17-3-R01

속성 분할을 이용한 릴레이 모델 체킹

(Relay Model Checking Based on Partitioning Properties)

이 태 훈, 권 기 현
경기대학교 정보과학부
{taehoon, khkwon}@kyonggi.ac.kr

요 약

모델 체킹은 유한 상태 모델과 시제 논리식을 받아서 모델과 논리식과의 만족성 관계를 결정한다. 만족성 관계를 판정하기 위해서 모델이 갖는 상태 공간을 모두 검사하기 때문에, 모델의 크기가 커질수록 고려해야 하는 상태의 수도 지수적으로 증가한다. 이를 상태 폭발 문제라고 한다. 이를 해결하기 위한 방법으로 추상화, 반순서 등의 여러 가지 방법이 제안되었다. 대부분의 방법들은 모델의 구조를 이용하여 상태 공간을 축소하고 있다. 이와는 달리, 본 논문에서는 속성 분할을 이용한 릴레이 모델 체킹을 제안하고, 이를 통해서 상태 폭발 문제를 해결한다. 제안된 릴레이 모델 체킹의 유용성을 확인하기 위해서 푸쉬 푸쉬 및 소코반 게임 풀이에 적용해 본 결과, 추상화 기법으로 풀지 못했던 푸쉬 푸쉬 50판과 A* 알고리즘으로 풀지 못했던 소코반 51판등을 제안된 릴레이 모델 체킹으로 풀 수 있었다.

1. 서론*

모델 체킹은 유한 상태 모델 M과 시제 논리식 ϕ를 받아서 만족성 관계 $M \models \phi$를 결정한다[1]. 모델이 갖는 모든 상태 공간을 철저하게 탐색하면서 만족성 여부를 결정하기 때문에, 테스팅이나 시뮬레이션으로는 찾을 수 없는 난해한 버그를 찾을 수 있다. 그리고 사용자의 개입 없이 검증 과정이 자동으로 진행되기 때문에, 정리 증명에 비해서 빠르다. 또한 모델이 논리식을 만족하지 않는 경우, 그 이유를 담은 반례를 제공함으로서 모델의 디버그 작업을 돕는다. 이와같은 이점으로 인해서 모델 체킹은 지난 20년 동안 하드웨어 검증, 소프트웨어 검증, 프로토콜 검증에 활발히 사용되고 있다.

앞에서 설명한 바와 같이 모데 체킹의 이점 중의 하나가 반례 생성이다[2]. 예를 들어 시제 논리식 $\mathrm{AG}\neg\phi$ 가 거짓인 경우 ϕ가 참이 되는 상태로 도달되는 경로를 보여준다. 이러한 경로를 반례라고 부르며 반례를 통해서 $\mathrm{AG}\neg\phi$가 만족하지 않는 이유를 파악할 수 있다. 반례는 시스템 디버그 작업에 사용될 뿐만 아니라 자동 추상화 유도[3], 테스트 케이스 생성등 다양한 분야에 사용되고 있다.

본 논문에서는 반례를 이용하여 푸쉬 푸쉬, 소코반과 같은 박스 밀기 게임을 풀고자 한다. 모델 체킹을 이용해서 성공적으로 게임을 풀기 위

* 본 연구는 정보통신부(MIC)와 한국소프트웨어공학협회(KSEA)의 한·카네기멜론 S/W 전문인력 교육 국내보급 사업의 지원으로 수행되었음.

해서는 상태 폭발 문제를 잘 다스려야 한다[4]. 일반적으로 모델 체킹에서 고려해야 할 상태의 수는 게임의 크기에 따라 지수적으로 증가한다. 따라서 게임의 크기가 커질수록 검사해야 하는 상태의 수도 크게 증가하여 성공적인 게임 풀이가 어렵다. 이것을 상태 폭발 문제라고 한다. 상태 폭발 문제를 해결하기 위한 방법으로 추상화(Abstraction), 반순서를 이용한 축소(Partial order reduction), 귀납법(Induction), 합성 추론(Compositional reasoning) 등의 여러 가지 방법이 제안되었다[1]. 대부분의 이들 최적화 기술들은 모델의 축소만을 고려하고 있다. 어떤 경우에는 이런 기술들을 다 적용하여도 상태 폭발 문제 때문에 모델 체킹을 완료하지 못하는 경우도 있다. 따라서 다른 접근 방법에서의 최적화 기술이 필요하다. 본 논문에서는 속성 분할을 이용하여 상태 폭발 문제를 해결하고자 한다. 이를 위해 릴레이 모델 체킹을 제안하고, 그 유용성을 확인하기 위하여 푸쉬 푸쉬 및 소코반 게임 풀이에 적용한다. 그 결과, 추상화 기법으로 풀지 못했던 푸쉬 푸쉬 50판을 제안된 릴레이 모델 체킹으로 풀 수 있었다. 마찬가지로, 인공지능의 A* 알고리즘으로 풀지 못했던 소코반 51판도 제안된 릴레이 모델 체킹으로 풀 수 있었다.

기존에 논의된 상태폭발방지 기법들 대부분은 모델에 적용되는 기술이다. 따라서 중복 적용이 힘든 경우도 있고, 시스템에 대한 행위를 변경하는 경우는 원래 시스템에서 일어날 수 있는 행위와 동일한지를 검사하는 추가적인 계산이 필요하다. 따라서 본 논문에서는 상태 폭발방지의 관점을 모델에서 속성으로 변경하였다.

본 논문의 구성은 다음과 같다. 2장에서는 모델 체킹에 관한 기본적인 내용을 살펴본다. 3장에는 릴레이 모델 체킹에 대해서 설명한다. 4장에서는 적용 사례에 대해서 살펴보고 5장에서는 결론과 향후연구 과제에 대해서 살펴본다.

2. 배경지식

2.1 모델

모델 체킹에 사용되는 모델의 핵심 요소는 상태와 상태간의 전이이며, 이들을 사용하여 시스템의 행위를 모델링 한다. 본 논문에서 사용할 CTL 모델 체킹에서는 크립키 구조라 불리는 모델 $M = (S, I, R, L)$을 사용한다. 여기서,

- S는 상태들의 집합이다.
- $I \subseteq S$는 초기 상태들의 집합이다.
- $R \subseteq S \times S$은 상태들 간의 전이를 나타내는 관계이다.
- $L : S \rightarrow 2^X$은 각 상태에서 참이 되는 단순 명제들을 해당 상태에 배정한 함수이다. 여기서 X는 단순 명제들의 집합이다.

모델 체킹은 시스템이 갖는 무한 행위에 대해서 조사를 하기 때문에 상태들 간의 전이를 나타내는 R은 전체 관계라고 가정한다. 즉 $\forall s \in S \cdot \exists s' \in S \cdot (s, s') \in R$로서, 모든 상태마다 전이할 수 있는 다음 상태가 최소한 하나 이상 존재한다. 경로 $\pi = s_0 s_1 s_2 s_3 s_4 \cdots$는 전이 가능한 상태들을 차례대로 나열한 것으로서 $(s_i, s_{i+1}) \in R, i \geq 0$이며 길이는 무한이다.

2.2 속성

모델에 관한 속성은 모델을 트리의 관점에서 해석하는 CTL 시제 논리로 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 CTL 연산자 AX, EX, AF, EF, AG, EG, AU, EU를

얻는다.

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.

$$\top \equiv \neg\bot$$
$$\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$$
$$\phi_1 \Rightarrow \phi_2 \equiv \neg(\phi_1 \wedge \neg\phi_2)$$
$$\phi_1 \Leftrightarrow \phi_2 \equiv (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1)$$
$$\mathrm{AX}\phi \equiv \neg\mathrm{EX}\neg\phi$$
$$\mathrm{AF}\phi \equiv \neg\mathrm{EG}\neg\phi$$
$$\mathrm{AG}\phi \equiv \neg\mathrm{EF}\neg\phi$$

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

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

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

CTL의 의미를 살펴보자. 모델 M의 상태 s에서 CTL 식 ϕ가 참인 경우를 $M, s \models \phi$로 표시한다. 그렇지 않다면 $M, s \not\models \phi$로 표시한다. 상태 s에서 CTL 식 ϕ의 값은 다음과 같이 재귀적으로 정의된다.

$M, s \not\models \bot$

$M, s \models p$ iff $p \in L(s)$

$M, s \models \neg\phi$ 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 $\exists_{\pi = s0, s1, \cdots} \cdot \forall_{k \geq 0} \cdot M, s_k \models \phi$

$M, s \models \mathrm{E}(\phi_1 \mathrm{U}\, \phi_2)$

iff $\exists_{\pi = s0, s1, \cdots} \cdot (\exists_{k \geq 0} \cdot M, s_k \models \phi_2 \wedge \forall_{0 \leq i < k} \cdot M, s_i \models \phi_1)$

2.3 만족성 검사 알고리즘

모델 M의 모든 초기 상태에서 CTL 식 ϕ가 참인 경우를 $M \models \phi$로 표시하며 "M이 ϕ를 만족한다"라고 읽는다. 모델 체킹은 M과 ϕ를 받아서 "M이 ϕ를 만족하는지를 결정"하는 문제이다. 이를 위해서 ϕ를 만족하는 상태들의 집합을 구한 후, 이 상태 집합에 초기 상태가 포함되어 있는지를 검사한다. CTL 식 ϕ를 만족하는 상태들의 집합을 $[\![\phi]\!]$ 라고 표시하자. 모델 체킹 알고리즘의 핵심은 상태 집합 $[\![\phi]\!]$를 구한 후 초기 상태 집합이 $[\![\phi]\!]$의 부분 집합인 것을 검사하는 것이다. 즉,

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

이다. 만족성을 검사하기 위해서는 모델이 갖는 상태 공간을 탐색해야 하는데, 여기에는 2가지 탐색 방법이 있다. 초기 상태에서 출발해서 목표 상태를 찾아나가는 정 방향 탐색과 목표 상태에서 출발해서 거꾸로 초기 상태를 찾아가는 역 방향 탐색 방법이 있다. 다음과 같이

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

함수 $pre_{\exists}$는 집합 Q로 도달할 수 있는 이전 상태들의 집합을 역방향으로 구하며, $post_{\exists}$는 현재 상태 Q로 부터 도달 가능한 다음 상태들의 집합을 정방향으로 찾는다.

3. 릴레이 모델 체킹

속성 $AG\neg\phi$ 에서 ϕ가 아래와 같이 여러 명제의 연접(conjunction)으로 연결된 경우를 살펴보자.

$$\phi = \{\phi_1 \wedge \ldots \wedge \phi_n\}$$

이 경우, 기존 모델 체킹 기법에서는 모든 속성을 만족하는 상태를 찾으려고 한다. 하지만 실제 시스템이 속성을 만족하는 순서를 살펴보게 되면 $\phi_1, \ldots, \phi_n$까지 한번에 모두 만족이 되는 것이 아니라 하나하나 순서를 가지고 차례대로 만족되어 간다. 기존 모델 체커에서는 모든 명제를 한번에 만족하는 상태를 찾기 위해 불필요한 모든 부분을 탐색을 수행한다. 따라서 모델검사 기법의 가장 큰 문제인 상태폭발문제가 발생할 수 있다. 이런 상태 폭발 문제를 해결하기 위해 여러 기법들이 연구되어 왔다. 하지만 대부분의 기법이 모델에 대해 적용이 될 수 있는 기술이다. 본 장에서는 상태 폭발 문제를 해결할 수 있는 방법 중의 하나로서 속성 분할을 이용한 릴레이 모델검사 기법을 제안한다.

만일 기존에 주어진 속성을 만족하는지 모델 체커에서 검사를 수행하지 못했다면, 다른 접근 방법으로 속성을 분할하는 방법으로 접근을 할 수 있다. 속성을 분할하기 위한 방법은 속성을 만족하기 위해서 명제가 만족되는 첫 번째 속성을 모델 체커에 입력을 하게 되면 모델 체커는 첫 번째 속성이 만족하는데 까지 갈 수 있는 경로를 보여주게 된다. 이 경로를 바탕으로 상태를 재구성한 후 다시 두 번째 속성이 만족하는 상태까지의 경로를 모델 체커를 이용해서 얻을 수 있다. 이러한 방식으로 분할된 속성을 이용하게 된다면 원래의 속성보다 좀더 적은 메모리에 빠른 시간 안에 결과를 얻을 수 있다.

릴레이 모델 체킹 기법은 시스템에 대한 반례를 얻기 위해 사용될 수 있다. 실제 릴레이 모델 체킹 기법을 이용해서 어떤 시스템에 대한 반례를 얻었다면 그 생성된 반례는 실제 시스템에서도 발생할 수 있는 도달 경로이다. 왜냐하면 시스템에 대한 행위는 변경된 사항이 없기 때문에 실제 시스템에서도 도달 가능한 상태이다. 하지만 반대로 릴레이 모델 체킹 기법으로 찾지를 못했다고 한다고 해도 시스템에는 다른 도달 가능한 경로가 있을 수 있다. 따라서 시스템에 다른 도달 경로가 있는지를 다시 검사해야 한다. $\phi = \{\phi_1 \wedge \ldots \wedge \phi_n\}$일때 $\phi_1, \ldots, \phi_n$ 에서 어떤 하나의 ϕ_x를 선택할지는 해당 도메인에 따라서 다르다. 이것은 도메인에 대한 경험으로서 지정을 해주어야만 한다. 일반적으로 어떤 ϕ_x가 한번 만족하게 된다면 그 이후 계속 안정적인 상태로 변하지 않는다면 우선 그 ϕ_x를 먼저 선택하여 검사를 수행하는 것이 좋다.

이제 릴레이 모델 체킹을 정형적으로 표현해 보자. 전 장에서 언급했듯이 모델 체킹은 주어진 모델 M이 속성 ϕ를 만족하는지를 검사한다. 만약 $M \not\models \phi$ 이라면, 모델 체커는 반례 π를 리턴한다. 따라서 생성된 반례 π는 속성 ϕ를 만족하는 상태로 안내하는 경로가 된다. 즉,

$$MC(M, \phi) = \{\pi \mid \exists_{\pi} \forall_{s \in Img(\pi, I)} \cdot M, s \models \phi\}$$

여기서 함수 Img는 반례와 상태를 받아서 주어진 상태로부터 반례에 의해 도달되는 최종 상태를 리턴한다. 반례는 시퀀스 타입이기 때문에 Img 함수는 다음과 같이 정의된다.

$$Img(\langle\rangle, S) = S'$$
$$Img(\langle \alpha \rangle \hat{} \beta, S) = Img(\beta, Img(\langle \alpha \rangle, S))$$

만약 $M \not\models \phi$ 인 경우, 릴레이 모델 체킹은 다음과 같이 정의할 수 있다.

$$MC(M_1, \phi_1) \hat{} \ldots \hat{} MC(M_n, \phi_n)$$

즉, 모델 체킹이 차례 대로 n번 진행되며, 각 단계 마다 생성된 반례를 모두 결합하면 (여기서 기호 ^는 반례의 결합을 나타냄) 속성 ϕ를 부정하는 반례가 된다. 릴레이 모델 체킹에서는

$$M_1 = M$$

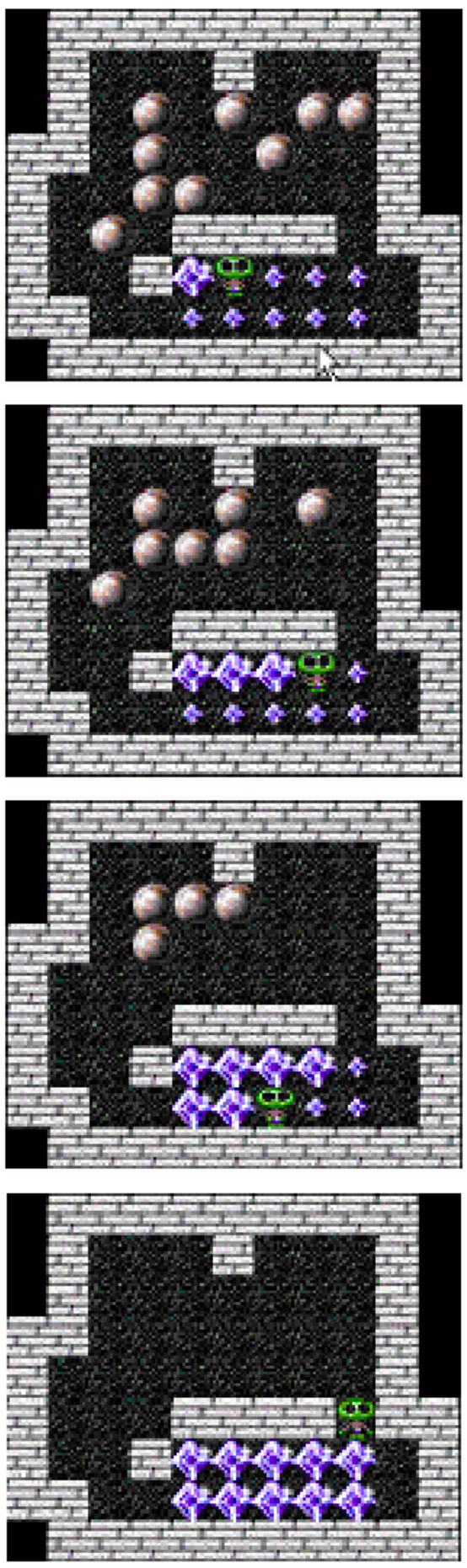
와 같이 주어진 원래 모델로 모델 체킹을 시작한다. 릴레이 경주에서 이전 주자의 바톤을 계승하듯이, i 번째 모델은 i-1 번째 모델 검사로부터 아래와 같이 얻는다.

$$I_i = Img(MC(M_{i-1}, \phi_{i-1}), I_{i-1})$$

4. 적용 사례

모델 체킹 기법에서 시스템에 대한 모델과 시스템이 만족해야하는 속성을 입력하면 참 혹은 반례를 생성한다. 이 반례를 이용해서 게임 풀이에 적용 할 수 있다. 게임을 규칙을 모델로 변경하고 게임의 목표를 속성으로 표현하고 게임 의 초기상태를 모델의 초기상태로 변경을 하게 되면, 모델검사기는 해당되는 게임에서 필요한 움직임을 반례를 이용해서 알려주게 된다. 이 점을 이용해서 푸쉬푸쉬 게임과 소코반 게임에 대해서 모델링을 했다. 핸드폰에 들어가 있는 푸쉬푸쉬 게임의 마지막 판인 50판을 이 릴레이모델검사 기법을 이용해서 검사를 수행하였고 XSokoban 에 51번째판을 역시 동일한 방법으로 수행을 해보았다.

푸쉬푸쉬 50판의 경우 총 4단계로 나누어서 수행을 하였다. 진행순서를 (그림 1)에 나와 있다. 맨 처음 (그림 1)의 맨 위 사진에서 볼 수 있듯이 하나의 공을 집어넣었고 그리고 2개를 더 집어넣었고 그다음에 3개를 더 집어넣었고 그다음에 모든 공을 넣도록 SMV프로그램을 작성했다. 각각의 경우 수행하는데 걸린 시간과 메모리는 (표 1)에서 볼 수 있다. (표 1)에 나와 있는 시간과 메모리는 NuSMV[5]에서 수행한 결과를 보여준다. NuSMV에서는 푸쉬푸쉬 50판의 정보를 한번에 해결할 수 없었지만 나누어서 풀었을 경우 결과를 볼 수 있었다. 하지만 자체적으로 수정한 NuSMV에서 동작을 시킬 경우엔 한번에 해결을 할 수 있었고 릴레이 모델 체킹 기법보다 좀더 짧은 해결책을 보여주었다. 릴레이 모델 체킹 기법을 이용했을 경우엔 총 338의 길이를 가지는 반례를 생성하였지만 한번에 수행을 했을 경우엔 22 번 만에 똑같은 결과를 보여줄 수 있었다. 이와 같이 릴레이 모델 체킹 기술을 적용할 경우 좀더 작은 메모리로 수행을 할 수는 있었지만 최단 경로를 얻을 수는 없었다.



(그림 1) 푸쉬푸쉬 50번째판 진행순서

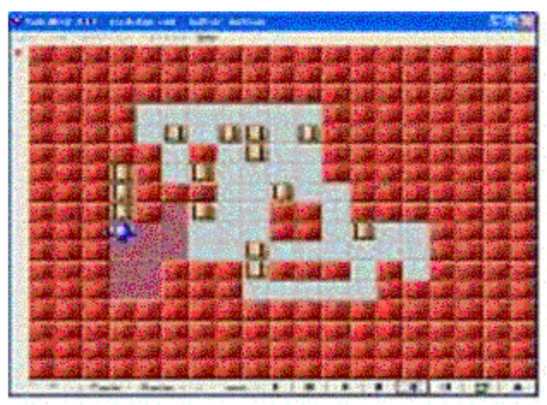
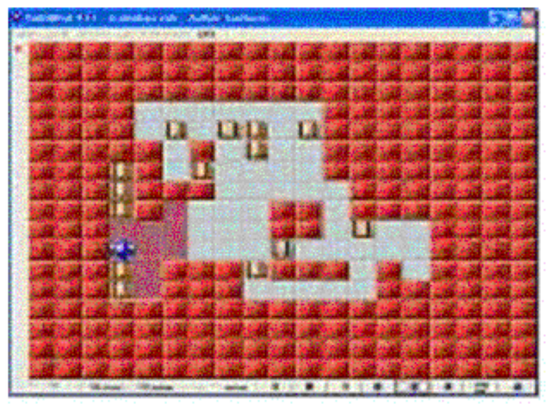
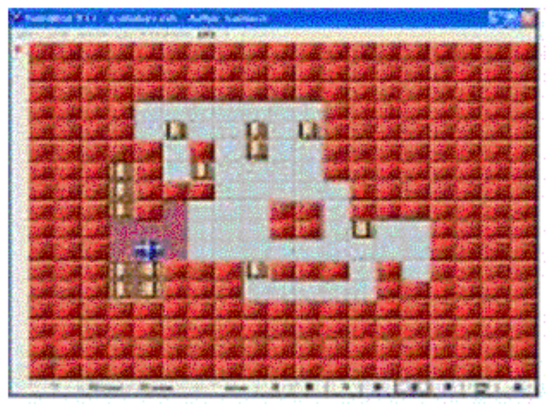
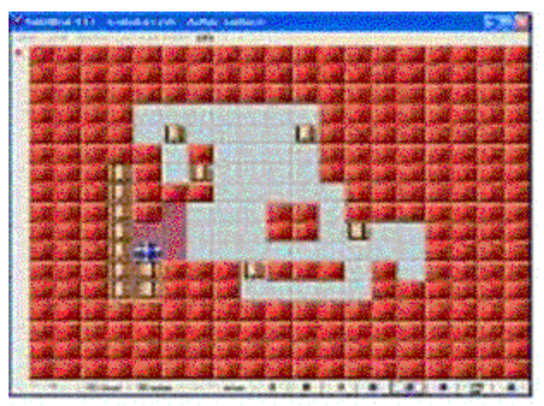
(표 1) 푸쉬푸쉬 50번째판 수행결과

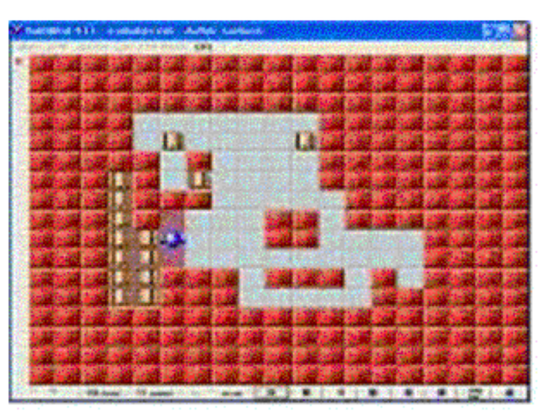
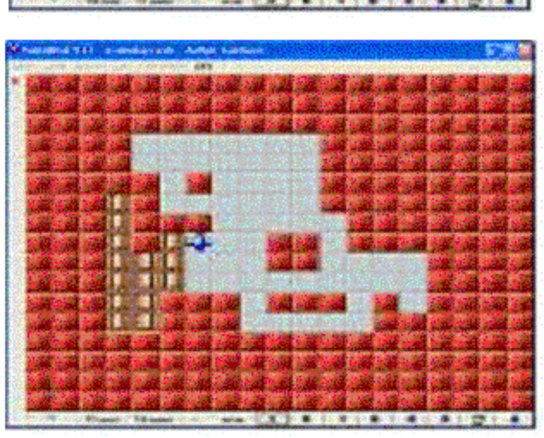
나눈횟수	시간(초)	메모리(Kb)	BDD
1	102.090	20,909	360,190
2	644.350	69,413	152,7837
3	1,422.400	132,849	287,4495
4	89.050	20,217	440,698

(표 2)는 두 번째 사례로서 XSokoban의 51번째판 에 대한 적용 결과를 보여준다. XSokoban의 경우 크기가 큰 모델이 많이 있었고, 변형된 A* 알고리즘을 이용해서 해결하려는 노력이 많이 있었다. 51판의 경우 다른 모델 체킹 기술에 적용한 상태폭발 방지 기술을 이용해서 해결을 할 수가 없었다. 하지만 릴레이 모델 체킹 기술을 이용해서 해결을 할 수 있었다.

(표 2) XSokoban 51번째판 수행결과

나눈횟수	시간(초)	메모리(Kb)	BDD
1	7.940	13,460	249,864
2	7.220	13,312	266,136
3	8.990	13,272	327,765
4	7.680	13,264	320,587
5	360.250	109,620	2,410,499
6	8.180	13,628	251,258
7	70.710	28,696	627,580
8	26.510	14,332	382,639
9	14.660	13,485	297,256

(그림 2) XSokoban51번째판 수행순서

XSokoban의 51번째판은 총 9단계로 나누어서 진행을 했다. 각각의 순서는 (그림 2)에 나와 있다.

모델 체킹을 진행하는 목적이 반례를 생성하는 것이라면, 사례 연구에서 보는 바와 같이 이전 여러 상태 폭발 방지 기술을 적용한 모델에서 상태 폭발이 발생했을 경우에 릴레이 모델 체킹 기법은 이전 에는 상태 폭발 문제 때문에 진행할 수 없었던 모델 체킹의 결과를 볼 수 있게 해주었다.

5. 결론 및 향후 연구 과제

푸쉬푸쉬와 XSokoban에 대한 적용사례에서 보았듯이 좀더 적은 메모리에 적은 시간에 수행이 가능하였고 기존 방법으로는 해결할 수 없었던 문제에 대해서도 해결을 할 수가 있었다. 따라서 속성을 분할하는 것은 상태 폭발 문제에 대한 최적화 기술 중에 하나가 될 수 있다. 모델의 행위에 대한 변경을 하지 않기 때문에 결과가 나오면 그 결과는 바로 실제 시스템에서 동작하는 결과로 신뢰할 수 있게 되고, 실제로 푸쉬푸쉬와 소코반문제를 해결하기 위한 중요한 최적화 기술 중에 하나로 사용되고 있다. 하지만 아직까지 모든 문제에 대해서 적용이 가능하지 않고 사람이 직접 손으로 코드를 수정해가면서 나누어야 했다. 그리고 나누는 기준에 대해서는 해당 도메인에 대한 전문적인 지식을 가지고 있어야 한다. 또한 최단 경로를 찾는데 는 사용될 수가 없다.

따라서 속성 선택의 자동화에 대한 연구와 릴레이 모델 체킹 기술의 자동화에 대한 연구가 필요하게 된다. 그리고 릴레이 모델 체킹 기법으로 결과가 나왔을 경우에 이 수행된 정보를 이용해서 최단 경로를 찾을 수 있는 방법에 대한 연구도 필요하다.

참고문헌

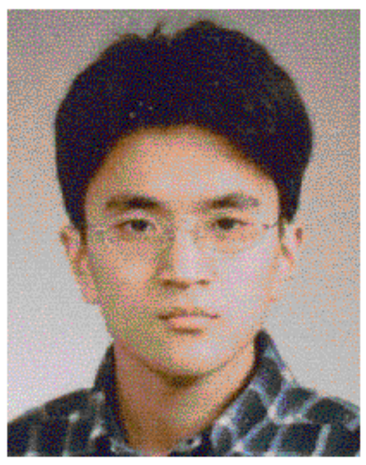
[1] E. M. Clarke, O. Grumberg and D. Peled, Model Checking, MIT Press, 1999.

[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. 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.

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

[5] 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.



이 태 훈

1997년~2003년 경기대학교 전자계산학과(학사)

2003년~현재 경기대학교 전자계산학과 석사과정

관심분야 : 모델 검사, 소프트웨어 모델링, 정형기법

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 : 프트웨어 모델링, 소프트웨어 분석, 정형 기법 등