Testing을 위한 Simulink mdl파일의 파싱과 C# 자료구조

최용재°, 최경희, 정기현
아주대학교
lnyarl@gmail.com, khchoi@ajou.ac.kr, khchung@ajou.ac.kr

Parsing simulink mdl file and C# data structure for Testing

Choi yong-jae°, Choi gyeong-hee, Chung gi-hyeon
Ajou University, Embedded Application and System Testing Lab

요 약

많은 분야에서 사용되는 matlab을 이용한 test방식을 좀 더 유연하고 원하는 기능을 만들어 사용하기 위해 matlab의 데이터 저장 파일인 mdl 파일의 내부 구조를 분석하고 구문 분석기로 파싱하여 실행 가능한 그래프를 나타내는 자료구조를 만든다.

1. 서론

산업에서는 지금도 수많은 전자제품들이 만들어지고 있다. 시대가 발전 할수록 많은 전자제품들이 빠른 cpu를 사용하고 더 많은 연산이 가능하게 되었다. 따라서 전자 제품들은 더 많은 일을 하게 되었다. 전자 제품들은 이렇게 복잡해졌고 그에 따라 설계또한 복잡해졌다.

복잡한 설계에 뒤따르는 것은 테스트의 중요성이다. 요구사항은 많아졌고 그에 따라 테스트 비용도 같이 올라갔다.

모델링 단계에서 테스팅을 하면, 더 적은 비용으로 오류를 잡을 수 있다. 그러한 면에서 matlab의 효용성은 대단히 크다. 모델을 이용한 테스트를 가능하게 해주기 때문이다. matlab에서 제공하는 simulink의 모델링 기능은 자동차, 선박등 여러 제품을 제작하는 데에 설계도구로써 손색이 없다. Matlab의 모델은 실행가능하며 그 결과 모델을 테스트하여 검증 가능하다.

하지만 simulink에서 제공하는 모델을 실행하는 기능이나 테스트를 하는 기능은 많은 기능을 제공하지 않고 제품의 종류에 따라 필요한 커스터마이징을 하기 힘들다. 따라서 더 편하고 제품에 알맞은 테스팅 프로세스를 위해 simulink로 작성한 결과물(mdl 파일)을 이용해 더 수정이 용이한 자료구조로 만들어내는 작업이 필요하다

따라서 본 논문에서는 그러한 자료구조를 만드는데에 필요한 기술들과 방법을 설명 하였다. 자료구조를 만드는데에 사용된 언어는 c# .net환경이다. C#은 다소 느리지만 개발의 생산성과 앞으로의 사용 빈도를 고려했을때 많은 장점이 있을 것이라고 판단했기 때문에 선택했다.

본 논문은 5 절로 되어 있다. 2절에서는 mdl 파일은 2차 가공하기 위한 관련 연구로 mdl file의 구조와 구문 분석기에 대해서 알아보고, 3절에서는 자료구조의 설계에 대해서 설명한다. 4절에서는 추가적인 연구의 필요성과 그 종류에 대해서 논한다.

2. 관련연구

자료구조를 만들기 위해 mdl파일을 분석하고, 분석한 것을 컴퓨팅 가능하게 하기 위한 구문 분석기를 설계한 것을 기술 하였다.

2.1 mdl 파일 구조

mdl은 simulink로 만든 모델을 저장했을 때에 나오는 파일의 확장자이다. mdl은 simulink로 만드는 모델과 그 안의 sub modeling파트인 state flow chart 로 나뉜다. 두 부분은 gui툴 상에서의 모습 뿐만 아니라 mdl상에서의 자료구조도 완전히 다르다. 예를 들면 state flow chart의 요소들은 모두 id가 있는 반면, simulink의 block에는 id가 존재하지 않는다. 앞으로 본 논문에서 이 두 모듈을 구분지을 것이다.[2]

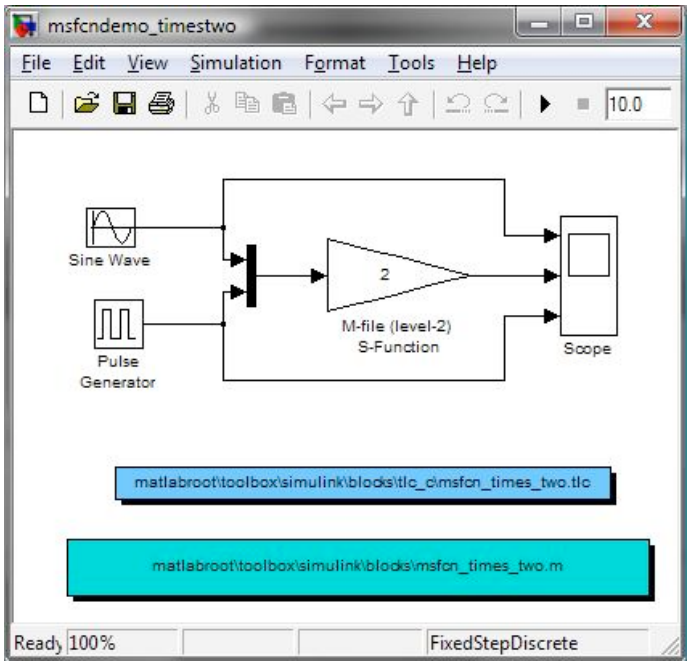

그림 1 Simulink model 예제

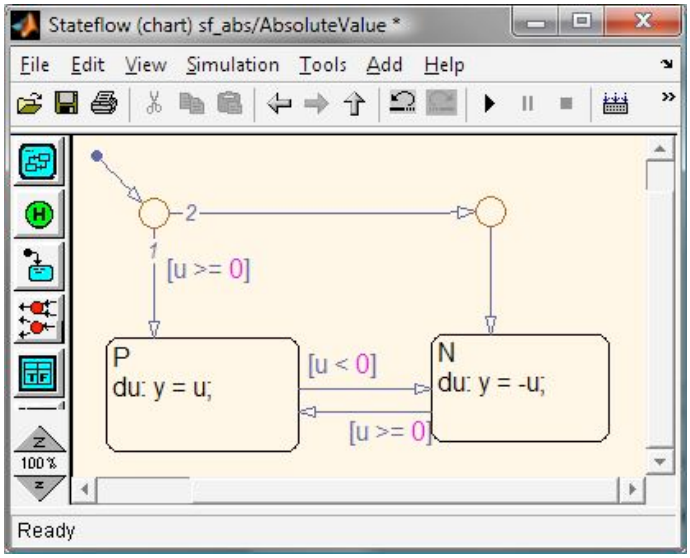

그림 2 Stateflow chart model 예제

mdl의 문법은 기본적으로 key/value 쌍의 집합으로 이루어지고 이러한 집합을 묶을 수 있는 class개념의 문법이 계층적으로 적용된다. key와 value는 1개 이상의 white space로 이루어져있다. value는 여러 타입이 있을 수 있는데, integer, floating point, string, array등이 있을 수 있다. 마치 JSON의 문법과 비슷하다. 예를 들자면 다음과 같다.

```
... 전략
State {
        id              3
        labelString     “Off”
        position        [10, 20, 30.3, 25.7]
        chart           2
        treeNode        [6 0 0 4]
        type            OR_STATE
}
…후략
```

코드 1 mdl 파일 예제

위 코드는 실제 mdl 파일에서 따온 State부분의 코드이다. id나 chart등은 integer 자료형을 사용하고 position이나 treeNode는 array 자료형을 사용한다. id이하의 key/value 쌍은 모든 State라는 이름으로 묶여있고, brace로 묶여있는 것들 또한 Mixed value로 보면 그룹도 key/value로 이루어진 요소로 바라볼 수도 있다.

여기서 주목할 것은 id와 chart, treeNode이다. 이 숫자들은 State flow chart를 구조화 하는데에 중요한 역할을 한다. State flow chart는 데이터를 tree형태로 저장하는데, 그 tree의 각 노드가 State, Transition등의 요소들이고, 노드들을 구분하기 위해 id를 사용한다. chart는 곧바로 tree의 root로 갈 수 있는 포인터가 되고, treeNode는 parent node, first Child node, next sibling node, prev sibling node가 차례로 기술 되어있다.

2.2 구문 분석기

Matlab에 의해서 만들어진 파일을 분석하기 위해선 그 파일을 파싱해서 파일의 각 요소를 분리하는 작업이 필요하다. 파싱을 할때는 가장 널리 사용되는 LR파서인 lex와 yacc을 사용하였다. c#용 파서로는 Queensland Univ의 gppg[1]를 선택했다. 문서화 지원이나 사용방법이 편리한것이 그 이유가 되겠다. gppg는 유닉스용으로 나와있는 lex와 yacc과같은 문법을 사용하면서 그에 c#을 위한 기능을 더한 LALR파서 생성기이다. 실험적으로 알아낸 mdl파일의 BNF는 다음과 같다.

```
<Object>
   ::= <Object name> <Object body>
<Object name>
   ::= <string> | <ID>
<Object body>
   ::= "{" <Object list> "}" | <Value>
<Object list>
   ::= [ <Object list> <EOL> ] <Object>
<Value>
   ::= <Array> | <String> | <Integer> | <float> | <ID>
<ID>
   ::= "$" <String>
<Array>
   ::= "[" <Value list> "]"
```

코드 2 mdl 파일의 BNF

위 BNF는 C언어의 문법적 구조를 배경지식으로 알고 있다는 가정하에 만들어진 약식 구조이다. 생략된 설명은 string, integer, float, value list에 대한 내용이다. value list를 제외한 모든 데이터 타입은 C 언어의 데이터 타입 개념과 같으며 value list는 white space또는 ';'를 구분자로 한 <Value>의 나열이다.

3. 파싱 및 구조화

C#을 이용한 mdl파일의 구조화는 다음과 같은 과정을 거친다.

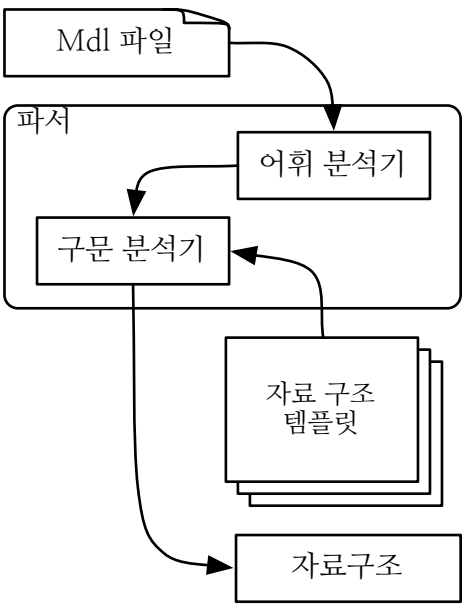

그림 3 파일 구조화 시스템의 구조

전체 모듈의 입력은 파일이며 출력은 c#으로 만들어진 구조체가 될수도 있고 xml파일이 될수도 있다. 출력부분은 얼마든지 다른 구조로 변환이 가능하나 본 논문에서는 C# 구조체가 주된 논제이므로 그것만 다루도록 한다.

파일을 입력받으면 gppg로 만든 lex와 yacc이 문법에 따라 mdl파일을 파싱하여 구조화 모듈로 토큰을 넘긴다. 토큰을 받으면 문맥에 따라 자료구조 템플릿, 이 경우에는 C#의 class를 이용해 데이터를 구성한다. 그리고 해당 구조체를 사용할 개발자나 연구원은 이 모듈을 사용하여 mdl의 모든 요소들을 접근할 수 있다.

어휘 분석과 구문 분석은 자세한 BNF가 나와있는 문서를 찾지 못해 여러 파일들을 실험적으로 파싱하면서 규칙들을 찾아냈다. 따라서 아직 처리하지 못하는 문법요소가 남아있을 가능성이 있다. 지금까지 약 20개가 넘는 충분히 복잡한 모델을 완전히 파싱에 성공했다.

mdl파일을 파싱했을 때 발생하는 토큰을 구문 분석이 내부에서 받아, 템플릿을 이용해 자료구조를 만든다. 자료구조를 설명하기 전에 템플릿, 즉 c#의 class구조부터 설명한다.

Block

simulink의 각 Block을 추상화했다. 많은 종류의 Block이 Block 클래스로 표현이 가능하다. 내부에 다음 block, 이전 block을 추적할 수 있는 기능이 있으며, block의 작동이나 내부 상태들은 외부에서 injection을 통해 조립이 가능하게 되어있어서 종류에 따라 다른 동작이 가능하다.

Line

simulink의 Block을 연결하는 선들을 추상화한다. brach되는 선들은 전부 여러 개의 Line으로 변환된다. 그 외에는 simulink의 line과 같은 역할을 한다.

State

simulink에 sub block으로 제공되는 state flow chart프로그램의 요소인 State를 추상화한다. 이름을 반드시 가지고 있고, 변수(Data)를 가질 수 있으며, enter, during, exit등의 이벤트를 감지하여 사용자가 입력한 코드를 실행시킬 수 있다. matlab은 simulink를 실행 시킬 때 이 코드들을 c코드로 변환해서 실행시킨다. 이를 위해 mdl파일 파싱 이외의 또다른 c스타일의 표현식을 파싱해야했다. 이 표현식은 이미 많은 곳에서 다루고 있으므로 여기서는 다루지 않는다.

Junction

State와 비슷한 요소이다. State에서 사용자가 입력할 수 있는 코드를 제외한 요소라고 할 수 있다.

Transition

State와 Junction의 Activation을 전달하는 통로이다. 각 Transition마다 Transition condition, Transition Action등, transition이 일어나기 위한 조건을 기술하기도 하고, transition이 일어날 때에 실행될 코드를 기술 하기도 한다. 이 코드는 위 State의 코드와 같은 문법을 가지고 있다.

Event

State flow chart의 event를 추상화한다. event는 특별히 내부구조가 복잡하지 않다. 이름을 가지고 있으며, 특정 state의 내부에 존재하는 scope를 가지고 있다.

Data

simulink의 각종 변수를 추상화한다. 여러 타입의 데이터를 담을 수 있다는 것 말고는 Event와 다를것이 없다.

2.1절에서 언급했듯이 Stateflow chart와는 달리 Simulink의 요소들에는 id가 없다. 하지만 관리의 일관성과 사용의 편리성을 위해 Block과 Line에도 id를 추가했다. Block과 Line의 id는 실제하지 않고, Stateflow chart의 id와 겹치면 안되기 때문에 Stateflow chart의 id map의 최대값을 알아낸 다음 그 이후의 id를 할당하였다.

본 모듈에서 mdl을 변환해 만든 기본적인 자료구조는 다음과 같다.

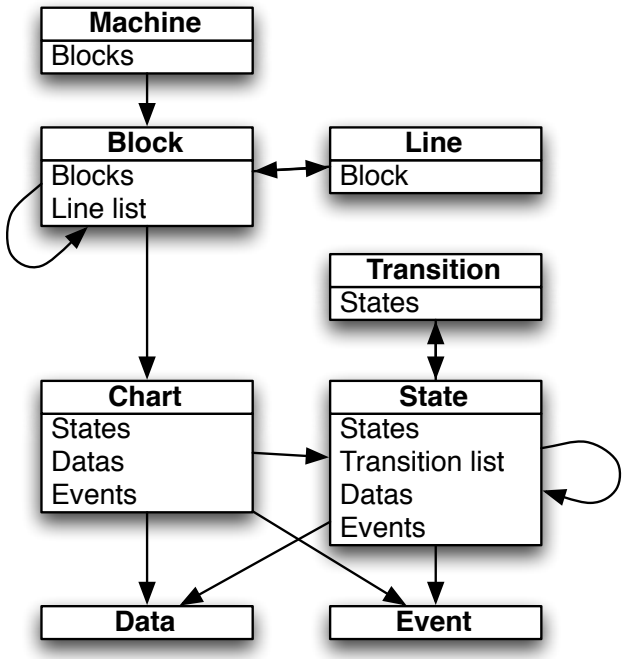

그림 4 만들어진 mdl 파일의 c# 자료구조 (object diagram)

기본적으로 각 객체들은 mdl 파일의 논리적 구조의 관점에서 만들어져 Tree형태를 띈다. 하나의 파일에 대응되는 Machine객체가 있고, 그 안에 많은 Block들이 포함된다. 각 Block은 그 Block의 id를 key로 하는 map에 존재하게 되어 검색시 이론적으로 O(1)의 속도가 걸리게 된다. 그리고 각 Block은 그 Block의 in-degree와 out-degree를 line객체로 소유하게 되고, Block은 계층적으로 존재할 수 있기 때문에 Block또한 다른 Block을 포함한다.

Block들 중 SubSystem이라는 타입을 가진 Block이 다른 Block을 포함하는 Block인데, 그 중 단순한 Block이 아닌 State flow chart를 포함할 수 있는 기능이 있다. 따라서 특정 타입의 Block객체는 State flow chart기능을 가지고 따라서 Chart객체를 가져야만 한다. 프로그램을 사용하는 입장에서는 그 Block은 Block이라는 이름없이 그냥 Chart로 추상화되어 사용할 수 있지만, 자료구조 사용의 일관성을 위해 여기에서는 분리했다.

Chart객체는 Machine처럼 하나의 State flow chart를 나타내고 이 안에 많은 State, Transition등의 객체가 포함되게 된다. Chart객체는 State와 Chart scope에서 선언된 Data와 Event를 가질 수 있다. 그리고 State또한 State scope에서 선언된 Data와 Event를 포함한다. 그리고 State도 Block과 같이 State를 연결해주는 Transition과 계층적인 구조를 위해 또 다른 State들을 가질 수 있게 했다. 이렇게 만들어진 객체의 네트워크를 일반화시킨 그림이 그림 4이다.

Action Code들

State와 Transition에는 사용자가 정의한 코드가 삽입 될 수 있다. State의 경우에는 Entry event, During Event, Exit Event가 일어 났을 때, 수행할 코드를 C 스타일로 설정 할 수 있고, Transition의 경우에는 transition이 일어날 조건이 되는 Event, Condition 부분과 transition이 일어날 때 실행되는 Condition action, Transition action부분을 C 스타일로 프로그래밍 할 수 있다. 이 부분들의 파싱은 변수 선언을 Data와 Event생성 메뉴에서 해야 한다는 것 말고는 C의 Statements 파싱과 크게 다를 것이 없다. 따라서 앞에서 말했듯이, 이 부분의 설명은 생략하도록 한다.

간단한 파싱 트리를 이용해 action code는 실행 가능한 상태가 되어있다. 이를 이용해 만들어진 자료구조를 직접 실행하여 Input에 대한 기대되는 output을 살펴볼 수 있다.

4. 개선 가능성

현재 만들어진 자료구조는 mdl 파일을 자료구조로 옮기고 각 요소들을 탐색하는 기능을 주로 가지고 있다. 사용자는 ID를 통해서 각 요소들을 한번에 접근할 수 있고, 그 요소와 직접적으로 이어진 다른 요소들을 손쉽게 접근할 수 있게 되었다.

하지만 test case를 만들기에는 아직 부족하다. 부족한 것을 크게 나누자면 기능상의 문제와 성능상의 문제로 나눌 수 있다.

실제로 자료구조를 사용을 해 본 결과, simulink의 Block들이 타입마다 매우 다른 행동을 취하고 있고, State flow chart의 transition의 조건 검사 순서또한 필요했다. 다시 말해서, '구조' 자체는 그다지 틀렸다고 할 만한 것이 없지만, 실행의 sementic의 연구가 제대로 되지 않았다고 할 수 있다.

그리고 성능상의 문제는 첫째로 C#이라는 것을 이유로 들 수 있다. C#이 Native code에 비해 90%이상의 퍼포먼스가 나온다고 하더라도 그 추상화 레벨때문에 콜 스택이 깊고, 하나의 행동을 하기 위해 하는 일이 매우 많은 듯 하다. 따라서 이는 multi thread 프로그램으로 다시 제작하거나, 사용상/개발상의 이점을 버리더라도 c나 c++을 이용해서 만드는 방법이 있을 것이다. c#을 이용해서 한번 제작하였으므로 그것을 기초로 다른 언어로 포팅 하는 작업은 작업의 양이 많을 지언정, 논리적으로 다시 생각하는 일은 많이 줄어들 것이라 생각한다.

5. 참고 문헌

[1] John Gough, Wayne Kelly(2009) The GPPG Parser Generator. retrieved April 2, 2009, http://plas.fit.qut.edu.au/gppg/files/gppg.pdf

[2] MathWorks(2010), Simulink - Documentation, www.mathworks.com/access/helpdesk/help/toolbox/simulink/