SPIN을 이용한 객체 지향형 소프트웨어의 정형검증 방법 비교 및 분석*

방기석, 최진영
고려대학교 컴퓨터학과
e-mail : {kbang,choi}@formal.korea.ac.kr

A Comparison and Analysis of Formal Verification for Object Oriented Software using SPIN

Ki-Seok Bang, Jin-Young Choi
Dept. of Computer Science and Engineering, KOREA University

요 약

객체 지향형 소프트웨어가 개발되고 발전함에 따라 동시성 프로그래밍의 수요가 증가하고 그 기법 역시 다양해 지고 있다. 이 결과 소프트웨어의 복잡도가 매우 높아지고, 특히 수행시간 복잡도(run-time complexity)가 매우 높아져서 소프트웨어의 설계와 안정성 검증 분야에 많은 문제가 발생되고 있다. 현재 객체 지향형 소프트웨어의 검증을 위한 연구로 소스 코드를 직접 변환시켜 검증하는 방법과 수행시점에서의 검증을 위해 정형 검증 도구가 직접 객체를 표현할 수 있는 방법을 연구하고 있다. 본 논문에서는 소스코드의 변환을 이용한 검증 도구인 java2spin 과 spin 의 동적인 확장인 dSPIN 에 대해 살펴보고 두 도구의 차이점과 장단점을 살펴본다.

1. 서론

최근에 객체 지향형 소프트웨어가 개발되고 발전함에 따라 동시성 프로그래밍의 수요가 증가하고 그 기법 역시 다양해 지고 있다. 그와 함께 동시수행 소프트웨어(concurrent software)의 재사용성과 안정성을 증대시키기 위한 기법이 개발되고있다. 이 결과 소프트웨어의 복잡도가 매우 높아지고, 특히 수행시간 복잡도(run-time complexity)가 매우 높아져서 소프트웨어의 설계와 안정성 검증 분야에 많은 문제가 발생되고 있다.

현재 C++나 Java 와 같은 객체지향형 프로그래밍 언어로 개발되는 소프트웨어를 정형 명세 및 검증하는 방법이 발표되고 있다. 그 중 대표적인 것으로 객체지향형 언어의 소스코드를 SPIN 과 같은 정형 검증 도구의 입력 언어의 형태로 변환시켜 검증하는 방법과 수행시점에서의 검증을 위해 정형기법 도구를 동적으로 확장시키려는 연구가 진행중이다.

전자의 경우 Java2Spin[2]이라는 도구가 개발중이며 현재 정형 검증 도구인 SPIN[3][4]과 함께 사용중이다. 그러나 이 방법은 객체지향 프로그램을 변환시키는 과정에서 매우 많은 제약이 따른다. 즉, 정형 기법 도구는 매우 정적인 특성이 있기 때문에 객체와 같은 동적인 특성을 모델링하기 위해서는 그 규모와 설계를 달리 해야 하기 때문이다. 특히 객체의 동적인 생성 혹은 소멸, 포인터와 같은 참조 변수, 함수의 호출과 같은 중요한 속성은 표현할 수가 없다.

후자의 경우 이런 단점을 극복하기 위해 SPIN 의 객체지향형 확장인 dSPIN[1]이 개발중이다. 이 도구는 정적인 분석 도구인 SPIN 에 동적인 특성을 표현할 수 있는 확장시켜 모델링 자체를 객체 지향형으로 할 수 있도록 한다. 이 도구를 이용하면 객체의 생성 및 소멸, 함수의 호출과 같은 특성을 용이하게 표현할 수 있을 것으로 보인다. 논문의 구성은 다음과 같다. 2 장에서는 객체지향형 소프트웨어를 검증하기 위한 방법론 중 Java2Spin 을 소개하고 dSPIN 을 설명한다. 그리고 3 장에서는 간단한 자바 프로그램에 대해 Java2Spin 과 dSPIN 을 이용한 검증 결과를 보이고 비교하며 4 장에서 결론을 맺는다.

2. 객체 지향형 소프트웨어의 검증

2.1 Java 코드의 변환을 이용한 검증 – Java2Spin

이 연구는 Java 프로그램을 SPIN 의 입력 언어인 Promela 로 변환하고, 변환된 입력을 SPIN 으로 검증하는 연구이다. 이 연구에서는 Java 소스 코드를 분석해서 모델 체킹 기법에 사용될 수 있도록 유한 상태 기계의 형태로 모델링하고 이를 SPIN 의 입력 언어의 형태로 자동 변환해 준다. 그리고 최종적으로 객체 지향형 소프트웨어 뿐 아니라 일반적인 형태의 소프트웨어로부터 정형 검증 도구의 입력 언어 형태로의 자동화 변환 도구를 개발하고자 한다.

본 연구는 1997 년도 특정기초연구지원사업(97-01-00-09-01-3)으로 이뤄졌습니다.

이러한 연구의 결과로 Java 소스코드를 분석해서 SPIN 의 입력 언어인 Promela 로 변환시켜주는 도구의 초기 버전이 발표되고 있다. 가장 대표적인 것으로 Java2spin 이 있다. 이 도구는 모두 Java 소스코드를 Promela 로 자동 변환해 주는 도구이다. 이렇게 변환된 입력을 SPIN 으로 정형 검증할 수 있다. 주로 deadlock 과 같은 특성을 검증할 수 있다.

이 도구는 크게 두 부분으로 나누어 진다. 첫번째 부분은 Java 소스 프로그램을 파싱하여 문법의 오류를 찾고 이것을 Promela 의 표현으로 바꿀 수 있는 symbol table 을 작성하게 된다. 그 후 파서에 의해 작성된 새로운 구조에 대한 의미적 오류를 발견하기 위해 static checker 가 동작한다. 이 검사 결과 연산자의 형태 오류와 제어문의 변환 시 생길 수 있는 잘못된 제어 변환이나 정지 등과 같은 제어의 흐름 오류를 검사한다. 또한 객체가 가질 수 있는 중요한 특성인 상속이나 쓰레드와 같은 부분의 올바른 변환을 검사하기 위해 변수의 사용과 순환 오류를 검사하게 된다.

오류의 검사가 끝나면 정형 검증 도구의 입력에 알맞은 내부적인 자료구조를 표현한다. 이 구조를 이용해서 Promela 의 형태로 변화가 된다. 이 단계에서 쓰레드와 같은 객체의 표현을 위해 매우 많은 새로운 정보가 추가된다. 이 정보들은 어떤 Java 프로그램을 변환하더라도 모두 공통적으로 들어가게 되므로 템플릿과 같은 형태가 된다. 대표적으로 객체의 동기화를 위한 정보가 추가되며, wait, lock, notify, sleep 과 같은 쓰레드의 내부적인 동작을 모두 추출하여 Promela 의 형태로 변환한다. 이런 변환 결과 생성되는 Promela 코드의 크기가 매우 커지며, 구성되는 상태가 매우 많아지게 된다. 그러나 현재 객체 지향형 소프트웨어의 특성을 표현할 만한 선형 시제 논리[5]가 존재하지 않아 많은 특성에 대한 검증은 사실상 어렵다.

2.2 정형 검증 도구의 동적인 확장 - dSPIN

대부분의 정형 검증 도구와 마찬가지로 SPIN 역시 객체 지향형 소프트웨어의 검증을 위해 개발된 것은 아니다. 따라서 객체 지향형 언어를 Promela 의 형태로 변환할 경우 매우 많은 제약사항이 가해진다. 또한 변환된 코드의 경우 그 규모가 매우 커지고 유한 상태 기계의 경우 상태의 수가 많아지게 되므로 상태 폭발과 같은 문제가 매우 심각하게 나타날 수 있다. 현재 개발된 도구들은 객체의 동적인 생성 및 소멸, 포인터와 같은 동적인 참조, 함수의 호출과 같은 특성을 그대로 표현할 수가 없다. 이러한 제약사항 때문에 검증 모델의 규모의 확장과 검증 가능한 특성의 제약이 불가피하다.

이러한 제약사항을 극복하기 위해 객체 지향형 언어의 변환과는 다른 방법으로 SPIN 을 객체 지향형으로 확장하려는 연구가 진행중이다. 가장 대표적인 것이 dSPIN 의 개발이다. 이것은 기본적인 SPIN 의 문법에 객체 지향형 언어의 명세를 위한 구문을 추가하여 확장한 것이다.

dSPIN 은 기존의 SPIN 이 검증할 수 있는 것 뿐 아니라 몇 가지 새로운 명세 및 정형 검증이 가능하다. 기본적인 SPIN 은 전형적인 정형 검증 도구이기 때문에 매우 정적인 수행을 한다. 따라서 대부분의 정적 분석 도구와 마찬가지로 프로그램이 컴파일되는 동안에 자료의 흐름 분석과 같은 분석을 수행하도록 되어 있다. 이는 특정한 수행과정에 있어서 일정한 수행 경로를 쫓아 가면서 변수의 변화를 분석하고 이 변수의 올바르지 못한 접근이나 원치 않은 변화와 같은 것으로 오류를 찾아내는 방법이다. 그러나 동적인 프로그램의 경우 이 방법은 완전하지 못하다. 동적인 프로그램에서는 함수가 수행되는 순간에 동적으로 호출되고 해당 함수가 호출되기 전까지는 모든 자료의 흐름을 관찰하기는 불가능하다. 즉 수행시점에서의 프로그램의 분석이 반드시 필요하다.

기존의 객체 지향형 소프트웨어 검증을 위한 연구에서는 정적인 정형 검증도구를 목표로 수행되어 이러한 동적인 특성을 완전히 보여주지 못한 것이 사실이다. 즉, 수행시점에서 나타날 수 있는 특성들을 추상화 하고 제한하여 미리 선언해 주는 방법을 사용하여 정형 검증을 수행한다. 이런 방법으로 정형 검증을 수행할 경우 나타날 수 있는 문제는 필요 정보의 결여와 필요 이상의 상태 폭발과 같은 것이 있다. 객체의 동작을 정적으로 표현하기 위해서는 객체 지향형 표현을 비객체의 형태로 해야 한다. 이렇게 하기 위해서는 쓰레드와 같은 기능을 모두 표현해 줘야 하기 때문에 정형 검증 모델의 크기가 매우 커지게 된다. 검증 모델의 크기가 커지게 되면 유한 상태 기계로 변환할 경우 상태의 수가 매우 많아지게 되어 검증 수행 시 상태의 폭발과 같은 문제를 야기하기가 쉽다.

dSPIN 은 이러한 제약을 줄이고 메모리의 효율적인 사용을 주된 목적으로 개발되고 있다. 이를 위해 dSPIN 에서는 기존의 SPIN 에는 없던 몇 가지 문법을 추가했다.

(1) dSPIN 에 추가된 문법

dSPIN 에서는 동적인 메모리의 조작과 객체의 생성을 명세하기 위해 Pointer 의 개념을 추가하여 동적인 메모리 참조 모델을 가능하도록 하였다. 포인터는 이전에 선언되었거나 동적으로 생성된 객체의 수행시점에서의 메모리 번지를 기억하고 있다. 일반적인 C 언어와 같은 언어와 마찬가지로 dSPIN 에서의 포인터 역시 lvalue 는 rvalue 의 메모리 번지를 저장하고 있으며, lvalue 는 객체의 생성을 나타내는 연산자에 의해 선언된다. 이 포인터 연산자가 사용 가능해짐에 따라 기존의 SPIN 에서 사용되었던 typedef 구문을 이용하여 연결 리스트와 같은 구조체의 선언이 가능하다.

객체의 생성과 소멸을 명세하기 위해 dSPIN 에서는 new 와 delete 라는 연산자를 새로 정의하였다. new 는 객체의 생성을 표현하기 위한 연산자이며 자료형에 따라 메모리를 할당하는 연산을 취하게 된다. 이것은 수행시점에 메모리를 할당하기 때문에 힙(heap)영역에 할당되게 된다. 따라서 동적인 객체의 표현이 가능하고 메모리의 관리가 용이해 진다. 생성된 객체의 소멸을 표현하기 위해 delete 라는 연산자를 추가하였다. 이 연산자는 힙으로부터 해당 번지를 없애는 동작으로 동적인 객체의 소멸을 나타낸다.

dSPIN 의 두드러지는 특징 중 하나는 함수의 호출이 가능한 것이다. 기존의 SPIN 에서는 함수의 호출이 지원되지 않았다. inline 이라는 형식을 빌어서 매우 길고 자주 쓰이는 부분을 짧게 사용할 수 있도록 했지만 이것이 함수를 나타내지는 않았다. 함수를 호출하였을 때 프로그램의 제어가 그 함수 쪽으로 넘어가지 않고 그대로 다음 구문으로 넘어간다. 즉 인터리빙과 병렬 수행에 중심을 둔 설계이다. dSPIN 에서는 함수를 호출하면 제어가 호출된 함수로 완전히 넘어간다. 즉 일반적인 프로그램과 같은 함수의 호출 동작이 가능하다. 또한 객체의 동작을 명세하기 위해 함수의 다형 호출(polymorphic function call)을 지원한다.

```
class Node
{
  int info;
  node next;

  public Node insert(Node x, int y)
  {
        Node temp = new Node();
        temp.info =y;
        temp.next = x;
        return temp;
  }

  public Node delete(Node x)
  {
        Node temp = new Node();
        temp = x.next;
        return temp;
  }
}
```

그림 1 간단한 연결 리스트

3. Java2Spin 과 dSPIN 을 이용한 검증

이번 장에서는 Java2Spin 과 dSPIN 을 이용한 간단한 예를 들고자 한다.

다음에 보인 예는 단순한 연결 리스트의 연산을 Java 로 구현한 것이다. 리스트의 연산에는 제일 앞 노드의 앞부분에 새로운 노드를 연결하는 insert()와 앞부분부터 하나씩 노드를 없애는 delete()의 두 가지 연산만을 가지고 있다. 그림 1 은 이러한 연산을 수행하는 소스코드의 일부분이다.

위의 코드를 Java2Spin 을 이용해서 변환 시켰을 때 다음과 같은 Promela 코드가 생성되게 된다.

그림에서 보는 바와 같이 객체의 동작을 표현하기 위한 매우 많은 부분이 추가된다. 사실 위의 코드에서는 쓰레드와 같은 객체의 표현이 전혀 나타나지 않았다. 그러나 모든 코드에 대해서 객체의 동기화를 나타내는 부분을 구현하기 때문에 분석이 복잡하고 불필요한 상태가 많이 생기게 된다.

```
/*** Generic lock/unlock definitions ***/

#define LOCK      0
#define UNLOCK    0
#define NOTIFY    1

#define _lock(object, inst_no, tid)              \
        sync_##object[inst_no].lock ? LOCK

#define _unlock(object, inst_no, tid)            \
        sync_##object[inst_no].lock ! UNLOCK

#define lock(object, inst_no, tid)               \
        if                                       \
        :: (sync_##object[inst_no].pid != tid) -> \
            sync_##object[inst_no].lock ? LOCK;  \
            sync_##object[inst_no].pid = tid;    \
        :: else ->                               \
            skip;                                \
        fi;                                      \
        sync_##object[inst_no].lockcnt ++
                    ⋮

#define wait1(object, inst_no, tid)      \
        sync_##object[inst_no].lock ! UNLOCK; \
        sync_##object[inst_no].nwait ++

#define wait3(object, inst_no, tid)      \
        sync_##object[inst_no].lock ? LOCK

#define MAX_INSTANCES       11
#define MAX_ARRAY_SIZE      10
                    ⋮
```

그림 2 Java2Spin 에 의해 변환된 코드의 일부분

이번 연구에서 나타난 문제점 중 하나는 Java2Spin 이 SPIN 의 버전에 매우 민감하게 반응하고 있는 것이다. 현재 SPIN 역시 매우 빠르게 개발이 진행중이기 때문에 그 버전의 변화가 자주 발생한다. 지금 개발된 Java2Spin 은 SPIN 3.2.0 을 기준으로 개발되어 현재 사용하는 SPIN 에서는 오류를 발생하고 있다. 따라서 이번 연구에서도 Java2Spin 의 일부를 수정해서 실행을 시켰어야 했다. 즉 Java2Spin 의 개발 시 SPIN 의 버전에 동적으로 대응할 수 있는 능력을 부여하는 것이 반드시 필요할 것으로 보인다. 이렇게 변환된 코드를 살펴보면 원래 Java 코드에 있는 출력문과 같이 프로그램의 흐름에 직접 연관되지 않은 부분은 삭제하도록 되어 있다. 따라서 SPIN 을 이용한 시뮬레이션을 수행하면 아무 결과도 외부로 보여주지 않는다. 따라서 반드시 변수의 변화를 살펴봐야만 동작의 오류를 발견할 수 있다. 결국 화면출력과 같은 구문의 보전이 필요할 것으로 보인다.

위의 연결 리스트를 dSPIN 으로 구현하면 그림 3 과 같다.그림에서와 같이 마치 C 를 이용해서 연결 리스트를 구현한 것과 동일하다.

이렇게 구현한 dSPIN 입력 형태는 Promela 와 거의 비슷하며 dSPIN 을 이용해서 시뮬레이션 및 검증이 가능하다. 위의 명세를 보면 기존의 SPIN 에서 제공되지 않던 함수의 호출과 포인터의 연산이 추가되어 있다. 그 결과 객체의 생성과 동작을 간결하게 직접적으로 명세할 수 있다.

```
typedef node
{
      int num;
      node &next;
```

```
};

function insert(node &list; int x) : node&
{
    node &temp;

    temp = new node;
    temp.info = x;
    temp.next = list;
    return temp;
}

function extract(node &list) : node&
{
    node &temp;

    temp = list.next;
    delete(list);
    return temp;
}
```

그림 3 dSPIN 을 이용해 구현한 연결 리스트

dSPIN 은 아직 X 윈도우 환경이 지원되지 않기 때문에 명령행 방식의 시뮬레이션과 검증만이 가능하다. 위의 리스트를 검증한 결과는 그림 4 와 같다.

```
State-vector 95 byte, depth reached 58, errors: 0
        43 states, stored
         0 states, matched
        43 transitions (= stored+matched)
         0 atomic steps
hash conflicts: 0 (resolved)
(max size 2^18 states)

1.508        memory usage (Mbyte)

   + 0 reference counting collector runs

unreached in proctype :init:
            (0 of 14 states)
```

그림 4 dSPIN 으로 검증한 결과

4. 결론

C++나 Java 와 같은 객체 지향형 언어로 구현된 소프트웨어를 SPIN 과 같은 정형 검증 도구의 입력 형태로 변환하는 연구가 활발히 진행되고 있다. 본 논문에서 살펴본 Java2Spin 과 dSPIN 이 그 대표적인 도구이다. Java2Spin 을 사용하면 Java 코드를 자동으로 SPIN 의 입력 언어인 Promela 로 변환할 수 있다. 그렇게 변환된 코드를 이용하면 정형 검증이 가능하다. 현재 검증 가능한 특성은 deadlock 과 같은 가장 기본적인 특성이다. 그러나, 변환 과정에서 객체의 특성을 나타내는 많은 부분이 제한되는 단점이 있다. 즉, 정적인 검증 도구로의 변환이 일어나기 때문에 객체의 동적인 생성, 소멸과 같은 것은 검증하기 불가능하다. 그리고 변환 과정에서 출력문과 같은 부분은 모두 추상화되어 사라지기 때문에 SPIN 으로 시뮬레이션 할 경우 사용자가 쉽게 이해할 수가 없는 단점이 있다. 또한 이 도구는 SPIN 의 버전에 매우 민감하기 때문에 현재 개발된 SPIN 을 이용해서는 시뮬레이션 및 검증이 불가능하다. 기본적으로 객체 지향형 소프트웨어를 정적인 분석방법으로 검사할 때 발생 가능한 모든 객체에 대해 배열의 형태로 미리 선언을 해 놓아야 한다. 이 도구는 최대 10 개의 객체에 대해 검사를 할 수 있도록 하고 있다. 대부분의 경우 이 규모의 크기에 적당한 결과를 보일 수 있지만, 10 개 이상의 객체가 발생될 경우에도 10 개 까지의 객체만이 정의가 될 수 있기 때문에 그 외의 객체가 어떤 동작을 보일 수 있는지는 검증이 불가능하다. 따라서 앞으로 도구를 개발할 때 배열의 크기를 소스코드에서 생성하는 객체의 개수에 민감하게 변화시킬 수 있는 구현이 반드시 필요할 것이다.

SPIN 의 동적인 확장 도구인 dSPIN 을 이용하여 객체의 모델링을 수행하고, 이를 검증하면 별다른 제약이나 상태의 폭발 문제 없이 검증이 가능하리라 본다. 또한 객체의 동적 생성 및 소멸의 명세가 가능하며 함수의 호출과 같은 기능이 제공되기 때문에 객체 지향형 소프트웨어의 모델링에 더욱 적합하리라고 본다.

또한 소스 레벨에서의 검증이 아니라 수행시점에서의 검증이기 때문에 더욱 명확한 객체의 정형검증이 가능하리라 생각된다. 그러나 이 방법은 아직 사용하기가 불편하고 특별한 객체의 표현이 없는 명세에 대해 수행할 경우에는 필요없는 메모리의 낭비가 유발된다.

현재 명세한 객체 지향형 소프트웨어의 특성을 표현할 수 있는 선형 시제 논리는 아직 존재하지 않는다. 기존의 선형 시제 논리로는 객체의 동적인 특성을 원활히 표현하기가 어렵다. 때문에 현재 소프트웨어의 deadlock 과 같은 특성만이 검증 가능하다. 향후 연구과제로 객체 지향형 소프트웨어의 특성을 표현할 수 있는 선형 시제 논리의 개발과 그 적용이 반드시 필요할 것이며, 이렇게 제안되는 선형 시제 논리로 보다 큰 객체 지향형 소프트웨어를 검증해 보아야 할 것이다.

참고문헌

[1] C. Demartini, R. Iosif, R. Sisto, "dSPIN: A Dynamic Extension of SPIN", in *Proc. 6th International SPIN Workshop, Theoretical and Practical Aspects of SPIN Model Checking*, Toulouse, September 1999.

[2] R. Iosif, R. Sisto, "Modeling and validation of Java multi-threading applications using Spin", in *Proc. 4th International SPIN Workshop, Theoretical and Practical Aspects of SPIN Model Checking*, Paris, November, 1998.

[3] Gerald J. Holzmann, "The Model Checker SPIN", IEEE Transactions on Software Engineering, VOL. 23, NO 5, MAY 1997.

[4] Gerald J. Holzmann, "Design and Validation of Computer *Protocols*", *Prentice Hall, 1991.*

[5] Zohar Manna, Amir Pnueli, "The Temporal Logic of Reactive and Concurrent Systems – Specification", Springer-Verlag, 1996.