KIPS Tr. Software and Data Eng.
Vol.6, No.8 pp.401~410 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2017.6.8.401

An Automatic Test Case Generation Method from Checklist

Hyun Dong Kim[†] · Dae Joon Kim[††] · Ki Hyun Chung[†††] ·
Kyung Hee Choi[††††] · Ho Joon Park[†††††] · Yong Yoon Lee[††††††]

ABSTRACT

This paper proposes a method to generate test cases in an automatic manner, based on checklist containing test cases used for testing embedded systems. In general, the items to be tested are defined in a checklist. However, most test case generation strategies recommend to test a system with not only the defined test items but also various mutated test conditions. The proposed method parses checklist in Korean file and figures out the system inputs and outputs, and operation information. With the found information and the user defined test case generation strategy, the test cases are automatically generated. With the proposed method, the errors introduced during manual test case generation may be reduced and various test cases not defined in checklist can be generated. The proposed method is implemented and the experiment is performed with the checklist for an medical embedded system. The feasibility of the proposed method is shown through the test cases generated from the checklist. The test cases are adequate to the coverages and their statistics are correct.

Keywords : Embedded System, Test Case, Medical Device, Checklist, Testing

한글 체크리스트로부터 테스트 케이스 자동 생성 방안

김 현 동[†] · 김 대 준[††] · 정 기 현[†††] · 최 경 희[††††] · 박 호 준[†††††] · 이 용 윤[††††††]

요 약

본 논문에서는 임베디드 시스템 테스트에 많이 사용되는 테스트 항목이 기술된 체크리스트를 기반으로 테스트 케이스를 자동으로 생성하기 위한 방법을 제안한다. 일반적으로 체크리스트에는 테스트하고자 하는 항목이 정의되어 있다. 하지만 대부분의 테스트 케이스 생성 전략에서는 테스트 하고자하는 항목뿐만 아니라 다양한 변이 조건에서도 테스트하기를 권하고 있다. 제안하는 방법은 한글로 기술된 체크리스트를 분석하여 시스템 입출력과 연산 정보를 찾아낸다. 그리고 찾아낸 정보와 설정하는 테스트 케이스 생성 전략에 따라 테스트 케이스를 자동으로 생성한다. 이 방법은 테스트 케이스를 수동으로 생성할 때, 일으킬 수 있는 오류를 줄일 수 있을 뿐만 아니라, 체크리스트에는 기술되지 않는 다양한 변이 테스트 케이스도 생성할 수 있다. 제안된 방법은 구현되고, 실제 의료기기용 임베디드 시스템의 체크리스트를 사용하여 실험을 진행한다. 실험에서는 체크리스트로부터 여러 커버리지에 적절한 테스트 케이스가 오류 없이 통계적으로 정확히 생성되었음을 확인할 수 있어서 제안된 방법의 유용성을 보여준다.

키워드 : 임베디드 시스템, 테스트 케이스, 의료 장비, 체크리스트, 테스팅

1. 서 론

최근 모바일 기기, 가전제품, 군사용 무기, 의료기기, 자동차 등에서 사용되는 임베디드 시스템은 과거에 비해 점점 더 복잡한 기능을 수행하고 있다[20]. 복잡한 시스템의 기능은 대부분 소프트웨어로 구현된다. 따라서 임베디드 시스템이 동작하면서 발생하는 오류의 많은 부분은 소프트웨어적인 오류이다[1]. 소프트웨어 오류는 시스템의 오류로 이어지고 그 영향은 간단한 장애로부터 중대한 장애까지 일으킬 수 있다. 예를 들면, 의료기기 시스템이 오류를 발생할 경우 생명에 위해를 가할 수 있는 큰 재앙을 불러오거나, 주어진 임무를 올바르게 수행할 수 없는 경우도 초래될 수 있을 것이다. 이런 경향과 더불어 임베디드 시스템에 내장된 소프트웨어 테스트는 그 중요성이 점점 강화되고 있다[21].

※ 본 연구는 2015년도 식품의약품안전처의 연구개발비(15172신기평403)로 수행되었음.
† 준 회 원 : 아주대학교 전자공학과 석사과정
†† 준 회 원 : 아주대학교 컴퓨터공학과 석사
††† 정 회 원 : 아주대학교 전자공학과 교수
†††† 정 회 원 : 아주대학교 컴퓨터공학과 교수
††††† 비 회 원 : 한국산업기술시험원 의료기기연구센터장/수석연구원
†††††† 비 회 원 : 한국산업기술시험원 의료헬스본부 팀원
Manuscript Received : December 2, 2016
First Revision : May 24, 2017
Accepted : June 17, 2017
* Corresponding Author : Ki Hyun Chung(khchung@ajou.ac.kr)

임베디드 소프트웨어를 테스트하기 위한 많은 기법들이 제안되어 있다. 크게는 소프트웨어의 코드를 직접적으로 테스트하는 화이트 박스(White Box)와, 스펙과 기능을 중심으로 테스트하는 블랙박스(Black Box) 기법으로 나눌 수 있다. 화이트 박스는 소스 코드를 기반으로 테스트 케이스를 자동 혹은 수동으로 작성하고 코드가 설계 사양에 맞게 올바르게 동작하는지 혹은 코딩 규칙을 제대로 지키고 있는지 등을 확인하는 용도로 사용한다. 소프트웨어 정적 및 동적 테스트 기법[19]이 이에 속한다. 화이트 박스 테스트를 위한 다양한 상용도구들[19]도 개발되어 있다.

이에 비해 블랙박스 테스트는 소프트웨어가 내장된 시스템이 요구사항에 나타난 기능을 제대로 수행하는지를 테스트하는 용도로 사용한다. 그렇기 때문에 요구사항을 기반으로 다양한 동작을 확인할 수 있도록 효율적인 테스트 케이스를 작성하는 것이 필요하다. 일반적으로 시스템 개발 시 화이트 박스 테스트는 소프트웨어 작성 후에 수행하고, 블랙박스 테스트는 개발된 소프트웨어를 하드웨어에 탑재한 후 통합 테스트를 하기 위해 많이 사용한다[12].

화이트 박스 테스트이거나 블랙박스 테스트에서 가장 중요한 문제 중의 하나는 효율적인 테스트 케이스를 생성하는 것이다. 이론적으로 거의 무한대의 가까운 입력 조합의 개수 중에서 소프트웨어 혹은 시스템의 오류를 가장 잘 찾아낼 수 있는 최소한의 테스트 케이스를 만들어 내는 것이 테스트 케이스 생성의 핵심이다. 화이트 박스 테스트에서는 코드를 분석하여 테스트 케이스를 생성한다. 블랙박스 테스트 케이스를 생성하는 방법은 다양하다. 요구사항 모델을 이용하는 모델 기반 테스트 케이스 생성 방법, 시스템 입력들의 필요에 따라 조합하여 사용하는 테스트 목적에 맞게 생성하는 다양한 입력조합(Combinatorial) 생성 방법[12], 의도된 무작위 생성 방법(Biased random) [12]등이 있다.

이와 같은 체계적인 생성 방법 이외에 개발 현장에서 흔히 사용하는 방법이 체크리스트(Checklist)를 작성하여 이를 기반으로 테스트하는 방법이다. 체크리스트는 개발 시스템에 대한 엔지니어의 지식, 과거의 경험, 사용자 피드백 등을 활용하여 테스트하고자 하는 항목을 엔지니어가 작성한다,

체크리스트에는 테스트 대상의 기능을 점검하기 위해 테스트해야 하는 입력 조건과 예상 출력을 나열해 놓은 목록. 즉, 테스트해야 하는 항목이 나열되어 있다. 그리고 테스터 엔지니어는 체크리스트를 참조하여 테스트 입력을 결정한다. 이러한 방법은 크게 두 가지 문제점이 있다. 1) 수많은 체크리스트 항목을 보고 테스트 케이스를 작성 도중. 테스터의 실수 또는 판단 착오로 인하여 잘못된 테스트 케이스를 생성할 수 있다. 2) 체크리스트에 작성된 항목은 테스트해야 하는 기능만을 나열하고 있어서, 만들어진 소프트웨어 혹은 시스템이 체크리스트에 없는 입력이 가해지는 이상 상황에서 제대로 동작하는 지를 체계적으로 테스트하기 힘들다. 많은 테스트 방법 및 전략에서는 엔지니어가 생성한 테스트 항목 혹은 시스템 사양에 나타나지 않는 변이 입력 조합에 대해서 테스트하기를 권하고 있다. 임베디드 시스템의 기능이 다양하고 복잡하여 체크리스트 항목과 내용이 복잡해지는 경우 위의 문제점을 엔지니어가 수동 분석을 통해 해결하기에는 많은 노력과 시간이 필요하고 경우에 따라서는 거의 불가능하기도 한다. 또한 시스템 요구사항이 바뀌고 이에 따라 체크리스트가 변할 시 수동으로 테스트 케이스를 생성하는 작업을 다시 해야 하는 데 이에 따른 시간과 비용 부담은 점점 커진다.

본 논문에서는 이러한 단점을 보완할 수 있는 체크리스트로부터 테스트 케이스를 직접 생성하는 방안을 제시한다. 제안하는 방법에서는 테스트 용도나 테스트 시간을 고려하여 용도에 따른 테스트 케이스를 자동으로 생성한다. 제안하는 방법에서는 한글 문서로 된 체크리스트를 해석(parsing)하고, 해석된 내용에서 사용자가 정의한 시스템 입출력 정보, 문서 해석 정보 내의 연산 정보 및 테스트 케이스 생성 전략을 이용하여 용도에 맞는 테스트 케이스 생성한다.

본 논문의 구성은 다음과 같다. 제 2장은 일반적인 테스트 생성 방법 및 체크리스트 위한 테스트 케이스 생성 방법에 대한 관련 연구를 소개하고, 제 3장은 체크리스트를 해석하고 테스트 케이스를 생성하는 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

화이트 박스 테스트 케이스는 소프트웨어 소스 코드의 코딩 규칙이나 실행 오류(Run time error) 등을 파악하기 위해 설계 단계에서 많이 사용한다. 이 방법은 시스템의 요구사항과 일치 여부를 검증하는데 사용하기는 어렵다.

반면 블랙박스 테스트 케이스는 요구사항이나 시스템 사양 등을 기반으로 작성된다. 따라서 블랙박스용 테스트 케이스는 하드웨어에 내장된 소프트웨어가 요구사항대로 동작하는 지를 테스트하기 위해서 사용된다. 이미 다양한 블랙박스 테스트 케이스 생성 방법이 연구되어 왔다. 블랙박스 테스트 케이스 생성 방법은 무작위 생성 방법, 시스템 입력 조합을 이용하는 생성 방법(Combinatorial)[16] 및 모델 기반(Model-based) 생성 방법[5, 10, 11] 등으로 대별할 수 있다. 무작위 생성 방법[6]에는 의도된 무작위 생성 방법[4]과 다양한 적응형 무작위 생성(Adaptive Random Generation) 방법[7] 등이 있다[2]. 입력 조합에 의한 생성 방법에는 페어와이즈(Pairwise)[3, 15] 및 클래스 분류 트리

방법(Classification Tree Method)[8, 9] 등이 연구되어 왔다. 모델 기반 생성 방법은 시스템의 요구사항 혹은 기능을 모델화하고 모델로부터 테스트 케이스를 생성하는 방법이다. 이외에도 동등 분할(Equivalence partitioning) 기법, 변이 값(Mutant) 주입을 통한 테스트 방법(Mutation) 및 경계 값 분석을 통한 생성 방법(Boundary value analysis) 등 다양한 방법[12] 있다.

소프트웨어가 내장된 시험 대상 시스템(SUT: System Under Test)의 완벽히 테스트하기 위해서는 무한대의 입력 조합 즉, 무한개의 테스트 케이스를 이용하여 테스트를 실시하여야 한다. 하지만 이와 같은 무한 테스팅(Exhaustive testing)은 현실적으로 불가능하다[12]. 따라서 실제 시스템의 효율적인 테스트를 위해서는 앞에서 소개한 구조적인 테스트 케이스 생성 방법과 더불어 엔지니어의 경험과 현장에서의 피드백을 고려하여 테스트 케이스를 생성하는 것이 바람직하다. 실제로 의료기기나 자동차 제어기 등을 시험하기 위해서 엔지니어의 경험을 기반으로 테스트 케이스를 생성하여 사용한다. 경험 기반 테스트 케이스는 구조적 방법으로 생성된 테스트 케이스가 찾아내지 못하는 시스템의 특성이나 복잡한 시나리오에 의해 발현되는 오류를 발견할 가능성이 높다.

엔지니어가 수동으로 테스트 케이스를 생성하여 테스트하기 위해 많이 사용하는 것이 체크리스트이다. 체크리스트는 테스트 엔지니어가 다루었던 유사한 시스템이나, 과거의 경험, 직관, 테스트 엔지니어의 지식, 사용자의 피드백을 활용하여 테스트 하고자하는 항목을 포함하고 있다. 응용 분야, 기관 그리고 엔지니어 마다 다른 형태의 체크리스트를 사용하고 있다. 하지만 그 형태와 무관하게 모든 체크리스트의 각 테스트 항목에는 어떤 입력 조건에서 어떤 출력이 나와야한다는 것이 명시되어 있다.

우리가 아는 지식으로는 체크리스트를 활용하여 체계적으로 테스트 케이스를 생성하는 방법은 많지 않다. 체크리스트와 유사한 방법으로 유스 케이스(Use case)를 활용한 테스트 시나리오 생성법과 고장 트리(FT: Fault tree)를 활용한 방법이 있다. [13]에서는 유스 케이스 다이어그램(Use case diagram)을 활용하여 유스 케이스 모델을 만들고, 모델을 활용하여 유스 케이스 리스트를 만들고 이를 테스트 시나리오로 사용한다.

고장 트리는 오류를 일으키는 원인과 관계를 AND, OR, Priority AND, XOR 등의 논리 연산자를 이용하여 트리 형태로 표현하는 방법이다[14]. 고장 트리를 활용하면 결과 이벤트 중 일부를 오류 발생 상황으로 만들고 이 상황이 발생할 수 있는 시나리오를 고장 트리에서 추적하여 생성할 수 있고 이를 테스트 케이스로 사용한다. 즉, 오류 발생의 경로를 결과 이벤트로부터 거슬러 올라가면 초기 상태부터 결과 이벤트에 이르는 시나리오를 역방향으로 생성할 수 있다.

연구[17]에서는 체크리스트 기반으로 테스트 케이스를 생성하는 방법에 대해 다룬다. 이 연구에서는 효율적으로 체크 항목을 작성할 수 있는 체크리스트의 체계를 제안하고, 이를 기반으로 체크리스트를 생성한다.

제안하는 체크리스트 체계는 “동작모드”, “기능분류”, “기능설명”, “테스트 명령” 으로 이루어져 있다. 이들 중 “동작모드”, “기능설명” 항목은 실제 테스트 스크립트 생성과 무관하며, “기능분류”와 “테스트 명령”을 사용하여 테스트 케이스를 만들고, 실제 테스트에 사용되는 테스트 스크립트를 자동으로 생성한다.

“기능분류”는 각 동작 모드 내에서 수행하는 기능을 말하며 “진입 시 조건”, “종료조건” 그리고 “주동작” 항목으로 구성된다. “진입시조건”은 해당 모드에 진입하기 위하여 만족시켜야하는 조건이다. “진입 시 조건”은 일부 혹은 전부 만족 시켜야 “주동작”으로 진입할 수 있다. “주동작”은 항상 실행 되어야하는 일련의 작업들이다. “종료조건”은 해당 모드를 종료하기 위한 사건이나 조건들이다.

SUT가 수행해야 할 단위 기능인 테스트 명령어에는 각 시간대 별로 SUT에 인가되는 모든 입력들의 물리적 값들이 정의되어 있다. 이들 명령어는 명령어 사전에 등록되어 테스트 케이스 및 스크립트 생성에 사용된다.

이 연구에서는 “진입 시 조건”과 “종료 조건”의 조합을 조건 커버리지(Condition coverage), 결정 커버리지(Decision coverage), 수정된 조건 결정 커버리지(MCDC: Modified Condition and Decision Coverage)와 같은 기준에 맞게 생성하여 테스트 케이스로 사용한다. 또한 동작 모드 별로 다양한 조합을 자동으로 생성하게 하여 모드 간 실행 순서에 따른 시스템 오동작 여부도 검출 할 수 있는 테스트 케이스를 생성한다.

하지만 이 방법은 체크리스트를 제안된 양식으로 정확하게 정리하여야 하고, 제안된 도구를 사용하여야 하므로 일반적인 자연어 문서로 만들어진 체크리스트에는 적용할 수가 없다.

연구[22]는 연구[21]에서 테스트 스크립트를 좀 더 효율적으로 생성할 수 있도록 계층 구조를 사용하여 테스트 스크립트를 생성하는 방법을 제안하고 있다. 하지만 여전히 요구사항으로부터 유연하게 테스트 케이스를 생성할 수 있는 구조는 아니다.

3. 한글 체크리스트 기반 테스트 케이스 생성

체크리스트는 시스템을 테스트할 때 확인해야하는 기능이나 동작을 기술한 문서이다. 체크리스트 기반 테스트에서 엔지니어는 시스템이 정상적으로 동작하는가를 확인하기 위해 체크리스트에 명시된 항목을 순차적으로 실행하며 테스

트를 진행한다. 엔지니어가 체크리스트를 통하여 테스트를 진행 할 경우 엔지니어는 시스템에 입력시킬 테스트 케이스가 필요하게 된다.

같은 체크리스트를 사용하더라도 엔지니어마다 테스트 케이스를 다르게 생성할 수 있고 이렇게 다르게 생성된 테스트 때문에 시스템의 정확한 테스트가 이루어지지 않을 수도 있다. 또한 같은 테스트 케이스를 사용하더라도 테스트 결과가 달라질 수 있다. 즉, 테스트가 엔지니어의 능력에 따라 달라질 수 있고 이는 산업 현장에서는 관리의 문제점이 될 수도 있다. 이러한 문제를 해결하기 위해 체크리스트로부터 테스트 케이스를 체계적으로 만드는 방법이 필요하다. 자연어로 된 체크리스트를 엔지니어가 해석하여 테스트 케이스를 만드는 과정에서 유입될 수 있는 오류를 줄일 수 있고, 엔지니어마다 다른 결과를 내는 것을 방지하기 위한 체계적인 방법의 하나로, 본 논문에서는 자연어로 작성된 체크리스트를 해석(parsing)을 통하여 테스트 케이스를 자동 생성하는 방법을 제안한다.

테스트 진행 시 SUT의 입력인 테스트 케이스뿐만 아니라 필요에 따라서는 테스트 입력에 대한 예상 결과 정보도 생성한다. 예상 결과를 만들어내는 시스템을 통칭하는 오라클도 테스트 자동화를 위해 반드시 필요한 요소이다.

본 논문에서는 자연어로 작성된 체크리스트를 파싱을 통해 얻은 정보로, 테스트 입력을 테스트 전략에 따라 조합한 테스트 케이스와 각 테스트 케이스에 대한 예상 출력을 시스템 출력으로 자동으로 생성하여 오라클로 사용할 수 있는 방법에 대해 다룬다.

3.1 자연어로 작성된 체크리스트 파싱

체크리스트는 테스트하는 주체에 따라 다양한 형식을 사용하고 있다. 또한 체크리스트 내에 포함된 테스트 항목도 다양하다. 형식은 다양하지만 형식과 무관하게 각 테스트 항목은 기본적으로 크게 입력 파트와 출력 파트로 나누어질 수 있다. 입력 파트는 시스템에 인가할 입력의 종류와 조합 조건을 정의한 파트이고 출력 파트는 해당 입력이 인가되었을 때 출력이 어떻게 되는가를 정의하고 있다. 예를 들어 “A일 경우 B 하여야 한다.”라는 문장이 있다면 “경우”를 기준으로 입력 파트와 출력 파트가 나뉘게 된다. 여기서는 A가 입력 파트이고 B가 출력 파트가 된다.

일반적으로 한글로 작성된 체크리스트의 입력 파트와 출력 파트에는 입출력의 이름, 조건이 있고 입출력의 조건에는 연산자인 비교어, 명령어, 접속어가 있다. 비교어는 “크다(>)”, “작다(<)”, “아니다”, “~값”과 같이 주어진 숫자 혹은 정해진 값을 기준으로 입출력의 값이 큰 경우인지 작은 경우인지 혹은 정확히 주어진 값이 나와야 하는가를 구분하게 해준다. 명령어는 “동작한다”, “실행한다”, “제어한다” 등의 시스템 동작에 관한 사양이다. 접속어는 두 개 이상의 조건이 한 번에 나올 때 A조건과 B조건이 동시에 만족해야 하는지(and) 아니면 둘 중 하나만 만족해도 되는지(or) 등을 나타낸다. 물론 한글의 특성상 비교어나 접속어의 다양한 변형이 나올 수 있다. 예를 들면, “크다” 대신, “크거나”, “클 때”, “크면서” 등 어미가 변형된 단어가 체크리스트 작성 시 사용될 수 있다.

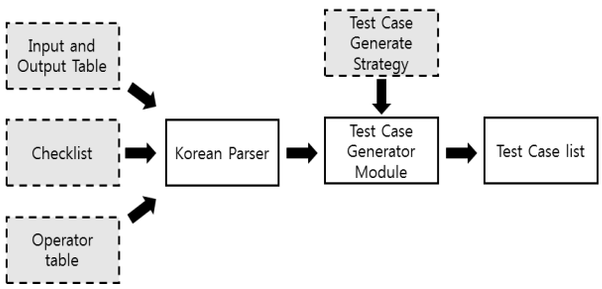


Fig. 1. Test Case Generation Flow in the Proposed Method

제안하는 방법은 크게 두 가지 테이블과 두 가지 기능 모듈로 구성된다. 첫 번째 테이블은 사용자가 정의한 시스템 입출력과 해당 변수 정보를 가지고 있는 ‘입출력 정보 테이블’이고, 두 번째 테이블은 비교어, 명령어와 접속어 및 그들의 변형어 들과 해당하는 로직들이 정의되어 있는 ‘연산자 정의 테이블’이다. 그리고 첫 번째 기능 모듈은 두 가지 테이블과 한글 파서(parser)를 이용하여 자연어로 된 입출력을 변수로 사용한 로직이나 수식 관계로 표현하는 “한글 파서 모듈”이다. 두 번째 기능 모듈은 첫 번째 모듈에서 처리된 표현을 이용하고 사용자가 정하는 테스트 케이스 생성 전략에 따라 테스트 케이스를 생성하는 “테스트 케이스 생성모듈”이다. Fig 1은 테이블과 모듈을 활용하여 테스트 케이스 리스트가 생성되는 흐름을 보여 준다.

입출력 정보 테이블에서는 사용자가 정의하는 시스템의 입력과 출력을 포함한다. 테이블 각 항은 모든 한글 입출력 명과 각 입출력의 변수(variable)명이 정의된다. 이 입출력 정보는 한글 파서에서 참조되고 테스트 케이스에서 변수명으로 나타난다. 연산자 정의 케이블에는 “크다”, “적다”와 같은 정보와 이에 해당하는 연산자가 정의된 테이블이다. 뿐만 아니라 “그리고”, “~거나”와 같은 접속어와 이에 대응되는 연산자를 정의하고 있다. 기본적으로 정의된 연산자 외에 사용자가 정의하여 추가할 수 있다. 또한 체크리스트에서 많이 사용하는 단위도 정의되어 있고, 역시 사용자가 추가할 수 있다. Table 1은 미리 정의된 연산자의 예를 보여주고 있다. 구분은 연산자 설명을 위한 항으로 연산자를 설명하는 부분이며, 명령어, 비교어 및 접속어는 해당하는 단어를 연산자로 바꾼다.

한글 파서는 체크리스트를 읽어 들여 입출력 정보 테이블

Table 1. The Pre-Defined Operator Table Example

Type	Comparative Word / Conjunctive	Operator
Comparative Word	이상	>=
	크다	>
	많다	>
	빠르다	>
	올라가다	>
	적다	<
	이하	<=
	미만	<
	같다	=
Command	동일하다	=
	이다	=
Conjunctive	또한	and
	이고	and
	및	and
	혹은	or
	거나	or
Unit	초	
	분	
	시간	
	미터	
	rpm	
	km/h	
Fixed Value	ON	
	OFF	
	True	1
	False	0
	받는다	1
	준다	1
	제어한다	1
	동작한다	1

과 연산자 정의 테이블의 연산자를 이용하여 한글 테스트 항목을 변수와 연산자로 표현하는 모듈이다. 이 때 비교어의 변형된 값은 어원을 찾아서 어원에 해당하는 연산자로 바꾼다. 예를 들면, “크거나”는 한글 파서에서 어원인 “크다”로 인식하고 이에 해당하는 로직 연산자 “>”로 바꾼다. 그리고 숫자는 그대로 사용하며 숫자 다음의 단위 및 고정 값은 Table 1에 정의되어 있으면 그대로 사용한다. 특정 단위나 고정 값도 연산자에 값이나 표현을 정의하면 정의된 값으로 치환한다.

테스트 케이스 생성 모듈은 한글 파서가 생성한 표현을 사용하여 테스트 케이스를 생성한다. 이 때, 테스트케이스 생성 전략을 참조한다. 예를 들면, 테스트 케이스 생성 전략이 “결정 커버리지(Decision coverage)”[12]라면 체크리스트의 표현에서 결정이 참일 때와 거짓일 때가 수행되게 하는 테스트 케이스를 생성하게 된다. 이렇게 함으로써, 체크리스트에 나타나지는 않지만 변형된 조건에서 시스템이 정상 동작하는 지 테스트 할 수 있는 테스트 케이스를 체계적으로 생성할 수 있다.

테스트 출력은 테스트 케이스가 만들어지고 나면 해당 테스트 케이스가 정상 수행될 때 시스템이 가져야 할 출력 값을 생성해 준다.

“흡인기의 동작이 10분 이상이고 전원이 ON이면 모터가 500rpm 이상으로 작동한다.”라는 체크리스트 항목으로부터 테스트 케이스와 오라클을 생성하는 예를 통하여 위에서 설명한 방법으로 테스트 케이스와 예상 결과 값이 생성되는 과정은 다음과 같다.

입출력 정보 테이블은 Table 2에서 보는 바와 같이 흡인기와 전원을 입력으로 모터를 출력으로 정의하고 각각에 변수 명을 할당한다. 또한 각 변수가 가질 수 있는 범위도 정의한다. 범위가 정의되지 않는 입력은 임의 값을 사용할 수 있으며, 범위가 정의되지 않는 출력은 임의의 값이 출력될 수 있다.

Table 2. Input and Output Table Example

Type	Name	Symbol	Range
Input	흡인기	suctor	0 ~ 30
	전원	power	ON, OFF
Output	모터	motor	200 ~ 900

Table 2의 정보를 참조하여 한글 파서는 흡인기를 “suctor”로, 전원을 “power”로 모터를 “motor”로 치환한다. 또한 각 변수들이 가질 수 있는 범위도 기술한다. 그리고 Table 1을 참조하여 “10분 이상이고”에서 “이상”을 “>=”로 치환하고 “이고”를 “and”로, 이면를 “=”로 치환한다. 그리고 숫자 10, “분” 및 “ON”은 테이블이 정의되어 있으므로 그대로 사용한다. 치환된 값을 이용하여 한글로 정의된 테스트 항목을 테스트 케이스 생성 모듈이 사용 가능한 형태로 표현한다. 출력 표현은 다음과 같다.

“(suctor >= 10분) and (power = ON)”

3.2 테스트 케이스 생성

한글 파서에 의해 표현된 테스트 항목을 이용하여 테스트 케이스 생성 모듈이 테스트 케이스를 생성한다. 이 때 테스트 케이스 전략에 따라 생성되는 테스트 케이스가 달라진다. 테스트 케이스 값은 입출력 테이블에 정의된 범위 내에서 생성 전략에 맞는 값을 임의로 정한다. 현재 정의된 테스트 케이스 생성 전략은 “결정 커버리지”, “조건 커버리지

(Condition coverage)"[12], 및 "수정된 조건/결정 커버리지(MCDC: Modified Condition and Decision Coverage)"[12]이지만, 필요에 의해 추가할 수 있다. 3.1의 예시의 "(suctor >= 10분) and (power = ON)"로부터 추출 가능한 테스트 케이스 생성 예가 Table 3에 정리되어 있다. 결정 커버리지, 조건 커버리지 및 수정된 조건 결정 커버리지 전략을 만족하는 가능한 테스트 케이스들이다. 앞서 언급하였지만, 테스트 케이스에 사용되는 입력 값은 Table 2에 정의된 입력 범위 내에서 임의로 선택된다.

Table 3. Possible Generated Test Cases

Strategy	Test Case			Expected Output	
	No	Input Variable	Value	Output Variable	Value
Decision Coverage	T1	suctor	15분	motor	>=500rpm
		power	ON		
	T2	suctor	9분	motor	< 500rpm
		power	ON		
Condition Coverage	T3	suctor	23분	motor	>=500rpm
		power	ON		
	T4	suctor	5분	motor	< 500rpm
		power	OFF		
Modified Condition / Decision Coverage	T5	suctor	11분	motor	>=500rpm
		power	ON		
	T6	suctor	16분	motor	< 500rpm
		power	OFF		
	T7	suctor	30분	motor	< 500rpm
		power	ON		

4. 구현 및 실험

본 실험에서는 본문에서 제시하는 이론을 바탕으로 구현한 테스트 케이스 자동 생성 프로그램이 실제 사용하는 체크리스트로부터 테스트 케이스를 원하는 대로 생성하는 지를 검증하였다.

4.1 테스트 생성기 구현

제안된 방법은 C#으로 Windows 7 환경에서 구현하였으며, 한글 파서는 [18]에서 제공하는 것을 사용하였다. 앞에서 언급한 바와 같이 사용한 파서 엔진은 단어의 어원을 찾아준다. 현재 처리 가능한 문서 형식은 텍스트(.txt), 한글(.hwp), MS 엑셀(.xlsx) 그리고 MS 워드(.docx)이다. 체크리스트 내에서 하나의 테스트 항목은 사용자가 지정하는 구분자로 구분된다. 예를 들면, 번호 (1)로 구분될 수 있다. 파서는 각 테스트 항목에서 입출력 정보 및 연산자 정보를 찾아내고 이에 해당되는 변수명이나 연산자를 추출하고 이를 논리적 표현으로 변환하여 파일에 저장한다.

테스트 케이스 생성기는 변환된 파일의 논리 표현을 하나씩 분석하고 사용자가 선택하는 테스트 케이스 생성 전략에 따라 테스트 케이스를 생성하여, 텍스트(.txt), 한글(.hwp), MS 엑셀(.xlsx) 그리고 MS 워드(.docx) 등의 사용자가 지정하는 포맷의 파일로 저장한다.

Fig. 2. The Initial Window of Implemented Method

Fig. 2는 구현된 테스트 생성기의 초기화면이다. 사용자는 입출력 정보 테이블, 연산자 정보테이블, 출력 파일등을 지정할 수 있다.

4.2 테스트 케이스 생성

실험에서 사용한 체크리스트 파일은 의료기기용 내장 소프트웨어의 여러 기능 중, 특정 하나의 기능을 테스트하기 위한 체크리스트로 총 80개의 테스트 항목으로 구성되어 있다. 시스템 입력은 Foot switch 등 7개이고 출력은 LED, DC motor 등 6개이다.

Table 4는 작성된 입출력 정보 테이블의 일부이다. 출력 LED는 범위가 정해지지 않아서 임의 값이 출력될 수 있음을 의미한다. 실제 테스트 시에는 테스트 엔지니어가 눈으로 직접 확인하며 그 동작의 정확성 여부를 테스트한다. 연산자 정의 테이블은 Table 1을 사용한다.

현재 지원하는 테스트 케이스 생성 전략은 "결정 커버리지", "조건 커버리지", 그리고 "수정된 조건 결정 커버리지"이다. 다음은 각 전략에 따른 테스트 케이스 생성 예이다. 테스트 항목은 "키버튼이 동작하고 Foot switch가 ON이고

press key가 ON일 때"이다. 파싱된 표현은 "key_butt=ON and foot_sw=ON and key_press=ON"과 같다.

Table 4. A Part of the Input and Output Table

Type	Name	Symbol	Range
Input	키버튼	key_butt	ON, OFF
	Foot switch	foot_sw	ON, OFF
	Overflow 센서	over_sensor	0~10
	Press key	key_press	ON, OFF
	Motor 속도	speed_motor	VL, L, M, H, VH
	Motor 엔코더	encode_motor	0~5000
	과부화 감지	over_current	ON, OFF
Output	DC motor	dc_motor	ON, OFF
	press1 LED	press1_led	
	press 2 LED	press2_led	
	FND	fnd	
	press1 speaker	press1_spk	
	press2 speaker	press1_spk	

Table 5. The Generated Test Cases for the Test Item

Strategy	Test Case		
	No	Input Variable	Value
Decision Coverage	T1	key_butt	ON
		foot_sw	ON
		key_press	ON
	T2	key_butt	OFF
		foot_sw	OFF
		key_press	ON
Condition Coverage	T3	key_butt	ON
		foot_sw	ON
		key_press	ON
	T4	key_butt	OFF
		foot_sw	OFF
		key_press	OFF
Modified Condition / Decision Coverage	T5	key_butt	ON
		foot_sw	ON
		key_press	ON
	T6	key_butt	ON
		foot_sw	ON
		key_press	OFF
	T7	key_butt	ON
		foot_sw	OFF
		key_press	ON
	T8	key_butt	OFF
		foot_sw	ON
		key_press	ON

이 때 각 전략마다 생성된 테스트 케이스는 Table 5와 같다. "결정 커버리지"는 표현이 참 일 때와 거짓 일 때를 테스트 할 수 있는 경우를 만들며, "조건 커버리지"는 표현 내 각 입력 정보가 참 일 때와 거짓 일 때를 테스트한다.

그리고 "수정된 결정 조건 커버리지"에서는 결정, 조건 커버리지 이 외에 각 조건이 표현의 결정에 독립적으로 영향을 미칠 때를 테스트하는데 사용하는 테스트 케이스를 생성한다. 위의 경우를 만족하는 입력 조합은 유일하지 않기 때문에 항상 똑 같은 수의 테스트 케이스가 생성되지 않을 수 있다. 또한 테스트 케이스도 달라질 수 있다.

사용된 체크리스트에 나온 테스트 항목에서 모호하거나 입력 정보가 없는 테스트 항목은 입출력 정보로 수정하여 사용하였다. 예를 들면, "흡인기를 동작시키면"과 같은 표현은 "키버튼을 동작시키면"으로 변환하여 사용하였다. 흡인기를 동작시키기 위해서는 키버튼을 동작시켜야 하기 때문이다. 그리고 이 표현은 파서가 "key_butt = ON"으로 변환하고 테스트 케이스를 생성한다.

체크리스트를 사용하던 기존의 방법에서는 체크리스트에 나타난 테스트 항목 수인 80개의 테스트 케이스를 테스트 엔지니어가 사용할 것이다. 하지만 제안하는 테스트 케이스 방법을 사용하면 테스트 전략에 따라 Table 5와 같이 테스트 목적에 맞는 다양한 테스트 케이스를 생성하여 사용할 수 있을 것이다. 또한, 각 테스트 항목을 사용하였을 경우 출력 정보도 정확히 전달하여 테스트 엔지니어의 오류를 줄일 수 있을 것이다.

앞에서 언급한 바와 같이 각 전략에 따른 테스트 케이스 및 갯수는 유일하지 않기 때문에 Table 6에 나타난 테스트 수는 각 전략에 따른 최소 테스트 케이스 수가 아닐 수 있다.

생성된 테스트 케이스가 해당되는 커버리지를 만족하면서 주어진 체크리스트를 정확히 만족하는 지는 모든 테스트 케이스를 모두 비교하여, 정확히 생성되는 지를 확인하였다. 확인 과정에서 입출력을 기준으로 오류없이 정확히 생성하였음을 확인하였다. 또한 결정, 조건 및 수정된 결정 조건 커버리지 기준으로 생성된 테스트 케이스는 체크리스트에 없는 입력이

Table 6. Number of Test Cases Generated by the Different Generation Strategies

Strategy	Number of Test Cases
Decision Coverage	160
Condition Coverage	262
Modified Condition / Decision Coverage	389

가해지는 경우에 시스템이 정상 동작하는 지를 검사할 수 있는 항목들을 정확히 포함하고 있었다. 이는 기존 체크리스트 기반 테스트 케이스 생성의 문제점인 수동 생성에 의한 오류 제거 및 체크리스트 항목 이외의 경우 테스트를 위한 항목 생성이란 목표를 잘 만족시키고 있음을 보여준다.

5. 결 론

본 논문에서는 한글로 작성된 체크리스트에서 테스트 케이스를 자동으로 생성하는 방법을 제시하였다. 제안된 방법은 임베디드 시스템 테스트 현장에서 일반적으로 사용하는 체크리스트 기반 테스트의 단점을 보완할 수 있다.

제안하는 방법에서는 한글로 작성된 체크리스트 내의 사용자 혹은 미리 정의된 입출력 정보 테이블과 연산자 정보 테이블을 이용하여 테스트 항목들을 파싱하여 논리 표현으로 바꾸고, 사용자가 정하는 테스트 케이스 생성 전략에 맞는 테스트 케이스를 논리 표현으로부터 도출한다.

논문에서 제안한 방법을 구현하였고, 실험을 통해 테스트 케이스를 자동으로 생성할 수 있음을 확인하였다. 생성된 케이스는 의료기기용 임베디드 시스템에서 사용하는 체크리스트로는 확인할 수 없는 다양한 케이스를 생성함을 확인할 수 있었다. 제안한 방법에서 체크리스트를 파싱하여 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 오류 발생 가능성이 있는 동작까지도 테스트할 수 있다. 이는 수작업으로 하기 힘든 체계적이고 다양한 조합의 테스트 케이스 생성를 생성 할 수 있는 장점을 의미한다. 본 연구에서 수행한 결과로 제안하는 방법의 효용성을 일반화할 수는 없지만, 그 가능성은 충분히 보여주고 있다.

제안한 방법을 보완하기 위해서는 연산자 테이블을 보완해야하고, 좀 더 다양한 표현을 파싱 할 수 있는 방안에 대한 지속적인 연구가 필요하다. 또한, 본 논문에서 제시한 테스트 케이스에서 실제 시스템을 테스트 할 수 있는 테스트 스크립트를 자동으로 생성할 수 있는 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*, p.9, Aug., 2007.

[2] 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.

[3] 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.

[4] 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., 9-13, 2002.

[5] M. Conrad, H. D¨orr, 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] P. S. Loo, and W. K. Tsai, “Random Testing Revisited,” *Information and Software Technology*, Vol.30, Issue 7, pp.402-417, Sep., 1988.

[7] 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*, 2008, QSCI'08, pp.145-154, Aug., 2008.

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

[9] P. M. Kruse and M. Luniak, “Automated test case generation using classification trees,” *Software Quality Professional Magazine*, Vol.13, No.1, pp.4-12, 2011.

[10] H. S. Park, “Generating Structural Test Cases for MATLAB Stateflow Model Using Rapidly-exploring Random Tree,” Ph.D. dissertation, Ajou University, 2014.

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

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

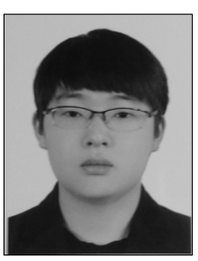
[13] Joseph Schmuller, “Teach yourself UML in 24 Hours, 3/E,” SAMS, 2004.

[14] U. S. NRC, “Fault Tree Handbook (NUREG-0492),” US, 1981.

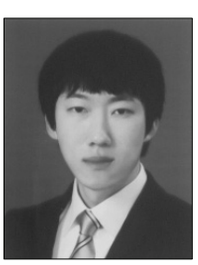
[15] Yu Lei and K. C. Tai, “In-Parameter-Order: A Test Generation Strategy for Pairwise Testing,” In *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium*, pp.254-261, 1998.

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

[17] 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, 2016.
[18] GitHub, Inc. [Internet], https://github.com/modamoda/TwitterKoreanProcessorCS.
[19] S. Zhang et al., “Combined Static and Dynamic Automated Test Generation,” *Proceedings of ISSTA'11*, Toronto, Canada, pp.353-363, Jul., 2011.
[20] S. Ananad et. al., “An orchestrated survey of methodologies for automated software test case generation,” *Journal of Systems and Software,* Vol.86, No.8, pp.1978-2001, 2013.
[21] O. Olajubu et. al., “Automated test case generation from domain specific models of high-level requirements,” *Proceedings of RACS'15,* ACM, pp.505-508, Oct., 2015.
[22] Dae Joon Kim, Ki Hyun Chung, and Kyung Hee Choi, “A Hierarchical Checklist to Automatically Generate Test Scripts,” *KIPS Tr. Software and Data Eng.*, Vol.6, No.5, pp.245-257, 2017.



김 현 동
e-mail : khd8009@ajou.ac.kr
2016년 아주대학교 전자공학과(학사)
2017년~현 재 아주대학교 전자공학과 석사과정
관심분야 : 임베디드 시스템 테스팅, 임베디드 시스템 설계, 임베디드 소프트웨어



김 대 준
e-mail : kdj4527@ajou.ac.kr
2016년 아주대학교 컴퓨터공학과 석사
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 : 임베디드 시스템 테스팅, 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

박 호 준
e-mail : hjpark97@ktl.re.kr
1995년 인하대학교 전기공학과(학사)
1997년 인하대학교 전기공학과(석사)
2013년 아주대학교 산업공학과(박사)
1997년~2000년 ㈜SKC 중앙연구소 주임연구원
2000년~2008년 한국산업기술시험원 의료기기본부 책임연구원
2008년~2010년 복지부 첨단의료복합단지 조성사업단 의료기기전문의원
2010년~2011년 한국산업기술시험원 의료기기기술팀장/책임연구원
2011년~2015년 한국산업기술시험원 의료기기평가센터장/수석연구원
2015년~현 재 한국산업기술시험원 의료기기연구센터장/수석연구원
관심분야 : 의료기기 시험평가, 소프트웨어 밸리데이션

이 용 윤

e-mail : yylee@ktl.re.kr

2002년 서울시립대학교 전자전기공학부 (학사)

2007년 서울시립대학교 전자전기컴퓨터 공학부(석사)

2013년 서울시립대학교 전자전기컴퓨터 공학부(박사)

2002년~2012년 서울시립대학교 양자정보처리연구단 연구조원

2012년~2014년 국내 IEC TC 62 COSD 활동

2012년~현 재 한국산업기술시험원 의료헬스본부 팀원

2015년~현 재 AHWP WG3 (사전허가-SW) 팀원

2016년~현 재 국내 ISO TC 210 전문위원

관심분야 : 의료기기 시험평가, 소프트웨어 밸리데이션