SMV를 이용한 확장된 유한상태 기계의 정형 검증

조민택°, 박사천, 권기현
경기대학교 전자계산학과
{tesnoi°, sachem, khkwon}@kyonggi.ac.kr

Formal Verification of the Extended Finite State Machine with SMV

Mintaek Cho°, Sachon Park, Gihwon Kwon
Computer Science Department, Kyonggi University

요 약

유한상태 기계는 신뢰성이 요구되는 내장형 시스템의 제어흐름을 표현하고 검증하는데 많이 사용되는 모델이다. 하지만 자체가 가지고 있는 단순함으로 인해 복잡한 시스템을 명세하기에는 부족하다. 이러한 유한상태 기계의 단점을 극복하기 위해 다양하게 확장시킨 유한상태 기계들이 나왔지만 이렇게 확장된 유한상태 기계들에 대한 정형 의미의 부재로 인해서 요구사항중 하나인 명세를 검증하는데 어려움이 따른다. 이에 우리는 확장된 유한상태 기계의 정형 단계 의미를 정의하고, 이를 사용하여 모델에 대한 정형검증을 수행하였다. 그 결과 레이스 조건(race condition)과 애매한 전이, 순환하는 전이 등의 버그들을 모델에서 정형적으로 검출 할 수 있었다.

1. 서 론

유한상태 기계는 신뢰성이 요구되는 내장형 시스템의 제어흐름을 표현하고 검증하는데 많이 사용되는 모델이다. 이러한 유한상태기계(Finite State Machine)는 순차적인 동작을 묘사하기에는 적합하지만 병렬성 이나 메모리를 사용하는 시스템과 같이 복잡한 시스템을 명세하기가 힘들다. 이에 우리는 병렬성과 변수를 추가하여 확장한 유한상태 모델인 *p*FSM의 정형 의미를 정의하고 이를 모델체킹 한다. 이 모델은 메모리를 표현하기 위해 변수를 사용하며, 의미적으로 계층성을 포함한다.

*p*FSM에서는 행위를 *N*개의 평탄화한 병렬 기계로 설명하는데 여기서 *N*은 상태도로 보면 *OR*-상태의 수에 해당한다. 따라서 형상(configuration)은 정확히 *N*개로 구성되고 그렇기에 형상에 비활성화된 상태들도 포함하게 된다. 이렇게 표현된 모델은 기호적 모델검사와 상태폭발 방지 기법 중 하나인 결합 모델체킹(compositional model checking)에 용이하다. 이러한 *p*FSM의 특징을 기호적 모델검사 도구인 SMV로 변환하여 *p*FSM을 정형검증 하였다.

모델체킹은 유한 상태 모델 M과 명제 시제 논리식 ϕ을 받아서 만족성 여부 $M \models \phi$를 결정한다[1]. SMV는 유한 상태 모델 M과 시제 논리식으로 CTL 식을 사용하여 만족성 여부를 검사하게 된다. 우리의 연구는 이러한 *p*FSM의 정형의미를 정의하고 이를 사용하여 *p*FSM을 SMV로 변환하는 방법과 필요한 속성의 CTL 식에 대해서 다룬다.

본 논문의 구성은 다음과 같다. 2장에서는 *p*FSM의 정형구문과 의미를 정의하고, 3장에서 *p*FSM이 SMV로 변환되는 규칙 및 주요 속성의 CTL 식을 소개하고, 이를 사용한 검증을 다룬다. 4장에서는 결론을 맺고 향후 연구 주제에 대해 언급한다.

2. *p*FSM의 구문과 의미

우리는 각 FSM을 평탄화한 시킨 *N* 병행기계인 *p*FSM을 제안한다. 여기서 *N*은 *p*FSM에 포함된 FSM의 수이다. *p*FSM은 이벤트와 변수상태, 상태와 전이가 존재한다. *I, O, IT* 는 각각 입력 이벤트, 출력 이벤트, 내부 이벤트의 집합이다. 이 세개의 집합은 서로 소인 집합으로 정의했다. $I \cup O \cup IT = \{e_1, ..., e_n\}$에 있는 각 이벤트 e_i는 도메인 D_i와 초기값 d_i로 구성되고, $val(e_i)$은 그 이벤트의 현재 값을 의미한다. 단순(*Simple*) FSM은 4개의 튜플 (S, s^0, T, scr)로 정의 되는데, 여기서 S 는 상태들의 집합이고 s^0 는 초기 상태이며, T 는 전이들의 집합이다. $scr : S \rightarrow 2^{Script}$ 는 각 상태에 스크립트를 레이블 하는 함수로 정의 된다.

정의 1 (***parallel* FSM**) $pFSM = (I, O, IT, M, \gamma, V)$, *I, O, IT* 이벤트들의 집합이다. V 전역변수의 집합이고, $v_j(D_j, d_j)$, $M = \{m_1, ..., m_n\}$ 는 단순 FSM의 집합이다. $\Sigma = \bigcup_{i=1}^{n} S_i$ 은 M 에 속한 모든 상태들의 집합이다. 계층 관계 는 상태를 그 상태가 포함하는 상태 기계들로 사상하는 함수이다: $\gamma : \Sigma \rightarrow 2^M$.

$sub : \Sigma \rightarrow 2^{\Sigma}$일 때, $sub(s) = \{s' | M_i \in \gamma(s) \wedge s' \in S_i\}$는 어떤 상태에 포함된 하위 상태들을 돌려주는 함수이다. sub^+ 은sub 의 추이 클로우져(transitive closure)이고 sub^*는 sub의 반사적 추이 클로우져(reflexive transitive closure)이다.

정의 2 (***simple* FSM**) $m_i = (S_i, s_i^0, T_i, scr_i)$, $S_i = \{s_i^0, s_i^1, ..., s_i^n\}$는$m_i$의 유한 상태들의 집합 s_i^0는 초기상태, T_i은 m_i의 전이들의 집합이고 전이 $t \in T_i = (s, g, A, s')$는 출발 상태, 도착 상태 s, sS_i와 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 같이 간단한 구문으로 정의된다.

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

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

여기서 n 은 정수를 대표하고, $v \in V$ 는 전역변수이며, $\bullet \in \{+, -, \times, /\}$는 이진 연산자를 대표한다. 전이 $t = (s, g, A, s')$의 한요소를 선택하기 위해서, 추출 함수(*projection function*)가 사용된다 : *source(t) = s, target(t) = s, guard(t) = g, action(t) = A*. 또한, 액션의 집합A의 각 원소 a는 아래와 같이 변수의 값 배정이나 출력 이벤트의 방출로 구성된다: a ::= *v:=Exp* | *e:=Exp*.

액션에 대해서도 다음과 같은 세 개의 추출 함수가 쓰인다, 이들은 각각 변수의 갱신, 출력 이벤트의 방출, 내부 이벤트의

* This work was supported in part by IT Leading R&D Support Project funded by Ministry of Information and Communication, Republic of Korea.

생성에 관한 부분을 추출할 때 사용된다.

$update(A) = \{v := Exp \mid \exists v \in V.(v := Exp) \in A\}$

$output(A) = \{e := Exp \mid \exists e \in O.(e := Exp) \in A\}$

$signal(A) = \{e := Exp \mid \exists e \in IT.(e := Exp) \in A\}$

정의 4 (형상) Δ는 *p*FSM 모델의 전제 형상을 나타낸다. *p*FSM의 각 m_i 에 대해서, 형상 집합의 정형적 정의는 $\Delta = \{\{s_1, \ldots, s_n\} \mid \exists s_i \in S_i, 0 < i \le n\}$ 가 된다. $\delta_0 \in \Delta$, $\delta_0 = \{s_1^0, \mathrm{K}, s_n^0\}$ 는 초기 형상이다.

정의 5 (활성화된 상태) 상태 s가 형상 에서 활성화 되었다는 것은 다음과 같이 정의될 수 있다: $\delta \models s$ *iff* $\forall s' \in \sum, s \in sub^*(s') \Rightarrow s' \in \delta$.

정의 6 (만족성) 어떤 전이가 활성화된 전이(enabled transition)인지를 결정하기 위해, 이벤트의 집합 $E \subseteq 2^V \cup 2^{IT}$ 과 현재의 형상가 주어졌을 때, 형상 δ 와 이벤트 E 는 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* $not \langle \delta, E \rangle \models G$

$\langle \delta, E \rangle \models G_1 \wedge G_2$ *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. $val(Exp_1 \bullet Exp_2) = val(Exp_1) \bullet val(Exp_2)$.

정의 7 (활성화된 전이) 활성화된 상태와 만족성 관계(satisfiability relation)의 정의에 의해서, 각 활성화된 상태에 대해서 활성화된 전이의 집합을 다음과 같이 정의한다.

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

정의 8 (실행 전이) 실행 전이(executable transition)들의 집합은 충돌이 없는 가장 큰 전이들의 집합이고 반드시 모든 단순 FSM에서 최대 하나의 전이가 포함된다.

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

정의 9 (LKS) *p*FSM 의 단계의미는 전이가 레이블 되는 크립키 구조인 *LKS*(Labeled Kripke Structure)로 정의된다. $LKS = (Q, q_0, R, L)$ 에서 $Q = \{q_0, \ldots, q_n\}$ 는 LKS의 유한 상태 집합, $q_0 = (\delta_0, \varnothing, c_0)$ 는 초기상태, $R \subseteq Q \times 2^{Act} \times Q$ 는 액션으로 레이블된 전이관계, $L : Q \to 2^{Script}$ 는 레이블 함수로써 $L(q_i) = \bigcup_{\forall s \in \delta} scr(s)$ 이다.

정의 10 (마이크로 스텝) $(\delta, E, c) \xrightarrow{Act} (\delta', E', c')$

현재 형상 와 이벤트의 집합 E가 주어졌을 때, 마이크로 스텝은 LKS 상에서 다음과 같이 정의된다:

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

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

$$E' = \bigcup_{\forall e \in XT} signal(action(t)),$$

and $c' = L(q')$,

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

정의 11 (매크로 스텝) *p*FSM의 단계 의미는 실행(*execution*)이라고 불리우는 $q \xrightarrow{Exe} q'$ 로써 정의된다. 하나의 입력 이벤트 집합과 연속해서 발생되는 내부 이벤트들의 집합은 더 이상 내부 이벤트가 없을 때까지 전이를 발생시킨다. 매 마이크로 스텝 후에, 델타-지연에 의해서 이전의 이벤트들은 모두 소멸된다. 아래의 정의에서, $k > 1$ 는 $E_{i,k}$ 가 이 되게 하는 첫 번째 k를 의미한다. 그리고 $Exe_i = \bigcup_{\forall 0<j<k} Act_j$ 는 매크로 스텝 동안에 이뤄지는 액션들의 집합을 의미한다.

$$(\delta_i, E_i, c_i) \xrightarrow{Exe_i} (\delta_{i+1}, \varnothing, c_{i+1}) \text{ iff } (\delta_{i,1}, E_{i,1}, c_{i,1}) \xrightarrow{Act_1} \Lambda \xrightarrow{Act_{k-1}} (\delta_{i,k}, \varnothing, c_{i,k}).$$

3. SMV를 사용한 정형 검증

3.1 *p*FSM 모델을 SMV로 변환

모델 체킹을 하기 위해서 상태도를 모델 체커의 입력언어로 모델을 변환하는 연구는 지금껏 많이 수행되어왔다. 우리는 *p*FSM의 모델 체킹을 위해서 Chan[2]과Clarke[3]의 방식에서 많은 힌트를 얻었다. SMV의 변수들은 모두 병렬적으로 동작하므로 *p*FSM의 머신과 상태들은 SMV의 변수와 변수 값으로 자연스럽게 변환할 수 있다.

변환규칙 1 (머신과 상태) $pFSM = (I, O, IT, M, \gamma, V)$일 때, M에 속한 머신 $m_i = (S_i, s_i^0, T_i, scr_i)$에 대해서 해당 상태들은 $VAR\ m_i : S_i$ 로 변환된다.

변환규칙 2 (전이) 전이는 *p*FSM에서 정의하고 있는 전이 조건 $\langle \delta, E \rangle \models guard(t) \wedge \delta \models source(t)$에 의하여 SMV로 변환한다. 즉, $m_i = (S_i, s_i^0, T_i, scr_i)$와 $t \in T_i$에 대해서 어떤 상태를 포함하고 있는 상위 상태들을 돌려주는 함수 $up(s) = \{s' \mid M_i \in \gamma(s') \wedge s \in S_i\}$는 이고, up^*는 up의 반사적 추이 클로우져일 때, 각각의 전이들은 다음과 같이 변환된다: $DEFINE\ m_i\_t := up^*(source(t)) \,\&\, guard(t)$.

변환규칙 3 (동기화 가정) 동기화 가정을 SMV에서 표현하기 위해 안정한 상태를 의미하는 특별한 이진 변수 *stable*을 추가한다. 안정한 상태는 내부 이벤트 {$internal_1$, , $internal_l$}가 발생하지 않는 상태로 간단히 정의할 수 있다.

```
ASSIGN  next(stable) := case
        (internal1=0) &  & (internall=0) : 1;
        1 : 0;
        esac;
```

변환규칙 4 (입력 이벤트) 초기값이 d인 입력 이벤트는 e의 도메인 D이고 도메인의 최소값과 최대값이 각각 min_e, max_e일 때, 다음과 같이 변환된다.

```
ASSIGN  init(e) := d
        next(e) := case
        stable : mine .. maxe
        1 : 0;
        esac;
```

변환규칙 5 (출력 이벤트) 어떤 전이 $t = (s, g, A, s')$이고, $action(t) = A$ 일 때, 초기값이 d를 갖는 출력 이벤트 e에 대해서 $output(A) = \{e := Exp \mid \exists e \in O.(e := Exp) \in A\}$을 만족하는 모든 전이 집합 {$t_1$, , t_n}과 각 식의 집합 {Exp_1, , Exp_n} 대

해서 다음과 같이 변환한다.

```
ASSIGN  init(e) := d
        next(e) := case
          t1 : Exp1
             &
          tn : Expn
           1 : 0;
        esac;
```

이벤트의 변환은 델타-지연을 고려해서 전이가 발생되지 않으면 0으로 값을 되돌리도록 했고, 변수에 대한 변환은 지속적으로 값이 유지되도록 했다.

3.2. 속성 표현 및 검증

*p*FSM은 내장형 시스템의 제어흐름을 명세하기 위해서 고안되었다. 내장형 시스템에서 만족해야 하는 속성에 대해서는 [4]에서 자세히 설명하고 있으며, 표 1은 *p*FSM에 적용할 속성들에 대한 CTL 표현을 보여준다.

표 1. 속성에 대한 CTL 식

속 성	CTL 식
No unused components	**EF** $component_1$ ∧...∧ **EF** $component_n$
No unreachable guard	**EF** t_1 ∧...∧ **EF** t_n
No ambiguous transitions	**AG** $\neg((t_1 \wedge t_2) \vee (t_2 \wedge t_3) \vee (t_1 \wedge t_3))$
No deadlocks	**EF AG** *Deadlock(pT)*,
No divergent behavior	**AG**($\neg$*stable* ⇒ **A**[$\neg$*stable* **U** *stable*])
Race condition violation	**AG** ((*update*(*v*) ∧ $\neg$*stable*) **AX**(**A**[*update*(*v*) **U** *stable*])), **AG** ((*emit*(*o*) ∧ $\neg$*stable*) **AX**(**A**[$\neg$*emit*(*o*) **U** *stable*]))

표 1의 첫 번째 항목은 *p*FSM의 모든 컴포넌트들이 사용되는지를 알아보는 속성이다. 여기서 컴포넌트는 상태와 이벤트가 된다. 두 번째 항목은 모든 가드조건이 만족하고 있는지 알아보는 속성이다. 가드조건이 만족하면 *p*FSM에서 전이가 발생되는 것으로 볼 수 있기 때문에 모든 전이 $pT=\{t_1, ..., t_n\}$로 도달하는지 검사함으로 알 수 있다. 세 번째 항목은 하나의 상태에서 두 개 이상의 전이가 동시에 발생되지 말아야하는 상황을 말하며, 위의 예는 어떤 상태 s의 전이 집합이 $\{t_1, t_2, t_3\}$일 경우를 나타낸다. 네 번째는 데드락을 검사하는 항목으로 모든 전이가 더 이상 활성화 되지 않는 상태가 영원히 지속되고 있는지를 확인하기 위한 CTL 식이다. 다섯 번째 항목은 외부 이벤트에 의해서 전이가 발생한 후 계속되는 내부 이벤트에 의해서 시스템이 안정화되지 않는 경우를 말한다. *stable*은 변환규칙 3번의 변수를 의미한다. 마지막 항목은 한 매크로 스텝 안에서 변수 값이 두 번 이상 변경되거나, 출력 이벤트가 두 번 이상 방출될 때를 말한다. *update*(*v*)는 시스템 내부의 변수 *v*의 변경 여부를 식별하는 함수이고, *emit*(*o*)는 출력 이벤트 *o*가 방출 되는 것을 의미한다.

지금까지 다룬 변환 규칙과 CTL 식을 사용하여 *p*FSM 모델을 검증한 결과 예제로 사용된 *p*FSM에서 변수가 중복 사용된 Race condition 위반을 검출 할 수 있었다. 그림 1은 앞서 다룬 변환규칙을 적용시켜 모델로 바꾸는 과정과 SMV를 호출하여 모델 검사를 수행하는 일련의 과정을 자동화 시킨 툴을 사용하여 버그를 찾아낸 것을 보여주고 있다.

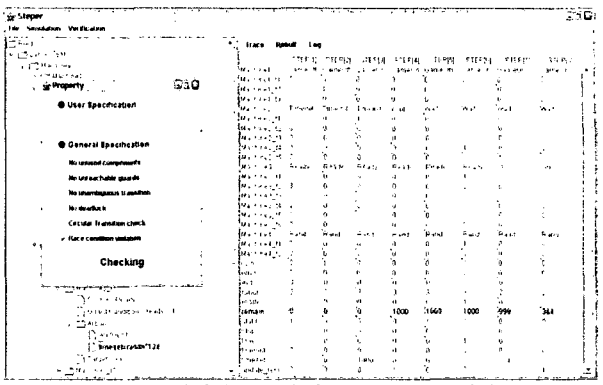

그림 1. 모델체킹를 이용한 레이스 조건 위반 검출

4. 결 론

본 논문에서는 내장형 시스템의 제어흐름 명세하기위해 유한상태 기계에 병렬성과 변수를 추가하여 확장한 유한상태 기계인 *p*FSM을 정의하였다. 이 모델은 메모리를 표현하기 위해 변수를 사용하며, 의미적으로 계층성을 포함한다.

*p*FSM의 정형 검증을 위해 *p*FSM의 정형구문과 의미를 정의하였고 *p*FSM을 SMV로 정형 검증하기 위해서 *p*FSM을 SMV의 입력언어로 변환시키는 규칙을 정의하였다. 또한 내장형 시스템을 검증하기 위해 필요한 속성들을 CTL 식으로 정의하였다.

우리는 내부적으로 SMV를 호출하여 모델 검사의 모든 과정을 자동화 시켜주는 도구를 개발하였으며, 향후 연구는 *p*FSM 모델을 위한 전용의 모델 체킹 도구를 개발하는 것이다.

참고문헌

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

[2] W. Chan, "Symbolic Model checking for Large software Specification", Dissertation, Computer Science and Engineering at University of Washington, pp. 13-32, 1999.

[3] E. M. Clarke, W. Heinle, "Modular translation of Statecharts to SMV", Technical Report CMU-CS-00-XXX, CMU School of Computer Science, August 2000.

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

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

[6] 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.

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

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