모바일 IPv6의 Alloy 모델 해석

박승순[0] 배민오 김영인
동덕여자대학교 전자계산학과
{sspark[0], bai, yikim}@dongduk.ac.kr

Interpretation of Alloy Model for Mobile IPv6

Seung-Soon Park[0] Mino Bai Young-Ihn Kim
Dongduk Women's University

요 약

본 논문에서는 Mobile IPv6에 사이클이 있음을 Alloy로 명세(specification)하고 Alloy 분석기로 검증(verification)한 것을 해석해 보겠다. Acyclicity를 만족하지 못함으로 해서 모바일호스트는 이동을 하지 못하고 계속 사이클을 돌게 된다. Alloy는 Rational의 UML과 같은 객체모델링 언어인데, 일차논리와 집합에 기반을 둔 Z에서부터 파생되었다. Alloy는 작은 모델들을 위한 명세 언어로, Alloy 모델은 그래픽과 텍스트를 모두 지원한다. Alloy로 명세 된 것을 쉽게 분석 할 수 있는 검증도구로 Alloy 분석기가 있는데, 이러한 도구를 이용함으로써 손쉽고 빠르게 검증을 할 수 있다.

1. 서 론

많이 사용되는 정형명세 언어인 Z는 일차논리와 집합론과 같은 수학적 이론을 기반으로 한다. 이러한 Z의 부분집합으로 NP라는 명세 언어가 있으나 자주 사용되는 객체지향 모델(Object-oriented Model)에는 적합하지가 않았다. 그래서 Z의 일차 논리와 집합론을 기반으로 하고, 객체지향모델의 핵심적인 사항들을 결합하여 만든 것이 Alloy이다. 본 논문에서는 1996년 IETF에 draft된 모바일 IPv6[1]에 사이클이 있음을 Alloy로 명세하고 검증한 것[2]을 해석해 보겠다. 2장에서는 모바일 IP에 대해서 설명하고, 3장에서는 Alloy에 대해서 간략히 소개를 하고, 4장에서는 Mobile IPv6의 Alloy 모델을 해석하고, 5장에서는 결론 및 향후과제를 제시한다.

2. Mobile IP

2.1 Mobile IP란?

휴대용 컴퓨터나 PDA와 같은 이동 단말들의 성능 향상과 무선 통신 기술의 발전등으로 이동 컴퓨터의 사용 요구가 점점 늘어나고 있다. 기존의 IP는 호스트의 이동성을 지원하지 않고 있으나 IPng(IP Next generation)라고도 불리는 IPv6는 이동성을 지원하는데, 모바일 IPv6는 단말의 이동을 검출하고, 이동후의 네트워크에서도 이동전의 네트워크와 동일하게 통신 할 수 있게 해주는 프로토콜이다. 네트워크는 하나 또는 그 이상의 서브넷으로 구성되어 있으며 네트워크 노드들은 이동이 가능한 호스트들과 서브넷 사이를 연결시켜주는 이동이 불가능한 라우터들로 구성되어 있다. 각각의 네트워크는 외부의 다른 네트워크와 연결되어 있으며 각각 호스트는 이동하기 전의 홈네트워크에 연결되어 있는 홈에이전트와 Home address가 있다. 호스트가 이동시 Home address만이 아니라 새로운 IP주소(care-of address)를 갖게되어 이동한 네트워크의 위치를 알 수 있다.

2.2 Mobile IPv6의 사이클

A 네트워크에 있는 호스트가 B네트워크로 이동하고 B네트워크에 있는 호스트가 C네트워크로 이동 그리고 C네트워크에 있는 호스트가 다시 A 네트워크로 이동하게 되면 이행관계(transitive relation)가 유도 될 수 있다. 이런 특성을 캐시 acyclicity라 하고, 자신에게 돌아가지 않는 것이 보장되어야만 한다. 새로운 네트워크로 호스트가 이동하면 이동된 호스트의 정보를 바로 전의 호스트가 있던 네트워크의 캐시로 보내게 되는데 이때도 메시지 acyclicity가 발생할 수 있으므로 보내는 메시지도 사이클의 형태가 없음을 보장해 주어야 한다

3. Alloy

객체모델링을 위한 Alloy는 작은 모델링들의 간단한 구조적인 특성을 기술하기 위한, 일차논리를 기반으로 설계된 모델링 언어이다. 또한 집합과 릴레이션을 기본으로 하는 formula들의 복잡한 제약조건(constraints) 들을 나타내는데 적합한 언어이다. Alloy 모델은 두 개의 부분으로 나뉘는데 하나는 그래픽 부분이고 또 하나는 그래픽 부분을 문자로 선언한 텍스트부분이다. Alloy는 텍스트와 그래픽을 모두 지원하므로 객체모델 뿐만 아니라 operation Model을 잘 지원해준다. Alloy는 Z[3]로부터 파생되었고, Alloy에 앞서 NP라는 명세 언어가 있었다. NP는 규모가 작은 사례연구(case study)에 적용하기 위한 명세 언어로 Z와 같은 엄격한 일차논리가 적용된다 또한 Z와 같이 스키마 구조를 가지고 있고, 각 스키마 구조를 결합해서 더 큰 스키마 구조를 갖는다. 그러나 많이 사용되는 객체모델링을 위한 언어는 아니므로, 객

본 연구는 한국과학재단 목적기초연구 (과제번호: R01-2000-00287) 지원으로 수행되었음

체모델링 언어를 위해 만든 것이 Alloy이다. Alloy 의미(semantic)의 기본은 집합과 관계를 기본으로 한 Z에 기초를 두었고, 다중성(multiplicity)와 변하기 쉬운 제약조건(constraint) 같은 기본구조 메카니즘은 객체모델링 언어에서 영향을 받았다. 그러나 Alloy는 클래스가 아니고 집합에 기반을 두었으므로 UML보다는 더 추상적이고 간단하다. Alloy로 명세한 객체 모델들을 분석하기 위한 도구로 Alloy 분석기가 있다. Alloy 분석기는 Alloy 모델의 구조를 이용해 명세 한 것을 분석해서 명세에 모순이 있는지, 주어진 요구사항을 만족하는지를 빠르고 쉽게 알아볼 수 있는 검증도구이다. 즉, Alloy 분석기는 function을 실행하거나 모델의 특성(property)을 체크해서 불변자(*invariant*)를 만족하는 상태(state)의 실례와 반례를 보여 줄 수 있다. Alloy 분석기의 기본 기능은 매우 간단하다. 제약조건(constraint)이 주어지면 그것을 검증해서 결과를 노드나 edge의 그래프 트리로서 보여주는 것이다. Alloy와 Alloy 분석기는 주로 추상화된 소프트웨어 디자인을 명세하고 검증한다.

Alloy 모델링 구조를 보면,

- Domain : 더 이상 분할 할 수 없는 집합을 나타낸다.
- State : domain의 기본적인 구성상태를 선언한다
- Invariants : 상태의 변하지 않는 특성(property)을 나타낸다
- Definitions : 편리하게 사용하기 위한 shortcut를 생성한다
- Assertions : 이전에 나왔던 식에서부터 다음에 오는 식들을 유도해서 property에 만족하는지를 체크한다.
- Condition : 모델이 constraint를 위반하지 않는 것을 증명할 수 있게 하기 위한 특성을 나타낸다

4. Mobile IPv6의 Alloy 모델 해석

각각의 호스트는 캐시를 가지고있고, 모바일 호스트에서 최근 메시지를 보내면 그것을 캐시에 저장한다. 모바일 호스트는 단지 한 개뿐이고 분리된 모바일 호스트의 행동들은 각각 독립적이라고 이 논문에서는 가정했다. 모바일 호스트가 새로운 목적지에 도착했을 때 update메시지를 호스트에 보내고, 이 메시지가 호스트에 도착하면, 호스트의 cache를 update 시킨다. 이 논문에서는 메시지가 호스트에 도착했을 때 캐시를 update 시키는 것으로 사이클 있는지를 검증해보았다.

```
domain {fixed Host, Msg, fixed Time}
state{   Home: Host!
  Caching: Host
  caches: Caching -> Host!
  cache_exp_time: Caching -> Time!
  to, from, where: Msg -> static Host!
  send_time, exp_time: Msg -> static Time!
  Clock: Time!
  before: static Time -> static Time  }
state Definitions {   Refing: Host
  msg_ref: Host -> Host
  combined_ref: Refing -> Host+   }
def  msg_ref {all h | h.msg_ref = h.~to.where}
def  combined_ref  {all  h  |  h.combined_ref  =  h.msg_ref  +
h.caches}
inv Structure {
①   no h | h in h.caches
②   all m1, m2 | m1.to = m2.to && m1.from = m2.from &&
      m1.send_time = m2.send_time -> m1 = m2
    all m | m.from != m.to
    all m | m.send_time in m.exp_time.before
③   all t | t.+before in t.before
    no t | t in t.before   }
op UpdateArrival (m: Msg!) {
  Clock' in m.exp_time.before
  m.to != Home
  m.send_time !in m.to.cache_exp_time.+before
  Clock in Clock'.before
  Home  = Home
  Msg' = Msg
  all h: Host - m.to | h.caches' in h.caches
  all h | h.cache_exp_time in Clock.before ->no h.caches'
  m.to.caches' = m.where     }
cond AcyclicCaches {no h | h in h.+caches}
cond AcyclicMsgs {no h | h in h.+msg_ref}
cond RefsReasonable {
  all h: Refing | Home in h.+combined_ref
  all m1, m2 | m1.send_time in m2.send_time.before -> m1.where
    !in m2.where.+combined_ref
  all m, h | m.send_time in h.cache_exp_time.before -> m.where
    !in h.caches.+combined_ref
  all m, h | h.cache_exp_time in m.send_time.before -> h !in
  m.where.+combined_ref  }
cond LocalKnowledge {  all h | h = Home -> no h.caches   }
assert A {   all m | UpdateArrival(m) && AcyclicCaches ->
AcyclicCaches'  }
assert B {   all m | UpdateArrival(m) && RefsReasonable &&
AcyclicCaches -> AcyclicCaches'   }
assert C {   all m | UpdateArrival(m) && RefsReasonable &&
LocalKnowledge && AcyclicCaches -> AcyclicCaches'  }
```

[그림1] 모바일 IPv6의 alloy 명세

1) Alloy는 집합[4]을 기본으로 한다. 도메인에서 fixed로 표시한 것은 모바일 호스트에 의해 Host가 고정되기 때문이다. domain {fixed Host, Msg, fixed Time}
다음과 같이 표시하면 Caching은 Host의 집합이고 Home은 Host 집합 중 하나에 해당된다. Msg가 Host로 전달이 될 때 static은 Msg가 정해지면 그 메시지에 대한 호스트가 1개로 고정되어 있음을 나타낸다.

```
Home: Host!
Caching: Host
to, from, where: Msg -> static Host!
```

2) 논리에서는 한정자가 필요 없다 그러나 한정자의 표현이 없으면 제약조건(constraint)을 읽거나 쓰는 데 매우 불편하다 한정자의 표현을 Alloy에서는 다음과 같이 나타낸다. 모든 메시지 m1, m2에서 두 메시지의 출발지와 도착지가 같고 보내는 시간이 같으면 같은 메시지라는 것을 다음과 같이 표시한다.

```
all m1, m2 | m1.to = m2.to && m1.from = m2.from    &&
  m1.send_time = m2.send_time -> m1 = m2
```

3) 이행적폐쇄(transitive closure)는 +로 표시한다. h의 모든 cache 안에 h가 있으면 안된다는 것을 다음과 같

이 나타낸다. no h | h in h.+caches

4) ~는 transpose를 의미하는데, 메시지를 받은 호스트에서 메시지를 보낸 호스트를 참조한다

```
all h | h.msg_ref = h.~to.where
```

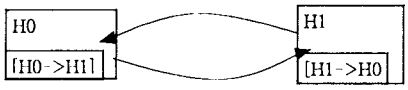
5) t.before를 t.+ before처럼 쓰겠다는 의미이다.

```
all t | t.+before in t.before
```

6) assert A를 실행해 보면, 다음과 같은 반례를 얻을 수 있다

```
Counterexample found:
Domains:
  Host = {H0,H1,H2}
  Msg = {M2}
  Time = {T0,T1,T2}
  Msg' = {M2}
Sets:
  Caching = {H0,H1}
  Clock = {T0}
  Home = {H2}
  Refing = {H0,H1}
  Caching' = {H0,H1}
  Clock' = {T1}
  Home' = {H2}
  Refing' = {H0,H1}
Relations:
  before = {T1 -> {T0}, T2 -> {T0,T1}}
  cache_exp_time = {H0 -> {T2}, H1 -> {T1}}
  caches = {H0 -> {H1}, H1 -> {H2}}
  combined_ref = {H0 -> {H1}, H1 -> {H0,H2}}
  exp_time = {M2 -> {T2}}
  from = {M2 -> {H0}}
  msg_ref = {H1 -> {H0}}
  send_time = {M2 -> {T1}}
  to = {M2 -> {H1}}
  where = {M2 -> {H0}}
  before' = {T1 -> {T0}, T2 -> {T0,T1}}
  cache_exp_time' = {H0 -> {T2}, H1 -> {T1}}
  caches' = {H0 -> {H1}, H1 -> {H0}}
  combined_ref' = {H0 -> {H1}, H1 -> {H0}}
  exp_time' = {M2 -> {T2}}
  from' = {M2 -> {H0}}
  msg_ref' = {H1 -> {H0}}
  send_time' = {M2 -> {T1}}
  to' = {M2 -> {H1}}
  where' = {M2 -> {H0}}
Parameters:
  m = {M0}
```

assert A의 반례를 간략히 살펴보자. 이 반례에서는 Alloy명세에 있는 3개의 invariant는 모두 만족하지만, 그림과 같이 사이클이 있음을 알게 된다.



[그림2] 캐시 사이클

① 자신의 캐시에 자신은 들어가면 안 된다

② 메시지 m1과 m2에서 두 메시지의 출발지와 도착지가 같고 보내는 시간이 같으면 같은 메시지이고, 메시지를 보내는 출발지와 받는 도착지가 같으면 안된다. (H0->H1) 메시지를 보내는 시간(T1)은 메시지의 소멸시간(T2)보다 먼저 와야한다.

③ T 이전의 시간에는 T가 포함되지 않아야 하고, T1 전에 T0가 오고 T2 전에 T0,T1이 오므로 시간이 T0<T1<T2의 순서가 된다.

7) assert B를 실행해 보면 반례가 나오지 않는데, 전에 있던 캐시의 내용이 없어지기(캐시소멸시간:T2) 전에 새로운 메시지가 오게 되면(보낸메시지시간:T1) 이 메시지가 전달하려는 내용(H0)이 캐시의 combined_ref 안에 (H1->{H0,H2}) 없어야 한다는 조건이 RefsReasonable 있으므로 assert A의 H1이 H0로 가지 못하게 되어 사이클이 없어지게 된다.

8) 이 모델은 assert B와 assert C가 똑같이 반례가 나오지 않으므로 RefsReasonable과 LocalKnowledge의 관계를 살펴보았다.

assert D { RefsReasonable && AcyclicCaches-> LocalKnowledge } 를 실행해보면 반례가 나오지 않는 것으로 보아 assert B와 assert C는 조건이 같음을 알 수 있다

5. 결론 및 향후과제

Alloy는 작은 릴레이션 범위에서 명세 되고 검증되는데 작은 범위에서 반례를 찾을 수 있다면 매우 신속하고 유용한 검증이 될 것이다. 그러나 반례를 찾을 수 없다면 더 큰 범위에서도 반례가 없다는 것을 꼭 보증해 주지는 않는다. 이런 제한에도 불구하고 사용하기가 쉽고 손으로 직접 명세하고 검증하는 것보다 훨씬 에러율이 낮고 명확하기 때문에 유용하다. Alloy 분석기로 검증해본 결과 서로 다른 assert를 실행 할 때 같은 모델에서 반례가 생성이 되지 않으므로 서로 비교 분석하기가 쉽지 않았다. 그리고 모바일 호스트가 여러 장소를 방문했을 때 방문한 시간 순서 즉, 메시지가 생성한 순서대로 시간에 맞게 명세 되었는지, 모바일 호스트가 이동 도중에 링크가 끊어졌다 다시 이동할 때 사이클은 어떻게 되는지 등의 상황이 명확하게 명세 되어 있지 않았다. 이런 점은 좀더 보완이 되어야 할 것이다.

5. 참고문헌

[1] David Johnson and Charles Perkins. "Mobility Support in IPv6," Mobile IP Working Group INTERNET-Draft, June 1996

[2] Daniel Jackson and Yuchung Ng and Jeannette Wing. A Nitpick Analysis of Mobile IPv6, CMU-CS-98-113. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, September 1997.

[3] Daniel Jackson and Ilya Shlyakhter and Manu Sridharan. A Micromodularity Mechanism. Proc. ACM SIGSOFT Conf. Foundations of Software Engineering/European Software Engineering Conference(FSE/ESEC '01),Vienna, September 2001.

[4] Daniel Jackson. Alloy: A Lightweight Object Modeling Notation. To appear, ACM Transactions on Software Engineering and Methodology, October 2001.