제 26 회 한국정보처리학회 추계학술발표대회 논문집 제 13 권 제 2 호 (2006. 11)

Rhapsody Statecharts 의 정형검증을 위한 변환 알고리즘 연구

황대연*, 박승현*, 이나영**, 김윤구*** 최진영*
*고려대학교 컴퓨터학과
**서울대학교 원자핵공학과
***삼창기업㈜
e-mail : dyhwang@formal.korea.ac.kr

A study on Rhapsody statecharts transformation for formal verification

Dae Yon Hwang*, SeungHyun Park*, Na-Young Lee**, Yun Goo Kim***, Jin-Young Choi*
*Dept. of Computer Science and Engineering, Korea University
**Dept. of Nuclear Engineering, Seoul National University
***Samchang Enterprise Co. Ltd,.

요 약

Statechart 는 상태 기반의 시각적 명세 언어로 UML 에 포함되어 매우 널리 사용되고 있다. 하지만 UML 의 특성에 의해 그 의미론이 비정형적이 되었으며, 시스템의 행위를 불안하게 만드는 비결정적인 경우가 발생하게 된다. 본 논문에서는 UML 지원 도구인 Rhapsody 의 statechart diagram 중 코어 부분을 정형적 의미론을 가지는 명세로 변환함으로써 비결정적 명세를 찾아낼 수 있음을 보였다.

1. 서론

David Harel 에 의해 만들어진 statechart[2, 3]는 상태 기반의 시각적 명세 언어이다. 시각적 명세 언어이기 때문에 텍스트 기반의 명세 언어에 비해서 쉽게 이해하고 작성할 수 있다는 장점을 가진다. 1990 년대 중반에 UML[6]에 포함되면서 현재는 자동차, 항공, 원자력 등등 많은 안전성 시스템들의 명세 언어로 널리 사용되고 있다. 시간이 지나면서 현재에는 다양한 버전의 statechart 가 존재한다. 그리고 각 버전은 조금씩 다른 의미론을 가진다. 현재는 똑같이 UML 에 기반을 두는 도구간에도 의미론의 차이를 보이고 있으며, UML 에 포함된 statechart 자체가 정형적인 의미론을 가지고 있지 않다.

Rhapsody[5]는 I-logix 사의 UML 지원 도구로 강력한 코드 생성 능력과 시뮬레이션 기능을 가지는 UML 도구이다. 하지만 UML statechart 를 기반으로 하기 있기 때문에 정형적 의미론을 가지고 있지 않다. 정형화된 의미론을 가지고 있지 않은 경우 명세상 여러 비결정성이 발생하며, Rhapsody 자체에서 제공하는 시뮬레이션 및 테스팅 방법으로는 이러한 비결정성을 쉽게 찾을 수가 없다.

UML 에 기반을 둔 도구들이 정확한 의미론[1]이 없기 때문에 검증이 어려운 반면, STATEMATE[4]는 매우 정형적인 의미론을 가지는 statechart 도구이다. 때문에 정확하게 비결정성을 찾아 알려주며 다양한 검증 기능을 제공해 준다.

본 논문에서는 두 명세간의 차이점을 간단히 살펴보고, statechart 의 핵심 코어 부분을 변환하는 알고리즘을 제시하며, 변환 후 검증 가능한 비결정성 예제들을 살펴본다.

논문은 다음과 같이 구성되어 있다. 2 장은 관련 연구로 statechart 에 대한 간략한 소개와 UML 명세를 정형적인 명세로 변환한 연구들을 소개하며, 3 장은 Rhapsody 와 STATEMATE 의 대표적 차이와 그 변환 알고리즘, 그리고 검증할 비결정성들에 대한 예제를 보여준다. 4 장은 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

2.1 Statechart

Statechart 는 David Harel 에 의해 만들어졌으며, 상태 기계(state machine) 기반의 시각적 명세 언어이다. 상태 기계를 기반으로 한다는 것은 시스템의 현재 상황을 나타내는 상태들을 설정하고, 그러한 상태들 간을 이동하는 조건과 이동시의 동작을 표현하는 전이를 이용하여 시스템의 행위를 명세 한다는 것이다.

Statechart 는 이러한 일반적인 상태 기계에 상태의 추상화를 위한 계층구조, 동시 진행을 나타낼 수 있는 병렬 구조, 그리고 이벤트의 broadcasting 등등을 추가하여서 때문에 보다 실제 시스템을 명세하면서도 정형적인 의미론을 가지고 있어, 의미의 정확한 전달과 검증을 가능하게 만들었다. 현재는 UML 에 포함되어 객체 지향적인 의미론을 가지게 되어 널리 사용되기도 하지만, I-logix 사의 STATEMATE 와 같은 statechart 전문 도구로 남아 높은 안정성을 요구하는 시스템 등의 명세에도 널리 사용되고 있다.

2.2 UML 의 정형적 명세로의 변환 연구

UML 은 많은 사람들이 사용하는 강력한 개발 명세 언어로 UML 명세를 보다 정형적인 의미를 가지는 언어로 변환하고자 하는 연구들이 진행되었다. 프랑스의 Omega 프로젝트[8, 9]의 경우 Rhapsody 로 만들어진 시스템 명세를 정형 검증 도구 VIS 에 맞는 입력으로 변환하여 검증이 가능하도록 하였다. 이들은 Rhapsody 명세를 XMI 로 변환한 후 몇 번의 변환을 거쳐서 VIS 의 입력 언어로 변환하고, Sequence chart 보다 표현력이 풍부한 Live Sequence Charts 로 속성을 명세하여 정형적으로 검증을 하는 모델 체킹을 실행하였다.

이외에도 UML 의 statechart 에 정형적인 의미를 부여하거나 다른 정형 명세 언어로 변환하여 검증을 하기 위한 많은 연구가 진행되고 있다.

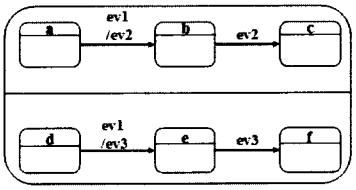
3. 변환 및 검증

3.1 변환을 통해 검증하고자 하는 특성

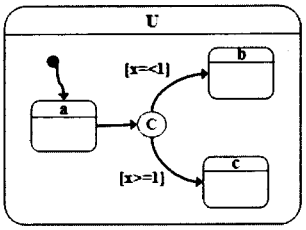
Rhapsody 의 statechart 는 정형적 의미론을 가지고 있지 않기 때문에, (그림 1)이나 (그림 2)와 같은 비결정성을 가지는 모델을 명세하게 되는 경우가 발생한다. (그림 1)의 경우 명세상에서는 상태 b 에서 상태 c 로 가는 전이와 상태 e 에서 상태 f 로 가는 전이가 동시에 발생해야 하지만, Rhapsody 가 동시에 두 개 이상의 이벤트를 처리 하지 못하는 이벤트 큐를 사용하기 때문에 두 전이 간에 명세 되지 않은 비결정적 순서가 발생하게 된다. 이때 두 전이의 동작이 상호간에 영향을 주는 변수의 할당인 경우 그 결과를 예측할 수 없게 된다.

(그림 2)의 경우, condition connector 의 분기 부분에 둘 이상의 분기 조건이 동시에 만족되는 경우가 존재하고 있다. 두 개 이상의 분기의 조건이 만족된 경우 어떤 전이가 실제로 일어날지에 대한 비결정성이 발생하게 된다.

이러한 비결정성은 프로그램의 행위를 예측할 수 없게 만들며, 개발자가 생각지 못한 방향으로 시스템이 동작하게 만들기, 때문에 명세상에서의 비결정성은 안정적인 프로그램을 개발하는데 매우 위험한 요소이다.



(그림 1) 순서의 비결정성



(그림 2) 선택의 비결정성

Rhapsody 의 경우 이러한 비결정성을 쉽게 찾을 수 있는 방법을 제공하지 못하는 단점이 있다. 이것을 해결하기 위해 정확한 의미론을 가지는 STATEMATE Statechart 명세로 변환하여 이러한 비결정적 요소들을 찾고, 또 검증이 필요한 특성을 정형적으로 검증함으로써 보다 안전한 모델을 만들 수 있다.

3.2 변환을 위한 제약 사항

Rhapsody 의 경우 실제 생성되는 코드의 활용에 많은 비중을 두고 있기 때문에 전이의 기본이 되는 E[G]/A 의 표현성에 보다 다양한 표현을 허용하였다. 전이를 일으키는 트리거 이벤트에 함수의 사용이 가능하며, 이벤트가 값을 전달하기도 하고, 전이 동작에도 역시 함수를 사용할 수 있도록 변경되었다. 이러한 추가적인 특징으로 인해 정확하게 의미가 일치하는 변환은 사실상 불가능하다. 검증하고자 하는 내용을 포함하는 핵심적인 부분을 정하고, 그에 해당하는 부분만을 변환하고자 한다. 이 핵심 부분을 코어 모델이라고 하자. 즉, 일정한 제약이 있는 모델만이 변환이 가능한다, 검증을 위해서는 이러한 제약 조건을 지키는 코어 모델만이 변환이 가능하다.

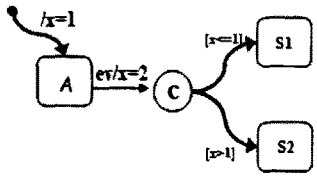
코어 모델 문법의 기본적인 모습은 서로가 같으나 의미적으로 Rhapsody 와 STATEMATE 는 다른 점을 가지고 있다. 이러한 차이점은 변환해야 할 부분과 검증해야 할 부분으로 나뉜다.

3.3 두 명세의 차이와 변환 알고리즘

다음은 설정한 코어 모델에서 두 명세가 의미적으로 차이가 나는 부분의 예와 그에 대한 변환 알고리즘이다.

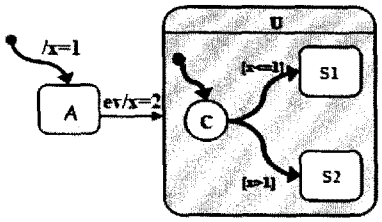
- 동적 선택

전이의 동작이 같은 스텝 내에서 영향을 발휘하여 분기에 영향을 주는 것을 동적 선택이라고 한다. (그림 3)에서 A 상태에서 출발하여 S1 또는 S2 의 상태가 되는 것이 하나의 스텝이다. 이때 트리거 이벤트 ev 에 의해 일어난 전이 동작 x=2 이 바로 상태 선택점(condition connector)에게 영향을 주어 S2 상태로 이동을 하게 되면 동적 선택이 일어났다고 한다. 반대로 선택점의 분기 판단을 동작과 상관없이 스텝 이전의 값을 기준으로 하게 되면 정적 선택이 일어났다고 하게 된다.



(그림 3) Condition connector

STATEMATE 의 경우에는 이중 버퍼링을 하는 특성 때문에 항상 상태가 완료되어야만 동작들에 의한 변수 값들이 업데이트되는 정적 선택만을 지원한다. Rhapsody 의 경우에도 일반적으로 정적 선택이 되지만 다음의 (그림 4)처럼 모델링을 하면 스텝이 완료되는 시점이 S1 또는 S2 에 도착하는 시점 임에도 불구하고 중간에 값이 업데이트되어 S2 로 가는 동적 선택이 일어난다.

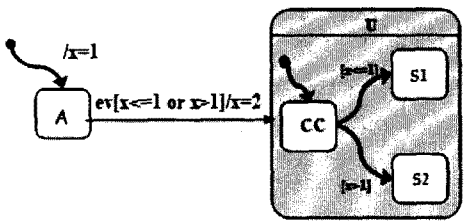


(그림 4) Rhapsody 의 동적 선택

따라서 Rhapsody 에서 (그림 4)과 같은 명세가 있는 경우 STATEMATE 에서도 동적 선택이 일어나게 변환을 해주어야 한다. 그 변환 알고리즘은 다음과 같다.

```
transDC{
for all s ∈ STATE
if default_t_target(s) = condition_connector {
    cc2state():
    if (!out_trans_has_else(cc){
        for all t ∈ transition
        if target(t) = s
            add_ccguard(t);
}}
```

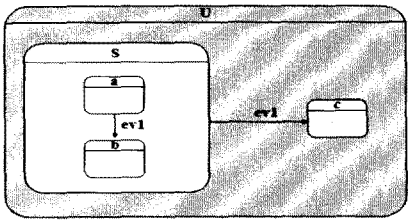
transDC 함수는 statechart 의 모든 상태들을 검사하여 하위상태가 존재하는 경우, 기본 전이(default transition)가 선택점으로 연결되는지 검사한다. 기본 전이가 선택점으로 연결되어 있는 경우 선택점을 상태로 바꾼다. 이 때 발생할 수 있는 문제가 선택 분기가 모든 경우에 대한 선택이 없을 경우 실제 전이가 일어나지 않아야 함에도 선택점이 상태로 전환되었기 때문에 A 상태로 돌아가지 않는 경우가 생긴다. 따라서 선택 분기 중에 나머지 경우를 나타내는 [else]가 없으면 선택이 가능한 경우를 나타내는 선택 분기 가드들을 이벤트의 가드에 포함 시켜야 한다 (그림 5).



(그림 5) 분기 가드의 추가

-우선 순위

두 전이가 동시에 가능한 경우, STATEMATE 에서는 전이들의 범위(scope)를 기준으로 우선순위를 판단하는 것과 대조적으로 Rhapsody 는 전이의 시작 상태의 계층을 기준으로 판단하게 된다. 즉, (그림 6)의 경우 ev1 이 발생하면 STATEMATE 에서는 c 상태로, Rhapsody 에서는 b 상태로 전이하게 된다.



(그림 6) 우선 순위

이때 (그림 6)와 같이 서로 같은 이벤트가 아니라 다르지만 동시에 발생하는 이벤트의 경우도 생각해야 하기 때문에 하위의 statechart 에서 전이가 발생하는 경우에는 상위의 전이가 일어나지 못하도록 가드 조건을 추가해 준다.

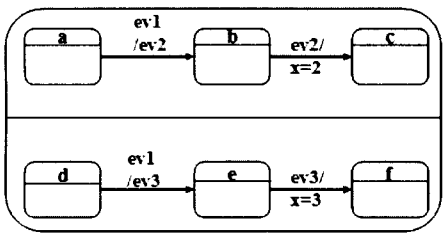
```
priority() {
for all t ∈T
if source(t) has sub-chart{
    add2guard_all_subtransition_condition(t);
```

3.4 변환된 명세의 검증

위의 두 가지 예제는 두 명세간의 차이로, 변환에서 고려해야 부분이다. 다음의 예제는 Rhapsody 명세가 비 정형적이기 때문에 일어나는 비결정성으로 변환 후 검증 할 부분들이다.

-분기 선택의 비결정성
(그림 2)의 경우 선택점의 분기 가드간에 겹치는 영역이 있어서 x=1 인 경우에 명세상에서 어디로 가야 할지를 정확하게 알 수 없다. 이것은 명세자의 실수이지만 Rhapsody 는 이러한 분기를 if~else~ 문의 구조로 처리하기 때문에 명세에는 나와 있지 않은 어떤 분기로 임의로 가게 된다. Rhapsody 에서는 이런 명세상의 오류를 직접 찾는 방법 말고는 알 수 있는 방법이 없다. 반면에 이 명세를 그대로 STATEMATE 로 옮겨서 검증할 경우 STATEMATE 의 검증 도구는 비결정성 오류라고 지적해준다.

-전이 동작간의 순서의 비결정성
(그림 7)의 명세상의 의미는 ev1 이 발생할때 상하의 두 병렬적으로 동작하는 statechart 가 동시에 이동을 하여 c 와 f 에 도착하게 된다. Rhapsody 에서는 이벤트를 큐에 넣어서 차례로 하나씩 처리하기 때문에 아래와 같이 특별히 순서에 대한 언급이 없이 명세가 된 경우 둘 간에 의존성이 있다면 한쪽 전이가 먼저 일어나게 되어 결과에 대한 확신을 할 수 없게 된다.



(그림 7) 순서에 영향을 받는 명세

즉, 전이 후 x 의 값이 2 가 될지 3 이 될지 알 수 없게 되는 것이다. 위의 예제에서는 개발자가 눈으로 쉽게 확인이 가능하지만 두 상태가 떨어져 명세 되고 동시에 같은 변수에 대한 접근을 하게 되면 시스템 설계자가 원하지 않는 결과값을 발생 시킬 수 있다.

4. 결론 및 향후 연구

UML 은 이미 많은 영역에서 널리 사용되고 있는 명세언어이다. UML 을 구현한 도구들이 많은 가운데 Rhapsody 는 실제 코드의 구현과 시뮬레이션 기능 등으로 개발자들을 지원해주는 도구이다. 그러나 UML 의 statechart 자체가 정형적인 의미론을 가지고 있지 않기 때문에 명세상의 오류를 검증하는 것이 쉽지 않다.
대조적으로 STATEMATE 는 약간 다른 의미론을 가지는 statechart 이며 정형적인 의미론을 가지고 있어서 모델의 정형 검증을 지원한다.
본 논문에서는 Rhapsody 의 statechart 를 STATEMATE 에서 사용하는 의미론을 가지는 명세로 변환함으로써 Rhapsody 에서 개발자가 실수할 수 있는 명세의 오류를 쉽게 찾아낼 수 있음을 보였다.
향후에는 현재의 코어를 부분을 조금 더 확장하여 원하는 속성에 대한 검증을 할 수 있도록 할 예정이며, 동시에 만들어낸 알고리즘을 지원하는 자동화된 도구를 개발하여 보다 편리하게 변환과 검증을 지원할 예정이다.

참고문헌

[1] Michael von der Beeck, "A Structured operational semantics for UML-statecharts", Software and Systems

[2] David Harel, "On Visual formalisms", Communications of the ACM, volume 31, number 5, pp. 514-530, 1988

[3] David Harel, "Statecharts: A Visual formalism for Complex Systems", Science of Computer Programming, vol. 8, issue 3, pp. 231-274, 1987

[4] David Harel and Amnon Naamad, "The STATEMATE semantics of Statecharts", ACM Transactions on Software Engineering and Methodology (TOSEM), volume 5 , issue 4, pp. 293-333, 1996

[5] David Harel and Hillel Kugler, "The RHAPSODY semantics of Statecharts (or, On the Executable Core of the UML) - Preliminary Version", In Proceedings of the 3rd International Workshop on Integration of Software Specification Techniques for Applications in Engineering (INT 2004), LNCS, vol. 3147, pp. 325-354, Springer-Verlag, 2004

[6] OMG, Unified Modeling Language (UML) 2.0 Superstructure Specification, Document formal/05-07-04, The Object Management Group (OMG), 2005

[7] Michelle L. Crane and Juergen Dingel, "UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal", In Proceedings of ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2005), Lecture Notes in Computer Science, vol. 3713, pp. 97-112, Springer-Verlag, 2005

[8] http://www-omega.imag.fr/index.php

[9] Ingo Schinz, Tobe Toben, Christian Mrugalla, Bernd Westphal "The Rhapsody UML Verification Environment" , In Proceedings of the SEFM 2004 IEEE