

Abstract State Machines  
Info 3  
A. Gargantini

---

# Scopo del Modulo

- Presentare
  - il formalismo delle Abstract State Machine (ASM)
  - e il metodo di sviluppo di sw complesso basato su di esse
- Imparare ad utilizzare i tools a supporto
  - [asmeta.sf.net](http://asmeta.sf.net)

# Materiale

- Queste slides
- manuali d'uso di asmeta
- Sito web di egon boerger
  - <http://www.di.unipi.it/~boerger/>
- Vedi syllabus

# Idee guida

ASM = FSM con stati generalizzati

Le ASM rappresentano la forma matematica di  
Macchine Virtuali che estendono la nozione di  
Finite State Machine

- Ground Model (descrizioni formali)
- Raffinamenti

Asm come estensione delle FSM

# Estensioni di FSM

Altri modelli di macchine a stati finiti, arricchiti di ulteriori informazioni, tra cui:

- **FSM con evento di output** (ad es. un'azione)
  - macchine di Mealy (output sulla transizione)
  - macchina di Moore (output nello stato)
- **FSM con variabili** (rappresentano la memoria interna della macchina)
- **le Statecharts di UML** dotate dei concetti di sottomacchina (modularità) e composizione sequenziale/parallela
- **le Abstract State Machines (ASM)** dotate dei concetti di sottomacchina, composizione sequenziale/parallela, e di **stato astratto**

# Differenze FSM/ASM

- Le ASM sono analoghe alle FSM
- Le differenze riguardano
  - la concezione degli stati:
    - nelle FSM esiste un unico stato di controllo (`ctl_state`), che può assumere valori in un insieme finito
    - Nelle ASM lo stato è più complesso
  - le condizioni di input e le azioni di output
    - Nelle FSM alfabeto finito
    - Nelle ASM: input qualsiasi espressione, azioni generiche

# Modello computazionale - brief

ASM = Abstract State + *virtual Machine* [Gurevich'95/'99]

Modello Computazionale

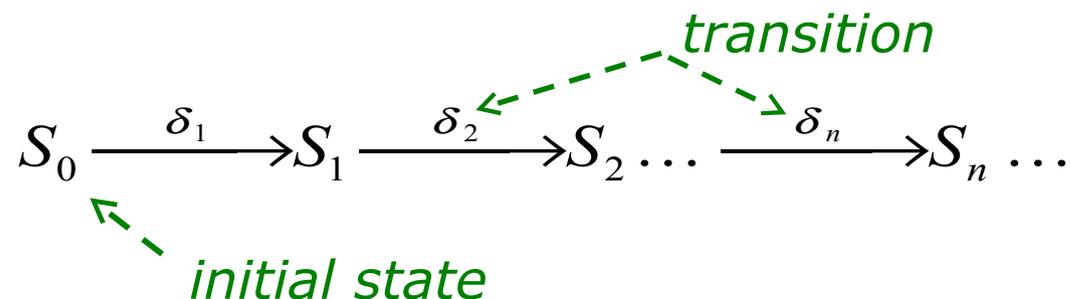
*Vocabolario o segnatura*

**Stato:** strutture del 1st ordine (domini, funzioni, predicati)

**Azioni:** transizioni di stato

*if Cond then Updates*

**Computazione** (una serie di *run* finite o infinite)



**Programmi** (istruzioni di aggiornamento)

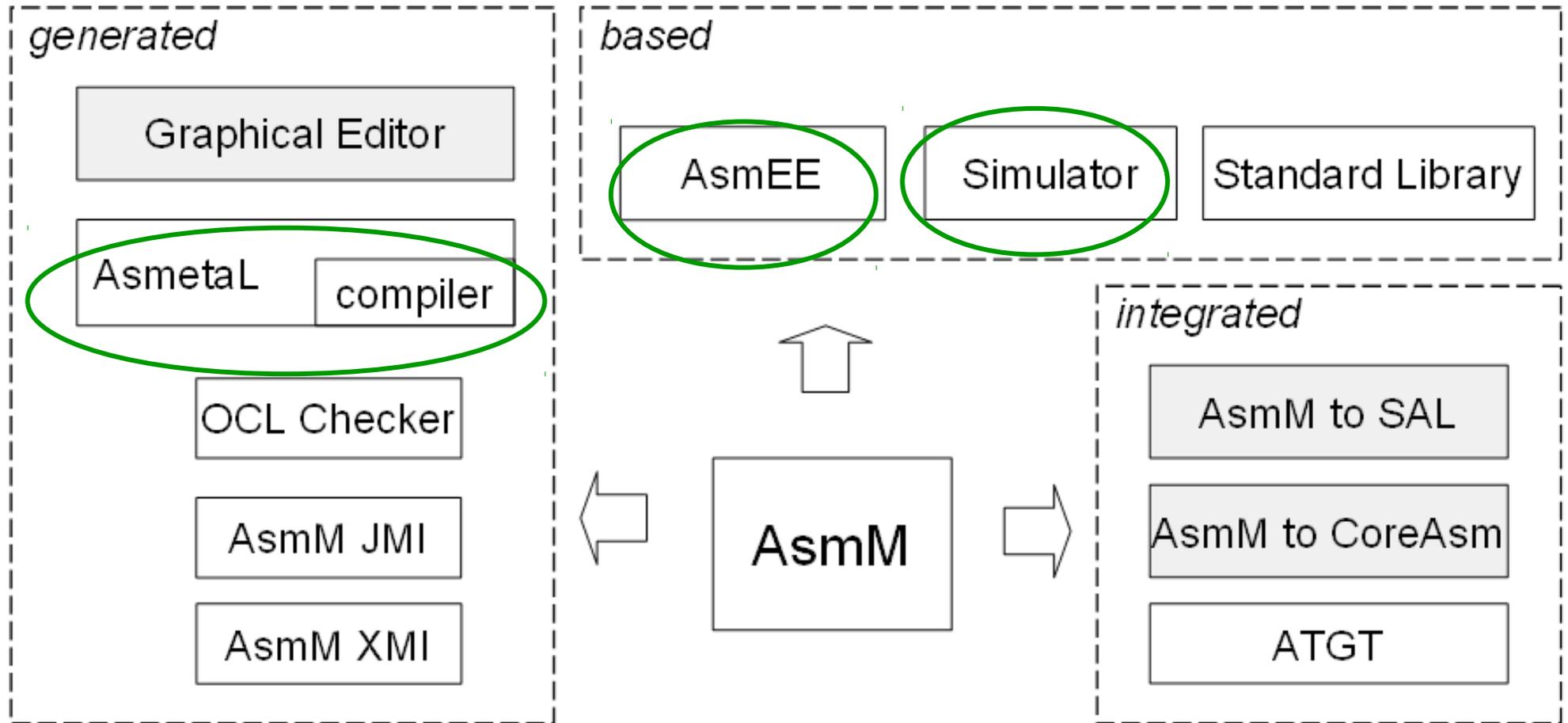
$f(t_1, t_2, \dots, t_n) := t_0$

# Asmeta

- Useremo Asmeta come tool per le ASM:
- Ha un linguaggio
  - AsmetaL
- Un compilatore
  - AsmetaLc
- Un simulatore
  - AmsetaS
- Un ambiente eclipse
  - Asmee
    - <http://fmlab.dti.unimi.it/asmee/update/>

# ASMETA toolset

Un insieme di tool per le *ASMs*, sviluppato con tecniche di *meta-modellazione* dell'approccio *Model-driven Engineering (MDE)*



## Architettura di ASMETA

Istanziamento del framework di meta-modellazione OMG (UML) per le ASMs

# ASMETA tool set 1/2

<http://asmeta.sf.net/>

**complete Java-based code under GPL licence**

**AsmM** Abstract Syntax (**ASM metamodel**) un metamodello in EMF

AsmM Concrete Syntax (**AsmetaL**)

- La [grammatica EBNF](#) sviluppata con [Sun/JavaCC](#)
- Una [quick guide](#) e [Esempi](#) ([rps\\_mono/](#) e [rps\\_agents/](#) ) di specifiche ASM

**AsmM Standard Library**

- [StandardLibrary.asm](#) per domini/funzioni ASM predefinite

**AsmM core Format** (conforms to EMF)

- Della Java API generate secondo EMF

# ASMETA tool set

2/2

**AsmetaLc** compiler per

- processare specifiche [AsmetaL](#)
- controllare la consistenza rispetto ai vincoli AsmM-[OCL](#)
- generare la rappresentazione [XMI](#) (XML-based) corrispondente
- Tradurle in istanze [AsmM](#) in oggetti Java usando le AsmM JMIs

**AsmetaS** simulator

- per simulare/eseguire una specifica ASM
- un [interprete](#) che simula la specifica ASM come istanza di AsmM

ASM Tests Generation Tool ([ATGT](#)), ora AsmM-compliant

Un front-end grafico [ASMEE](#) ([ASM Eclipse Environment](#))

- [Eclipse plug-in](#) che fa da IDE per editare, manipolare, esportare nel formato XMI le spec. ASM usando i tool di cui sopra

# Altri tools

- Validazione tramite scenari
  - Avalla
    - Scrive degli scenari tipo Junit
- Prova di proprietà
  - Dimostrazione automatica di proprietà
    - Tipo key-hoare ma automatico
- Model advisor
  - Per trovare difetti (analisi statica)

# Standard Library

Operatori aritmetici

- +, -, \*, /, %

Operatori relazionali

- =, !=, <, >, <=, >=

Operatori booleani

- not, and, or, implies, iff

Funzioni per insiemi e sequenze

- first(), tail(), union()

I domini standard: Naturali, Integer, ...

- La StandardLibrary.asm deve essere importata

**StandardLibrary.asm**

# Asmeta come si installa e come si avvia

- Simulatore e compilatore possono essere usati a linea di comando
- Useremo il plugin
  - check your AsmetaL specification
  - run (3 modi)
  - stop press in the console to stop the simulation (press twice to abort)

# Come usare AsmetaLc

Per invocare il parser in maniera standalone, da linea di comando:

```
java -jar AsmetaLc.jar <filename.asm>
```

Altri comandi:

```
java -jar AsmetaLc [-xmi] [-log <log4j file>] <filename.asm>
```

-xmi: genera l'output in formato XMI

-log: genera debug info

<log4j file>: configuration file of the logger

<filename.asm>: path name of the ASM spec.

Attenzione! Il nome del file deve essere uguale al nome della ASM

AsmetaLc viene anche invocato automaticamente da AsmetaS, e dall'IDE ASMEE!

# Come usare AsmetaS

Da linea di comando:

```
java -jar AsmetaS.jar <filename>
```

E' possibile specificare alcune opzioni per la terminazione della computazione, tra cui:

-n 3 per un numero fisso (ad es. 3) di passi

-n? per eseguire fino a che l'insieme degli aggiornamenti risulta vuoto

```
java -jar AsmetaS.jar -n 3  
<filename.asm>
```

```
java -jar AsmetaS.jar -n?  
<filename.asm>
```

# AsmetaS output

Il simulatore produce come output la traccia d'esecuzione della macchina.

L'output è disponibile all'utente in due forme:

- come **testo** non formattato inviato **sullo standard output**
- come **documento *xml*** memorizzato nel file ***log.xml*** residente nella directory di lavoro

Per cambiare stili di presentazione e media di memorizzazione, l'opzione `-log` consente di precisare un file `log4j` che sarà considerato dal simulatore in sostituzione di quello di default.

# How to debug your ASM spec

- Il logger per il tracing dell'attività di parsing e la valutazione dei termini e delle regole
- può essere attivato cambiando le **proprietà di del file log4j**:

```
log4j.logger.org.asmeta.interpreter.ReflectiveVisitor=WARN
log4j.logger.org.asmeta.interpreter.TermEvaluator=DEBUG
log4j.logger.org.asmeta.interpreter.RuleEvaluator=OFF
log4j.logger.org.asmeta.interpreter.TermSubstitution=INFO
log4j.logger.org.asmeta.interpreter.RuleSubstitution=OFF
log4j.logger.org.asmeta.interpreter=OFF
```

# AsmetaS modalità

Immettere valori per le funzioni monitorate:

- **Modalità Interattiva**

Per default, l'utente fornisce manualmente i valori quando vengono richiesti

- **Modalità batch (valori letti da file)**

Specificando da linea di comando come ultimo argomento il pathname di **un file d'ambiente .env**, da dove i valori saranno letti

```
java -jar AsmetaS.jar <filename.asm>  
<fileambiente.env>
```

# AsmetaS key features

- **Axiom checker**

Se un assioma viene violato, AsmetaS lancia l'eccezione `InvalidAxiomException` che tiene traccia dell'assioma violato

- **Consistent Updates checking**

In caso di update inconsistenti, AsmetaS lancia l'eccezione

`UpdateClashException` che tiene traccia della coppia di locazioni oggetto dell'inconsistenza

- **Random simulation**

per mezzo di un *ambiente random* per le funzioni monitorate

Per **maggiori info** vedi scarica dal sito la guida:

[AsmetaS\\_quickguide\\_it.pdf](#)

# L'IDE ASMEE

## ASMETA Eclipse Environment



Caratteristiche disponibili in ambiente Eclipse:

- un wizard per **creare un nuovo file AsmetaL:**

File->new File-> Other -> AmsetaL new File

- **synthax hightlighting**

I colori predefiniti possono anche essere modificati:

Window -> Preferences -> Asmee

Azioni per simulare con AsmetaS un file AsmetaL:

-  **check** la tua specifica AsmetaL

 run **until empty**

 run step by step

-  press  nella console per arrestare la simulazione (premere due volte  per abortire)

## Riferimenti bibliografici

- ASMETA Web Site

<http://asmeta.sf.net/>

- ASMETA project

<http://asmeta.sourceforge.net/>

- Pisa Workshop on ASM Open Source Tools

Metamodelling-based ASM Tool-set Development

<http://rotor.di.unipi.it/AsmCenter/Pisa%20Workshop%20on%20ASM%20Open%20Source%20>

## Come contribuire

ASMETA è un progetto open source ospitato da *sourceforge*

- ASMETA Web Site <http://asmeta.sourceforge.net/index.html>

E' possibile contribuire come

- **Sviluppatore** (occorre richiedere un'account)
- **User/tester** per riportare eventuali bug tramite il *Bug Tracking System* (alla voce **Submit New**)  
[http://sourceforge.net/tracker/?group\\_id=155949](http://sourceforge.net/tracker/?group_id=155949)

# Contribuire da sviluppatore

Progetto ospitato da sourceforge.net

- asmeta.sf.net
- funzionante:
  - web site, bug tracking, svn per il codice sorgente
- in futuro:
  - mailing lists, forum, news

log4j for logging

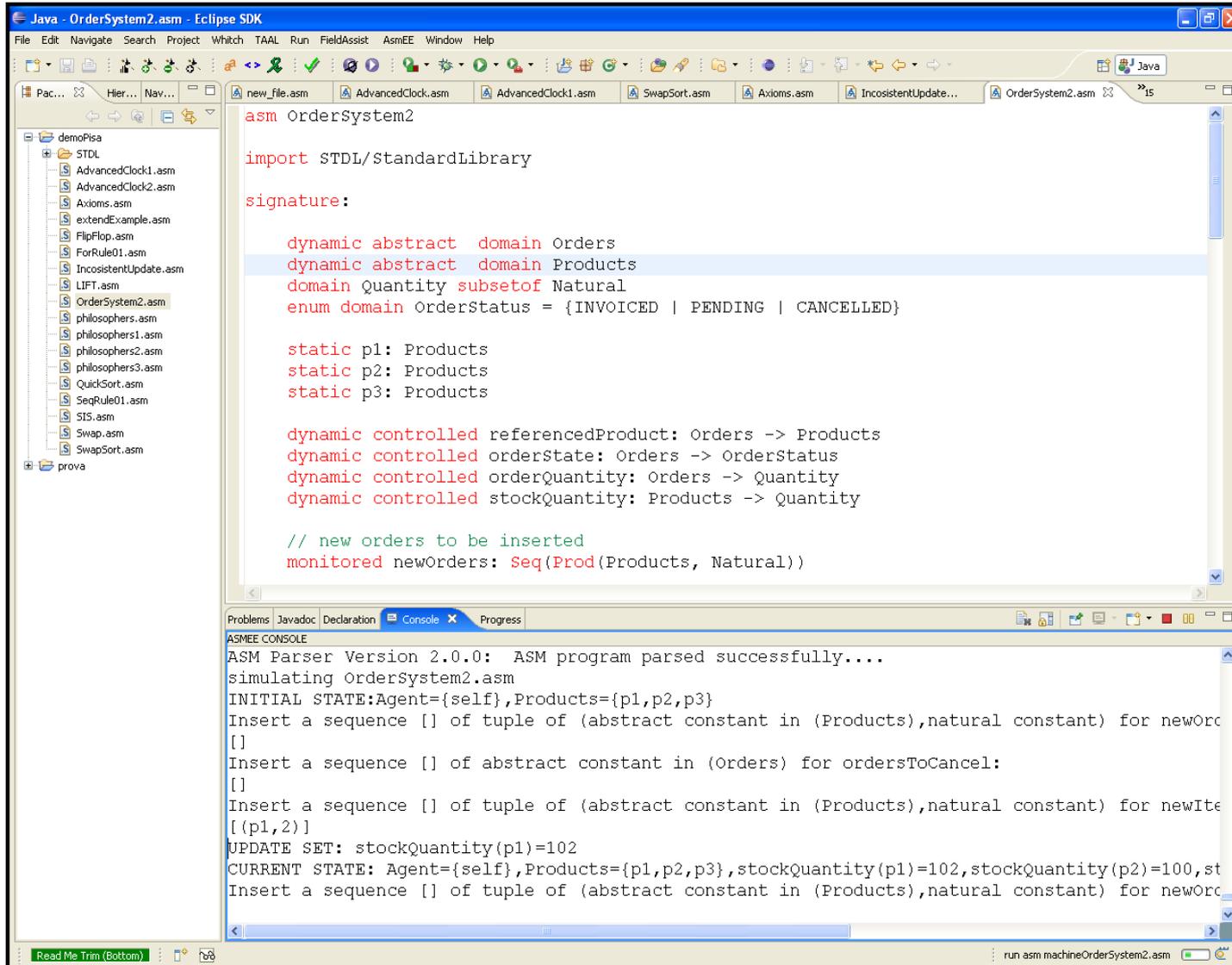
- Ogni utente può personalizzare il suo log (per scopi di debugging di specifiche ASM e per l'interprete)

JUnit and Fit (<http://fit.c2.com/>) per lo unit testing

Emma per la valutazione della copertura del codice “code coverage”

# L'IDE ASMEE

## ASMETA Eclipse Environment



# Il linguaggio AsmetaL

# Il Linguaggio AsmetaL

## Linguaggio strutturale

- costrutti per definire la struttura (scheletro) di una ASM mono-agente o sinc./asinc. multi-agente

## Linguaggio delle definizioni

- costrutti per introdurre (dichiarare e definire) domini (*tipi del linguaggio*), funzioni (con domini e codomini), regole di transizione, e assiomi

## Linguaggio dei termini

- **termini di base** come nella logica del primo ordine (costanti, variabili, termini funzionali  $f(t_1, t_2, \dots, t_n)$ )
- **termini speciali** come tuple, collezioni (insiemi, sequenze, bag, mappe), ecc.

## Linguaggio delle regole

- **regole di base** come skip, update, parallel block, ecc.
- **turbo regole** come seq, iterate, turbo submachine call, ecc.

# 0. Struttura di una ASM in AsmetaL:

## Header section

*Clause di Import/export ,  
vocabolario ASM*

**asm** flip\_flop

Nome deve coincidere  
con il nome del file

**import** StandardLibrary

**signature:** ....

Qui ci va la definizione  
di stato

**definitions:** ....

## Body section

*Static domains/functions  
defs., rule decls., e  
assiomi*

## Initialization section

*Inizializzazioni per  
domini e funzioni dinamici*

**default init** initial\_state:

.....

# Il Linguaggio delle definizioni

Commenti (2 forme possibili):

```
// text to be commented  
/* text to be commented*/
```

# 1. ASM Stati

- Nelle ASM lo stato è definito da un insieme di valori di qualsiasi tipo, memorizzate in apposite **locazioni**
- come nella programmazione **le variabili**
- in ASM si chiamano **funzioni**
- distinguiamo **la cardinalità delle funzioni**
  - **Variabili e costanti (0-arie)**  $x, y, \dots$
  - **Mappe/array/funzioni n-arie**
    - Es:  $\text{name}(1), \text{name}(2), \dots$

# ASM Stati

- Nelle ASM lo stato è definito da un insieme di valori di qualsiasi tipo, memorizzate in apposite **locazioni**
- come nella programmazione **le variabili**
- in ASM si chiamano **funzioni**
- distinguiamo **la cardinalità delle funzioni**
  - **Variabili e costanti (0-arie)**  $x, y, \dots$
  - **Mappe/array/funzioni n-arie**
    - Es:  $\text{name}(1), \text{name}(2), \dots$

# Dinamiche/statiche

- Le **funzioni** possono essere dinamiche o statiche a seconda che il valore della funzione cambia o no da uno stato al successivo
  - Funzioni in senso matematico non informatico come "procedure"
  - Funzioni dinamiche: cambiano nel tempo
  - Funzioni statiche: la loro interpretazione rimane costante
    - Funzioni statiche di arità zero sono dette **costanti**
    - Funzioni dinamiche di arità zero sono le comuni **variabili** dei linguaggi di programmazione
- Vedremo dopo come dichiarare le funzioni - **Domini**

# Definizione dello stato

**header** è:

```
[ import m1 [ ( id11,...,id1h1 ) ] ... ]  
[ export id1,...,ide ] or [ export * ]  
signature :  
  [ dom_declarations ]  
  [ fun_declarations ]
```

import/export di simboli (id) di domini, funzioni (e loro domini e codomini), e regole da/verso altre ASM

**export \*** per esportare tutto

Ricordare: la segnatura contiene dichiarazioni (non definizioni) di domini e funzioni!

Domini

# ASM Domini

*Lo stato di una ASM è solitamente usato per modellare domini eterogenei*

Nelle applicazioni pratiche, il superuniverso  $A$  di uno stato  $A$  è suddiviso in piccoli *universi*, rappresentati dalle loro funzioni caratteristiche.

Ogni universo rappresenta un DOMINIO (che vediamo)

L' *universo* rappresentato da  $f$  è l'insieme di tutti gli elementi  $t$  del superuniverso di  $A$  t.c.  $f(t) = \text{true}$

# ASM Domini

- I soliti domini predefiniti sono disponibili
  - Interi, String
- L'utente può definirne altri
  - Da niente, come tipi astratti, come enumerativi
  - A partire da altri domini (strutturati)
    - Come alias ad esempio
    - ...

# Il Linguaggio delle definizioni – type domain

Caratterizzazione dei domini

**type-domain**: caratterizzano il superuniverso

**basic type-domains**: **Complex, Real, Integer, Natural, String, Char, Boolean, Rule**, e **Undef** definiti nella **standard library**!

**basic domain Real**

**basic domain Integer**

Questi non devono essere definiti, ci sono già

L'utente può definirne di suoi type domains

NOTA: devono iniziare con la maiuscola !

**ID\_DOMAIN** una stringa che inizia con una lettera maiuscola. Esempi: **Integer** **X** **SetOfBags**  
**Person**

# Il Linguaggio delle definizioni dei domini

- **abstract domain**: elementi di natura "astratta", non definiti se non attraverso funzioni definite su tale dominio

## **abstract domain D**

dove D è il nome del type domain

- Esempio: abstract domain Student
- **enum**: enumerazioni,
- **enum domain D = { EL1 | ... | ELn }**
- dove D è il nome e EL1, ..., ELn le costanti dell'enumerazione
  - **ID\_ENUM** una stringa di lunghezza  $\geq 2$ , fatta di sole lettere maiuscole. Esempi: **ON** **OFF** **RED**
- ad esempio enum domain Color = {RED | GREEN | BLUE}

# Domini Concreti

Dichiarazioni di **domini concreti** (`dom_declarations`)

user-defined e subset dei type-domain

[ **dynamic** ] **domain D subsetof td**

dove:

- D è il nome del dominio da dichiarare
- td è il type-domain di cui D è subset
- La parola chiave **dynamic** è opzionale e denota che l'insieme è *dinamico* (stesso concetto delle funzioni). Per default, un dominio è *statico* e va *definito* nella sezione **definitions**

*Esempio:*

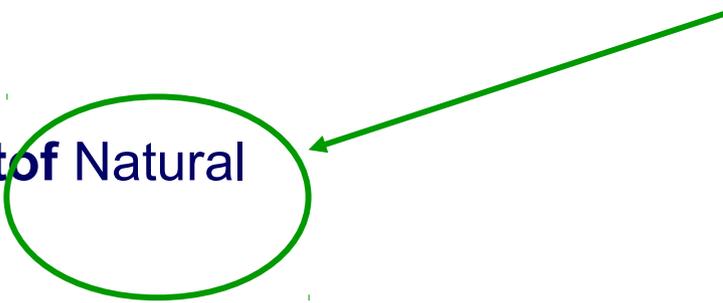
**signature:**

**domain State subsetof Natural**

**definitions:**

**domain State = {0n,1n}**

*Basic domain  
della standard library*



# Caso di studio

- Proviamo a definire un orologio che:
  - Memorizza ora minuti e secondi (stato)
  - ad ogni passo incrementa secondi (e aggiorna lo stato in modo corretto)
  -
- Variante:
  - Ha un input che è il segnale, ed incrementa lo stato solo se il segnale è vero

# Esempio - clock

- Un orologio che ad ogni passo se arriva un segnale avanza di un secondo

domain Second subsetof  
Integer

domain Minute subsetof  
Integer

domain Hour subsetof Integer

# Domini Astratti e classi Java

- I domini astratti sostituiscono i costrutti come ADT (abstract data type) o le struct e le classi Java
- Esempio
- Java: `class Student{ ...}`
- Asm: abstract domain Student

# Structured domain

**structured**: per costruire insiemi finiti, sequenze, bag, mappe, e tuple a partire da altri domini

**Vedremo più avanti**

# Funzioni

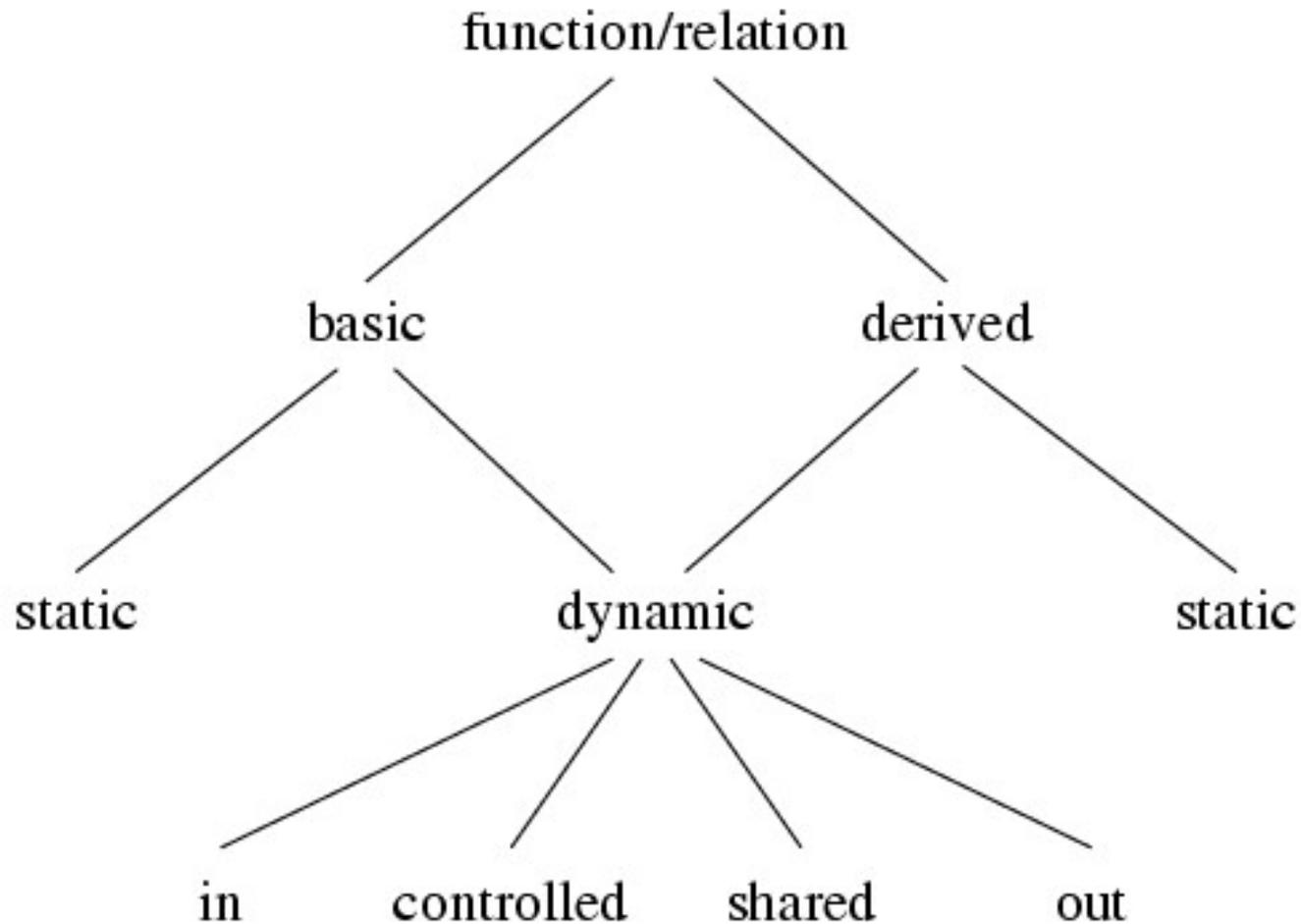
# Costanti

- Le costanti sono funzioni 0-arie statiche
- Sono simboli definiti una volta per tutte
- Per definizione, ogni vocabolario ASM contiene le **costanti undef, True, False**
- I numeri sono costanti numeriche
  - 1,2, ...
- L'utente può aggiungerene di sue
  - Costante minimoVoto = 18

# Variabili

- Le funzioni dinamiche unarie corrispondono alle variabili
- classificazione:

# ASM Function Classification



Realizza l'incapsulamento/information hiding dei linguaggi OO

# ASM Function Classification

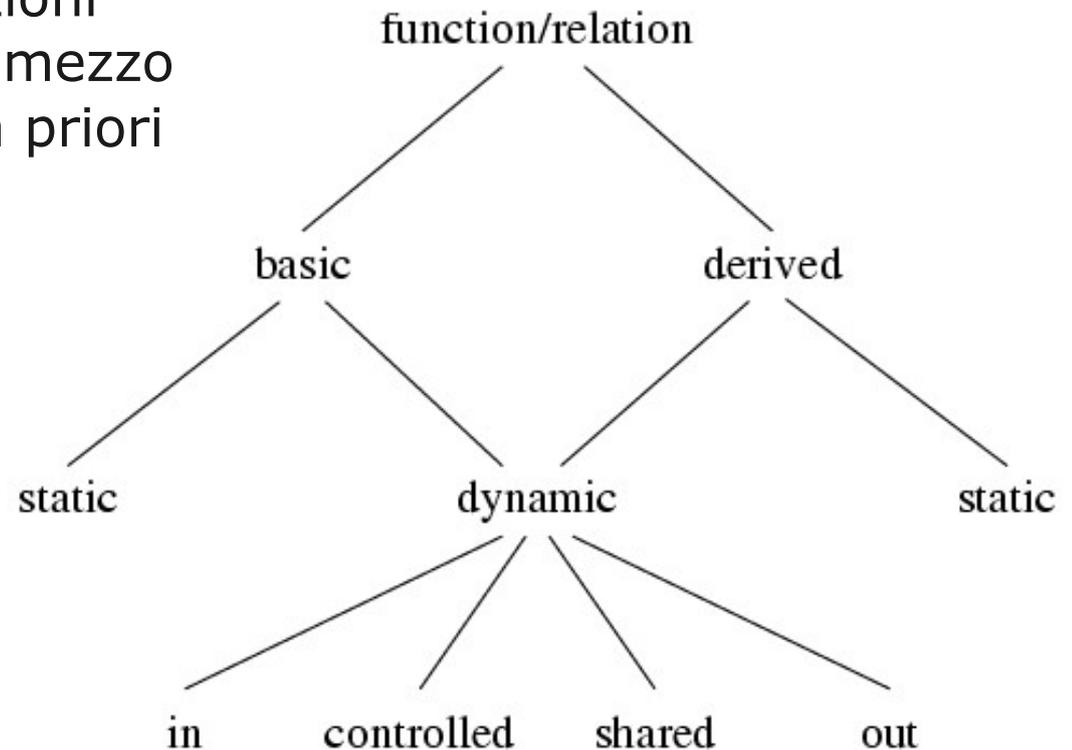
Detta ***M*** l'ASM corrente e ***env*** l'ambiente di ***M***:

**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***

**Richiede la definizione di un protocollo di comunicazione per garantire la consistenza degli aggiornamenti!**

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



# Definizione delle funzioni in AsmetaL

## Dichiarazioni di funzioni 0-arie

Funzioni **statiche**

**static f: C**

[dynamic] **monitored f: C**

[dynamic] **controlled f: C**

Funzioni **dinamiche**

[dynamic] **shared f: C**

[dynamic] **out f: C**

[dynamic] **local f: C**

· C è il codominio di f (f prende valori in C)

### **ID\_FUNCTION**

una stringa che inizia con una lettera minuscola diversa da "r\_" e da "inv\_". Esempi: **plus** **minus** **re**

# Esempio - orologio

monitored signal:Boolean

controlled seconds:Second

controlled minutes:Minute

controlled hours:Hour

# Funzioni statiche n-arie

- Le funzioni statiche sono definite tramite una legge fissa
- Esempio di funzioni statiche sono le usuali operazioni tra numeri
  - +, -, ...
  - Tra booleani AND, ...
  - Sono “standard”
- L'utente può definirne di sue
  - es.:  $\max(n,m)$

# Concetto di funzione dinamica n-aria

Alcune funzioni n-arie possono cambiare “valori”

## **Funzioni dinamiche n-arie**

- Per esprimere dal punto di vista informatico il concetto di funzione, possiamo pensarla come una tabella contenente valori
- Quando si parla di location si può pensare all'indicizzazione di una cella della tabella

# Funzione dinamica n-aria

- Esempio
- voto: Studenti- $\rightarrow$  interi

Funzione:  
Intera tabella

locazione

| Studente | Voto |
|----------|------|
| verdi    | 18   |
| ...      | ...  |
| Rossi    | 30   |
|          |      |

Aggiorno la funzione, mediante aggiornamenti di locazioni,  
Esempio: voto("Rossi") := 30

# Asm funzioni vs campi Java

- Le funzioni asm sono simili ai campi Java
- Java:

```
class Student{ String name...}
```

- **Asm:**

abstract domain Student

controlled name: Student → String

# Definizione delle funzioni in AsmetaL

Dichiarazioni di funzioni (`fun_declarations`)

Funzioni statiche    **static f: [ D -> ] C**

                  [dynamic] **monitored f: [ D -> ] C**

                  [dynamic] **controlled f: [ D -> ] C**

Funzioni dinamiche [dynamic] **shared f: [ D -> ] C**

                  [dynamic] **out f: [ D -> ] C**

                  [dynamic] **local f: [ D -> ] C**

- D e C sono risp. Il dominio ed il codominio di f
- D è opzionale; non va messo se f è 0-aria (cioè una variabile)

# Definizione di funzioni – esempio 1

Dichiarazioni di funzioni (`fun_declarations`)

*Esempi (Flip\_Flop):* **variabili**

**dynamic controlled** `ctl_state : State`

**dynamic monitored** `high : Boolean`

**dynamic monitored** `low : Boolean`

**Costanti**

**static** `value : Integer`

**Altri esempi funzioni n-arie dinamiche (n = 1):**

// una funzione che associa un intero ad ogni intero

**controlled** `votoByID: Integer -> Integer`

// una funzione che dice quali interi sono scelti

**monitored** `interoscelto: Integer -> Boolean`

**demo**

# Definizione delle funzioni statiche

- Le funzioni statiche vanno prima dichiarate e poi definite
- Le costanti:
  - Va dato il loro valore
- Le variabili statiche di domini astratti rappresentano istanze predeterminate
  - Non vanno definite

```
static angelo: Student
```

# Esempio Lift

- Lift: un sistema che gestisce più ascensori

Ogni ascensore può avere due direzioni, e può essere ferma o in movimento

abstract domain Lift

enum domain Dir = {UP | DOWN} //direction

domain Floor subsetof Integer

enum domain State = {HALTING | MOVING} //lift control states

//lift direction of travel, initially UP (see initial state s0)

dynamic controlled direction: Lift -> Dir

dynamic controlled ctrlState: Lift -> State

dynamic controlled floor: Lift -> Floor

## Esempio 2

### **signature:**

**abstract domain** BancomatCard

**enum domain** Pressure\_type = {TOO\_LOW | NORMAL |HIGH}

monitored currCard: BancomatCard //n. della carta presente nel bancomat

controlled pressure : Pressure\_type

# Funzioni n-arie (statiche e dinamiche)

- Il dominio delle funzioni n-arie sono n domini
- In questo caso diciamo che il dominio è un prodotto di domini

in asmetaL:

**Prod ( d1,d2,...,dn )**

d1,...,dn sono i domini del prodotto cartesiano

Esempio:

**// static: del massimo**

**static max: Prod(Integer,Integer) → Integer**

**// dinamica**

**controlled voto: Prod(Student,Class) → Integer**

# Altri domini strutturati

Altri **type-domain** (non dichiarati nella segnatura)

Sequenze

**Seq ( d )**

d è il dominio base delle possibili sequenze

Insiemi (dominio di insiemi)

**Powerset ( d )**

d è il dominio base dell'insieme delle parti (l'insieme di tutti i possibili insiemi di elementi di d)

**Bag ( d )**

d è il dominio base dei possibili bag (borsa)

**Prod ( d1,d2,...,dn )**

d1,...,dn sono i domini del prodotto cartesiano

# Esempi

Dichiarazioni di funzioni (`fun_declarations`) da più domini

```
// for every Lift gives if it is attracted in a directio  
monitored attracted: Prod(Dir, Lift) -> Boolean
```

```
//  
monitored f1: Seq(Integer) -> Boolean
```

```
monitored f2: Seq(Prod(Integer, Boolean))  
// es. f2=[(1,true),(5,false)]
```

```
controlled f3: Boolean -> Prod(Real,Real) //es. f3(true) = (3.0,4.5)
```

```
// given a set of Orders, return the quantity  
static totalQuantity: Powerset(Orders) -> Quantity  
// a constant list of integers  
static list:Seq(Integer) // es. list=[1,2,5,8]
```

# Funzioni statiche n-arie

- **Le funzioni statiche n-arie servono per definire delle leggi per il calcolo**
- **Esempio: massimo tra due numeri**
  - `mymax: prod(Integer,Integer) -> Integer`
- Queste funzioni vanno definite prima di poter essere usate
- Psuedo:  
$$\text{mymax}(x,y) = \text{if } x > y \text{ then return } x \text{ else return } y$$
- In AsmetaL dobbiamo definire la funzione con una espressione (termine)

# Definizioni

# Definizione di domini e funzioni

**body** è:

**definitions :**

**domain** D1 = Dterm1

...

**function** F1[(p11 in d11,...,p1k1 in d1k1)] = Fterm1

...

[ rule\_declarations ]

[ axiom\_declarations ]

Solo **domini concreti statici** possono essere definiti

Solo **funzioni statiche** possono essere definite

per una regola o assioma, dichiarazione e definizione sono la stessa cosa

Termini

# AsmetaL termini

- In AsmetaL i termini o le espressioni sono del tutto simili alle espressioni dei linguaggi di programmazione
  - Funzioni e relazioni
    - Quelle dalle standard library
      - Es:  $3 + 2$
      - $\text{exist } \$x \text{ in Student with voto}(\$x) > 10$
    - Introdotte dall'utente
  - Costanti

# Variabili logiche

Il primo termine sono variabili logiche

Varibili logiche che non fanno parte dello stato

Si possono usare come termini

**\$v**

**ID\_VARIABLE logica** una stringa che inizia con "\$".

Esempi

**\$x**

**\$abc**

**\$pippo**

**definitions :**

**function F1 (p11 in d11, ..., p1k1 in d1k1) = Fterm1**

Esempio una funzione che resituisce se stessa

**signature:**

**static itself : Integer → Integer**

**definitions:**

**function itself(\$x in Integer) = \$x**

# Applicazione difunzioni

**[id . ]f [ ( t1,...,tn ) ]**

**dove:**

- **f è il nome della funzione da applicare**
- **(t1,...,tn) una tupla di termini**
- **id è il riferimento all'agente (se presente) che detiene la funzione f**

Esempio opposto

signature:

**static opp : Integer → Integer**

**definitions:**

**function opp(\$x in Integer) = minus(\$x)**

# Funzioni matematiche

Nota: le funzioni matematiche possono essere usate normalmente:

definitions:

```
function opp($x in Integer) = -$x
```

...

```
max(2,3)
```

```
abs(-4)
```

```
abs(max(-2,-8))
```

```
self.f(5) o f(self,5)
```

# Sequenze e insiemi

|          |  |
|----------|--|
| Sequence | $[t_1, \dots, t_n]$ con $n$ termini della stessa natura<br>$[]$ per la sequenza vuota  |
| Set      | $\{t_1, \dots, t_n\}$ con $n$ termini della stessa natura<br>$\{\}$ per l'insieme vuoto  |
| Bag      | $\langle t_1, \dots, t_n \rangle$ con $n$ termini della stessa natura<br>$\langle \rangle$ per il bag vuoto  |
| Map      | $\{t_1 \rightarrow s_1, \dots, t_n \rightarrow s_n\}$ con $n$ termini della stessa natura, e $S$ termini della stessa natura pure<br>$\{- \rightarrow \}$ per la mappa vuota |

# Esempi

- Esempi di termini: sequence, set e bags
- **Set utili a definire concrete domains**

Sequence     ["hello","bye"]  
              [[],[1,2]]  
              [1..4]  $\equiv$  [1,2,3,4]

Set            {[],[1,2],[1]}  
              {'a','b'}  
              {1..2,0.5}  $\equiv$  {1.0,1.5,2.00}

Bag            <1,2,1>  
              <'a','b','a','b'>  
              <1..10,2>  $\equiv$  <1,3,5,7,9>

# If e let terms

IfTermC

**if G then tthen [else telse] endif**

**dove G è un termine booleano, tthen e telse sono termini della stessa natura**

Usato per definire un valore condizionale tipo operator ?: di Java

LetTerm

**let(v1=t1,...,vn=tn)in tv1,...,vn endlet**

**dove vi sono variabili logiche e t1,...,tn,tv1,...,vn sono termini**

Usato per introdurre nuove variabili „locali“. Queste variabili sono logiche nel senso che non fanno parte dello stato

# Esempio temine condizionale e let

```
if $x>0 then 1  
else if $x=0 then 0  
      else 5
```

```
      endif  
endif
```

----

```
(2x) * (2x)
```

```
let ( $double_x = $x+$x )
```

```
      in $double_x * $double_x
```

```
endlet
```

## Comprehension Term

- Se si vogliono definire liste, insiemi... condizionati

variabile Domini

guardia/condizione

termine

list  $[v_1 \text{ in } S_1, \dots, v_n \text{ in } S_n \mid G_{v_1, \dots, v_n} : t_{v_1, \dots, v_n}]$

set  $\{v_1 \text{ in } D_1, \dots, v_n \text{ in } D_n \mid G_{v_1, \dots, v_n} : t_{v_1, \dots, v_n}\}$

$\langle v_1 \text{ in } B_1, \dots, v_n \text{ in } B_n \mid G_{v_1, \dots, v_n} : t_{v_1, \dots, v_n} \rangle$

$\{v_1 \text{ in } D_1, \dots, v_n \text{ in } D_n \mid G_{v_1, \dots, v_n} : t_{v_1, \dots, v_n} \rightarrow s_{v_1, \dots, v_n}\}$

# Esempi

La lista con i numeri pari da 1 a 100

```
[ $x in [0..2*$n-1] | $x mod 2=0 : g($x)]
```

```
{ $x in {0..$n} : 2+$x }
```

```
domain Primi100 = {1..100}
```

```
domain Primi100Doppi = { $x in {1..100} : 2 * $x }
```

```
domain Pari100 = { $x in {1..10} | mod($x,2) = 0 : $x }
```

```
function listapari = [ $x in [0..100] | mod($x,2) = 0 : $x ]
```

...

# Exists/forall term

- Termini che controllano una condizione su un insieme
- resituiscono true false

Exist Term      **(exist v1 in D1,...,vn in Dn with Gv1,...,vn)**

ExistUnique Term      **(exist unique v1 in D1,...,vn in Dn with Gv1,...,vn)**

Forall Term      **(forall v1 in D1,...,vn in Dn with Gv1,...,vn)**

**(exist \$x in {2,5,7} with \$x=2)**

**(exist unique \$x in X with \$x=0)**

Esercizio: funzion che dato un insieme di interi e un intero dice se quel valore è più grande di ogni elemento

# Caso di studio - clock

- Definizione dei domini:

`definitions:`

`domain Second={0..59}`

`domain Minute={0..59}`

`domain Hour={0..59}`

Regole/transizioni di stato

# ASM transitions

In matematica le *algebre sono statiche* : non cambiano col passare del tempo.

In Informatica, gli *stati sono dinamici* : evolvono essendo aggiornati durante le computazioni.

**Aggiornare stati astratti** (*abstract states*) significa cambiare l'interpretazione delle (o solo di alcune) funzioni della segnatura della macchina.

# ASM transitions

Il modo in cui una macchina ASM aggiorna il proprio stato è descritto da **regole di transizione (transitions rules)** di una certa "forma"

L'insieme delle regole di transizione di una ASM definiscono la sintassi di un **programma ASM**

Sia  $\Sigma$  un vocabolario. Le *regole di transizione di una ASM sono espressioni sintattiche generate come segue* attraverso l'uso di **costruttori di regole**

Dichiarazioni(definizioni) di regole (`rule_declarations`)

**[macro] rule R [( x1 in D1,...,xn in Dn )] = rule**

**R = r\_Fsm**

*Esempio:*

**rule r\_Fsm(\$i in State, \$cond in Boolean,\$rule in Rule, \$j in State) =**

```
if ctl_state=$i and $cond
then par $rule
      ctl_state := $j
endpar
endif
```

*rule: una regola condizionale*

## **ID\_RULE**

una stringa che inizia con "r\_". Esempi: **r\_SetMyPerson**      **r\_update**

# Il Linguaggio Strutturale

**main rule** è:

**main rule**  $R = \text{rule}$

$R$  è il nome della main rule

la main rule è sempre una (macro-)regola **chiusa**, cioè senza parametri

**rule** è proprio il corpo della regola di transizione

Se l'ASM è multi-agent, la main rule deve far partire in parallelo i programmi degli agenti

I programmi degli agenti sono specificati nello stato iniziale (vedi **initialization**)

# Skip

- *Skip Rule:*

**skip**

**Significato:** non fare niente

# Update rule

**Update Rule:** per aggiornare lo stato della macchina

$$f(t_1, \dots, t_n) := t$$

dove:

- $f$  è un nome di funzione **dinamica**  $n$ -aria di  $\Sigma$
- $t_1, \dots, t_n$  e  $s$  sono termini di  $\Sigma$
- $t$  è un termine (un valore)

**Significato:** Nello stato successivo, il valore di  $f$  per gli argomenti  $t_1, \dots, t_n$  è aggiornato a  $t$ . Se  $f$  è 0-aria, cioè una variabile, l'aggiornamento ha la forma  $c := s$

$$f(t_1, \dots, t_n)$$

viene detta locazione L

# Il linguaggio delle regole (alcune)

**Update Rule: aggiornamenti di locazioni**

**$L := t$**

dove  $t$  è un termine e  $L$  (detta **locazione**) è o un termine funzionale  $f(t_1, \dots, t_n)$  con  $f$  dinamica e non monitorata, o è una variabile

**Significato:** Nello stato successivo, la locazione  $L$  prende valore  $t$

Esempio:

Output := 1

voto(rossi) := 30

***Nota: il risultato dell'update si vede solo dopo che sono applicati, cioè nello stato successivo***

***Esempio - demo***

# Conditional Rule

- Serve per condizionare una certa azione:

**if cond then R1 [else R2] endif**

- Dove cond è una condizione booleana, R1 e R2 sono due regole

## Il linguaggio delle regole (alcune)

Per introdurre variabili „locali“ usiamo il let

**let** ( $v_1 = t_1, \dots, v_n = t_n$ ) **in**

$R_{v_1, \dots, v_n}$

**endlet**

dove  $v_1, \dots, v_n$  sono variabili (logiche),  $t_1, \dots, t_n$  sono termini, e  $R_{v_1, \dots, v_n}$  è una regola

**Le variabili hanno valore solo per lo scope del let**

# BlockRule

Per eseguire regole in parallelo

**par R1 R2 ... Rn endpar**

dove  $R_1, R_2, \dots, R_n$  sono regole da eseguire in parallelo

## Esempio

### Par

**a(\$y) := 8**

**foo := 10**

...

# Simulazione

- Il modello di computazione è il seguente:
- Nello stato corrente valuta la main rule
  - E da quella le regole chiamate o interne alla main
- Valuta le regole che sono abilitate con i loro aggiornamenti
- Il valore delle monitorate è chiesto all'ambiente
  - Nel simulatore all'utente
- Applica tutti gli update in modo da avere lo stato successivo (nella parte controllata)

# Aggiornamenti Consistenti

- A causa del parallelismo (la regola Block e Forall), una regola di transizione può richiedere più volte l'aggiornamento di una stessa funzione per gli stessi argomenti
- si richiede in tal caso che tali aggiornamenti *siano consistenti*.

**DEF:** Un update set  $U$  è *consistente*, se vale:

if  $(f, (a_1, \dots, a_n), b) \in U$  and  $(f, (a_1, \dots, a_n), c) \in U$ ,  
then  $b = c$

**Nota che si potrebbero avere degli update incosistenti:**

**par**

**X := 1**

**X := 2**

**endpar**

# Aggiornamenti Consistenti

- Se l'update set  $U$  è consistente, allora i suoi **aggiornamenti** possono essere effettivamente eseguiti (*-fired*) in un dato stato.

Il risultato è un nuovo stato (di arrivo) dove le **interpretazioni** dei nomi **delle funzioni dinamiche** sono cambiati secondo  $U$ .

- Le **interpretazioni** dei nomi **delle funzioni statiche** sono gli stessi dello stato precedente (di partenza).

- Le **interpretazioni** dei nomi **delle funzioni monitorate** sono date dall'ambiente esterno e possono dunque cambiare in maniera arbitraria.

# Il linguaggio delle regole (alcune)

**Forallrule:** per iterare una operazione sugli elementi di un insieme:

**forall**  $v_1$  **in**  $D_1, \dots, v_n$  **in**  $D_n$   
**with**  $G_{v_1, \dots, v_n}$  **do**  $R_{v_1, \dots, v_n}$

dove  $V_i$  sono variabili,  $D_i$  termini che rappresentano domini,  
 $G_{v_1, \dots, v_n}$  termine booleano che rappresenta la condizione, e  
 $R_{v_1, \dots, v_n}$  è una regola

**Esempio:**

forall \$s in Student with voto(\$s) = 10 do voto(\$s):= 20

**Per chiamare un'altra regola**

$r [t_1, \dots, t_n]$

**dove  $r$  è il nome della regola e  $t_i$  sono termini che rappresentano gli effettivi argomenti passati**

$r[]$

**per chiamare una regola che è senza parametri**

Esempio

rule r\_1 = ...

Rule r\_2 = if c then r\_1[] endif

Nota: il passaggio dei parametri è per sostituzione. La macro viene espansa (come inline di C++)

# Rule constructors macro

- Una *definizione di regola per un nome di regola  $r$*  di arietà  $n$  è un'espressione

$$r(x_1, \dots, x_n) = R$$

dove  $R$  è una regola di transizione.

In una call rule  $r[t_1, \dots, t_n]$  le variabili  $x_i$  che occorrono nel corpo  $R$  della definizione di  $r$  vengono sostituite dai parametri  $t_i$  (*modularità*)

# Choose rule

- *Choose Rule:*

**choose**  $x$  **with**  $\varphi$  **do**  $R$

**Significato:** Esegui  $R$  in parallelo per un  $x$  che soddisfa  $\varphi$

Implementa il concetto di **non-determinismo**

# Illustrare il potere espressivo di “choose” e “forall”

**Problema:** ordinare un array  $a$

**Soluzioni possibili:**

- usare una funzione statica, ad es. `qsort`

**`a := qsort(a)`**

- iterare localmente con swap:

```
choose i,j in dom(a) with (i < j & a(i) > a(j))  
    a(i) := a(j)  
    a(j) := a(i)
```

**Nota!** *Non occorrono variabili di appoggio per lo swap, perchè i nuovi valori degli aggiornamenti saranno disponibili solo nello stato successivo!*

# Axioms

Dichiarazioni(definizioni) di assiomi (`axiom_declarations`)

**invariant [ID] over id1,...,idn : term**

- ID (opzionale) è il nome dell'assioma
- idi sono nomi di domini, funzioni\* e regole (con nome) vincolati dall'assioma
- term è un termine che rappresenta l'espressione booleana del vincolo

\*In caso di overloading di funzioni, occorre indicare anche il loro dominio, come in  $f(D)$  (o  $f()$  per funzioni 0-arie) con  $f$  nome di funzione e  $D$  nome del dominio di  $f$ .

## **ID\_AXIOM**

una stringa che inizia con "inv\_". Esempio: `inv_I1`

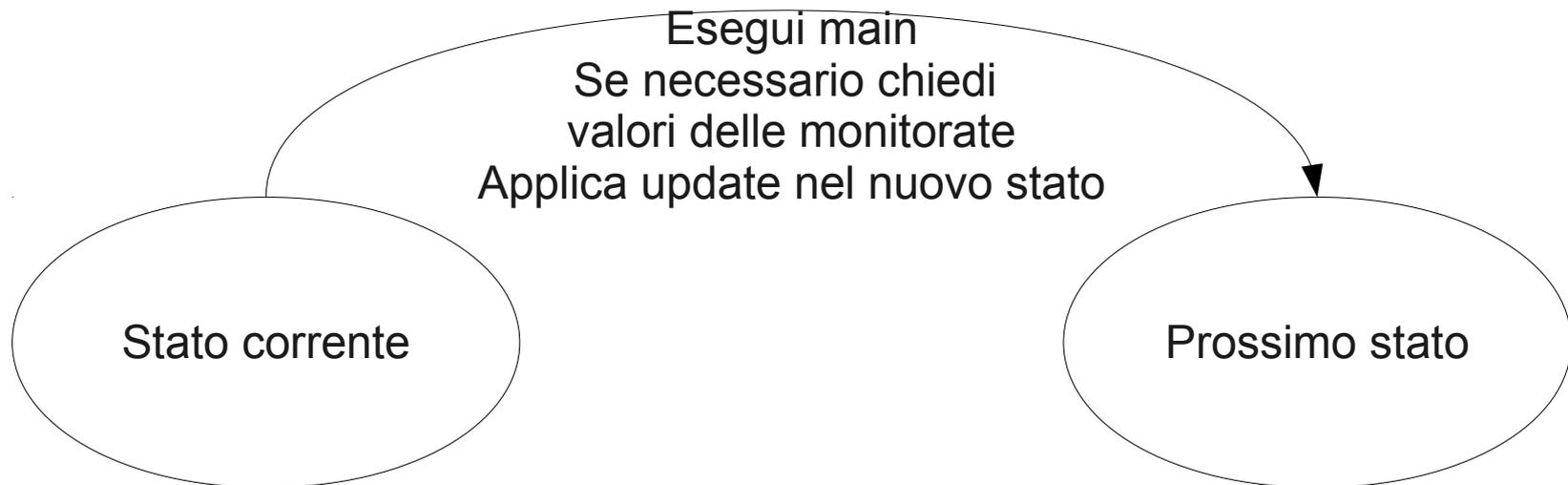
**Attenzione: prima del main e dopo le rule**

*Esempio:*

**axiom** `inv_neverBoth over high(), low(): not(high and low)`

# Modello computazionale

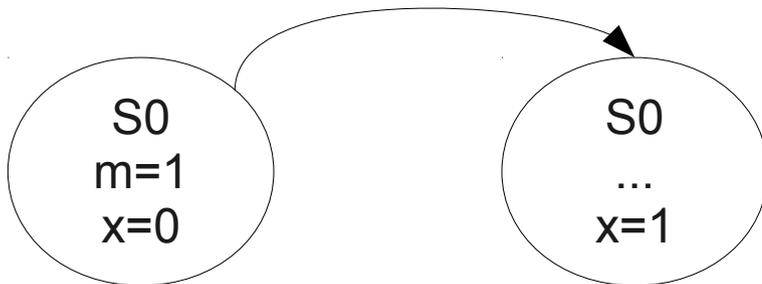
- **Ad ogni „passo“ (step) vengono lette le variabili monitorate dall'ambiente (chieste all'utente), viene eseguita la main rule e si produce un nuovo stato**
- **L'esecuzione della main rule può richiedere l'esecuzione delle sotto regole**
- **ATTENZIONE: gli update non sono immediati (tranne seq) ma sono fatti alla fine e visibili solo nel nuovo stato**



# Esempio

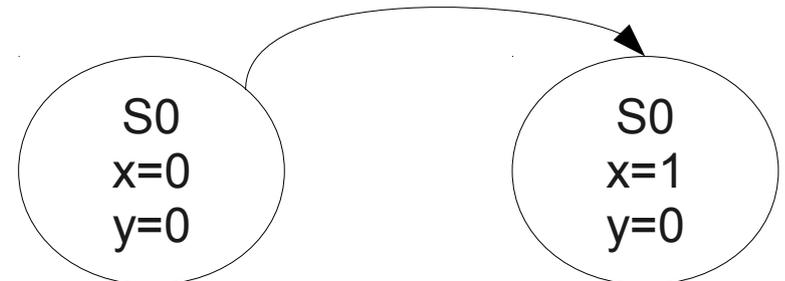
- **Le variabili monitorate vengono riportate nel prossimo stato**

```
main rule r_main =  
  if m > 0 then  
    x := 1  
  endif
```



- La variabile x **non** viene aggiornata immediatamente

```
main rule r_main =  
  if m > 0 then  
    par  
      x := x + 1  
      y := x  
    endpar  
  endif
```



# AdvancedClock

## AdvancedClock1

- *Un clock avanzato incrementa ad ogni passo i secondi (e se necessario i minuti e le ore)*

```
domain Second subsetof Integer
domain Second={0..59}
macro rule r_IncMinHours = par
main rule r_AdvancedClock =
  par
    if seconds = 59 then r_IncMinHours[] endif
    seconds := (seconds+1) mod 60
  endpar
```

# AdvancedClock

Le funzioni monitorate sono aggiornate dall'ambiente

L'ambiente può essere

- Un file
- Lo standard input

## AdvancedClock2

- *Come AdvancedClock ma c'è una funzione monitorata signal che incrementa i secondi*

# Sequential computation

Alcune volte vogliamo applicare subito l'update sete valutare il resto delle regole immediatamente:

**seq R1 R2 ... Rn endseq**

**dove R1,R2,...,Rn sono regole da eseguire in sequenza**

# Seq Rule

L'interpre supporta la regola seq

## SeqRule01

```
main rule r_main =  
  seq  
    f := 10  
    g := 5 + f  
    f := 33  
  endseq
```

- All'inizio di ciascun passo
- f è 33
- g è 15 !

**Stato iniziale**

# Il Linguaggio Strutturale

**Initialization** è una sequenza di stati iniziali:

[**default**] **init** Id :

**domain** Dd1 = Dterm11

...

**function** Fd1[(p11 **in** d11,...,p1s1 **in** d1s1)] = Ftermd1

...

Uno stato iniziale deve essere denotato come *default*.

Solo **domini concreti dinamici** possono essere inizializzate

Solo **funzioni dinamiche**, non monitorate, possono essere inizializzate

# La tesi ASM

*The ASM thesis is that any algorithm can be modeled at its natural abstraction level by an appropriate ASM. (Gurevich, 1985)*

Sequential thesis:

Sequential ASMs capture sequential algorithms. (Gurevich, 2000)

Parallel thesis:

ASMs capture parallel algorithms. (Blass/Gurevich, 2003)

... ?

# ASM Distribute (o multi-agenti)

- Agenti computazionali
- Stati globali condivisi tra gli agenti
- Movimenti concorrenti sincroni/asincroni

Una *sync/async ASM* è una famiglia di coppie  $(a, ASM(a))$  di

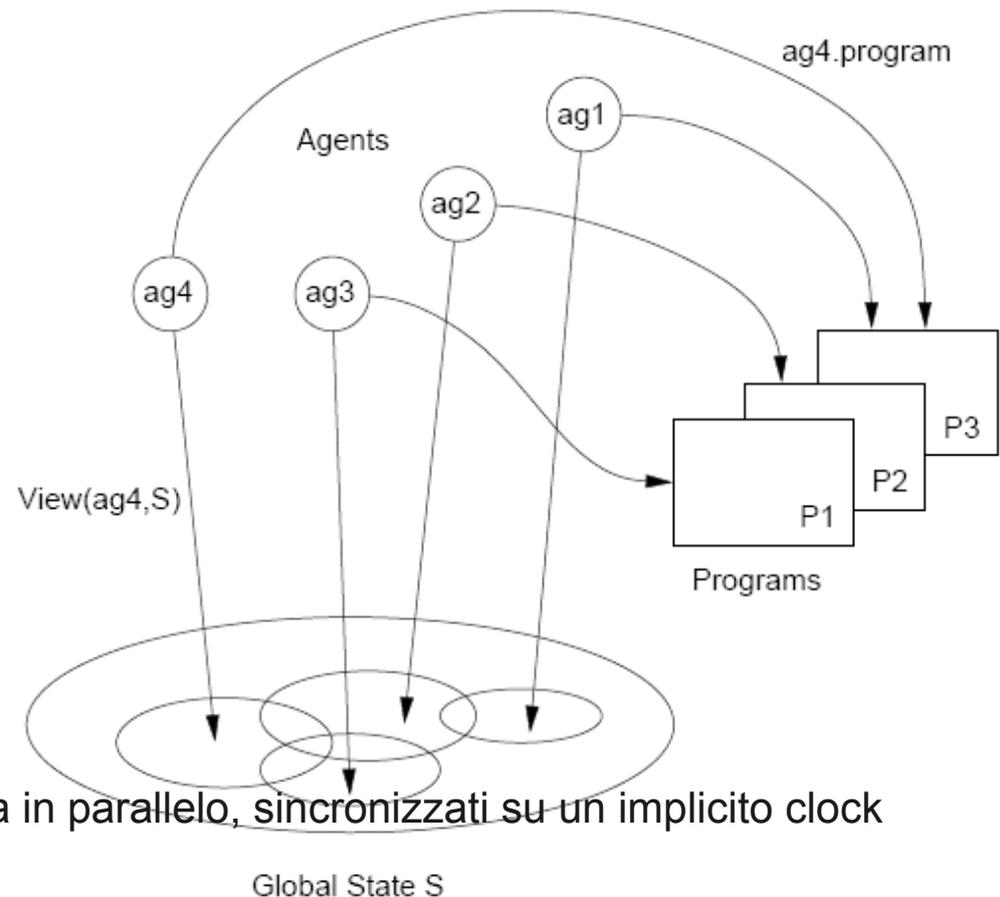
- agenti  $a \in Agent$  (il dominio degli agenti)
- programmi ASM  $(a)$  di ASM di base (sequenziali)
- $f(self,x)$  per una  $f : A \rightarrow B$ , denota la *versione privata di  $f(x)$*  dell'agente corrente  $self$ .

La dichiarazione di  $f$  diventa  $f : Agent \times A \rightarrow B$

In una *sync ASM* gli agenti eseguono il loro programma in parallelo, sincronizzati su un implicito clock globale del sistema.

Asynchronous computation model (Gurevich, 1995)

*Semantic model resolves potential conflicts according to the definition of *partially ordered runs**



- **ASM Web Site** <http://www/eecs.umich.edu/gasm>  
<http://www.di.unipi.it/~boerger>
- **Abstract State Machines Research Center**  
<http://rotor.di.unipi.it/AsmCenter/Lists/AboutLinks/AllItems.aspx>
- **Libro ASM** E. Boerger and R. Staerk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Verlag, 2003. <http://www.di.unipi.it/AsmBook/>
- **ASM Survey** E. Börger High Level System Design and Analysis using ASMs LNCS Vol. 1012 (1999), pp. 1-43
- **ASM History** E. Börger The Origins and the Development of the ASM Method for High Level System Design and Analysis. J. Universal Computer Science 8 (1) 2002
- **Original ASM Definition** Y. Gurevich Evolving algebra 1993: Lipari guide. Specification and Validation Methods. (Ed. E. Börger) OUP 1995
- **Libro sul caso di studio Java-ASM** R. Stärk, J. Schmid, E. Börger. Java and the Java Virtual Machine: Definition, Verification, Validation. Springer-Verlag 2001. <http://www.inf.ethz.ch/~jbook>

# Orologio

- Monitorato:
  - Segnale booleano
- Controllato:
  - Secondi, minuti ed ore
- Comportamento:
  - Se segnale è vero incrementa i secondi (e se necessario minuti e ore in modo corretto)