UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

Asmeta ed AsmetaS

Contatti:
Prof. Angelo Gargantini – angelo.gargantini@unibg.it
Dott. Andrea Bombarda – andrea.bombarda@unibg.it

Abstract State Machines: ASMs (1)

- Sono estensione delle *Finite State Machines*, in cui
  - Stati: sono strutture ordinate del prim'ordine (domini, funzioni, …)
  - Transition rules: descrivono il cambiamento da uno stato al successivo

- Le ASM
  - Sono rigorose, formali ed eseguibili
  - Hanno infiniti stati
  - Hanno una serie di tools che si possono utilizzare nel processo di analisi (inclusi nel framework ASMETA)

Abstract State Machines: ASMs (2)

Domain

State: S_1, S_2

Function

ctl_state: State

input: String

Transition rules

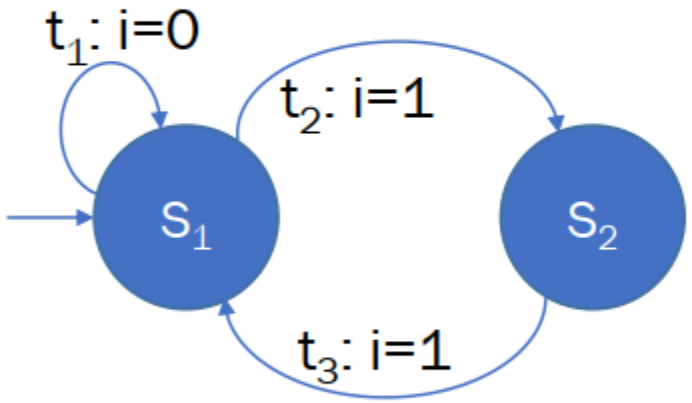
t_1: if (*ctl_state* = S_1 and *input* = "0") then
ctl_state := S_1

t_2: if (*ctl_state* = S_1 and *input* = "1") then
ctl_state := S_2

t_3: if (*ctl_state* = S_2 and *input* = "1") then
ctl_state := S_1

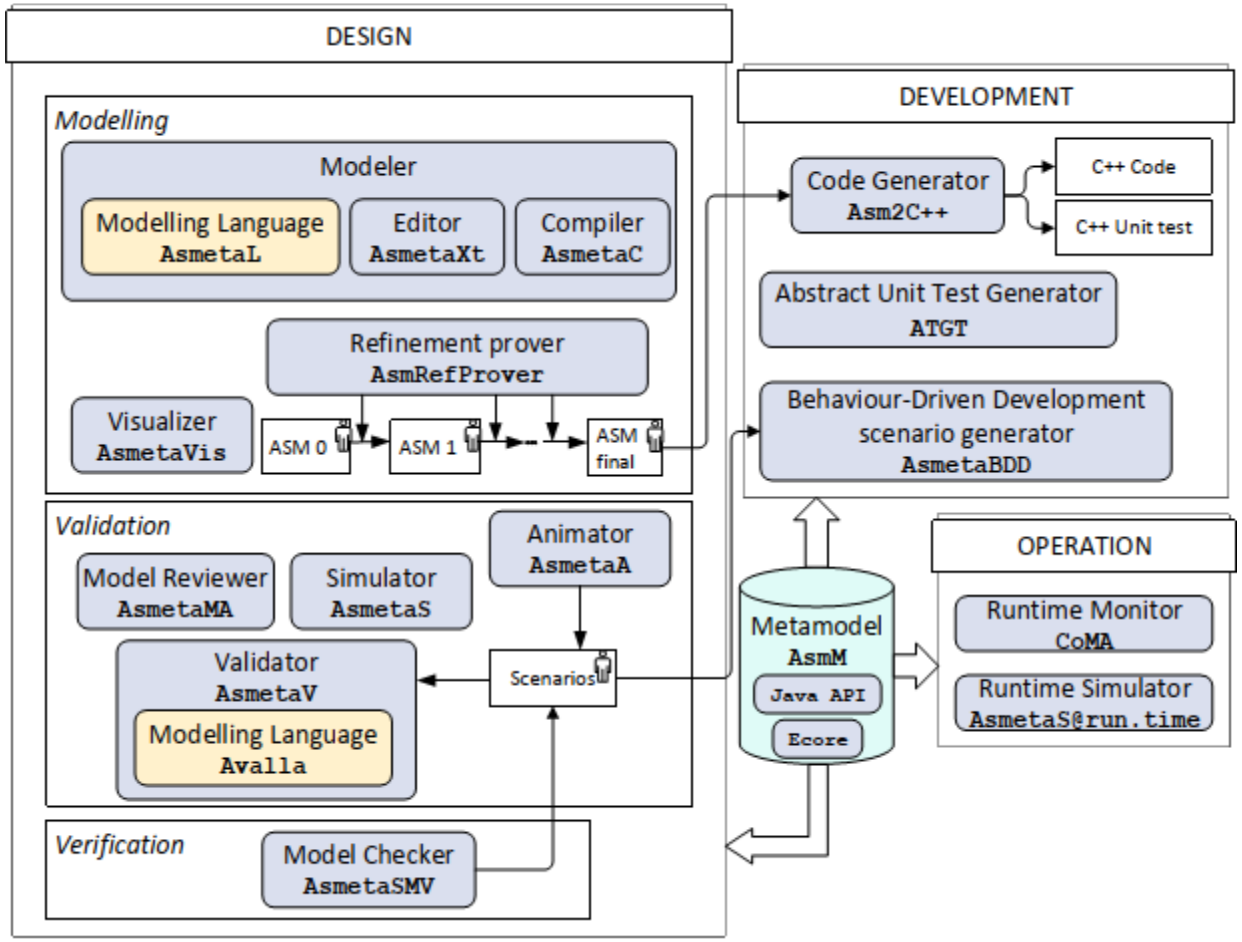
Initial state

ctl_state = S_1



ASMETA

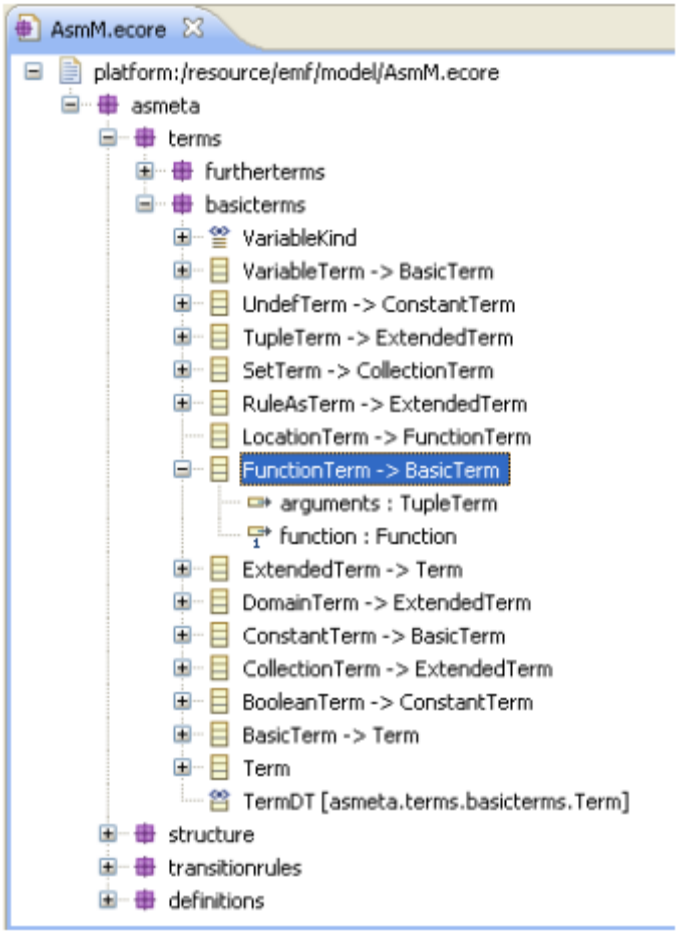
ASMETA (1)


DESIGN
Modelling
Modeler
Modelling Language
AsmetaL
Editor
AsmetaXt
Compiler
AsmetaC
Refinement prover
AsmRefProver
Visualizer
AsmetaVis
ASM 0
ASM 1
ASM final
Validation
Animator
AsmetaA
Model Reviewer
AsmetaMA
Simulator
AsmetaS
Validator
AsmetaV
Modelling Language
Avalla
Scenarios
Verification
Model Checker
AsmetaSMV
DEVELOPMENT
Code Generator
Asm2C++
C++ Code
C++ Unit test
Abstract Unit Test Generator
ATGT
Behaviour-Driven Development
scenario generator
AsmetaBDD
Metamodel
AsmM
Java API
Ecore
OPERATION
Runtime Monitor
CoMA
Runtime Simulator
AsmetaS@run.time

ASMETA (2)

Il framework ASMETA è stato sviluppato partendo dalla definizione di AsmM, un metamodello per le ASMs.

AsmM e stato definito utilizzando l'Eclipse Modeling Framework (EMF)



ASMETA Tools (1)

- **AsmetaL**, una sintassi concreta che permette di scrivere modelli ASMs in formato testuale;
- **AsmetaLc**, un compilatore text-to-model che permette di effettuare il parsing di modelli AsmetaL e controllare che rispettino i vincoli imposti dal metamodello AsmM (espressi come vincoli OCL);
- **AsmetaS**, un simulatore per eseguire modelli ASMs;
- **Avalla**, un linguaggio per la scrittura di scenari; gli scenari Avalla sono interpretati ed eseguiti dal tool **AsmetaV**;
- **ATGT,** un tool per la generazione di casi di test basato sul model checker SPIN;
- **AsmetaXT,** un front-end grafico per la scrittura di modelli AsmetaL;
- **AsmetaSMV**, un model checker, basato su NuSMV, di modelli AsmetaL;
- **AsmetaVis**, visualizza con un grafo i modelli ASM;
- **Asm2C++**: traduce i modelli ASM in linguaggio C++;
- **AsmetaA**: animatore dei modelli ASM;
- AsmetaMA, un model advisor in grado di rilevare proprietà strutturali del modello.

ASMETA Tools (2)

- Tutto il codice e rilasciato sotto licenza GPL.

- È ospitato su Github e può essere scaricato: https://github.com/asmeta

- La StandardLibrary può essere scaricata da https://github.com/asmeta/asmeta/blob/master/asm_examples/STDL/StandardLibrary.asm

- Potete scaricare il tutto dal marketplace di Eclipse (cercate Asmeta)

ASM

HEADER

asm *name*
import StandardLibrary
signature:
[dom_declarations]
[fun_declarations]

BODY

definitions:
[domain D1 = Dterm1 …
domain Ds = Dterms]
[function F1 [(p11 in d11 ,….,p1k1 in d1k1)] = Fterm1 …
function Ff [(pf1 in df1,…,pfkf in dfkf)] = Ftermf]

[rule_declarations]

MAIN RULE

[main rule R = rule]

INITIALIZATION

[default] init I_d:
[dom_initialization]
[fun_initialization]

Esempio di una ASM

```
/** at every step increments the seconds
*/

asm AdvancedClock

import ../../STDL/StandardLibrary

signature:
domain Second subsetof Integer
domain Minute subsetof Integer
domain Hour subsetof Integer
controlled seconds: Second
controlled minutes: Minute
controlled hours: Hour

definitions:
domain Second = {0 : 59}
domain Minute= {0 : 59}
domain Hour = {0 : 23}
```

```
macro rule r_IncMinHours =
par
if minutes = 59 then
hours := (hours + 1) mod 24
endif
minutes := (minutes + 1) mod 60
endpar

main rule r_Main =
par
if seconds = 59 then
r_IncMinHours[]
endif
seconds := (seconds + 1) mod 60
endpar

default init s0:
function seconds = 0
function minutes = 0
function hours = 0
```

Domini

- I **domini** ASMETA si suddividono in:
  - Predefiniti: Integer, Boolean, String, …
  - User-defined: tipi astratti, enumerativi, definiti a partire da altri domini

  Esempi di domini user-defined sono:

```
abstract domain MyDomain
enum domain Colors={RED, BLUE, WHITE}
domain Hours subset of Integer //0-23
```

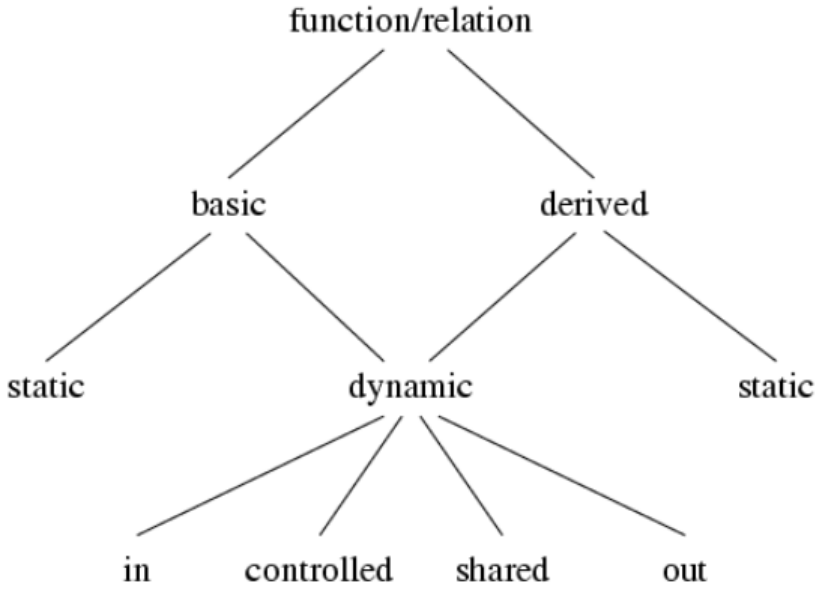
Funzioni

*Sia **M** un ASM e **env** l'ambiente di M*

- **Dynamic**: i valori dipendono dagli stati di ***M***

- in (**monitored**): lette (non aggiornate) da ***M***, scritte da ***env***
- **out**: scritte (ma non lette) da ***M***, lette da ***env***
- **controlled**: lette e scritte da ***M***
- **shared**: lette e scritte da ***M*** e da ***env***

- **Derived**: valori computati da funzioni monitorate e funzioni statiche per mezzo di una "legge" o "schema" fissati a priori



Rules (1)

Skip Rule:

skip

Significato: non fare niente

Update rule:

x := t

Significato: Assegna il vaolre di t a x

Rules (2)

Block Rule (composizione parallela):

```
par
    R
    S
endpar
```

Significato: R e S sono eseguite in parallelo

```
par
        f := 5
        g := f
endpar
```

Nello stato successivo ottengo -> f=5 e g=3

```
par
        f := 5
        f := 8
endpar
```

Update inconsistente -> non posso aggiornare nello stesso stato una funzione con due diversi valori

Rules (3)

Sequence Rule (sequenza):

```
seq
    R
    S
endseq
```

Significato: R e S sono eseguite in sequenza

```
seq
        f := 5
        g := f
endseq
```

Nello stato successivo ottengo -> f=5 e g=5

```
seq
        f := 5
        f := 8
endseq
```

Nello stato successive ottengo -> f=8

Rules (4)

Conditional Rule:

if j **then** *R* **else** *S* ***endif***

Significato: se j è vera, allora esegui R, altrimenti esegui S

Let Rule (se R ha parametri, realizza il passaggio per valore):

let x = t **in** R

Significato: Assegna il vaolre di t a x e esegui R

Forall Rule:

forall *x* **with** j **do** *R*

Significato: Esegui R in parallelo per ogni x che soddisfa j

Implementa il concetto di *parallelismo sincrono* (bounded)

Choose Rule:

choose *x* **with** j **do** *R*

Significato: Esegui R in parallelo per un x che soddisfa j

Implementa il concetto di *non-determinismo*

(Macro) Call Rule:

r [t1, . . . , tn]

Significato: Chiama r (regola con parametri) con argomenti t1, . . . , tn

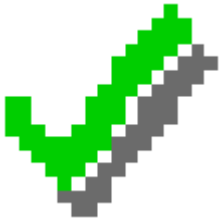
Rule Definition:

rule r (x1, …, xn) = R

Significato: In una call rule r [t1, . . . , tn] le variabili x_i che occorrono nel corpo R della definizione di r vengono sostituite dai parametri t_i

AsmetaLc

Parser



- Il parser AsmetaLc (AsmetaL Compiler) permette di processare specifiche AsmetaL e controllarne la consistenza rispetto ai vincoli OCL del metamodello

AsmetaS

Simulatore

- Il simulatore può essere eseguito in due modalità:

- **Interattiva**, in cui le variabili monitorate sono inserite nella console

- **Random**, in cui le variabili monitorate vengono scelte casualmente dal simulatore

In ogni caso, il simulatore invoca automaticamente il parser prima di simulare l'ASM

AsmetaA

Animatore

- L’animatore può essere eseguito in due modalità:

- **Interattiva**, in cui le variabili monitorate sono inserite nelle box

- **Random**, in cui le variabili monitorate vengono scelte casualmente

In ogni caso, l’animatore si basa sul simulatore

Algoritmo di Euclide

Algoritmo di Euclide

```
Function MCD (a,b)
        while a!=b
                if a>b
                        a:=a-b
                else
                        b:=b-a
        return a
```

- Creare un'ASM in AsmetaL che implementi l'algoritmo di Euclide nel seguente modo:
  - Ogni step della macchina deve corrispondere ad un'iterazione del ciclo while;
  - I valori di cui bisogna calcolare l'MCD devono essere impostati nello stato iniziale;
- (Variante) Modificare il modello visto in precedenza per fare in modo che, in simulazione, venga richiesto all'utente il valore dei due numeri su cui eseguire l'MCD

CoffeeMachine

Coffee Machine (1)

- Una macchinetta automatica dispensa caffe, te e latte.
- La macchinetta accetta solo monete da 50 centesimi e da 1 euro.
- Se viene inserita una moneta da 50 centesimi, la macchinetta dispensa latte (se disponibile); se viene inserita una moneta da 1 euro, invece, la macchinetta decide in modo casuale di dispensare caffè o tè (se disponibili).
- Se viene dispensata una bevanda, la sua disponibilità viene decrementata e la moneta viene conservata nella macchinetta.

Coffee Machine (2)

- Ogni passo di ASM deve corrispondere all'inserimento di una moneta e all'eventuale erogazione di una bevanda corrispondente.
- L'utente del sistema decide, ad ogni passo di simulazione, il tipo di moneta da inserire.
- La macchina all'inizio contiene 10 unita per ogni bevanda; l'atto di erogazione di una bevanda corrisponde alla diminuzione di un'unita della disponibilità della stessa e alla conservazione della moneta (nelle monete conservate, non bisogna distinguere tra monete da 50 centesimi ed 1 euro).
- Se la bevanda non e disponibile, non viene erogata e la moneta non viene conservata.
- La macchina può contenere al massimo 25 monete. Quando la macchina e piena di monete, non accetta altre monete e, quindi, non eroga più alcuna bevanda.
- All'inizio la macchinetta non contiene alcuna moneta.

Morra Cinese

Morra Cinese (1)

- Lo scopo del gioco e quello di sconfiggere l'avversario scegliendo un segno (arma) in grado di battere quello dell'altro, secondo le seguenti regole:
    - Il sasso spezza le forbici (vince il sasso)
    - Le forbici tagliano la carta (vincono le forbici)
    - La carta avvolge il sasso (vince la carta)
    - Se i due giocatori scelgono la stessa arma, il gioco è pari e si gioca di nuovo.

Morra Cinese (2)

- Creare un modello ASM che permetta di giocare a Morra cinese contro il computer. Il modello deve avere le seguenti caratteristiche:
    - ogni step della macchina corrisponde ad una singola giocata;
    - l'utente deve avere la possibilità di scegliere uno dei tre simboli;
    - il computer deve scegliere in modo casuale uno dei tre simboli;
    - la macchina deve valutare la giocata e segnalare il vincitore.

Morra Cinese (3)

- Tre possibili raffinamenti successivi del modello Asm possono essere:

1. Memorizzare il numero di partite vinte dall'utente e dal computer;
2. Impostare il numero massimo di partite che devono essere giocate;
3. Se, dopo aver giocato il numero massimo di partite, il computer e l'utente sono in parità, permettere che continuino a giocare fino a quando uno dei due non vince.