SMT 해결기를 이용한 자바 메모리 모델 시뮬레이션

(Java Memory Model Simulation using SMT Solver)

이 태 훈 † 권 기 현 ††
(Taehoon Lee) (Gihwon Kwon)

요 약 많은 컴파일러는 속도를 높이기 위해서 최적화를 수행한다. 최적화의 결과로 프로그램의 구문이 변경된다. 단일 스레드 소프트웨어에서는 변경된 구문이 영향을 받지 않지만 멀티스레드 소프트웨어에서는 변경된 구문 때문에 예상하지 못한 실행 결과가 발생할 수 있다. 멀티 스레드 자바 소프트웨어는 스레드들 간에 메모리를 통한 상호작용을 자바 메모리 모델을 통해서 명세 한다. 자바 메모리 모델은 자바언어의 표준 메모리 모델이다. 하지만 현재까지 나와 있는 어떠한 자바 모델 체킹 도구도 자바 메모리 모델을 지원하지 않는다. 본 논문에서는 자바 메모리 모델을 지원하는 모델 체킹 도구를 개발하기 위해서 자바 메모리 모델을 지원하는 시뮬레이터를 많은 모델 체킹 도구에서 사용되는 SMT 기반으로 구현했다. SMT 기반 메모리 모델 시뮬레이터는 기존의 메모리 모델 시뮬레이터에서 몇 분이 걸리는 계산 결과를 1초 이내에 계산하였다. 또한 이를 통해 기존 소프트웨어 모델 체킹에서 표현할 수 없는 실행 결과를 빠르게 표현할 수 있다.

키워드 : 모델 체킹, 자바 메모리 모델, SMT 해결기

Abstract Recently developed compilers perform some optimizations in order to speed up the execution time of source program. These optimizations require the transformation of the sequence of program statements. This transformation does not give any problems in a single-threaded program. However, the transformation gives some significant errors in a multi-threaded program. State-of-the-art model checkers such as Java-Pathfinder do not consider the transformation resulted in the optimization step in a compiler since they just consider a single memory model. In this paper, we describe a new technique which is based on SMT solver. The Java Memory Model Simulator based on SMT Solver can compute all possible output of given multi-thread program within one second which, in contrast, Traditional Java Memory Model Simulator takes one minute.

Key words : Model checking, Java Memory Model, SMT Solver

· 본 연구는 경기도의 경기도지역협력연구센터사업의 일환으로 수행하였음 [2008-111-0,웹 기반 소프트웨어 모델 체킹 도구 개발]
· 이 논문은 2008 한국컴퓨터종합학술대회에서 'SMT를 이용한 자바 메모리 모델 시뮬레이션'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 경기대학교 정보과학부
taehoon@kgu.ac.kr
†† 종신회원 : 경기대학교 정보과학부 교수
khkwon@kgu.ac.kr

논문접수 : 2008년 8월 28일
심사완료 : 2008년 11월 16일

Copyright©2009 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제1호(2009.1)

1. 서 론

다중 스레드 프로그램은 시스템을 효율적으로 이용하기 위한 중요한 기술로 고려되어 왔다. 특히 최신의 하드웨어에서는 다중 프로세서를 장착하고 있다. 다중 프로세서를 가지고 있는 시스템에서 기존의 순차적인 프로그램을 동작시킬 경우 다중 프로세서 중 일부의 프로세서만을 사용할 수 있다. 따라서 시스템을 효율적으로 이용하기 위해서 다중 스레드와 같은 병행성 소프트웨어 개발 기술이 중요하다. 다중 스레드 프로그램의 중요성이 증가함에 따라서 다중 스레드 프로그램의 정확한 실행 의미를 정의하고 다중 스레드 프로그램의 검증을 수행하는 것도 중요하다.

현재 사용되는 대부분의 컴파일러는 최적화를 수행하기 위해 프로그램의 구문을 변경한다. 단일 스레드 소프트웨어 에서는 변경된 구문이 실행 결과에 영향을 미치지는 않지만 멀티 스레드 소프트웨어 에서는 변경된 구문 때문에 발생할 수 없는 실행 결과가 발생할 수 있다. 따라서 정확한 프로그램의 실행 의미를 정의하는 것이 중요하다.

자바 프로그램의 경우 멀티 스레드 자바 프로그램의 정확한 실행 의미인 자바 메모리 모델을 표준으로 정의했다[1]. 그리고 자바 메모리 모델의 동작을 설명하기 위해서 자바 메모리 모델 시뮬레이터는 주어진 멀티스레드 모델에 대해서 가능한 실행 결과를 계산한다[2]. 자바 메모리 모델 시뮬레이터는 실행 결과를 계산하기 위해서 모든 가능한 경우를 나열하고 하나씩 검사를 수행한다. 모든 경우를 하나씩 검사하기 때문에 검사할 스레드의 개수가 증가하면 가능한 경우의 수는 지수적으로 증가한다. 이 때문에 큰 모델에 대해서는 가능한 실행 결과를 생성하지 못했고 실제 자바 프로그램에 대해서는 적용할 수 없다. 또한 메모리 모델을 적용한 다양한 연구들이 존재한다. Tuan Quang Huynh 은 C# 메모리 모델을 정의하고 소프트웨어 검증을 수행하였다[3]. Fong Dong

는 Murphi라는 모델 체킹 도구를 이용하여 하드웨어 시스템의 메모리 모델에 대한 검증을 수행하였다[4]. 하지만 아직까지 자바 메모리 모델을 대상으로 분석을 시도한 것은 자바 메모리 모델이 유일하다.

SAT 해결기(SAT Solver)는 명제 논리식의 만족성 여부를 판단한다. 최근 SAT 해결기의 성능은 비약적으로 발전하였다. 이를 바탕으로 바운디드 모델 체킹과 같은 연구에서 SAT 해결기를 이용하여 하드웨어와 프로토콜과 같은 시스템의 검증을 수행하는 연구가 많이 진행 되었다[5]. 하지만 SAT 해결기에서는 명제논리만 다룰 수 있기 때문에 프로그램 언어의 다양한 특성을 다루기 힘들었다. SMT(Satisfiability Modulo Theories) 해결기는 이론들(Theories)이 추가된 일차 논리의 만족성 여부를 판단한다. SAT 해결기의 최적화된 기술을 바탕으로 다양한 이론들에 대한 만족성 여부를 판단하는 추가 해결기들로 다양한 형식의 논리식의 만족성 여부를 판단한다. 최근에 프로그램 언어의 다양한 특성을 지원하기 위해서 SAT 해결기 대신에 SMT 해결기를 모델 체킹에 적용하는 사례가 증가하고 있다[6]. 그 중 대표적인 연구는 SLAM 이다[7]. SLAM 은 마이크로소프트에서 개발 중인 모델 체킹 도구이다. 술어 추상화 기반 반례 개선 기법(Counter Example Guided Abstraction Refinement)을 이용하여 디바이스 드라이버를 검사한다[8]. 그 중에 추상화 수행 부분에서 SMT 를 이용하고 있다. 마이크로 소프트에서는 SLAM에서 사용될 SMT 해결기를 Z3라는 이름으로 개발하였다[9]. SLAM 이외에 CBMC[10] 등의 모델 체킹 도구가 개발되었고, 실제 동작하는 C 프로그램의 정형 검증에 사용되었다. 다양한 연구에서 SMT 기반 기술이 소프트웨어 검증에 효율적으로 사용되었다.

본 논문에서는 자바 메모리 모델 시뮬레이터의 성능을 향상시키기 위해서 최근 소프트웨어 검증 도구에서 많이 사용되고 있는 SMT 기반으로 자바 메모리 모델의 행위를 표현하였다. 이를 기존 작성된 자바 메모리 모델 시뮬레이터에 적용하여 기존에 비해 효율적인 결과를 얻을 수 있었다. SMT 기반 자바 메모리 모델 시뮬레이터를 이용하여 쉽게 자바 메모리 모델을 지원하는 모델 검증 도구로 확장할 수 있다. SMT 기반 메모리 모델 표현 기법은 향후 자바 메모리 모델을 지원하는 검증도구 개발을 위한 기본 이론이 된다.

본 논문의 구성은 다음과 같다. 2장에서는 자바 메모리 모델에 대해서 살펴본다. 3장에서는 SMT 기반 자바 메모리 모델 표현 방법에 대해 살펴보고, 4장에서는 사례연구를 살펴보고 5장에서 결론을 맺는다.

2. 배경 지식

초기값 x=0 , y=0	
Thread 1	Thread 2
1: r2=x;	3: r1=y;
2: y=1;	4: x=2;

그림 1 예제 프로그램

메모리 모델은 프로그램의 메모리 행위가 어떻게 동작하는지를 명세하여 프로그램이 어떻게 실행될지를 명세한다. 일반적으로 널리 사용되는 메모리 모델은 순차 일관성 메모리 모델(Sequential Consistency Memory Model)이다[11]. 순차 일관성 모델은 프로그래머가 이해하기 쉽고 쉽게 사용 가능하다. 하지만 순차 일관성 메모리 모델은 프로그램 최적화 때문에 발생하는 구문의 변경(transformation)를 표현하기에는 너무 제약이 강하다.

그림 1의 예제 프로그램의 실행 결과는 순차일관성메모리 모델에 의하면 {r1==0, r2==0}, {r1==1,r2==0}과 {r1==0,r2==2}인 결과만 가능하다. 하지만 Thread2의 구문이 실행될 때 구문의 최적화를 수행하기 위해서 컴파일러 혹은 가상기계 등에서 3,4의 실행 순서를 변경할 수 있고 구문 4가 실행된 다음에 구문 1이 실행되고 구문 1일 실행된 후에 구문 2가 실행되고 구문 2가 실행된 다음에 구문 3이 실행된다면 r1==1, r2==2인 결과가 발생할 수 있다. 이러한 최적화를 위한 구문의 변경을 기존의 순차 일관성 메모리 모델에서는 표현할 수 없다.

자바 메모리 모델은 주어진 프로그램과 주어진 실행결과(Execution)를 받아들여서 프로그램에서 실행결과가 허용되는지를 알려준다. 현재 자바 메모리 모델은 자바 표준으로 정의되었다[12]. 따라서 자바 프로그램의 모든 가능한 행위에 대해서 정형 검증을 수행하기 위해서는 주어진 프로그램의 모든 가능한 실행 결과를 생성하고 실행 결과에서 오류가 존재하는지 검사해야 한다. 하지만 아직까지 자바 메모리 모델의 규칙에 따라서 실행 결과를 계산해주는 효율적인 방법은 존재하지 않는다. 자바 메모리 모델 시뮬레이터에서는 주어진 다중 스레드 프로그램의 모델에 대해서 실행 결과를 계산해주는 기본적인 방법을 설명하고 있다.

자바메모리 모델은 동기화 되지 않은 소프트웨어와 실행 경로가 주어졌을 때 실행 경로가 소프트웨어에서 올바른 실행 경로인지 판단한다. 만일 잘 동기화된 소프트웨어의 경우 주어진 소프트웨어의 구문 순서대로 실행 하여도 된다. 하지만 동기화 되지 않은 소프트웨어의 경우 컴파일러 혹은 하드웨어 등에서 최적화를 수행할 수 있고 최적화를 수행하면 예상하지 못한 결과가 발생할 수 있다. 이런 모든 경우를 고려하면서 모든 가능한 모든 실행 결과를 표현하기 위해 자바에서는 자바 메모리 모델을 표준 메모리 모델로 제정하였다.

3. SMT 기반 변환 규칙

자바 메모리 모델은 수행 결과(Execution)를 이용해서 정의된다. 수행 결과 E는 다음과 같은 튜플로 구성된다.

$$<P, A, \xrightarrow{po}, \xrightarrow{so}, W, V, \xrightarrow{sw}, \xrightarrow{hb}>$$

- P는 프로그램을 의미한다.
- A는 행위 의 집합을 의미한다.
- $\xrightarrow{po}$ 는 프로그램 순서(Program Order) 을 의미한다.
- $\xrightarrow{so}$ 는 동기화 순서(Synchronization order)를 의미한다.
- W는 각각의 변수값 읽기 행위 r에 대해서 $W(r)$은 쓰기 행위를 되돌려준다.
- V는 각각의 쓰기 행위 w에 대해서 $V(w)$는 값을 읽어오는 읽기 행위를 되돌려준다.
- $\xrightarrow{sw}$ 는 synchronized-with 를 나타낸다.
- $\xrightarrow{hb}$ 는 happen-before 를 나타낸다.

실행 결과중에 잘 구성된 실행 결과는 다음과 같은 조건을 만족한다.

1. 변수 x의 각각의 읽기 행위는 x에 대한 쓰기 행위가 존재한다. 모든 읽기 행위 $\forall r \in A$에 대해서 $W(r) \in A$이다.
2. 동기화 순서는 프로그램 순서와 일관성을 가진다.
3. 실행결과는 스레드 내부 실행의미를 따른다.
4. 모든 읽기 행위 $r \in A$에 대해서 $r \xrightarrow{so} W(r)$인 경우는 존재하지 않는다. 또한 $W(r) \xrightarrow{so} W \xrightarrow{so} r$인 volatile 쓰기 행위 w는 존재하지 않는다.

잘 구성된 실행 결과는 수행된(committing) 행위들에 의해 유효하게 된다. 만일 A에 있는 모든 행위들이 수행되었다면 수행 결과는 자바 메모리 모델의 요구사항을 만족한다. 수행된 행위들의 집합 C는 공집합 C_0에서 시작해서 행위들의 집합 A에서 행위들을 가져와서 수행된 행위의 집합 C_i에서 새로운 수행된 행위의 집합 C_{i+1}을 생성한다. 정형적으로 표현하면 행위의 집합 $C_0, C_1, \ldots$이 있을 때, 첫 번째 수행된 행위의 집합은 공집합 $C_0 = \varnothing$이고 $C_i \subset C_{i+1}$이다. 그리고 주어진 행위들의 집합 $C_0, \ldots$과 실행 결과들의 집합 $E_1, \ldots$이 주어졌을 때 C_i에 있는 모든 행위는 E_i에 있는 행위 중에 하나여야 하고 동일한 사전 발생순서와 동기화 순서가 있어야 한다. V_i와 V는 C_i에 있는 행위에 대해서 동일해야한다. 그리고 읽기 행위 $r \in A_i - C_{i-1}$에 대해서 $W_i(r) \xrightarrow{hb_i} r$이어야 한다. 읽기 행위 $r \in C_i - C_{i-1}$에 대해서 $W_i(r) \in C_{i-1}$이어야 한다.

이와 같은 조건을 만족하는 수행된 행위들의 집합 C가 만들어 진다면 대응되는 실행 결과 E는 프로그램 P에서 허용되는 실행 결과이다.

최근 SAT 해결기는 수천만 개의 절(Clause)을 가지고 있는 식을 몇 초 이내에 만족하는 경우를 찾아준다. SMT 해결기는 SAT 해결기 정도는 아니지만 수만 개의 절을 가지고 있는 식에 대해서 쉽게 만족 불만족을 판단한다. 이런 빠른 속도 때문에 프로그램을 논리식으로 변경하고 SMT 해결기로 검사를 수행하명 기존 탐색 방법에 비해 효율적인 탐색을 수행 할 수 있다. 자바 메모리 모델의 잘 구성된 실행 결과를 기호적으로 표현하기 위해서 우리는 SMT를 이용한다. 기호적 표현은 2단계로 표현한다. 첫 번째 단계는 수행된 행위의 집합을 기호적으로 표현한다. 두 번째 단계에서는 잘 구성된 실행 결과가 되기 위한 조건을 기호적으로 표현한다.

자바 메모리 모델에서는 수행된 집합 C를 반복적으로 계산한다. 수행된 행위의 집합을 표현하기 위해 행위가 현재 수행된 집합 C에 속해있는지를 나타내는 변수를 도입한다. 모든 행위 $a \in A$에 대해서 이진 변수 a_i는 현재 i번째 단계에서 행위가 수행된 집합에 포함되는지를 나타낸다. 정형적으로 나타내면 아래와 같다.

$$a_i \Leftrightarrow a \in C_i \wedge a \notin C_{i-1}$$

이진변수 $a_{committed_i}$는 i번째 단계 혹은 그 이전에 수행된 행위의 집합에 속해 있는지를 나타낸다. 이를 정형적으로 나타내면 아래와 같다.

$$a_{committed_i} \Leftrightarrow a \in C_i$$

수행된 행위의 집합에 들어갈 행위는 다음과 같은 조건을 만족해야한다.

첫 번째로 이전에 실행되지 않은 행위가 선택되어야 한다.

$$a_i \Rightarrow \neg a_{committed_{i-1}}$$

두 번째로 현재 행위가 수행되었다면 수행된 행위들의 집합에 포함되어야한다.

$$a_i \Rightarrow a_{committed_i}$$

세 번째로 과거에 주어진 행위가 수행된 적 없고 현재 수행된 적 없다면 주어진 행위는 수행된 행위의 집합에 포함되면 안 된다.

$$(\neg a_{committed_{i-1}} \wedge \neg a_i) \Rightarrow \neg a_{committed_i}$$

네 번째로 이전에 수행된 집합에 포함되었다면 현재 수행된 집합에 포함된다.

$$a_{committed_{i-1}} \Rightarrow a_{committed_i}$$

다섯 번째로 모든 사전 수행 순서에 속한 행위 $j \xrightarrow{hb_i} k$ 에 대해서 j가 실행되지 않았다면 k도 실행되지 않는다.

$$\forall j,k \in A \cdot \neg j_{committed_i} \Rightarrow \neg k_i$$

수행된 구문의 집합을 구했다면 변수의 값을 쓰는 행위를 표현한다. 이진 변수 $R_{i,j,k}$는 i번째 단계에서 쓰기 행위 w_j에서 쓰인 값이 읽기 행위 r_k에 의해 읽어졌다면 참이 된다.

$$\forall i,j,k \cdot R_{i,j,n} \Rightarrow (Value(w_j)_i = Value(r_k)_i)$$

모든 i,j,k에 대해서 이진 변수 $R_{i,j,k}$가 참이라면 w_j에서 변수에 쓰는 값과 r_k에서 변수에서 읽어오는 값은 같은 값이어야 한다. 이를 통해서 변수의 값의 어떤 값을 가지게 되는지 표현할 수 있다.

위의 자바 메모리 모델의 제약 조건을 SMT 해결기에서 입력할 수 있는 논리식 형태로 변경하고 실행을 하게 되면 메모리 모델의 제약 조건을 만족한다고 나오고 만족하게 만드는 변수들의 값이 나온다. 이때 나온 변수들의 값을 부정한 식을 원래 식에 추가하면 이전에 나온 변수들의 값 이외의 다른 변수 값이 존재할 수 있는지 SMT 해결기가 검사를 한다. 이 과정을 불만족(Unsat)이라는 결과가 나올 때 까지 반복적으로 수행하면 모든 가능한 실행 결과를 계산할 수 있다.

4. 사례 연구

3장에서 설명한 변환 규칙을 바탕으로 자바 메모리 모델 시뮬레이터를 구현하였다. 아래의 표에서 기존 자바 메모리 모델 시뮬레이터와 SMT 기반 자바 메모리 모델 시뮬레이터의 수행 시간을 비교한다. SMT 해결기는 Yices[13]를 이용하였고 동일한 컴퓨터에서 검사를 수행하였다.

표 1 시뮬레이터의 성능 비교

실험 모델	기존 구현	SMT 기반 구현
cnflictingingWrites	38.3 초	0.21초
read_elimination2	0.01 초	0.01초
read_elimination3	149.2 초	0.52초

표 1에서는 총 3개의 프로그램에 대해서 기존 자바 메모리 모델 시뮬레이터와 SMT 기반 메모리 모델 시뮬레이터의 성능 비교를 수행한다. 첫 번째 예제는 자바 메모리 모델 시뮬레이터에 예제로 포함된 conflicting Writes이다. 예제의 프로그램은 그림 2와 같다.

conflictingWrites 에서는 총 2개의 스레드가 하나의 공유 자원에 대해서 접근을 한다. 이 모델에서는 공유 변수 p가 존재하고 지역변수 a, b가 있다. 자바 메모리

초기값 p=0	
Thread 1	Thread 2
1: p=1;	5: p=4;
2. p=2;	6. p=5;
3. p=3;	7. p=6;
4. a=p;	8. b=p;

그림 2 conflictingWrites 예제 모델

초기값 a=0, b=1	
Thread 1	Thread 2
1. r1=a;	
2. r2=a;	5. r3=b;
3. if(r1==r2)	6. a=r3;
4. b=2;	

그림 3 read_elimination2 예제 모델

모델 시뮬레이터에서는 38초의 시간이 걸려서 15 개의 실행 결과를 계산하였다. SMT 기반 자바 메모리 모델 시뮬레이터를 이용할 경우 동일한 실행결과를 0.2 초 정도의 시간 만에 결과를 얻을 수 있었다.

두 번째 예제는 자바 메모리 모델 논문에 있는 read_elimination 예제를 자바 메모리 모델 시뮬레이터의 문법에 맞게 구현하였다. 논문에 예제로 나온 프로그램은 총 2개의 스레드가 존재하고 하나의 공유 자원에 접근한다.

이 예제에서는 공유 변수 a와 b가 있고 첫 번째 스레드에서는 r1과 r2가 있고 두 번째 스레드에서는 지역변수로 r3가 있다. 이 예제를 기존 자바 메모리 모델 시뮬레이터로 실행 결과를 계산할 경우 0.01 초 이내에 결과를 얻는다. 하지만 만일 두 번째 스레드와 동일한 스레드를 추가하여 총 3개의 스레드가 실행될 경우 실행 결과를 계산하는데 걸리는 시간은 149초로 증가한다. 하지만 SMT 기반 자바 메모리 모델 시뮬레이터에서는 1초 이내의 시간에 모든 결과를 계산할 수 있었다. 이처럼 기존에 구현된 자바 메모리 모델 시뮬레이터에 비해 SMT 기반 자바 메모리 모델 시뮬레이터는 작은 시간 안에 동일한 실행 결과를 계산해준다.

5. 결 론

멀티 스레드 소프트웨어의 개발이 증가하면서 멀티 스레드 소프트웨어의 안전성이 계속 중요해진다. 최신의 컴파일러 시스템은 시스템의 속도를 증가하기 위해서 다양한 최적화 기술을 사용한다. 이런 최적화 기술 때문에 프로그램의 구문이 변경될 수 있다. 변경된 구문은 다중 스레드 프로그램에서 일반적인 방법으로는 불가능한 실행 결과가 발생한다. 이러한 실행 결과는 자바 메모리 모델을 통해서 설명가능하다. 하지만 기존의 대부분의 자바 기반 멀티 스레드 소프트웨어 검증 도구는

자바 기반 멀티 스레드 소프트웨어의 모든 행위를 검사하지 못했다. 이를 위해 본 논문에서는 멀티 스레드 소프트웨어의 모든 행위를 계산하는 시뮬레이터를 개발하였고 기존 개발된 도구가 몇 분의 시간이 필요한 것에 비해 약 1초정도에 동일한 결과를 얻을 수 있었다. 또한 최근 소프트웨어 검증 도구에서 많이 사용되는 SMT 기반 기법을 적용하여 쉽게 소프트웨어 검증 도구로 확장이 가능하도록 하였다.

하지만 실행 결과를 계산할 수 있는 프로그램은 실제 자바 소프트웨어가 아니라 멀티스레드 소프트웨어의 간략 화된 표현이다. 따라서 실제 멀티 스레드 자바 소프트웨어를 대상으로 적용하는 연구가 필요하다.

참 고 문 헌

[1] J. Gosling, B. Joy, G. Steele and G. Bracha, "The Java Language Specification Third Edition," Addison-Wesley, 2005.

[2] J. Manson and W. Pugh "The Java Memory Model Simulator," In Workshop on Formal Techniques for Java-like Programs, in association with ECOOP. June, 2002.

[3] T. Q. Huynh and A. Roychoudhury, "Memory model sensitive bytecode verification," Formal Methods In System Design, Volume 31, Number 3, pp. 271-305, October 18, 2007.

[4] F. Pong and M. Dubois, "Formal Automatic Verification of Cache Coherence in Multiprocessors with Relaxed Memory Models," IEEE Transactions on Parallel and Distributed System, Volume 11, Issue 9, pp. 989-1006, 2000.

[5] A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zhu, "Bounded Model Checking," Vol. 58 of Advances in Computers, 2003.

[6] A. Armando, J. Mantovani, and L. Platania, "Bounded Model Checking of Software using SMT Solvers instead of SAT Solvers," In the proceedings of the 13th SPIN workshop, Vienna(A). LNCS 3925, Springer Verlag. 2006.

[7] T. Ball, R. Majumdar, T. Millstein, and S.K. Rajamani, "Automatic Predicate Abstraction of C Programs," In the Procedings of PLDI, SIGPLAN Notices 36(5), pp. 203-213, 2001.

[8] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," In Proc. Computer-Aided Verification 2000, Lecture Notes in Computer Science. Springer, 2000.

[9] L. Moura and N. Bjorner, "Z3: An Efficient SMT Solver," Conference on Tools and Algorithms for the Construction and Analysis of Systems(TACAS), Budapest, Hungary, 2008.

[10] E.M. Clarke, D. Kroening, and F. Lerda, "A Tool for Checking ANSI-C Programs," Lecture Notes in Computer Science 2988, pp. 168-176, 2004.

[11] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," IEEE Transactions on Computers 9, 29, 690-691, 1979.

[12] J. Manson, W. Pugh, S.V. Adve, "The Java Memory Model," In the Proceedings of the POPL 2005, pp. 378-391, 2005.

[13] B. Dutertre and L. Moura, "The YICES SMT Solver," Computer Science Laboratory, SRI International, 2006. http://yices.csl.sri.com.