Applying Predicate Abstraction to Abstract State Machines

Alessandro Bianchi, Sebastiano Pizzutilo, Gennaro Vessio
Department of Informatics – University of Bari, Italy

EMMSAD'15

Overview

- Introduction
- Background
- Our proposal
- Two examples
- Conclusion

The Context

Formal methods for analyzing computationally interesting properties

- safety: deadlock-freedom, …
- liveness: starvation-freedom, …

Abstract State Machines (ASMs) are successfully used for this purpose

The Problem

ASMs are Turing-equivalent: their formal verification **cannot be fully automatized**

Model checking approaches to ASMs suffer from:

- The loss of expressive power
- The difficulty in using temporal logics

The Idea

Applying **predicate abstraction** to ASMs

Traditionally, it approximates program states into a finite number of predicates

Two advantages

A Note

Predicates over the states provide an abstraction, but we use them only for **supporting static** ASM verification

Abstract State Machines

ASMs are finite sets of *rules* of the form:

if *condition* ***then*** *updates*

which transform *abstract* states

An ASM state is an algebraic structure: pairs of function names together with arguments values are called *locations*

A Question

Why predicates over ASM states?

An algebraic structure can model any object of arbitrary complexity: understanding the semantics of the model is **hard**

We need an **abstraction framework** capable of capturing this semantics

The Answer

Using **predicates over ASM states** allows a modeler to represent the **local** properties of each state

Predicates over ASM states are first-order formulas defined over ASM locations: in each state they can hold or not

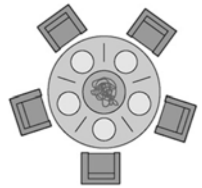
Global properties of the ASM model can then be verified by composing these local properties

Dining Philosophers

Five philosophers are sitting around a table with a bowl of spaghetti in the middle

For them life consists of two moments:

- thinking
- eating
  (with two forks)



Dining Philosophers

Each philosopher is modeled by an ASM:

PhilosopherProgram(*pi*) =
if *owner*(*rightFork*(**self**)) = *undef* ∧ *ower*(*leftFork*(**self**)) = *undef* **then** {
  owner(*rightFork*(**self**)) := **self**
  owner(*leftFork*(**self**)) := **self**
}
if *owner*(*rightFork*(**self**)) = **self** ∧ *owner*(*leftFork*(**self**)) = **self then** {
  owner(*rightFork*(**self**)) := *undef*
  owner(*leftFork*(**self**)) := *undef*
}

Dining Philosophers

Each ASM can traverse different states

They are characterized by the following predicates:

- *thinking*: ¬(*owner*(*rightFork*(**self**)) = **self** ∨ *owner*(*leftFork*(**self**)) = **self**)
- *eating*: *owner*(*rightFork*(**self**)) = **self** ∧ *owner*(*leftFork*(**self**)) = **self**

Introduction
Background
Our proposal
Two examples
Conclusion

Dining Philosophers

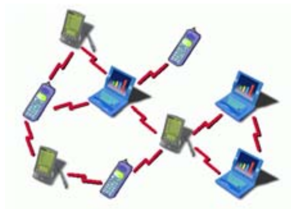
Starvation: There is at least one ASM that cyclically returns to states characterized by the same predicate expressing the waiting (*thinking*)

Deadlock: All ASMs are in a state in which the predicate expressing the waiting holds

Introduction
Background
Our proposal
Two examples
Conclusion

A MANET Routing Protocol

A MANET is a wireless network supporting communications among nomadic hosts in absence of a fixed infrastructure



Introduction
Background
Our proposal
Two examples
Conclusion

A MANET Routing Protocol

Each host is modeled by an ASM:

```
HostProgram(hi) =
 if ¬isEmpty(requests(self)) then {
   RREQ = top(requests(self))
   nextHop = sender of
   top(requests(self))
   updateRoutingTable(self, RREQ)
   receivedRREQ(self, dest) := true
   Router(RREQ, nextHop)
 }
…
```


Introduction
Background
Our proposal
Two examples
Conclusion

A MANET Routing Protocol

```
 …
if wishToInitiate(self, dest) = true then
 Initiator(dest)
if ¬isEmpty(replies(self)) {
 RREP = top(replies(self))
 if RREP.init ≠ self then {
  nextHop = select c.nextHop ∈
  hostsInRT(routingTable(self))
  with RREP.init = c.dest
  updateRoutingTable(self, RREP)
  UnicastRREP(RREP, nextHop)
  dequeue RREP from replies(self)
 }
}
```

Introduction
Background
Our proposal
Two examples
Conclusion

A MANET Routing Protocol

Each ASM can traverse different states

They are characterized by the following predicates:

- idle: *wishToInitiate*(**self**, *dest*) = *false* ∧ *receivedRREQ*(**self**, *dest*) = *false* ∧ *isEmpty*(*replies*(**self**)) = *true*, ∀ *dest* ∈ *hosts*
- router: *receivedRREQ*(**self**, *dest*) = *true*
- initiator: *wishToInitiate*(**self**, *dest*) = *true*
- forwarding: *isEmpty*(*replies*(**self**)) = *false*

Introduction
Background
Our proposal
Two examples
Conclusion

A MANET Routing Protocol

In a host several computational activities are executed **in parallel**

The simultaneous fulfillment of different predicates captures this beahvior

Introduction
Background
Our proposal
Two examples
Conclusion

Lessons Learned

A given predicate can hold in several states

In a given state several predicates can hold

Introduction
Background
Our proposal
Two examples
Conclusion

Conclusion & Future Work

Predicates over the states can support the static verification of ASM models by **overcoming** the main limitations of model checking approaches

Specific features of predicate abstraction with respect to the **different kinds** of properties need to be investigated