사양 정합성 자동 검사 방법

김영수*, 김장복*, 최경희*, 정기현**, 장중순***, 박승규*
*아주대학교 정보통신전문대학원
**아주대학교 전자공학부
***아주대학교 산업정보시스템공학부
e-mail : sooyasarang@gmail.com

Automated Checking of Specification Consistency

Young-Soo Kim*, Jang-Bok Kim*, Kyung-Hee Choi*,
Gi-Hyun Jung**, Joong-Soon Jang***, Seung-Kyu Park*
*Graduate School of Information and Communication, Ajou University.
**School of Electronics Engineering, Ajou University
***Industrial and Information Systems Engineering, Ajou University.

요 약

최근 임베디드 시스템의 안정성이 제품의 상품성에 매우 중요한 요인이 되면서, 임베디드 시스템의 내장된 소프트웨어의 품질 검증이 중요해졌다. 내장된 소프트웨어를 검증을 위해서 자동 테스트 방식을 사용할 때, 테스트 오라클이 필요하다. 테스트 오라클을 정확하게 구축하기 위해서는 시스템의 요구사항을 정확하고 수행 가능한 형태로 기술하여야 한다. 따라서 테스트 오라클 생성의 기반이 되는 시스템 사양에서 오류를 검출하는 작업은 매우 중요한 작업이다. 본 논문에서는 사양에 내재가 가능한 다양한 오류 중에서 정합성 오류를 검출하는 방법을 제안한다.

1. 서론

과거의 임베디드 시스템은 간단하고 단순한 반복 작업을 필요로 하는 곳에 주로 사용되었으나 최근에는 핸드폰, 자동차 에어컨, 의료장비 및 각종 가전기기에 이르기까지 점점 그 사용의 범위가 넓어지고 있다. 이와 함께 임베디드 시스템의 안전성이 제품의 상품성에 매우 중요한 요인이 되면서, 개발 및 생산뿐만 아니라 검증 과정도 중요해지고 있다. 특히 임베디드 시스템의 하드웨어의 품질 검증뿐만 아니라 내장된 소프트웨어의 품질 검증도 매우 중요해 지고 있다. 아울러 소프트웨어의 효과적 검증 기법에 대한 다양한 관점에서 많은 연구가 진행되고 있다.

내장된 소프트웨어를 검증하는 하나의 방법으로서 시스템을 테스트하는 전략이 사용되고 있다. 테스트란, 시스템에 시험 목적에 맞는 값을 입력한 후, 시스템의 출력을 측정하고, 그 출력값이 시스템의 사양과 일치하는 가를 검사하는 행위를 말한다. 시험을 사람이 직접 실시하는 수동 테스트(Manual Test)는 시간 비용이 많이 들고, 시스템의 품질을 확인할 정도의 충분한 입력에 대하여 테스트를 실시하기 어려워서 테스트의 신뢰성이 떨어진다. 이에 따라 값의 입력이나 출력을 프로그램에 의하여 자동으로 실시하는 자동 테스트(Automate Test)방식이 매우 효과적이다.

출력이 사양과 일치하는 지를 자동으로 확인하기 위하여는 테스트 오라클(Test Oracle)이 필요하다. 테스트 오라클이란 임베디드 시스템의 동작에 대한 답안지에 비유될 수 있다. 즉, 입력을 문제로 볼 때, 그 문제에 대한 답인 출력을 기술하고 있다. 따라서, 자동 테스트를 위하여는 테스트 오라클을 정확하게 작성하여야 한다. 테스트 오라클을 구축하기 위하여는 시스템의 요구사항(Requirements)을 정확하고 수행 가능한 형태로 기술되어야 한다. 이는 테스트를 수행하는 과정에서 다양한 테스트 케이스에 대한 시스템의 응답을 계산하여 시스템이 올바르게 동작하는지를 판단하는 기준을 제공하여야 하기 때문이다.

따라서, 테스트 오라클 생성의 기반이 되는 시스템의 사양에서 오류를 검출하는 작업은 매우 중요한 작업이다. 본 논문에서는 사양에 내재가 가능한 다양한 오류 중에서 정합성 오류를 검출하는 방법을 제안한다. 정합성 오류란, 사양 내에 성립될 수 없는 경우를 정의하였거나 혹은 기술 방법에 오류가 발생한 경우 등을 의미한다. 본 논문에서 추진하는 검사와 유사한 것으로 SCR(Software Cost Reduction)을 들 수 있다[1]. SCR 은 시스템의 요구사항을 사용자에게 친근한 테이블 표기 방식을 사용하여 기술하는 방법이다. [1]에서는 SCR 에 대한 개요와 함께 정합성 검사(Consistency Checking)라고 불리는 유형 오류(Type Errors), 비결정

성(nondeterminism) 그리고 적용 범위(Coverage)와 같은 각종 오류들을 자동으로 검출하는 분석 방식을 제공한다. 본 논문에서는 SCR 과 다른 방법으로 사양이 입력되었을 때, 그 사양에서의 정합성을 검사하는 방법을 제안한다.

본 논문의 구성은 아래와 같다. 2 장에서는 임베디드 시스템의 요구사항을 이해하기 쉽고, 간결하게 기술하는 정형화된 방식을 간단히 소개한다. 3 장에서는 정합 오류 검사를 위하여 사용한 용어들을 정의하였다. 4 장에서는 입력된 사양의 정합성(Consistency)을 자동으로 체크하는 방법에 대해서 기술하고, 이어서 5 장에서 결론을 기술한다.

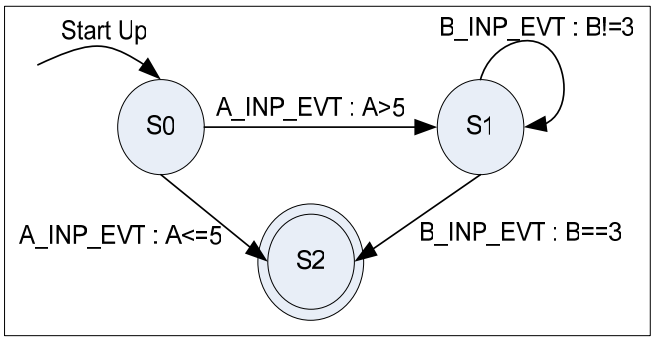
2. 테이블 기반의 사양의 입력 방법

본 논문에서는 임베디드 시스템의 요구사항을 사용자에게 친근하며 직관적인 테이블 방식으로 기술하는 방법을 CSPEC 이라 부른다. CSPEC 은 크게 시스템의 전체적인 행동을 기술하기 위한 유한 상태 기계(Finite State Machine)을 기반으로 한 방식과 단위 기능을 기술하기 위한 로직(Logic)이라는 두 가지 방식을 이용하여 임베디드 소프트웨어의 요구사항을 기술한다.

2.1 CSPEC 테이블

CSPEC 은 시스템 전체에 사용될 변수를 정의하는 테이블, FSM 관련 테이블 3 개 그리고 로직을 기술한 1 개의 테이블로 구성되어 있다. FSM 관련해서는 FSM 의 상태를 정의하는 테이블, FSM 에 사용되는 이벤트를 정의하는 테이블, 각 상태에서 이벤트 처리를 기술한 테이블이 있다.

예를 들면 다음과 같다. 다음 그림과 같은 간단한 FSM 이 있을 때 이를 CSPEC 에서 정의한 방식으로 표현하면 아래 [표 1~4]와 같다.



[그림 1] 간단한 FSM 의 예

이름	초기값	최소값	최대값
A	0	0	10
B	0	0	5

[표 1] 변수 테이블

이름	상위 상태	값
S0	System_On	0
S1	System_On	1
S2	System_On	2

[표 2] FSM 상태 테이블

이벤트	우선 순위
A_INP_EVT	0
B_INP_EVT	1

[표 3] FSM 이벤트 정의 테이블

상태	이벤트	결정	다음상태
S0	A_INP_EVT	A>5	S1
		A<=5	S2

[표 4] FSM 이벤트 처리 테이블

3. 용어 정의

정합성 검사를 설명하기 전에, 본 논문에서 사용하는 용어들을 정의하고자 한다.

- 조건(Condition) – 논리 연산자가 없는 논리식을 의미하며, true 혹은 false를 값으로 가진다[5]. 예를 들면, A>5 등을 가리킨다.
- 결정(Decision) –논리 연산자로 구성된 논리식을 가리킨다. 논리 연산자가 없는 결정은 조건과 같다. 만약 결정에 조건이 한번 이상 나타난다면 각 조건은 서로 다른 것이라고 가정한다[5].
- Explicit Condition – 하나의 변수, 관계 연산자, 그리고 상수로 이루어진 논리식을 가리킨다.
- Implicit Condition – 두 개 이상의 변수로 이루어진 논리 식을 가리킨다.
- Functional Condition – 미리 정의된 함수를 사용하여 정의된 논리 식을 가리킨다.
- Type 1 Decision – Explicit Condition과 논리 연산자로 구성된 논리 식을 가리킨다. 논리 연산자가 없는 경우도 포함한다.
  - Type 1.A : 하나의 Explicit Condition만 있는 논리식을 가리킨다.
  - Type 1.B : Explicit Condition과 1개 이상의 AND 논리 연산자만으로 구성된 논리식을 가리킨다.
  - Type 1.C : Explicit Condition과 1개 이상의 OR 논리 연산자만으로 구성된 논리식을 가리킨다.
  - Type 1.D : Explicit Condition과 AND와 OR 논리 연산자가 각각 1개 이상씩 포함된 논리식을 가리킨다.
- Type 2 Decision – Implicit Condition과 논리 연산자로 구성된 논리식을 가리킨다. 논리 연산자가 생략이 가능하다.
- Type 3 Decision – Functional condition과 논리 연산자로 구성된 논리 식을 가리킨다. 논리 연산자가 생략이 가능하다.
  - Type 3.A : compile time에 수행할 수 있는 함수로 구성된 functional condition과 논리 연산자로

구성된 논리식을 가리킨다. 논리 연산자가 생략이 가능하다.

- Type 3.B : run time에만 수행할 수 있는 함수로 구성된 functional condition과 논리 연산자로 구성된 논리식을 가리킨다. 논리 연산자가 생략이 가능하다.

4. 정합성 검사

4.1 정합성 검사항목

본 논문에서는 정합성(Consistency) 검사를 위하여 사양을 기술 할 때, 반드시 지켜야 하는 사항과 기술상 범하기 쉬운 논리적 오류를 기준으로 다음과 같이 분류한다.

① 문법 검사 – 사전에 정의된 문법에 잘 따르고 있는지 검사한다.
② 키워드 검사 – 사양을 기술할 때 사용이 가능한 키워드가 사용 규칙에 맞게 적절히 사용되었는지를 검사한다. 또한, 사용한 키워드가 사전에 등록이 된 키워드 인지를 검사한다.
③ 초기값 검사 – 사양 기술에 사용된 변수들의 초기값이 변수의 최소~최대 범위 안에 존재하는지 검사한다.
④ 모호성 검사 – 시스템의 모든 동작이 결정적(deterministic)인지를 검사한다. 동일한 상태에서 동일한 사건이 벌어 졌을 때, 두개 이상의 상태로 전이가 가능하다면 모호하다고 정의한다.
⑤ 모순성 검사 – 논리적으로 모순이 되는 결정이 있는지 검사한다. 예를 들면, 절대로 벌어질 수 없는 사건을 이용하여 상태전이 조건이 기술되었다면 이 사양은 모순을 가지고 있다고 정의한다.
⑥ 적용 범위 검사 – 조건에 사용된 각각의 변수가 가질 수 있는 모든 값을 대상으로 사양이 정의 되었는지를 검사한다.

①~③의 항목은 ④~⑥의 항목을 검사하기 전에 먼저 검사되어야 하며, 쉽게 검사할 수 있는 것들로 자세한 설명은 생략한다.

4.2. 모호성 검사

결정들이 모호하다는 말은 시스템의 동작이 예측불가능 하다는 말과 같다. 예를 들어, 표 6 과 같이 현재 상태가 S1 이며 A_INP_EVT 이벤트가 발생했을 때 전이가 가능한 다음 상태가 정의되어 있다. 만약 A 와 B 의 값이 각각 5 와 3 일 때, 첫 번째 조건과 세 번째 조건이 모두 만족된다. 이 경우 시스템은 다음 상태를 S2 로 변경할지 아니면 S1 상태를 유지할지를 결정할 수 없다. 시스템이 결정적이어야 할 경우, 모호성을 체크해야 한다.

상태	이벤트	결정	다음상태
S1	A_INP_EVT	A==5 && B==3	S2
		A==5 && B!=3	S1
		A==5	S1

[표 5] FSM 이벤트 처리 테이블

모호성을 검사하기 위해서는 해당 결정들이 항진식(tautology)인지를 검사하면 된다[1]. 예를 들어 D1, D2, D3 라는 결정들이 있을 때 [D1 && D2 && D3]=false 이면 각 결정들은 서로 겹치는 경우가 없다. 하지만 결정들을 구성하는 서로 다른 조건들이 모두 n 개일 때 이를 검사하기 위해서는 2^n 개의 interpretation 들을 검사해보아야 한다[3]. 각 조건들 마다 참과 거짓을 모두 대입해봐야 하기 때문이다.

더욱이, 조건을 참 또는 거짓으로 만드는 것조차 쉽지 않다. 정합성 검사기(Consistency checker)는 조건을 참 또는 거짓으로 만들기 위해서 조건 내에 사용된 변수의 값을 결정해야 하기 때문이다. 다음 [표 7]의 변수 테이블과 [표 8]과 같은 조건들을 고려하여 보자.

이름	초기값	최소값	최대값
A	0	0	10
B	0	0	5
C	0	0	10

[표 6] 변수 테이블

ID	유형	조건
C1	Explicit	A>5
C2	Implicit	A+B>C
C3	Functional	Range(A,3,7)
C4	Functional	Current_Time() - Changed_Time(A)>400

[표 7] 유형별 조건의 예

C1 을 참으로 만드는 정수 A 의 값은 {6,7,8,9,10} 등 5 가지이다. 그리고 거짓으로 만드는 A 의 값은 {0,1,2,3,4,5} 6 가지 이다. C2 는 A, B, C 의 값의 조합에 의해서 결정되므로 테스트 실행 시에 실제 변수들이 가지는 값에 따라 참 또는 거짓이 결정된다. 만약 C2 에 대해서 모든 경우를 다 테스트 해보고자 한다면 726 가지의 값을 대입해보아야 할 것이다. 사용된 변수의 수(n)가 많고 그 범위(m)가 클수록 경우의 수는 m^n 에 비례하여 증가하게 된다.

C3 는 Range 라는 미리 정의된 함수를 사용하여 [A>3 && A<7]을 표현한 것이다. 따라서 C3 를 참으로 만드는 A 의 값은 {4,5,6} 3 가지 이며, 거짓으로 만드는 값은 {0,1,2,3,7,8,9,10} 8 가지 이다. 마지막으로, C4 의 의미는 “A 의 값이 변한 후로 400ms 가 지났으면” 이라는 뜻이다. 이 때 Current_Time()이라는 현재 시간을 얻어오는 함수를 사용하는데다 A 의 값이 언제 변할지 알 수 없기 때문에 C4 는 실행 도중이 아니면 참 또는 거짓을 결정할 수 가 없다.

아래의 알고리즘은 C1 과 C3 의 경우 조건을 참으로 만드는 원소들의 집합(이하 True Set)을 O(1)의 시간 복잡도로 구할 수 있다는 점을 바탕으로 하여 4.1 장에서 정의한 Type 1 에 해당하는 결정들이 항진식인지를 검사하는 알고리즘이다. 제안된 알고리즘은 $O(2^n)$의 시간 복잡도를 O(nm)으로 줄이고 있음을 알 수 있다.

```
For each Condition(Variable)
   Mark Elements of True Set
For i=0 to Numer_of_Decisions – 1
   If type of decision[i] is Type 1 Then
      Cnt_Conflict_Variable=0;
      For j=i+1 to Number_of_Decisions
         For each Variable of Decision[i]
            Decision[j]에 같은 변수가 없으면
               Cnt_Conflict_Variable++;
               Continue next variable;
            Calculate Intersection of True Sets
               If Intersection is not Empty Set Then
                  Cnt_Conflict_Variable++;
         End for
         If type of decision[i] and [j] is Type 1.B Then
            If Cnt_Conflict_Variable ==
            Number of Elements of Union of Variables in
            Decision[i] and Decision[j] Then
               Return –EAMBIGUTY;
         Else
            If Cnt_Conflict_Variable >= 1 Then
               Return –EAMBIGUTY;
    Next j
Next i
```

4.3 모순성 검사

사용자가 방대한 분량의 사양을 작성할 때 논리적으로 모순된 결정을 기술할 수도 있다. 모순된 결정이란 논리적으로 동시에 발생할 수 없는 두 조건이 논리 연산자로 연결된 것으로 정의한다. 예를 들면 다음과 같다.

ID	유형	조건
D1	Type 1.B	A>5 && A<5
D2	Type 1.C	B==4 || B!=4

[표 8] 논리적 모순이 있는 결정의 예

D1 에서 A 의 어떤 원소도 A<5 와 A>5 를 동시에 만족시킬 수 없다. 따라서 D1 은 항상 거짓이 된다. 반대로 D2 는 항상 참이 된다. 이처럼 결정의 모순성(Contradiction)은 모호성(Ambiguity)처럼 시스템의 수행에 결정적인 영향을 미치지는 않지만, 대부분이 사용자의 고의가 아닌 실수 때문에 발생하는 논리적인 모순이라고 판단된다.

다음은 Type 1.B 와 Type 1.C 의 결정에 대해서 True Set 을 이용하여 모순성을 검사하는 알고리즘이다. 변수의 수를 n, 범위를 m 이라고 할 때 이 알고리즘의 시간 복잡도는 O(nm)이 된다. 알고리즘에서는 우선 각 조건에 사용된 변수 별로 True Set 을 구하여 마크한다. 그리고 D1 처럼 Type 1.B 의 경우에는 교집합을 구한다. 만약 교집합이 공집합이면 결정을 참으로 만드는 원소가 없는 것이므로 항상 거짓이 된다. 그리고 D2 처럼 Type 1.C 의 경우에는 합집합을 구하고 이것이 전체 집합이면 항상 참이 된다. 이렇게 참 또는 거짓이라고 명시하지 않았는데 논리식에 의해 항상 참이 되거나 거짓이 되는 경우를 모순이라고 한다.

```
If type of a decision is Type 1.B or Type 1.C Then
      For each Condition(Variable)
            Mark Elements of True Set
      For each Variable
            If Type 1.B Then
                  Calculate Intersection of True Sets
                  If Intersection is Empty Set Then
                        Return –ECONTRADICTION;
            If Type 1.C Then
                  Calculate Union of True Sets
                  If Union is Universal Set Then
                        Return –ECONTRADICTION;
End If
```

4.4. 적용 범위 검사

적용 범위 검사란 조건에 사용된 각각의 변수가 자신의 모든 원소에 대해서 처리사항을 빠짐없이 기술하였는지 검사하는 것이다. 적용범위 검사는 True Set 을 이용하면 간단히 구할 수 있다. 각 조건들의 True Set 을 구하고 이들의 합집합이 전체 집합이 아니면 누락된 조건이 있는 것이다.

5. 결론

본 논문에서는 테이블 형식을 기반으로 시스템의 요구사항을 기술하는 방법에 대해서 살펴보고, 이 때 발생 가능한 정합성 오류를 검출하는 방법에 대해서 살펴 보았다. 모호성, 모순성, 그리고 적용 범위 검사로 나누고 각 검사별 알고리즘을 제시하였다. 앞으로 실제 시스템의 요구사항을 테이블로 기술하고, 기술된 사양의 정합성 검사를 통해서 제안한 검사방법의 효율성 및 실용성을 측정할 예정이다.

참고문헌

[1] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. “Automated consistency checking of requirements specifications.” ACM Trans. Software Eng. and Methodology, 5(3):231-261, April-June 1996.

[2] C. L. Heitmeyer, J. Kirby, and B. Labaw, “Tools for Formal Specification, Verification, and Validation of Requirements”, In Proc. 12th Annual Conf. on Computer Assurance (COMPASS ’97), Gaithersburg, MD, June 1997.

[3] Smullyan, R. M. “First-Order Logic.” Springer-Verlag, 1968. Republished by Dover Publications Inc., 1993.

[4] K. J. Hayhurst, D. S. Veerhusen, “A Practical Approach to Modified Condition/Decision Coverage”, 20th Digital Avionics Systems Conference (DASC), Daytona Beach, Florida, USA, October 14-18, 2001, Vol. 1, pp. 1B2/1-1B2/10.

[5] RTCA/DO-178B, Software Considerations in Airborne Systems and Equipment Certification, Washington, D. C. RTCA, Inc., December 1992.