Ubiquitous Application을 위한 소프트웨어 검증 기법

°선태봉[1], +인호[1], Hong Lu[2]¶, Steve Liu[2]¶
고려대학교 컴퓨터학과[1], Texas A&M University[2]
{jiyo°, hoh_in}@korea.ac.kr, {luhong, jcliu}@cs.tamu.edu

A Software Verification Technique for Ubiquitous Applications

Tae Bong Sun°, Hoh Peter In, Hong Lu[2]¶ Steve Liu[2]¶
Dept. of Computer Science, Korea University[1], Texas A&M University[2]

요 약

시스템의 안정성을 검증하는 Model Checking(모델 검증)은 여러 가지 분야에서 기본적으로 중요하게 사용되어왔다. 하지만 Model Checking의 특성상 Computing Power가 많이 필요한 점과 최근 소프트웨어의 대형화, Ubiquitous 환경에서의 잦은 변화가 필요한 점 등으로 Model Checking를 그대로 적용하기 어렵다. 따라서 이 논문은 점증적인 변화에 따른 필요한 부분만 검증하는 점증적 모델 검증을 소개한다.

1. 서 론

Model Checking(모델 검증)은 특정 시스템의 안정성이나 설계상 오류가 없음을 검증하는 것으로 시스템의 흐름이 설계한대로 올바르게 작동하는지 논리적으로 검사한다.

그러나 Model Checking은 시스템의 모든 가능성을 논리적으로 따져나가기 때문에 Computing Power가 많이 요구되는 것이 일반적이다. 게다가 최근 Ubiquitous Computing 환경에서 잦은 변화를 거치는 시스템이 많고 큰 시스템의 경우 작은 시스템부터 시작하여 조금씩 늘려나가는 것도 많다. 이런 변화가 있을 시에 Model Checking은 새롭게 Model을 만들고 검증을 시작하게 되기 때문에 기존의 Model Checking은 비효율적이다.

효율적인 Model Checking을 위해서 Incremental Model Checking(점증적 모델 검증)이 등장하였다. Oleg와 Scott[1]은 μ-calculus logic을 사용하여 가장 먼저 Incremental Model Checking을 보였다. 이 방법은 Model의 변화를 입력으로 받아들여 Model의 변화가 기존의 Model의 안정성에 부합하는지 검사하는 것이다. Swamy[2]는 Computation Tree Logic(CTL)을 사용하여 Model의 변화와 검사를 위한 특정 조건의 변화를 모두 다룰 수 있게 하였다.

+ : Corresponding Author; °: The person who will present

이 논문에서는 Ubiquitous Computing에서 저전력, 대용량 검증을 위하여 Graph를 기반으로 하는 Incremental Model Checking을 제안한다. 간략하게 설명하자면 우선 Model Checking을 수행한다. 그 후, 변화에 따른 검증 시 변화 전의 검증 결과를 사용한다. 이 때, 변화에 따라 영향을 받는 부분을 Color Scheme을 사용해 구분하고 이 부분만을 검증하여 기존의 Model Checking보다 계산을 줄인다.

2장에서는 Model Checking의 일반적인 방법에 대해 간략하게 소개를 한다. 3장에서는 우리의 Incremental Model Checking를 소개한다. 4장에서는 Ubiquitous 환경 적용할 때, 기존 Model Checking의 난점과 우리의 Incremental Model Checking의 장점을 비교해 본다.

2. Model Checking

Model checking에 대한 일반적인 방법을 소개한다. 앞서 설명한 것처럼 우선 시스템을 Model화 하고 검증하고자 하는 조건을 특정 Formula로 표현하여 적용시켜 검증한다. 일반적으로 Model은 kripke structure로 표현하고 입력이 되는 Formula는 CTL(Computation Tree Logic)을 사용한다. 자세한 내용은 참고문헌인 "Model Checking"[3]에 나와있다.

2.1 Kripke Structure

Kripke Structure는 4개의 tuple로 정의된다.

M = (S, A, T, L)

이 중에 참고문헌(2장 Modeling Systems)과 다른 것이 하나 있다. 보통은 Set of Initial States 가 있지만 생략이 가능하다. 그러므로 'A'로 나타나는 AP(Atomic Proposition)를 사용하여 Model을 표현하기로 한다.

2.2 CTL(Computation Tree Logic)

Model Checking에서 특정 조건을 기술하기 위한 Formula로는 CTL이 쓰인다. CTL에는 5가지 Time Operator인 'X', 'F', 'G', 'U', 'R' 이 있고 Path Operator인 'A', 'E' 가 있다. 이 들 operator들은 시간적인 개념을 지니면서 만족하는 state를 set으로 나타낼 수 있기 때문에 Model Checking에 적합하다. 각 operator에 대한 자세한 설명은 위에 언급한 참고문헌 "Model Checking" [3]의 3장에 나와 있다.

CTL의 가능한 Formula는 Time Operator와 Path Operator가 조합되어 10개가 존재한다. 그리고 'NOT' 'OR' 'AND' 등의 기본적인 Operator도 쓰인다. 그 중에 위 2.1 kripke structure에서 설명한 AP이기도 한 5개의 Formula는 '$\sim f$', '$f \vee f$', 'EX(f)', 'EG(f)', 'E(f U f)' 이다. 이 5개의 조합으로 모든 Formula를 표현할 수 있는데, 이에 대한 자세한 내용은 참고문헌 "Model Checking" [3]에 나와있다.

CTL의 Formula들이 다섯 개의 Atomic Proposition들의 조합으로 표현이 가능하게 됨으로써 Model Checking 과정에서 이 다섯 개에 대해서만 검증 과정을 고려하면 된다.

2.3 Model Checking

2.2에서 보인 CTL 형식의 Formula가 2.1에서 보인 kripke structure로 Model화한 시스템이 특정 조건을 만족하는가 검증하는 입력이 된다. 이에 따라 Model 중에 Formula에 따른 특정 state들을 찾아낼 수 있게 된다.

3. Our Approach: Coloring Scheme IMC

3.1 IMC(Incremental Model Checking)의 개념

Incremental Model Checking은 Model의 변화에 따라 생기는 기존 Model Checking의 검증 과정이 변화와 상관이 없는 부분이 있어 비효율성 때문에 등장했다. 때문에 새로운 방법에서는 Model이 변화된 부분에 따라 검증이 필요한 부분과 필요 없는 부분을 구분하는 것이 필요하다. 이 논문에서는 state를 검증이 필요한 것과 그렇지 않은 것들을 구분할 수 있게 색깔로 표시하여 집합을 구성하는 방법을 소개한다.

Model 변화에는 Transition의 추가와 삭제, state의 추가와 삭제, state의 property 추가와 삭제가 있다. 이 중에 간단한 예제로 보일 것은 Transition의 추가이고, 해당 Formula는 'EG(f)'이다. 5개의 기본 Formula 중에서 'NOT' 'OR' 'EX(f)'는 state의 property를 보고 간단히 검증이 간단하므로 'EG(f)', 'E(f U f)'에 대해서만 설명할 필요가 있다. 두 개의 Formula중에 'EG(f)'가 보다 간단하므로 이를 예제로 들겠다.

3.2 EG(f) 의 state 구분

'EG(f)'의 경우는 이 Formula를 만족하는 path상의 모든 path가 'f'를 만족해야 하므로 'f'를 만족하면서 서로 SCC(Strongly Connected Components) state들을 Red로 Labeling한다. 'EG(f)'가 SCC를 만족해야 하는 것에 대한 자세한 설명과 증명은 "Model Checking" [3]에 나와있다. 이들 Red state(이하 R로 표기)들은 지금 현재 Model에서 'EG(f)'을 만족하고 있는 state들이다.

그 다음 RED로 Transition에 의해 갈 수 있는 state들은 Blue로 Labeling한다. 즉, 다음 state가 Red state이거나 Blue state인 경우의 state가 이에 속한다. 이들 Blue state(이하 B로 표기)들은 현재 'EG(f)'를 만족하지는 못하지만 Model의 변화에 따라 'EG(f)'를 만족할 수 있는 가능성이 있는 state이다.

그 다음 'f'를 만족하지만 RED, BLUE로 갈 수 없는 state는 Green으로 Labeling한다. 이들 Green state(이하 G로 표기)들 역시 'EG(f)'를 만족할 가능성이 있는 state이지만 추가되어야 할 Transition의 방향이 다르다.

마지막으로 'f'를 만족하지 못하는 state를 Yellow state(이후 Y로 표기)로 Labeling한다.

이에 따라 Labeling한 예제는 그림 1에 보이고 있다.

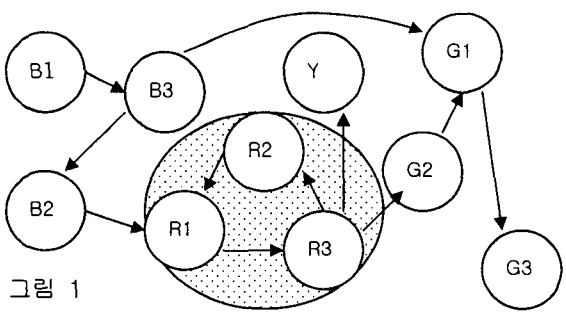

그림 1

3.3 EG(*f*)에서의 Transition 추가

4가지로 구분한 state간의 조합가능한 Transition의 개수는 16개이다. 추가되는 Transition에 의해 state의 Color가 변하기도 하고 그렇지 않기도 하다. 이를 표1에 정리해보았다. Transition은 'Src. -> Dest. 형식으로 나타낸다.

추가된 Transition	Color 변화	예
R -> R	No	R1 -> R2
R -> B	Yes(R) / No	R1->B2 / R1->B3
R -> G	No	R3 -> G3
B -> R	No	B2 -> G3
B -> B	Yes(R) / No	B2->B1 / B1->B2
B -> G	No	B2 -> G3
G -> R	Yes(R / B)	G2->R3 / G3->R3
G -> B	Yes(R / B)	G1->B3 / G1->B1
G -> G	Yes / No	G3->G2 / G2->G3
Y -> *	No	
* -> Y	No	

표 1

첫 번째 열에서는 추가되는 Transition을 설명하고 있고, 두 번째 열에서는 추가되는 Transition에 의해 Source state의 Color가 변하는지 그렇지 않은지 여부가 기술되어 있다. 세 번째 열에서는 「그림 1」의 Model에서 Transition 종류에 따른 예를 보이고 있다. 두 번째 열의 'Yes / No' 같은 경우, 예 역시 각각 따로 보여 주고 있다. G->R Transition의 경우 source인 G의 Color가 변하는데 R로 변할 것인가 B로 변할 것인가에 따라 예도 따로 보이고 있다.

표에서처럼 source가 변하는 경우가 있다면 그 경우는 검증이 필요하지만 그렇지 않다면 검증이 필요없다. 또한 source가 변한 경우 변한 state와 관계된 다른 state도 영향을 받아 변할 수 있으므로 더 이상 변화가 없을 때까지 점증적으로 검증을 하게 된다.

위와 같이 Color Scheme을 이용하여 검증이 필요한 부분과 그렇지 않은 부분을 구분한다. 이에 따라 Model이 변했을 경우 Model 전체를 검증하지 않고 변화한 부분에 관련된 곳, 그리고 변화의 가능성이 있는 경우에만 검증을 할 수 있게 된다.

4. Ubiquitous 환경에서의 Model Checking

Ubiquitous 환경이나 *Situation-Aware* 환경에서의 Model Checking을 생각해보면 서론에서 언급한 것처럼 기존 Model Checking은 시스템의 변화에 따라 매번 검증을 처음부터 Computing을 많이 하게 된다는 점에서 비효율적이다. 하지만 3장에서 살펴본 것처럼 Incremental Model Checking의 경우에는 정확한 계산량이나 수행시간을 정의할 수 없지만 기존의 것보다 변화에 대해서 효율적이다. 더구나 갈수록 작아지는 Computer size와 Embedded 환경을 생각해봤을 때, 보다 계산이 적은 Incremental Model Checking의 장점이 활용될 분야가 많을 것으로 생각된다.

예를 들면, 음식물의 상태를 체크하고 관리하는 냉장고에 내장된 Ubiquitous 시스템이 있다고 하자. 만약 주인이 냉장고에서 특정 음식물을 꺼내거나 넣었다고 하면 냉장고가 관리해야 할 음식류의 개수, 종류 등이 수시로 변하게 된다. 이 상황에서의 Model 변화는 Property나 State의 추가, 삭제로 생각할 수 있다. 그렇다면 이런 변화에 따른 Model Checking을 생각해보자. 하루에도 수십번씩 냉장고 안의 음식물의 종류나 개수 등이 바뀌는 상황에서 매번 Model Checking을 수행한다면, 작은 변화에도 Model Checking을 자주 하게 된다. 대신에 위에서 소개한 Incremental Model Checking을 이용하게 되면 전체적으로 Model Checking을 하지 않고 추가, 제거되는 음식물에 관련된 시스템의 일부만 Model Checking만 해도 된다.

이처럼 잦은 *Situation*의 변화는 기존 *Model Checking*을 비효율적으로 만들기 때문에 Incremental Model Checking은 Ubiquitous Computing 분야에서 중요한 소프트웨어 검증 기법(Software Verification Technique)이 될 것이다.

Acknowledgement

이 논문은 고려대학교 특별연구비에 의하여 수행되었음.

참고문헌

[1] Oleg V. Sokolsky and Scott. A. Smolka, *Incremental model checking in the modal mu-calculus.* Sixth Conference on Computer Aided Verification(CAV), pages 351-363, 1994.

[2] G. Swamy. *Incremental Methods for Formal Verification and Logic Synthesis.* PhD thesis, UC Berkeley, 1996.

[3] Edmund M. Clarke, Jr. and Orna Grumberg and Doron A. Peled, *Model Checking.* 1999.