STATEMATE MAGNUM 모델체킹을 위한 정형명세 기법 연구

김진현°, 안영아, 장상철, 이나영#, 최진영
고려대학교, #서울대학교
{jhkim°, ellaahn, scjang,choi}@formal.korea.ac.kr, #grasia2@sis1.snu.ac.kr

Study on Specification Method for Model Checking in STATEMATE MAGNUM

Jin Hyun Kim° Young Ah Ahn, Sang Chul Jang,Na Young Lee#, Jin Young Choi
Dept. of Computer Science, #Dept. of Nuclear Engineering

요 약

STATEMATE는 Statecharts로 시스템의 행위를 설계하는 도구이다. 근래 들어, STATEMATE MAGNUM은 설계 뿐 아니라 모델체킹을 이용한 정형검증의 기능을 가지고 있다. 모델체킹은 상태 기반의 설계명세 된 시스템을 시제 논리로 그 요구 명세를 기술하여 설계명세가 요구명세를 만족시키는지를 검증하게 된다. 하지만 설계명세가 큰 경우, 모델체킹 시 상태폭발을 일으켜 시스템을 검증하지 못하게 한다. 모델체킹의 상태 폭발을 줄이기 위해서는 기본적으로 모델체커의 알고리즘을 개선시키거나, 모델을 추상화 시킨다. 본 연구에서는 모델을 추상화 시키더라도 검증 결과에는 별 영향을 주지 않는 부분을 추상화하고, 검증 결과에 직접 적인 영향을 주는 부분을 상세 명세하는 기법을 적용하여 실시간 운영체제의 코드를 어떻게 검증을 하는지를 보여준다.

1. 서 론

STATEMATE MAGNUM[1]은 Statecharts[2]를 통해 행위를 명세하고 이를 시뮬레이션 하는 도구로 알려져 있다. 주로 자동차나 항공 관련의 시스템 레벨의 설계에 이용되어 왔으며 근래 들어 STATEMATE MAGNUM은 자동차의 내장형 시스템을 설계하고 그 설계로부터 코드를 생성하여 시스템에 적용시키고 있다.

원자력 발전, 자동차, 항공 관련 시스템 등의 Safety-critical 시스템의 중요성이 증가되면서 STATEMATE MAGNUM의 도구 역시 시뮬레이션 뿐 아니라 정형검증(Formal verification)도구를 적용하여 설계에 대한 검증을 수행한다. 이 도구에서는 사용하는 정형검증 기법이 모델체킹[3]이다. 모델체킹은 상태 기반의 검증 기법으로서 시스템의 설계(model)를 Kripke[3] 구조로 모델링 한 후, 모든 상태에선 시제논리로 명세한 요구명세를 만족시키는지를 검사(checking)하게 된다. 모델체킹은 모든 검증을 자동으로 수행함으로써 설계자나 검증자에게 상당한 부담을 덜어주는 장점을 지나고 있으나 시스템의 리소스를 사용하는 면에서 상태폭발을 일으켜 시스템의 검증을 어렵게 한다. 이 문제를 해결하기 위한 방법으로, 모델체커의 모델체킹 알고리즘과 자료구조를 개선시키는 방법과 모델을 추상화 하는 방법이 있다. 하지만 모델을 추상화 하는 방법은 검증하려는 요구 명세를 만족시킨다 하더라도 추상화 때문에 만족하는 경우가 생길 수 있다. 즉 추상화된 모델이 요구명세를 만족시킨다 하더라도 추상화되지 않은 모델이 요구명세를 만족시키지 않을 수 있다. 본 연구에서는 이러한 것을 체계적으로 분류하고 비교하여 보다 정제된 명세 기법을 제안한다.

본 연구의 최종 목표는 설계 명세를 검증할 때, 상태폭발을 최소화하면서 검증된 추상화된 모델이 실제 모델의 검증 성질을 반영하는 정형 명세 방법을 연구하는 것이다.

본 논문은 다음과 같이 구성되어 있다.

2. STATEMATE MAGNUM의 정형명세 및 검증

I-LOGIX사의 STATEMATE MAGNUM에서 제공하는 언어는 다음과 같이 나눌 수 있다.

- Activity-charts
- Control-charts(Statecharts)
- Module-charts

Activity-charts는 주로 시스템의 기능적인 관점에서 시스템을 설계하게 된다. 즉 특정 기능에 어떠한 입출력이 있는지를 명세하는데 그 목적을 지니고 있다. Statecharts로 명세 되는 Control-charts는 Activity-charts에서 주고 받는 입출력의 조건과 그 반응을 구체적으로 명세하게 된다. 다시 말해, 시스템이 어떠한 행위를 하는가를 명세하게 된다. Module-chart는 주로 물리적 관점에서 시스템이 어떻게 구성되어 있는가에 관심을 두고 시스템을 명세하게 된다. 즉 CPU나 메모리 등의 모듈 관점에서 시스템을 명세하게 된다. 여기서 염두 해 두어야 할 점은 소프트웨어라는 특성이다. 소프트웨어는 여러 구조적 특징을 지니기 때문에, 소프트웨어는 모듈, 기능, 객체 등 다양하게 나누어 질 수 있다.

본 연구에서 관심을 두고자 하는 부분은 소프트웨어의 정형명세, 정형검증 그리고 구현에 관점이다. 다시 말해, 정형 검증된

정형명세(설계)가 어떻게 소프트웨어 구현 언어로 변환될 수 있는지, 혹은 구현된 소프트웨어가 어떻게 정형검증을 위해 정형명세로 변환될 수 있는지에 대해 연구한다. 이 때, 기준이 되는 것은 설계에 대한 검증이다. 정형검증을 현재로서 구현 언어에 할 수 없기 때문에, 설계 단계에서 설계 언어(정형명세언어)로 검증되게 되는데 이때 우리가 관심을 갖는 검증 기법은 모델체킹이다. 앞에서 언급한 것처럼 모델체킹의 약점이 상태폭발을 극복하기 위해 실제 시스템을 추상화 해야 하는데, 이때 추상화 기법 및 방법이 제안되어야 한다. 본 논문에서는 이러한 추상화 기법을 제안하기 위해 두 언어 즉 정형명세 언어로 구현 언어를 비교한다. 그리고 어떻게 변환될 수 있는지를 비정형적으로 기술한다.

본 연구에서는 설계나 C코드 검증을 위한 정형명세 언어로 Statecharts를 사용한다. 그리고 구현 언어로는 C를 사용하며 검증 대상은 내장형 시스템을 위한 실시간 운영체제이다. 검증 성질을 스케줄링 알고리즘의 정확성과 우선순위 전도 문제, 그리고 교착상태를 대상으로 검증을 수행한다.

3. STATECHARTS와 C 언어의 비교

Statecharts는 반응형 시스템의 정형명세 언어로서 다음과 같은 구성을 갖는다.

Statecharts = State-diagram + Hierarchy + Orthogonality + Communication

State-diagram은 상태(State)와 전이(Transition)으로 구분되는데, 이때 전이를 일으키는 것은 각 전이에 붙은 레이블이 된다. 이 레이블에는 다음과 같은 요소들이 들어 있다.

e[c]/a

e는 이벤트로서 broadcasting된 이벤트 가리키고, c는 조건으로서 전이 될 때의 제약사항을 가리킨다. 즉 c를 만족시킬 때에만 전이가 일어나는 것이다. a는 전이가 일어날 때에 수행하는 행위이다.

C언어는 전형적인 imperative 언어로서 프로그램의 구조는 다음과 같이 분류할 수 있다.

- Module(Function, Procedure)
- Statement
  - Assignment
  - Condition
  - Loop

C는 잘 알려진 언어임으로 본 논문에서는 자세한 설명은 생략한다.

따라서 두 언어의 특징을 비교 하면 다음 표 1과 같다.

[표1]에서 볼 수 있는 것처럼 Statecharts는 외부 환경에 대한 반응을 처리하는 시스템으로 주요 명세 대상이다. 이에 반해 C는 명령이 주로 데이터의 변수로 들어온다. 만약 모델체킹을 위해 Statecharts로 C를 명세한다면 C의 모듈에 대한 파라미터에 대응되는 것을 Statecharts의 broadcasting 이벤트 혹은 액션의 데이터 변수로 대응될 수 있다. 이때 발생할 수 있는 문제는 Statecharts의 Broadcasting 때문에, 설계의 한 부분에서 특정 이벤트가 발생하거나 혹은 변수가 변한다면 전체 시스템의 모듈에서 알 수 있는 경우도 생긴다. 이 문제는 변수의 *Scope*으로 해결해야 한다.

	Statecharts	C
명세 대상	Reactive system	Imperative system
Communication	Broadcasting	Parameter Return variable Global variable
Statement	Assignment	Assignment
Condition	전이조건 Action 조건	If…else
Loop	Self transition	While… Do…while

표 1 Statecharts와 C언어의 비교

4. 소프트웨어 정형 검증을 위한 정형명세

본 논문에서는 C언어로 된 소프트웨어를 검증하기 위해 정형명세 언어인 Statecharts로 변환하기 위한 규칙을 살펴보자.

먼저 소프트웨어의 특성을 검증하기 위해 Statecharts로 모든 소프트웨어 프로그램을 1:1로 변형하는 것은 상당히 어렵고 까다로운 일이다. 따라서 적절한 추상화 혹은 생략이 필요하다. 특히 모델체킹의 상태폭발을 막기 위해서는 상당한 수준의 추상화가 필요하다.

본 논문에서의 추상화 할 때 기준은 아래와 같다.

- *Property*에 직접적인 영향을 주지 않는 모듈:
  - 상태와 이벤트로 처리
    - 파라미터 :이벤트
    - 함수 : 상태 :
    - 파라미터 출력 :
- 데이터 변수처리:
  - 변수가 가질 수 있는 값의 범위를 제한

이 외에 시간적인 검증 위해 즉, 특정 프로그램의 시간적 진행을 추상화 하기 위해 프로그램 counter를 step에 따라 진행시키는 것으로 프로그램의 진행을 추상화 시켰다.

4.1 정형명세의 예

본 논문에서는 소프트웨어 검증을 위해 대상 소프트웨어 시스템으로 삼은 것은 내장형 시스템을 위한 실시간 운영체제이다.

내장형 실시간 운영체제는 safety-critical 시스템의 핵심적인 부분이기 때문에 철저한 검증이 필요하다. 따라서 이에 대한 검증의 일환으로 정형검증을 실시한다.

우선 본 논문에서 실시간 운영체제의 검증 특성으로 다음과 같은 특성을 검증하고 한다.

- 스케줄링 가능성
- 데드락

```
void OSSched (void)
{
   INT8U y;


   OS_ENTER_CRITICAL();
   if ((OSLockNesting | OSIntNesting) == 0) {
      y = OSUnMapTbl[OSRdyGrp];
      OSPrioHighRdy = (INT8U)((y << 3) +
OSUnMapTbl[OSRdyTbl[y]]);
   if (OSPrioHighRdy != OSPrioCur) {
      OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
      OSCtxSwCtr++;
       OS_TASK_SW();
     }
   }
   OS_EXIT_CRITICAL();
}
```

그림 1 RTOS의 소프트웨어 코드[5]

- 우선순위전도

이를 검증하기 위해 앞절에서 언급된 방식을 시스템을 추상화하여 정형명세 하였다.

이를 앞절에서 언급한 추상화에 따라 정형명세 한 그림이 [그림 2]이다. 정형 검증하면서 다음과 같은 잘못된 명세 결과를 찾을 수 있었다.

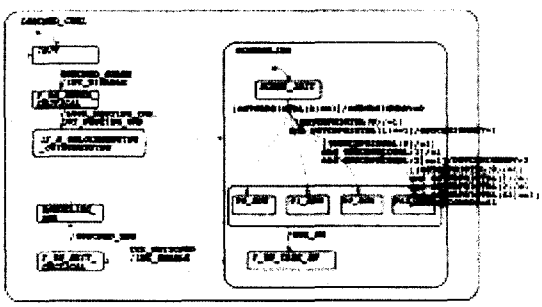

그림 2 Statechart로 정형명세된 스케줄링 모듈

위의 그림에서 검증 특성과 관련이 없다고 여겨진 INT_ENABLE이 데드락 발생에 중요한 요인이 되었다. INT_ENABLE은 [그림2]에서 이벤트로 처리 되어 있어서 스케줄링 도중에 스케줄링 요청이 다시 들어오면 데드락이 모델체킹을 통해 발견되었다. 이를 수정한 그림이 [그림3]이다.

[그림 3]에서는 INT_ENABLE이 이벤트가 아니라 데이터로 처리되고 있다. 이렇게 함으로 특정 step 구간 동안 그 값을 계속 유지하게 된다.

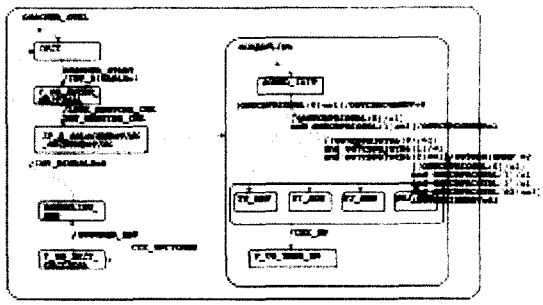

그림 1 수정된 스케줄링 모델

5 결론 및 향후 연구 과제

본 논문에서는 소프트웨어 검증에 대해 정형검증을 적용하기 위해 소프트웨어 코드에서 정형명세 언어로 어떻게 변환하고 추상화하는지에 대해 기본적인 개념을 기술하였다. 이러한 변환 규칙을 사용해서 정형검증 한 결과, 특정한 변수의 추상화로 인해 전체 시스템의 검증에 영향을 변수를 발견하였다.

전역 변수의 경우는 시스템의 모든 부분에 영향을 줄 수 있기 때문에, 이벤트로 추상화 시키는 것 보다는 데이터로 추상화 시킬 필요가 있음을 알게 되었다.

본 논문에서는 정형명세 언어인 Statecharts를 이용하여 소프트웨어를 분석하기 위해 추상화 할 필요가 있는데, 이를 어떻게 할 것인지에 대한 기초적인 조사 결과와 추상화, 모델체킹을 통해 발견한 추상화의 문제점을 등을 기술하였다.

이렇게 해서 소프트웨어를 검증하기 위한 정형명세 언어의 사용에 대한 추상화 규칙을 세울 수 있었다.

향후 연구 과제로서는 이를 정형적인 규칙을 세워 보다 세부적인 검증코드를 생성하고 특히 모델체킹의 상태 폭발을 줄이기 위한 코드를 어떻게 생성할 것인지를 연구해야 할 것이다.

참고문헌

[1] http://www.ilogix.com

[2] D.Harel " Statecharts: A Visual Formalism for Complex Systems." Science of Computer Programming, vol. 8, pp. 231-274, 1987

[3] Edmumd M. Clarke, Orna Grumberg, Doron A.Peled, " Model Checking" The MIT Press, pp.1. 1999

[5] Jean J. Labrosse, " MicroC/OS-II, The Real-Time Kernel" , CMP Books, 2001, pp 85