UML state chart 를 이용한 flow graph 기반 테스트 케이스 생성 방법

박현상*, 최경희*, 정기현**
*아주대학교 정보 통신전문대학원 정보통신공학과
**아주대학교 전자공학부
e-mail : elvaimay@ajou.ac.kr

Test case generation method based on flow graph using UML state chart

HyunSang Park*, KyungHee Choi*, KiHyun Jung**
*Graduate School of Information and Communication, Ajou University
**Division of Electronics Engineering, Ajou University

요 약

소프트웨어 테스팅은 소프트웨어의 개발 과정에 있어서 가장 중요하고 많은 비용이 드는 부분이다. 소프트웨어 테스팅을 수동으로 행하는 것은 많은 문제를 발생시킬 수 있다. 소프트웨어 자동 테스팅을 하기 위해서 최근 활발히 연구되고 있는 부분이 모델 기반 소프트웨어 자동 테스팅 기법이다. 본 논문에서는 UML 모델 기반 테스트 케이스 자동 생성 기법을 제안한다. UML state chart 로 모델링 된 테스트 대상 소프트웨어를 제안된 자료구조에 저장 한 후, 이를 flow graph 로 변환한다. 최종적으로 변환된 flow graph 에서 테스트 케이스를 생성한다.

1. 서론

최근 임베디드 시스템이 널리 보급되면서, 임베디드 시스템의 신뢰성에 대한 중요성이 대두되고 있다. 특히 임베디드 시스템에 내장되어 있는 소프트웨어에 대한 블랙 박스 테스트 기법의 연구가 활발히 진행되고 있다.

블랙 박스 소프트웨어 테스트 기법 중에서 소프트웨어 명세 기반의 소프트웨어 테스트 기법이 활발히 연구되고 있다. 소프트웨어의 명세를 작성하기 위해서 많이 사용 되는 것이 UML 이다.

Unified Modeling Language(UML)[7]은 소프트웨어를 모델링을 하기 위해 널리 쓰이는 모델링 언어이다. UML 을 이용하여 소프트웨어의 여러 측면을 모델링 할 수 있다. 소프트웨어의 상태, 통신, 작업 흐름을 UML 에서 제공하는 그래픽 표기 방법을 이용하여 UML 다이아그램으로 모델링 할 수 있다.

UML 은 소프트웨어의 명세를 표현 할 수 있는 강력한 도구인 만큼, UML 을 이용한 소프트웨어 모델을 통하여 테스트 케이스를 생성하는 기법에 대한 연구 또한 많이 이루어지고 있다. 그 중에서도 UML state chart 는 소프트웨어의 동작 기간 동안의 행동을 자세하게 모델링 할 수 있으므로 테스트 케이스 생성 기법도 UML state chart 를 기반으로 한 연구가 많이 이루어지고 있다.

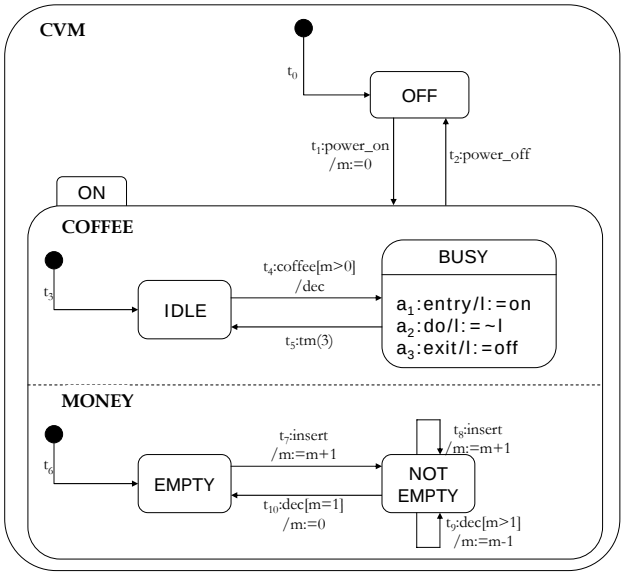
UML state chart 를 이용한 테스트 케이스 생성 방법은 크게 state chart 를 그대로 이용하여 생성하는 방법과, state chart 를 변형한 형태에서 테스트 케이스를 생성하는 방법 두 가지 분류로 나뉘어 진다. [1]은 state chart 를 그대로 이용하여 테스트 케이스를 생성하지만 state chart 자체의 복잡도가 높기 때문에 enabled transition 과 change event 만을 테스트 케이스 생성 대상으로 하는 한계가 있다. [2,3]은 state chart 를 각각 EFSM 과 IOLTS 로 변환한다. 하지만 이 방법에는 계층적인 UML state chart 의 구조 때문에 state 가 폭발하는 단점을 가진다.

[4]에서는 state chart 를 test flow graph 로 변환하여 테스트 케이스를 생성한다. 이 방법은 기존의 연구에서 발생되었던 문제점을 어느 정도 해소할 수 있다.

그러나 이 연구에서는 반복 구조나 동시 실행에 대한 테스트 케이스 생성에 관한 고려를 하고 있지 않다.

이 논문에서는 [4]에서 제시한 테스트 케이스 생성 기법을 확장하여, 소프트웨어의 흐름에 반복 구조, 동시 실행이 포함되어 있을 경우를 고려한 UML state chart 자료구조를 설계하고, 그에 따른 테스트 케이스 생성 기법을 제안한다.

이 논문에서는 [5]에서 사용한 그림 1 의 음료수자판기 UML state chart 를 예제로 테스트 케이스 생성 기법을 설명한다.



(그림 1) 음료수자판기의 UML state chart

2. State chart Diagrams DS

UML state chart 를 이용하여 테스트 케이스를 생성하기 위해서 UML state chart 의 내용을 자유롭게 표현할 수 있는 자료구조를 설계한다.

2.1 Diagram table

Diagram table 은 state chart 의 각 state diagram 들의 정보를 표현한다.

Diagram Table

ID	Name	Lv
0	D0	0
1	D1	1
2	D2	1
..	..	..

(그림 2) Diagram table

- ID(Identification) : Table 내에서의 각 diagram 에 대한 identification
- Name : table 의 이름
- Lv(Level) : Diagram 이 hierarchical 하게 어느 정도 깊이 정의되어 있는지를 나타내는 값.

2.2 State table

State table 은 state chart 의 각 state 들의 정보를 표현한다.

State Table

ID	Name	OT	UD	RD	Type	Entry	Exit	Do
0	S0	0	0	-	Init.	-	-	-
1	S1	1,2	0	1,2	Comp.	-	-	-
2	S2	3,4	1	-	Norm.	-	-	-
3	S3	5	1	-	Norm.	-	-	-

(그림 3) State table

- ID(Identification) : Table 내에서의 각 state 에 대한 identification.
- Name : State 의 name. Pseudo state 의 경우 생략 가능하다.
- OT(Output Transitions) : State 의 output transition 을 나타내는 것으로 transition table 의 id 를 가리킨다. (복수가능)
- UD (UsedIn) : state 가 사용된 diagram 의 id 를 가리킨다. (복수불가)
- RD (RedefinedAs) : Compound state 가 새로운 diagram 을 형성 할 때, 정의한다. (복수가능)
- Type : State 의 종류를 나타낸다. 일반적인 경우는 'Norm.'이고, compound state 의 경우는 'Comp.', pseudo state 의 경우는 각각의 경우에 따라 'Init', 'Junc', 'Final', 'Fork', 'Join'등으로 나타낼 수 있다.
- Entry : state 의 entry action 의 action table id
- Exit : state 의 exit action 의 action table id
- Do : state 의 do action 의 action table id

2.3 Transition table

Transition table 은 state chart 의 각 transition 들의 정보를 표현한다.

Transition Table

ID	SS	TS	Ev	Gd	Act
0	0	1	-	-	-
1	1	2	0	-	-
2	1	5	1	-	-
3	2	3	2	-	-
4	2	4	3	-	-
5	3	5	4	-	-

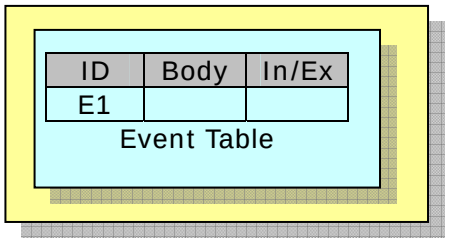
(그림 4) Transition table

- ID(Identification) : Table 내에서의 각 transition 에 대한 identification.
- SS(Source State) : transition 의 source state
- TS(Target State) : transition 의 target state

- Ev(Event) : transition 의 triggering event 로서, event table 의 id. (복수 불가)
- Gd(Guard) : transition 의 guard condition 으로서, guard table 의 id. (복수 불가)
- Act(Action) : transition 의 action 으로서, action table 의 id. (복수 가능)

2.4 Event table

Event table 은 state chart 의 transition 을 발생시키는 event 들에 대한 정보를 포함한다.

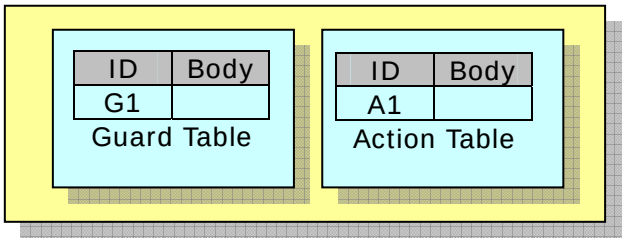


(그림 5) Event table

- ID(Identification) : Table 내에서의 각 item 에 대한 identification.
- Body : 각 item 의 상세 내용
- In/Ex : Event 가 내부에서 발생되는 event 인지(In), 외부에서 발생하는 event 인지(Ex)를 구분하여 나타낸다. (Modeling 시에 내부/외부를 구분할 수 있도록 구분자를 집어 넣는다.)
- Internal Event : 내부에서 발생되는 event 로서, 다른 action 에 의해서 trigger 되는 event 를 말한다.
- External Event : 외부에서 발생되는 event 로서, timer 나 사용자의 조작에 의해서 발생하는 event 를 말한다. Black-box testing 에서의 test case 는 external event 의 조합이다.

2.5 Guard and action table

Guard table 은 state chart 의 transition 을 발생시키는 조건인 guard 에 대한 정보를 나타낸다. Action table 은 state 의 entry, exit, do action 과 transition 의 action 에 대한 action 정보를 나타낸다.



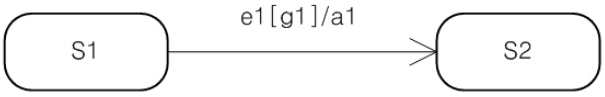
(그림 6) Guard and action table

- ID(Identification) : Table 내에서의 각 item 에 대한 identification.
- Body : 각 item 의 상세 내용

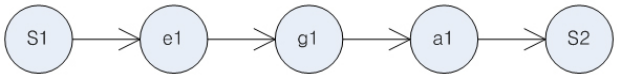
3. 테스트 케이스 생성을 위한 flow graph 구축

2 장에서 구축한 UML state chart 자료구조를 바탕으로 테스트 케이스 생성을 위한 flow graph 를 구축한다. Flow graph 는 다음과 같은 순서로 생성한다.

1) UML state chart 다이아그램 별로 flow graph 를 생성한다.
2) 각 diagram 의 initial state 로부터 시작하여, state, event, guard, action 순서로 flow graph 를 생성한다. 예를 들어 그림 6 과 같은 state chart 는 그림 7 과 같은 flow graph 로 다음과 같이 변환된다.



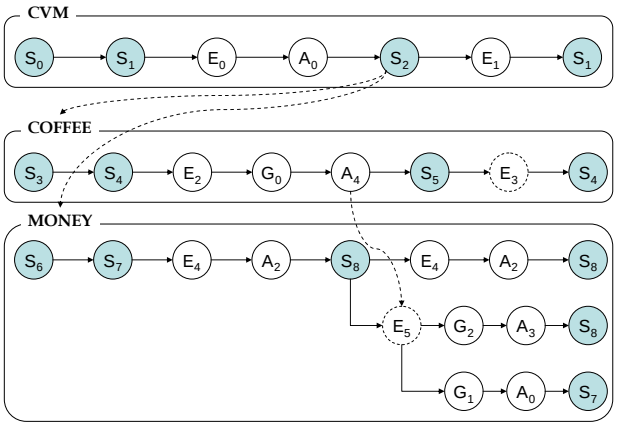
(그림 7) 예제 state chart



(그림 8) 변환된 flow graph

3) State 나 action 에서 다른 event 를 발생시킬 경우 관계를 점선 화살표로 표현한다.
4) 다이아그램의 마지막 state 에 도달할 때까지 2 번, 3 번 과정을 반복한다.

그림 1 의 state chart 를 flow graph 로 변경하면 그림 9 와 같이 된다.



(그림 9) 음료수 자판기의 flow graph

4. Flow graph 기반 테스트 케이스 생성

3 장에서 생성한 flow graph 를 바탕으로 테스트케이스를 생성한다. 테스트 케이스는 다음과 같은 순서로 생성한다.

1) 각 flow graph 의 다이아그램 별로 테스트케이스를 생성한다.
2) 생성된 테스트 케이스에서 반복 구조로 대체할 수 있는 부분을 대체한다. Flow graph 에서 순회하면서 이미 찾았던 반복 구조와 동일한 부분

이 있다면 대체한다. 찾아진 테스트 케이스에서 반복 구조의 시작 state 가 있다면, 그 부분을 다시 반복구조로 변환한다.

3) Non-deterministic Loop(L_0, L_2, L_4)는 0 회, 1 회, 2 회, Max, Max±1 만큼 반복하고, Max 값은 default 값으로 10 으로 설정하고, 사용자가 수정할 수 있다. [6]

4) 일단 Non-deterministic loop 를 포함한 test case 에 대하여 0 회, 1 회, 2 회 반복 test case 만 생성되었다고 가정하자. 0 회일경우는 Loop 는 start state 로 대체되고, 1 회나 2 회일 경우는 Loop 의 sequence 전체를 반복한다.

5) 우선적으로 Nested Loop 를 찾아서 Loop 키워드를 이용하여 0 회, 1 회, 2 회 반복 test case 로 update 한다.

6) Nested Loop 를 이미 구한 loop 의 sequence 로 update 한다.

7) 이제 Non-deterministic loop 를 찾아 update 하면, 생성된 test case 는 다음과 같다.

8) 중복된 내용을 제거한다. 다음과 같은 경우에 중복을 제거한다.

- Loop 의 start state 와 Loop 가 연속해서 나온 경우.
- 동일한 Loop 가 연속해서 나오는 경우

9) Diagram 의 계층적 관계를 고려하여 test case 를 update 한다.

10) 9 단계에서 까지 생성된 모든 테스트 케이스에서 event.만을 모아 최종 테스트 케이스를 생성한다.

위의 과정을 통해 그림 9 의 flow graph 로부터 생성된 테스트케이스는 다음과 같다.

① NULL

② E0 → E4 → E1

③E0 → E4 → E4 → E1

④E0 → E4 → E4 → E4 → E1

~~E0 → E4 → E1~~ (②와 중복이므로 제거한다.)

⑤E0 → E1

⑥ E0 → E4 → E2 → E3 → E1

⑦ E0 → E4 → E4 → E2 → E3 → E2 → E3 → E1

⑧ E0 → E4 → E4 → E4 → E2 → E3 → E2 → E3 → E2 → E3 → E1

<중략>

⑨ E0 → E4 → E4 → E4 → E4 → E4 → E2 → E3 → E2 → E3 → E2 → E3 → E2 → E3 → E2 → E3 → E1

⑩ E0 → E4 → E4 → E4 → E2 → E3 → E4 → E4 → E2 → E3 → E2 → E3 → E2 → E3 → E2 → E3 → E1

⑪ E0 → E4 → E4 → E4 → E2 → E3 → E2 → E3 → E4 → E4 → E2 → E3 → E2 → E3 → E2 → E3 → E1

<중략>

⑫ E0 → E4 → E1 → E0 → E4 → E1

<후략>

5. 결론 및 차후 연구 계획

이 논문에서는 UML state chart 다이아그램을 이용한 테스트케이스 생성 기법을 제안하였다. 제안된 테스트 케이스 생성 기법을 이용하여 테스트케이스를 자동으로 생성할 수 있다. 생성된 테스트 케이스는 구축된 flow graph 의 state 와 transition coverage 를 만족하며, 부분적으로 반복 구조 coverage 도 만족한다.

추후로 연구되어야 할 부분은 테스트 케이스의 유효성 여부이다. 테스트 케이스를 자동 생성하는 방법의 연구와, 생성된 테스트 케이스의 실행 가능 여부에 대한 연구는 다르게 이루어져야 한다. 특히 반복구조가 포함되어 있는 테스트 케이스는 소프트웨어의 내부 조건에 따라 실행 가능 여부가 판단되기 때문에 집중적으로 연구가 필요하다.

참고문헌

[1] Offutt, J., and Abdurazik, A. 1999. Generating test cases from UML specifications. In proceeding of the 2nd International Conference on the Unified Modeling Language(UML99)

[2] Kim, Y.G., Hing, H.S., Cho, S.M., Bae, D.H., and Cha, S.D. 1999. Test cases generation from UML state diagrams. In IEEE Software

[3] Stefania Gnesi, Diego Latella and Mieke Massink. 2004. Formal Test-case Generation for UML Statecharts. Proceedings of the Ninth IEEE International Conference on Engineering Complex Computer

[4] Supaporn Kansomkeat and Wanchai Rivepiboon. 2003. Automated-Generating Test Case Using UML Statechart Diagrams. In proceedings of SAICSIT

[5] Fabrice Ambert, Fabrice Bouquet, Bruno Legeard, Fabien Peureux, Laurent Py and Eric Torrebore. Automated test case and test driver generation for embedded software.

[6] Boris Beizer. 1995. Black-Box Testing Techniques for

Functional Testing of Software and Systems. Wiley.

[7] Booch, G., Rumbaugh, J., and Jacobson, I. 1998. The Unified Modeling Language User Guide. Addison Wesley