An Infrastructure for Implementing Compilers for Concurrent Abstract State Machine Languages

Kristian Magnani

Mariza A. S. Bigonha

Roberto S. Bigonha

Fabíola F. Oliveira

Vladimir O. Di Iorio

kristian@ufmg.br

August 2005

Introduction

- Abstract State Machines (ASM)
  - formal semantic method created by Yuri Gurevich
  - provide operational semantics for algorithms
  - used in:
    - Semantics of programming languages
    - Distributed systems
    - Architectures (hardware and software)
    - etc

Introduction

This work:

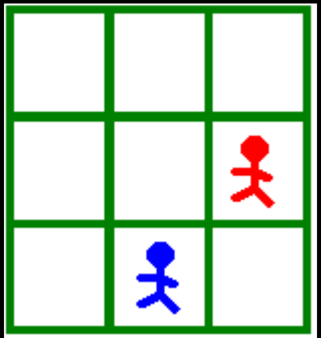
- describes a new model for specifying concurrent systems based on ASM
- proposes an infrastructure, called MIR, to implement ASM-like languages

Abstract State Machines (ASM)

- Machines whose states are algebraic structures, represented by ***functions***
- Basic transition rules:
  - function update
  - conditional constructor
  - block constructor (parallel execution)
- Runs:
  - sequences of states
  - next state generated by the application of the transition rule over the previous state

Abstract State Machines (ASM)

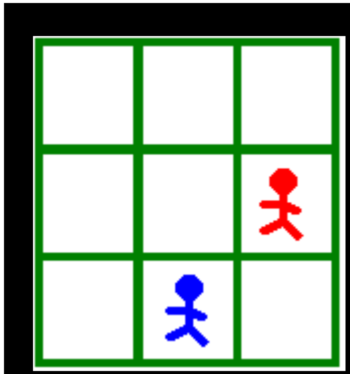
Example:



State:

```
lin(RED)  = 2
col(RED)  = 3
lin(BLUE)  = 3
col(BLUE)  = 2
```

Abstract State Machines (ASM)

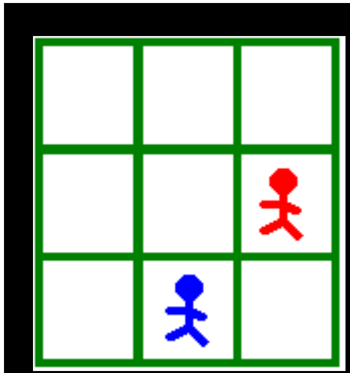


```
lin(RED) = 2
col(RED) = 3
lin(BLUE) = 3
col(BLUE) = 2
```

Transition rule:

```
if col(RED) > 1 then
   col(RED) := col(RED) - 1
end
```

Abstract State Machines (ASM)

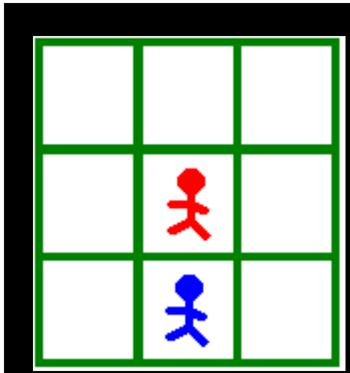


```
lin(RED) = 2
col(RED) = 3
lin(BLUE) = 3
col(BLUE) = 2
```

```
if col(RED) > 1 then
   col(RED) := col(RED) - 1
end
```

Run: …

Abstract State Machines (ASM)



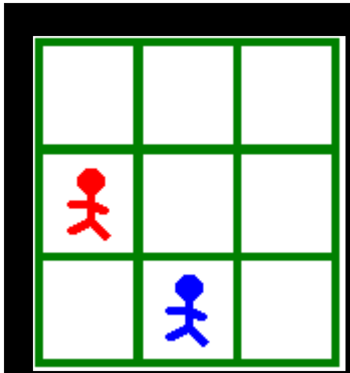
```
lin(RED) = 2
col(RED) = 2
lin(BLUE) = 3
col(BLUE) = 2
```

```
if col(RED) > 1 then
   col(RED) := col(RED) - 1
end
```

Run:

```
   Step 1 …
```

Abstract State Machines (ASM)



```
lin(RED)  = 2
col(RED)  = 1
lin(BLUE) = 3
col(BLUE) = 2
```

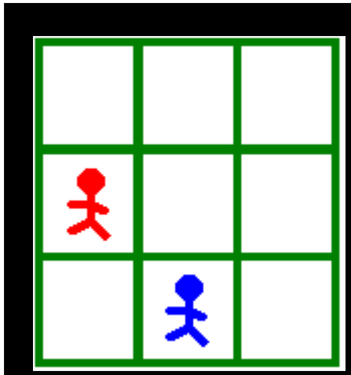
```
if col(RED) > 1 then
  col(RED) := col(RED) - 1
end
```

Run:

Step 1

Step 2 …

Abstract State Machines (ASM)



```
lin(RED) = 2
col(RED) = 1
lin(BLUE) = 3
col(BLUE) = 2
```

```
if col(RED) > 1 then
  col(RED) := col(RED) - 1
end
```

Run:

```
  Step 1
  Step 2
  Step 3 …
```

Abstract State Machines (ASM)

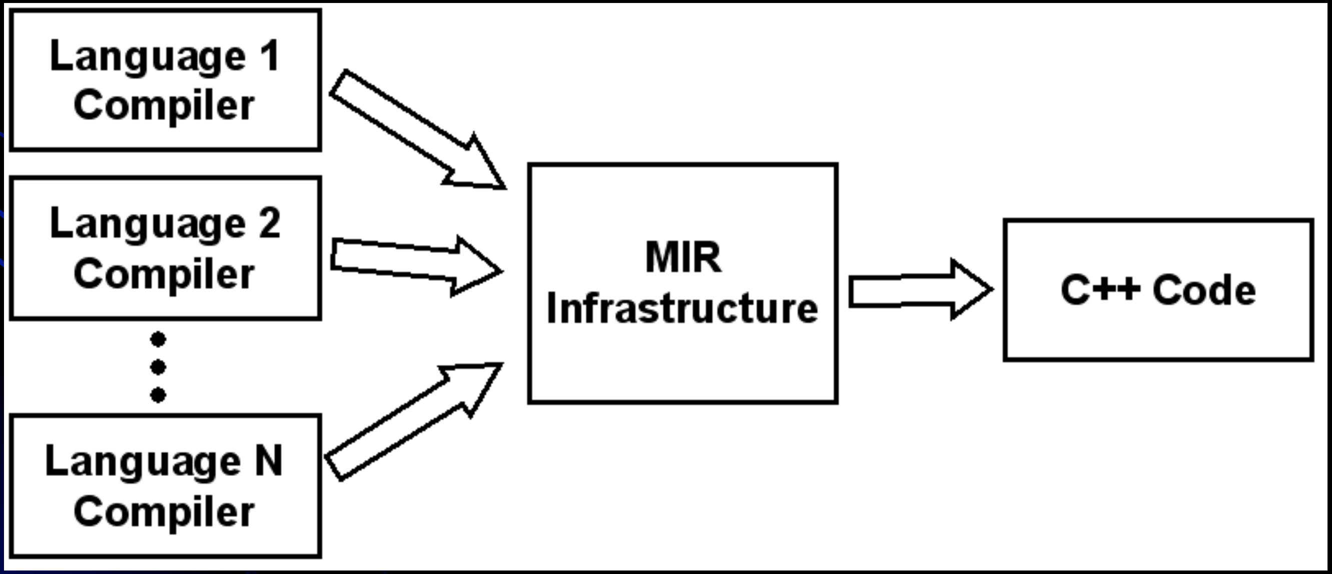
- Advantages:
  - Precision
  - Simplicity
  - Executability
  - Scalability
  - Generality

MIR

- Infrastructure for ASM implementation
- Implements a concurrency model based on Lamport's concept of delayed knowledge
- Designed to allow optimizations to be performed over ASM specifications

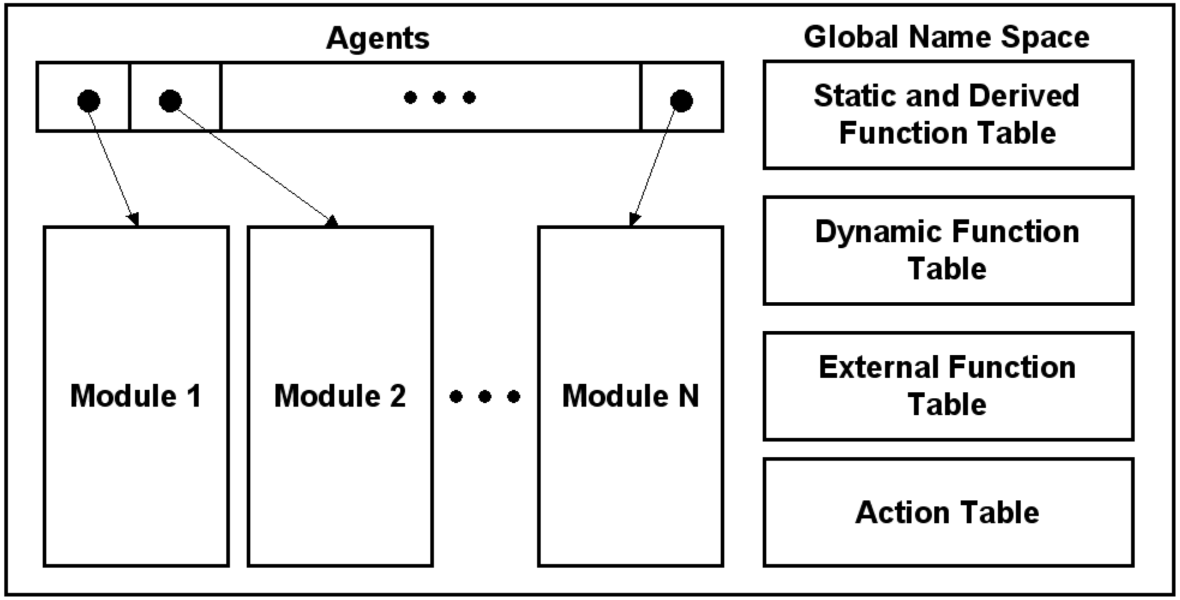
The MIR Architecture

- Motivation:
  - general infrastructure used as the basis of compilers aiming at different ASM oriented languages;



Agents, Modules and Other Elements

- A MIR specification:
  - a set of agents, each of them of a given type
  - a common Global Name Space, which is accessible by each agent.



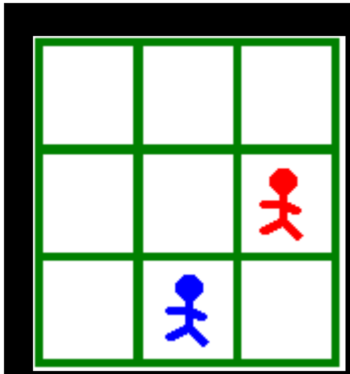
The MIR Approach for Concurrency

- An agent of MIR :
  - Autonomy: there is a transition rule associated with each agent, giving it an autonomous and independent behaviour from other agents;
  - Situatedness: every agent is immersed in a common global context where they can interact among each other, which are referenced by their names or by the special word self;
  - Proactivity: an agent has the freedom to call actions over other agents.

The MIR Approach for Concurrency

- Delayed knowledge (Lamport):
  - agents may store copies of the global state which may not always be kept up to date.
- Systems with agents of different speed of execution
- Unpredictable delay between the moment some changes are performed by one agent and the moment when these changes are perceived by another agent

The MIR Approach for Concurrency



```
lin(RED) = 2
col(RED) = 3
lin(BLUE) = 3
col(BLUE) = 2
```

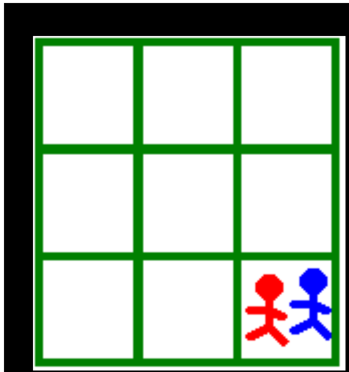
Rule for RED:

```
if isFree (lin(self)+1,
      col(self)) then
   lin(self) := lin(self)+1
end
```

Rule for BLUE:

```
if isFree (lin(self),
      col(self)+1) then
   col(self) := col(self)+1
End
```

The MIR Approach for Concurrency



```
lin(RED)  = 3
col(RED)  = 3
lin(BLUE)  = 3
col(BLUE)  = 3
```

Rule for RED:

```
if isFree (lin(self)+1,
      col(self)) then
   lin(self) := lin(self)+1
end
```

Rule for BLUE:

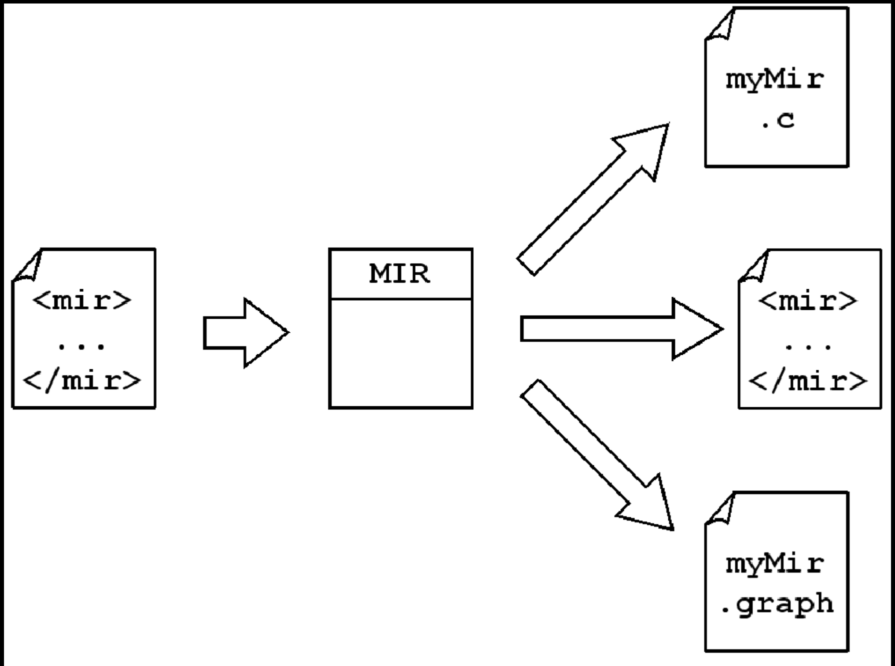
```
if isFree (lin(self),
      col(self)+1) then
   col(self) := col(self)+1
End
```

The MIR Approach for Concurrency

- The proposed model does not provide any primitive for synchronization between agents
- All communication must be explicitly programmed by the designer of an ASM concurrent specification
- The workbench may provide a library with primitives for synchronization (written in ASM itself)

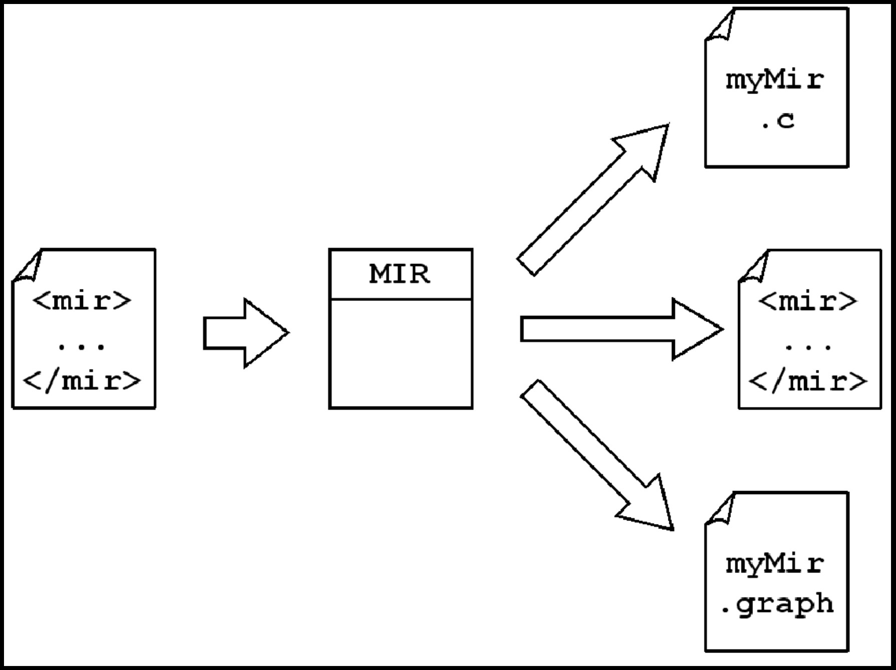
Highlights of MIR Implementation

- Serialization: The MIR representation of an ASM specification can be saved in persistent store as a XML file
- Compilation: It can be compiled into C++ Code
- Direct Execution: MIR can be also interpreted



Highlights of MIR Implementation

- Visualization: It is also possible to obtain a visual representation of a MIR specification through the generation of its description in the DOT language of the GraphViz software.



MIR and Optimization

- There are some optimization opportunities that are particular to the ASM model, and therefore they are not performed by the existing C++ compilers
- In order to address this special situation, it is under development a framework that provides the proper environment to plugin specific MIR optimizations

Conclusions

- Contributions of the proposed infrastructure:
  - It can be used to implement a whole family of languages targeting the ASM model
  - A new approach for concurrent ASM, which can be used to precisely describe distributed algorithms in ASM
  - optimizations can be plugged in, allowing enhancements of the generated code