한국게임학회 논문지
제4권 제1호 2004년 3월

게임 풀이를 위한 상태 공간 축소

이태훈*, 권기현**
경기대학교 정보과학부
{taehoon*, khkwon**}@kyonggi.ac.kr

Reductions of State Space for Solving Games

Taehoon Lee, Gihwon Kwon
Department of Computer Science, Kyonggi University

요 약

본 논문에서는 반례를 도달성 게임 풀이에 활용했다. 도달성 게임이란, 게임 규칙을 준수하면서 초기 상태에서 목표 상태로 가는 경로를 찾는 것이다. 게임의 초기 상태를 유한 상태 모델로 표현하고 목표 상태의 부정(오답)을 시제 논리식으로 표현해서 모델 검사기에 입력하면, 반례로 오답에 대한 부정 즉 정답을 출력한다. 다시 말해서 반례로 출력된 것이 문제를 해결하는 정답인 것이다. 이와 같은 아이디어를 푸쉬푸쉬 게임 풀이에 적용했다. 그 결과 크기가 작은 푸쉬푸쉬 게임의 경우 게임을 해결하는 최단 경로를 쉽게 찾아냈다. 그러나 게임의 크기가 큰 경우 제한된 시간(실험에서는 3시간)내에 찾아내지 못했다. 왜냐하면 게임의 크기가 커질 경우 검사해야할 상태의 수가 지수적으로 증가하는 상태 폭발 문제가 발생했기 때문이다. 상태 폭발 문제를 해결하기 위해 추상화 기법을 이용해서 검사해야 할 상태 공간을 축소하였다. 상태 공간을 줄인 결과, 기존 모델체킹으로 해결하지 못했던 게임을 풀 수 있었다.

Abstract

This paper uses counterexamples for solving reachability games. An objective.of the game we consider here is to find out a minimal path from an initial state to the goal state. We represent initial states and game rules as finite state model and the goal state as temporal logic formula. Then, model checking is used to determine whether the model satisfies the formula. In case the model does not satisfy the formula, model checking generates a counterexample that shows how to reach the goal state from an initial state. In this way, we solve many of small-sized Push Push games. However, we cannot handle larger-sized games due to the state explosion problem. To mitigate the problem, abstraction is used to reduce the state space to be che cked. As a result, unsolved g ames are solved with the abstra ction technique we propose inthis paper.

Keyword : ModelChecking , Abstraction , GameSolving , Pushpush game

1. 서 론

본 논문에서는 핸드폰에서 이용가능하고 대중적으로 잘 알려진 푸쉬 푸쉬 게임을 자동으로 해결한다. 이 게임은 일종의 박스 이동 게임으로서, 주어진 박스를 목표 지점으로 모두 옮기는 게임이다. 그러므로 주어진 박스를 목표 지점으로 모두 이동시킬수 있는 경로를 찾아야 한다. 게다가, 최단 경로를 찾는 것이 본 연구의 주 관심사이다.

최단 경로를 찾기 위해서 본 논문에서는 모델 체킹 기술을 이용한다. 모델 체킹은 유한 상태 모델과 시제 논리식을 입력 받은 후, 모델과 논리식간의 만족성 관계를 검사한다 [1]. 만일 모델이 논리식을 만족하면 참을 출력한다. 그렇지 않은 경우, 거짓과 함께 반례를 출력한다. 반례는 모델이 왜 논리식을 만족하지 않는지에 대한 설명으로서, 디버깅 작업에 유용한 정보를 제공한다. 따라서 반례 생성은 모델 체킹의 주요 이점 중의 하나이다 [2]. 예를 들어 그림 1에서 모델이 CTL (Computation Tree Logic, [3]) 논리식 $AG\neg\phi$[2] 를 만족하지 않는 경우를 살펴 보자 (그림에서 ○은 $\neg\phi$인 상태를, ●은 ϕ인 상태를 나타냄). 만약 도달 가능한 상태 중에서 ϕ가 참이 되는 상태가 최소한 하나라도 존재한다면 논리식 $AG\neg\phi$는 거짓이다. 이 경우, 모델 체킹은 초기 상태로 부터 ϕ가 참인 상태로 도달되는 경로를 보여준다. 이러한 경로를 반례라고 하고, 반례를 통해서 논리식이 만족되지 않는 이유를 알 수 있다. 그림 1의 경우, 논리식 $AG\neg\phi$의 반례로서 경로 1, 2, 3, 4, 5 가 출력된다.

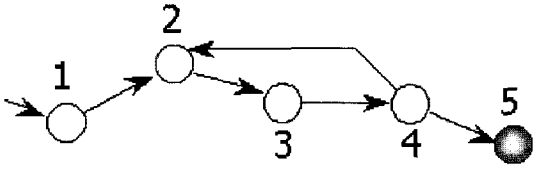


그림 1. $AG\neg\phi$가 거짓인 경우

반례는 유용한 정보를 제공하기 때문에 이를 활용한 연구들이 많다. Clarke[4]은 반례를 이용한 자동 추상화 기법을 연구하였고, Ammann[5]은 반례로부터 테스트 케이스 생성 방법을 연구했다. 본 연구에서는 반례를 이용해서 푸쉬 푸쉬 게임을 풀이한다. 푸쉬 푸쉬 게임은 제약 조건(또는 게임 규칙)을 준수하면서 초기 상태를 목표 상태로 옮기는 것이다. 게임을 풀기 위해서, 게임과 관련된 사항(예를 들어 초기 상태, 게임 규칙, 상태 공간등)을 유한 상태 모델로 표현하고, 목표 상태에 절대 도달할 수 없음을 CTL 논리식으로 표현해서 모델 체킹에 입력한다. 만일 목표 상태로 가는 경로가 게임에 존재한다면, CTL 논리식은 거짓이 되고 이때 모델 체킹은 거짓과 함께 목표 상태로 도달되는 경로를 반례로 출력한다. 다시 말해서 반례로 출력된 경로가 게임을 푸는 정답인 것이다. 이와 같은 아이디어를 이용한 결과, 크기가 적은 푸쉬 푸쉬 게임 풀이의 경우 게임을 해결하는 최단 경로를 쉽게 찾아냈다 [6]. 그러나, 게임의 크기가큰 경우에는 제한된 시간(실험에서는 3시간)내에 최단 경로를 찾아내지 못했다. 왜냐하면 게임의 크기가 커질수록 검사해야 할 상태의 수가 지수적으로 증가하는 상태 폭발 문제가 발생했기 때문이다 [7]. 상태 폭발 문제를 해결하기 위해 본 논문에서는 추상화기법을 이용해서 검사해야 할 상태 공간을 축소하였다. 다시 말해서, 추상화 기법을 이용해서 모델로부터 원래 행위를 보존하는 추상화된 모델을 생성하고, 추상화된 모델상에서 모델 체킹을 수행하여 게임의 풀이 경로를 구했다. 그 결과, 기존 모델 체킹으로 해결하지 못했던 게임을 풀 수 있었다. 본 논문에서는 추상화된 모델과 원래 모델간의 논리식 보존 관계에 대해서 정형적으로 표현하고 보존이 된다는 것을 증명을 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 이 논문을 이해하기 위한배경지식에 대해서 살펴본다. 3장에서는 모델 체킹 기술을 어떻게 게임 풀이에 적용했는지 살펴본다. 4장에서는 추상화를 적용해서 어떻게 상태 폭발 문제를 해결할 수 있는지를 살펴본다. 5장에서는 실험 결과에 대해 살펴보고, 6장에서 결론 및 향후 연구 과제를 기술한다.

1) 본 연구는 과학기술부목적기초연구 (R05-2004-000-10612-0) 지원으로 수행되었음.

2) $AG\neg\phi$는 도달가능한 모든 상태에서 ϕ가 항상 거짓임을 의미한다.

2. 배경지식

2.1 모 델

모델 체킹에 사용되는 모델의 핵심 요소는 상태와 상태 간의 전이이며, 이들을 사용하여 시스템의 행위를 모델링 한다. 특히 CTL(Computation Tree Logic) 모델 체킹에서는 크립키 구조라 불리는 모델 $M=(S,I,R,L)$을 사용한다 [8]. 여기서, S는 상태들의 집합, $I \subseteq S$는 초기 상태들의 집합, $R \subseteq S \times S$는 상태들 간의 전이 관계, $L: S \rightarrow 2^{AP}$는 각 상태에서 참이 되는 단순 명제들을 해당 상태에 배정하는 함수이다. 여기서 AP는 단순 명제들의 집합이다. 모델 체킹은 시스템이 갖는 무한 행위에 대해서 조사를 하기 때문에 상태들 간의 전이를 나타내는 R은 전체 관계이다. 즉 $\forall s \in S \bullet \exists s' \in S \bullet (s,s') \in R$ 로서, 모든 상태마다 전이할 수 있는 다음 상태가 최소한 하나 이상 존재한다. 경로 $\pi = s_1 s_2 s_3 s_4 \cdots$는 전이 가능한 상태들을 차례대로 나열한 것으로서 $(s_i, s_{i+1}) \in R$, $i \geq 1$이며 그 길이는 무한이다.

2.2 속 성

모델에 관한 속성은 모델을 트리의 관점에서 해석하는 CTL 논리식으로 표현한다. 모델의 초기 상태를 루트로 해서 모델을 풀어헤치면 트리를 얻게 되며, 트리는 모델의 가능한 모든 행위를 표현한다. 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.

전 장에서 언급했듯이 본 논문에서 사용하는 주요 CTL 논리식은 AG이며 이것의 쌍대는 EF 이다.[3] 따라서 본 절에서는 두 연산자에 국한해서 설명한다.

AGφ 의미는 '도달 가능한 모든 상태에서 항상 φ' 이며, EFφ의 의미는 '언젠가는 φ인 상태로 도달 가능하다' 이다. 모델 M의 상태 s에서 CTL논리식 φ가 참인 경우를 $M,s \models \phi$ 로 표시한다. 그렇치 않다면 $M,s \not\models \phi$ 로 표시한다. 상태 s에서 AGφ, EFφ 의 정형적 의미는 다음과 같다:

$$M,s \models AG\phi \quad \text{iff} \quad \forall \pi = s_1, s_2, \cdots \bullet \forall i \geq 1 \bullet M, s_i \models \phi$$
$$M,s \models EF\phi \quad \text{iff} \quad \exists \pi = s_1, s_2, \cdots \bullet \exists i \geq 1 \bullet M, s_i \models \phi$$

2.3 모델 체킹 알고리즘

모델 M의 모든 초기 상태에서 CTL 논리식 φ가 참인 경우를 $M \models \phi$으로 표시하며 'M이 φ를 만족한다' 라고 읽는다. 모델 체킹은 M과 φ를 받아서 이들간의 만족성 관계를 결정한다. 이를 위해서 φ를 만족하는 상태 집합을 구한후, 이 상태 집합에 초기 상태가 포함되어 있는지를 검사한다. CTL 논리식 φ를 만족하는 상태 집합을 $[\![\phi]\!]$ 라고 하자. AGφ, EFφ를 만족하는 상태 집합은 다음과 같이 역 방향으로 계산된다.

$$[\![AG\phi]\!] = \nu Z.([\![\phi]\!] \cap pre_{\forall}(Z))$$
$$[\![EF\phi]\!] = \mu Z.([\![\phi]\!] \cup pre_{\exists}(Z))$$

여기서 μ,ν는 최소 고정점과 최대 고정점이다. 상태 집합 $[\![\phi]\!]$를 계산할 때 최소 고정점 μ는 공집합을 초기값으로 하여 계속해서 증가되다가 더 이상 증가하지 않는 집합을 구할 때 사용하며 반대로 최대 고정점 ν는 전체 집합을 초기값으로 하여 계속해서 감소하다가 더 이상 감소하지 않는 집합을 계산할 때 사용한다[4]. AGφ, EFφ의 상태집합 계산에 사용된 역방향 탐색 함수는 다음과 같다.

$$pre_{\forall}(Q) = \{ s \in S \mid \forall_{s' \in Q} \bullet (s,s') \in R \}$$
$$pre_{\exists}(Q) = \{ s \in S \mid \exists_{s' \in Q} \bullet (s,s') \in R \}$$

함수 $pre_{\forall}$는 집합 Q 로만 도달되는 이전 상태들의 집합을 출력한다. 함수 $pre_{\exists}$는 $pre_{\forall}$와 비슷하지만 Q로 도달되는 이전 상태들을 모두 출력한다는 점에서 다르다. CTL 논리식 φ에 대한 상태집합 [φ] 를 구한후 모델 체킹 알고리즘의 핵심은 초기 상태 집합 I 가 [φ] 의 부분집합인지를 검사하는 것이다. 즉 $M \models \phi$ iff $I \subseteq [\phi]$ 이다.

3) $\neg\phi = \psi$인 경우 φ,ψ는 쌍대(dual) 관계이다. $\neg(AG\neg\phi) = EF\phi$ 이기 때문에 두 식은 쌍대이다.
4) 최소 고정점 μ는 EF,EU,AF,AU를 계산할 때 사용되며, 최대고정점 ν는 EG,AG를 계산할 때 사용한다.

2.4 반례 생성

$M \models \phi$인 경우, 다시 말해서 $I \not\subset [\phi]$ 인 경우 모델 체킹은 그 이유를 담은 반례를 생성한다. 여기서는 $M \not\models AG\neg\phi$의 반례 생성을 설명한다. 위에서 설명한대로 AG $\neg\phi$의 쌍대는 EF ϕ이기 때문에, AG $\neg\phi$의 반례는 EF ϕ의 증거이다. 그러므로 EF ϕ의 증거로서 AG $\neg\phi$의 반례를 설명할 수 있다. 정의한대로 EFϕ를 만족하는 것은 $[EF\phi] = \mu Z.([\phi] \cup pre_{\exists}(Z))$ 이기 때문에, 이것은 함수 $\tau(Z)=([\phi] \cup pre_{\exists}(Z))$의 최소 고정점이다. 공집합으로부터 시작해서 함수 τ를 반복적으로 적용함으로서 최소 고정점을 구할 수 있다. 즉, 함수를 적용한 순서 $\tau^1(\phi) \subseteq \ldots \subseteq \tau^n(\phi) \subseteq \ldots$ 는 언젠가 더 이상 증가되지 않는 상태 $\tau^n(\phi)=\tau^{n+1}(\phi)$에 이르게 되는데, 이때 $\tau^n(\phi)$ 이 함수 τ의 최소 고정점이 된다. 함수 τ의 중간 계산 값을 $S_1=\tau^1(\phi), S_2=\tau^2(\phi), \ldots, S_n=\tau^n(\phi)$라고 하자. 사실, $S_1=\tau^1(\phi)=[\phi]$ 이다. 왜냐하면 $pre_{\exists}(\phi)=\phi$ 이기 때문이다. 그림 1에서 보듯이 $S_n \cap I \neq \emptyset$ 이면 AG $\neg\phi$는 거짓이다.

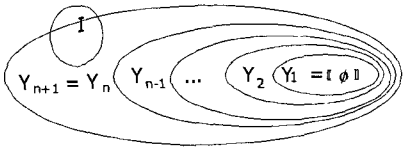


그림 1. $S_n \cap I \neq \emptyset$ 이면 AG $\neg\phi$는 거짓이다.

거짓인 경우라면, 그림 2에서 보는 것처럼 두 단계를 거쳐서 반례를 생성한다. 첫번째 단계는

$S_1 = I$

$S_{i+1} = post_{\exists}(S_i) \cup S_i$

와 같이 초기 상태에서 ϕ가 참인 상태까지 정방향 탐색을 진행하면서 도달 가능한 상태를 저장한다 ($S_n \cap I \neq \emptyset$ 일 때 종료한다). 여기서 $post_{\exists}(Q) = \{s' \in S \mid \exists_{s \in Q} \cdot (s, s') \in R\}$는 Q로 부터 도달 가능한 다음 상태들의 집합을 리턴하는 함수이다. 두 번째 단계는 ϕ가 참인 상태에서 초기 상태로 역방향으로 오면서 초기 상태에서 ϕ가 참인 상태로 도달 가능한 경로 즉 반례 $<s_1,\cdots,s_n>$를 찾는다.

$s_n \in S_n \cap [\phi]$

$s_{i-1} \in pre_{\exists}(S_i) \cap S_{i-1}$

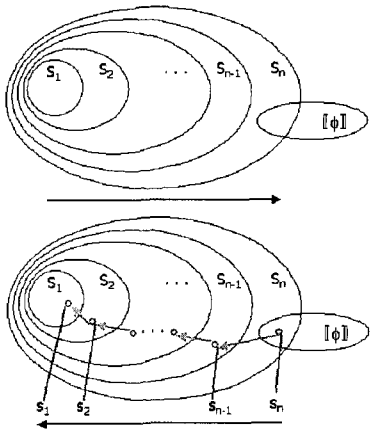


그림 2. 정방향과 역방향 탐색을 통해서 반례 생성

3. 게임 풀이

본 논문에서는 범용 모델 검사 도구인 NuSMV[9]를 이용해서 푸쉬푸쉬 게임을 해결한다. 이 게임은 일종의 블록 이동 게임으로서 주어진 공을 목표 지점으로 모두 옮기는 게임이다. 그러므로 게임의 목표는 목표지점으로 옮기는 이동 경로를 찾는 것이다. 이동 경로를 모델 검사 기법으로 찾아내기 위해서는 유한 상태모델과 CTL 속성을 기술해야 한다. 먼저 에이전트의 행위를 유한 상태기계 $T_{agent}=(S_{agent}, I_{agent}, \delta_{agent})$로 모델링 하면 다음과 같다.

- $S_{agent}=\{(x,y)\}$는 이동 가능한 위치들의 집합이다. 위치는 순서쌍 (x,y) 로 표현된다.
- $I_{agent} \in S_{agent}$는 에이전트의 초기 위치이다.
- δ_{agent}: $S_{agent} \times D \rightarrow S_{agent}$는 전이 함수이다

여기서 $D=\{left, right, up, down\}$는 에이전트의 이동 방향을 나타낸다. 움직일 수 있는 방향은 매번 하나이기 때문에 방향을 집합으로 모델링 하였다. 전이 함수 $\delta_{agent}((x,y),d)=(x',y')$는 아래와 같이 정의 된다.

$$\delta_{agent}((x,y),d) = \begin{cases} (x'=x-1, y'=y) \\ \quad \text{if } d = left \wedge movableToLeft(x,y) \\ (x'=x+1, y'=y) \\ \quad \text{if } d = right \wedge movableToRight(x,y) \\ (x'=x, y'=y+1) \\ \quad \text{if } d = up \wedge movableToUp(x,y) \\ (x'=x, y'=y-1) \\ \quad \text{if } d = down \wedge movableToDown(x,y) \\ (x'=x, y'=y) \\ \quad \text{otherwise} \end{cases}$$

여기서 술어 $movableToLeft(x,y)$는 (x,y)에 위치한 에이전트가 바로 왼쪽 위치 $(x\text{-}1,y)$로 이동 가능함을 나타내며 다음과 같이 정의된다.

$$borderLeft(x,y)=(x\text{-}1,y)\notin S_{agent}$$

$$borderLeftBallLeft(x,y)=$$
$$(x\text{-}2,y)\notin S_{agent}\wedge\exists_{cell}\bullet pos(cell)=(x\text{-}1,y)$$

$$ballLeftBallLeft(x,y)=\exists_{cell1,cell2}\bullet$$
$$(pos(cell_1)=(x\text{-}2,y)\wedge cell_1)\wedge pos(cell_2)=(x\text{-}1,y)\wedge cell_2)$$

$$movableToLeft(x,y)=$$
$$\neg borderLeft(x,y)$$
$$\wedge\neg borderLeftBallLeft(x,y)$$
$$\wedge\neg ballLeftBallLeft(x,y)$$

여기서 보조 함수 pos는 셀의 현재 위치를 출력한다. 나머지 *movableToRight*, *movableToUp*, *movableToDown* 술어들도 비슷하게 정의된다.

게임에는 셀들이 있다. 셀은 비어있거나 또는 공을 갖고 있다. 매번 게임이 진행될 때 마다 셀의 상태가 변경된다. 임의의 셀 $cell_i$의 행위를 상태 기계 $T_{cell}=(S_{cell}, I_{cell}, \delta_{cell})$로 모델링 하면 다음과 같다:

- $S_{celli}=B$ 는 부울 변수이다. 즉 1이면 공을 갖고 있고, 0이면 비어 있다.
- $I_{celli}\in S_{celli}$ 는 $cell_i$의 초기 상태를 나타낸다.
- δ_{celli}: $S_{celli}\times D\rightarrow S_{tcelli}$는 전이 함수를 나타낸다.

여기서 전이 함수 $\delta_{celli}:B\times D\rightarrow B$ (왜냐하면 $S_{celli}=B$)는 다음과 같이 정의된다.

$$\delta_{cell_i}(c,d)=\begin{cases}0 & \text{if } c\wedge d=left\wedge\exists_{agent,cell}\bullet pos(agent)=(x+1,y)\\ & \quad\wedge pos(cell)=(x-1,y)\wedge\neg cell\\ 1 & \text{if } \neg c\wedge d=left\wedge\exists_{agent,cell}\bullet pos(agent)=(x+2,y)\\ & \quad\wedge pos(cell)=(x+1,y)\wedge cell\\ 0 & \text{if } c\wedge d=right\wedge\exists_{agent,cell}\bullet pos(agent)=(x-1,y)\\ & \quad\wedge pos(cell)=(x+1,y)\wedge\neg cell\\ 1 & \text{if } \neg c\wedge d=right\wedge\exists_{agent,cell}\bullet pos(agent)=(x-2,y)\\ & \quad\wedge pos(cell)=(x-1,y)\wedge cell\\ 0 & \text{if } c\wedge d=up\wedge\exists_{agent,cell}\bullet pos(agent)=(x,y-1)\\ & \quad\wedge pos(cell)=(x,y+1)\wedge\neg cell\\ 1 & \text{if } \neg c\wedge d=up\wedge\exists_{agent,cell}\bullet pos(agent)=(x,y-2)\\ & \quad\wedge pos(cell)=(x,y+1)\wedge cell\\ 0 & \text{if } c\wedge d=up\wedge\exists_{agent,cell}\bullet pos(agent)=(x,y+1)\\ & \quad\wedge pos(cell)=(x,y-1)\wedge\neg cell\\ 1 & \text{if } \neg c\wedge d=up\wedge\exists_{agent,cell}\bullet pos(agent)=(x,y+2)\\ & \quad\wedge pos(cell)=(x,y+1)\wedge cel\\ c & \text{otherwise}\end{cases}$$

에이전트의 상태 기계와 셀의 상태 기계를 동기식으로 결합하면 게임 전체의 상태 기계 $T_{game}=T_{agent}\otimes T_{cell1}\otimes\ldots\otimes T_{celln}$를 얻는다. 여기서 기호 $\otimes$는 동기식 결합 연산자이다. 즉, 게임의 행위는 상태 기계 $T_{game}=(S_{game}, I_{game}, D, \delta_{game}, F_{game})$으로 모델링 할 수 있다:

- $S_{game}=S_{agent}\times S_{cell1}\times\ldots\times S_{celln}$는 게임이 갖는 상태 공간이다.
- $I_{game}=(I_{agent}\times I_{cell1}\times\ldots\times I_{celln})$는 게임의 초기 상태이다.
- $\delta_{game}:S_{agent}\times S_{cell1}\times\ldots\times S_{celln}\times D\rightarrow S_{agent}\times S_{cell1}\times\ldots\times S_{celln}$ 는 상태 전이 함수이다. 즉, $\delta_{game}((x,y),c_1,\ldots,c_n,d)=(\delta_{agent}((x,y),d),\delta_{cell1}(c_1,d),\ldots,\delta_{celln}(c_n,d))$이다.
- F_{game}는 목표 셀들의 집합이다.

이제까지 게임에 있는 개체와 그들의 행위를 유한 상태 기계로 모델링 하였다. 이들 모델의 의미는 크립키 구조로 $M=(S, I, R, X, L)$으로 정의할 수 있다:

- $S=S_{agent}\times S_{cell1}\times\ldots\times S_{celln}\times D$는 상태 공간의 집합이다.
- $I=I_{agent}\times I_{cell1}\times\ldots\times I_{celln}\times\{d\in D\}$는 초기 상태 집합이다.
- $R((x,y),c_1,\ldots,c_n,d,\delta_{agent}((x,y),d),\delta_{cell1}(c_1,d),\ldots,\delta_{celln}(c_n,d),d\in D)$는 전이 관계이다.
- $X=\{cell_1,\ldots,cell_n\}$는 단순 명제들의 집합이다.
- $L(S)=L((x,y),c_1,\ldots,c_n,d)=\{cell_1\mid c_1\}\cup\ldots\cup\{cell_n\mid c_n\}$는 라벨 함수이다.

진술한 바와 같이, 푸쉬 푸쉬 게임 풀이란 초기 상태에서 목표 상태로 공을 모두 이동시키는 경로를 찾는 것이다. 목표 상태는 $F_{game}=\{cell_1,\cdots,cell_n\}$ 이다. 즉 게임마다 목표 상태에 도달하는 경로가 최소한 하나 이상 존재한다. 만약 그러한 경로가 결코 존재하지 않는다고 하면 이는 틀린 것이 되고 모델 검사는 이에 대한 반례로 초기 상태에서 목표 상태로 가는 경로를 생성한다. 이러한 경로가 푸쉬 푸쉬 게임을 푸는 답이 되는 것이다. 게임을 풀 수 있는 답을 유도하기 위해서 '목표 상태로 갈 수 있는 경로가 결코 존재하지 않는다'를 CTL로 표현하면

$$AG\ \neg\phi\equiv\bigwedge_{i=1}^{|F_{game}|}cell_i\in F_{gam}$$ 이다. 여기서 이와 같은 아이디어를 푸쉬 푸쉬 게임의 최단 풀이 경로를 찾아내었다. 그러나 표1에서 보듯이 아직도 많은 부분을 이러한 기법으로 풀지 못했다. 왜냐하면 게임이 복잡할수록 최단 풀이 경로를 구하는데 고려되는 상태들도 매우 증가되는 상태 폭발 문제가 발생했기 때문이다.

4. 추상화를 통한 상태 공간 축소

추상화는 원래의 시스템의 행위를 가지고 있는 좀더 적은 크기의 모델을 생성한다 [10]. 시스템의 크기는 작지만 동일한 행위를 하는 모델을 동치 관계에 있다고 한다. 추상화된 모델이 원래 모델보다 좀더 많은 행위를 가지고 있다면 원래 모델과 상위 근사화 (over approximation) 관계에 있는 추상화된 모델이 생성이 된 것이다. 만일 원래 모델보다 작은 행위를 가지고 있는 추상화된 모델을 생성했다면 하위 근사화(under approximation) 관계에 있는 모델을 생성한 것이다. 이 관계를 가지고 있는 추상화된 모델을 생성했다면 상위 근사화 관계에 있는 모델은 원래 모델의 행위를 모두 따라 할 수 있다. 그리고 추가적인 행위를 가지고 있게 된다. 비록 행위의 수는 많게 되지만 일반적으로 추상화 된 모델의 크기는 원래 모델의 크기보다 적은 모델을 생성할 수 있다. 이 경우 추상화된 시스템에서 속성을 만족한다면 적은 행위를 가지는 원래 시스템 에서도 속성을 만족한다. 하지만 속성을 만족하지 않는다면 그것이 원래 모델에 있는 행위 때문에 속성을 만족하지 않는 것인지, 추가된 행위 때문에 속성을 만족하지 못하는 것인지 모르게 된다.

한편, 하위 근사화 관계에 있는 추상화된 모델은 원래 시스템에 비해 추상화된 시스템이 적은 행위를 가지고 있다. 따라서 적은 행위에서 속성을 만족한다고 해서 원래 시스템에서 속성이 만족 된다고 할 수는 없다. 추상화된 모델에서 속성이 만족 되지 않는다면 추상화된 모델 보다 더욱 많은 행위를 가지고 있는 원래 모델에서는 당연히 속성을 만족하지 않게 된다.

원래 시스템의 속성을 보존하고 있으면서 상태공간을 축소 할 수 있는 방법에 대해서 아래에서 설명을 한다. 두개의 모델 M, M'이 있을때 두 모델간의 관계에 대해서 살펴보자. 두 모델의 행위가 동일할 경우 두개의 모델은 바이시뮬레이션 관계에 있다고 하며 $M \equiv M'$와 같이 표현한다. 만일 두 모델중에 하나의 모델이 다른 하나의 모델의 행위를 흉내낼수 있다면, 두 모델은 시뮬레이션 관계에 있으며 $M \leq M'$ 로 표시한다. 바이시뮬레이션 관계에 있는 경우, 모든 CTL 식 ϕ 은 두 모델 사이에서 보존된다. 즉, $M \models \phi \Leftrightarrow M' \models \phi$ 이다. 하지만 이런 관계를 가지는 모델에서는 모델의 크기를 크게 축소하기는 어렵다. 시뮬레이션 관계에 있는 경우 모든 ACTL[5]식 ϕ에 대해 $M \models \phi \Rightarrow M' \models \phi$ 가 성립한다. 즉, M이 ϕ를 만족하면 M'도 ϕ를 만족한다. 하지만 그 반대의 경우는 성립하지 않는다.

이제 본 논문에서 제안하는 상태 축소 방법을 제시한다. 먼저, 모델은 상태 변수 $V=\{v_1,...,v_n\}$를 갖는다. 각 변수 v_i 가 가질 수 있는 값의 범위를 도메인 D_i 라 부르며, 변수는 도메인 값중의 하나를 항상 갖는다. 즉,

$$v_i=d_i$$

이다. $\sigma_i: \{v_i\} \rightarrow D_i$는 변수 v_i에 값을 배정하는 함수로서 아래와 같이

$$\sigma_i(v_i)=d_i \in D_i$$

정의된다. $\sigma: \{v_1\} \times ... \times \{v_n\} \rightarrow D_1 \times ... \times D_n$는 상태를 나타내는 함수로서 아래와 같이

$$\sigma(v_1,...,v_n)=(\sigma_1(v_1),...,\sigma_n(v_n))=(d_1,...,d_n)$$

정의된다. 따라서 모델이 변수 집합 V를 가질때, 모델의 전체 상태 공간은 $S=D_1 \times ... \times D_n$이다. $s=(d_1,...,d_n)$를 모델의 한 상태라고 하자. 부분 집합 $V' \subseteq V$ (여기서 부분 집합의 크기는 $m=|V'|$ 이며 $m<n$) 이 주어졌을때, 상태 s를 집합 V' 으로 투영하는 하는 것은 다음과 같이

$$proj(s,V')=(d_1,...,d_m) \text{ such that } \sigma_i^{-1}(d_i) \in V' \text{ for all } i$$

정의된다. 이것을 상태들의 집합으로 확장한 투영 함수는 다음과 같다.

$$proj(S,V')=\{proj(s,V') \mid s \in S\}$$

본 논문에서 다루는 모델 M은 이진 변수의 집합 $V=\{v_1,...,v_n\}$을 갖는다. 각 변수 v_i 는 게임에서의 하나의 셀을 나타낸다. 셀은 공을 가질수도 있고, 공을 갖지 않을수도 있기 때문에 이진 변수로 나타낼 수 있고, 그래서 변수에 값을 배정하는 함수의 정의는 $\sigma_i: \{v_i\} \rightarrow B$ 이다. 상태 변수 집합 V 는 크게 두 부분 집합으로 분할할 수 있다.

$$V = V_p \cup V_d$$

5) CTL 논리식의 일부분으로서, 모든 경로를 나타내는 A가 항상 먼저 나오는 논리식을 ACTL 식이라 부른다. 보다 상세한 사항은 [1]을 참조.

여기서 V_p는 공을 움직일 수 있는 이동가능한 셀들을 나타내고, V_d는 한 번 공이 위치하면 더이상 진행할 수 없는 셀들을 나타낸다. 줄여서 V_d를 데드락 변수 집합이라고 하자. 함수 $sel(S,V_d)$는

$$sel(S,V_d)=\{(d_1,...,d_n) \mid \text{there is some } d_i \text{ such that } \sigma_i^{-1}(d_i)\in V' \wedge d_i=1\}$$

데드락 변수에 공이 위치하는 상태를 리턴하는 함수이다. $V'=V_p$ 일때 원래 모델 M의 하위 근사화인 $M'=<S',I',R',L'>$를 다음과 같이 구할수 있다.

- $S' = proj(S\text{-}sel(S,V_d),V')$
- $I' = proj(I,V')$
- $R' = \{(s,s') \mid s=proj(s_1,V'),\ s'=proj(s_2,V'),\ (s_1,s_2)\in(sel(S,V_d)R)sel(S,V_d))\}$
- $L'(s')=\{q \mid s'=proj(s,V') \wedge s \models q\}$

따라서 원래 모델 M과 본 논문에서 제안한 축소 모델 M' 사이에는 시뮬레이션 관계인 $M' \leq M$ 이 성립한다. 그래서, 그래서 다음 정리가 성립한다.

정리 : $M' \models EF\phi \Rightarrow M \models EF\phi$

(증명)1. $M' \leq M$

2. $M \models AG\neg\phi \Rightarrow M' \models AG\neg\phi$
3. $M' \not\models AG\neg\phi \Rightarrow M \not\models AG\neg\phi$
4. $M' \models \neg AG\neg\phi \Rightarrow M \models \neg AG\neg\phi$
5. $M' \models EF\phi \Rightarrow M \models EF\phi$

정리에서 보듯이 축소된 모델에서 목표 상태인 ϕ에 도달 할 수 있다면(EFϕ), 원래 모델에서도 도달 가능하다.

5. 실험

이 방법을 이용해서 푸쉬푸쉬 게임을 풀었다. 전장에서 살펴보았듯이 푸쉬푸쉬 게임의 목적은 게임 규칙에 따라서 최종 상태에 도달하는 것이다. 게임을 NuSMV 입력언어 형식으로 변환하고, 최종 상태에 도달할 수 없다는 것을 CTL 시제 논리식으로 입력하면, 모델 체킹은 반례로서 초기 상태에서 목표 상태에 도달하는 최단 경로를 출력한다. 이 경로를 이용하면 모든 푸쉬푸쉬 게임을 풀 수 있다. 게임 풀이에 소요된 시간과 메모리 사용량이 표 1에 있다. 여기서 기

판	추상화 이전		추상화 이후	
	시간(초)	메모리(Mb)	시간(초)	메모리(Mb)
1	0.6	2	0.1	1
2	3.6	6	0.5	5
3	33.8	32	2.6	7
4	0.7	3	0.5	1
5	1.0	3	0.3	1
6	123.3	93	7.5	11
7	53.6	37	8.1	13
8	∞		53.9	105
9	11.0	11	0.3	1
10	5.0	8	0.4	1
11	6.2	8	0.3	1
12	10.7	12	1.5	6
13	355.3	240	0.4	1
14	125.8	80	2.3	7
15	∞		167.8	190
16	16.5	16	0.5	1
17	∞		415.0	470
18	1200.0	508	8.1	21
19	1200.0	613	209.0	325
20	∞		224.0	358
21	2880.0	1109	14.8	26
22	420.0	219	52.5	59
23	3000.0	870	79.0	67
24	600.0	317	7.1	17
25	10.0	305	13.5	25
26	60.0	34	5.4	8
27	30.0	20	3.0	7
28	420.0	210	4.8	13
29	16.0	13	1.4	5
30	∞		248.1	1
31	1020.0	500	8.3	16
32	∞		9.9	20
33	∞		168.2	298
34	∞		101.8	113
35	∞		98.6	133
36	∞		73.6	122
37	2200.0	854	7.7	18
38	28.0	18	3.0	8
39	960.0	336	10.4	11
40	∞		4.3	13
41	120.0	58	3.4	10
42	1260.0	532	25.0	40
43	480.0	299	223.3	272
44	1860.0	539	12.9	18
45	300.0	109	6.5	13
46	780.0	413	10.2	17
47	35.0	25	1.8	5
48	1980.0	841	16.6	35
49	∞		234.2	321
50	∞		∞	

표 1 푸쉬푸쉬 게임 수행 결과

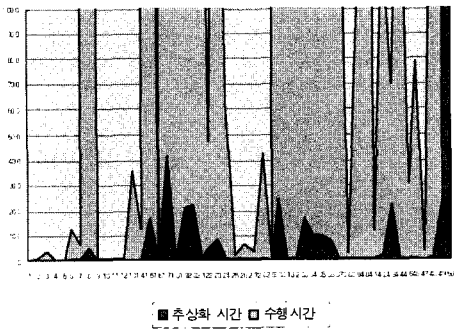


그림 3. 원래 모델과 추상화된 모델간의 시간 비교

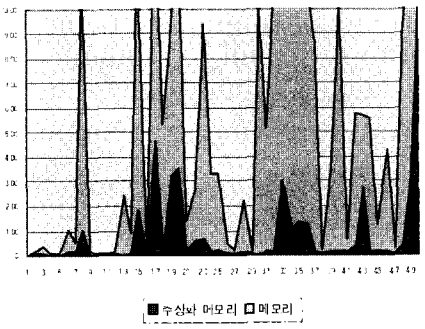


그림 4. 원래 모델과 추상화된 모델간 메모리 비교

호 ∞은 3시간 이상 경과해도 결과를 볼 수 없음을 나타낸다. 실험은 펜티엄 4에서 1 G 메모리를 가진 컴퓨터에서 수행하였고, 추상화 이전에는 총 50 개의 게임 중에서 37 개 게임을 해결하였고, 13 게임은 상태 폭발 문제 때문에 해결할 수 없었다. 그래서 위에서 제시한 추상화 기법을 적용하여 좀더 적은 행위를 가지는 작은 크기의 모델(하위 근사화)을 만들었고, 이를 이용해서 게임을 풀었다. 그 결과 50 개의 게임 중에서 1개만을 제외한 게임을 전부 해결할 수 있었다.

그림 4와 그림 5는 원래 모델과 추상화된 모델간의 수행 시간과 메모리 사용량을 표시한 그래프이다. 그림에서 보듯이 모든 게임에서 원래 모델보다 추상화 모델이 시간과 메모리를 적게 사용하였다. 평균적으로, 추상화를 수행했을 경우에 원래 모델에 비해서 15%의 시간만 사용하고서도 결과를 얻을수 있었고, 또한 24%의 메모리만 사용하고서도 결과를 얻었다. 그러나 추상화를 수행 했지만 마지막 50번째 게임은 3시간이내에 결과를 얻을 수 없었다. 따라서 추상화 이외에 다른 접근 방법이 필요했고, 이를 해결하기 위해 릴레이 모델 체킹을 제안하여 마지막 50판을 풀었다 [11].

6. 결론

모델 체킹을 이용해서 푸쉬 푸쉬 게임을 풀 수 있는 최단 경로를 찾아내었다. 하지만 모델의 크기가 큰 경우, 상태 폭발 문제가 발생하였다. 이를 위해서 본 논문에서는 모델의 크기를 축소하는 추상화 기법을 제시하였고, 이를 이용해서 상태 폭발 문제를 해결하였다.

본 논문에서 예제로 삼은 푸쉬 푸쉬의 경우 전체 50 레벨로 구성되어 있는데, 레벨의 나열이 일관적이지 못하다. 일반적으로 낮은 레벨은 쉬운 반면에 높은 레벨은 복잡하지만, 항상 그렇지만은 않다. 이와 같은 유형의 게임을 나열할 때 게임의 복잡도가 고려되어, 복잡도가 낮은 순서에서 높은 순서로 배열되는 것이 바람직하다. 본 연구에서 찾아낸 최단 경로가 게임의 복잡도를 나타내는 지표가 될 수 있으며, 최단 경로의 길이를 이용한다면 게임을 일관성있게 배열할 수 있을 것이다.

참고문헌

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

[2] E. M. Clarke, O. Grumberg, K. L. McMillan and X. Zhao, "Efficient Generation of Counterexamples and Witness in Symbolic Model Checking," in Proceedings of Design Automation Conference, pp.427-432, 1995.

[3] E. A. Emerson, Temporal and modal logic, in the Handbook of Theoretical Computer Science : Formal Models and Semantics, J. van Leeuwen, editor, Elsevier, pp.995-1072, 1990.

[4] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith, " Counterexample-Guided Abstraction Refinement," in Proceedings of Computer Aided Verification, pp.154-169, 2000.

[5] P. E. Ammann, P. E. Black and W. Majurski, "Using Model Checking to Generate Tests from Specifications," in Proceedings of ICFEM ' 98, pp.46-54, 1998.

[6] 권기현, "모델 검증을 이용한 게임 풀이", 정보과학회 학회지, 제21권 제1호, pp.7-14, 2003.

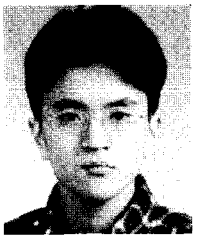
[7] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith, "Progress on the State Explosion Problem in Model Checking," in Proceedings of 10 Years Dagstuhl, LNCS 2000, pp.154-169, 2000.

[8] K. L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.

[9] Y. Lu, Automatic Abstraction in Model Checking, Ph.D. thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, 2000.

[10] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, "NuSMV 2 : An OpenSource Tool for Symbolic Model Checking," In Proceedings of CAV' 02, 2002.

[11] T. Lee, G. Kwon, "Relay Model checking for Avoiding The State Explosion Problem," In Proceedings of SERA' 2004, pp.305-310, 2004.



이 태 훈

2003년 경기대학교 전자계산학과(학사)
2003년 ~ 현재 경기대학교 전자계산학과 석사과정
관심분야 : 모델 체킹, 소프트웨어 모델링, 소프트웨어 공학

권 기 현

1985년 경기대학교 전자계산학과(학사)
1987년 중앙대학교 전자계산학과(이학 석사)
1991년 중앙대학교 전자계산학과(공학 박사)
1998년 ~ 1999년 독일 드레스덴 대학 전자계산학과 방문교수
1999년 ~ 2000년 미국 카네기 멜론 대학 전자계산학과 방문교수
1991년 ~ 현재 경기대학교 정보과학부 교수
관심분야 : 소프트웨어 모델링, 소프트웨어 분석, 정형 기법 등