을 구성하는 각 문장 성분 규칙을 정의한다.

Fig. 2는 [10]에서 소개한 BNF로 표현된 요구사항 템플릿의 예이다. 템플릿의 요소는 주어, 목적어, 부사어, 서술어, 보어, 관형어로 이루어져있다.

Table 1. STK System Rule Examples[10]

System Rule	
Rule 1.1	요구사항은 소프트웨어요구사항명세서 작성방법을 따른다.
Rule 1.2	유효한 요구사항은 식별자가 포함된 표 안에 기술한다.
Rule 1.3	단위 SRS의 행동 명세는 번호 체제에 따라 작성한다.

Table 2. STK Writing Rules[10]

Writing Rule	
Rule 2.1	동사의 시제는 현재형만 사용한다.
Rule 2.2	동사는 “~한다”를 사용한다.
Rule 2.3	반드시 능동태만 사용한다.
Rule 2.4	용어나 단어를 선택할 때, 항상 일관된 문구를 사용한다.
Rule 2.5	짧고 명확한 문장으로 쓴다.
Rule 2.6	번호 체제에 따라 단계적으로 행동 명세를 작성한다.
Rule 2.7	행동 명세 첫 문장에는 반드시 주어가 있어야 한다.
Rule 2.8	주 문장과 주어가 같은 경우는 2번째 문장부터 생략 가능하나, 다를 경우는 반드시 명시해야 한다.
Rule 2.9	한 문장에는 하나의 행동만을 표현한다.
Rule 2.10	항목이 여러 개일 경우 항목리스트(Vertical List)를 이용한다.
Rule 2.11	조건문은 ‘~ 경우’로 작성한다.
Rule 2.12	조건이 만족되지 않았을 경우에 수행할 행동이 있으면 반드시 작성해야 한다. 생략된 경우, 어떠한 행동도 하지 않는다.
Rule 2.13	조건 항목이 복수 개이면 조건 항목을 항목리스트로 작성하고, 조건이 유효한 기준을 명시한다.
Rule 2.14	‘~후, ’, ‘~전, ’, ‘~동시’를 사용하여 행동 실행이나 사건 발생 순서를 표현한다.
Rule 2.15	인터페이스의 자료타입은 따로 표로 작성한다.
Rule 2.16	인터페이스 항목에 송신/수신은 콜론(:)과 콤마(,)를 사용하여 작성한다.

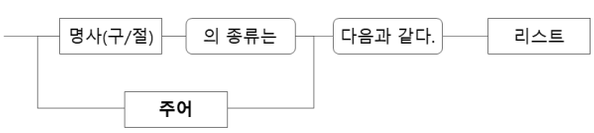

Fig. 2. A STK Template Example[10]

요구사항 규칙은 문장을 작성하는 일종의 문법이기 때문에 요구사항은 반드시 그 규칙에 따라 작성되어야 한다. 작성에 어려움을 줄이기 위해서는 제안한 요구사항 규칙은 제한된 분야에서 현재 사용 중인 요구사항의 표현과 크게 다르지 않다. 하지만 작성된 요구사항을 프로그램으로 프로세싱하여 필요한 정보를 추출하여 테스트 케이스 작성에 사용하기 위해서는 정해진 규칙대로 작성되었는지를 검사를 거쳐 확인한다.

요구사항 검사를 위해서 요구사항 검사기를 사용한다. 요구사항 검사기는 요구사항에 작성할 문장을 입력하면 요구사항 규칙과 템플릿을 참조하여 작성된 요구사항이 적절히 작성되었는지를 알려준다. 다음 Fig. 3과 Fig. 4는 요구사항 검사기의 실행 결과이다. Fig. 3은 요구사항 규칙에 맞는 요구사항을 입력했을 때의 결과를 보여주며, Fig. 4는 요구사항 규칙에 맞지 않는 요구사항을 입력했을 때의 결과를 보여준다. 앞에서 언급하였듯이 본 논문에서 제안하는 규칙은 특정 무기체계의 요구사항에 한정되어 적용 가능하다.

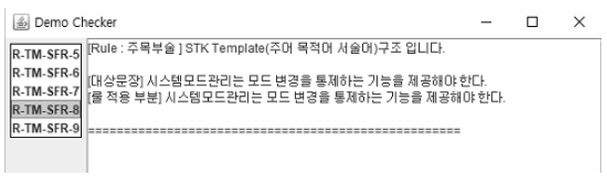

Fig. 3. A Screen Shot for Correct Requirement

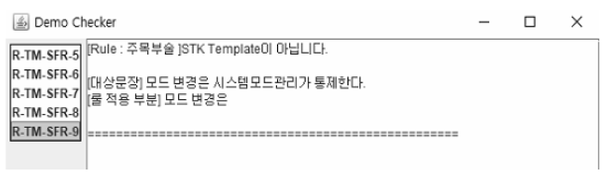

Fig. 4. A Screen Shot for Incorrect Requirement

3.2 요구사항 기반 테스트 케이스 생성

1) 요구사항 데이터 생성

테스트 케이스를 생성하기 위해서는 요구사항으로부터 테스트 케이스 생성에 필요한 정보를 추출해야 한다. 이러한 정보는 요구사항의 입/출력 정보 및 연산자 정보(Fig. 1 참조)를 이용하여 추출된다.

요구사항 문서는 한글(.hwp), MS워드(.doc, docx), MS엑셀(.xlsx, xls) 형식으로 작성할 수 있다. 본 연구에서는 요구사항 파싱(parsing)을 위해서 Java 기반의 한국어 형태소 분석기인 “KOMORAN”[13]을 이용한다. 요구사항 템플릿 문장에서 사용된 주어, 목적어, 부사어 등을 이용하여 추출하여 형태소 분석을 통해 요구사항에 작성된 주어, 목적어, 부사어 등의 내용을 추출할 수 있다. Table 3은 임의로 작성한 요구사항과 형태소 분석을 통해 추출된 요구사항 정보를 보여준다.

본 연구에서 사용하는 “KOMORAN”은 사용자 정의 사전을 제공하여 사용자가 원하는 용어와 품사를 지정할 수 있다. 예를 들면 Table 3에 작성된 시스템A는 실제로 “시스템”, “A”로 분석이 되어야 하지만 사용자 정의 사전에 “시스템A(\Tab)NNP”로 지정하면 “시스템A”로 추출된다. 여기에서 NNP는 KOMORAN에서 명사를 의미한다. Table 3의 “SensorA”, “SensorB”, “시스템B”, “OutputA” 모두 사용자 정의 사전에 미리 정의한 후 요구사항을 추출하면 Table 3의 추출된 요구사항으로 테스트 케이스 생성에 필요한 데이터를 추출할 수 있다.

Table 3. An Example of Data Extracted from Requirement

Requirement	Extracted Data from Requirement
시스템A의 SensorA가 200 이상이고, SensorB가 300 이하이면, 시스템C의 OutputA를 가동한다.	시스템A, 의, SensorA, 가, 200, 이상, 이고, SensorB, 가, 300, 이하, 이면, 시스템C, 의, OutputA,를, 가동, 한다.

요구사항을 파싱하여 얻은 데이터와 시스템 입출력을 결합하기 위해서 요구사항에 작성된 입/출력 정보와 연산자 정보는 별도의 파일에 사용자가 직접 입력하여 작성하며, 요구사항에 사용된 입력과 출력의 정보와 조건 정보를 담고 있다.

입/출력 파일은 요구사항에 사용되는 모든 입/출력과 이에 해당하는 변수의 정보를 가지고 있다. 파일 내에 표시되어야 하는 입/출력 정보는 입/출력의 시스템 형태(Type), 입/출력명(Name), 입/출력 신호 명(Symbol), 입/출력에 사용되는 값의 최대, 최소 범위(Max, Min Range), 입/출력 단위(Unit), 입/출력 값 형태(Data Type)이다.

Fig. 5의 입/출력 정보의 예에서 시스템 형태는 SystemA, SystemB, SystemC 이다. 입/출력 명(Name)은 실제 요구사항에서 사용된 입/출력 명을 의미한다. Fig. 5에서는 SystemA라는 시스템 형태 내에 SensorA, SensorB, SensorC라는 입/출력명이 작성되어 있다. 입/출력 신호명(Symbol)은 입/출력으로 만들어진 테스트 케이스로부터 테스트 스크립트를 사용할 때 사용되는 것으로 테스트 스크립트에서 사용되는 입/출력 명을 의미한다. 테스트 스크립트의 입력을 의미하면 “SysIn”을, 출력을 의미하면 “SysOut”을 붙인다. 예를 들면, SysIn_SensorA는 테스트 대상 시스템의 입력을 의미한다.

Type	Name	Symbol	Range		Unit	DataType
			Min	Max		
SystemA	SensorA	SysIn_SensorA	-50	1000		real
	SensorB	SysIn_SensorB	0	2000		real
	SensorC	SysIn_SensorC	0	1		bool
SystemB	SensorD	SysIn_SensorD	0	7000		int
	SensorE	SysIn_SensorE	-500	500		int
	SensorF	SysIn_SensorF	0	500		int
SystemC	OutputA	SysOut_OutputA	0	1		bool
	OutputB	Sysout_OutputB	0	1		bool
	OutputC	Sysout_OutputC	0	1		bool
	OutputD	Sysout_OutputD	0	1		bool

Fig. 5. An Example of Input/Output Information

입/출력에 사용되는 값의 최대, 최소 범위(Max, Min Range)는 실제 입/출력이 가질 수 있는 최대 및 최솟값을 의미한다. 최대, 최소 범위는 테스트 케이스를 생성 시 요구사항으로부터 작성된 입/출력의 유효범위 내에서 임의의 값을 추출할 때 사용한다.

입/출력 단위(Unit)은 입/출력에 사용되는 값의 단위를 나타내며, 본 논문에서는 단위를 직접적으로 이용하진 않지만 추후 테스트 케이스 및 스크립트 생성에 필요할 수 있어 표시한다.

입/출력 값 형태(Data Type)는 입/출력이 가질 수 있는 값

의 형태가 정수, 소수 또는 Boolean(0, 1)이다.

연산자 정보 파일은 입/출력 외에 요구사항에 작성된 조건들의 정보를 가지고 있다. 조건은 비교(Table 4), 명령(Table 5), 접속어(Table 6)와 같은 연산자들이 해당된다. 비교 명령어와 해당 연산자는 “크다(>)”, “크거나 같다(>=)”, “작다(<)”, “작거나 같다(<=)”, “같다(==)”와 같이 입/출력과 요구사항에 적힌 특정 값을 비교하기 위해 사용된다. 명령 연산자는 “대기”, “발사준비”, “동작”과 같이 해당 변수에 입/출력을 설정하기 위해 사용된다. 접속 연산자는 앞의 조건과 뒤의 조건이 동시에 만족해야 하는 경우(AND), 앞의 조건과 뒤의 조건 중 하나만 만족해도 되는 경우(OR)로 나뉘어 사용할 수 있다.

Table 4. The Comparison Operators

Operator	Comparison
>	크다 많다 빠르다 올라가다 초과
>=	이상 크거나 같다
==	같다 동일하다 이다
<	작다 적다 느리다 내려가다 미만
<=	이하 작거나 같다
,	일 때 이면 경우 시점

Table 5. The Command Operators

Operator	Command
0	FALSE 대기 발사준비
1	TRUE 발사 동작 작동

Table 6. The Link Operators

Operator	Link
&&	또한 면서 이고
||	및 또는 혹은 거나

이와 같은 단어는 사전에 등록된 단어만을 사용하게 된다. 무기체계와 같은 특수한 분야에서 제한적으로 사용하고 있는 요구사항은 등록된 단어만으로 충분히 표현 가능하며, 필요할 경우, 추가하여 사용할 수 있도록 한다.

Table 3과 같이 요구사항 규칙과 템플릿을 활용하여 추출된 데이터에서 입/출력 정보 파일과 연산자 정보 파일을 이용하여 해당하는 입/출력 변수 명과 연산자 기호로 변환하면 테스트 케이스 생성에 필요한 데이터를 추출한다.

예를 들어, Table 3의 요구사항과 추출된 데이터에서 입/출력 변수 명과 연산자 기호로 변환하면 다음 Table 7과 같이 변환 할 수 있다. Table 7의 요구사항 데이터는 Table 8과 같이 구조화된 데이터로 변환된다.

Table 7. The Converted Requirement

Extracted Data from Requirement	Converted Data
시스템A, 의, SensorA, 가, 200, 이상, 이고, SensorB, 가, 300, 이하, 이면, 시스템C, 의, OutputA,를, 가동, 한다.	SystemA SysIn_SensorA >= 200 && SysIn_SensorB <= 300, SystemC SysOut_OutputA == 1

Table 8. The Structured Requirement Data for Table 7

Data Structure		Data
Input Data	System	SystemA
	Input	SysIn_SensorA >= 200 && SysIn_SensorB <= 300
Output Data	System	SystemC
	Output	SysOut_OutputA == 1

2) 테스트 케이스 생성 전략

일반적으로 테스트 엔지니어는 요구사항에 작성된 내용만을 중점적으로 테스트한다. 이는 요구사항에 작성된 내용은 테스트가 가능하지만 요구사항에 작성되지 않은 그 밖에 오류가 발생할 수 있는 동작들은 테스트하기 어렵다. 따라서 요구사항에 작성되지 않았지만 오류가 발생할 수 있는 동작들에 대해 테스트 케이스를 생성할 필요가 있다.

하지만 테스트 케이스의 양이 많으면 많을수록 요구사항 이외의 동작들을 통해 시스템의 오류 가능성을 낮출 수 있지만 그만큼 개발 기간이 길어지고, 개발 비용 또한 증가한다. 이를 위해 효과적으로 테스트 케이스를 추출하는 것이 중요하다. 요구사항으로부터 추출된 데이터를 통해 요구사항에 작성되지 않았지만 오류가 발생할 수 있는 다양한 입력 조건에 의한 동작을 확인하기 위해 여러 가지 테스트 기법을 사용한다.

특히 전기, 전자 및 프로그램 가능한 전자 안전 관련 시스템의 기능 안전성에 대해 정의한 IEC61508 표준[17]에서 결정 커버리지, 조건 커버리지, 수정된 조건 결정 커버리지 등

Table 9. Test Case Examples for Different Test Strategies

Coverage	Test Case		
	No.	Input	Output
Decision Coverage	TC1	SensorA >= 200 && SensorB <= 300	OutputA == ON
	TC2	SensorA < 200 && SensorB <= 300	OutputA == OFF
Condition Coverage	TC3	SensorA >= 200 && SensorB <= 300	OutputA == ON
	TC4	SensorA < 200 && SensorB > 300	OutputA == OFF
MC/DC	TC5	SensorA >= 200 && SensorB <= 300	OutputA == ON
	TC6	SensorA < 200 && SensorB <= 300	OutputA == OFF
	TC7	SensorA >= 200 && SensorB > 300	OutputA == OFF

여러 수준의 테스트 커버리지를 만족할 것을 권고한다. 이를 위해 본 논문의 프로그램은 테스트 케이스 생성 시 결정 커버리지, 조건 커버리지, 수정된 조건 결정 커버리지를 적용하여 테스트 케이스를 생성하는 예를 보인다.

Table 8의 Input과 Output처럼 “SysIn_SensorA >= 200 && SysIn_SensorB <= 300”, “SysOut_OutputA == 1” 같은 데이터에서 조건 커버리지, 결정 커버리지, 수정된 조건 결정 커버리지를 적용하면 Table 9와 같은 테스트 케이스를 생성할 수 있다.

Table 9와 같이 생성된 테스트 데이터에서 적절한 입력 값은 SMT Solver인 Yices와 같은 방법[16]을 이용하여 찾을 수 있다. 본 연구에서는 Yices를 통해 추출한 입력 값을 테스트 스크립트로 사용한다.

3) 테스트 스크립트 생성

테스트 스크립트는 실제 테스트 대상 시스템의 물리적인 값을 의미한다. 일반적으로 테스트 스크립트는 테스트 엔지니어가 임베디드 시스템 하드웨어 정보를 정확히 모를 경우, 테스트 케이스로부터 적절한 스크립트를 만들기가 어렵고, 이에 따라 생성하는 데 시간이 소요된다. [14]에서 제시한 계층 구조를 이용하면 테스터 엔지니어가 시스템에 인가되는 입력 데이터인 테스트 스크립트를 쉽게 생성하다, 또한, 각 계층이 독립적으로 수정이 가능하고, 사전 관리가 매우 쉬워 재활용성이 높다는 장점이 있다.

테스트 스크립트를 생성하는 방법은 연구 [14]에서 제시한 방법인 명령어 사전을 이용한다. [14]의 명령어 사전은 이벤트(Event), 컴포넌트(Component), 입력 명령어(Command) 및 시스템 입력(System Input) 4단계 계층으로 구조화 되어있다.

이벤트 계층은 사용자 또는 환경에 의해 발생되는 행위를 나타내며, 컴포넌트 계층은 기능 단위의 부품 집합을 의미한다. 명령어 계층은 컴포넌트를 구성하는 명령어를 의미하고, 시스템 입력 계층은 해당 명령어를 수행하기 위해 실제 시스템에 인가되는 시스템의 물리적 입력 데이터이다.

본 논문에서는 테스트 케이스 생성 시, 실제 시스템 입/출력에서 사용하는 입력 및 출력이 [14]에서 제안하는 입력 명령어와 동일하여 입력 명령어 계층과 시스템 입력 계층을 이용하여 테스트 스크립트를 생성한다. 입력 명령어 사전은 테스트 대상 시스템에 사용되는 명령어가 정의된다. 각 명령어는 해당 명령어 수행에 사용되는 입력의 각 시간대 별 물리적인 값이 정의된다.

Fig. 6의 입력 명령어 사전 형식은 “명령어”, “TIME”, “시스템 입력 명”으로 구성된다. “명령어”열은 입/출력 정의 파일에 있는 입/출력 명을 의미하며, 명령어 이름과 매개변수(Parameter)가 함께 작성된다. 매개변수는 특수문자 “#”으로 시작한다.

Inst.	TIME	SysIn_SensorA
Inst.A(#p)	10	#p

Inst.	TIME	SysIn_SensorB
Inst.B(#g)	0	#g

Inst.	TIME	SysIn_SensorC
Inst.C(#a)	0	-50
	15	1000 * #a

Fig. 6. The Input Command Dictionary Format

SysIn_SensorA가 표현된 행의 “시스템 입력 명”은 테스트 대상 시스템에 인가되는 입력을 정의하는 공간으로 소수, 정수, 참/거짓 또는 매개변수를 입력할 수 있다. “시스템 입력 명”열에는 실제 명령어가 사용하는 신호 명으로 작성된다.

예를 들어, Fig. 6의 명령어A의 시스템 입력명은 ‘SysIn_SensorA’이고, 입/출력 정의 파일에 정의된 명령어A의 신호명도 ‘SysIn_SensorA’로 정의되어 있어야 한다.

[14]에는 “명령어”열의 매개변수도 복수개로 입력할 수 있고, 시스템 입력도 복수개로 정의할 수 있으나, 본 논문에서는 입/출력 정의 파일에 정의된 입/출력 명이 각각 1:1로 입/출력 신호명과 대응하여 사용되므로 복수로 사용하지 않는다.

“TIME”열은 테스트 스크립트가 인가되는 시간을 의미하며, 각 행에 적혀있는 시간 간격은 테스트 스크립트가 가해지는 상대시간이다.

Fig. 6에 정의된 명령어C(#a)의 시스템 입력명은 ‘SysIn_SensorC’로 정의되어있다. 테스트 케이스에서 생성된 명령어C의 값이 “200”이라면 #a에 200이 참조되어 명령어C(200)이 되고, 가장 처음 생성되는 스크립트(TIME이 0)는 “-50”이고, 15ms 이후에는 “1000*#a”로 생성된다. 앞서 #a는 200으로 참조되어 “1000*200 = 200000”의 값이 계산되어 스크립트가 생성된다.

Fig. 7. Test Case & Test Script Generator Screen Shot

- 시스템은 실시간 측정을 통하여 조도가 600 이하 일 경우 전조등을 On 한다.
- 시스템은 실시간 측정을 통하여 온도가 30 이하 일 경우 히터를 동작 시킨다.
- 시스템의 고도가 100m 미만 일 경우 Channel0으로 통신한다.
- 시스템의 고도가 100m 이상이고 300m 미만 일 경우 Channel1로 통신한다.
- 시스템의 고도가 300m 이상이고 500m 미만 일 경우 Channel2로 통신한다.
- 시스템의 고도가 500m 이상이고 1000m 미만 일 경우 Channel3으로 통신한다.
- 시스템의 고도가 1000m 이상일 경우 Channel4로 통신한다.
- 시스템의 타겟과의 거리가 1000m 이상 일 경우 미사일을 대기한다.
- 시스템의 타겟과의 거리가 200m 이상이고 1000m 미만 일 경우 미사일을 발사한다.

Fig. 9. Requirement Examples

4. 실 험

본 논문에서 제안하는 방법의 효과를 확인하기 위하여 제안된 방법으로 테스트 케이스 및 스크립트를 자동 생성 하는 프로그램을 구현하고 실험을 진행하고 그 결과를 분석하였다.

4.1 실험 환경

Fig. 7은 정형화된 요구사항으로부터 테스트 케이스 및 스크립트를 자동으로 생성하는 프로그램 화면이다. 프로그램은 C#으로 Windows 7 환경에서 구현하였다. 파일 선택 부분에서는 요구사항 문서, 요구사항에 사용되는 입/출력 정보 파일, 연산자 파일, 스크립트 생성을 위한 명령어 사전 파일이 사용되고, 최종으로 저장되는 출력 디렉토리를 지정할 수 있다. 테스트 케이스는 조건 커버리지, 결정 커버리지, 수정된 조건 결정 커버리지로 선택하여 생성되며, 만들어진 테스트 케이스를 통해 테스트 스크립트를 생성한다. 본 프로그램에서 처리가능한 문서 형식은 텍스트(.txt), 한글(.hwp), MS엑셀(.xlsx) 그리고 MS워드(.docx)이다.

테스트 대상 시스템은 가상의 전투기 시스템이다. 실험에서 사용한 시스템의 입출력 정보는 Fig. 8이며 요구사항은 Fig. 9와 같다. 각 요구사항은 '.'으로 구분되므로 총 9개의 요구사항으로 인식하고 각 요구사항은 parsing 하는 과정에서 Req. ID 1 ~ Req. ID 9가 부여된다.

구현된 프로그램은 요구사항으로부터 테스트 케이스 및 스크립트를 자동으로 생성하기 위해 정형화된 요구사항 파일, 요구사항에 사용되는 입/출력 정보 파일, 연산자 파일을 요구한다.

Fig. 9의 요구사항 사용하는 입/출력 정보는 Fig. 9와 같이 정의되었으며, 이들은 테스트 케이스 및 테스트 스크립트를 만들 때 필요로 되는 정보로, 테스트 대상 시스템의 입력과 출력의 정보를 정해진 포맷에 맞게 사용자가 입력하여야 한다.

Type	Name	Symbol	Range		Unit	DataType
			Min	Max		
엔진	조도	SysIn_Brightness	1	1000	flux	real
	온도	SysIn_Temp	-50	300	degree	real
	모드	SysIn_Mode	0	1		bool
궤도	고도	SysIn_Altitude	0	7000	m	int
	가속	SysIn_Accelation	-500	500	km/h	int
	시간	SysIn_Time	0	500	sec	int
포	Channel0	SysOut_Ch0	0	1		bool
	Channel1	SysOut_Ch1	0	1		bool
	Channel2	SysOut_Ch2	0	1		bool
	Channel3	SysOut_Ch3	0	1		bool
	거리	SysIn_Distance	0	2000	km	int
	전조등	SysOut_Lamp	0	1		bool
	미사일	SysOut_Missile	0	1		bool
	히터	SysOut_Heater	0	1		bool

Fig. 8. Input/Output Information for the Requirements

Table 10. The Structured Data to Generate Test Cases

Req.	Data Structure		Data
1	Input Data	System	System
		Input	SysIn_Brightness <= 600
	Output Data	System	System
		Output	SysOut_Lamp == 1
...	...	...	...
4	Input Data	System	System
		Input	SysIn_Altitude >= 100 && SysIn_Altitude < 300
	Output Data	System	System
		Output	SysOut_Ch1 == 1
...	...	...	...

실험에서 사용되는 연산자 파일은 앞 절의 Table 4, Table 5, Table 6과 동일하다. 프로그램에 요구되는 파일들을 이용하여 테스트 케이스 생성에 필요한 데이터를 추출할 수 있다.

Table 10은 전체 요구사항으로부터 추출한 테스트 케이스 생성에 필요한 요구사항 데이터 일부를 나타낸다. Table 10에서 보이는 바와 같이 요구사항(Req.)별로 데이터가 추출되며, 이를 기반으로 테스트 케이스 및 테스트 스크립트가 요구사항 별로 생성된다.

4.2 테스트 케이스 및 스크립트 생성

요구사항으로부터 추출된 Table 11의 요구사항 데이터, Fig. 8의 입출력 정보 그리고 Fig. 12의 명령어 사전 정보를 이용하여 테스트 케이스 및 스크립트를 생성한다. 본 논문에서 사용하는 테스트 케이스 생성 전략은 결정 커버리지, 조건 커버리지, 그리고 수정된 조건 결정 커버리지이다.

실험에서 사용된 프로그램은 요구사항의 입력 조건이 1개인 경우는 결정 커버리지를 적용하고 표현이 2개 이상인 경우는 조건 커버리지 및 수정된 조건 결정 커버리지를 적용한다. 실험에서 사용된 요구사항은 입력 조건이 최대 2개로 조건 커버리지와 수정된 조건 결정 커버리지 테스트 케이스가 동일하다. 결정 커버리지와 수정된 조건 결정 커버리지를 적용하여 생성된 테스트 케이스는 Fig. 10과 같으며, TestCaseID", "RequirementID", "Input", "Output"로 구성된다.

"TestCaseID"는 생성된 테스트 케이스 번호를 의미한다. "RequirementID"는 요구사항 ID이다. "Input"은 요구사항에서 작성된 입력 부분을 의미하며, 실제 테스트 시 사용되는 입력명과 입력 값이 생성되어 있다.

"Output"은 해당하는 테스트 케이스에 대한 예상 결과 정보를 의미한다. 해당 테스트 케이스의 입력에 따라 출력된 결과와 예상 결과 정보를 비교하여 테스트 결과를 확인할 수 있다.

출력된 값 중 "-"은 값이 변동하지 않음을 의미하며, 이전의 값을 유지한다. Fig. 10과 같이 생성한 테스트 케이스 값과 미리 시스템의 명령어 및 입력 값을 정의한 명령어 사전을 이용하여 테스트 스크립트를 자동으로 생성할 수 있다.

생성한 테스트 케이스 값 파일과 테스트 대상 시스템의 명령어 및 입력 데이터가 작성된 명령어 사전 파일을 함께 프로그램의 입력으로 넣어 테스트 스크립트를 생성한다.

본 논문에서 사용하는 명령어 사전 파일의 내용은 Fig. 11과 같다. 명령어 사전에는 요구사항에 작성된 입력에 상응하는 명령어, 명령어에 대한 신호 명, 명령어를 입력해야 하는 상대 시간(TIME)이 정의되어 있다. 본 실험에서 사용된 요구사항은 "고도", "조도", "온도", "거리"를 사용한다.

Fig. 12는 명령어 사전과 생성한 테스트 케이스를 기반으로 생성한 테스트 스크립트이다. "TC1"과 "TC2"는 첫 번째 요구사항에 대한 내용이며, "조도 <= 600", "조도 > 600"에 대한 테스트 케이스와 Fig. 11에 정의된 조도 명령어를 참조하여 테스트 스크립트가 생성된다. 명령어 테이블의 "조도"는 "SysIn_ Brightness" 시스템 입력을 정의하고 있다. "SysIn_Brightness"의 입력 신호는 "1000*#b"으로 정의되어 있으며, "#b" 값은 앞서 Fig. 10의 테스트 케이스에 의해 생성된 값 "1"과 "601"로 치환되어 "1000*1"인 "1000", "1000*601"인 "601000"로 계산된 것을 확인할 수 있다. 또한 "TIME" 은 Fig. 11의 "조도"의 정의한 "TIME"을 참조하여 "0"으로 생성되었다.

"TC5"와 "TC6"은 세 번째 요구사항에 대한 테스트 스크립트이며, "고도 < 100", "고도 >= 100"에 대한 테스트 케이스와 Fig. 11에 정의된 고도 명령어를 참조하여 테스트 스크립트가

TestCaseID	RequirementID	Input				Output							
		SysIn_Brightness	SysIn_Temp	SysIn_Altitude	SysIn_Distance	HeadLight	Heater	Channel0	Channel1	Channel2	Channel3	Channel4	Missile
TC1	R1	1	-	-	-	0	-	-	-	-	-	-	-
TC2	R1	601	-	-	-	1	-	-	-	-	-	-	-
TC3	R2	-	0	-	-	-	1	-	-	-	-	-	-
TC4	R2	-	31	-	-	-	-	-	-	-	-	-	-
TC5	R3	-	-	0	-	-	-	1	-	-	-	-	-
TC6	R3	-	-	100	-	-	-	-	-	-	-	-	-
TC7	R4	-	-	100	-	-	-	-	1	-	-	-	-
TC8	R4	-	-	0	-	-	-	-	-	-	-	-	-
TC9	R4	-	-	300	-	-	-	-	-	-	-	-	-
TC10	R5	-	-	300	-	-	-	-	-	1	-	-	-
TC11	R5	-	-	0	-	-	-	-	-	-	-	-	-
TC12	R5	-	-	500	-	-	-	-	-	-	-	-	-
TC13	R6	-	-	500	-	-	-	-	-	-	1	-	-
TC14	R6	-	-	0	-	-	-	-	-	-	-	-	-
TC15	R6	-	-	1000	-	-	-	-	-	-	-	-	-
TC16	R7	-	-	1000	-	-	-	-	-	-	-	1	-
TC17	R7	-	-	0	-	-	-	-	-	-	-	-	-
TC18	R8	-	-	-	1000	-	-	-	-	-	-	-	0
TC19	R8	-	-	-	0	-	-	-	-	-	-	-	1
TC20	R9	-	-	-	200	-	-	-	-	-	-	-	1
TC21	R9	-	-	-	0	-	-	-	-	-	-	-	-
TC22	R9	-	-	-	1001	-	-	-	-	-	-	-	-

Fig. 10. The Generated Test Cases Adequate to MCDC

Inst.	group	TIME
TIME(#t)		#t

Inst.	group	TIME	SysIn_Altitude
Altitude(#a)		0	-50
		15	1000 * #a

Inst.	group	TIME	SysIn_Speed
Speed(#s)		0	5000 + #s

Inst.	group	TIME	SysIn_Atm
Atm(#a)		0	#a

Inst.	group	TIME	SysIn_Gyro
Gyro(#g)		0	10 * #g

Inst.	group	TIME	SysIn_Mode
Mode(#m)		0	#m

Inst.	group	TIME	SysIn_Launch
Launch(#l)		0	#l

Inst.	group	TIME	SysIn_Distance
Distance(#d)		1000	#d * 511 / 600 * 5 / 1023

Inst.	group	TIME	SysIn_Brightness
Brightness(#b)		0	1000 * #b

Inst.	group	TIME	SysIn_Temp
Temp(#t)		10	100 * #t

Fig. 11. The Command Dictionary for the Experiment

	SysIn_Altitude	SysIn_Distance	SysIn_Brightness	SysIr_Temp	TIME
TC 1	-	-	1000	-	0
TC 2	-	-	601000	-	0
TC 3	-	-	-	0	10
TC 4				3100	10
TC 5	-50	-	-	-	0
	0	-	-	-	15
TC 6	-50	-	-	-	0
	100000	-	-	-	15
TC 7	-50	-	-	-	0
	100000	-	-	-	15
TC 8	50				0
	0	-	-	-	15
TC 9	-50	-	-	-	0
	300000	-	-	-	15
TC 10	-50	-	-	-	0
	300000	-	-	-	15
TC 11	-50	-	-	-	0
	0	-	-	-	15
TC 12	-50	-	-	-	0
	500000	-	-	-	15
TC 13	-50	-	-	-	0
	500000	-	-	-	15
TC 14	-50	-	-	-	0
	0	-	-	-	15
TC 15	-50	-	-	-	0
	1000000	-	-	-	15
TC 16	-50	-	-	-	0
	1000000	-	-	-	15
TC 17	-50	-	-	-	0
	0	-	-	-	15
TC 18	-	4.162593679	-	-	1000
	-	0	-	-	1000
TC 19	-	0.832518736	-	-	1000
	-	0	-	-	1000
	-	4.166756272	-	-	1000

Fig. 12. Generated Test Scripts

생성된다. 명령어 테이블의 “고도”는 “SysIn_Altitude”시스템 입력을 정의하며, “SysIn_Altitude”의 입력 신호는 “TIME”이 0초 일 때 “-50”의 값을 갖고, 15ms 이후 “1000*#a”로 정의되어 있다. 본 테스트 스크립트에서 “#a”에 참조되는 값은 Fig. 10과 같이 MC/DC 기반 테스트 케이스에서 생성된 “SysIn_ Altitude”의 “0”과 “100”이 된다. “TC5”에 생성된 테스트 스크립트는 가장 처음 “SysIn_Altitude”에 “-50”의 값이 인가되고, 15ms 이후 “1000*0”인 “0” 값이 인가된다. “TC6”에 생성된 테스트 스크립트는 실행하자마자“SysIn_Altitude”에 “-50”이 인가되고, 15ms 이후 “1000*100”인 “100000”값이 인가된다.

이와 같이 계층 구조를 이용하여 “조도”, “온도”, “거리”와 같이 입력 명령어 당 시간 별 하나의 값을 갖는 테스트 스크립트를 생성할 수 있으며, “고도”와 같이 시간 별 다수의 값을 갖는 테스트 스크립트도 쉽게 생성할 수 있다.

4.3 실험 결과 분석

실험을 통하여 본 논문에서 제안하는 방법으로 제한된 문장 구조를 가진 요구사항, 요구사항에 사용되는 입출력 정보 그리고 스크립트 생성을 위한 명령어 사전 정보를 이용하여 요구사항으로부터 테스트 케이스 및 테스트 스크립트 생성까지 가능함을 보였다. 본 실험에서 사용된 요구사항 및 입출력 정보 등은 국방 분야의 특정 무기 체계에 사용하는 요구사항과 매우 유사하다. 이는 본 연구에서 제안한 방법으로 특정 분야에서 제약을 둔 문장 구조로 작성된 자연어로 작성된 요구사항부터 테스트 케이스 생성이 가능함을 보인 것이라 하겠다.

하지만 본 논문이 출발점인 제한된 구조를 가진 자연어 요구사항의 범위를 넘어서는 사용하기 힘들다. 따라서 본 연구의 결과를 새로운 분야에 적용하기 위해서는 해당 분야에 적절한 문장 구조를 개발하여야 한다는 한계점을 가진다. 본 연구의 결과를 확장하여 사용하기 위해서는 보다 다양한 문장 구조 및 다양한 단어를 처리할 수 있게 하여야 한다.

5. 결 론

본 논문은 자연어로 작성된 요구사항을 요구사항 규칙에 맞게 정형화하고, 정형화된 데이터로부터 응용 분야에서 요구하는 테스트 전략을 적용하여 테스트 케이스 값과 테스트 스크립트를 효율적으로 생성하는 방법을 제안하였다.

요구사항으로부터 빠른 시간 내에 효율적으로 테스트 케이스를 생성하기 위해 요구사항을 정형화하였으며, STK 규칙 및 Template을 통해 자연어가 가진 모호함과 복합성을 줄일 수 있다. 또한 추출된 테스트 데이터를 통해 정형화된 수

식을 얻을 수 있고, 정형화된 수식은 Simulink나 UML에서 사용하는 포맷으로 쉽게 변경할 수 있어, 이는 모델링 생성에 사용할 수 있다[15].

추출된 데이터를 통해 결정 커버리지, 조건 커버리지, 수정된 조건 결정 커버리지 등 다양한 테스트 전략을 통해 테스트 엔지니어가 예측하기 어렵고 요구사항에 작성되지 않았으나, 요구사항 입력의 다양한 변형된 조건에 오류 발생 가능성이 있는 동작에 대한 다양한 테스트 케이스를 생성할 수 있다.

생성된 테스트 데이터는 SMT Solver인 Yices를 통해 자동으로 유효범위 값을 추출할 수 있다. 추출된 값과 계층 구조의 명령어 및 시스템 입력으로 테스트 엔지니어가 시스템 하드웨어 입력에 대한 자세한 지식 없이 빠른 시간 내에 테스트 스크립트 생성이 가능함을 실험을 통해 보였다. 계층 구조의 명령어 및 시스템 입력 정보는 독립적으로 쉽게 수정이 가능하여 사전 관리가 쉽고, 문서화되어 있어 재활용성이 매우 높다. 이는 명령어 사전을 통해 테스트 엔지니어가 실제 물리적인 값을 몰라도 쉽게 스크립트를 생성할 수 있도록 도와준다.

본 논문에서는 제안한 방법을 이용하여 제한된 표현을 사용하는 간단한 임베디드 시스템 요구사항에서는 적용 가능성을 보였다. 하지만 실제 복잡한 임베디드 시스템에 적용하기 위해서는 복잡한 요구사항으로부터 테스트 케이스 및 테스트 스크립트를 생성할 수 있도록 하는 추가 연구가 필요하다.

References

[1] Rolf Schwitter, “Controlled Natural Languages for Knowledge Representation,” Coling 2010: Poster Volume, Beijing, pp. 1113-1121, 2010.

[2] 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*, pp.489-495, 2001.

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

[4] Rick Kuhn, Raghu Kacker, Yi Lei, and Justin Hunter, “Combinatorial Software Testing,” *IEEE Computer Society*, Vol.42, Iss.8, pp.94-96, Aug. 2009.

[5] 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.

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

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

[8] C. Denger and M. Mora, “Test case derived from Requirement Specifications,” IESE-Report No.033.03/E version 1.0, Apr. 24 2003.

[9] Kang Tae Hoon, Kim Dae Joon, Chung Ki Hyun, and Choi Kyung Hee, “A Method to Automatically Generate Test Scripts from Checklist for Testing Embedded System,” *KIPS*, Vol.5, No.12, pp.641-652, May 2016.

[10] K. W. Kwon, K. H. Chung, H. S. Yang, J. J. Jang, D. S. Lee, U. H. Jo, J. H. Shin, and H. J. Cho, “Rules of SRS for the SRS Adequacy Test,” KIMST, 2018.

[11] A. P. Mathur, “Foundations of Software Testing,” Pearson Education, 2008.

[12] Farrell, James A. (August 1995), “Compiler Basics: Extended Backus Naur Form,” Archived from the original on 5 June 2011. Retrieved May 11, 2011.

[13] KOMORAN [Internet], https://github.com/shin285/KOMORAN

[14] Dae Joon Kim, Ki Hyun Chung, and Kyung Hee Choi, “A Hierarchical Checklist to Automatically Generate Test Scripts,” *KIPS*, Vol.6, No.5, pp.245-256, Jun. 2017.

[15] H. D. Kim, H. J. Cho, J. H. Shin, K. H. Chung, and K. H. Choi, “Automatically Generating Simulink/Stateflow Model and Test Case from Requirements,” KIMST, 2017.

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

[17] Functional-Safety [Internet], https://www.iec.ch/functionalsafety/

[18] ASD Simplified Technical ENGLISH ASD-STE100 [Internet], http://www.asd-ste100.org/

[19] Alun Preece, Diego Pizzocaro, Dave Braines, David Mott, Geeth de Mel, and Tiem Pham, “Integrating Hard and Soft Information Sources for D2D Using Controlled Natural Language,” In *2012 15th International Conference on Information Fusion*, pp.1330-1337, Jul. 2012.

[20] FlaLavia A. Barros, Lai.s Neves, L Erica Hori and Dante Torres, “The ucsCNL: A Controlled Natural Language for Use Case Specifications,” in SEKE, pp.250-253, 2011.

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
1984년 서강대학교 전자공학과(학사)
1988년 미국 Ilinois주립대 EECS(석사)
1990년 미국 Purdue대학 전기전자공학부 (박사)
1993년~현 재 아주대학교 전자공학과 교수
관심분야 : 컴퓨터구조, VLSI설계, 실시간 시스템, 테스팅

최 경 희

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