모델 검증을 이용한 게임 풀이

경기대학교 권기현*

1. 서 론

시스템의 크기와 종류에 관계없이 기능과 관련해서 반드시 만족해야 할 요구 사항이 있다. 예를 들어, 사거리 교통 제어 시스템은 안전성(직진 신호를 동시에 받아 사거리에서 충돌하는 경우는 결코 발생해서는 안됨) 및 궁극성(대기중인 차량은 언젠가는 직진 신호를 받아서 사거리를 통과해야 함)을 만족해야 한다. 이러한 요구 사항을 속성이라고 부르며, 고 품질의 시스템 개발을 위해서는 속성이 시스템 내에서 만족되는지를 반드시 검증해야 한다. 그러나 시스템 및 속성의 복잡도가 높기 때문에, 시스템이 속성을 만족하는지를 직접 검증할 수 없다. 불행하게도 이것은 결정 불가능한 문제이다. 어쩔 수 없이 시스템 및 속성의 표현을 축소해야 한다. 만약 시스템과 속성을 유한 상태 모델과 명제 시제 논리식으로 각각 축소해서 표현한다면, 유한 상태 모델이 명제 시제 논리식을 만족하는지를 검증할 수 있다. 다행히 이것은 결정 가능한 문제이다.

모델 검증은 유한 상태 모델 M과 명제 시제 논리식 ϕ을 받아서 만족성 여부 $M \models \phi$를 결정한다[1]. 모델이 갖는 모든 상태 공간을 철저하게 탐색하면서 논리식의 만족성 여부를 결정하기 때문에, 시뮬레이션으로는 찾을 수 없는 난해한 버그를 찾을 수 있다. 그리고 사용자의 개입 없이 검증 과정이 자동 처리되기 때문에, 정리 증명에 비해서 빠르다. 또한 모델이 논리식을 만족하지 않는 경우, 그 이유를 담은 반례를 제공함으로서 모델의 디버그 작업을 돕는다. 이와 같은 이점으로 인해서 모델 검증은 하드웨어 검증 및 소프트웨어 검증에 활발히 사용되고 있다[2,3]. 특히 모델 검증의 핵심 엔진인 고정점 계산 및 도달성 분석은 프로그램의 정적 분석에도 유용하다[4].

앞에서 설명한 바와 같이 모델 검증의 이점 중의 하나가 반례(counterexample) 생성이다. 예를 들어, 시제 논리식 AGϕ가 거짓인 경우 $\neg\phi$가 참이 되는 경로를 생성한다[5]. 이러한 경로를 반례라고 부르며, 반례를 통해서 AGϕ의 값이 거짓인 이유를 알 수 있다. 반례는 모델의 디버깅에 이용될 뿐만 아니라 모델의 축소 및 추상화에 활용된다[6]. 여기에서는 반례를 이용하여 문제 풀이(problem solving)를 하려고 한다. 문제를 유한 상태 모델로 표현하고 오답을 시제 논리식으로 표현해서 모델 검증기에 입력하면, 모델 검증기는 반례로 오답에 대한 부정 즉 정답을 출력한다. 다시 말해서 반례로 출력된 것이 문제를 해결하는 정답인 것이다.

이러한 아이디어를 게임 풀이에 적용해 보았다. 게임은 제약 조건(또는 게임 규칙)을 준수하면서 초기 상태를 목표 상태로 옮기는 것이다. 게임 풀이를 위해서 정형 기법 연구회 이외의 다른 연구 분야(예를 들어 인공 지능 연구회)에서 오래 전부터 제약 만족 문제, 순서 계획, CLP(Constraint Logic Programming) 등의 기법을 연구해 왔다[7]. 이들의 접근과는 달리, 본 연구에서는 모델 검증 기법으로 게임을 풀었다. 실험 대상으로 핸드폰에서 이용 가능하고 대중적으로 알려진 푸쉬 푸쉬 게임을 선정했다. 모델 검증으로 찾아낸 우리의 답은 전문가의 답보다 경제적이었다. 즉 보다 적은 움직임으로 게임을 해결하였다. 가장 어려웠던 점은 역시 상태 폭발 문제(state explosion problem)였다. 설명한 바와 같이 모델 검증은 모든 상태 공간을 철저하게 탐색하기 때문에 상태 폭발 문제를 잘 다루는 것이 게임 풀이의 핵심 사항이다. 상태 폭발을 방지하기 위해서 여러 가지 추상화 방법을 사용하여 상태 공간의 크기를 축소하였

* 종신회원

고, 그 결과 푸쉬 푸쉬 게임을 해결할 수 있었다. 모델 검증과 같은 이론적인 연구가 산업체에 실질적으로 활용되기 위해서 반드시 풀어야 할 문제가 상태 폭발과 같은 규모 처리 문제이다. 왜냐하면 산업체에서 실제적으로 처리하려는 모델의 크기는 매우 거대하기 때문이다.

2장에서는 모델 검증 기법을 간략히 소개한다. 3장에서는 모델 검증 기법을 이용해서 푸쉬 푸쉬 게임을 해결한 과정을 기술하고, 4장에서는 상태 폭발을 방지했던 방법을 설명한다. 그리고 5장에서 결론을 맺는다.

2. CTL 모델 검증

모델 검증에 사용되는 모델은 대부분 유한 상태 기계와 유사하다. 그래서 모델의 핵심 요소는 상태와 상태간의 천이이며, 이들을 사용하여 시스템의 행위를 간결하게 모델링 한다. 특히 CTL(Computation Tree Logic) 모델 검증에서는 크립키 구조라 불리는 모델 $M = (S, I, R, AP, L)$을 사용한다[1].

- S 는 상태들의 집합이다.
- $I \subseteq S$ 는 초기 상태들의 집합이다.
- $R \subseteq S \times S$ 은 상태들간의 천이를 나타내는 관계이다.
- AP 는 단순 명제들의 집합이다.
- $L: S \rightarrow 2^{AP}$은 각 상태에서 참이 되는 단순 명제들을 해당 상태에 배정하는 함수이다.

여기서 R은 전체 관계(total relation)라고 가정한다. 즉 $\forall s \in S \bullet \exists s' \in S \bullet (s,s') \in R$로서, 모든 상태마다 천이할 수 있는 다음 상태가 최소한 하나 이상 존재한다. 경로 $\pi = s_0\ s_1\ s_2\ s_3\ s_4 \cdots$ 는 천이 가능한 상태들을 차례대로 나열한 것으로서 $(s_i, s_{i+1}) \in R$, $i \geq 0$이며 그 길이는 무한이다.

모델에 관한 속성은 분기 시제 논리 언어인 CTL로 표현한다. CTL은 모델을 트리의 관점에서 해석한다. 즉 초기 상태를 루트로 해서 모델을 풀어헤치면 트리를 얻을 수 있다. 트리는 모델의 가능한 행위를 모두 표현하며, 트리의 각 경로는 모델이 가질 수 있는 특정한 행위를 나타낸다. 모델의 속성을 정형적으로 기술하기 위해서 CTL 은 두 개의 경로 한정자 A(All), E(Exists)와 네 개의 시제 연산자 X(neXt), F(Future), G(Globally), U(Until)를 갖는다. 경로 한정자와 시제 연산자를 조합하면 8개의 CTL연산자 AX, EX, AF, EF, AG, EG, AU, EU를 얻는다. CTL은 이들 연산자를 추가하여 기존의 명제 논리를 확장한 논리 언어이다. CTL 전체 구문을 BNF(Backus-Noar Form) 형식으로 나타내면 다음과 같다.

$$\begin{aligned}\phi :=\ & p \mid \bot \mid \top \mid \neg\phi \\ & \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \Rightarrow \phi_2 \mid \phi_1 \Leftrightarrow \phi_2 \\ & \mid \mathrm{AX}\phi \mid \mathrm{EX}\phi \mid \mathrm{AF}\phi \mid \mathrm{EF}\phi \\ & \mid \mathrm{AG}\phi \mid \mathrm{EG}\phi \mid \mathrm{A}(\phi_1 \mathrm{U} \phi_2) \mid \mathrm{E}(\phi_1 \mathrm{U} \phi_2)\end{aligned}$$

여기서 $p \in AP$ 는 임의의 단순 명제를 나타내며 $\bot$, $\top$ 는 각각 참과 거짓을 나타내는 상수이다. CTL 식 ϕ, Ψ의 값이 모든 모델과 모든 상태에서 동일하다면 두 식을 동치라고 부르며 $\phi \equiv \Psi$로 표시한다. 동치 관계에 있는 식은 다음과 같다.

$$\top \equiv \neg\bot$$

$$\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$$

$$\phi_1 \Rightarrow \phi_2 \equiv \neg(\phi_1 \wedge \neg\phi_2)$$

$$\phi_1 \Leftrightarrow \phi_2 \equiv \neg(\phi_1 \wedge \neg\phi_2) \wedge \neg(\phi_2 \wedge \neg\phi_1)$$

$$\mathrm{AX}\phi \equiv \neg\mathrm{EX}\neg\phi$$

$$\mathrm{AF}\phi \equiv \neg\mathrm{EG}\neg\phi$$

$$\mathrm{EF}\phi \equiv \mathrm{E}(\neg\bot \mathrm{U} \phi)$$

$$\mathrm{AG}\phi \equiv \neg\mathrm{EF}\neg\phi \equiv \neg\mathrm{E}(\neg\bot \mathrm{U} \neg\phi)$$

$$\mathrm{A}[\phi_1 \mathrm{U} \phi_2] \equiv \neg\mathrm{E}[\neg\phi_2 \mathrm{U} (\neg\phi_1 \wedge \neg\phi_2)] \wedge \neg\mathrm{EG}\neg\phi_2$$

두 식이 동치이기 때문에 좌변의 식은 우변의 식으로 대치 가능하다. 따라서 우변에 나오는 연산자의 모임이 CTL 식을 정의하는데 요구되는 최소한의 연산자 집합이다. 위의 경우는 {$\bot$,$\neg$, $\wedge$, EX, EG, EU}이다. 이들을 이용해서 CTL 구문을 정의하면 다음과 같다.

$$\begin{aligned}\phi :=\ & p \mid \bot \mid \neg\phi \mid \phi_1 \wedge \phi_2 \\ & \mid \mathrm{EX}\phi \mid \mathrm{EG}\phi \mid \mathrm{E}(\phi_1 \mathrm{U} \phi_2)\end{aligned}$$

이제 CTL 의 의미를 정형적으로 살펴보자. 모델 M의 상태 s에서 CTL식 ϕ가 참인 경우를 $M,s \models \phi$로 표시한다. 그렇치 않다면 $M,s \not\models \phi$ 로 표시한다. 상태 s에서 CTL식 ϕ의 값은 아래와 같이 재귀적으로 정의된다.

$M,s \models \neg\phi$ iff $M,s \not\models \phi$

$M,s \models \phi_1 \wedge \phi_2$ iff $M,s \models \phi_1$ and $M,s \models \phi_2$

$M,s \models \mathrm{EX}\phi$ iff $M,s' \models \phi$ for some state

s' with $(s,s') \in R$

$M,s \models \mathrm{EG}\phi$ iff $\exists_{\pi=s0,s1,s2,\ldots} \bullet \forall_{i \geq 0} \bullet$

$M, s_i \models \phi$

$M,s \models \mathrm{E}(\phi_1 \mathrm{U} \phi_2)$ iff $\exists_{\pi=s0,s1,s2,\ldots} \bullet$

$(\exists_{k \geq 0} \bullet M,s_k \models \phi_2 \wedge \forall_{0 \leq i<k}$

$\bullet M,s_i \models \phi_1)$

특별히 모델 M 의 모든 초기 상태에서 ϕ가 참인 경우를 $M \models \phi$로 표시하며 'M 이 ϕ를 만족한다' 라고 읽는다. 모델 검증 문제란 M과 ϕ를 받아서 M 이 ϕ를 만족하는지를 결정하는 문제이다. 이를 위해서 ϕ를 만족하는 상태들의 집합을 구한 후, 이들 상태 집합에 시작 상태가 포함되어 있는지를 검증한다. CTL 식 ϕ를 만족하는 상태들의 집합을$[[\phi]]$로 표시하자. 모델 검증 알고리즘은 모델의 초기 상태 집합이$[[\phi]]$의 부분 집합인 것을 검증한다. 즉

$$M \models \phi \quad \text{iff} \quad I \subseteq [[\phi]]$$

이다. 단순 명제 p의 경우 집합$[[\phi]]$를 쉽게 계산할 수 있다. EX, EU, EG 를 다루는데 함수 $pre_\exists$가 사용된다.

$pre_\exists(X)=\{s \in S \mid \exists_{s' \in S} \bullet (s,s') \in R \wedge s' \in X\}$

이 함수는 집합 X로 천이 할 수 있는 이전 상태(predecessor)들의 집합을 역 방향으로 계산한다. CTL 식 ϕ를 만족하는 상태 집합$[[\phi]]$은 다음과 같다.

$[[p]] = \{s \in S \mid p \in L(s)\}$

$[[\bot]] = \varnothing$

$[[\neg\phi]] = S \setminus [[\phi]]$

$[[\phi_1 \wedge \phi_2]] = [[\phi_1]] \cap [[\phi_2]]$

$[[\mathrm{EX}\ \phi]] = pre_\exists([[\phi]])$

$[[\mathrm{EG}\ \phi]] = \nu Z.([[\phi]] \cap pre_\exists(Z))$

$[[\mathrm{E}(\phi_1 \mathrm{U} \phi_2)]] = \mu Z.([[\phi_2]] \cup ([[\phi_1]] \cap pre_\exists(Z)))$

여기서 μ, ν는 각각 최소 고정점과 최대 고정점이다[8]. 상태 집합$[[\phi]]$을 계산할 때, 최소 고정점 μ는 공집합을 초기값으로 하여 계속해서 증가되다가 더 이상 증가하지 않는 집합을 얻게 될 때를 말하고 EF, EU, AF, AU를 계산할 때 사용한다. 최대 고정점 ν은 반대로 전체 집합을 초기값으로 하여 계속해서 감소하다가 더 이상 감소하지 않는 집합을 얻게 될 때를 말하고 EG, AG 를 계산할 때 사용한다.

모델 검증의 수행 시간은 역 방향 함수와 고정점 계산에 좌우된다. 함수 $pre_\exists$와 고정점을 계산하는데 소요되는 시간은 모델의 크기에 선형 비례한다. 여기서 모델의 크기는 $|M| = |S| + |R|$ 로서 상태 수와 천이 수를 합한 것이다. 주어진 식 ϕ에 대한 상태 집합$[[\phi]]$을 계산하기 위해서, 모델 검증 알고리즘은ϕ 의 서브 식을 재귀적으로 구하면서 길이가 짧은 식부터 긴 식 순서로 상태 집합을 계산한다. 따라서 알고리즘의 복잡도는 $O(|M| * |\phi|)$ 로서, 모델의 크기 및 식의 길이에 모두 선형 비례한다[9].

3. 게임 풀이

본 연구에서는 CTL 모델 검증 기법을 게임 풀이에 적용한다. 실험 대상으로 핸드폰에서 이용 가능하고 대중적으로 알려진 푸쉬 푸쉬 게임을 선정했다. 푸쉬 푸쉬 게임은 그림 1과 같이 게임 규칙을 준수하면서 시작 상태를 목표 상태로 옮기는 이동 경로를 찾아내는 것이다.

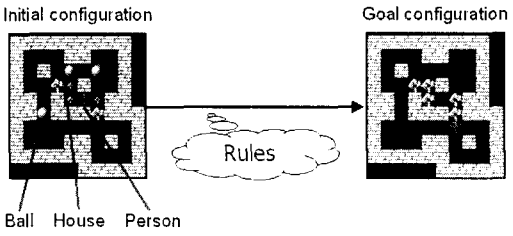


그림 1 푸쉬 푸쉬 게임

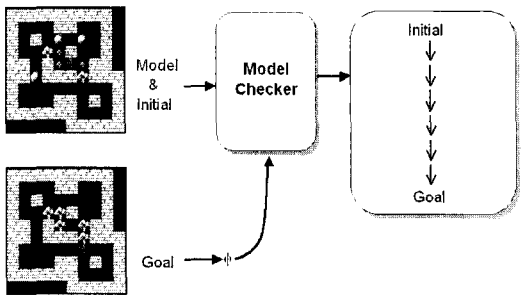


그림 2 모델 검증을 이용한 게임 풀이

게임 풀이 즉 이동 경로를 구하기 위해서 그림 2와 같이 게임 및 시작 상태를 유한 상태 모델로 표현하고, 목표 상태의 부정을 시제 논리식으로 표현해서

모델 검증기에 입력하면, 모델 검증기는 부정에 대한 반례 즉 시작 상태를 목표 상태로 옮기는 이동 경로를 제공한다.

표 1 전문가와 본 방법의 이동 경로 수 비교

단계	전문가	본 방법	차이	단계	전문가	본 방법	차이
1	10	10	0	26	106	83	23
2	93	89	4	27	65	61	4
3	120	114	6	28	209	162	47
4	36	33	3	29	95	89	6
5	50	50	0	30	124	101	23
6	94	79	15	31	194	116	78
7	45	44	1	32	160	126	34
8	213	170	43	33	301	261	40
9	38	34	4	34	89	59	30
10	63	29	34	35	100	56	44
11	47	29	18	36	191	154	37
12	60	56	4	37	165	120	45
13	55	55	0	38	58	38	20
14	86	72	14	39	88	64	24
15	115	101	14	40	83	77	6
16	66	64	2	41	114	108	6
17	119	97	22	42	151	137	14
18	141	129	12	43	67	35	32
19	76	60	16	44	203	171	32
20	130	105	25	45	122	169	-47
21	67	57	10	46	72	106	-34
22	88	82	6	47	101	96	5
23	219	122	97	48	230	173	57
24	115	89	26	49	136	84	52
25	99	83	16	50	78	341	-263

SMV(Symbolic Model Verifier)[10]를 사용하여 푸쉬 푸쉬 게임 1단계부터 50단계까지를 모두 풀었다. 게임의 인기가 높기 때문에 인터넷에 소위 족보라고 불리는 모범 답안들이 공개되어 있다.[2] 모범 답안을 공개한 자를 전문가라고 하자. 전문가가 제시한 이동 경로 수와 모델 검증기로 찾아낸 이동 경로 수의 비교가 표 1에 있다. 전문가와 본 방법의 결과는 모두 정확했다. 하지만 모델 검증으로 찾아낸 풀이가 훨씬 경제적이었다. 즉 적은 움직임으로 게임을 풀기 때문이다. 예를 들어 23 단계를 풀기 위해 전문가가 제시한 이동 경로 수는 219 회 였으나, 본 방법의 이동 경로 수는 122 회로 전문가 보다 무려 97 회 적은 움직임으로 게임을 풀었고, 움직임에 있어서 44.3%가 효율적이었다. 전체 평균으로 본다면, 전문가의 이동 경로 수는 본 방법보다 평균 22 회 길었고 따라서 전문가의 이동 경로 중 평균 18.3% 가 중복 움직임이었다. 모델 검증으로 찾아낸 답이 전문가 보다 효율적인 것은 당연하다. 왜냐하면 모델 검증은 전체 상태 공간을 조사하는 반면, 전문가는 전체 숲을 보지 못한 채 나무만을 보고서 움직이기 때문이다.[3] 각 단계마다의 전문가와 본 방법의 이동 경로 수의 편차를 도식화하면 그림 3과 같다 (부분적인 답만을 제공한 45, 46, 50 단계는 제외하였다).

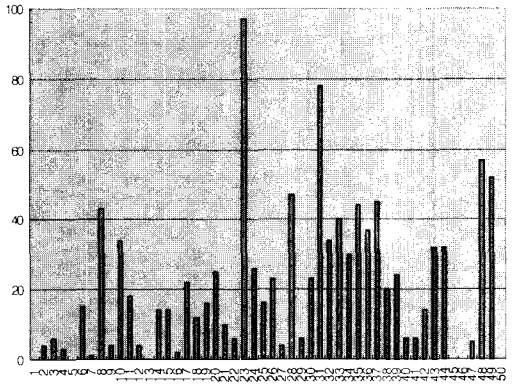


그림 3 전문가와 본 방법의 이동 경로 수 차이

4. 상태 폭발 방지

CTL 모델 검증을 적용하여 게임 풀이를 하는 과정에서 가장 어려웠던 점은 역시 상태 폭발 문제였다. 2장에서 살펴본 바와 같이 모델 검증 알고리즘의 복잡도는 $O(|M| * |\phi|)$로서 모델의 크기 및 논리식의

1) 푸쉬 푸쉬 게임의 정답을 공개하고 있는 사이트의 주소는 다음과 같다.
http://www.koha.co.kr/bbs/free_view.asp?idx=127&page=3
http://m-game.simmani.com/freeqna/freeqna_content.asp?BOARDID=3&pagenum=6&ANO=254
http://free3.ttboard.com/bugbear/ttboard/data/BOARD1/pushpush.htm

2) 표 1에서 보듯이 모든 단계마다 본 방법이 전문가 보다 우수했다. 그러나 45, 46, 50 단계는 전문가가 오히려 우수한 결과를 보였다. 과연 이러한 일이 있을까 하는 의문을 갖고서 전문가의 답을 조사해 본 결과, 전문가는 정답 전체 대신 일부분만을 올려놓은 것으로 판명되었다. 즉 어느 정도의 이동 경로만을 보여주면 나머지는 사용자가 해결할 있으리라 기대하고 정답의 일부분만을 올려놓은 것이다.

길이에 모두 선형 비례했다. 일반적으로 시제 논리식의 중첩 수준은 기껏해야 2-3 수준 정도로 논리식의 길이가 짧기 때문에, 모델 검증 시간에 소요되는 대부분의 시간은 모델 크기에 좌우된다. 간단한 시스템을 모델 검증하는 경우조차도 상태들의 수는 보통 수백만 또는 수천만이다. 예를 들어, 현재 사용중인 프로그램에 이진 변수를 하나 추가한다면, 모델 검증에 요구되는 복잡도는 두 배가 된다. 이와 같이 고려해야 할 상태 공간이 지수적으로 증가하는 현상을 상태 폭발 문제라고 한다. 일반적으로 상태 폭발이 갖는 문제점은 다음과 같다.

- 상태 공간이 너무 커서 메모리에 로드 할 수 없다.
- 설령 메모리에 로드 할지라도 전체 상태 공간을 탐색하기에 너무 크다. 따라서 해쉬 테이블 또는 가상 메모리를 사용해야 하는데, 상태 수가 증가할수록 시간은 매우 느려진다.

이것을 극복하기 위하여 많은 연구들이 진행되고 있다[11]. 연구들은 크게 영리한 방법(clever method)과 힘으로 밀어 붙이는 방법(brute-force method)으로 구분할 수 있다. 첫째, 상태 폭발을 방지하기 위해 사용되는 영리한 방법들은 다음과 같다.

- 효율적인 자료 구조 : OBDD(Ordered Binary Decision Diagram) 라는 자료 구조를 이용하여 모델의 개별적인 상태 대신 상태들의 집합을 표현한다[12].
- 추상화 : 검증할 식과 무관한 변수들을 제거하여 모델의 크기를 줄인다[13].
- 중복 검증 회피 : 대칭성 등을 이용해서 이미 검증된 부분과 동일한 부분의 재 검증을 회피한다[14].
- 결합 추론(compositional reasoning) : 복잡한 검증 문제를 보다 단순한 검증 문제로 분해하여 해결한 후 그 결과를 결합한다[15].

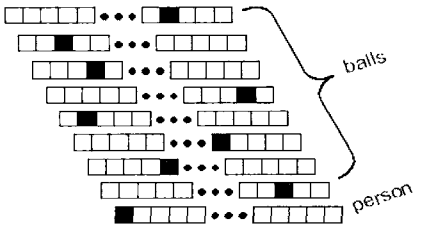
둘째, 힘으로 밀어붙이는 방법은 다음과 같다.

- 보다 큰 상태 공간을 다루기 위해서 메모리와 하드웨어 처리력을 증가한다.
- 슈퍼 컴퓨터, 인터넷 그리드 컴퓨팅, 분산 및 병렬 처리력을 활용한다.

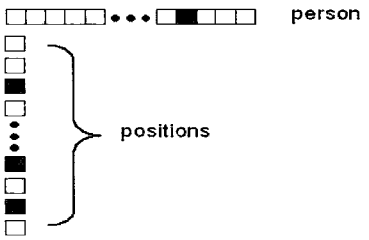
푸쉬 푸쉬 게임을 풀기 위해서, 일반적인 모델 검증 기법을 먼저 사용하였다. 게임을 SMV로 효율적으로 인코딩 하였으나 게임이 갖는 상태 공간의 수 때문에 하루가 지나도 풀리지 않는 단계가 많았다. 일반적 모델 검사 기법을 사용하여 게임을 풀 때 요구된 수행 시간 및 매모리 사용량은 표 2와 같다.

표 2 일반적인 기법을 사용했을 때 소요된 시간 및 메모리

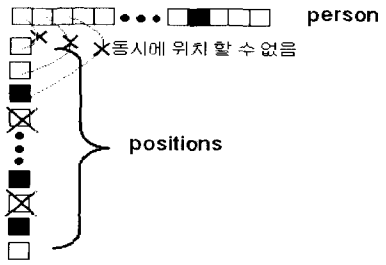
단계	수행시간 (sec)	메모리 (MB)	단계	수행시간 (sec)	메모리 (MB)
1	0.6	2	26	60.0	34
2	3.6	6	27	30.0	20
3	33.8	32	28	420.0	210
4	0.7	3	29	16.0	13
5	1.0	3	30	〉24hours	
6	123.3	93	31	1020.0	500
7	53.6	37	32	〉24hours	
8	〉24hours		33	〉24hours	
9	11.0	11	34	〉24hours	
10	5.0	8	35	〉24hours	
11	6.2	8	36	〉24hours	
12	10.7	12	37	2200.0	854
13	355.3	240	38	28.0	18
14	125.8	80	39	960.0	336
15	〉24hours		40	〉24hours	
16	16.5	16	41	120.0	58
17	〉24hours		42	1,260.0	532
18	1,200.0	508	43	480.0	299
19	1,200.0	613	44	1,860.0	539
20	〉24hours		45	300.0	109
21	2,880.0	1,109	46	780.0	413
22	420.0	219	47	35.0	25
23	3,000.0	870	48	1,980.0	841
24	600.0	317	49	〉24hours	
25	10.0	305	50	〉24hours	



(a) 첫 번째 시도: 상태 수 39^{9}



(b) 두 번째 시도: 상태 수 $39*2^{39}$



(c) 세 번째 시도: 상태 수 $39*2^{(39)\ 18)}$

그림 4 푸쉬 푸쉬 게임 23단계의 상태 표현

표 3 영리한 기법을 사용했을 때 소요된 시간 및 메모리

단계	수행시간 (sec)	메모리 (MB)	단계	수행시간 (sec)	메모리 (MB)
1	0.1	< 1	26	5.4	8
2	0.5	5	27	3.0	7
3	2.6	7	28	4.8	13
4	0.5	< 1	29	1.4	5
5	0.3	< 1	30	248.1	< 1
6	7.5	11	31	8.3	16
7	8.1	13	32	9.9	20
8	53.9	105	33	168.2	298
9	0.3	< 1	34	101.8	113
10	0.4	< 1	35	98.6	133
11	0.3	< 1	36	73.6	122
12	1.5	6	37	7.7	18
13	0.4	< 1	38	3.0	8
14	2.3	7	39	10.4	11
15	167.8	190	40	4.3	13
16	0.5	< 1	41	3.4	10
17	415.0	470	42	25.0	40
18	8.1	21	43	223.3	272
19	209.0	325	44	12.9	18
20	224.0	358	45	6.5	13
21	14.8	26	46	10.2	17
22	52.5	59	47	1.8	5
23	79.0	67	48	16.6	35
24	7.1	17	49	234.2	321
25	13.5	25	50	> 24hours	

예를 들어, 23단계의 경우 그림 4의 (a)와 같이 모델링 하였더니, 상태 공간의 수가 39^9로 매우 커서 모델 검증기로 풀 수 없었다. 그래서 (b)와 같이 효율적으로 상태를 다시 표현하였더니 상태 수가 $39*2^{38}$로 줄어들었고, 모델 검증에 소요된 시간과 메모리는 각각 3,000 초와 870MB 였다. 시간을 보다 절감하기 위해서 추상화 기법을 이용해서 (c)와 같이 축소하였다. 즉 움직일 수 없는 공간을(23단계의 경우 18개의 공간으로 공을 이동시킬 수 없음) 고려 대상에서 모두 제외하였다. 그 결과 상태 수가 $39*2^{(39)\ 18)}$로 줄어들었다. 축소 전의 모델과 축소 후의 모델은 검증할 시제 식의 관점에서 동치이다. 그러나 512MB 메모리로 전체 상태를 담을 수 없었다. 그래서 메모리를 증설하는 한편 Chan[16]의 방법과 같이 불필요한 탐색 공간을 잘라내기(pruning) 하였다. 그 결과 79초만에 모델 검증을 완료하였으며 사용된 메모리도 67MB였다. 추상화와 잘라내기 등의 영리한 기법을 사용하여 모델 검증한 결과가 표 3에 있다. 일반적인 모델 검증에 비해서 영리한 기법을 사용했을 때, 시간 및 메모리 사용량이 크게 절감되었다 (그림 5와 그림 6 참조).

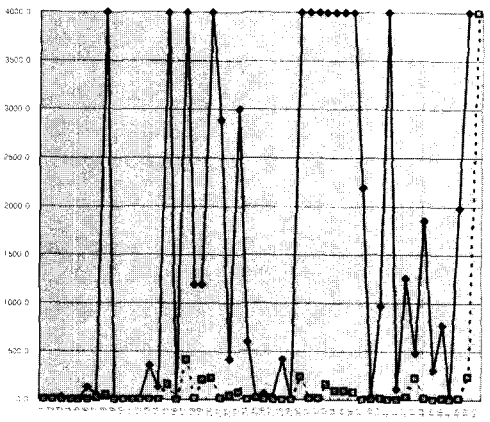


그림 5 시간 절감

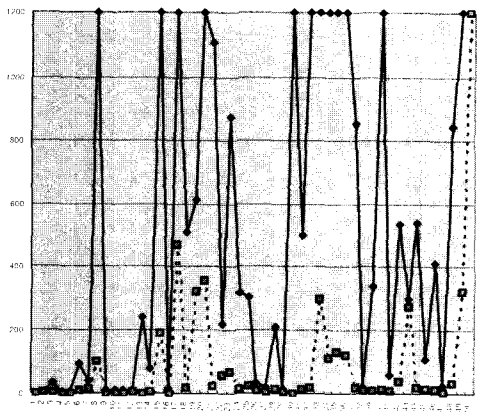


그림 6 메모리 절감

그러나 여러 가지 영리한 기법을 사용하였음에도

불구하고 마지막 단계인 50 단계는 풀지 못하였다. 왜냐하면 상태 공간이 매우 크기 때문이었다. 50 단계를 풀기 위해서 '계주 모델 검증(Relay Model Checking)' 기법을 사용하였다. 계주 경기에서 주자가 다음 주자에게 바톤을 넘겨주듯이, 전체 문제를 여러 개의 부분 문제로 나눈 후, 순서대로 검증하였다. 여기서는 50 단계를 4개로 나누어 계주 검증을 하였고, 그 결과는 표 4와 같다. 표 3에서 보듯이 전체 문제를 한번에 검증하고자 하였을 때, 하루가 지나도 풀리지 않던 것이 계주 검증 기법을 사용한 결과 1,920초만에 해결되었다.

표 4 계주 검증 기법을 사용했을 때
소요된 시간 및 메모리(50단계의 경우)

단계	회	수행시간 (sec)	메모리 (MB)
50	1	420.0	460
	2	600.0	764
	3	360.0	454
	4	540.0	762

5. 결 론

여기에서는 모델 검증 기법을 문제 풀이(problem solving)에 적용하였다. 문제를 유한 상태 모델로 표현하고 오답을 시제 논리식으로 표현해서 모델 검증기에 입력하면, 모델 검증기는 반례로 오답에 대한 부정 즉 정답을 출력한다. 다시 말해서 반례로 출력된 것이 문제를 해결하는 정답인 것이다.

이와 같은 아이디어를 푸쉬 푸쉬 게임에 적용하여 1 단계부터 50단계를 모두 해결하였다. 푸쉬 푸쉬 게임 풀이는 규칙을 준수하면서 시작 상태를 목표 상태로 옮기는 이동 경로를 찾는 것이다. 본 방법으로 찾아낸 이동 경로 수는 전문가의 움직임에 비해 평균 22회 적었다. 바꾸어 말해서 전문가보다 적은 움직임으로 게임을 풀었다. 게임 풀이의 가장 큰 장애 요소는 상태 폭발 문제였다. 상태 폭발을 방지하기 위해서 자동화, 잘라내기, 계주 모델 검증 등의 영리한 방법을 사용하였다. 더불어서 메인 메모리도 512MB에서 1.5GB 로 확장하였다. 이와 같이 다양한 방법을 활용한 덕분에 푸쉬 푸쉬 게임을 해결할 수 있었다.

모델 검증과 같은 이론적인 연구가 산업체에 실질적으로 활용되기 위해서 반드시 풀어야 할 문제가 상태 폭발과 같은 규모 처리 문제이다. 왜냐하면 산업체에서 실제적으로 처리하려는 모델의 크기는 매우 거대하기 때문이다.

참고문헌

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

[2] J. Bormann, J. Lohse, M. Payer, G.Venzl, "Model Checking in Industrial Hardware Design", In Proceedings of DAC' 95, pp.298-303, 1995.

[3] K. Laster, O. Grumberg, Modular model checking of software, In Proceedings of TACAS'98, LNCS 1384, pp.20-35, 1998.

[4] G. Brat, K. Havelund, S.J. Park, W. Visser, Model Checking Programs, In Proceedings of ASE, 2000.

[5] E.M. Clarke, S. Jha, Y. Lu, H. Veith, Tree-like Counterexamples in Model Checking, In Proceedings of LICS'02, 2002.

[6] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided Abstraction Refinement, In Proceedings of CAV'00, 2000.

[7] Artificial Intelligence and Games, see http://www.brl.ntt.co.jp/people/kojima/links/ai-game.html

[8] I.H. Moon, J. Kukula, T. Shiple, F. Somenzi, Least Fixpoint Approximations for Reachability Analysis, In Proceedings of the ICCAD, pp.41-44, 1999.

[9] E.M. Clarke, E.A. Emerson, 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] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Progress on the State Explosion Problem in Model Checking, In Proceedings of

10 Years Dagstuhl, LNCS 2000, pp.154-169, 2000.

[12] R.E. Bryant, Graph-Based Algorithms for Boolean Function Manipulation", IEEE Transactions on Computer, Vol. 35, No. 8, pp.677-691, 1986.

[13] E.M. Clarke, O. Grumberg, D.E. Long, "Model Checking and Abstraction", ACM Transac tions on Programming Languages and Systems, Vol. 16, No. 5, pp.1512-1542, 1994.

[14] D. Peled, Combining Partial1. Order Reductions with on-the-fly Model Checking, Formal Methods in System Design, No. 8, pp.39-64, 1996.

[15] S. Berezin, S. Campos, E.M. Clarke, Compositional Reasoning in Model Checking, Technical Report CMU-CS-98-106, Carnegie Mellon University, 1998.

[16] W. Chan, R.J. Anderson, P. Beame, D. Notkin, Improving Efficiency of Symbolic Model Checking for State-Based System Requirements, In Proceedings of ISSTA'98, pp.102-112, 1998.

권 기 현

1985 경기대학교 전자계산학과 학사
1987 중앙대학교 전자계산학과 석사
1991 중앙대학교 전자계산학교 박사
1998~1999 독일 드레스덴 대학 전자계산학교 방문교수
1999~2000 미국 카네기 멜론 대학 전자계산학과 방문교수
1991~현재 경기대학교 정보과학부 교수
관심분야: 정형 기법, 시제 논리, 모델 검증, 정리 증명
E-mail : khkwon@kyonggi.ac.kr