Abstract State Machine

Presented By

Lin Han
Vanithadevi Devarajan
Xuesong Chen
Yun Wu

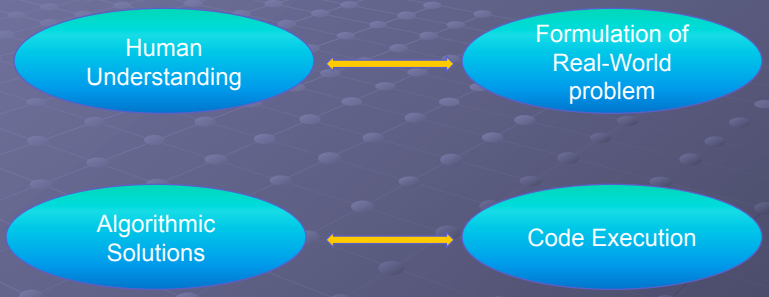
Outline

- Introduction to ASM
- ASM Tools
- Tool Comparison
- ASML & Tool Demo
- Applications

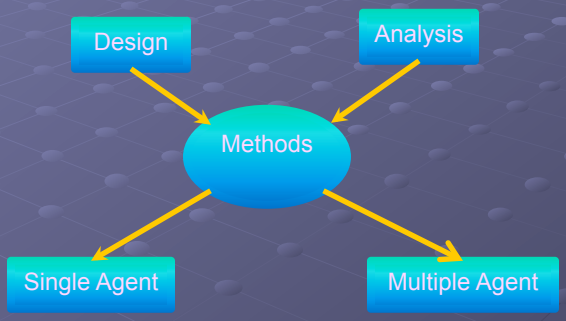
What is ASM?

- ASM – Abstract State Machine.
- Proposed by Yuri Gurevich in mid 1980's.
- Further developed by Egon Borger, as a Software Engineering Method for Software development.
- Embedded Hardware and Software Systems.

Motivation



Software Reliability



Hierarchial Modeling Technique

- Ground Model
  - Result of Requirement Capture Process
  - Binding Contract between Customer and Design Engineer
- Refined Model
  - Refining of Ground Model
  - Design Decisions

Basic Definition of ASM

- A basic ASM M is a tuple of the form ($\sum$; £; R; P_M)

  $\sum$: Signature (i.e. a finite set of function names f)
  £ : Set of initial states for signature
  R : Set of rule declarations
  P_M : Distinguished rule of arbitrary zero

ASM Terms

- Mathematical Logic

  Eg: Terms, Variable assignment, Formula,
  Evaluation of Terms, Semantics of Formula

- State Transitions

$$S_0 \xrightarrow{\Delta_{S_0}} S_1 \xrightarrow{\Delta_{S_1}} S_2 \xrightarrow{\Delta_{S_2}} \cdots,$$

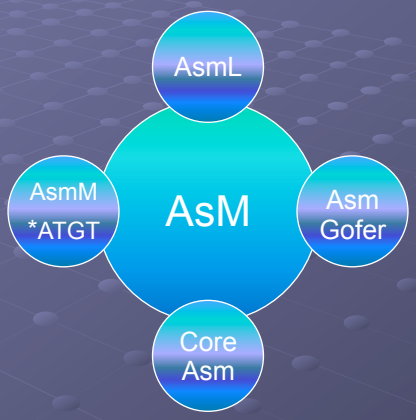
- Transition Rules

  Eg: Skip rule : Skip
  Conditional rule : if x then P else Q

Characteristics of ASM Methodology

- Precision
  - ASM itself semantically clear and Mathematics underpin.
- Faithfulness
  - How does one know that the specification accurately describes the corresponding real system?
- Understandability
  - Easy to read and write specification

Asm Tools



ASM Tool Comparison
ASML

- Executable Specification Language
- A precise, non-ambiguous way to specify a Computer System
- An ideal way for teams to communicate design decisions
- Benefits
- seful before you commit yourself to coding the entire system

ASMGofer

- An advanced Abstract State Machine programming system.
- Provides a Modern ASM Interpreter embedded in the well known functional programming language Gofer.
- AsmGofer is an extension of TkGofer, developed in order to support GUI.

AsmM

- ASMETA, the Abstract State Machine Metamodel and its Tool Set.
- Guidelines of the Model-Driven Engineering.
- Includes ATGT.

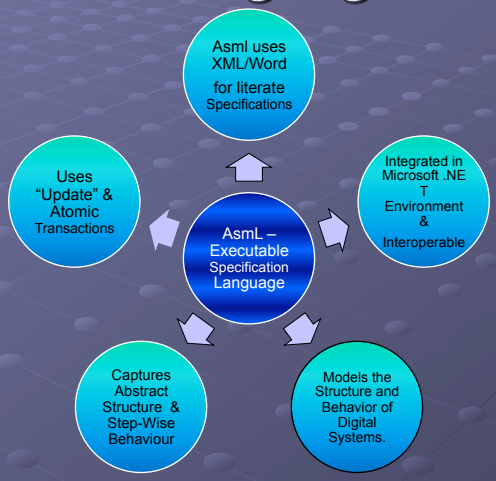
ASM & ASML

- Can one generalize Turing Machines, so that any algorithm, never mind how abstract, can be modeled by a generalized machine very closely and faithfully? – NO
- Suppose, such generalized Turing Machines exists, What would be their states be? – First Order Structures.
- What instructions should the generalized machines have?
- Can one get away with only a bounded set of instructions?

⇒ Computation as an Evolution of the State
Evolving Algebra / Dynamic Algebra / ASM

Non-Executable ASM ⇒ Executable ASML

ASML - Abstract State Machine Language



Modeling Approach of ASML

- Models
  - How existing system works and How new system works?
- Abstract State
  - Encodes only those aspects of System's structure that affects the behavior being modeled
  - Goal – Use minimum details to reproduce the behavior of the System
- Distinct Operational Steps
  - Behavior of the model matches, step by step, the behavior of the System being modeled
  - Step-for-Step Correspondence is an important aspect of ASML
  - Technical name for the way ASML models describe operational steps – ASM
  - Behavior of the machine – RUN – Sequence of states linked by State Transitions
- Evolution of State Variables
  - Machines State as a dictionary of (name,value) pairs, called State Variables.
  - Names are given by Machine's Symbolic Vocabulary.
  - Values are Fixed Elements, like Numbers and Strings of Characters.

Example

- Fig. shows the RUN of a Machine that models how orders might be processed.
- Each Transition Operation is a result of invoking the Machine's Control Logic on the current state. Each Operation calculates the subsequence state as output.

Control Logic

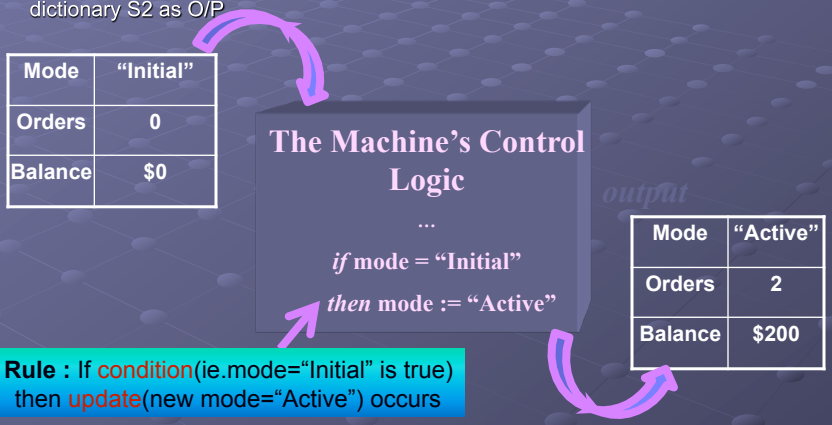
- Control Logic behaves like a fixed set of transition rules that say how state may evolve.
- Think of the Control Logic as a text that precisely specifies,
  - "For any given State, What the values of the Machine's Variables will be in the next Step"

Typical form of the Operational Text is:

"if *condition* then *update*"

Control Logic as a Black Box

- The MCL is a black box that takes state dictionary S1 as input and gives state dictionary S2 as O/P



Rule : If condition(ie.mode="Initial" is true) then update(new mode="Active") occurs

Update Operations

- We use the
  Symbol := → Update Operator (Read as "gets")
  - Used to indicate the value that a Name will have in the resulting state.

Eg: mode := "Active" ⇒ mode gets Active
a := b ⇒ Update Operations.

Atomic Transaction

- A Consequence of handling states to the Control Logic Unit and observing only the new state is that changes are not visible as intermediate results. New states can be seen only during the following step.
- Contrast to Java,C,Visual Basic etc, where changes takes place immediately in sequential order.
- In ASML, all changes happen simultaneously when we move from one step to another and then all Updates happen at once – Atomic Transaction.
- Treating updates as Atomic Transactions is an essential feature of modeling.
- Order of update operations within a step is not specific.

Programs

- A machine is defined by a program that has State Variables and Control Logic.
- Syntax – Similar to simple programming languages.
- But programs with this simple syntax have different properties from procedural programs.
- ASML programs use identation to denote block structure, unlike {..} or special keywords like “Begin” and “end”.
- ASML does not recognises “tab”.
- The keyword “step” marks the transition from one state to another.

Stepping into ASML

```
// An AsmL Program for I/O
// Variable Initialisation
var A as String
myInt1 as Integer=100

Main()
  WriteLine("Integer Value : " + myInt1)
  Write("Enter your name: ")
  step
   A = ReadLine()
   WriteLine("Entered Name : " + A)
  step
   WriteLine("Hello " + A + ".")
```

Output : Integer value : 100
Enter your name : John
Hello John.

The Update Statement & Importance of "Step"

```
var x = 0
var y = 1
Main()
 step
   WriteLine("In the first step, x =" + x)   // x is 0
   WriteLine ("In the first step, y =" + y)  // y is 1
   x:=2
 step
  y:=3
  // updates occur here
  WriteLine("In the second step, x =" + x)  // x is 2
  WriteLine("In the second step, y =" + y)  // y is 1
```

→ Value of "x" is upadated to 2
→ Value of "y" is NOT updated to 3

⇨ Updates don't actually occur until the Step following the one in which they are written.

Special Feature of ASML – "Parallelism"

Parallel Execution

```
// parallel.asml
var A as Integer = 0
Main()
// parallel block begins WriteLine(A)
A := A + 1
WriteLine(A)
A := A + 1
WriteLine(A)
// parallel block ends
```

Non-Parallel Execution

```
// nonparallel.asml
var A as Integer = 0
Main()
// non-parallel block begins
 step
WriteLine(A)
 step
A := A + 1
step
WriteLine(A)
step A := A + 1
step WriteLine(A)
// non-parallel block ends
```


Importance of "Parallelism"

In C, Java

- Enforced Sequential Operations.
- Need for Temporary Variable

```
t := a
a := b
b := t
```

In AsmL

- No Enforced Sequential Operation – "Parallelism"
- One-Step Operation

```
var A as Integer = 1
var B as Integer = 2
Main()
 step
  A := B
  B := A
  // update not yet taken effect
  WriteLine("In the first step, A = " + A)
  WriteLine("In the first step, B = " + B)
step
  // updates have taken effect
  WriteLine("In the second step, A = " + A)
  WriteLine("In the second step, B = " + B)
```

Consistency of updates

- The order within a step does not matter, but all of updates in the step must be consistent.
- None of the updates given within a step may contradict each other.
- If updates do contradict, then they are called **"inconsistent updates"** and an error occur

Eg : step
x:=1
x:=2

Error : Clash in the Update set, since we may not know which one of the two values takes effect.

Total and Partial Updates

- An **update** of the variable can either be **total** or **partial**
- **Total update** is a simple replacement of variable's value with a new value
- **Partial updates** apply to variables that have structure such as sets.

- Total Update

```
var Students as Set of String = {}
Main()
step
  WriteLine ("The initial roster is = " + Students)
  Students := {"Bill", "Carol", "Ted", "Alice"}
step
  WriteLine ("The final roster is = "+ Students)
```

Partial Update

```
var Students as Set of String = {}
Main()
 step
  WriteLine("The initial roster is = " + Students)
  Students("Bill") := true
  Students("Carol") := true
  Students("Ted") := true
  Students("Alice") := true
 step
  WriteLine("The updated roster is = " + Students)
  Students("Bill") := false
 step
  WriteLine("The current roster is = " + Students)
```


Class

- Similar to other languages.
- Templates for user-defined types
  - Fields
  - Methods
- Difference
  - Objects have meaning only as Identities
  - Abstract element or Object-Identities

Class

- Fields
- Instance of class

```
// Sample Program
class Person
  var name as String
  var age as String
var p1 = new Person("William",40)
var p1 = new Person("Amy",20)
Main()
   step
      p1.name := "Bill"
    step
     WriteLine("Name of p1 : " + p1)
   step
     WriteLine("Name of p2 : " + p1)
```


Non-Determinism

- Finite set of possibilities
- Within that set the result may be any value, but we don't know which one

- Non-Deterministic Choice

```
A = {1..10}
Main()
   step
      x= any y | y in A
       WriteLine("X is " + x)
```

Conditionals and Loops

- If statement

Symbol	Meaning
<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

- Logical Operator
  - and, or, not, iff, implies

Importance of "AsmL"

- Precise, non-ambiguous way to specify a Computer System.
- Ideal way for design decisions.
- Program Managers, Developers and Testers, all can use an AsmL specification to achieve Single Unified Understanding.
- Greatest Benefit – Execution.
- Useful before committing to code.
- Answers the following questions by exploring the design
  - Does it do everything you intended to do?
  - How do the features interact?
  - Are there any unintended behaviors?

Application

- Game of Life

http://www.di.unipi.it/~boerger/ASMTutorialEtaps.html

- Is a cellular automation devised by British Mathematician John Horton Conway in 1970.
- It is the best-known example of a Cellular Automaton.

Game of Life - Rules

- **For a space that is 'populated':**
  - Each cell with one or no neighbors dies, as if by loneliness.
  - Each cell with four or more neighbors dies, as if by overpopulation.
  - Each cell with two or three neighbors survives.
- **For a space that is 'empty' or 'unpopulated':**
  - Each cell with three neighbors becomes populated.

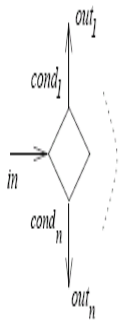
AsmGofer – Games Of Life

- Demo

Dynamic Behavior of the Game

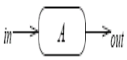
- letDie(cell, n) = **if** status(cell) = Alive
  and (n<2 **or** n>3) **then** status(cell) := Dead
- letLive(cell, n) = **if** status(cell)=Dead
  and n=3 **then** status(cell) := Alive
- gameOfLife = **forall** cell **in** allCells
  do
  letDie(cell,numberOfAliveNeighbors(cell))
  letLive(cell, numberOAlivefNeighbors(cell))

Simulation of UML Charts
Use of ASM in UML



Rule Branching Node
if $currTarget$ *is* $node(in, (cond_i)_{i \le n}, (out_i)_{i \le n})$
then
if $cond_1$ then $active := \rightarrow out_1$
⋮
elseif $cond_n$ then $active := \rightarrow out_n$

Another Example



Rule Action Node
if $currTarget$ *is* $node(in, A, out, isDynamic, dynArgs, dynMult)$
then if ¬ $isDynamic$ then A
elseif $|dynArgs| = dynMult$ then
forall $L_i \in dynArgs$
$A(L_i)$
$active := \rightarrow out$

- isdynamic means if A may be executed DynArgs times in parallel, each time with an argument L from a set dynArgs of object [L1,L2…]

Based on ASML

- Coin Puzzle

References

- http://www.matt-webster.com/misc/asml/
- http://lamspeople.epfl.ch/rychkova/PhD/Research.html
- http://en.wikipedia.org/wiki/Abstract_state_machines
- http://www.bitstorm.org/gameoflife/
- http://research.microsoft.com/en-us/groups/foundations/
- http://edoc.hu-berlin.de/series/informatik-berichte/203/PDF/203.pdf