SMV를 이용한 유한 상태 기계의 동치 검사
(Equivalence Checking of Finite State Machines with SMV)

권 기 현 † 엄 태 호 ††
(Gihwon Kwon) (Tae-Ho Eom)

요 약 본 연구에서는 유한 상태 기계들 간의 동치 여부를 검증하고자 한다. 즉 모든 입력에 대하여 유한 상태 기계의 반응이 항상 동일한지를 판정하고자 한다. 만약 두 개의 유한 상태 기계가 동치라고 판정된다면, 복잡한 유한 상태 기계는 단순한 기계로 대치될 수 있다. 또한 명세와 구현이 모두 유한 상태 기계로 표현된 경우, 동치 검사를 이용해서 구현이 명세를 만족하는지 결정할 수 있다. 본 논문에서는 이와 같은 유한 상태 기계의 동치 검사를 모델 검사 기법으로 다음과 같이 해결한다. 주어진 유한 상태 기계 M_A와 M_B를 조합하여 모델 $M = M_A \times M_B$을 구축하고, 검사할 동치 조건을 시제 논리식 ϕ로 기술한다. 만일 모델이 시제 논리식을 만족한다면($M \models \phi$) 두 기계는 동치이다. 그렇지 않다면 두 기계는 비동치이며 그 이유를 설명하는 반례를 제공한다. 전 과정이 자동화되었으며, 여러 개의 사례 연구에 적용한 결과 만족할 만한 결과를 얻었다.

키워드 : 유한 상태 기계, 동치 검사, 곱 기계, 모델 검사, 시제 논리

Abstract In this paper, we are interested in checking equivalence of FSMs(finite state machines). Two FSMs are equivalent if and only if their responses are always equal with each other with respect to the same external stimuli. Equivalence checking FSMs makes complicated FSM be substituted for simpler one, if they are equivalent. We can also determine the system satisfies the requirements, if they are all written in FSMs. In this paper, we regard equivalence checking problem as model checking one. For doing so, we construct the product model $M = M_A \times M_B$ from two FSMs M_A and M_B. And we also get the temporal logic formula ϕ from the equivalence checking definition. Then, we can check with model checker whether M satisfies ϕ, written $M \models \phi$. Two FSMs are equivalent, if $M \models \phi$. Otherwise, it is not equivalent. In that case, model checker generates counterexamples which explain why FSMs are not equivalent. In summary, we solve the equivalnce checking problem with model checking techniques. As a result of applying to several examples, we have many satisfiable results.

Key words : Finite state machine, Equivalence checking, Product machine, Model checking, Temporal logic

1. 서 론

유한 상태 기계는 시스템의 행위를 모델하는 모델링 언어이다 [1, pp.358-404]. 다른 모델에 비해서 단순하고 이해하기 쉬울 뿐만 아니라, 타겟 언어의 코드로 쉽게 변환될 수 있다. 이러한 이점 때문에 반응형 시스템 및 실시간 시스템의 행위를 기술하는데 널리 사용되고 있으며, 표현력을 높이기 위해 유한 상태 기계를 확장한 다양한 모델링 언어들이(예를 들어 Statechart [2], RSML [3], SCR [4]) 산업 현장에서 활발히 사용중이다.

유한 상태 기계에 대한 대표적인 검사로 동치 검사가 있다. 동치 검사란, 모든 입력에 대해서 주어진 두 유한 상태 기계의 출력이 항상 동일한지를 판정하는 것이다 [5]. 예를 들어, 그림 1에 있는 유한 상태 기계 M_A와 M_B에 101110이 입력되면 두 기계의 출력은 000010 으로 동일하다. 어떠한 입력을 주더라도 두 기계의 출력이 항상 동일하기 때문에 M_A와 M_B는 동치이다.

이와 같은 동치 검사는 시스템 최적화 및 정확성 확인 등에 활용될 수 있다[6]. 만약 두 개의 유한 상태 기계가 동치라고 판정된다면, 복잡한 유한 상태 기계는

· 본 연구는 전자통신연구원 위탁과제(3010-2002-0090) 연구비 지원으로 수행되었음
† 종신회원 : 경기대학교 정보과학부 교수
khkwon@kuic.kyonggi.ac.kr
†† 비 회 원 : 경기대학교 전자계산학과
ever9@kyonggi.ac.kr
논문접수 : 2002년 11월 29일
심사완료 : 2003년 3월 19일

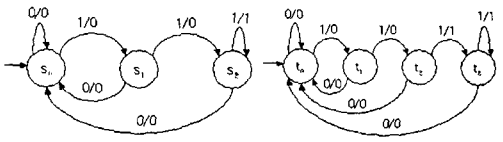

그림 1 예제: 유한 상태 기계 M_A, M_B

보다 단순한 기계로 대치될 수 있다. 또한 명세와 구현이 모두 유한 상태 기계로 표현된 경우, 동치 검사를 이용해서 구현이 명세를 만족하는지를 결정할 수 있다.

본 논문에서는 하드웨어 및 소프트웨어 검증에 널리 사용되고 있는 모델 검사 기법을 이용하여 유한 상태 기계의 동치 검사를 다음과 같이 수행한다. 먼저, 주어진 유한 상태 기계 M_A와 M_B를 조합하여 $M = M_A \times M_B$을 구축하고, 동치가 되기 위해 반드시 만족해야할 조건을 시제 논리식 ϕ로 표현한다. 그런 후, 모델 검사를 이용하여 모델 M이 시제 논리식 ϕ를 만족하는지를 검사한다[7]. 만약 $M \models \phi$[1]이면 두 기계는 동치이고 (그 이유에 대한 상세한 설명은 3장을 참조하기 바람), 그렇지 않다면 두 기계는 비동치이며 모델 검사에 의해서 생성된 반례를 통하여 비동치의 원인을 파악할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 동치 검사와 관련된 기본 개념을 다루며, 3장에서는 동치 검사 문제를 모델 검사 문제로 변환하는 과정을 기술한다. 4장에서는 유한 상태 기계를 SMV 언어로 변환하는 규칙과 시스템 구현을 설명하며 그리고 본 방법의 효과를 분석한다. 마지막으로 5장에서 결론을 맺는다.

2. 동치 검사 문제

유한 상태 기계 M_A, M_B가 동치라면 $M_A \equiv M_B$라고 표시하자. 본 절에서는 $M_A \equiv M_B$를 판정하는 동치 검사의 원리를 설명하고 필요한 기본 개념을 기술한다.

정의 1: 유한 상태 기계는 다음과 같이 6-튜플 (I, O, S, d, δ, λ)로 정의된다.

- I 는 입력 집합이다.
- O 는 출력 집합이다.
- S 는 상태 집합이다.
- $d \in S$는 초기(default) 상태이다.
- $\delta : I \times S \rightarrow S$는 천이 함수이다. 다음 상태 $s' \in S$는 입력 $i \in I$와 현재 상태 $s \in S$에 의해서 결정된다. 즉 $\delta(i, s) = s'$이다.
- $\lambda : I \times S \rightarrow O$는 출력 함수이다. 출력 $o \in O$는 입력과 현재 상태에 의해서 결정된다. 즉 $\lambda(i, s) = o$ 이다. ■

만일 유한 상태 기계들의 입력과 출력 공간이 동일하다면, 두 기계의 동치 여부를 검사하는 것이 가능하다. 동치 검사를 위해서 두 기계를 조합한 곱 기계를 아래와 같이 구성한다.

정의 2: 두 개의 유한 상태 기계 $M_A = (I_A, O_A, S_A, d_A, \delta_A, \lambda_A)$ 와 $M_B = (I_B, O_B, S_B, d_B, \delta_B, \lambda_B)$ 가 주어졌을 때, 곱 기계 $M = M_A \times M_B = (I, O, S, d, \delta, \lambda)$ 는 다음과 같다.

- $I = I_A = I_B$ 입력 집합이다.
- O = {0,1} 은 출력 값이다. 단 $O_A = O_B$ 이다.
- $S = S_A \times S_B = \{(s_A,s_B) \mid s_A \in S_A, s_B \in S_B\}$는 상태 쌍들의 집합으로서 두 기계에 있는 상태들의 곱이다. s = (s_A,s_B)는 곱 기계의 임의의 상태를 나타낸다.
- $d \in S$은 곱 기계의 초기 상태를 나타낸다. 여기서 $d = (d_A,d_B)$ 이다.
- $\delta : I \times S \rightarrow S$는 천이 함수로서, 다음 상태는 입력과 현재 상태에 의해서 결정된다. 즉 $\delta(i,(s_A,s_B)) = (\delta_A(i,s_A), \delta_B(i,s_B))$이다.
- $\lambda : I \times S \rightarrow O$는 출력 함수로서, 상태 s에서 입력 i에 대해서 두 기계의 출력 값이 같으면 1을, 그렇지 않으면 0을 출력한다. 즉 $\lambda(i,s) = \lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B)$이다. ■

정의에서 보듯이, 출력 함수 λ는 곱 기계의 상태 s에서 하나의 입력 i에 대해서 두 기계의 출력이 동일한지를 판정한다. 상태 s에서 모든 입력에 대해 두 기계의 출력이 동일한지를 판정하는 것은 다음과 같다.

정의 3: $E(s)$는 곱 기계의 상태 s = (s_A,s_B)에서 모든 입력 값 i에 대해서 두 기계의 출력이 같음을

$$E(s) = \forall i \cdot \lambda_A(i,sA) \Leftrightarrow \lambda_B(i,s_B)$$

나타내는 술어이다. ■

두 기계가 동치이기 위해서는, 곱 기계의 초기 상태로부터 도달 가능한 모든 상태에서 두 기계의 출력이 동일해야 한다. 즉 도달 가능한 모든 상태에서 $E(s)$가 만족되어야 한다. 따라서 동치 관계의 정의는 다음과 같다.

정의 4: 곱 기계의 초기 상태 d로 부터 도달 가능한 모든 상태 집합을 $P(d)$ 라고 하자. 동치이기 위해서는, 도달 가능한 모든 상태에서 두 기계의 출력이 동일해야 한다. 즉,

$$\exists d \in S \cdot \forall s \in S \cdot s \in P(d) \Rightarrow E(s)$$

---

1) 모델 검사에서는 'M이 ϕ를 만족한다'를 $M \models \phi$로 표기한다. 여기서 기호 $\models$ 는 '만족성 관계'를 나타낸다.

이다. ■

그러므로 곱 기계의 초기 상태 d로 부터 도달 가능한 상태들을 방문해 가면서, 매 상태마다 $E(s)$가 만족되는지를 검사함으로서 동치 여부를 판정할 수 있다. 동치 검사를 수행하기 위해서 현재 상태에서 도달 가능한 다음 상태들의 집합을 계산해야 한다. 이것을 상 계산(image computation) 이라고 한다[7]. 상 계산은 현재 상태에서 한 번의 전이를 통해서 이동할 수 있는 다음 상태들의 집합을 구하는 작업이다. 초기 상태 d로 부터 고정점(fixed point)에 도달할때까지 반복적으로 상 계산을 적용하면, 도달 가능한 상태 집합 $P(d)$를 얻을 수 있다. 새로운 상태를 구할 때 마다 $E(s)$를 검사한다. $E(s)$를 만족하지 않는 상태가 하나라도 존재하면, 두 기계는 비동치이기 때문에 반례를 출력하고 즉시 중지한다. $E(s)$를 만족한다면, 고정점에 도달할 때 까지 이 과정을 반복 수행한다. 고정점에 도달할 때까지 모든 상태가 $E(s)$를 만족하면 두 기계는 동치이다.

3. 모델 검사를 이용한 동치 검사 문제 해결

본 연구에서는 동치 검사 문제를 모델 검사 기법으로 해결한다. 먼저 전 장에서 살펴본 유한 상태 기계의 동치 조건을 시제 논리식으로 변환하는 과정은 다음과 같다. 정의 4에 의하면 동치 조건은

$$\exists d \in S \cdot \forall s \in S \cdot s \in P(d) \Rightarrow E(s)$$

이다. 정의 3을 이용해서 $E(s)$ 부분을 치환하면

$$\exists d \in S \cdot \forall s \in S \cdot s \in P(d) \Rightarrow (\forall i \cdot \lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))$$

을 얻는다. 초기 상태는 항상 존재하기 때문에 ∃-제거 규칙(∃-elimination rule, [8])를 이용하여

$$\forall s \in S \cdot s \in P(d) \Rightarrow (\forall i \cdot \lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))$$

앞쪽에 있는 ∃ 한정사를 제거한다. '모든 가능한 입력'을 나타내는 뒤 쪽의 ∀는 ∀-제거 규칙(∀-elimination rule, [8])를 이용하여

$$\forall s \in S \cdot s \in P(d) \Rightarrow (\lambda_A(i,s_A) \Leftrightarrow \lambda s(i,s_B))$$

제거한다. 맨 앞의 $\forall s \in S \cdot s \in P(d)$은 'for every reachable state(도달 가능한 모든 상태)'를 나타내기 때문에, 위의 식을

for every reachable state, $\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B)$

로 개서할 수 있다. 시제 논리 언어인 CTL (Computation Tree Logic, [9]) 연산자 중에서 AG는 'for every reachable state'를 나타낸다. 따라서 위의 식을 CTL 시제 논리식으로 나타내면

$$AG(\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))$$

이다.

정리 1: 두 개의 유한 상태 기계 M_A, M_B와 곱 기계 $M = M_A \times M_B$가 주어졌을 때, 다음은

$$M_A \equiv M_B \text{ iff } M \models AG(\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))$$

성립한다.

증명: 증명은 매우 직관적이다. 먼저 좌변에서 우변으로 증명을 한다. 두 기계가 동치라면, 두 기계를 조합한 곱 기계의 모든 도달 가능한 상태에서 어떠한 값이 입력되더라도 두 기계의 출력은 같다. 따라서 좌변에서 우변으로는 참이다. 우변에서 좌변으로 증명도 이와 비슷하다. ■

정리에 의해서, 동치 검사를 모델 검사로 해결할 수 있게 되었다. 모델 검사를 이용하여 동치 검사를 수행하는 방법 및 예제를 살펴보면 다음과 같다. 먼저 모델 M에서 CTL 식 ϕ를 만족하는 상태 집합을 $[\![\phi]\!]$라고 하자. 만일 초기 상태가 $[\![\phi]\!]$에 포함된다면 $M \models \phi$ 이고, 포함되지 않는다면 $M \not\models \phi$이다. 그러므로 모델 검사 알고리즘의 핵심은 $[\![\phi]\!]$을 구하는 것이다[7]. 이를 위해서 역방향 탐색(backward search) 함수 $pre_\forall$를 다음과 같이

$$pre_\forall(Q) = \{s \in S \mid \forall s' \cdot \text{if } (i,s,s') \in \delta \text{ then } s' \in Q\}$$

정의한다. 함수 $pre_\forall$는 집합 Q로만 천이되는 이전 상태들을 리턴한다. 함수 $pre_\forall$와 최대 고정점을 이용하여 동치 조건 $AG(\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))$을 만족하는 상태 집합을 아래와 같이

$$[\![AG(\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))]\!] = \nu Z.([\![\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B)]\!] \cap pre_\forall(Z))$$

계산할 수 있다. 여기서 ν는 최대 고정점이다. 최대 고정점 계산을 이용하여 $\phi = AG(\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))$를 만족하는 상태 집합을 계산한 후, 초기 상태 포함 유무를 통해서

$$\begin{cases} M_A \equiv M_B \text{ if } d \in [\![\phi]\!] & (M \models \phi\text{인 경우}) \\ M_A \not\equiv M_B \text{ otherwise} & (M \not\models \phi\text{인 경우}) \end{cases}$$

두 기계의 동치 여부를 판정할 수 있다.

예제 1: 그림 1에 주어진 유한 상태 기계 M_A와 M_B로부터 곱 기계 M을 구축하면 그림 2와 같다. 각 기계의 상태수가 3과 4이기 때문에 곱 기계의 상태 수는 3*4=12 이다.

M에서 동치 조건인 $AG(\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))$를 만족하는 상태들의 집합은 다음과 같다:

$$[\![AG(\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))]\!] = \nu Z.([\![\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B)]\!] \cap pre_\forall(Z)) = \{(s_0,t_0), (s_1,t_1), (s_2,t_2), (s_2,t_3)\}$$

왜냐하면, 최대 고정점 $\nu Z.([\![\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,sS)]\!] \cap pre_\forall(Z))$의 계산이 아래와 같기 때문이다.

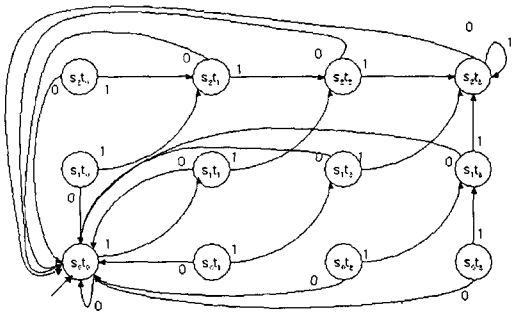

그림 2 곱 기계 $M = M_A \times M_B$

$\tau^0 = S$

$\tau^1 = [\ \lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B)\ ] \cap pre_\forall(\tau^0)$

$=\{(s_0,t_0), (s_0,t_1), (s_1,t_0), (s_1,t_1), (s_2,t_2), (s_2,t_3)\} \cap S$

$=\{(s_0,t_0), (s_0,t_1), (s_1,t_0), (s_1,t_1), (s_2,t_2), (s_2,t_3)\}$

$\tau^2 = [\ \lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B)\ ] \cap pre_\forall(\tau^1)$

$= \{(s_0,t_0), (s_0,t_1), (s_1,t_0), (s_1,t_1), (s_2,t_2), (s_2,t_3)\} \cap$
$\{(s_0,t_0), (s_1,t_1), (s_1,t_2), (s_1,t_3), (s_2,t_1), (s_2,t_2), (s_2,t_3)\}$

$= \{(s_0,t_0), (s_1,t_1), (s_2,t_2), (s_2,t_3)\}$

$\tau^3 = [\ \lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B)\ ] \cap pre_\forall(\tau^2)$

$= \{(s_0,t_0), (s_0,t_1), (s_1,t_0), (s_1,t_1), (s_2,t_2), (s_2,t_3)\} \cap$
$\{(s_0,t_0), (s_1,t_1), (s_1,t_2), (s_1,t_3), (s_2,t1), (s_2,t_2), (s_2,t_3)\}$

$= \{(s_0,t_0), (s_1,t_1), (s_2,t_2), (s_2,t_3)\}$

$= \tau^2$

$(s_0,t_0) \in [\ AG(\ \lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))\ ]$ 이기 때문에 곱 기계 모델은 동치 검사 조건을 만족한다. 따라서 두 기계 M_A와 M_B는 동치이다. 그림 3에서 보듯이 고정점 계산으로 구해진 $\{(s_0,t_0), (s_1,t_1), (s_2,t_2), (s_2,t_3)\}$ 집합은 초기 상태로부터 도달 가능한 상태 집합이다. ■

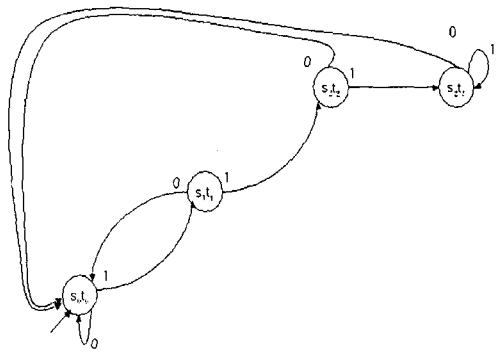

그림 3 초기 상태로부터 도달 가능한 상태 집합

4. SMV를 이용한 동치 검사

전 장에서 동치 검사를 모델 검사로 해결할 수 있는 이론적인 배경을 살펴보았다. 본 장에서는 모델 검사 도구인 SMV(Symbolic Model Verifier, [10])를 이용하여 동치 검사를 수행하는 과정을 기술한다. 유한 상태 기계를 SMV로 변환하는 규칙과 시스템 구현 사항을 차례대로 기술한다.

4.1 SMV 입력 언어로의 변환 규칙

SMV 에서는 개별적으로 정의된 각 기계의 상태는 내부적으로 상태들의 카테시안 곱을 형성한다. 예를 들어 두 개의 집합 $S_A = \{s_1,\cdots,s_n\}$, $S_B = \{t_1,\cdots,t_m\}$으로 각 기계의 상태를 정의하면, SMV 내부에서는 이들의 곱 상태 $\{(s_1,t_1),\cdots,(s_n,t_n)\}$를 갖는다. 이러한 SMV 특성을 이용한다면 유한 상태 기계로부터 SMV 코드를 생성하는 변환 규칙을 간단히 기술할 수 있다. 아래의 변환 규칙은 M_A의 변환 규칙만을 다룬다. 먼저 변환 규칙 1 - 3은 변수 선언을 나타낸다. 규칙 4 - 6은 초기 값, 상태 천이, 그리고 출력 함수를 기술하는 규칙이다. 규칙 7은 검사할 동치 조건을 SMV로 변환한다.

규칙 1: 유한 상태 기계는 여러 개의 상태 중 하나의 상태에만 머무르기 때문에 상태 집합 $S_A = \{s_1,\cdots,s_n\}$는 열거형 (enumeration type)으로 변환한다.

VAR S_A:$\{s_1,\cdots,s_n\}$;

규칙 2: 입력 집합 I = $\{i_1,\cdots,i_m\}$는 두 기계에서 같으며, 하나의 입력만이 선택되므로 열거형으로 변환한다.

VAR I : $\{i_1,\cdots,i_m\}$;

규칙 3: 출력 집합 $O_A = \{o_1,\cdots,o_k\}$는 입력 집합과 마찬가지로 열거형으로 변환한다. 그러나 동치를 위해 각 기계의 출력 값을 비교해야 하므로 각 기계마다 따로 출력 값을 선언한다. 그리고 초기의 출력 값이 미정의되어 있기 때문에 이를 고려해서 아래와 같이 변환한다.

VAR O_A: $\{0,o_1,\cdots,o_k\}$;

여기서, 0은 미정의를 나타낸다.

규칙 4: 입력은 외부 환경에서 주는 것이므로 비결정적으로 처리해야 한다. SMV에서는 아무런 값을 배정하지 않음으로서 비결정성을 기술할 수 있다. 입력을 비결정적으로 다루기 위해서 초기값과 다음 상태 값을 고의적으로 지정하지 않는다. 그러나 출력의 경우, 초기 값이 미정의 되어 있기 때문에 0을 배정한다. 시작 상태는 $d_A \in S$로 배정하면 된다. 따라서 초기 값은 다음과 같이 표현된다.

INIT $S_A = d_A$ & $O_A = 0$

규칙 5: 정의에 의하면 상태 천이 함수는 δ(i, s) =

s′이다. 즉, 다음 상태는 현재 상태와 입력에 의해서 결정된다. M_A의 상태 천이를 나타내는 식은

$$\vee_{i\in I, s\in S_A}(I=i \wedge S_A=s \wedge next(S_A)=\delta(i,s))$$

이며, SMV 표현은 다음과 같다.

```
TRANS
  ((I=i1 & SA = s1 & next(SA)=s′)
  |.....
  |(I=i1 & SA = sn & next(SA)=s′)
  |(I=i2 & SA = s1 & next(SA)=s′)
  |.....
  |(I=im & SA = sn & next(SA)=s′))
```

규칙 6: 정의에 의하면 출력 함수는 λ(i, s) = o이다. 즉, 현재 상태와 입력에 의해서 출력 값이 결정된다. M_A의 출력을 나타내는 식은

$$\vee_{i\in I, s\in S_A}(I=i \wedge S_A=s \wedge next(O_A)=\lambda(i,s))$$

이며, SMV 표현은 다음과 같다.

```
TRANS
  ((I=i1 & SA = s1 & next(OA)=o)
  |.....
  |(I=i1 & SA = sn & next(OA)=o)
  |(I=i2 & SA = s1 & next(OA)=o)
  |.....
  |(I=im & SA = sn & next(OA)=o))
```

규칙 7: 전장에서 살펴본 바와 같이 두 기계의 동치 조건은

$$AG(\lambda_A(i,s_A) \Leftrightarrow \lambda_B(i,s_B))$$

이다. 즉 두 기계의 출력이 도달 가능한 모든 상태에서 동일해야 한다는 것이다. 이것을 SMV 입력 언어로 변환하면 다음과 같다.

```
DEFINE equivalence := OA <-> OB;
SPEC AG equivalence
```

4.2 구현 및 분석

변환 규칙을 이용하여 유한 상태 기계로부터 SMV 코드를 자동 변환하였다. 전체 시스템의 개요 및 입력 텍스트 파일은 그림 4, 그림 5와 같다.

유한 상태 기계를 기술한 텍스트 파일을 받아서 유한 상태 기계, 크립키 구조(Kripke structure, [7]), 그리고 곱 크립키 구조를 차례대로 구축한다. 그런 다음에 변환

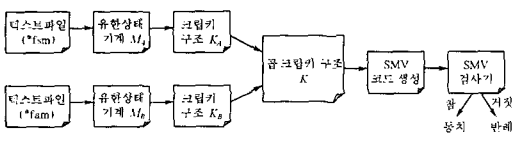

그림 4 변환 시스템

.i 1	(입력수)	.i 1	(입력수)
.o 1	(출력수)	.o 1	(출력수)
.s 3	(상태수)	.s 4	(상태수)
.p 6	(천이수)	.p 8	(천이수)
.d st0	(초기상태)	.d st0	(초기상태)
0 st0 st0 0	(상태천이)	0 st0 st0 0	(상태천이)
1 st0 st1 0	(상태천이)	1 st0 st1 0	(상태천이)
0 st1 st0 0	(상태천이)	0 st1 st0 0	(상태천이)
1 st1 st2 0	(상태천이)	1 st1 st2 0	(상태천이)
0 st2 st0 0	(상태천이)	0 st2 st0 0	(상태천이)
1 st2 st2 1	(상태천이)	1 st2 st3 1	(상태천이)
		0 st3 st0 0	(상태천이)
		1 st3 st3 1	(상태천이)
(a) M_A		(b) M_B	

그림 5 유한 상태 기계 M_A와 M_B를 기술한 텍스트 파일

규칙을 적용하여 SMV 코드를 최종적으로 생성한다. 그림 5에 있는 M_A와 M_B의 텍스트 파일로 부터 SMV를 코드를 생성하여 실행시킨 결과가 그림 6에 있다. 그림에서 보듯이, 곱 크립키 구조의 전체 상태 수는 96개이며,[2] 검사에 소요된 시간은 0.01초이었다.

이와 같은 방법으로 여러 개의 유한 상태 기계에 대해서 동치 검사를 수행하였고 그 결과는 표 1과 같다. 다루었던 예제 중에서 모델 7이 가장 복잡했다. 모델 7의 경우 변환된 SMV 코드의 행수는 6,958행이었고 생성된 전체 상태 수와 도달 가능한 상태 수가 각각 2^{54} 및 2^{11}였지만, 수행 시간은 1초 이내로(0.84초) 만족스러웠다(모델 7의 경우 M_A와 M_B는 동일한 모델이다. 동일한 모델을 검사하면 그 결과는 당연히 동치이기 때문에, 동일한 모델은 동치 검사의 가장 간단한 시험 사례라고 할 수 있다).

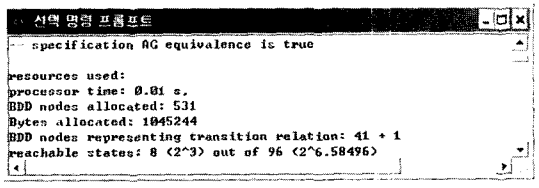

그림 6 SMV 를 이용하여 두 기계의 동치 검사 결과

2) 유한 상태 기계가 고급 언어라면, 크립키 구조는 저급 언어이다. 고급 언어의 한 문장이 변환될 때 저급 언어의 여러 문장이 필요하다. 유한 상태 기계와 크립키 구조도 이와 비슷하다. 2장에서 살펴본 대로 M_A와 M_B의 곱 기계의 상태 수는 3*4=12 이다. 그러나 M_A와 M_B의 곱 크립키 구조의 상태 수는 2*2*2*3*4=96 이다. 2, 2, 2, 3, 4는 각각 입력수, M_A의 출력 수, M_B의 출력 수, M_A의 상태 수, M_B의 상태 수 이다.

표 1 실행 결과 분석

유한상태 기계		상태 수	천이 수	입력 수	출력 수	SMV 라인 수	전체 상태 공간	도달한 상태수	수행 시간 (단위초)
모델1	M_A	2	4	1	1	66	48 ($\approx 2^6$)	8 ($\approx 2^3$)	0.01s
	M_B	3	6	1	1				
모델2	M_A	2	4	1	1	56	32 ($\approx 2^5$)	8 ($\approx 2^3$)	0.04s
	M_B	2	4	1	1				
모델3	M_A	4	8	1	1	94	96 ($\approx 2^7$)	10 ($\approx 2^3$)	0.01s
	M_B	3	6	1	1				
모델4	M_A	3	6	1	1	94	96 ($\approx 2^7$)	8 ($\approx 2^3$)	0.01s
	M_B	4	8	1	1				
모델5	M_A	9	25	2	1	765	1440 ($\approx 2^{10}$)	48 ($\approx 2^6$)	0.06s
	M_B	10	27	2	1				
모델6	M_A	9	25	2	1	711	1296 ($\approx 2^{10}$)	48 ($\approx 2^6$)	0.09s
	M_B	9	25	2	1				
모델7	M_A	48	115	7	19	6958	1.7e+16 ($\approx 2^{54}$)	2049 ($\approx 2^{11}$)	0.84s
	M_B	48	115	7	19				

한편, 유한 상태 기계가 비 동치인 경우 SMV는 반례를 생성한다. 반례를 통해서 비 동치의 원인을 파악할 수 있다. 예를 들어, 그림 7에 있는 두 기계 M_A와 M_B를 살펴보자(표 1의 모델 6에 해당). SMV 코드를 자동 생성하여 실행한 결과 AG equivalence 는 거짓이었으며, 비동치의 원인을 설명하는 반례가 그림 8과 같이 생성되었다. 반례중에서 두 기계의 상태가 각각 s6, t6인 경우를 보자(그림 8에서 state1.8 에 해당). 가능한 입력 수가 총 4개(00, 01, 10, 그리고 11)이기 때문에 두 개의 부울 변수 in0, in1으로 입력을 나타낸다. state1.8 에서 in0가 0으로 변경되었고 in1은 계속해서 1을 유지하기 때문에 현재 입력은 01이 된다. 따라서 상태 천이가 발생해서 두 기계의 상태는 각각 s7, t7이 된다 (state1.9 에 해당). 입력 변수에 대해서 아무런 변경이 없기 때문에, 현재 입력은 다시 01이 된다. 따라서 두 기계의 상태는 s7, t7에서 s6, t6로 각각 천이되지만, 이때 출력은 1과 0으로 서로 다르다. 따라서 비동치임을 알리고 종료한다.

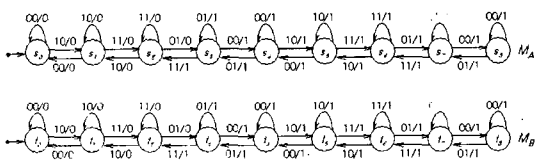

그림 7 비동치 관계에 있는 유한 상태 기계 M_A와 M_B

```
-- specification AG equivalence is false
-- as demonstrated by the following execution
   sequence
state 1.1:
equivalence = 1
MA = s0
MB = t0
in1 = 0
in0 = 0
OA0 = 0
OB0 = 0

state 1.2:
in0 = 1

state 1.3:
MA = s1
MB = t1
in1 = 1

state 1.4:
MA = s2
MB = t2
in0 = 0

state 1.5:
MA = s3
MB = t3
in1 = 0

state 1.6:
MA = s4
MB = t4
in0 = 1
OA0 = 1
OB0 = 1

state 1.7:
MA = s5
MB = t5
in1 = 1

state 1.8:
MA = s6
MB = t6
in0 = 0

state 1.9:
MA = s7
MB = t7

state 1.10:
equivalence = 0
in1 = 0
OB0 = 0
```

그림 8 비동치의 원인을 설명하는 반례

5. 결 론

요약하자면, 본 연구에서는 유한 상태 기계의 동치 검사($M_A \equiv M_B$)를 모델 검사($M \models \phi$)로 해결했다. 검사될 두 기계를 카테시안 곱해서 모델 M을 얻었고, 두 기계의 동치 조건을 정의한 후 이를 CTL 식 ϕ로 표현하였다. 그런 후, 모델 검사를($M \models \phi$) 통해서 두 기계의 동치 여부를 판정하였다. 유한 상태로부터 SMV 코드를 생성하는 변환 규칙을 제시하고, 변환 시스템을 구현하였다. 여러 개의 예제에 본 시스템을 적용해 본 결과 동치와 비동치를 정확하게 구분하였다. 다루었던 예제 중에서 가장 큰 상태 수가 2^{54}였지만 수행 시간은 1초 이내로 만족스러웠다. 이와 같은 유한 상태 기계의 동치 검사는 시스템 최적화 및 정확성 확인등에 활용될 수 있다. 만약 동치라고 판정된다면, 복잡한 유한 상태 기계는 단순한 기계로 대치될 수 있다. 또한 명세와 구현이 모두 유한 상태 기계로 표현된 경우, 동치 검사를 이용해서 구현이 명세를 만족하는지를 결정할 수 있다.

전술한 바와 같이, 본 연구에서는 Clarke [7] 등이 제시한 시제 논리 모델 검사 기법을 동치 검사 [11,12] 문제에 응용하였다. 그 결과 동치 검사 시스템을 새롭게 개발하기 보다는, 범용 모델 검사 도구인 SMV 를 사용하여 동치 검사를 자동화할 수 있었다. 동치 검사와 관련해서 앞으로 계속 연구해야 할 내용은 다음과 같다. 첫째, SMV 대신 최근에 활발히 연구되고 있는 BMC (Bounded Model Checker, [13]) 등을 이용하여 동치 검사를 수행하는 것이다. 둘째, 유한 상태 기계 보다 표현력이 높은 상태도를 동치 검사하는 것이다.

참 고 문 헌

[1] R. Skvarcius and W.B. Robinson, Discrete Mathematics with Computer Science Applications, The Benjamin/Cummings Publishing Company, 1986.

[2] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," ACM Transactions on Software Engineering and Methodology, Vol.5, No. 4, pp.293-333, 1996.

[3] David Y.W. Park, et.al., "Static Analysis to Identify Invariants in RSML Specifications," In Proceedings of Formal Techniques in Real-Time and Fault-Tolerant'98, LNCS 1486, 1998.

[4] C. Heitmeyer, et.al., "Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications," IEEE Transactions on Software Engineering, Vol.24, No.11, 1998.

[5] S.Y. Huang, K.T. Cheng, K.C. Chen, C.Y. Huang, and F. Brewer, "AQUILA: An Equivalence Checking System for Large Sequential Designs," IEEE Transactions on Computer, Vol.49, No.5, pp.443-464, 2000.

[6] Robert Meolic, Tatjana Kapus, Zmago Brezocnik, "Computing Testing Equivalence with Binary Decision Diagrams," In Proceedings of the Seventh Electrotechnical and Computer Science Conference ERK'98, pp.51-54, 1998.

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

[8] M. Huth and M. Ryan, Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge University Press, 2000.

[9] E.M. Clarke, E.A. Emerson, and A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications," ACM Transactions on Programming Languages and Systems, Vol.8, No.2, pp.244-263, 1986.

[10] K.L. McMillan, "Symbolic Model Checking: An approach to the state explosion problem," PhD thesis, Carnegie Mellon University, 1992.

[11] C.A.J. van Eijk and J.A.G. Jess, "Detection of Equivalent State Variables in Finite State Machine Verification," In Proceedings of the 1995 ACM/IEEE International Workshop on Logic Synthesis, pp. 3.35-3.44, 1995.

[12] C.A.J. van Eijk and J.A.G. Jess, "Exploiting Functional Dependencies in Finite State Machine Verification," In Proceedings of the European Design and Test Conference ED&TC, pp.9-14, 1996.

[13] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," In Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS '99), LNCS 1579, 1999.

권 기 현

정보과학회논문지 : 소프트웨어 및 응용 제 30 권 제 3 호 참조

엄 태 호

2001년 2월 경기대학교 정보과학부 이학사. 2003년 2월 경기대학교 전자계산학과 이학석사. 2003년 3월 경기대학교 전자계산학과 박사과정