Statechart 명세의 프로세스 알제브라로의 변환[1]

박명환*°, 김영미*, 김진현*, 강인혜**, 최진영*
고려대학교 컴퓨터학과*
숭실대학교 정보과학대학**

Translation of Statechart Specification to Process Algebra

Myung-Whan Park*, Young-Mi Kim*, Jean-Hyun Kim*, Inhye Kang**, Jin-Young Choi*
Department of Computer Science and Engineering, Korea University*
Graduate School of Information Science, Soongsil University**

요 약

본 논문에서는 정형명세 언어인 Statechart 의 의미론을 프로세스 알제브라로 설명하는 방법을 제시한다. 이렇게 함으로써 두개의 Statechart 명세간의 bisimulation 을 정의할 수 있게 된다.

1. 서론

컴퓨터 하드웨어, 소프트웨어의 기술 발전으로 그 이용도는 대형 과학계산으로부터 소형 가전제품에 이르기까지 다양해지고 있다. 특히, mission critical system 이나 safety critical system 분야에서의 컴퓨터는 다른 어떤 시스템에서보다 더욱더 중요한 역할을 수행하고 있다. 이런 시스템에서 발생한 오류는 수많은 인명이나 재산의 손실을 초래하기 때문에 시스템 설계나 구현에 완벽이 기해져야 한다. 정형기법(Formal Method)은 위와 같은 시스템의 설계와 분석에 수학적, 논리학적 기반을 제공하고 있다. 그러나, 대부분의 정형명세 언어는 사용하거나 이해하기 어려운 수학이나 논리의 notation 을 사용하기 때문에 전문적인 지식이 없는 사람은 사용하기가 곤란했다. 따라서 사용하기 쉬운 graphic 기반의 명세 언어의 필요성이 대두 되었는데 이런 배경에서 Statechart[1]와 Modechart, Petri-net 등의 graphic 기반 정형명세 언어가 탄생했다. 하지만, 이런 graphic 명세언어는 사용하거나 이해하기는 쉽지만 명확한 semantics 의 정의가 곤란하기 때문에 보통 다른 방법으로 semantics 를 정의하곤 했다[5]. 본 논문에서는 Statechart 의 semantics 를 ACSR[4]로 정의한다. ACSR 은 time 과 resource, priority 의 개념을 가진 process algebra 로서 bisimulation 과 여러 법칙들에 기반한, 잘 정의된 semantics 를 가진 정형명세 언어이다. 또한 ACSR 은 VERSA[6]라는 검증 tool 을 가지고 있기 때문에 Statechart 로 명세한 시스템을 수학적으로 검증할 수 있는 장점을 가질 수 있다.

이 논문의 구성은 2 장, 3 장에서 Statechart 와 ACSR 에 대해 간단히 소개하고 4 장에서 Statechart 명세를 ACSR 로 변환하는 규칙을 제시하고 5 장에서 결론을 맺도록 한다.

2. Statechart

Statechart 는 reactive system 의 명세에 적합한 graphic 명세 언어이다. 그 근본 구조는 State-transition diagram 에 hierarchy, concurrency, communication 개념을 도입한 것이다. Hierarchy 는 tree 에서 sub-tree 가 파생되듯이, 하나의 state 가 그 substate 를 가질 수 있게 하는 것으로 계층적 구조를 표현한다. Concurrency 는 state 들간에 병렬성을 제공하는 것인데, 어떤 시스템이 두개 또는 여러 개의 parallel component 가 모이서 하나의 state 를 구성하게 된다. Communication 은 system 의 component 들간에 broadcasting 을 통해 정보를 주고받고 시스템이 진행하는 것을 말한다.

3. ACSR (Algebra of Communicating Shared Resources)

ACSR 은 CCS(Calculus for Communicating System)에 기반한 process algebra 로서 시간, 자원, 우선순위, 동시성 등 실시간 시스템에 필요한 여러 개념을 포함하고 있다. 또한 ACSR 은 두개의 process 를 비교하기 위해서 여러 개의

[1] 본 연구는 1998 년 과학기술 기초중점지원사업으로 이루어졌습니다.

equivalence 개념을 제공한다. 이것을 기반으로 요구명세와 설계 명세를 비교하여 그들 간에 equivalence 관계가 성립하는 지를 보임으로써 설계하고 있는 명세가 요구조건을 만족하는 지를 보일 수 있다.

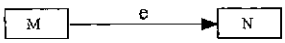
4. Translation from Statechart to ACSR

이 장에서는 Statechart 를 ASCR 로 바꾸는 방법을 설명한다. 먼저 Statechart 와 ACSR 사이에 의미론적 차이를 알아보고 그 차이를 극복할 수 있는 방법을 제시한다.

4.1 의미론적 차이

가. Non-delayed vs. delayed event

Statechart 에서는 transition 의 triggering condition 이 참이 되지 않는 한 control 이 그 state 에 계속해서 머무를 수 있다. 즉, 다음 그림에서 event 'e'가 발생하지 않는 한 control 은 M 에 머물러 있다.



반면에, ACSR 에서는 control 이 특정 process 에 머물 수 없다. 예를 들면, $Q = e.P$ 의 expression 에서 Q process 에 control 이 t time 에 온다면 그 후 바로 event e 를 받고 같은 t time 에 control 이 process P 로 넘어가게 된다. Statechart 의 delay event 를 ACSR 에 적용하기 위해서 다음과 같이 process 를 정의한다.

$$Q = recX.(e.P + \phi : X)$$

나. Broadcasting vs. Synchronization

ACSR 은 동기적인 통신 mechanism 을 가지고 있는 반면에 Statechart 는 broadcasting mechanism 을 가지고 있다. broadcasting 의 개념은 receiver 쪽의 a?? process 와 sender 쪽의 a!! process 로 나타낼 수 있다. 즉, a??.P process 는 sender 가 message a 를 보내줄 때까지 시간에 무관하게 기다린다는 의미이며, a!!.P process 는 시간 진행 없이 a 를 필요로 하는 모든 프로세스에게 a 를 보낸 후에 P process 로 전이된다는 의미이다. ACSR 에서 process a?.P 와 a!.Q 는 synchronize 기능을 가지고 있기 때문에 statechart 의 1:n 통신을 위해서는 다음 process 처럼 명확한 waiting step 과 multiple sending procedure 를 첨가해야 한다.

$$a??.P = recX.(a?.P + \phi : X)$$

$$a!!.Q = recX.(a!.X + Q)$$

그러나, 위의 표현에서 ACSR 과 Statechart 간의 시간에 대한 개념에 있어서 미묘한 차이가 발생하는데, 즉 ACSR 에서는 event 가 발생하고 그것을 받기 위해서는 시간이 소모되지 않는 반면에 Statechart 에서는 한 step 이 소모된다. 따라서 Statechart 의 step 과 시간의 흐름과는 관계가 없다고 가정한다

다. State variable

Statechart 에서는 control 이 state M 에 들어가고 나올 때 en(M), ex(M)이라는 event 가 발생된다. 이 event 들은 ACSR 의 syntax 로 enter_M!, exit_M!으로 표현할 수 있다. 그리고, Statechart 에서 transition 의 trigger 로 나타나는 en(M),과 ex(M)은 ACSR 에서 enter_M?, exit_M?으로 표현할 수 있다. 또한, Statechart 에서 control 이 state M 에 있으면 in(M)은 true 가 되고 control 이 없으면 false 가 되는데 이것은 ACSR 에서 true 일 경우 t_M! event 를 계속 발생하고 false 일 경우 f_M! event 를 계속 발생하는 것으로 변환할 수 있다. 다음 표현은 Statechart 에서 state M 에 정의되는 semantics 를 ACSR 로 표현한 것이다.

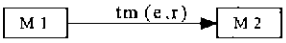
$$P\_Mtest = P\_Mf$$

$$P\_Mf = f\_M!.P\_Mf + enter\_M?.P\_Mt + \phi : P\_Mf$$

$$P\_Mt = t\_M!.P\_Mt + exit\_M?.P\_Mf + \phi : P\_Mt$$

라. Timing construct

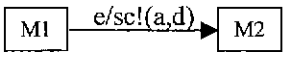
Statechart 에서 timing construct 는 timeout 과 schedule 이 있다. timeout 의 syntax 는 tm(e,r)인데 이것은 event e 가 발생한 후 r time unit 후에 tm(e,r) event 가 발생한다는 것을 의미한다. schedule 의 syntax 는 sc!(a,d)인데 이것은 transition 이 발생한 후 d time unit 후에 event a 가 발생한다는 의미이다.



위 그림은 Statechart 에서 M1 state 와 M2 state 간의 transition 에 timeout event 가 적용된 경우이다. 이것을 ACSR 표현으로 고치면 다음과 같다.

$$M1 = e??.IDLE\Delta_r(M2, NIL)$$

아래는 Statechart 의 schedule 을 ACSR 로 표현한 것이다.



$$M1 = e??.(M2 \parallel IDLE\Delta_d(a!!.IDLE, NIL))$$

4.2 From Statechart to ACSR

이 절에서는 Statechart 명세를 ACSR 명세로 바꾸는 방법을 소개한다.

가 State

각 state M 은 두개의 process 를 가지고 있는데 하나는 그 state 의 행위를 기술하는 P_M process 이고 다른 하나는

State variable 의 상태를 나타내는 P_Mtest 이다 먼저 시스템의 가장 상위 state P_M 은 enter_M!을 broadcasting 하면서 state 속으로 control 이 들어간다.

$$P\_M = recX.(enter\_M!.X + dummy.P\_M')$$

Dummy event 는 priority 가 가장 낮은 임의의 event 로서 enter_M! event 를 필요로 하는 모든 process 와 synchronize 한 후 P_M' process 로 전이하게 하기 위해 필요하다. P_M' process 는 다시 다음 3 가지 process 로 분류된다.

1) For a basic state M, $P\_M' = IDLE$

2) For a concurrent state M with substates, M1, M2, . . , Mn

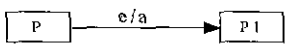
$$P\_M' = P\_M_1 \| ..... \| P\_M_n$$

3) For a OR state M with initial state M1, $P\_M' = P\_M_1$

나 Transition

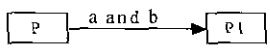
State 들간의 transition 은 trigger condition 에 따라 다음과 같이 표현된다.

1) trigger 가 event 이고 action 을 가진 경우.



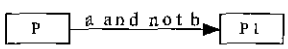
$$P = (e?,1).exit\_P!!.a!!.P1 + \phi : P$$

2) trigger 가 event 의 논리곱으로 되어 있을 경우,



$$P = (a?,1).((b?,2).P1 + P) + (b?,1).((a?,2).P1 + P) + \phi : P$$

3) trigger 가 ***a and not b*** 의 형태로 되어 있을 경우,



$$P = (a?,1).((b?,1).P + P1) + \phi : P$$

4) trigger 가 ***en(M)*** 일 경우,

$$P = (enter\_M?,1).exit\_P!!.P1 + \phi : P$$

5) trigger 가 ***ex(M)*** 일 경우,

$$P = (exit\_M?,1).exit\_P!!.P1 + \phi : P$$

6) trigger 가 ***in(M)*** 일 경우,

$$P = (t\_M?,1).exit\_P!!.P1 + \phi : P$$

7) trigger 가 ***not in(M)*** 일 경우,

$$P = (f\_M?,1).exit\_P!!.P1 + \phi : P$$

다. External Event Generator

Statechart 에서 외부 event 는 broadcasting 되지만 ACSR 에서는 synchronize 되기 때문에 이 부분 또한 동일한 semantics 를 가지도록 해 주어야 한다. 아래 ACSR 표현은 외부에서 들어오는 event 를 가장 높은 priority 로 받아들여서 그 event 를 필요로 하는 모든 process 에게 제공하는 mechanism 이다.

$$EG = \phi : EG + (a_1?.k).a_1!!.EG$$
$$+ ..... + (a_n?.k).a_n!!.EG$$

위의 식에서 a 는 외부 event 이고 k 는 가장 높은 priority 이다.

라. System

Statechart 명세에서 가장 외부의 state 는 전체 시스템을 나타낸다. 그 state 를 M 이라 하면 그 state 의 substate M1.M2....,Mn 들은 ACSR 명세에서 각각 P_M1test,..., P_Mntest 라는 process 들을 가지고 있다 P_additional 을 아래 식으로 정의하면 전체 시스템은 다음과 같이 나타낼 수 있다.

$$P\_additional = (P\_M_1 test \| .... \| P\_M_n test)$$
$$P\_M = (P\_M' \| P\_additional \| EG) \setminus F$$

위의 식에서 F 는 synchronize 되어야 하는 모든 event 들을 나타낸다.

5. 결론

지금까지 이 논문에서 Statechart 의 subset 를 process algebra 의 일종인 ACSR 로 변환하는 방법을 설명하였다. ACSR 로 statechart 의 semantics 를 정의하는 것에 의해 statechart 의 ambiguity 를 제거하였고 또한 statechart 로 명세된 real time system 을 ACSR 의 분석 tool 인 VERSA 를 통하여 명세의 correctness 를 검증할 수도 있게 되었다. 하지만 David Harel 에 의해 제안되고 Statemate tool 에 의해 채택된 Statechart 의 full syntax 와 semantics 를 변환하기 위해서는 static reaction, value assignment, complex transition condition 등에 대해서 더 많은 확장이 필요하다.

6. 참고문헌

[1] David Harel, Steatecharts,:A Visual Formalism For Complex Systems, Science of Computer Programming 8, 1987.

[2] David Harel and Amnon Naamad, The Statemate Semantics of Statecharts, ACM Trans. Soft. Method. 1996.

[3] David Harel and Michal Politi, Modeling Reactive Systems with StateCharts, McGraw-Hill, 1998.

[4] Jin-Yong Choi, Insup Lee, A Process Algebraic Method for the Specification and Analysis of Real-Time Systems, Formal Methods for Real-Time Computing, 1996

[5] Jin-Yong Choi, Inhye Kang, Translation of Modechart Specification to Algebra of Communicating Shared Resources

[6] Duncan Clarke, VERSA: Verification, Execution and Rewrite System for ACSR, 1998