Modeling and Verification of MPI based distributed software (using ASMs)

Igor Grudenić, Nikola Bogunović

Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of Electrical Engineering and Computing, Zagreb, Croatia
http://www.fer.hr

Presentation overview

- **Motivation**
- **Formal verification – three interesting properties**
- **Introduction to ASMs (really short one)**
- **Using ASMs to model basic MPI communication**
- **Mutual exclusion case study**
- **Conclusion & Future work**

Motivation

- **Complex distributed systems are hard for design and analysis**
- **Typical problems**

	Synchronous communication	Asynchronous communication
Parallelism (scalability)	HARD	EASY?
Deadlock	POSSIBLE	NOT POSSIBLE
System complexity	MODERATE	HUGE

Formal verification

- **Checks whether system conforms to specification**
- **Theorem proving & model checking**
- **Model checking with SPIN (Siegel 2005)**
  - theorems and manipulation to avoid state explosion
  - three interesting properties for model checking and distributed sytems
    - Safety (e.g. deadlock absence)
    - Liveness
    - Fairness

Introduction to ASM (abstract state machines)

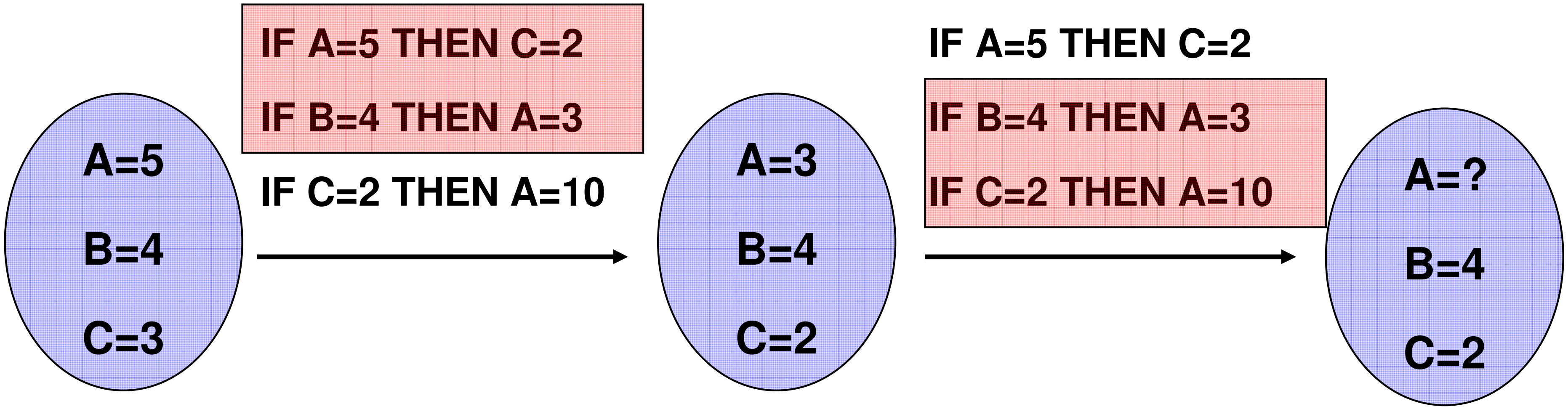
- **State machines – known concept**
- **Abstract state machines – similar to state machine but**
  - You need to define only the initial state and give the 'recipe' how to evolve (no enumeration)

  - Math:
    - Initial state (and every other state it evolved to) is a static algebra (set X, signature $\Sigma=\{f \mid X^r \rightarrow X\}$
    - State evolution:

```
If Condition then Update
```

`Update:` $f(t_1,\ldots,t_2)=t$, $(f\in\Sigma \quad t_x,t\in X)$

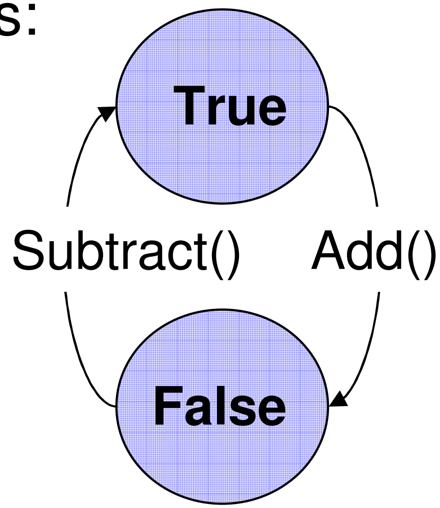
Introduction to ASM (example)



Using ASMs to model basic MPI communication

- **ASML (abstract state machine language) – object oriented**
- **Model exploration – Spec Explorer**
- **How to define MPI processes ?**
  - every process is instance of the special *process class*
  - Every method in the process class is considered as atomic action
  - Example: process that executes sequence of actions: Add(),Subtract(),Add(),...

```
class process()
      var state as Boolean=true

Add()
  require state=true
  state:=false

Subtract()
  require state=false
  state:=true
```



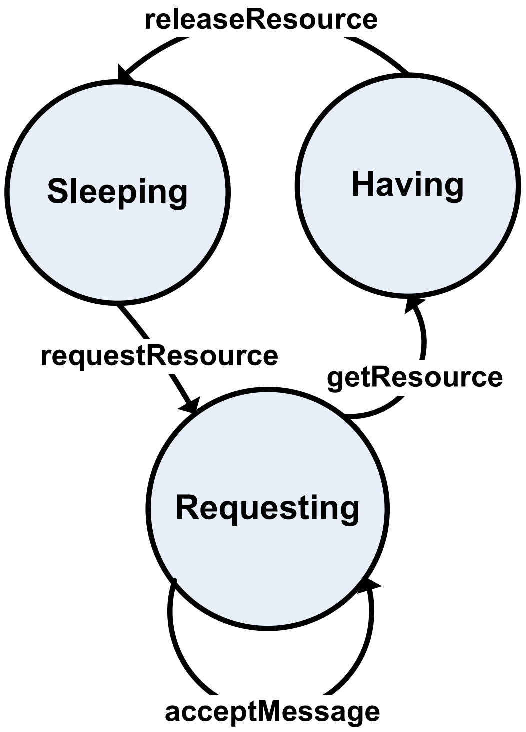
Using ASMs to model basic MPI communication

- **Encoding MPI communication**

<table>
<tr><th></th><th>BLOCKING</th><th>NON BLOCKING</th></tr>
<tr><td>SEND</td><td>Blocking Send()
  require state=DESIRED STATE
  receiver.messageQueue+=[Message]
  state=MESSAGE SENT</td><td>Make the same transfer as blocking but only when MPI_WAIT is posted?</td></tr>
<tr><td>RECEIVE</td><td colspan="2">AnnounceMessageReceive()
   require state=DESIRED STATE
   me.announces+={Announce()}
   state:=DESIRED STATE + RECEIVE ANNOUNCED
ReceiveMessage()
   require messageQueue contains message
   (BL)require DESIRED STATE + RECEIVE ANNOUNCED
   applicationBuffer=getMessage(messageQueue)
   state:=MESSAGE RECEIVED</td></tr>
</table>

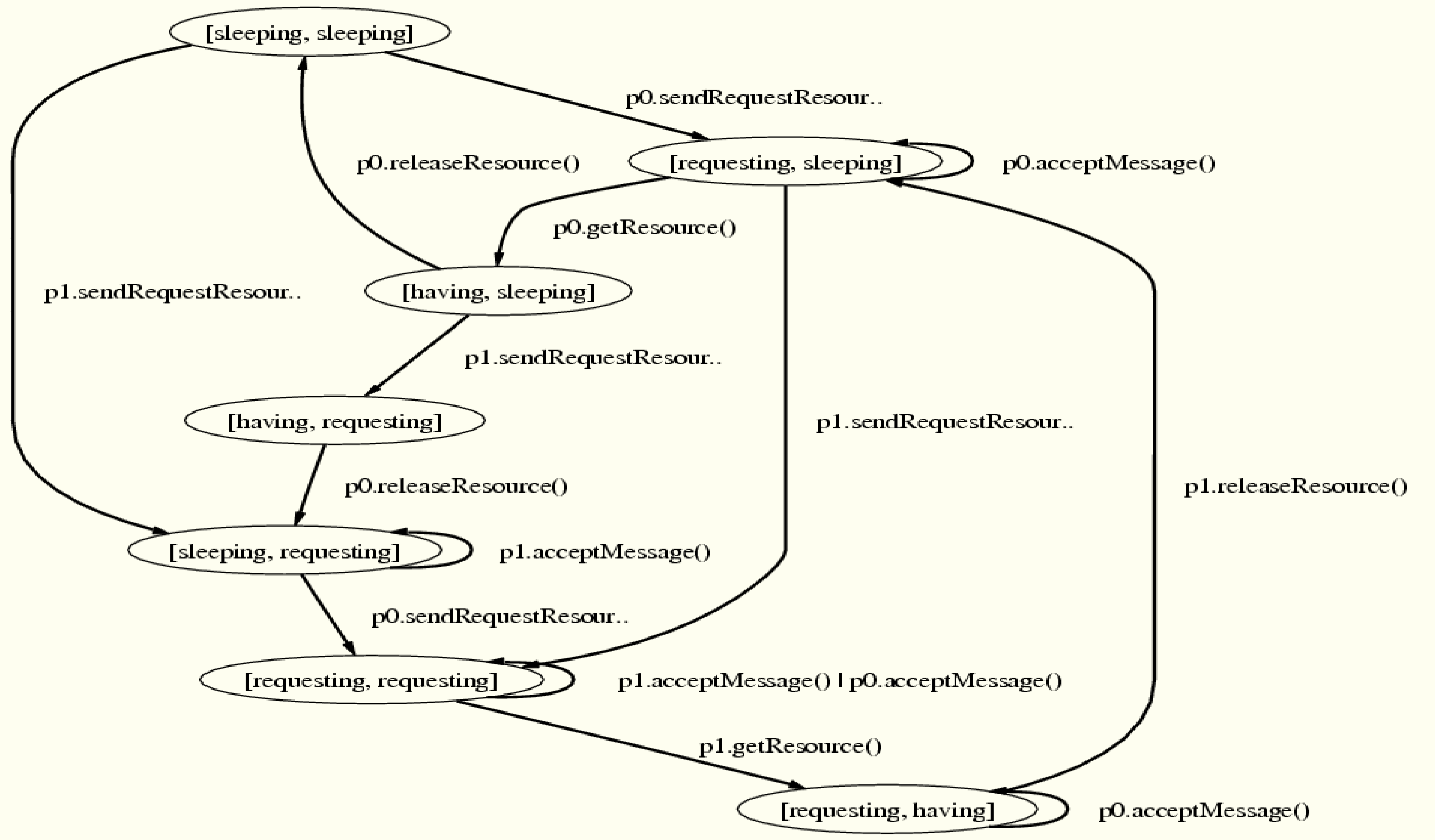
Mutual exclusion – case study

- **Lamport 1978, only non-blocking communication**
- **Three properties (safety, liveness, fairness)**



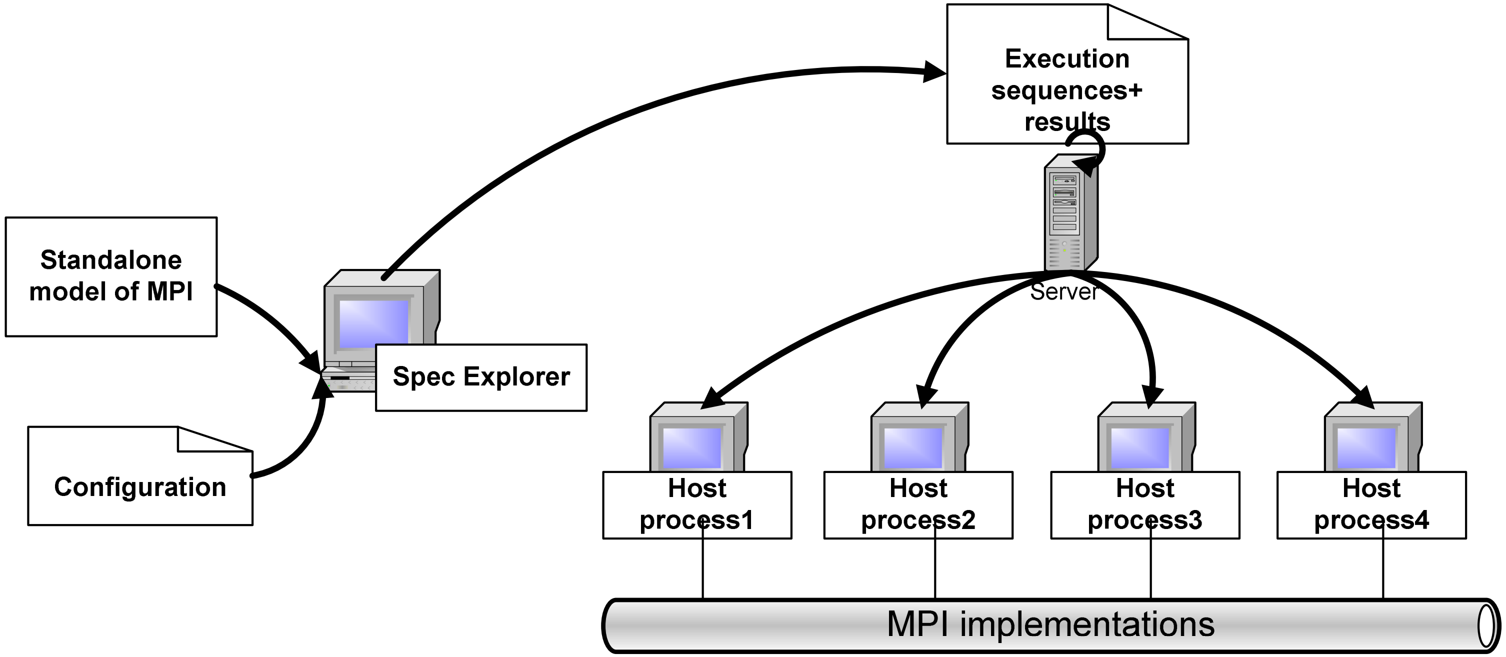

```
requestResource()
 require state=tState.sleeping
 state:=tState.requesting
 foreach iProcess in processes
                  where iProcess<>me
  iProcess.messageQueue+=[NewRequest]
  requestQueue+=[NewRequest]
  clock+=1,nOfReceivedAck:=0

acceptMessage()
 require Size(messageQueue)>0
 require state=tState.requesting
 messageQueue:=Tail(messageQueue)
 clock:=max(messageQueue(0).clock+1,clock+1)
 match messageQueue(0).messageType
  REQ: requestQueue+={messageQueue(0)}
       sender.messageQueue+=[NewAcknowledge]
  REL: requestQueue-=request(sender)
  ACK: nOfReceivedAck+=1
```

Mutual exclusion – exploration & verification



Future Work

- Standalone model of MPI (only basic communication)
- Configurations (made several by hand now)
- Host processes (support execution of MPI functions supported by the model)



Conclusion

- **ASMs are powerfull for capturing requirements**
- **ASMs can give an insight to a distributed system**
- **Some basic (but important) properties can be verified using available tools**
- **Questions?**