실시간 시스템의 효과적 명세를 위한 Statecharts의 ACSR 변환

장성호[O] 최진영
고려대학교 컴퓨터학과
{shjang, choi}@formal.korea.ac.kr

Translation statecharts to ACSR for Real-Time System with shared resource

Sung-Ho Jang[O] Jin-Young Choi
Dept. of Computer Science, Korea University

요 약

Statecharts는 복잡한 Reactive System의 행위적인 부분을 가시적으로 명세하는데 효과적이다. CCS에 근간을 둔 ACSR은 실시간 시스템을 명세하기 위하여 시간, 우선순위, 자원의 개념을 추가하여 엄격한 명세가 가능하다. 본 논문에서는 Statecharts와 ACSR의 공통점과 차이점을 보이고, 동기적인 시간을 적용한 Statecharts를 ACSR로 변환할 수 있음을 보인다. 또한, 변환된 ACSR에 공유자원에 대한 명세를 추가함을 보인다.

1. 서 론

실시간 시스템[1]을 개발함에 있어서 정형기법은 시스템의 개발 초반부 뿐만 아니라 개발의 후반부까지 영향을 끼친다. 실시간 시스템에서 정형기법의 중요성은 더욱 크다고 할 수 있으며, 다양한 정형기법들이 발전해 왔고 계속적으로 발전하고 있다.

Statecharts[2][3]는 가시적인 방법으로 복잡한 Reactive System [4]의 행위적인 부분을 효과적으로 명세할 수 있다. CCS[5] (Calculus of Communicating Systems)에 기반한 ACSR(Algebra of Communication Shared Resources) [6][7]은 실시간 시스템을 명세하기 위하여 시간(time), 우선순위 (priority), 자원(resource)의 개념이 추가되었다. Statecharts는 ACSR에 비교하여 사람이 인지하기 쉽게 명세할 수 있는 장점이 있고, ACSR은 실시간 시스템에 대하여 Statecharts에서 표현 못하는 부분을 명세하거나 동치성을 확인 할 수 있다.

본 논문[1]에서는 Statecharts와 ACSR의 차이점 및 공통점을 보이고, 동기적 시간(Synchronous time scheme)을 적용한 Statecharts를 ACSR로 변환함을 보인다.

2. 관련 연구

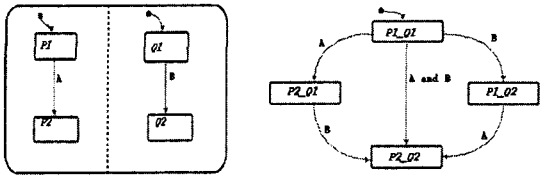
Statecharts와 Process Algebra[8]와 접목하려는 노력은 많이 있었다. 특히, ACSR에 접목하려는 경우도 있었다. GCSR[9]은 Statecharts와 유사하게 가시적인 명세를 하였고, ACSR로 변환하여 동치성 검증이 가능하도록 되어있다. 그러나, GCSR에 표현된 명세는 ACSR로 변환하는데 적합한 것으로 Statecharts[2][3]와 다르다. [10]에서 Statecharts를 ACSR로 변환하려 하였다. 그러나, Statecharts에서 사용된 의미론적 시간 적용(time scheme)이 불명확하고, 변환과정에서 부족한 부분이 있다. 그래서, [10] 논문에서 제시한 방법으로 변환을 하였을 때, 변환된 ACSR는 Statecharts 행위가 다르게 발생하는 경우가 존재한다.

1) 본 연구는 과학기술부 원자력연구개발사업 중 원전계측제어시스템 연구개발사업 위탁연구 과제로 수행되었음.

3. 공통점 및 차이점

Statecharts와 ACSR은 병렬적이고 복잡하고 동적인 시스템을 효율적으로 또는 엄격하게 명세하는 것을 목적으로 한다. 그래서, 2개의 언어는 구조적인 또는 표현상의 공통점을 많이 가지고 있다. 물론, 같은 것에 대하여서도 내포하고 있는 의미가 다르고, 심지어는 서로간 표현하지 못하는 부분도 있다. 여기에서는 병렬성(parallelism), 통신(communication)을 주된 대상으로 공통점과 차이점을 기술한다.

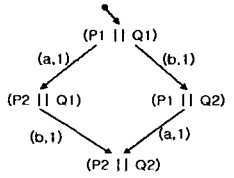
공통점으로는 병렬성에 대한 표현과 외부 및 내부간의 통신 그리고 시간에 대한 표현들로 이루어짐을 공통점으로 볼 수 있다. 첫 째 병렬성의 공통점 및 차이점은 아래의 그림들에서 확인할 수 있다.



[그림 1] Statecharts 병렬성 해석

```
Model = (P1 || Q1) ;
P1 = (a,1).{}:P2+{}:P1 ;
P2 = {}:P2 ;

Q1 = (b,1).{}:Q2+{}:Q1 ;
Q2 = {}:Q2 ;
```



[그림 2] ACSR 병렬성 해석

[그림 1]과 [그림 2]의 공통점은 상태가 데카르트의 곱으로 표현될 수 있음에 있고, 차이점은 [그림 2]에서 (P1||Q1) → (P2||Q2)로 바로가는 행위가 빠져있음에서 살펴볼 수 있다. 그 이유는 병렬성을 표현함에 있어서, Statecharts는 내부 또는 외부에서 발생하는 자극(event, action)이 동시에 여러개 발생하는 것을 허용한다. 그러나, ACSR에서는 발생하는 자극(input·

output event, Tau)은 1개만 허용한다. 그래서, 병렬성을 표현하는데 상태의 갯수는 같지만, 외부 또는 내부 자극에 연계된 상태 전이(Transition)의 갯수는 다르게 표현된다.

둘 째 통신 방식의 차이점으로 볼 수 있다. Statecharts는 Broadcasting 방식을 사용하기 때문에 내·외부 자극에 대하여 해당하는 모든 상태가 상태 전이 가능하다. ACSR은 채널을 통한 Interleaving 방식을 사용하기 때문에 내·외부 자극에 대하여 1개만 상태 전이 가능하거나, 동기화(synchronization)된 경우 Tau에 의하여 2개 상태가 동시에 상태전이 하는 것이 가능하다.

위의 언급한 두가지 차이점은 결국 통신 방식에 대한 차이점에서 파생된 결과라고 볼 수 있다.

첫 째 차이점은 Statecharts에서 event A 와 B가 동시에 발생한 것을 ACSR에서 시간이 소모하기 전에 event A -> event B 순으로 발생하거나, event B ->event A 순으로 발생한 것으로 변환함으로써 해결한다. 그리고, 둘 째 차이점은 Statecharts의 action에 대하여 Broadcasting하는 것 처럼, ACSR에서 Broadcasting 할 수 있도록 변환함으로써 해결한다.

4. 변환

상태(State), Broadcasting, Transition으로 구분하여 변환해 보도록 하겠다. Statecharts는 STATEMATE를 ACSR은 VERSA[7]를 사용하였다.

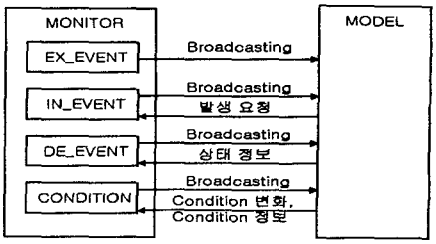
4.1 상태(State)

Statecharts는 크게 Basic states , OR-states, AND-states 3가지로 구성되어 있다. Parents state는 계층적인 상위의 상태를 의미한다. 아래과 같이 ACSR과 직접적으로 대응으로 상태는 변환이 가능하다. 다음은 각 상태에 대한 변환과 재해석이다.

- Basic states에 대한 ACSR 표현과 재해석
  P1 = {}:P1
  {}: P1 : 시간을 소모하고 P1 process로 전이함
  재해석 : P1은 Basic state이다.
- Parent state에 대한 ACSR 표현과 재해석
  scope(P, dummy, infty, NIL, NIL, (E,1).Q)
  P : P Process는 event E에 의하여 Q process로 상태 전이한다.
  재해석 : P는 Parents state 이고, event E가 발생하면 Q상태로 탈출한다.
- AND-states에 대한 ACSR 표현과 재해석
  MODEL = (P || Q)
  (P||Q) : MODEL process는 P와 Q process가 병렬적으로 구성되어 있는 상태로 전이한다.
  재해석 : MODEL은 병렬적인 P 상태와 Q 상태로 구성되어 있다.
- OR-states는 위에서 언급된 상태들이 배타적 관계에 있음으로 표현가능하다.

4.2 Broadcasting

[그림 3]은 Broadcasting을 위한 ACSR 변환을 도식적으로 나타낸 것이다. MONITOR를 통해서만 MODEL에 자극을 줄 수 있다. 즉, *MONITOR*는 내·외부에서 발생하는 자극을 Broadcasting 할 수 있는 부분이고, MODEL은 자극에 의하여 반응하는 Reactive system으로 해석된다.



[그림 3] Broadcasting을 위한 ACSR의 구조

Statecharts를 ACSR로 변환하기 위하여 자극은 크게 4가지로 구분한다. 첫 째는 MODEL 외부에서 발생한 event이고, EX_EVENT에 표현한다. 둘 째는 MODEL 내부에서 발생한 action에 의한 event 발생 요청이고, IN_EVENT에 표현한다. 셋 째는 MODEL의 상태 정보와 연관하여 내부에서 발생한 derived event이고, DE_EVENT에 표현한다. 넷 째는 참 또는 거짓을 가지는 condition으로 CONDITION에 표현한다.

IN_EVENT의 action에 의한 event 발생요청과 CONDITION의 condition 값 변화 요청은 시간 경과 후 broadcasting 되고, 나머지 MONITOR의 요소는 시간을 소모하기 전에 broadcasting 된다.

위와 같이 구성된 MONITOR를 MODEL과 병렬적으로 구성하여 ACSR로 변환한다. 이 때 MONITOR와 MODEL간의 통신에 사용되는 event들은 동기화(synchronization)를 통하여 외부적으로 보이지 않는다. 그래서, STATEMATE처럼 사용자는 외부에서 발생한 event만 발생시킬 수 있게 된다.

변환된 ACSR에서는 DE_EVENT > CONDITION > IN_EVENT > EX_EVENT 순으로 Broadcasting 되도록 한다. EX_EVENT보다 다른 event가 먼저 Broadcasting 되게 함으로써 외부자극을 입력하기 전에 내부적인 자극은 Tau에 의하여 Broadcasting 된다. 즉, Statecharts Semantics에서 외부에서 발생한 event가 내부적으로 발생한 Broadcasting event list에 추가된 것과 같게 된다. DE_EVENT 와 CONDITION의 우선순위를 높임으로 인하여 Transition을 효과적으로 변환할 수 있게 된다.

4.3 Transition

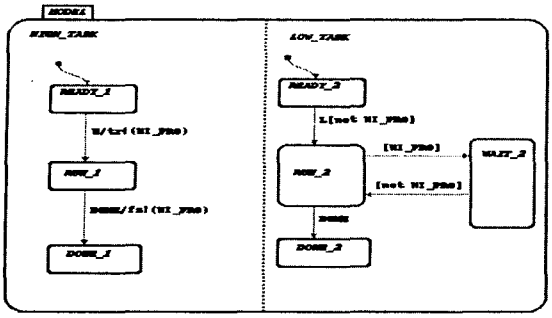
[그림 4]는 Statecharts의 Transition을 ACSR로 변환한 것이다. <8>에서 DE_EVENT가 CONDITION 보다 우선 순위가 높음으로 인하여 <7>보다 효율적으로 변환할 수 있다.

```
<1> Transition :
   P1 = (EMPTY,1).{}:P2 + {}:P1 ;
<2> Transition : E
   P1 = (E, 1).{}:P2 + {}:P1 ;
<3> Transition : E1 AND E2
   P1 = (E1,1).({}:P1 + (E2,1).{}:P2) + (E2,1).({}:P1 + (E1,1).{}:P2) + {}:P1 ;
<4> Transition : E1 OR E2
   P1 = (E1,1).{}:P2 + (E2,1).{}:P2 + {}:P1 ;
<5> Transition : E / A
   P1 = (E,1).('A_GEN,1).{}:P2 + {}:P1;
<6> Transition : E [in(Q1)] / A
   P1 = (inQ1,1).({}:P1 + (E,1).('A_GEN,1).{}:P2) + {}:P1;
<7> Transition : E [C1 and C2] / A
   P1 = (C1, 1).({}:P1 + (C2,1).({}:P1 + (E,1).('A_GEN){}:P2)) +
        (C2, 1).({}:P1 + (C1,1).({}:P1 + (E,1).('A_GEN){}:P2)) + {}:P1;
<8> Transition : E [in(Q1) and C] / A
   P1 = (inQ1,1).({}:P1 + (C,1).({}:P1 + (E,1).('A_GEN,1).{}:P2)) + {}:P1 ;
```

[그림 4] Statecharts의 Transition 변환

5. 변환 예제와 자원 소모 추가

지금 까지 논한 것을 바탕으로 Statecharts로 명세된 것을 ACSR로 변환하겠다. 대상은 2개의 TASK가 병렬적으로 작동하는 것이다.



[그림 5] STATEMATE로 명세된 MODEL

[그림 5]는 우선 순위가 다른 2개 TASK의 행위를 Statecharts로 명세한 것이다. 2개 TASK는 병렬적으로 구성되어 있고, 각 TASK는 실행을 하기 위한 준비 상태에 있게 된다. [그림 5]는 HIGH_TASK에게 우선권이 있음을 표현하고 있다. LOW_TASK는 HIGH_TASK가 실행되지 않을 때만 실행될 수 있으며, LOW_TASK가 실행도중 HIGH_TASK가 실행되면 HIGH_TASK가 종료할 때 까지 대기하게 된다.

[그림 6]은 [그림 5]를 위에서 언급한 방법으로 ACSR로 변환한 것이다.

```
SYSTEM = ( MONITOR || MODEL )\
{H,L,DONE, H_GEN, L_GEN, DONE_GEN, TR_HI_PRO, FS_HI_PRO,
HI_PRO_TRUE, HI_PRO_FALSE, EMPTY};

MONITOR = ( EX_EVENT || IN_EVENT || DE_EVENT || CONDITION);

 EX_EVENT = (EX_H || EX_L||EX_DONE);
   EX_H = (ex_H_GEN,1).EX_H_GEN + {}:EX_H ;
   EX_H_GEN  =('H,1).EX_H_GEN + {}:EX_H ;

   EX_L = (ex_L_GEN,1).EX_L_GEN + {}:EX_L ;
   EX_L_GEN  =('L,1).EX_L_GEN + {}:EX_L ;

   EX_DONE = (ex_DONE_GEN,1).EX_DONE_GEN + {}:EX_DONE;
   EX_DONE_GEN = ('DONE,1).EX_DONE_GEN + {}:EX_DONE;

 IN_EVENT = (IN_H || IN_L || IN_DONE || IN_EMPTY);

   IN_H = (H_GEN,1).{}:IN_H_GEN + {}:IN_H ;
   IN_H_GEN  =('H,2).IN_H_GEN + IN_H ;

   IN_L = (L_GEN,1).{}:IN_L_GEN + {}:IN_L ;
   IN_L_GEN  =('L,2).IN_L_GEN + IN_L ;

   IN_DONE = (DONE_GEN,1).{}:IN_DONE_GEN + {}:IN_DONE;
   IN_DONE_GEN = ('DONE,2).IN_DONE_GEN + IN_DONE;

   IN_EMPTY = ('EMPTY,2).IN_EMPTY + {}:IN_EMPTY;

 DE_EVENT = {}:DE_EVENT;

 CONDITION = (HI_PRO);

   HI_PRO = HI_PRO_F_GEN + (TR_HI_PRO,2).{}:HI_PRO_T ;
   HI_PRO_F_GEN = ('HI_PRO_FALSE,3).HI_PRO_F_GEN +
                  (TR_HI_PRO,1).{}:HI_PRO_T + {}:HI_PRO ;

   HI_PRO_T = HI_PRO_T_GEN + (FS_HI_PRO,2).{}:HI_PRO ;
   HI_PRO_T_GEN = ('HI_PRO_TRUE,3).HI_PRO_T +
                  (FS_HI_PRO,1).{}:HI_PRO + {}:HI_PRO_T ;
```

(계속)

```
MODEL = (HIGH_TASK || LOW_TASK);

HIGH_TASK = READY_1;
 READY_1 = (H,1).('TR_HI_PRO,1).{}:RUN_1 + {}:READY_1 ;
 RUN_1 = (DONE,1).('FS_HI_PRO,1).{}:DONE_1 +{{CPU,2}}:RUN_1;
 DONE_1 = {}:DONE_1;

LOW_TASK = READY_2;
 READY_2 = (HI_PRO_FALSE,1).({}:READY_2 + (L,1).{}:RUN_2 ) +
           {}:READY_2 ;
 RUN_2 = (HI_PRO_TRUE,1).{}:WAIT_2 + (DONE,1).{}:DONE_2 +
         {{CPU,1}}:RUN_2 ;
 WAIT_2 = (HI_PRO_FALSE,1).{}:RUN_2 + {}:WAIT_2;
 DONE_2 = {}:DONE_2;
```

[그림 6] 자원 소모 표현을 추가한 변환된 VERSA

밑 줄로 표시된 부분은 CPU 사용에 있어서 HIGH_TASK 가 LOW_TASK 보다 우선순위를 가지고 사용할 수 있음을 표현하고 있다.

5. 결론 및 연구 계획

본 논문에서 동기적인 시간을 적용한 Statecharts를 ACSR로 변환하였다. 기대 효과는 시스템의 행위적인 부분은 Statecharts를 이용하여 효과적으로 명세하고, 공유된 자원 소모에 대한 부분은 변환된 ACSR을 이용하여 추가할 수 있다. 또한, Statecharts에 자원소모에 대한 의미론적인 해석을 추가할 수 있으며, Statecharts에 사용되는 시간에 대한 표현을 ACSR로 보다 엄격하게 재해석 할 수 있다.

향후 STATEMATE에서 VERSA로 자동 변환할 수 있도록 도구화할 계획이다. 또한, 변환된 ACSR을 이용하여 Statecharts의 자원 표현과 시간표현 확장에 대하여 연구할 계획이며, Statecharts와 ACSR 각 언어의 장점을 살릴 수 있는 실시간 시스템 개발 방법론에 대하여 연구할 계획이다.

참고 문헌

[1] Editors C. Heitmeyer and D. Mandrioli, "Formal Methods for Real-Time Computing", Trends in Software series, volume 5, John Wiley & Sons, 1996.

[2] Harel.D., "Statecharts: A Visual Formalism for Complex Systems", *Sci. Comput. Prog.* 1987.

[3] David Harel and Amnon Naamad, "The STATEMATE Semantics of Statecharts", *ACM Trans. soft. eng. Method*, 1995.

[4] Harel, D. and M. Politi, *"Modeling Reactive Systems with Statechsrts : the STATEMATE Aproach", McGraw-Hill, 1998*

[5] Robin Milner, "Communication and Concurrency", Prentice--Hall, 1989.

[6] J. Y. Choi, I. S. Lee and H. Ben-Abdallah, A Process algebraic method for the specification and analysis of real-time systems, Formal Methods for Real-Time Computing, 1996.

[7] D. Clarke, VERSA : Verification, Execution and Rewrite System for ACSR, Real-Time Group Report, University of Pennsylvania, 1996.

[8] Gerald Luettgen , Michael von der Beeck , Rance Cleaveland, Statecharts via process algebra, NASA/CR-1999-209713 ICASE Report, NASA Langley Research Center Hampton, VA 23681-2199, 1999.

[9] H. Ben-Abdallah, I. Lee, and J.-Y. Choi, A Graphical Language with Formal Semantics for the Specification and Analysis of Real-Time Systems, Proceedings of the 16th IEEE Real-Time Systems Symposium, 1995.

[10] M. H. Park, K. S. Bang, J. Y. Choi, Equivalence Checking of Two Statechart Specifications, IEEE RSP'2000, Paris, 2000.