JANT 한국항행학회논문지
Journal of Advanced Navigation Technology

응용 및 융합 기술

J. Adv. Navig. Technol. 27(2): 228-234, Apr. 2023

메타모델 기반 요구사항 명세 기법인 의사 결정표를 통한 자동 테스트 케이스 생성 메커니즘

Automatic Test case Generation Mechanism from the Decision Table of Requirement Specification Techniques based on Metamodel

손 현 승
국립목포대학교 컴퓨터공학과

Hyun Seung Son
Department of Computer Engineering, Mokpo National University, Jeollanamdo 58554, Korea

[요 약]

고품질 소프트웨어의 요구 증가로 국제표준, 산업 기능안전(IEC 61508), 자동차(ISO 26262), 무기체계 내장형 소프트웨어 지침 등 품질 인증 요구가 많다. 스타트 업, 벤처, 중소 업체들은 비용 및 인력 측면에서 체계적인 품질 획득이 어려움이 있다. 그들 업체에게 자동 테스트 케이스 생성은 비용, 시간, 인력 문제에도 소프트웨어 품질을 향상할 수 있는 해법으로 제시 될 수 있다. 이를 위해, 시스템 및 소프트웨어 설계 검증이 가능한 "의사 결정표" 기반 테스트 케이스 자동화를 제안한다. 이는 OMG의 표준 기법인 메타모델과 모델 변환 기법을 사용해 각각 의사 결정표(Model)와 테스트 케이스(Text)의 메타모델 설계 및 모델변환을 정의한다. 즉 의사 결정표 입력으로 테스트 케이스 발생 자동화이다. 이를 통해 MC/DC 커버리지등도 쉽게 적용 가능하다.

[Abstract]

As the increasing demand for high-quality software, there is huge requiring for quality certification of international standards, industrial functional safety (IEC 61508), automotive (ISO 26262), embedded software guidelines for weapon systems, etc., in the industry. Software companies are very difficult to systematically acquire the quality certification in terms of cost and manpower of Startup, venture small-sized companies. For their companies one test case automatic generation is considered as a core technique to evaluate or improve software quality. This paper proposes a test case automatic generation method based on the design decision table for system and software design verification. We apply the proposed method with OMG's standard techniques of metamodel and model transformation for automatically generating test cases. To do this, we design the metamodels of design decision table (Model) and test case document (Text) and define model transformation to automatically generate test cases, which will expect to easily work MC/DC coverage.

Key word : Metamodel, Model Transformation, Software Quality, Test case, Testing.

http://dx.doi.org/10.12673/jant.2023.27.2.228

Received 22 March 2023; **Revised** 31 March 2023
Accepted (Publication) 24 April 2023 (30 April 2023)

***Corresponding Author: Hyun Seung Son**

Tel: +82-061-450-2441
E-mail: hson@mnu.ac.kr

Ⅰ. 서 론

고품질 소프트웨어에 대한 요구의 증가로 국제표준, 산업 기능안전(IEC 61508)[1], 자동차(ISO 26262)[2], 무기체계 내장형 소프트웨어 지침[3] 등 품질 인증에 대한 요구는 많아지고 있다. 그러나 국내 벤처/중소 소프트웨어 기업들은 비용 및 인력 측면에서 영세하기 때문에, 이러한 소프트웨어 품질 인증에 획득에 많은 어려움을 겪고 있다.

일반적인 테스트 케이스는 테스터에 의해서 수작업으로 진행되지만 테스터의 성숙도에 따라서 불규칙한 형태의 테스트 케이스가 생성되고 일부는 누락될 수 있다. 그래서 테스트 케이스 자동 생성 방법은 소프트웨어 품질을 평가 또는 인증받거나 소프트웨어 품질을 향상시킬 수 있는 핵심 기법으로서 간주되고 있고 이에 대한 요구는 계속 증가하고 있다.

본 논문은 시스템 및 소프트웨어 설계 검증을 위해 의사 결정표(Decision Table) 기반으로 테스트 케이스를 모델변환[4]으로 자동 발생하는 것이다. 이 방법은 효율적으로 테스트 케이스를 생성하고, 다른 도구와 상호운영 될 수 있도록 아래와 같이 수행한다. OMG의 메타모델[5]을 통한 의사 결정표를 만들고 모델변환을 사용하여 테스트 케이스 자동 생성한다.

메타모델 기반 의사 결정표는 테스트 케이스로 변형 하기 위해서는 메타모델 기반으로 테스트 케이스 모델 설계가 필요하다. 또한, 모델변환 기반 테스트 케이스 자동 생성에서는 MC/DC(Modified Condition and Decision Coverage) 커버리지[6]를 기반으로 테스트 케이스를 자동 생성할 수 있도록 모델변환 기법을 사용해 구조 설계 및 도구를 구현한다.

적용사례를 통해 수작업으로 테스트 케이스를 생성하기 위해서는 현실적으로 비용이나 인력 면에서 제약이 크다. 또한 인증을 받기 위해서는 평가 기간, 컨설팅 및 심사비용이 많이 든다. 테스트 케이스 자동생성 도구는 이러한 점에서 테스트 케이스 생성 비용을 줄일 것으로 기대한다.

Ⅱ. 관련연구

2-1 심볼릭 기반 테스트 케이스 생성

화이트 박스 기반의 테스팅 접근법은 프로그램의 소스 또는 바이너리 코드로부터 테스트 데이터를 생성한다. 화이트 박스 접근법 중 하나인 심볼릭 기반 테스트 케이스 생성 기법은 프로그램의 코드를 분석하여 프로그램의 테스트 데이터를 자동으로 생성하는 프로그램 분석기술이다.

심볼릭 실행(Symbolic Execution)[7]은 구체적인 값 대신에 심볼릭 값을 프로그램 입력으로 사용하고, 프로그램 변수의 값을 그 입력의 심볼릭 표현으로 나타낸다. 또한 동적 심볼릭 수행을 통해 더 높은 커버리지를 달성하는 테스트 케이스 자동 생성기법에는 오픈소스 도구인 CREST[8]와 KLEE[9]이 있다.

2-2 모델 기반 테스트 케이스 생성

모델 기반 테스팅(MBT, Model-based Testing)은 소프트웨어 요구사항을 표현한 모델로부터 생성된 테스트 케이스를 테스트 대상 프로그램에 수행 함으로써 테스트 대상 프로그램이 모델에 명세된 대로 작동하는지 여부를 확인하는 기법이다.

테스트 케이스의 기대 결과 값을 갖고 있기 때문에 기존의 자동화 테스팅 기법들이 생성할 수 없었던 구체적인 기대 결과 값을 자동으로 생성할 수 있는 장점이 있다. 메타모델 기반의 테스트 연구들에는 M2M(Model to Model) 모델변환 기반의 UML 스테이트 다이어그램을 통한 테스트 케이스 자동 추출 메커니즘에 관한 연구[10], 메타모델링과 모델변환을 적용한 메시지 시퀀스 다이어그램으로 테스트 케이스 생성에 관한 연구 등이 있다[11].

2-3 검색 기반 테스트 케이스 생성

검색 기반 테스트 케이스를 생성하는 방법은 효율적인 검색 알고리즘을 사용하는 방법이다. 가장 일반적으로 적용되는 글로벌 검색 알고리즘 중 하나인 유전 알고리즘(Genetic Algorithm)이다.

검색 기반은 객체 지향 코드에 대한 단위 테스트 스위트를 자동으로 생성할 수 있고 높은 코드 커버리지를 달성하는 테스트 스위트를 효율적으로 생성할 수 있지만, 코드의 어려운 부분을 다루는 데 필요한 특정 값을 생성하는데 어려움이 있을 수 있다. 또한, 검색기반은 휴리스틱을 기반으로 하며 검색자가 해당 문제를 선호하지 않을 경우 비효율적일 수 있다. 동적 심볼릭 수행으로 검색기반을 발전시킨 연구 사례가 있다[12].

Ⅲ. 의사 결정표기반 테스트 케이스 생성방법

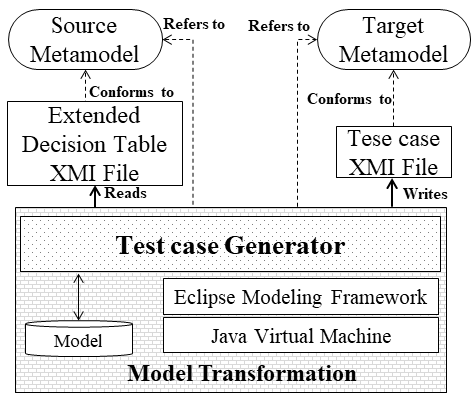


그림 1. 테스트 케이스 생성을 위한 모델 변환 절차
Fig. 1. The process of model transformation for test case generation

의사 결정표에서 테스트 사례를 자동으로 생성하기 위해 모

델 변환을 사용한다. 제안한 테스트 케이스 생성 방법은 그림 1과 같이 입력과 출력 메타모델 설계와 모델 변환 규칙의 세 단계로 구성된다. 메타모델 설계에서는 의사 결정표와 테스트 케이스를 통해 각각 메타 모델을 설계한다. 모델 변환을 위한 규칙은 EMF(Eclipse Modeling Framework)를 기반으로 Java를 이용하여 프로그래밍한다[13].

3-1 의사 결정표의 메타모델 설계

의사 결정표는 요구사항의 논리와 발생 조건에 따라서 생성될 수 있는 결과를 테이블의 형태로 나열한 것으로, 조건과 행동의 모든 가능한 조합을 고려하여 테스트 케이스를 생성하는 기법으로 표 1과 같다. 이 의사 결정표를 기반으로 MC/DC 커버리지를 만족하기 위해서는 개발되는 모든 조건문이 결정 테이블에 반영되어야 한다.

표 1. 의사 결정표의 예
Table 1. The example of decision table

Condition	c1: a<b+c	F	T	T	T	T	T	T	T	T	T
	c2: b<a+c		F	T	T	T	T	T	T	T	T
	c3: c<a+b			F	T	T	T	T	T	T	T
	c4: a = b				T	T	T	T	F	F	F
	c5: a = c				T	T	F	F	T	T	F
	c6: b = c				T	F	T	F	T	F	T
Action	a1: Not a Triagle	X	X	X							
	a2: Scalene										X
	a3: Isosceles							X		X	X	
	a4: Equilateral				X						
	a5: Impossible					X	X		X		

의사 결정표의 메타모델은 그림 2와 같다. 의사 결정표는 조건과 행위는 표의 수직 축으로 나열되고 테스트 케이스는 표의 수평 축에 배치된다.

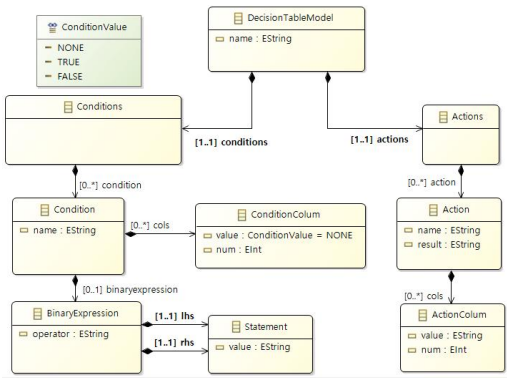
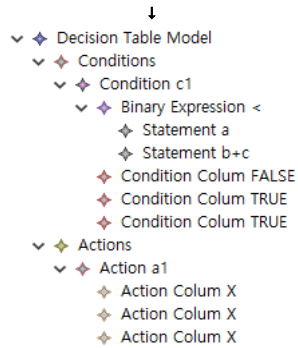


그림 2. 의사 결정표의 메타모델 설계
Fig. 2. The metamodel design of decision table

의사 결정표에 대한 표준화된 메타모델이 정의되어 있지 않으므로 표 1의 표현을 이용하여 메타모델을 설계한다. 설계된 메타모델의 "Decision Table Model"은 모든 요소를 포함하는 의사결정 테이블의 루트 노드이다. "Decision Table Model"에는 여러 개의 "Decision Table Row" 집합이 있으며, 이는 "Condition"과 "Action"의 데이터로 관련 조건과 행위만 포함한다. "Binary Expression"은 조건 수식을 입력 받는다.

예를 들어 1개의 조건 값이 1개의 행위와 연관된다면 데이터는 그림 3과 같이 표현된다. 메타모델의 데이터는 XMI(XML Metadata Interchange)[14] 파일 형태로 저장된다.

c1: a<b+c	F	T	T
a1: Not a Triagle	X	X	X




```
<decisionTableModel:DecisionTableModel>
 <conditions>
  <condition name="c1">
   <binaryexpression operator="&lt;">
    <lhs value="a"/>
    <rhs value="b+c"/>
   </binaryexpression>
   <cols value="FALSE" num="1"/>
   <cols value="TRUE" num="2"/>
   <cols value="TRUE" num="3"/>
  </condition>
 </conditions>
 <actions>
  <action name="a1" result="Not a Triagle">
   <cols value="X" num="1"/>
   <cols value="X" num="2"/>
   <cols value="X" num="3"/>
  </action>
 </actions>
</decisionTableModel:DecisionTableModel>
```

그림 3. 의사 결정표 모델의 입력 결과
Fig. 3. The input result of decision table model

3-2 테스트 케이스의 메타모델 설계

테스트 케이스는 입력되는 조건과 기대 결과로 정의된 문서이다. IEEE 표준 829는 테스트 케이스를 "테스트 항목에 대한 입력, 예측 결과 및 실행 조건 세트를 지정하는 문서"로 정의되어 있다[15]. 표 2는 테스트 케이스의 예이다. 테스트 케이스의 데이터가 테이블 형태로 변환되고 입력(Input), 조건(Condition), 기대 결과(Expected Result)로 구성된다. 입력은 입력값들이고 조건은 입력되는 조건들의 조합의 논리식이다. 기대 결과는 입력과 조건의 값에 따른 기대되는 값이다.

표 2. 테스트 케이스의 예
Table 2. The example of test case

ID	Input	Condition	Expected Result
TC-1			
TC-2			

테스트 케이스의 메타모델은 테이블의 기본적인 이름을 이용하여 그림 4와 같이 설계 가능하다. 의사 결정표와 마찬가지로 테스트 케이스에 대한 표준화된 메타모델이 정의되지 않아 표의 기본 이름을 사용하여 메타모델을 설계한다. 설계된 메타모델에서 “Test case Model”은 “Test case”의 루트 노드이다. “Test case Model”는 여러 개의 “Test case”가 있다. “Test case”는 입력(Input), 조건(Condition) 및 예상 결과(Expected Result)로 구성된다. 데이터는 각각 “value” 속성에 문자열로 저장된다.

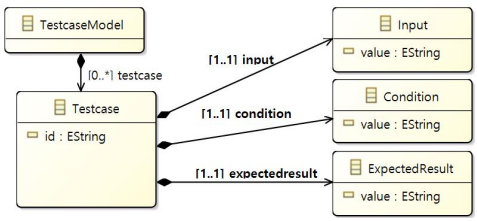
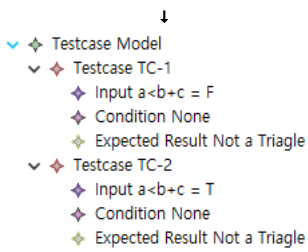


그림 4. 테스트 케이스의 메타모델 설계
Fig 4. The metamodel design of test case

예를 들어 2개의 테스트 케이스가 있을 때 그림 5와 같이 표현된다.

ID	Input	Condition	Expected Result
TC-1	a<b+c = F	None	Not a Triagle
TC-2	a<b+c = T	None	Not a Triagle




```
<testcaseModel:TestcaseModel>
  <testcase id="TC-1">
    <input value="a&lt;b+c = F"/>
    <condition value="None"/>
    <expectedresult value="Not a Triagle"/>
  </testcase>
  <testcase id="TC-2">
    <input value="a&lt;b+c = T"/>
    <condition value="None"/>
    <expectedresult value="Not a Triagle"/>
  </testcase>
</testcaseModel:TestcaseModel>
```

그림 5. 테스트 케이스 모델의 입력 결과
Fig. 5. The input result of test case model

3-3 테스트 케이스의 변환 규칙

제안하는 테스트 케이스 자동 생성 방법은 메타모델을 기반으로 모델에서 모델로 변환한다. 모델 변환 규칙은 EMF[13]를 이용하여 Java 프로그램으로 작성한다. 테스트 케이스 생성은 그림 6과 같이 테이블의 열의 크기 동안에 반복하면서 행 1칸씩 이동하면서 테스트 케이스를 생성한다. 이때 조건 행의 개수와 수식을 사용하여 MC/DC 커버리지 진리표를 생성한다.

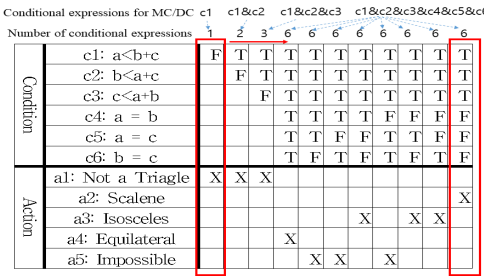


그림 6. 의사 결정표와 조건문의 관계
Fig. 6. The relationship between decision table and condition expression

의사 결정표의 각 조건식은 “and”에 해당하므로 이를 통해서 MC/DC 커버리지에 해당하는 진리표를 자동으로 만들 수 있다. 진리표를 자동으로 생성하기 위해서는 다음과 같이 3단계로 처리한다. 1) 해당 조건식에 가능한 모든 조합을 만들고, 2) 모든 조합에서 각 조건식의 값을 변경하여 결과에 영향을 주는 값만 수집하고, 3) 수집된 결과를 리스트로 만든다. 그림 7은 조건식의 개수에 따라 진리표를 자동으로 생성하는 코드이다.

```
public ArrayList<int[]> getTruthTable(int numTerms) {
    int numCombinations = (int) Math.pow(2, numTerms);
    ArrayList<int[]> outlist = new ArrayList<int[]>();
    int[][] terms = new int[numTerms][numCombinations];
    for (int i = 0; i < numTerms; i++) {
        int numRepeats = (int) Math.pow(2, numTerms-i-1);
        int value = 0;
        for (int j = 0; j < numCombinations; j++) {
            terms[i][j] = value;
            if ((j+1) % numRepeats == 0)   value = value == 0 ? 1 : 0;
        }
    }
    for (int i = 0; i < numCombinations; i++) {
        boolean result = true, out = false;
        for(int j=0;j<numTerms;j++)  result = result && (terms[j][i]==1);
        for (int k = 0; k < numTerms; k++) {
            boolean mcdc_test = true;
            for (int j = 0; j < numTerms; j++) {
                if(k == j)   mcdc_test = mcdc_test && (terms[j][i] != 1);
                else    mcdc_test = mcdc_test && (terms[j][i] == 1);
            }
        if(result != mcdc_test)    out = true;
    }
    if(out == true) {
        int[] outline = new int[numTerms+1];
        for (int j = 0; j < numTerms; j++)   outline[j] = terms[j][i];
            outline[numTerms] = (result ? 1 : 0);  outlist.add(outline);
        }
    }
    return outlist;
}
```

그림 7. 조건식에 따른 진리표를 자동 생성 알고리즘
Fig. 7. Automatic generation algorithm for truth tables based on conditional expressions

조건식에 따른 진리표 자동 생성 알고리즘을 통해 최종 테스트 케이스 생성 방법은 다음과 같다.

1. 의사 결정표에서 최대 행의 개수를 카운트한다.
2. 최대 행의 개수까지 반복한다.
2.1 행에 입력된 조건식만 리스트로 만든다.
2.2 조건식의 개수로 MC/DC 진리표를 생성한다.
2.3 조건식과 진리표를 매칭하여 테스트 케이스를 만든다.

표 1의 의사 결정표를 제안한 알고리즘으로 프로그램을 실행하면 그림 8과 같이 총 65개의 테스트 케이스를 생성한다.

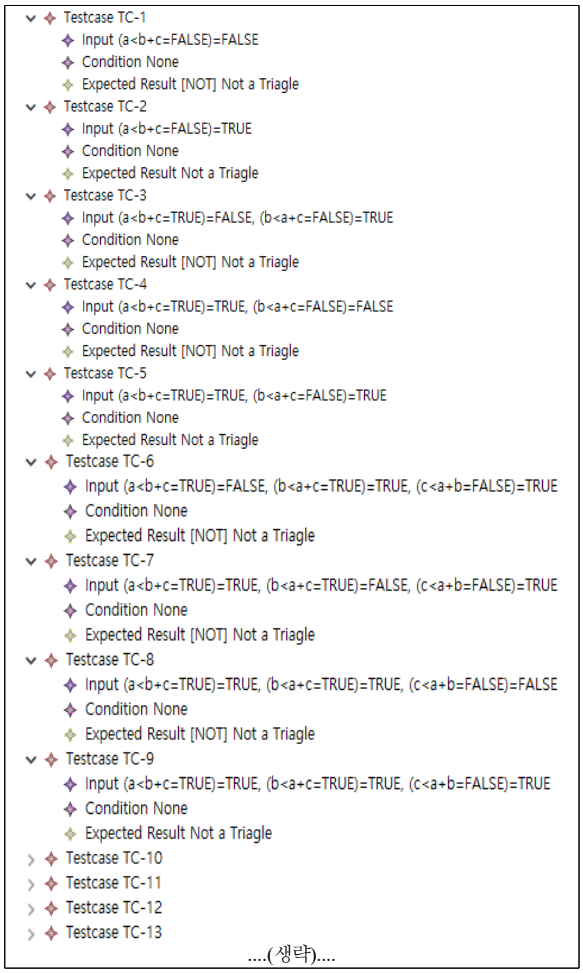


그림 8. 자동 생성된 테스트 케이스 결과
Fig. 8. Test results of automatic generated test cases

Ⅳ. 적용사례

미사일 시스템은 작전 환경(항공전투)에서 미사일 무기를 사용하여 선택한 표적을 탐지하고 격추하는 능력을 개발하고 생산하는 데 필요한 하드웨어, 소프트웨어, 데이터, 서비스 및 시설의 복합체다[16]. 본 논문에서는 항공 방어 시스템에서 사용하는 미사일 발사 요구사항 일부를 적용사례로 하였다.

표 3. DPS 미사일 시스템 요구사항 예제
Table 3. The requirement example of DPS missile system

요구 사항	동력비행신호가 On으로 되어있고 해안 비행신호가 On일 때, 미사일 발사체 각도를 계산할 수 있다.

표 3은 미사일 시스템 요구사항 예제를 나타낸다. 자연어 요구사항에서 조건 요소들을 분리해 의사 결정표를 통해 발생 가능한 모든 경우의 수를 측정하여, 의사 결정표에서 테스트 케이스를 생성한다. 의사 결정표를 만들 때는 발생할 수 있는 모든 조건(Condition)에 대한 행위(Action)의 수행 유무를 포함하여 발생 가능한 모든 행위를 추출하여 만든다.

그림 9는 Cx 는 미사일시스템 체계에서 Cone(미사일 헤드)의 축력 계수에 대한 단순화된 이론적 모델을 나타낸다. 축 방향 공기역학적 힘 계수 Cx의 계산을 위해서 계산을 수행할 수 있는 모든 경우의 수를 나타내기 위해 의사 결정표를 사용해 나타낸다.

$$C_x = \begin{cases} 2\sin^2\theta_c + C_{x2}\alpha^2, & M \le 0.5, \\ 2\sin^2\theta_c\{1.0 + [((k_1 + k_2\sin\theta_c)/(k_3 + k_4\sin\theta_c)) - 1.0 \\ \quad + (k_5\kappa/2\sin^2\theta_c)(M - 0.5)\} + C_{x2}\alpha^2, & 0.5 \le M \le 0.5, \\ 2\sin^2\theta_c[(k_6 + \sqrt{M^2-1}\sin\theta_c)/(k_7 + \sqrt{M^2-1}\sin\theta_c)] \\ \quad + \kappa/M^2 + C_{x2}\alpha^2, & M \ge 1.5, \end{cases}$$

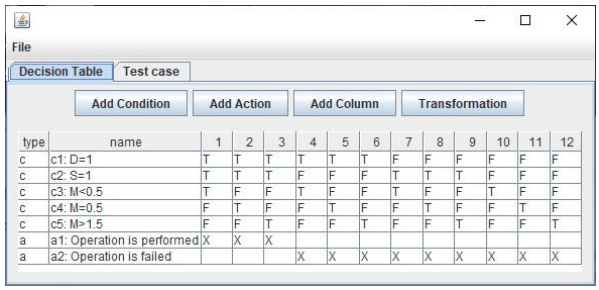
그림 9. 조건 생성을 위한 축 방향 공기역학 힘 계수 수식
Fig. 9. Axial aerodynamic force factor formula for conditions generation

표 4는 미사일 시스템의 Cone 축력계수 계산수행을 위한 의사 결정표를 나타낸다. D는 동력 비행 신호로서, D = 1일 경우, 동력 비행 신호 스위치가 On 상태임을 나타낸다. S는 해안 비행 신호로서, S = 1일 경우, 해안 비행 신호 스위치가 On 상태임을 나타낸다. M 값은 Operation에서 c3, c4, c5 중에 하나만 선택될 수 있으므로 이를 반영하여 의사 결정표를 작성한다.

표 4. 미사일 시스템의 Cone 축력계수 계산수행 의사 결정표
Table 4. The decision table for performing Cone axial force factor calculations for missile systems

Condition	c1: D=1	T	T	T	T	T	T	F	F	F	F	F	F
	c2: S=1	T	T	T	F	F	F	T	T	T	F	F	F
	c3: M<0.5	T	F	F	T	F	F	T	F	F	T	F	F
	c4: M=0.5	F	T	F	F	T	F	F	T	F	F	T	F
	c5: M>1.5	F	F	T	F	F	T	F	F	T	F	F	T
Action	a1: Operation is performed	X	X	X									
	a2: Operation is failed				X	X	X	X	X	X	X	X	X

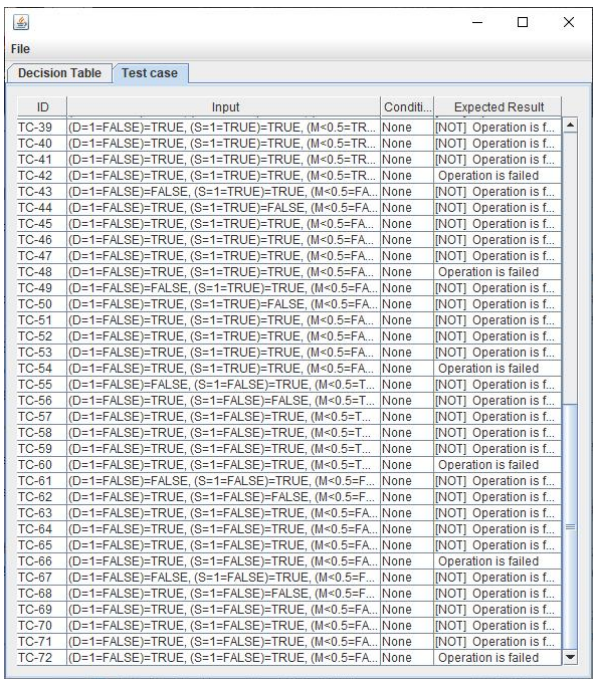
그림 10은 테스트 케이스를 생성하기 위해 의사 결정표를 도구에 입력하는 것을 나타낸다. 테스트 케이스를 생성을 위해 의사 결정표의 조건(Condition)과 행위(Action)를 새로 생성하고 칼럼(Column)을 추가하여 의사 결정표와 같이 입력한다.



type	name	1	2	3	4	5	6	7	8	9	10	11	12
c	c1: D=1	T	T	T	T	T	T	F	F	F	F	F	F
c	c2: S=1	T	T	T	F	F	F	T	T	T	F	F	F
c	c3: M<0.5	T	F	F	T	F	F	T	F	F	T	F	F
c	c4: M=0.5	F	T	F	F	T	F	F	T	F	F	T	F
c	c5: M>1.5	F	F	T	F	F	T	F	F	T	F	F	T
a	a1: Operation is performed	X	X	X									
a	a2: Operation is failed				X	X	X	X	X	X	X	X	X

그림 10. 테스트 케이스 생성을 위한 의사 결정표 도구 입력화면
Fig. 10. Decision table input screen for test case generation

"Transformation" 버튼을 누르면 의사 결정표는 그림 11처럼 MC/DC 커버리지 기반으로 72개의 테스트 케이스로 생성된다. 텍스트 형태로 입력된 의사 결정표는 도구에 의해서 자동으로 입력 메타 모델로 변환되고 모델 변환 알고리즘에 의해서 테스트 케이스 XMI파일을 생성하고 화면에 값을 표시한다. 의사 결정표의 조건 c1~c5의 값은 MC/DC 커버리지를 기준으로 진리표 자동 생성 알고리즘에 의해 "011110", "101110", "110110", "111010", "111100", "111111"의 진리표 생성한다. 여기서 1은 True, 0은 False이다. 이 진리표를 활용하여 각 조건 값이 진리표가 될 수 있도록 테스트 케이스를 만든다. 예를 들어 "011110"은 (D=1=TRUE)=FALSE, (S=1=TRUE)=TRUE, (M<0.5=TRUE)=TRUE, (M=0.5=FALSE)=TRUE, (M>1.5=FALSE)=TRUE으로 만든다.



ID	Input	Conditi...	Expected Result
TC-39	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=TR...	None	[NOT] Operation is f...
TC-40	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=TR...	None	[NOT] Operation is f...
TC-41	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=TR...	None	[NOT] Operation is f...
TC-42	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=TR...	None	Operation is failed
TC-43	(D=1=FALSE)=FALSE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-44	(D=1=FALSE)=TRUE, (S=1=TRUE)=FALSE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-45	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-46	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-47	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-48	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	Operation is failed
TC-49	(D=1=FALSE)=FALSE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-50	(D=1=FALSE)=TRUE, (S=1=TRUE)=FALSE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-51	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-52	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-53	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-54	(D=1=FALSE)=TRUE, (S=1=TRUE)=TRUE, (M<0.5=FA...	None	Operation is failed
TC-55	(D=1=FALSE)=FALSE, (S=1=FALSE)=TRUE, (M<0.5=T...	None	[NOT] Operation is f...
TC-56	(D=1=FALSE)=TRUE, (S=1=FALSE)=FALSE, (M<0.5=T...	None	[NOT] Operation is f...
TC-57	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=T...	None	[NOT] Operation is f...
TC-58	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=T...	None	[NOT] Operation is f...
TC-59	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=T...	None	[NOT] Operation is f...
TC-60	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=T...	None	Operation is failed
TC-61	(D=1=FALSE)=FALSE, (S=1=FALSE)=TRUE, (M<0.5=F...	None	[NOT] Operation is f...
TC-62	(D=1=FALSE)=TRUE, (S=1=FALSE)=FALSE, (M<0.5=F...	None	[NOT] Operation is f...
TC-63	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-64	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-65	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-66	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=FA...	None	Operation is failed
TC-67	(D=1=FALSE)=FALSE, (S=1=FALSE)=TRUE, (M<0.5=F...	None	[NOT] Operation is f...
TC-68	(D=1=FALSE)=TRUE, (S=1=FALSE)=FALSE, (M<0.5=F...	None	[NOT] Operation is f...
TC-69	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-70	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-71	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=FA...	None	[NOT] Operation is f...
TC-72	(D=1=FALSE)=TRUE, (S=1=FALSE)=TRUE, (M<0.5=FA...	None	Operation is failed

그림 11. 의사 결정표 입력으로 자동 생성된 테스트 케이스 결과
Fig. 11. Automated test case generation results from the decision table

(D=1=TRUE)=FALSE의 의미는 "D=1"의 값이 True일 때 테스트 케이스는 조건 값이 False가 될 수 있도록 테스트를 수행해야 한다는 것이다. 그러므로, 생성되는 테스트 케이스 수는 조건의 숫자와 규칙의 수에 비례하여 생성된다.

Ⅴ. 결 론

본 논문은 시스템 및 소프트웨어 설계 검증을 위해 의사 결정표 기반으로 테스트 케이스를 모델변환 기법을 적용해 자동으로 발생하는 방법에 대해서 제안한다. 제안한 방법은 효율적으로 테스트 케이스를 생성하고, 다른 도구와 상호운영 될 수 있도록 의사 결정표와 테스트 케이스를 메타모델과 EMF를 사용하여 모델변환 규칙을 만들어 테스트 케이스 자동 생성하였다. 또한 적용사례를 통해 모델변환 기반 테스트 케이스 자동 생성에서는 입출력 변수와 값과 MC/DC 커버리지로 테스트 케이스를 자동 생성할 수 있는 것을 보여주었다.

적용사례를 통해 수작업으로 테스트 케이스를 생성하기 위해서는 현실적으로 비용이나 인력 면에서 제약이 크다. 또한 인증을 받기 위해서는 평가 기간, 컨설팅 및 심사비용이 많이 든다. 테스트 케이스 자동 생성 도구는 이러한 점에서 테스트 케이스 생성 비용을 줄일 것으로 기대한다.

향후 연구로 기존의 MC/DC 커버리지 뿐만아니라 분기, 경계 값 기준 등의 다양한 기법을 적용할 수 있도록 확장하고자 한다.

Acknowledgments

본 논문(저서)은 2021학년도 목포대학교 교내연구비 지원에 의하여 연구되었음

References

[1] International Electrotechnical Commission. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems [Internet]. Available: https://webstore.iec.ch/publication/2469

[2] International Organization for Standardization. ISO 26262: Road vehicles − Functional safety [Internet]. Available: https://www.iso.org/standard/68383.html

[3] Defense Acquisition Program Administration. Practical Guidelines for the Development and Management of Weapon System Software [Internet]. Available: https://www.korea.kr/archive/expDocView.do?docId=35418

[4] K. Czarnecki, S. Helsen, "Feature-Based Survey of Model Transformation Approaches," *IBM Systems Journal,* Vol. 45

No. 3, pp. 621-664, 2006.

[5] OMG. MOF 2.0/XMI Mapping, v2.1.1. OMG Available Specification [Online]. Available: https://www.omg.org/spec/XMI/2.1.1/PDF

[6] J. J. Chilenski, and S. P. Miller, "Applicability of modified condition/decision coverage to software testing," *Software Engineering Journal*, Vol. 9, No. 5, pp. 193-200, 1994.

[7] T. Chen, X. S. Zhang, S. Z. Guo, H. Y. Li, and Y. Wu, "State of the art: Dynamic symbolic execution for automated test generation," *Future Generation Computer Systems*, Vol. 29, No. 7, pp. 1758-1773, 2013.

[8] Google Code. CREST Project Page [Internet]. Available: http://code.google.com/p/crest

[9] C. Cadar, D. Dunbar, and D. Engler, "Klee: unassisted and automatic generation of high-coverage tests for complex systems programs," In *OSDI*, Vol. 8, pp. 209-224, Dec. 2008.

[10] D. H. Kim, R. Y. C. Kim, "A Study on Automatic Test Case Extraction Mechanism from UML State Diagrams Based on M2M Transformation," *The Journal of the Institute of Internet, Broadcasting and Communication,* Vol. 13, No. 1, pp. 129-134, 2013.

[11] S. Woo, H. S. Son, and R. Y. C. Kim, "A study on extending message-sequence diagram for mapping cause-effect diagram," In *Proceedings of the Korea Information Processing Society Conference*, pp. 1251-1254, 2012.

[12] J. P. Galeotti, G. Fraser, and A. Arcuri, "Improving search-based test suite generation with dynamic symbolic execution," In *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*, pp. 360-369. Nov. 2013.

[13] Eclipse. Eclipse Modeling Framework Project (EMF) [Internet]. Available: http://www.eclipse.org/modeling/emf

[14] OMG. XML Metadata Interchange (XMI) Specification, Version 2.5.1 [Online]. Available: https://www.omg.org/spec/XMI/2.5.1/PDF

[15] IEEE, *IEEE Standard for Software and System Test Documentation (IEEE Std 829-2008)*, New York, 2008.

[16] GOV.UK. Guidance for Completing DPS for a Missile System [Internet]. Available: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/437322/DPS_principles_guidance.PDF

손 현 승 (Hyun Seung Son)
2015년 8월 홍익대학교 전자전산공학과 공학박사
2017년 10월 ~ 2021년 2월: (주)모아소프트 SBAS 사업부 팀장
2021년 3월 ~ 현재 : 국립목포대학교 컴퓨터공학과 조교수
※관심분야 : 소프트웨어공학, 메타모델, 모델변환, 테스팅