모델 체킹을 이용한 네모라이즈 게임 풀이

이정림* · 권기현**

1. 서 론

본 논문에서는 게임 풀이 기법을 소개한다. 모델 체킹을 이용한 게임 풀이 방법을 소개하기 위해서 네모라이즈를 예제 게임으로 선정했다. 네모라이즈는 핸드폰에 탑재되어 이용되고 있으며, 아래 그림에서 보듯이 그래프의 한 붓 그리기 게임처럼 플레이어가 시작 지점을 출발해서 이동가능한 모든 지점을 한번만 거쳐서 목적 지점에 도달되는 경로 찾기 게임이다. 이미 방문한 지점은 다시 방문할 수 없으며 모든 지점이 방문되어야 하기 때문에, 만약 지점의 수가 n개 이면 풀이 경로의 길이는 n-1이다. 예를 들어 아래 게임에서 지점의 수는 13이기 때문에 풀이 경로 <32, 42, 43, 44, 34, 24, 14, 13, 23, 22, 12, 11>의 길이는 12이다.

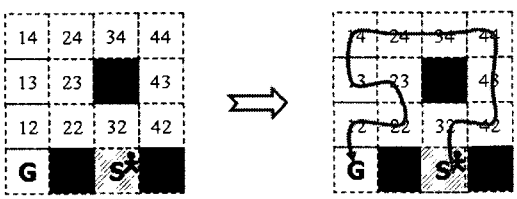

그림 1. 네모라이즈 게임

네모라이즈 풀이 경로는 모델 체킹으로 찾을 수 있다. 모델 체킹은 유한 상태 모델과 논리식을 받아서, 모델과 논리식간의 만족성 관계를 결정한다[1]. 모델이 논리식을 만족하는 경우 모델 체킹은 참을 출력하지만, 모델이 논리식을 만족하지 않는 경우에 모델 체킹은 거짓과 함께 그 이유를 담은 반례(counterexample)를 출력한다. 예를 들어 그림 2에 있는 모델이 '도달 가능한 모든 상태에서 항상 p'의 의미를 갖는 논리식 AG p를 만족하는지를 살펴보자. 만약 도달 가능한 상태 중에서 $\neg p$인 상태가 하나라도 존재한다면 이 논리식의 값은 거짓이다. 그림을 살펴보면 모델에는 5개의 상태가 있는데, 나머지 모든 상태에서는 p를 만족하지만 5번 상태에서만 p가 만족되지 않는다. 따라서 논리식 AG p의 값은 거짓이며, 초기 상태에서 $\neg p$인 상태로 도달되는 경로 1 2 3 4 5를 반례로 출력한다. 이 경로를 살펴보면 논리식이 왜 만족되지 않는지 그 이유를 쉽게 알 수 있다. AG p가 EF $\neg p$의 쌍대(dual)이기 때문에, AG p

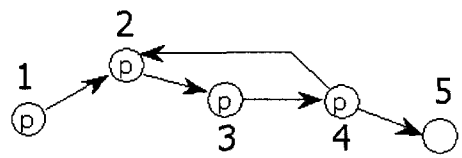

그림 2. 반례 생성 예

* 경기대학교 전자계산학과 석사과정 중
** 경기대학교 정보과학부 교수
※ 본 연구는 한국과학재단 특정기초연구(R01-2005-000-11120-0) 지원으로 수행되었음.

의 반례는 EF ¬p의 증거(witness)이다[2]. 여기서 논리식 EF ¬p의 의미는 '언젠가는 ¬p인 상태로 도달 가능하다'이다.

네모라이즈 게임의 유한 상태 모델이 주어졌다고 가정하자. 유한 상태들 중에서 게임의 종료를 나타내는 목표 상태를 *goal*이라 하자. 게임을 풀 수 있는 목표 상태로의 도달 가능 여부를 시제 논리식으로 나타내면

EF *goal*

이다. 주어진 게임 모델에 대해서 위의 논리식을 체크함으로서 목표 상태로의 도달 가능 여부를 판정할 수 있다. 만약 도달가능하다면, 모델 체킹은 참을 출력한다. 한편, 반례를 이용하여 목표 상태로 도달되는 경로를 얻기 위해서 위의 식을 다음과 같이 전체 부정한다.

!EF *goal* ≡ AG !*goal*

여기서 !는 부정을 나타내는 기호이다. EF *goal*의 부정은 AG !*goal*로서 "어떠한 시도를 해도 목표 상태에 도달할 수 없음"을 나타낸다. 만약 EF *goal*이 참이면, AG !*goal*은 반드시 거짓이다. 왜냐하면 두 식은 쌍대 관계이기 때문이다. 만약 목표 상태로 도달되는 경로가 하나 이상 존재한다면 AG !*goal*의 모델 체킹 결과는 거짓이다. 이때 모델 체킹 도구에 의해서 생성되는 반례가 게임의 풀이 경로이다. 본 논문에서는 이러한 원리를 이용하여 네모라이즈 게임을 풀이한다.

논문의 구성은 다음과 같다. 2장에서는 모델 체킹에 관한 기본 지식을 소개한 후, 모델 체킹을 이용해서 크기가 작은 게임을 푸는 방법을 소개한다. 그러나 게임의 크기가 큰 경우에는 상태 폭발 문제가 발생되는데, 이를 해결하기 위한 추상화 기법을 3장에서 제시하며 여러 가지 모델 체킹 도구를 사용하여 크기가 큰 게임을 풀었던 경험과 결과를 소개한다. 마지막으로 4장에서 결론 및 향후 연구 과제를 기술한다.

2. 배경지식

2.1 모델

모델 체킹에 사용되는 모델의 핵심 요소는 상태와 상태간의 전이이며, 이들을 사용하여 시스템의 행위를 모델링 한다. 특히 CTL(Computation Tree Logic, [3]) 모델 체킹에서는 크립키 구조라 불리는 모델 $M=(S, I, R, L)$을 사용한다. 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 속성

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 CTL은 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.

8개의 연산자 중에서 본 논문에서 사용하는 주요 연산자는 AG이며 이것의 쌍대는 EF이다. 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, EFϕ의 정형적 의미는 다음과 같다:

$M,s\models AG\phi$ iff $\forall\pi=s_1,s_2,\cdots \bullet \forall i\geq 1 \bullet M,s_i\models\phi$

$M,s\models EF\phi$ iff $\exists\pi=s_1,s_2,\cdots \bullet \exists i\geq 1 \bullet M,s_i\models\phi$

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 구한 후, 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. 상태집합 $[\![\phi]\!]$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. AGϕ, EFϕ의 상태집합 계산에 사용된 역 방향 탐색 함수는 다음과 같다.

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

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

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 상태 집합 $[\![\phi]\!]$를 구한후 모델 체킹 알고리즘의 핵심은 초기 상태 집합 I가 $[\![\phi]\!]$의 부분집합인지를 검사하는 것이다. 즉 $M\models\phi$ iff $I\subseteq[\![\phi]\!]$이다.

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 ¬ϕ의 쌍대는 EF ϕ이기 때문에, AG¬ϕ의 반례는 EF ϕ의 증거이다. 그러므로 EF ϕ의 증거로서 AG ¬ϕ의 반례를 설명할 수 있다. 정의한 대로 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 $\tau^1(\phi) \subseteq ... \subseteq \tau^n(\phi) \subseteq ...$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 $S_1=\tau^1(\phi), S_2=\tau^2(\phi), ..., S_n=\tau^n(\phi)$라고 하자. 사실, $S_1=\tau^1(\phi)=[\![\phi]\!]$이다. 왜냐하면 $pre_{\exists}(\phi)=\phi$이기 때문이다. 그림에서 보듯이 $S_n \cap I \neq \varnothing$이면 AG¬ϕ는 거짓이다.

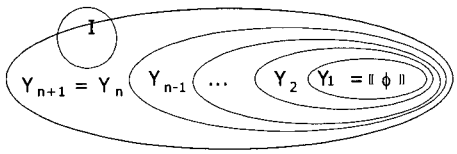

그림 3. $S_n \cap I \neq \varnothing$ 이면 AG$\neg\phi$는 거짓이다.

거짓인 경우라면, 그림 4에서 보는 것처럼 두 단계를 거쳐서 반례를 생성한다. 첫 번째 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($S_n \cap I \neq \varnothing$ 일 때 종료한다). 여기서 $post_{\exists}(Q) = \{s' \in S \mid \exists_{s \in Q} \cdot (s,s') \in R$는 Q로부터 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 $\langle s_1,\cdots,s_n\rangle$를 찾는다.

$$s_n \in S_n \cap [\![\phi]\!]$$
$$s_{i-1} \in pre_{\exists}(S_i) \cap S_{i-1}$$

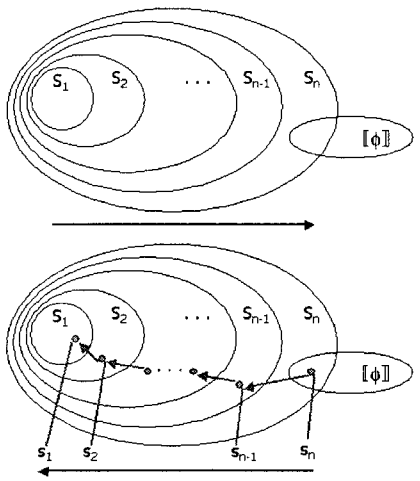

그림 4. 정방향과 역방향 탐색을 통해서 반례 생성

2.5 게임 풀이

네모라이즈와 같은 도달성 게임은 대부분 그래프 구조로 표현된다. 우리는 그래프 구조의 일종인 유한 상태 모델 M으로 게임을 나타냈다 (게임에 대한 유한 상태 모델은 [4]번을 참고). 유한 상태들 중에서 게임의 종료를 나타내는 목표 상태를 *goal*이라 하자. 게임을 풀 수 있는 목표 상태로의 도달 가능 여부를 시제 논리식으로 나타내면 EF *goal* 이다. 주어진 게임 모델에 대해서 위의 논리식을 체크함으로서 목표 상태로의 도달 가능 여부를 판정할 수 있다. 만약 도달가능하다면, 모델 체킹은 참을 출력한다. 한편, 반례를 이용하여 목표 상태로 도달되는 경로를 얻기 위해서 위의 식을 다음과 같이 전체 부정한다.

!EF *goal* ≡ AG !*goal*

여기서 !는 부정을 나타내는 기호이다. EF *goal*의 부정은 AG !*goal*로서 "어떠한 시도를 해도 목표 상태에 도달할 수 없음"을 나타낸다. 만약 EF *goal*이 참이면, AG !*goal*은 반드시 거짓이다. 왜냐하면 두 식은 쌍대 관계이기 때문이다. 만약 목표 상태로 도달되는 경로가 하나 이상 존재한다면 AG !*goal*의 모델 체킹 결과는 거짓이다. 이때 모델 체킹 도구에 의해서 생성되는 반례가 게임의 풀이 경로이다.

공개 모델 체킹 도구인 SMV(Symbolic Model Verifier, [5])을 사용하여 그림 1의 네모라이즈 1판을 풀이한 결과가 아래에 있다. 이 경로를 살펴보면, 플레이어는 시작 지점에서 출발해서 12개 지점을 모두 한 번씩 통과한 후에 목표 지점에 도달한다. SMV로 게임을 풀기 위해서, [4]번에서 유한 상태로 기술된 게임 모델을 SMV의 입력 언어로 변환했으며 반례를 얻기 위한 목적으로 속성은 AG !*goal*로 명세했다.

MoveToUp	1	0	1	1	0	0	0	0	0	0	0	0	-	
move	up	right	up	up	left	left	left	down	right	down	left	down	-	
s	31	32	42	43	44	34	24	14	13	23	22	12	11	

그림 5. 모델 체킹 도구 SMV로 구한 게임 풀이 경로

우리는 게임을 풀기 위해서 SMV 뿐만 아니라 SPIN[6], Esterel[7] 등 세 가지 모델 체킹 도구를 사용했다. 사용된 모델 체킹 도구의 간단한 설명과 함께 이들을 네모라이즈 1판 풀이에 적용한 결과가 아래 표에 있다. 다행히도 게임이 복잡하지 않아서 모든 모델 체킹 도구가 1초 안에 풀이 경로를 찾아냈다.

3. 상태 공간 축소를 위한 추상화

3.1 추상화 개념

게임의 크기가 작은 경우 모델 체킹으로 도달 경로를 찾아냈지만, 게임의 크기가 큰 경우 상태 폭발 문제(state explosion problem)가 발생해서 도달 경로 찾기에 요구되는 시간이 크게 증가하거나 또는 불가능한 경우가 있다[8]. 예를 들어 아래 그림과 같이 네모라이즈 10판의 경우 고려할 수 있는 상태 공간이 2^{65}(=36,893,488,147,419,100,000) 으로 모델 체킹 도구의 현재 능력을 초과한다. 그 결과 위에서 소개한 세 가지 모델 체커로 게임을 풀 수 없었다. 따라서 추상화를 통한 상태 공간의 축소가 필요하다.

모델 체킹을 수행하기 위해서는 주어진 유한

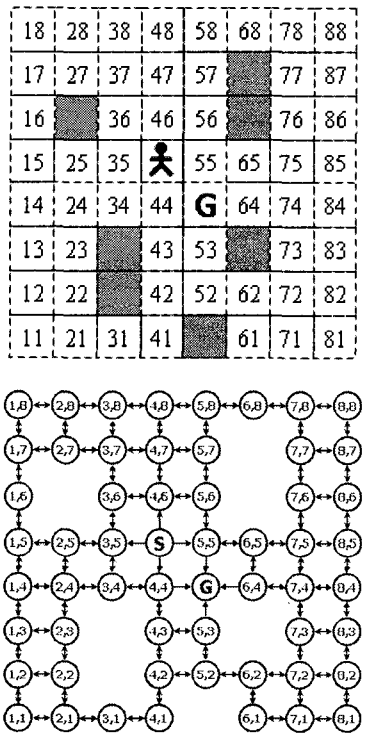

그림 6. 추상화 없이 풀 수 없었던 네모라이즈 10판과 플레이어의 이동을 나타내는 그래프

상태 모델의 상태 공간을 전부 조사해야 한다. 만일 모델의 크기가 작다면, 모델의 상태 공간 전체

표 1. 여러 모델 체킹 도구로 네모라이즈 1판의 풀이 결과

	모델 체킹 도구	SMV	SPIN	Esterel
설명	탐색 방법	너비우선탐색	깊이우선탐색(DFS) 너비우선탐색(BFS)	너비우선탐색
	탐색 방향	역방향	정방향	역방향
	상태 공간 표현	묵시적	명시적	묵시적
결과	시간(Sec)	00:01	00:01	00:01
	메모리(MB)	1.1	2.6(DFS)/2.3(BFS)	11.2

를 조사할 수 있다. 그러나 모델의 크기가 커서 상태 공간을 전부 조사하는 것이 불가능한 경우, 어쩔 수 없이 추상화(abstraction)를 통해서 모델의 크기를 축소해야 한다[9]. 그러나 모델을 추상화할 때에는 세심한 주의가 요구된다. 즉, 축소된 모델에서 얻어낸 도달 경로 정보가 원본 모델에서도 보존되도록 추상화해야 한다.

원본 모델에 대해서 모델 체킹을 수행하는 것이 아니라, 추상화를 통해서 얻어낸 축소된 모델에 대해서 모델 체킹을 수행하기 때문에 그 결과가 참 또는 거짓일 수 있다. 축소된 모델에서 얻어낸 도달 경로와 원본 모델의 도달 경로와의 관계는 아래 그림에서 보듯이 4가지가 있다. 첫째, 축소 모델에서 도달 가능하면, 원본 모델에서도 도달 가능하다(true positive). 둘째, 축소 모델에서 도달 불가능이면, 원본에서도 도달 불가능이다(true negative). 셋째, 축소 모델에서는 도달 가능하지만, 원본에서는 도달 불가능이다(false positive). 넷째, 축소 모델에서는 도달 불가능하지만, 원본에서는 도달 가능하다(false negative).

원본 모델의 크기가 커서 모델 체킹으로 도달 경로를 구할 수 없는 경우 추상화를 통해서 모델의 크기를 축소해야 한다. 이 경우 추상화는 false positive를 허용해서는 안 된다. 다시 말해서, 축소된 모델에서 도달 경로가 존재한다면 원본 모델에서도 반드시 존재해야 한다. 그래야지만 원본 모델을 조사하지 않고도 축소된 모델을 조사하는 것이 의미 있다. 만약 그렇지 못해서 false positive를 허용한다면, 축소 모델에서 도달 경로가 존재한다고 해도 원본 모델에서는 도달 경로가 존재하지 않기 때문에 축소된 모델에서의 작업이 무의미하다.

3.2 모델 축소

방금 전에 살펴본 대로, 네모라이즈와 같은 도달성 게임 플이에 사용되는 추상화는 false positive를 허용하지 않아야 한다. 즉, 축소된 모델에서 도달 경로가 존재한다면 원본 모델에서도 존재해야 한다. 그래야지 축소 모델에서 작업하는 것이 의미가 있다. 만약 그렇지 못해서 false positive를 허용한다면, 축소 모델에서 도달 경로가 존재한다고 해도 원본 모델에서는 도달 경로가 존재하지 않기 때문에 축소 모델에서 작업하는 것이 무의미하다. 즉, 추상화가 전혀 도움되지 않는다.

게임 풀이에 요구되는 추상화를 얻기 위해서, 상태 폭발을 일으켰던 네모라이즈 10판을 분석해서 일반적인 추상화 규칙을 정의한다. 먼저 플레이어의 이동을 그림 6과 같이 양방향 그래프로 나타내보자. 게임을 여러 차례 수행한 경험에 의해서 차수가 2인 지점이 추상화 가능한 지점임을 발견했다. 왜냐하면 이 게임은 한 붓 그리기와 같이 지나온 지점을 재방문할 수 없기 때문에, 차수

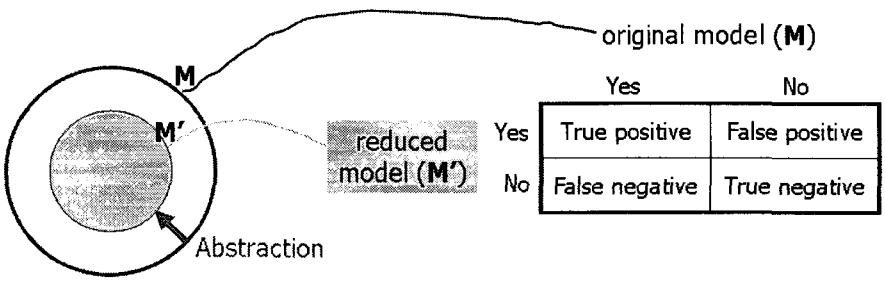

그림 7. 축소된 모델과 원본 모델에서 도달 경로의 존재 관계

가 2인 지점은 다른 곳으로 빠져나갈 수 없이 한 방향으로만 계속 지나가야 한다. 예를 들어 차수가 2인 지점 (1,6)을 살펴보자. 지점 (1,5)에서 위로 올라가면 반드시 지점 (1,7)에 도달한다. 반대 방향도 마찬가지이다. 그래서 지점 (1,6)을 제거할 수 있다. 지점 (1,8)의 경우도 마찬가지이다. 지점 (1,7)에서 위로 올라가면 반드시 지점 (2,8)에 도달해야 한다. 반대 방향도 마찬가지이다. 따라서 지점 (1,8)도 제거할 수 있다. 이것을 일반화한 것이 그림 8의 규칙 1이다.

지점 (1,7)에서 지점 (2,8)로 올라가는 경우를 살펴보자. 이미 지점 (1,7)과 지점 (2,8)이 추상화되었기 때문에 지점 (1,7)에서 위로 올라가면 반드시 지점 (2,7)에 도달한다. 따라서 지점 (2,8)을 축소할 수 있다. 이것을 일반화한 것이 규칙 2이다.

뿐만 아니라 지점 (1,5)에서 지점 (1,7) 까지 이미 추상화되었고, 지점 (1,7)에서 지점 (2,7)까지가 추상화되었기 때문에 지점 (1,5)에서 위로 올라가면 반드시 지점 (3,7)에 도달한다. 따라서 지점 (1,7)을 제거할 수 있다. 이것을 일반화한 것이 규칙 3이다.

이러한 규칙을 점진적으로 반복 적용하면 그림 9의 오른쪽과 같이 축소된 플레이의 이동 그래프를 얻는다. 규칙의 적용 순서에 무관하게 항상 동일한 결과 그래프를 얻는다. 그 결과 플레이어가 이동할 지점이 총 57개에서 30개로 거의 절반이 줄어들었다. 주의할 것은 네모라이즈 게임을 모델링할 때 각각의 지점이 부울 변수로 표현된다는

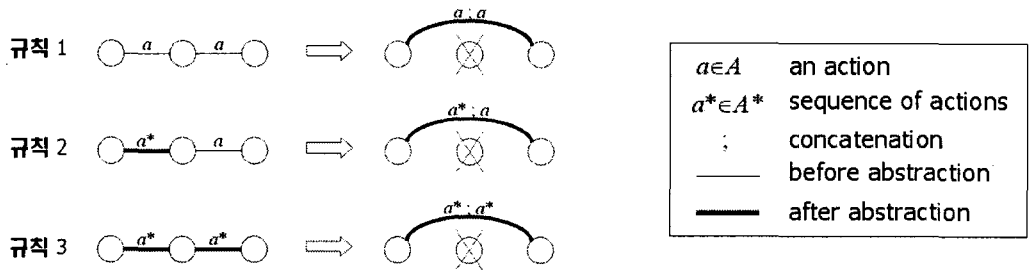

그림 8. 추상화 규칙과 범례

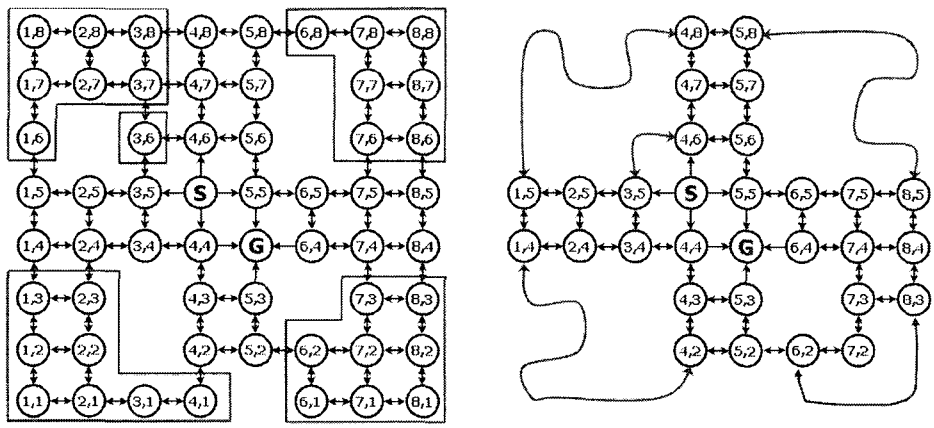

그림 9. 규칙을 반복 적용해서 얻어낸 축소된 플레이어의 이동 그래프

것이다. 왜냐하면 각각의 지점은 방문 또는 미 방문의 두 가지 상태를 갖기 때문이다. 또한 플레이어가 이동할 수 있는 지점의 모임은 열거형으로 나타낸다. 매 단계에서 플레이어가 선택할 수 있는 움직임 수가 4이기 때문에 축소된 이동 그래프를 활용한다면, 모델 체킹에 사용되는 모델의 크기를

$$2^{30} \times 2^{5} \times 2^{2} = 2^{37}$$

로 축소할 수 있다.

3.3 추상화 결과

아래 표 2에서 보듯이 추상화를 수행하기 이전에는 게임을 풀기 위한 도달 경로를 찾지 못했었다. 왜냐하면 상태 폭발이 발생되었기 때문이다. 표에서 기호 ∞은 3시간 이상 경과해도 결과를 볼 수 없었음을 나타낸다. 그러나 앞에서 제안된 도달 경로를 위한 추상화 기법을 적용한 이후에는 3개의 모델 체킹 도구로 게임을 푸는데 모두 성공했다(실험환경 : CPU 2.6Mhz, 메모리 1GB). 특히 우리의 예상과는 다르게 SPIN을 사용했을 때 단지 1초 내에 풀이 경로를 찾아내어서 성적이 가장 좋았다. 사실 실험을 하기 이전에 우리는 SMV의 성적이 제일 좋을 것이라고 미리 기대했었다. 그러나 결과는 예상과는 다르게 SPIN(DFS, 깊이 우선 탐색 옵션을 사용한 경우)이 제일 우수했다.

표 2. 네모라이즈 10판의 추상화이전과 이후의 수행 결과

네모라이즈 10판	추상화 이전		추상화 이후	
	시간 (Sec)	메모리 (MB)	시간 (Sec)	메모리 (MB)
Esterel	∞	∞	*04:59*	*4.00*
SMV	∞	∞	00:59	133.1
SPIN(DFS)	∞	∞	00:01<	4.9

표 3에서도 보듯이 모든 경우에 있어서 SPIN(DFS)을 이용했을 때 가장 적은 시간이 소요되었다.

표 3. 네모라이즈 1~10판의 수행 결과

레벨	Esterel	SMV	SPIN (BFS)	SPIN (DFS)
1판	00:01<	00:01<	00:01<	00:01<
2판	00:01<	00:01<	00:01<	00:01<
3판	00:46	00:04	00:01<	00:01<
4판	00:27	00:01	00:01<	00:01<
5판	06:05	00:25	00:04	00:01<
6판	07:46	00:18	00:03	00:03
7판	15:43	00:30	00:04	00:03
8판	∞	∞	∞	∞
9판	39:08	00:26	00:11	00:02
10판	04:59	00:38	00:03	00:01<

우리는 이전 선행 연구를 통해서 네모라이즈와 비슷한 푸쉬 푸쉬 게임을 모델 체킹으로 풀었었다[10,11]. 기존 연구와 본 연구의 큰 차이점은 모델 체킹 도구 활용에 있다. 기존 연구에서는 SMV만이 사용되었다. 그러나 본 연구에서는 SMV 이외에도 SPIN과 Esterel을 사용했다. 세 가지 모델 체커들 모두 산업계에서 널리 사용되기 때문에 도구에 대한 경험을 얻을 수 있을 뿐만 아니라 세 가지 모델 체킹 도구를 다양하게 비교해 볼 수 있는 좋은 기회였다. 위에서 언급한바와 같이 네모라이즈 게임을 푸는데 있어서 예상을 깨고 SPIN의 기록이 가장 우수했다. 알려져 있는 바와 같이 SPIN은 상태를 직접 표현하는 명시적 모델 체킹 도구로서 SMV에 비해서 성능이 저하된 것으로 알려졌다. 그래서 푸쉬 푸쉬 게임을 풀 때 SPIN의 사용 결과가 좋지 못해서 제외를 했던 경험이 있다. 그러나 네모라이즈 게임은 푸쉬 푸쉬

에 비해서 반례의 길이가 적다. 대략 반례의 길이가 40~60 정도 되는 경우에는 오히려 SPIN의 결과가 좋은 경우가 많다. 반례의 길이에 따라서 어느 모델 체킹 도구가 우수한지를 밝히는 것은 매우 좋은 연구 주제라 생각한다.

4. 결 론

핸드폰에 탑재되는 네모라이즈는 게임 규칙을 준수하면서 초기 상태에서 목적 상태로 가는 경로를 찾는 게임이다. 게임의 크기가 작은 경우 모델 체킹으로 상태 공간 전체를 조사해서 최단 풀이 경로를 찾아냈다. 그러나 게임의 크기가 큰 경우 상태 폭발 문제로 인해서 상태 공간 전체를 조사할 수 없었다. 그래서 본 논문에서는 false positive를 인정하지 않는 추상화 방법을 사용했다. 그 결과 축소된 모델에서 구한 도달 경로는 원래 모델에서도 그대로 유효했다. 즉, 도달 경로 입장에서 볼 때, 축소된 모델을 조사하고서도 마치 원본 모델을 조사한 것과 같은 결과를 얻을 수 있다.

그러나 제안된 추상화 규칙만으로는 해결할 수 없는 단계들이 아직 많이 남아있기 때문에 더 정교한 추상화 규칙이 요구된다. 표 3에서 8판의 경우, 제안된 추상화 규칙으로 세 도구 모두에서 경로를 풀어내지 못했다. 8판을 추상화하면, 플레이어가 이동할 지점이 총 56개에서 35개로 줄어든다. 또한 플레이어가 이동할 수 있는 지점의 모임을 열거형으로 나타내고 플레이어가 움직일 수 있는 수를 조합하면 모델의 크기는

$$2^{35} \times 2^{6} \times 2^{2} = 2^{43}$$

으로 축소된다. 이는 추상화 된 10판의 경우보다

$$2^{43} / 2^{37} = 2^{6}$$

2^{6}배 정도 크다. 10판보다 도달해야하는 지점이 5개 더 많을 뿐이지만 탐색해야 하는 모델의 크기는 지수적으로 더 크기 때문에 경로를 찾는 것이 매우 어려워진다.

따라서 본 논문에서는 차수 2인 지점에 대해서 집중적으로 다루었는데, 향후에는 2보다 큰 차수를 갖는 지점에 대해서 추상화 규칙을 연구하고자 한다. 또한 게임을 대상으로 얻어낸 추상화와 도달성 분석 결과를 소프트웨어 검증에 활용하는 것도 빠른 시간 내에 연구해야 할 과제이다.

참 고 문 헌

[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 Counter examples and Witness in Symbolic Model Checking," In Proceedings of Design Automation Conference, pp.427-432, 1995.

[3] R. Bloem, K. Ravi, and F. Somenzi, "Symbolic Guided Search for CTL Model Checking," in Proceedings of Design Automation Conference, pp.29-34, 2000.

[4] 이정림, 권기현, "모델 추상화를 이용한 네모라이즈 게임 풀이," 정보과학회 학술발표논문집, 제32권 제1(B)호, pp.367-369, 2005.

[5] http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/

[6] http://spinroot.com/spin/whatispin.html#A

[7] http://www-sop.inria.fr/esterel.org/Html/About/AboutEsterel.htm

[8] 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

[9] Y. Lu, "Automatic Abstraction in Model Checking," Ph.D. thesis, Carnegie Mellon

University, Department of Electrical and Computer Engineering, 2000.
[10] G. Kwon, "Applying Model Checking Techniques to Push Push Game Solving," In Proceedings of SERA2003, LNCS 3026, pp. 290-303, 2003.
[11] 권기현, "모델 체킹에서 상태 투영을 이용한 모델의 추상화," 정보처리학회 논문지, 한국정보처리학회, 제11-D권 제6호, pp.1295-1300, 2004.

이 정 림

• 2004년 경기대학교 정보과학부 졸업(학사)
• 2004년~현재 경기대학교 전자계산학과 석사과정 중
• 관심분야>: 정형 기법, 임베디드 소프트웨어, 소프트웨어 모델링

권 기 현

• 1985년 경기대학교 전자계산학과(학사)
• 1987년 중앙대학교 전자계산학과(이학석사)
• 1991년 중앙대학교 전자계산학과(공학박사)
• 1998년~1999년 독일 드레스덴 대학 전자계산학과 방문교수
• 1999년~2000년 미국 카네기 멜론 대학 전자계산학과 방문교수
• 1991~현재 경기대학교 정보과학부 교수
• 관심분야 : 소프트웨어 모델링, 소프트웨어 분석, 정형 기법등