특집 04

파라미터를 고려한 컴포넌트 인터페이스의 최적 테스트 시퀀스 생성 기법

신영술·이우진 (경북대학교)

목 차 »

초 록

컴포넌트의 외부 행위는 파라미터를 가진 인터페이스로 정의된다. 소프트웨어 개발자는 컴포넌트를 테스트하기 위해 인터페이스를 통해 각기 다른 파라미터의 값을 반복적으로 입력하고, 입력값에 따른 출력값을 관찰한다. 테스팅에 소요되는 시간을 줄이기 위해 테스트 케이스를 자동으로 실행하는 테스팅 자동화 도구가 효율적이지 않은 테스트 시퀀스를 수행한다면 테스팅 자동화의 효과는 줄어든다. 유한 상태 머신을 기반으로 하는 기존의 테스트 시퀀스 생성 기법들은 파라미터를 가진 인터페이스 테스팅에 최적화된 테스트 시퀀스를 제공하지 않는다. 이 연구는 컴포넌트 인터페이스를 파라미터를 고려한 상태 모델로 표현하고 최적의 시퀀스 생성 기법을 제안한다. 최적의 시퀀스 생성 기법은 파라미터를 가진 상태 기반의 행위 모델에서 특정 간선을 원하는 회수만큼 수행을 보장하는 시퀀스를 생성하며, 생성된 시퀀스는 최적의 테스트 수행 시간을 갖는다.

* 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No.2010-0010606).

1. 서 론

도요타는 2010년형 프리우스 일부 모델에서 발생하는 ABS(Anti-lock Brake System)의 소프트웨어 문제로 자발적 리콜을 발표하였다[1]. 프리우스 운전자들은 브레이크를 작동시킬 때마다 일정하지 않은 제동력 때문에 불만을 표시하였다. 도요타는 13만대 이상의 프리우스를 리콜하고 ABS 제어 소프트웨어를 업그레이드 할 것을 결정하였다. 2001년에는 파나마 시티의 암치료 전문 의료기관에서 방사선 치료 도중 환자에게 방사선을 과다 사용하여 인명 피해가 발생하였다[2]. 오랜 조사 끝에 방사능을 제어하는 소프트웨어의 문제로 밝혀졌지만, 이미 2005년까지 28명의 방사선 과다 노출 환자 중에서 23명의 환자가 사망하였

고 그 중 적어도 18명이 방사선 과다 노출의 직접적인 피해를 입었다.

소프트웨어의 오류 수정에는 많은 비용이 소요되므로 소프트웨어 개발 초기 단계에서 오류를 검출하고자 노력하고 있다. 구현 단계의 개발자들은 소스 코드를 작성하는 순간부터 테스팅을 시작하며, 단위 테스팅을 가능한 많이 수행하여 통합 테스팅의 비용을 줄이려 노력한다. 구글의 약 6000명 정도의 개발자와 테스트 전문가들은 수동 혹은 자동화 방식으로 하루에 150만 번의 테스트를 수행한다[3]. 그들은 디버깅에 소요되는 시간을 줄이기 위해 테스트를 더욱 중요시한다. 효율적인 테스트 케이스와 테스트 절차가 개발 주기 동안 항상 존재하기 때문에 오류가 있는 코드는 즉시 수정될 수 있다. 테스트를 중요시 하는 개발 방법론 중에서 테스트 기반 개발 방법에서는 개발자가 자신의 모듈을 구현하기 전에 테스트 케이스를 먼저 작성하도록 한다[4]. 애자일(Agile) 소프트웨어 개발 방법에서는 짧은 테스트 실행 주기를 통해 피드백을 제공한다[5].

마이크로소프트는 테스트 자동화에 많은 관심을 기울이고 있다[6]. 많이 이용되는 테스팅 방법 중의 하나가 입력 공간 분할인데 테스트 자동화를 통해 테스팅 시간을 줄인다. 테스팅 자동화 도구의 목적은 테스팅 시간을 줄이는 것이지만, 테스팅 자동화 도구는 주어진 테스트 케이스를 실행하고 결과값을 기대값과 비교한다. 테스팅 자동화 도구가 테스팅 시간을 줄이기 위해서는 테스팅 시간 측면에서 최적화된 테스트 시퀀스를 수행해야만 한다.

컴포넌트 기반 개발방법(Component-Based Development, CBD)에서는 시스템을 구성하는 단위가 컴포넌트이다. 컴포넌트는 내부 구현과 독립적인 인터페이스에 의해 캡슐화된다[7]. 컴포넌트의 내부는 인터페이스에 의해 숨겨지고 인터페이스를 통해서만 접근이 가능하다. 컴포넌트의 인터페이스는 다른 컴포넌트와 상호작용하는 방식의 프로토콜로 이해할 수 있다. 컴포넌트 기반 시스템에서 발생하는 오류는 주로 컴포넌트의 인터페이스 상에서 발생한다[8]. 시스템을 구성하는 단위인 컴포넌트의 신뢰성이 전체 시스템의 신뢰성을 결정한다[9]. 따라서 CBD에서는 컴포넌트의 외부 행위를 검증하는 인터페이스 테스팅의 필요성이 강조된다.

기존의 테스트 시퀀스 생성 기법은 유한 상태 머신을 이용하여 행위를 모델링 한다[10]. 유한 상태 머신은 외부 트리거에 의해서 상태 천이와 출력이 결정된다. 시스템이 가지는 데이터 흐름은 모델링 되지 않는다. 따라서 기존의 테스트 시퀀스 생성 기법은 파라미터의 제약 사항을 고려한 컴포넌트 인터페이스의 동적인 행위를 표현하는 모델에는 부적합하다. 또한 테스트 시퀀스를 최적화하는 기법은 상태 기반의 모델에서 모든 간선을 적어도 한 번 이상 수행하는 테스트 시퀀스를 생성한다[11]. 하지만 파라미터를 고려한 테스팅을 위해서는 특정 간선을 $n(\geq 1)$번 이상 수행할 수 있는 최적의 테스트 시퀀스를 생성해야만 한다. 이 연구는 데이터 제약 사항을 표현할 수 있는 상태 기반의 행위 모델을 이용하여 컴포넌트의 동적 행위를 모델링하였다. 또한 상태 기반의 행위 모델에서 특정 간선의 수행을 $n(\geq 1)$번 이상 보장하며 테스트 수행 시간이 최적인 테스트 시퀀스를 생성하는 알고리즘을 제시한다.

이 연구의 구성은 다음과 같다. 제 2장에서 기존의 테스트 시퀀스 생성의 관련 연구 내용을 기술하고 제 3장에서는 제안하는 최적 테스트 시퀀스 생성 기법을 기술한다. 제 4장에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

블랙박스 테스팅은 소프트웨어 모듈에 입력된 특정 값에 따른 출력값을 관찰하여 소스 코드가 명세서와 동일한지를 검사한다[12]. 블랙박스 테스팅은 소프트웨어 모듈을 외부 환경의 관점에서 접근하는 방법이다. 소프트웨어의 내부 구조 정보는 블랙박스 테스팅에서 사용되지 않는다.

모델 기반의 테스팅은 시스템의 요구사항 정보를 표현하는 정형 모델을 이용한다. 모델 기반의 테스팅에는 주로 유한 상태 머신이 사용된다[13, 14]. 유한 상태 머신은 테스트 시퀀스를 생성하기 위한 Distinguishing Sequence(DS)나 Unique Input/Output(UIO) 시퀀스 기법에서 널리 이용된다. 하지만 유한 상태 머신은 데이터 흐름 측면은 나타내지 못한다. 유한 상태 머신을 확장한 확장 유한 상태 머신은 간선의 수행을 제어하는 제약사항을 모델링 한다. 이 제약사항은 상태 변수와 입력 파라미터 값을 이용하여 평가된다[15]. 따라서 확장 유한 상태 머신은 컴포넌트의 동적인 행위를 모델링 할 수 있다. 유한 상태 머신 외에 UML이나 페트리넷도 테스트 시퀀스 생성에 이용된다.

컴포넌트의 인터페이스는 컴포넌트의 외부에서 접근할 수 있는 함수로 구성되어 있다. 일반적인 함수의 프로토타입은 입력 파라미터를 가지며, 인터페이스의 행위 모델은 함수의 파라미터와 파라미터의 제약 사항을 기술해야 한다. 파라미터를 고려하지 않은 인터페이스의 행위 모델은 콜 트리거가 아닌 시그널 트리거만 표현할 수 있다[16]. Supaporn Kansomkeat와 Wanchai Rivepiboon은 UML 상태 모델을 이용한 테스트 케이스 자동 생성을 연구하였다[17]. 하지만 모델에 사용된 파라미터는 트랜지션 수행에 영향을 미치지 않는다. 일반적으로 입력 파라미터의 값은 제어 흐름에 영향을 갖는다[18, 19]. 따라서 컴포넌트 인터페이스를 올바르게 모델링하기 위해서 입력 파라미터의 제약 사항을 기술해야 한다.

입력 파라미터를 가진 인터페이스를 테스팅하는 테스트 시퀀스는 같은 함수를 여러 번에 걸쳐서 수행한다. 예를 들어, 동등 분할과 경계값 분석은 블랙박스 테스팅에서 많이 사용되는 테스팅 기법이다[20, 21]. 동등 분할은 입력 파라미터 값의 도메인을 파티션으로 나누고, 각 파티션을 대표하는 테스트 케이스를 작성한다. 경계값 분석은 소프트웨어의 오류가 입력 도메인의 경계 부근에서 주로 발생한다는 특성을 이용한다. 경계값 분석의 테스트 케이스는 동등 분할의 파티션 경계값에서 테스트 케이스를 생성한다.

테스트 시퀀스 생성 방법 중에서 널리 사용되는 것은 UIO-method, D-method, W-method, T-method 이다[22]. UIO-method와 이에 기반한 방법들은 UIO 시퀀스를 생성한다[23]. 특정 상태의 UIO 시퀀스는 다른 상태에서 생성되지 않는 출력 시퀀스를 생성한다. D-method는 DS(Distinguishing Sequence)를 생성한다. 입력 문자열 x가 각 상태에 대해서 다른 출력 문자열을 생성하면 x를 DS라 한다. DS의 입력은 모든 상태에 대해 동일하지만 출력이 다르다. UIO 시퀀스의 입력은 각 상태에 따라 다르다. W-method는 특성화(characterization) 집합을 생성한다. 특성화 집합은 입력 문자열 $a_1,\ldots,a_k$ 로 구성되어 있으며, 이 입력 문자열은 각 상태에 따라 출력되는 마지막 문자가 다르다. T-method는 모든 트랜지션이 적어도 한 번 이상 수행될 때까지 임의의 입력 시퀀스를 생성한다. 위의 모든 테스트 시퀀스 생성 방법론은 완전히 명세된 밀리 머신을 가정한다. 모델의 모든 상태에서 모든 입력에 대해 출력이 정의되면 그 모델은 완전히 명세되었다라고 한다. 그러나 일반적으로 인터페이스의 모든 함수가 출력값을 가지지는 않는다.

유한 상태 머신 기반의 테스트 시퀀스 최적화는 Chinese Postman Tour 알고리즘을 사용한다 [24], [25]. 이 알고리즘은 오일러 투어가 존재하는 그래프의 모든 간선을 적어도 한 번 포함하는 경로를 찾는다. 비대칭적인 그래프는 오일러 투어를 가지지 않기 때문에 비대칭 그래프의 간선을 복사하여 비대칭 그래프를 대칭 그래프로 변환한다. 방향그래프의 모든 정점에 대해 각 정점으로 들어오는 간선과 정점으로부터 나가는 간선의 개수가 같으면 대칭이라 한다. 대부분의 경우에 Chinese Postman Problem(CPP) 알고리즘은 NP-complete 문제로 알려져 있다. 입력 파라미터를 가진 모델의 테스트 시퀀스를 생성하는 알고리즘은 특정 간선을 적어도 $n(n>=1)$의 수행을 보장해야 한다. 하지만 CPP 알고리즘은 적어도 한 번 이상의 수행만 보장한다.

본 논문은 컴포넌트의 인터페이스를 테스트하는 방법론을 제시한다. 컴포넌트의 인터페이스는 입력 파라미터를 가지며, 인터페이스를 구성하는 함수들은 각각 다른 수행 시간을 가진다. 본 논문에서 제시하는 방법론은 파라미터를 고려한 테스트 수행 시간이 최적인 시퀀스를 생성한다.

3. 최적 테스트 시퀀스 생성 기법

본 장에서는 테스트 시퀀스를 생성하기 위한 프로세스와 파라미터를 가진 컴포넌트 인터페이스의 행위를 표현할 수 있는 상태 기반의 모델을 기술한다. 또한 최적의 인터페이스 테스팅을 위해 컴포넌트 인터페이스 행위 모델과 주어진 테스트 케이스를 맵핑하는 기법과 테스트 케이스가 맵핑된 모델을 이용하여 최적의 테스트 시퀀스를 생성하는 기법을 제시한다.

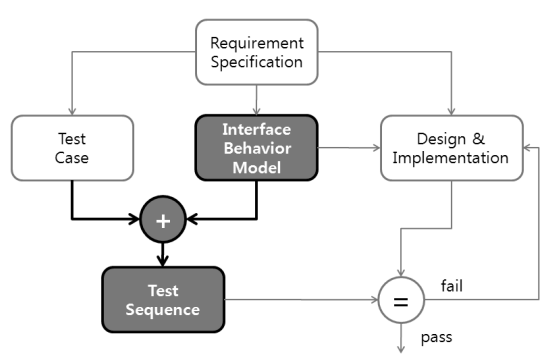
3.1 테스트 시퀀스 생성 프로세스와 모델

컴포넌트의 인터페이스 테스팅을 위한 테스트 시퀀스를 생성하기 위해서는 컴포넌트의 외부 행위 관점을 표현하는 인터페이스 행위 모델이 필요하다. 요구 명세서로부터 테스트 케이스와 인터페이스 행위 모델이 정의된다. 본 논문에서는 테스트 케이스는 주어진다고 가정한다. 인터페이스 행위 모델은 주어진 테스트 케이스와 맵핑되고 최적 테스트 시퀀스 생성 알고리즘이 테스트 시퀀스를 생성한다. 인터페이스 행위 모델은 테스트 시퀀스 생성만이 아니라, 세부 디자인 및 구현에 이용될 수 있다. (그림 1)은 테스트 시퀀스 생성 프로세스를 보인다. 최적 테스트 시퀀스 생성 알고리즘은 (그림 1)의 짙게 표시된 부분에서 사용된다.

컴포넌트의 외부 행위를 표현하기 위한 인터페이스 행위 모델은 Normal Form EFSM(NF-EFSM)으로 나타낸다.

[정의 1] NF-EFSM M은 8-투플 $(S, s_0, V, \sigma_0, P, I, O, T)$이다.

· S는 논리적 상태의 유한 집합이다.



(그림 1) 테스트 시퀀스 생성 프로세스

· $s_0 \in S$는 초기 상태이다.
· V는 상태 변수의 유한 집합이다.
· σ_0는 상태 변수에 초기값을 할당하는 함수이다.
· P는 입력 및 출력 파라미터의 집합이다.
· I는 입력 선언 집합이다.
· O는 출력 선언 집합이다.
· T는 트랜지션 집합이다.

[정의 2] 트랜지션 $t \in T$는 5-투플 (s_s, g_I, g_D, op, s_f)이다.

· s_s는 t의 시작 상태이다.
· g_I는 3-투플(i, P^i, g_{p^i})로 표현되는 입력의 조건이다.
  ○ $i \in I \cup \{NIL\}$
  ○ $P^i \subseteq P$
  ○ g_{P^i}는 V'와 P'의 변수들의 논리적 표현으로 나타내는 입력 파라미터의 조건이다.

$$V' \subseteq V, \varnothing \neq P' \subseteq P^i.$$

· g_D는 V''의 변수들의 논리적 표현으로 나타내는 상태 변수의 조건이다.

$$\varnothing \neq V'' \subseteq V$$

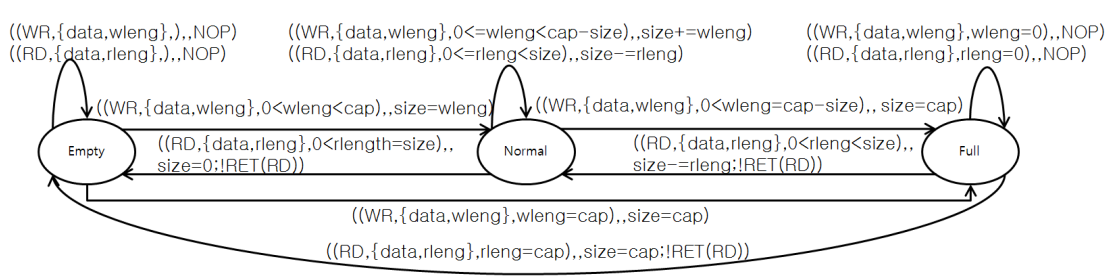
· op는 할당문과 출력문으로 구성된 순차적 연산이다.
· s_f는 t의 목적 상태이다.

[정의 3] 도메인 변환 함수 dom은 상태나 논리 표현의 도메인을 출력한다.

· $s \in S, dom(s)$는 s가 인코딩하는 도메인이다.
· 논리 표현 exp의 $dom(\text{exp})$는 exp의 도메인이다.

NF-EFSM으로 표현된 인터페이스 행위 모델 M에서 $s_i, s_k \in S, s_i \neq s_k \Leftrightarrow dom(s_i) \neq dom(s_k)$이다.

인터페이스 행위 모델의 예를 (그림 2)에 보인다. (그림 2)의 메시지 큐는 고정된 크기 cap (> 2)를 가지며, 두 개의 함수 Write(data,wleng)와 Read(data,rleng)가 메시지 큐의 인터페이스를 구성한다. 함수 Write(data,wleng)는 길이 wleng의 메시지인 data를 메시지 큐에 입력하고, 함수 Read(data,rleng)는 길이 rleng의 메시지를 메시지 큐로부터 읽어 파라미터 data에 입력하고, 읽어들인 data에 해당하는 메시지를 메시지 큐에서 삭제한다. 함수 Write(data,wleng)와 Read(data,rleng)는 줄여서 모델에 WR과 RD로 표기한다. (그림 2)의 예제에서는 호출 함수의 반환값을 RET로 표현한다. (그림 2)의 메시지 큐가 가지는 세 가지의 상태는 Empty(size=0), Normal(0<size<cap), Full(size=cap)이다.



(그림 2) 크기 cap의 메시지 큐

3.2 테스트 케이스 맵핑

테스트 케이스는 테스트 시퀀스 생성 이전에 미리 주어진다고 가정한다. 테스트 케이스의 테스트 데이터는 인스턴스 형태로 구현되지 않고 추상적인 형태로 표현되므로 상세 설계와 구현 단계에 독립적으로 사용될 수 있다. 테스트 케이스는 다음과 같이 정의된다.

[정의 4] 주어진 테스트 케이스 집합 TC라 할 때, $tc \in TC$는 정의 1에 정의된 NF-EFSM M으로 표현되며 다음 사항을 만족한다.

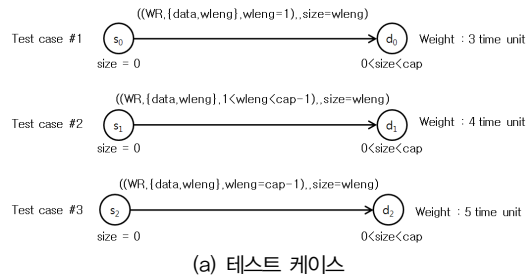
- $|S| = 2$이다.
- $|T| = 1$이다.
- $s_i, s_j \in S$는 연결되어 있다.

테스트 케이스 NF-EFSM $M = (S, s_0, V, \sigma_0, P, I, O, T)$이 인터페이스 모델 NF-EFSM $M' = (S', s_0', V', \sigma_0', P', I', O', T')$에 맵핑되는 조건은 다음과 같다.

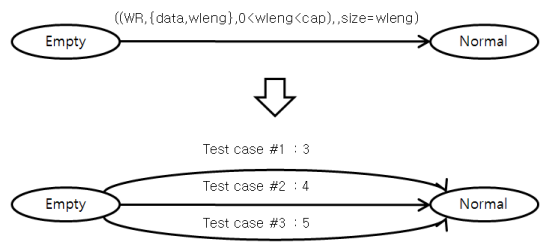
- $t \in T = (s_s, g_I g_D, op, s_f)$이고
  $t' \in T' = (s_s', g_I' g_D', op', s_f')$일 때,
- $dom(s_s) \cap dom(g_D) \supseteq dom(s_s') \cap dom(g_D')$
- $dom(g_I) \subseteq dom(g_I')$
- $dom(s_f) \subseteq dom(s_f')$
- 위의 조건을 만족하면 M은 M'에 맵핑된다.

(그림 3)-(a)는 테스트 케이스의 예를 보여준다. 제시된 세 개의 테스트 케이스는 (그림 2)에 나타난 메시지 큐의 상태 Empty와 상태 Normal 사이의 간선에 맵핑된다. (그림 3)-(b)는 세 개의 테스트 케이스가 맵핑된 모습을 보여준다.

테스트 케이스는 미리 주어지기 때문에 인터페이스 행위 모델의 모든 간선이 테스트 케이스와

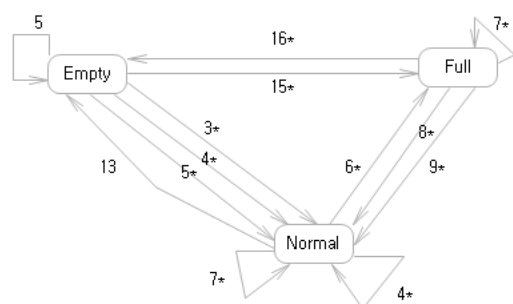


(a) 테스트 케이스



(b) 맵핑된 테스트 케이스

(그림 3) 테스트 케이스 맵핑의 예



(그림 4) 메시지 큐에 맵핑된 테스트 케이스 예

맵핑되지는 않는다. (그림 2)의 메시지 큐 행위 모델에 테스트 케이스가 맵핑된 모습이 (그림 4)에 보인다. 간선의 레이블에 '*'가 표시된 간선이 테스트 케이스가 맵핑된 간선이고 숫자는 테스트 케이스 수행의 가중치이다. 테스트 케이스가 맵핑되지 않은 간선의 가중치는 입력 파라미터에 따라 달라지는 가중치의 평균으로 표시된다.

3.3 최적 테스트 시퀀스 생성

테스트 케이스가 맵핑된 후, 최적의 테스트 시

퀀스 생성 이전에 재시작 간선이 모델에 추가된다. 재시작 간선은 인스턴스를 삭제하고 다시 생성하는 과정이나 하드웨어 상의 재시작을 의미한다. 테스트 케이스가 맵핑된 후의 행위 모델은 다중그래프 $G=(V,E)$로 볼 수 있으며, 최적 테스트 시퀀스 생성 문제는 그래프 상의 최적 경로를 찾는 문제로 바뀐다. 최적 테스트 시퀀스 생성의 구체적인 문제 특성은 세 가지이다. 첫째, 가중치 방향 그래프를 이용한다. 둘째, 임의의 두 정점을 연결하는 간선이 두 개 이상일 수 있다. 셋째, 최적 테스트 시퀀스의 경로는 모든 간선 혹은 특정 간선을 포함해야 한다.

최적 테스트 시퀀스 생성은 CPP 문제와 비슷하지만, 테스트 케이스가 모든 간선에 맵핑되는 것이 아니기 때문에, 모든 간선이 아닌 특정 간선만을 커버해야 하는 시퀀스 생성도 가능해야 한다는 점이 다르다. 위에서 정의된 문제의 특성을 만족하는 최적 테스트 시퀀스 생성 알고리즘은 크게 두 단계의 알고리즘으로 이루어진다. Floyd 알고리즘을 이용하여 임의의 두 정점 사이의 최적 거리를 구는 단계와 분기한정법 알고리즘을 이용하여 최적의 시퀀스를 찾는 단계이다.

분기한정법 알고리즘이 최적의 시퀀스를 찾기 위해 사용하는 최소바운드(Minimum Bound) 함수는 정의 5와 같다.

[정의 5] 최소바운드 함수는 $MinBound()$이다.

$$MinBound(E^{un}, v_{last}, path) = path.length + \underset{0\le i\le|S|}{Min}(Vir(p_i \in S, v_{last}))$$

$$Vir(p_i, v_{last}) = dist(v_{last}, v^{src}) + p_i.length + Observ(p_i, v_{last})$$

집합 S는 E^{un}의 원소를 이용한 $|E^{un}|$-순열을 원소로 갖는다.

E^{un}는 E^{map}의 원소 중에서 커버되지 않은 원소의 집합이다. E^{map}는 테스트 케이스와 맵핑된 간선의 집합이다.

$dist(s,d)$는 정점 s와 d 사이의 최단 거리이다.

$Observ(seq,v)$는 정점 v에서 시작하는 테스트 부분 시퀀스 seq를 수행하는데 필요한 메모리 덤프의 가중치이다.

생성된 테스트 시퀀스를 이용하여 테스트를 수행할 때, 트랜지션의 수행결과와 천이 상태를 검증해야 한다. 이를 검증하기 위해 일반적으로 출력값을 기대값과 비교하는 방법을 사용한다. 하지만 이러한 방법은 모든 입력에 따라 출력이 존재해야 하기 때문에 입력에 따른 출력이 없는 트랜지션의 경우에는 사용될 수 없다. 테스트 대상이 되는 시스템의 메모리를 덤프하는 방식은 입력에 따른 출력이 존재하지 않더라도 트랜지션의 수행결과와 천이 상태를 검증할 수 있다. 최소바운드 함수에 의해 사용되는 함수 $Observ()$가 메모리 덤프의 가중치이다.

3.4 최적 테스트 시퀀스 생성 알고리즘 평가

최적 테스트 시퀀스 생성 알고리즘으로 생성된 테스트 시퀀스의 커버리지는 미리 정의되어 주어지는 테스트 케이스에 의해 결정된다. 트랜지션의 수행결과와 천이 상태를 검증하기 위해 메모리 덤프 방식을 사용하므로, 입력에 따른 출력이 존재하지 않더라도 테스팅의 관찰성은 제공된다. 최적 테스트 시퀀스 생성 알고리즘의 장점은 네 가지이다. 첫째, 최적 테스트 시퀀스 생성 알고리즘은 행위 모델의 모든 간선을 커버할 수 있다. 둘째, 행위 모델의 특정 간선들만 선택적으로 커버할 수 있다. 셋째, *n(n>=1)*번의 특정 간선 수행을 보장할 수 있다. 넷째, 최적의 수행 시간을 가지는 테스트 시퀀스를 생성한다.

4. 결론 및 향후 연구

본 연구에서는 입력 파라미터를 가진 컴포넌트 인터페이스의 최적 테스트 시퀀스 생성 방법을 제시하였다. 기존의 유한 상태 머신 기반의 테스트 시퀀스 생성 방법들은 컴포넌트 인터페이스의 데이터를 고려한 행위를 모델링할 수 없으며, 모든 간선을 적어도 한 번 이상의 수행만 보장하였다. 따라서 파라미터를 고려한 인터페이스 테스트 시퀀스를 생성할 수 없다.

제시된 최적 테스트 시퀀스 생성 기법은 인터페이스 행위 모델의 특정 간선을 테스터가 원하는 회수의 수행을 보장할 수 있다. 생성된 테스트 시퀀스는 최적화된 테스트 수행 시간을 가지기 때문에 테스트 자동화의 효율을 향상시킬 수 있다. 테스팅 수행 시에 관찰성을 제공하기 위해 메모리 덤프 함수가 제안되었으며, 최적의 테스트 시퀀스 생성 시에 메모리 덤프에 소요되는 시간도 고려하였다.

앞으로 최적 테스트 시퀀스 생성 알고리즘의 성능 평가를 위한 실험이 필요하며, 파라미터를 고려한 인터페이스 행위 모델에 입력 파라미터 변수의 제약 사항과 상태 변수의 제약사항을 함께 모델링하여 항상 수행 가능한 테스트 시퀀스 생성 기법 연구가 필요하다.

참 고 문 헌

[1] Toyota Motor Sales, U.S.A, Inc., http://www.toyota.com/recall/abs.html, February, 2010.

[2] Michael Zhivich and Robert K. Cunningham, “The Real Cost of Software Errors,” IEEE Security & Privacy, Vol.7, pp.87-90, March, 2009.

[3] Patrick Copeland, “Google's Innovation Factory: Testing, Culture, and Infrastructure,” Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation, 2010.

[4] Kent Beck, Test-Driven Development By Example, Addison-Wesley, 2003.

[5] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen and Juhani Warsta, Agile software development methods, VTT Publications, 2002.

[6] Alan Page, Ken Johnston, and Bj Rollison, How We Test Software at Microsoft, Microsoft Press, 2008.

[7] OMG Unified Modeling Language Superstructure Version 2.2, February, 2009.

[8] Stephen H. Edwards, “Black-Box Testing Using Flowgraphs: An Experimental Assessment of Effectiveness and Automation Potential,” Software Testing, Verification and Reliability, Vol.10, No.4, pp.249-262, December, 2000.

[9] S. Mahmood, R. Lai, and Y.S. Kim, “Survey of component-based software development,” IET Software, Vol.1, pp.57-66, April, 2007.

[10] R. Lai, “A survey of communication protocol testing,” The Journal of Systems and Software, Vol.62, pp.21-46, May, 2002.

[11] Rob M. Hierons, and Hasan Ural, “Optimizing the Length of Checking Sequences,” IEEE Transactions on Computers, Vol.55, pp.618-629, May, 2006.

[12] Boris Beizer, Black-Box Testing: Techniques for Functional Testing of Software and Systems, John Wiley & Sons, Inc., 1995.

[13] Gregor V. Bochmann and Alexandre Petrenko, “Protocol Testing: Review of Methods and Relevance for Software Testing,” Proceedings of ISSTA, pp.109-124, 1994.

[14] Chul Kim and J. S. Song, “Test Sequence Generation Methods for Protocol Conformance Testing,” Proceedings of COMPSAC, pp.169-174, November, 1994.
[15] Yanping Chen, Robert L. Probert, and Hasan Ural, “Model-based Regression Test Suite Generation Using Dependence Analysis,” Proceedings of AMOST, pp.54-62, 2007.
[16] Luca de Alfaro and Thomas A. Henzinger, “Interface-based Design,” Engineering Theories of Software-Intensive Systems, NATO Science Series, Vol.195, pp.83-104, 2005.
[17] Supaporn Kansomkeat and Wanchai Rivepiboon, “Automated-Generating Test Case Using UML Statechart Diagrams,” Proceedings of SAICSIT, pp.296-300, 2003.
[18] Sylvain Halle, Tevfik Bultan, Graham Hughes, Muath Alkhalaf, and Roger Villemaire, “Runtime Verification of Web Service Interface Contracts,” Vol.43, pp.59-66, March, 2010.
[19] Muzammil Shahbaz, Keqin Li, and Roland Groz, “Learning Parameterized State Machine Model for Integration Testing,” 31st Annual International Computer Software and Applications Conference, Vol.2, pp.755-760, July, 2007.
[20] Victor R. Basili, and Richard W. Selby, “Comparing the Effectiveness of Software Testing Strategies,” IEEE Transactions on Software Engineering, Vol.SE-13, pp.1278-1296, 1987.
[21] Stuart C. Reid, “An Empirical Analysis of Equivalence Partitioning, Boundary Value Analysis and Random Testing,” Proceedings of Fourth International Software Metrics Symposium, pp.64-73, November, 1997.
[22] Deepinder P. Sidhu, and Ting-Kau Leung, “Formal Methods for Protocol Testing: A Detailed Study,” IEEE Transactions on Software Engineering, pp.413-426, April, 1989.
[23] Krishan Sabnani and Anton Dahbura, “A protocol test generation procedure,” Computer Networks and ISDN Systems, Vol.15, pp.285-297, September, 1988.
[24] Hasan Ural and Keqin Zhu, “Optimal Length Test Sequence Generation Using Distinguishing Sequences,” IEEE Transactions on Networking, Vol.1. pp.358-371, 1993.
[25] Raymond E. Miller and Sanjoy Paul, “On the Generation of Minimal-Length Conformance Tests for Communication Protocols,” IEEE/ACM Transactions on Networking, Vol.1, pp.116-129, February, 1993.

저 자 약 력

신 영 술

이메일 : youngsulshin@gmail.com

- 2005년 경북대학교 전자전기컴퓨터학부(학사)
- 2007년 경북대학교 컴퓨터과학과(이학석사)
- 2007년~현재 경북대학교 IT대학 컴퓨터학부 박사과정
- 관심분야 : 임베디드 소프트웨어 모델링 및 분석, 소프트웨어 테스팅, 시뮬레이션 등

이 우 진

이메일 : woojin@knu.ac.kr

- 1992년 경북대학교 컴퓨터과학과(학사)
- 1994년 한국과학기술원 전산학과(공학석사)
- 1999년 한국과학기술원 전산학과(공학박사)
- 1999년~2002년 한국전자통신연구원 S/W공학연구부 선임연구원
- 2002년~현재 경북대학교 IT대학 컴퓨터학부 부교수
- 관심분야 : 임베디드 실시간 시스템 모델링 및 분석, 요구 공학, Petri nets, 컴포넌트 개발 기술 등