평탄화를 이용한 계층형 상태 기계의 단계 의미 정의

박 사 천[†] · 권 기 현[††] · 하 순 회[†††]

요 약

하드웨어와 소프트웨어를 통합 설계하는 프레임워크인 PeaCE(Ptolemy extension as a Codesign Environment)가 개발되었다. PeaCE 에서는 데이터 흐름과 제어 흐름을 모두 표현할 수 있는데, 제어 흐름은 상태 기계를 확장한 fFSM으로 나타낸다. fFSM은 계층형 상태 기계로서 제어 흐름을 표현하기 위해 많은 구문을 제공하지만, 모델에 대한 의미가 정의되어 있지 않아서 명세를 검증하는데 어려움이 많다. fFSM의 의미를 정의하기 위해서, 본 논문에서는 계층형 상태 기계를 먼저 평탄화한 후에 평탄화된 모델에 대해서 단계 의미를 정의하였다. 그 결과 레이스 조건, 애매한 전이, 순환 전이 등의 주요한 버그들을 정형적으로 검출할 수 있었다.

키워드 : 상태 기계, 평탄화, 검증, 통합 설계, 단계 의미

Definition of Step Semantics for Hierarchical State Machine based on Flattening

Sachoun Park[†] · Gihwon Kwon[††] · Soonhoi Ha[†††]

ABSTRACT

Hardware and software codesign framework called PeaCE(Ptolemy extension as a Codesign Environment) was developed. It allows to express both data flow and control flow which is described as fFSM which extends traditional finite state machine. While the fFSM model provides lots of syntactic constructs for describing control flow, it has a lack of their formality and then difficulties in verifying the specification. In order to define the formal semantics of the fFSM, in this paper, firstly the hierarchical structure in the model is flattened and then the step semantics is defined. As a result, some important bugs such as race condition, ambiguous transition, and circulartransition can be formally detected in the model.

Key Words : State Machine, Flattening, Verification, Codesign, Step Semantics

1. 소 개

PeaCE(Ptolemy extension as a Codesign Environment)는 임베디드 시스템 개발을 돕기 위한 하드웨어/소프트웨어 통합 설계 환경이다[1]. 이 환경에서는 시스템의 데이터 흐름과 제어 흐름을 모두 표현할 수 있는데, 시스템의 제어 흐름은 *f*FSM(*flexible* Finite State Machine) 모델로 나타낸다. 이 모델은 표현력을 높이기 위해서 유한 상태 기계를 확장했으며, 하렐의 스테이트챠트[2]와 같이 병행성 및 계층성을 표현할 수 있는 구문과 여러 기계간의 동기화를 위해 이벤트 및 상태 변수 등의 구문을 제공한다.

비록 fFSM이 시스템의 제어 흐름을 표현하기 위해 풍부한 구문을 제공하지만, 모델에 대한 정형 의미가 정의되지 않아서 외부에서 입력된 이벤트에 대한 시스템의 반응을 논리적 또는 수리적으로 계산하는데 어려움이 있다. 이러한 문제점 때문에 모델에 대한 정형 분석(예를 들어 레이스 조건, 애매한 전이, 순환 전이 등이 모델에 없는지를 검증)이 어렵다. 따라서 본 연구에서는 모델의 정형 분석을 돕기 위해서 모델의 의미를 정형적으로 정의한다.

유한 상태 기계 모델의 의미를 정의하는데 단계 의미(step semantics)가 전통적으로 사용된다[3]. 단계 의미란 모델이 이벤트를 받아서 한 형상(모델의 스냅샷을 일컫는 용어로서 형상은 상태 집합으로 표현된다)에서 다른 형상으로 변하는 과정을 정의하는 동시에 출력될 이벤트 집합을 정의한다. 본 연구에서는 *f*FSM 모델의 의미를 정의하기 위해서 모델에 있는 계층 구조를 평탄화한 후에 단계 의미를 정의했다. SMV(Symbolic Model Verifier, [4])와 같이 유한 상태 기계를 검증하는 대부분의 모델 체커들이 평탄화된 모델을 사용하기 때문에, 본 논문에서 정의된 의미는 이들 모델 체커에 효율적으로 사용될 수 있다. 정의된 의미의 정확성을 확인하기

※ 본 과제는 정보통신부 선도기반기술개발사업의 지원으로 수행되었습니다.
† 준 회 원 : 경기대학교 전자계산학과 박사과정
†† 종신회원 : 경기대학교 정보과학부 교수
††† 정 회 원 : 서울대학교 컴퓨터공학과 교수
논문접수 : 2005년 7월 4일, 심사완료 : 2005년 11월 7일

위해서 모델에 버그를 고의로 집어넣은 후에 이들 버그를 찾는 실험을 했다. 그 결과 레이스 조건, 애매한 전이, 순환 전이와 같은 버그를 정의된 의미를 이용하여 검출 할 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 *f*FSM 모델의 구문을 설명하고, 3장에서는 계층 구조를 평탄화시키는 방법을 설명한다. 그리고 4장에서는 평탄화된 모델을 바탕으로 단계 의미를 정의하고 이를 바탕으로 버그를 검출하는 방법을 보인다. 마지막으로 5장에서 결론을 맺는다.

2. *f*FSM 구문

이 장에서는 시스템의 제어 흐름을 표현하기 위해서 유한 상태 기계를 확장한 fFSM 모델의 구문을 설명한다 (지면 관계상 단계 의미 정의와 관련된 구문만을 설명할 것이며 나머지 상세한 구문은 [5]를 참조하기 바람).

[정의 1] (이벤트): 이벤트 e는 (e_n, e_v) 쌍으로 정의되는데, 여기서 e_n은 이벤트의 이름이며 e_v는 이벤트가 취할 수 있는 가능한 값을 나타낸다. 특히 ε는 값을 갖지 않는 이벤트가 발생한 경우를 나타낸다.

이벤트는 입력 이벤트, 출력 이벤트, 그리고 내부 이벤트 세 종류로 구분된다. 입력 이벤트는 값을 읽는데, 출력 이벤트는 값을 쓰는데, 그리고 내부 이벤트는 값을 읽고 쓰는데 사용된다. 따라서 내부 이벤트는 입력 이벤트와 출력 이벤트의 교집합으로 기술된다. 정의 2는 이벤트 집합을 정의한다.

[정의 2] (이벤트 집합): 입력 이벤트, 출력 이벤트, 그리고 내부 이벤트 집합은 각각 $I=\{(i_{n_1}, i_{V_1}), (i_{n_2}, i_{V_2})\cdots\}$, $O=\{(o_{n_1}, o_{n_1}), (o_{n_2}, o_{V_2})\cdots\}$, $IT = I \cap O$이다.

*f*FSM 모델은 표현력을 높이기 위해서 계층성 및 병행성 그리고 변수 상태(variable state)를 지원한다. 이들은 정의 3에서와 같이 상태 집합의 이름과 허용된 상태들의 집합, 그리고 초기 상태들로 정의된다. 계층적인 *f*FSM을 위한 상태 집합에서 특별한 값인 Φ가 x_v의 원소가 될 수 있고 이것은 비활성화된 상태를 식별한다. 비록 변수 상태가 상태처럼 정의되었다고는 하나 이것은 값이 유지되는 특별한 이벤트처럼 다루어질 수 있다. 이 변수 상태의 값을 읽거나 씀으로써 내부 이벤트와 비슷하게 사용할 수 있다.

[정의 3] (상태 집합과 초기값): $X=\{(x_{n_1}, x_{V_1}), (x_{n_2}, x_{V_2})\cdots\}$은 상태집합 이름과 각 상태집합이 가지는 실제의 유한 집합의 쌍으로 된 유한 집합이다. $V \subset X$는 변수 상태의 이름과 해당 값의 범위 집합으로 짝지어진 쌍의 유한 집합이다. $R \subseteq \{(x_n, x_v) | (x_n, x_v) \in X,\ x_v \in x_V\}$은 초기 상태들의 집합이다. 모든 (x_n, x_V)에 대해서 유일한 $(x_n, x_V) \in R$을 갖는다.

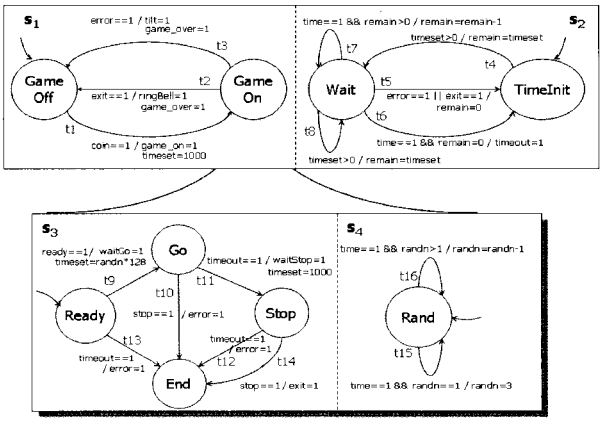
상태간의 움직임은 전이로 나타낸다. 전이에는 움직임의 조건을 나타내는 가드 조건과 상태를 전이할 때 수행해야할 액션이 레이블된다. 가드 조건은 입력 이벤트와 변수 상태의 조합으로 구성된 부울식이다. 액션은 출력 이벤트 또는 내부 이벤트 발생 또는 변수에 값을 배정한다. 전이의 가드 조건이 만족되면, 전이에 레이블된 액션이 실행되고 상태가 변경된다. 정의 4는 전이 집합을 정의한다.

[정의 4] (전이 집합): 전이 집합은 $F \subseteq f^{XI} \times f^{G} \times f^{XO} \times f^{A}$이다. 여기서, $f^{XI} \cup f^{XO} \subseteq \{(x_n, x_V) | (x_n, x_V) \in X,\ x_n \in x_V\}$는 전이의 출발 상태와 도착 상태의 집합이다. $f^G = f(e_{n_1}, e_{n_2}, \cdots)$는 가드 조건을 나타내는 부울식이다. 여기서 각각의 e_{n_i}는 $(e_{n_k}, e_{V_k}) \in I \cup V$로서 입력과 변수 상태로 구성된 부울 식이다. $f^A \subseteq \{(r_n, f^R) | (r_n, r_v) \in O \cup V,\ f^R = f(e_{n_1}, e_{n_2}, \cdots) \subseteq r_V, (e_{n_k}, e_{V_k}) \in I \cup V\}$는 액션들 $(r_n = f^R)$의 집합이다. 따라서 전이는 가드 조건 f^G이 만족할 때, 액션 f^A을 수행하면서 출발 상태 f^{XI}로부터 도착 상태인 f^{XO}상태로 움직인다.

위에서 정의된 정의를 이용해서 *f*FSM을 다음과 같이 정의할 수 있다.

[정의 5] (*f*FSM) 시스템의 제어 흐름을 나타내는 *f*FSM 모델은 6-튜플 (I, O, IT, X, V, F)이다. 위에서 정의한 것과 같이 I, O, IT는 입력 이벤트, 출력 이벤트, 내부 이벤트를 나타내며 X, V, F는 상태 집합, 변수 상태 집합, 전이 집합을 각각 나타낸다.

*f*FSM으로 리플렉스 게임(reflex game)을 모델링한 예가 (그림 1)에 있다. 이 게임에서 다루는 이벤트에는 *coin*, *ready*, *stop*, *time*이 있다. 마지막 이벤트는 게임에서 잔여 시간을 줄이는데 사용된다. 이 게임의 시나리오는 다음과 같다. 먼저 동전을 넣으면 *coin* 이벤트가 발생하고 시스템



(그림 1) *f*FSM 모델 예제

은 대기한다. 임의의 시간 후에 *ready* 이벤트가 발생하면 게이머는 재빨리 *stop* 버튼을 누른다. 이때 게임은 *stop* 이벤트를 발생시키고 *ready* 이벤트와 *stop* 이벤트가 발생한 시간 차이를 계산한다. 이 계산을 위해서 변수 상태 *remain*과 *randn*이 사용된다. (그림 1)에 대한 *f*FSM 정의 (I, O, IT, X, V, F)는 아래와 같다:

I = {(coin, { ε}), (ready, { ε}), (stop, { ε}), (time, { ε}), (timeout, { ε}), (error, { ε}), (exit, { ε}), (timeset, {0, ···, 1000})}

O = {(timeout, { ε}), (error, { ε}), (exit, { ε}), (timeset, {0, ···, 1000}), (game_on, { ε}), (waitGo, { ε}), (waitStop, { ε}), (ringBell, { ε}), (tilt, { ε}), (game_over, { ε})}

IT = {(timeout, { ε}), (error, { ε}), (exit, { ε}), (timeset, {0, ···, 1000})}

X = {(s_1, {GameOff, GameOn}), (s_2, {wait, TimeInit}), (s_3, {Ready, Go, Stop, End}), (s_4, {Rand}), (remain, {0, ···, 1000}), (randn, {1, ···, 3})}

V = {(remain, {0, ···, 1000}), (randn, {1, ···, 3})}

R = {(s_1, GameOff), (s2,TimeInit), (s3, Ready), (s4, Rand), (remain, 0), (randn, 3)}

F = {((s_1, GameOff), (coin == ε), (s_1, GameOn), {game_on= ε, timeset=1000}), ((s_1, GameOn), (exit == ε), (s_1, GameOff), {ringBell= ε, game_over= ε}), ((s_1, GameOn), (erro r== ε), (s_1, GameOff), {tilt= ε, game_over= ε}), ···}

*f*FSM 모델의 실행 방식은 다음과 같다. 맨 처음 모델은 입력 이벤트에 의해서 트리거 된다. 그리고 가드 조건이 만족될 때 전이가 실행된다. 만일 전이가 실행될 때 내부 이벤트가 발생되면 그 내부 이벤트에 의해서 트리거 되는 전이는 더 이상 내부 이벤트가 없을 때까지 반복적으로 발생된다. 매 델타-지연 후에 존재하는 이벤트는 모두 삭제된다. 그리고 이전 델타-지연에서 새롭게 생성된 내부 이벤트들을 준비한다. 요약하면 원소 *f*FSM 모델의 실행은 입력 이벤트에 의해서 트리거 되는 전이와 이전 전이에 의해서 발생된 내부 이벤트에 의해 트리거 되는 전이들로 구성된다. 유의할 것은 변수 상태와 출력 이벤트는 자신의 값을 유지한다는 것이다. 이러한 실행 규칙은 다음과 같은 비결정적 행위를 만들 수 있다.

1. 하나의 상태에서 다수의 전이가 동시에 활성화되면 그 중 하나를 비결정적으로 선택한다.
2. 이벤트에 대한 쓰기가 다수 있을 때, 이벤트의 최종 값은 비결정적으로 정해진다.
3. 순차적인 내부 전이에 의해서, 순환되는 전이가 존재할 수 있다.

이것을 애매한 전이, 레이스 조건, 순환 전이라고 부른다. 모델에 이러한 요소의 발생한 잠재적인 에러를 유발할 수 있어서, 이러한 에러들이 모델에 없는지를 분석해야 한다[6].

3. *f*FSM의 평탄화

*f*FSM의 단계 의미를 정의하기 위해서 계층 구조를 없앤 평탄화된 모델을 정의할 수 있다. 평탄화된 모델은 다음과 같은 단순 FSM들이 병행적으로 연결된다. 단순 FSM은 4개의 튜플 (S, s^0, T, scr)로 정의된다. 여기서 S는 상태 집합이고 s^0는 초기 상태이며, T는 전이 집합이다. PeaCE에서 데이터 흐름 모델들은 *f*FSM에서 생성되는 외부 신호에 의해서 제어된다. 이러한 신호들은 해당 상태에 원소 명제의 집합으로 레이블 된다. 우리는 이러한 레이블을 스크립트(scripts)라고 부르고, *f*FSM 내에서 나타나는 모든 스크립트들의 집합을 Script로 표시한다. 따라서 $scr: S \rightarrow 2^{Script}$는 각 상태에 스크립트를 레이블 하는 함수이다.

[정의 6] (평탄화된 모델) 평탄화 된 모델은 6-튜플 (I, O, IT, M, γ, V)로 구성된다. 여기서 I, O, IT는 전과 같이 이벤트들의 집합이다. V 전역 변수의 집합이고, $M = \{m_1, ..., m_n\}$는 단순 FSM의 집합이다. $\Sigma = \bigcup_{i=1}^{n} S_i$은 M에 속한 모든 상태들의 집합이다. 계층 관계 $\gamma: \Sigma \rightarrow 2^M$는 상태를 그 상태가 포함하는 상태 기계들로 사상하는 함수이다.

계층함수 γ는 다음의 세 가지 속성을 갖는다[7]. 첫째, 유일한 루트 상태 기계를 갖는다. 둘째, 루트를 제외한 모든 상태 기계들은 정확하게 하나의 조상 상태를 갖는다. 셋째, 계층 수는 싸이클을 포함하지 않는다. $sub : \Sigma \rightarrow 2^{\Sigma}$일 때, $sub(s) = \{s' | M_i \in \gamma(s) \wedge s' \in S_i\}$는 어떤 상태에 포함된 하위 상태들을 돌려주는 함수이다. sub^+은 sub의 추이 클로우저(transitive closure)이고 sub^*는 sub의 반사적 추이 클로우저(reflexive transitive closure)이다.

[정의 7] (단순 기계) 각각의 단순 기계 m_i는 4-튜플 (S_i, s_i^0, T_i, scr_i)이다. 여기서, $S_i = (s_i^0, s_i^1, ..., s_i^n)$는 m_i의 유한 상태들의 집합이며, s_i^0는 초기상태, T_i는 m_i의 전이들의 집합이고 T_i에 속하는 전이 $t = (s, g, A, s')$는 출발 상태와 도착 상태 s, s' 및 부울식으로 표현되는 가드 조건 g 그리고 액션 집합 A로 구성된다. $scr_i: S_i \rightarrow 2^{Script}$ 는 상태에 스크립트를 사상하는 함수이다.

변수와 이벤트를 포함하는 가드는 아래와 같은 구문으로 정의 된다:

$$G ::= true \mid \neg G \mid G_1 \wedge G_2 \mid e < Exp \mid e = Exp \mid v < Exp \mid v = Exp$$

$$Exp ::= n \mid v \mid Exp_1 \cdot Exp_2,$$

여기서 n은 정수를 대표하고, $v \in V$는 전역변수이며, $\cdot \in \{+, -, \times, /\}$는 이진 연산자를 대표한다. 전이 $t = (s, g, A, s')$의 한 요소를 선택하기 위해서 추출 함수가 다음과 같이 사용된다: $source(t) = s$, $target(t) = s'$, $guard(t) = g$, $action(t) = A$. 또한, 액션 집합 A의 각 원소 $a \in A$들은 아래와 같이 변수의 값 배정이나 출력 이벤트의 방출로 구성 된다:

$$a ::= v := Exp \mid e := Exp.$$

액션에 대해서도 다음과 같은 세 개의 추출 함수가 쓰인다, 이들은 각각변수의 갱신, 출력 이벤트의 방출, 내부 이벤트의 생성에 관한 부분을 추출할 때 사용된다.

$$update(A) = \{ v := Exp \mid \exists v \in V.(v := Exp) \in A \}$$
$$output(A) = \{ e := Exp \mid \exists e \in O.(e := Exp) \in A \}$$
$$signal(A) = \{ e := Exp \mid \exists e \in IT.(e := Exp) \in A \}$$

(그림 2)는 (그림 1)을 평탄화한 모델이다. 그리고 평탄화된 모델을 정형적으로 나타내면 다음과 같다:

I = {coin, ready, stop, time}, O = {game_on, waitGo, waitStop, ringBell, tilt, game_over}
IT = {timeset, timeout, error, exit}, V = {randn, remain}
$M = \{ m_1, m_2, m_3, m_4 \}$, $\gamma(m_1) = \{ m_3, m_4 \}$, $\gamma(m_2) = \gamma(m_3) = \gamma(m_4) = \phi$,
$m_1 = (S_1, s_1^0, T_1, scr_1)$, s_1={GameOff, GameOn},
s_1^0 = GameOff,
T_1 = {(GameOff, coin=1, {game_on=1, timeset=1000}, GameOn), (GameOn, exit=1, {game_over=1, ringBell=1}, GameOff), (GameOn, error=1, {game_over=1, tilt=1}, GameOff}, scr1 = ϕ

4. fFSM의 의미

fFSM의 단계 의미는 입력 이벤트나 내부 이벤트에 의해서 모델의 형상이 변해가는 과정과 동시에 액션의 수행을 묘사하는 것인데, 여기서 액션은 출력 이벤트의 방출과 전역변수의 갱신 그리고 또 다른 내부 이벤트의 생성으로 구성된다.

[정의 8] (형상) Δ는 평탄화된 모델의 전제 형상을 나타낸다. 각 m_i에 대한 형상 집합의 정의는 $\Delta = \{\{ s_1, \ldots, s_n \} \mid \exists s_i \in S_i, 0 < i < n\}$이다. 특히 $\delta_0 \in \Delta$는 초기 형상으로서 $\delta_0 = \{ s_1^0, \ldots, s_n^0 \}$이다.

[정의 9] (활성화된 상태) 상태 s가 형상에서 활성화 되었다는 것은 다음과 같이 정의될 수 있다:

$$\delta \models s \quad iff \quad \forall s' \in \Sigma.\, s \in sub^*(s') \Rightarrow s' \in \delta.$$

[정의 10] (만족성) 어떤 전이가 활성화된 전이(enabled transition)인지를 결정하기 위해, 이벤트의 집합 $E \subseteq 2^I \cup 2^{IT}$과 현재의 형상 δ가 주어졌을 때, 형상 δ와 이벤트 E는 가드 g를 만족한다는 것, $\langle \delta, E \rangle \models guard(t)$을 아래와 같이 귀납적으로 정의한다.

$\langle \delta, E \rangle \models true$ iff $true$
$\langle \delta, E \rangle \models \neg G$ iff $\langle \delta, E \rangle \models G$
$\langle \delta, E \rangle \models G_1 \wedge G_2$ iff $\langle \delta, E \rangle \models G_1$ and $\langle \delta, E \rangle \models G_2$
$\langle \delta, E \rangle \models e < Exp$ iff $e \in E$ and $val(e) < val(Exp)$
$\langle \delta, E \rangle \models e = Exp$ iff $e \in E$ and $val(e) = val(Exp)$
$\langle \delta, E \rangle \models v < Exp$ iff $val(v) < val(Exp)$
$\langle \delta, E \rangle \models v = Exp$ iff $val(v) = val(Exp)$

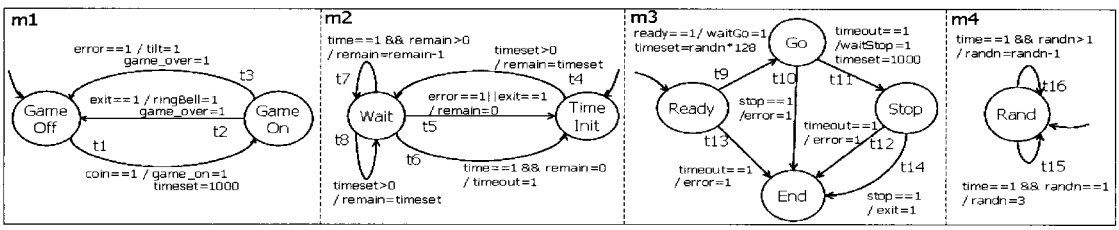
여기서 $val(n) = n$이고, $val(e)$은 이벤트 e의 현재 값을 나타내며, $val(v)$는 변수 v의 현재 값을 나타낸다.

$$val(Exp_1 \cdot Exp_2) = val(Exp_1) \cdot val(Exp_2).$$

[정의 11] (활성화된 전이) 활성화된 상태와 만족성 관계의 정의에 의해서, 각 활성화된 상태에 대해서 활성화된 전이의 집합을 다음과 같이 정의한다.

$$EF = \{ t \mid \forall i \in \{1, \ldots, n\}.\, t \in T_i \wedge \langle \delta, E \rangle \models guard(t) \wedge \delta \models source(t) \}$$

[정의 12] (실행 전이) 실행 전이(executable transition)들의 집합은 충돌이 없는 가장 큰 전이들의 집합이고 반드시 모든 단순 FSM에서 최대 하나의 전이가 포함된다. fFSM 모델은 계층간 전이를 허용하지 않기 때문에, 충돌은 단지 우선순위를 비교할 수 있는 두 전이 사이에서만 발생한다[8, 9].



(그림 2) 평탄화된 상태기계들

$XT = \{t \in ET \mid \neg \exists t' \in ET.source(t) \in sub^{+}(source(t'))\},\ \forall m_i \in M.|XT \cap T_i| \le 1$

[정의 13] (LKS) 단계 의미를 정의하는 프레임워크로서 전이가 레이블된 크립키 구조 LKS(Labeled Kripke Structure)가 사용된다. LKS는 4-튜플 (Q, q_0, R, L)로 구성된다. 여기서, $Q = \{q_0, \ldots, q_n\}$는 LKS의 유한 상태 집합, $q_0 = (\delta_0, \varnothing, c_0)$는 초기상태, $R \subset Q \times 2^{Act} \times Q$는 액션으로 레이블된 전이관계, $L: Q \to 2^{Script}$는 레이블 함수로써 $L(q_i) = \bigcup_{\delta_i \models s} scr(s)$이다.

단계는 큰 걸음인 매크로 스텝(macro step)과 작은 걸음인 마이크로 스텝(micro step)으로 구분해서 설명할 수 있다. 마이크로 스텝은 입력 또는 내부 이벤트에 의해서 한번 전이되는 과정을 말한다. 매크로 스텝은 입력 또는 연속해서 발생되는 내부 이벤트들에 의해서 추가적인 입력 이벤트 없이 더 이상 전이가 없을 때까지의 이루어지는 모든 전이 과정을 의미한다. 따라서 매크로 스텝은 입력 이벤트가 발생된 후 그 다음 입력을 기다리게 될 때까지 발생되는 마이크로 스텝들의 연속을 의미한다.

[정의 14] (마이크로 스텝): 현재 형상 δ와 이벤트의 집합 E가 주어졌을 때, 마이크로 스텝 $(\delta, E, c) \xrightarrow{Act} (\delta', E', c')$은 LKS 상에서 다음과 같이 정의 된다:

$$\delta' = \{s' \mid \forall s \in \delta.\ \exists t \in XT.\ (s = source(t) \Rightarrow s' = target(t)) \vee (s \in sub^{+}(source(t)) \Rightarrow s' = reset(s)) \vee (s \notin sub^{*}(source(t)) \Rightarrow s' = s)\}$$

여기서 $reset(s) = s_i^0,\ s \in sub(s'') \wedge m_i \in \gamma(s'') \wedge s \in S_i$이다.

$$E' = \bigcup_{\forall t \in XT} signal(action(t)),\quad c' = L(q'),$$

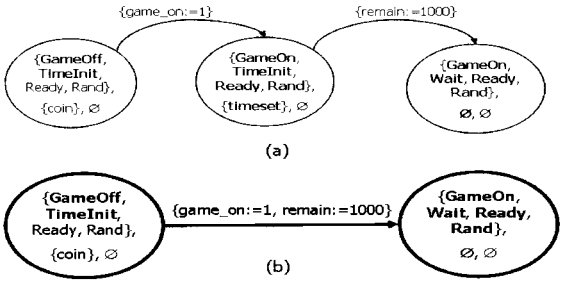
$$Act = \bigcup_{\forall t \in XT} output(action(t)) \cup \bigcup_{\forall t \in XT} update(action(t))$$

[정의 15] (매크로 스텝) 단계 의미는 실행(execution) $q \xrightarrow{Exe} q'$로 정의된다. 하나의 입력 이벤트 집합과 연속해서 발생되는 내부 이벤트들의 집합은 더 이상 내부 이벤트가 없을 때까지 전이를 발생시킨다. 매 마이크로스텝 후에, 델타-지연에 의해서 이전의 이벤트들은 모두 소멸된다. 아래의 정의에서, $k > 1$는 $E_{i,k}$가 $\varnothing$이 되게 하는 첫 번째 k를 의미한다.

그리고 $Exe_i = \bigcup_{\forall 0 < j < k} Act$는 매크로 스텝 동안에 이뤄지는 액션들의 집합을 의미한다.

$$(\delta_i, E_i, c) \xrightarrow{Exe_i} (\delta'_{i+1}, E'_{i+1}, c'_{i+1}) \quad iff$$

$$(\delta_{i,1}, E_{i,1}, c_{i,1}) \xrightarrow{Act_1} \ldots \xrightarrow{Act_{k-1}} (\delta'_{i,k}, \varnothing, c'_{i,k}).$$



(그림 3) *coin* 이벤트에 대한 마이크로 스텝(a)과 매크로 스텝(b)

(그림 3)은 입력 이벤트 coin이 발생했을 때, 시스템이 초기형상 {GameOff, TimeInit, Ready, Rand}로부터 다음 형상 {GameOn, wait, Ready, Rand}로 움직여 나가는 모습을 보여준다. 이제 모델에서 나타날 수 있는 세 종류의 버그(레이스 조건, 모호한 전이, 순환 전이)에 대해서 정의한다.

[정의 16] (레이스 조건) *f*FSM 모델의 실행 동안 출력 이벤트나 변수 상태에 중복해서 쓰기가 일어나는 경우. 아래에서 LHS는 배정문의 왼편을 의미한다.

$$(\delta_{i,1}, E_{i,1}, c_{i,1}) \xrightarrow{Act_1} \ldots \xrightarrow{Act_{k-1}} (\delta_{i,k}, E_{i,k}, c_{i,k}) \xrightarrow{Act_k} \ldots$$

$$\exists_{1 \le j < k}.\ LHS(Act_j) = LHS(Act_k)$$

[정의 17] (모호한 전이) 하나의 상태에 동시에 여러 개의 전이가 활성화 될 경우.

$$\exists t, t' \in XT.source(t) = source(t')$$

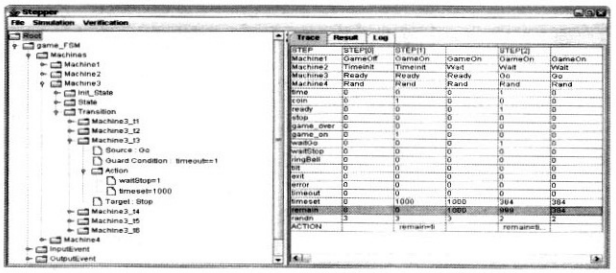
[정의 18] (순환 전이) 연속되는 내부 이벤트에 의해서 순환하는 전이가 발생될 경우.

$$(\delta_{i,1}, E_{i,1}, c_{i,1}) \xrightarrow{Act_1} \ldots \xrightarrow{Act_{k-1}} (\delta_{i,k}, E_{i,k}, c_{i,k}) \xrightarrow{Act_k} \ldots$$

$$\exists_{1 < j < k}.\ \delta_{i,j} = \delta_{i,k} \wedge Act_j = Act_k \wedge E_{i,j} = E_{i,k}$$

〈표 1〉 레이스 조건의 위반

events	Configuration	Actions
{time, ready}	{GameOn, Wait, Ready, Rand}	{waitGo:=1, remain:=0}
{timeset, timeout}	{GameOn, TimeInit, Go, Rand}	{waitStop:=1, remain:=randn*128}
{timeset}	{GameOn, Wait, Stop, Rand}	{ remain:=1000}
	{GameOn, Wait, Stop, Rand}	



(그림 4) 시뮬레이션을 이용한 레이스 조건 위반 검출

예를 들어, 만일 현재의 형상이 {GameOn, Wait, Ready, Rand} 이고, 발생된 사용자 이벤트와 시스템 이벤트가 각각 *ready* 와 *time*일 때, 그리고 변수 상태 *remain* 의 값이 0일 때, 드문 경우이기는 하지만, 출력 이벤트와 변수 상태의 값이 한 실행 동안 지속되어야 한다는 제약사항을 위반(레이스 조건 위반)하게 된다. <표 1>을 보면 이와 같은 조건에서 *remain* 값이 여러 번 변경됨을 알 수 있다.

우리는 *f*FSM 모델에 대한 버그 검출을 위해 이 논문에서 제안한 단계의미를 적용해서 시뮬레이터를 구현하였다. 아래 (그림 4)는 시뮬레이터를 이용해서 <표 1>에서 예시된 레이스 조건 위반을 검출하는 과정을 보여준다. STEP[0]은 시스템이 초기 형상으로 초기화 되어 있음을 나타낸다. STEP[1]은 *coin* 이벤트가 발생된 후, 내부 이벤트 *timeset*이 연속해서 발생된 것을 나타낸다. 이때의 형상이 <표 1>에서와 같이 {GameOn, Wait, Ready, Rand}이고 *ready* 와 *time* 이벤트가 동시에 발생하면 한 매크로 스텝 내에서 *remain* 값이 두 번 이상 변경되므로 시뮬레이터는 레이스 조건 위반을 검출하게 된다.

5. 결론과 향후 연구

복잡한 임베디드 시스템을 설계하는 방법으로, 하드웨어/소프트웨어 통합설계가 제품의 생산성을 높일 새로운 방법론으로 주목 받고있다. PeaCE[1]는 하드웨어/소프트웨어 통합 설계 프레임워크로써 데이터 흐름과 제어 흐름을 표현하는 모델을 가지고 있다. *f*FSM는 PeaCE의 제어 흐름을 표현하는 모델이다. 그러나 그 정형성의 부재로 명세를 검증하는데 어려움이 있었다.

본 논문에서는, 통합설계 프레임워크에서 코드생성의 신뢰성을 높이고 제어 모델의 정형검증을 가능하게 하기 위해, *f*FSM 모델의 단계의미를 정의하였다. *f*FSM의 마이크로와 매크로 스텝이 정의되었고 이것은 입력 이벤트에 의해서 모델의 형상이 어떻게 변해 가며 어떠한 액션을 수행하는지를 묘사하는 것이었다. 또한 우리는 세 가지의 버그(레이스 조건, 모호한 전이, 순환 전이)를 정형적으로 정의했다. 그 결과 해당 버그를 정형적으로 검사할 수 있게 되었다. 현재 우리는 본 논문에서 정의된 *f*FSM의 단계 의미를 기반으로 *f*FSM 전용의 모델체커를 개발하고 있다.

참 고 문 헌

[1] D. Kim, S. Ha, "Static Analysis and Automatic Code Synthesis of flexible FSM Model", ASP-DAC 2005 Jan 18-21, 2005

[2] D. Harel, "Statecharts: a visual formalism for complex systems", Sci. Comput. Program, Vol.8, pp.231-274, 1987.

[3] A. Pnueli and M. Shalev. "What is in a step: On the semantics of Statecharts" In TACS '91. Vol.526 of LNCS, pp.244-264. Springer-Verlag, 1991.

[4] E. M. Clarke, O. Grumberg and D. Peled, Model Checking, MIT Press, 1999.

[5] D. Kim, "System-Level Specification and Co-simulation for Multimedia Embedded Systems," Ph.D. Dissertation, Computer Science Department, Seoul National University, 2004.

[6] J. B. Lind-Nielsen, "Verification of Large State/Event Systems," Ph.D. Dissertation, Department of Information Technology, Technical University of Denmark, 2000.

[7] E. Mikk, Y. Lakhnech, M. Siegel, "Hierarchical automata as model for statecharts", LNCS Vol.1345, Proceedings of the 3rd ACSC, pp.181-196, 1997.

[8] J. Lind-Nielsen, H. R. Andersen, H. Hulgaard, G. Behrmann, K. J. Kristoffersen, K. G. Larsen, "Verification of Large State/Event Systems Using Compositionality and Dependency Analysis", FMSD, pp.5-23, 2001.

[9] D. Harel, A. Naamad, "The STATEMATE semantics of statecharts", ACM Transactions on Software Engineering Methodology, 5(4), October, 1996.

박 사 천

e-mail : sachem@kyonggi.ac.kr
2001년 경기대학교 전자계산학과(학사)
2003년 경기대학교 전자계산학과 (이학석사)
2004년~현재 경기대학교 전자계산학과 박사과정
관심분야 : 모델 체킹, 정형기법, 소프트웨어 공학 등

권 기 현

e-mail : khkwon@kyonggi.ac.kr
1985년 경기대학교 전자계산학과(학사)
1987년 중앙대학교 전자계산학과(이학석사)
1991년 중앙대학교 전자계산학과(공학박사)
1998년~1999년 독일드레스덴 대학 전자계산학과 방문교수
1999년~2000년 미국 카네기 멜론 대학 전자계산학과 방문교수
1991년~현재 경기대학교 정보과학부 교수
관심분야 : 소프트웨어 모델링, 소프트웨어 분석, 정형 기법 등

하 순 회

e-mail : sha@iris.snu.ac.kr
1985년 서울대학교 컴퓨터공학과(학사)
1987년 서울대학교 컴퓨터공학과(공학석사)
1992년 University of California, Berkeley (PhD in Electrical Engineering and Computer Science)
1993년~1994년 현대전자근무
1994년~현재 서울대학교 컴퓨터공학과 교수
관심분야 : Hardware-software codesign, design methodology for embedded system