Abstract State Machines
Info 3 19/20
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

Materiale

- Queste slides
- manuali d'uso di asmeta
  - http://fmse.di.unimi.it/asmeta/download/AsmetaL_guide.pdf
- Tools:
  - http://asmeta.sourceforge.net/

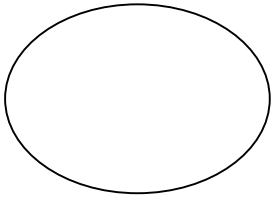
Idea guida

- 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 (che non vedremo)
- Idee guida

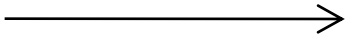
Asm come estensione delle FSM

FSM

- Insieme (finito) di stati



- Collegati da transizioni

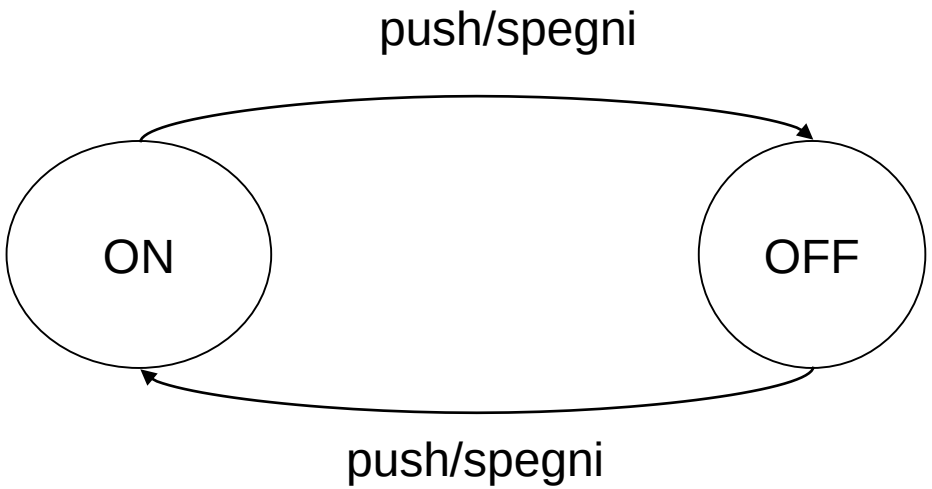


- Con condizioni e azioni sulle transizioni
  - Insieme finito di input e di azioni

cond/action

Esempio : interruttore di luce

- Un interruttore con un solo bottone che quando viene premuto (push) invia il comando alla lampada di accendersi o spegnersi



Due soli stati: On e Off
Un solo input push
Due output spegni e accendi

Macchina a Stati Finiti (1)

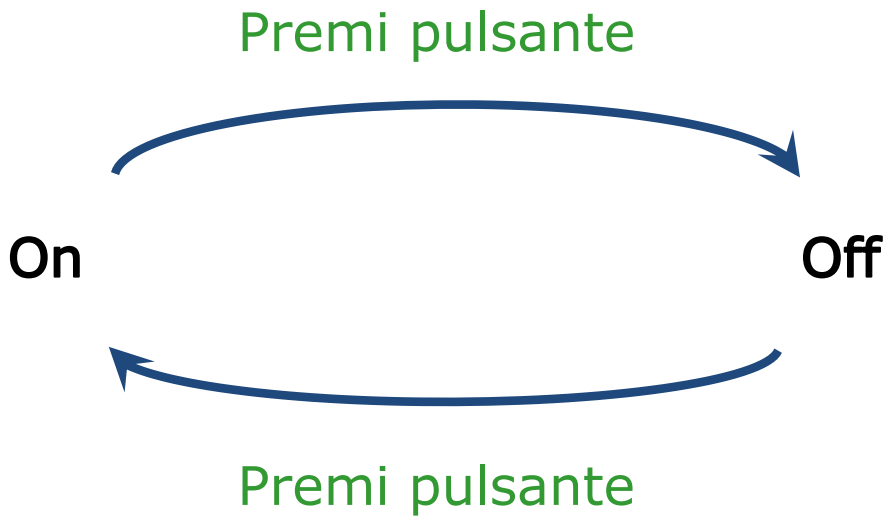
Una macchina a stati finiti, abbreviata FSM (Finite State Machine), è una notazione formale che permette la rappresentazione astratta del comportamento di un sistema

Le FSM hanno:

- una rigorosa definizione matematica
- una intuitiva rappresentazione grafica tramite diagrammi di stato

FSM: un primo esempio

Esempio: comportamento di una lampadina



- I nodi rappresentano gli stati del sistema
- Gli archi rappresentano il passaggio di stato

Estensioni di FSM

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

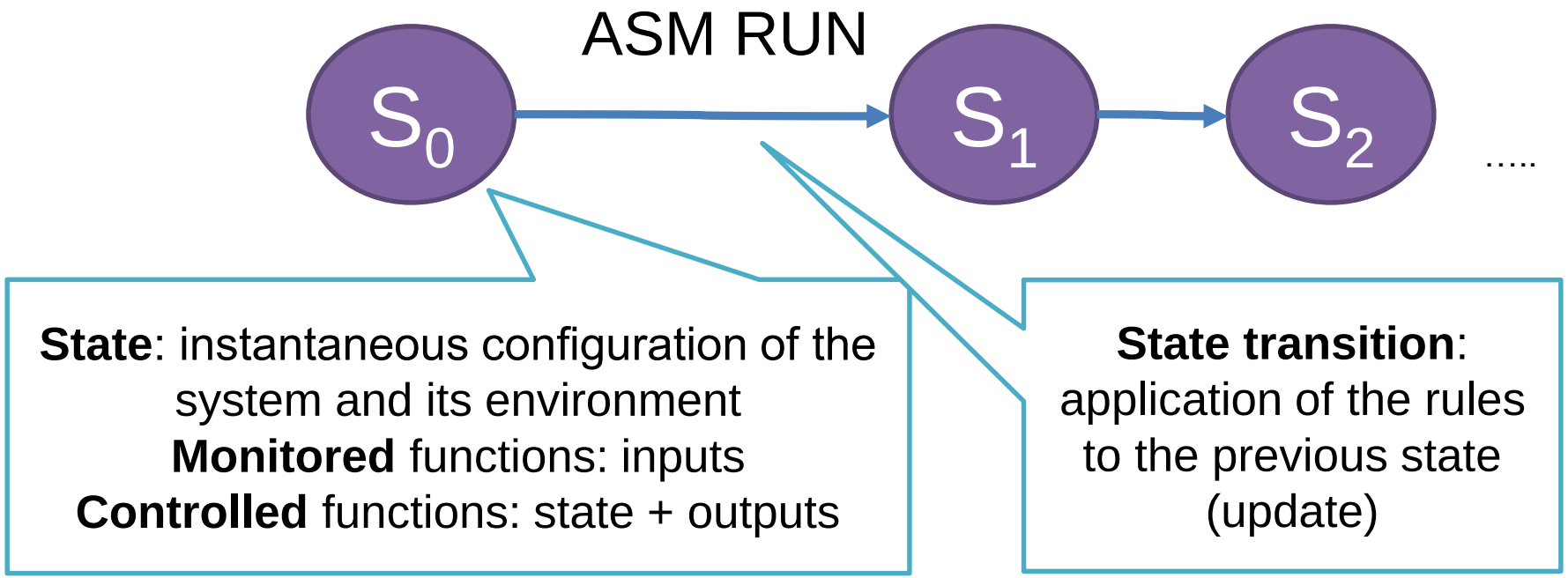
- **FSM con evento di output** (ad es. un'azione)
  - macchine di Mealy (output sulla transizione)
  - macchina di Moore (output nello stato)
- **FSM con variabili** (rappresentano la memoria interna della macchina)
- **le Statecharts di UML** dotate dei concetti di sottomacchina (modularità) e composizione 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
      - che però stanno nello stato
      - la transizione dipende solo dallo stato corrente

Abstract State Machines

- ASMs are an extension of Finite State Machine
  - unstructured control states are replaced by states with arbitrary complex data.
  - Transition rules describe the change of state.



Modello computazionale - brief

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

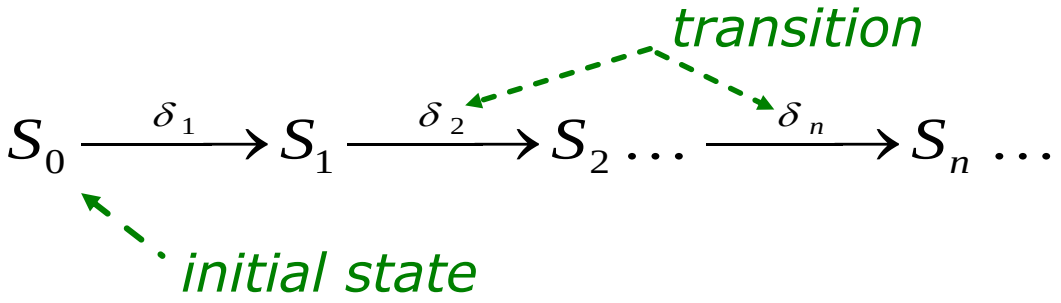
Modello Computazionale *Vocabolario o segnatura*

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

Azioni: transizioni di stato

if *Cond* **then** *Updates*

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



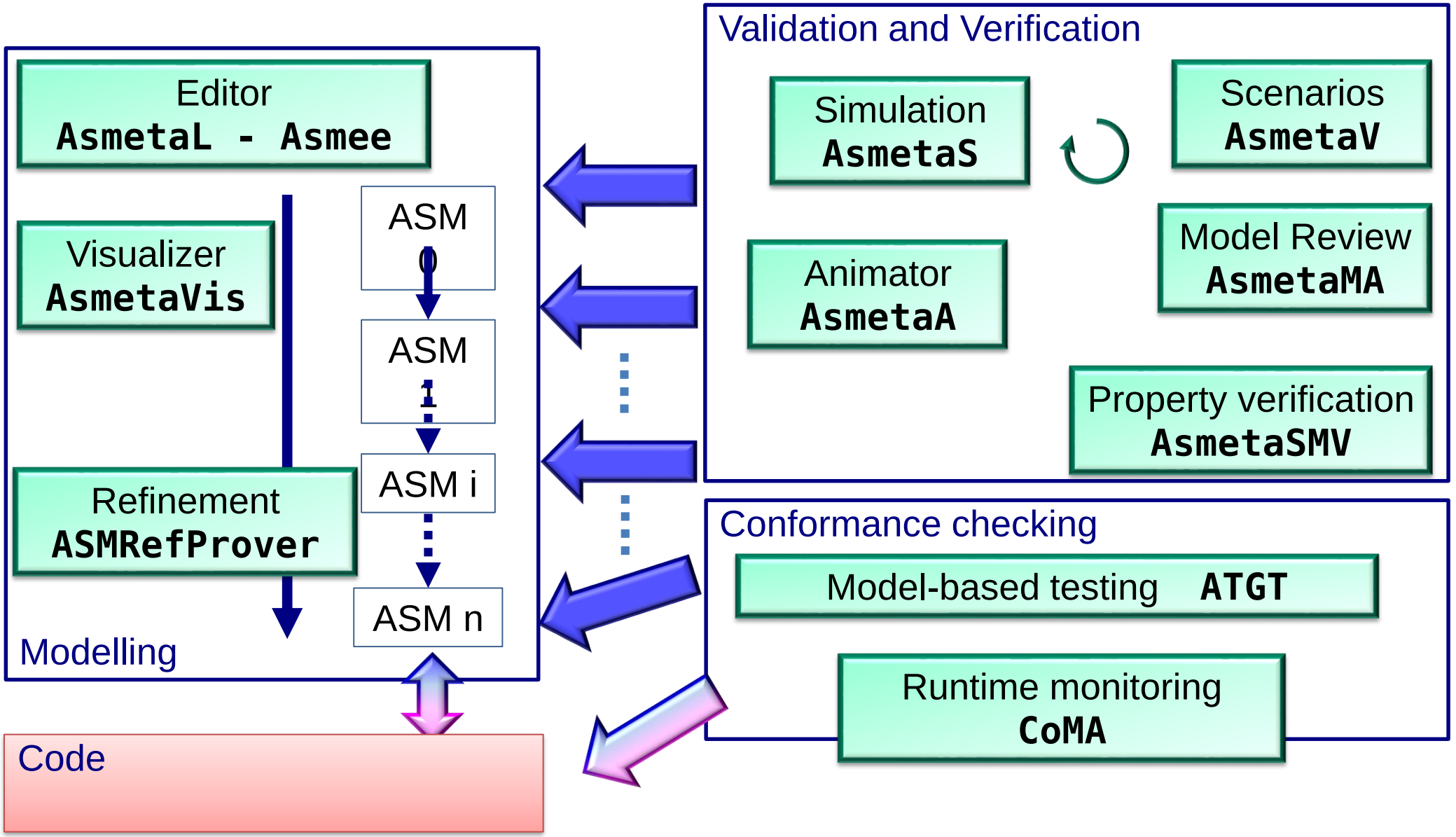
Programmi (istruzioni di aggiornamento)

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

Asmeta

- Useremo Asmeta come tool per le ASM:
- Ha un linguaggio/editor
  - AsmetaL
- Un simulatore e animatore
  - AmsetaS
- Un ambiente eclipse
  - Asmee
- Un linguaggio per scaneri
  - Avalla
- Si installa dall'update site
- **http://svn.code.sf.net/p/asmeta/code/code/stable/asmeta_update/**

ASMETA framework



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

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
- 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
- Model advisor
  - Per trovare difetti (analisi statica)
- Visualizzatore
- Model checker
  - Basato su NuSMV
- Tests generator
  - ASM Tests Generation Tool (ATGT)

Sul linguaggio

- The EBNF grammar for the AsmM textual notation.
- A quick guide of the concrete notation
- A more complete guide about the language
- several examples of ASM specs
  - Cerca sotto sf la directory esempi
- The AsmM Standard Library A library of predefined ASM domains and functions: StandardLibrary.asm.(you need this)

Standard Library

Operatori aritmetici
- +, -, *, /, %

Operatori relazionali
- =, !=, <, >, <=, >=

Operatori booleanioolean
- 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 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
  - anche dall'interfaccia di Asmee
- 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 lancial'eccezione InvalidAxiomException che tiene traccia dell'aasioma 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 **E**clipse Environment

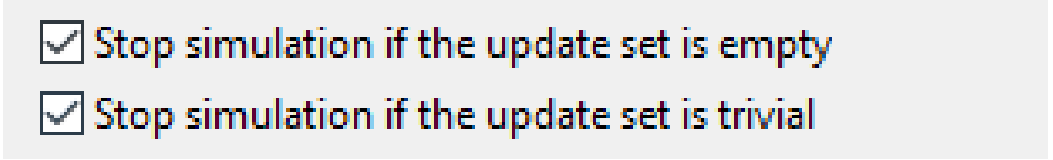
- Caratteristiche disponibili in ambiente Eclipse:
- un wizard per **creare un nuovo file AsmetaL**:
  - File->new File-> Other -> AmsetaL new File
- **synthax hightlighting**
- I colori predefiniti possono anche essere modificati:
  - Window -> Preferences -> Asmee

Azioni per simulare con AsmetaS un file AsmetaL:

- **check** la tua specifica AsmetaL 

Simulazione in asmee

- Per simulare una ASM usa:
  - Simulazione interattiva (domanda gli input all'utente): 
  - Simulazione random (scegli valori a caso per input): 
- Puoi fermare la simulazione al verificarsi di due condizioni:



Oppure usa 

Noi useremo però l'animatore principalmente

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

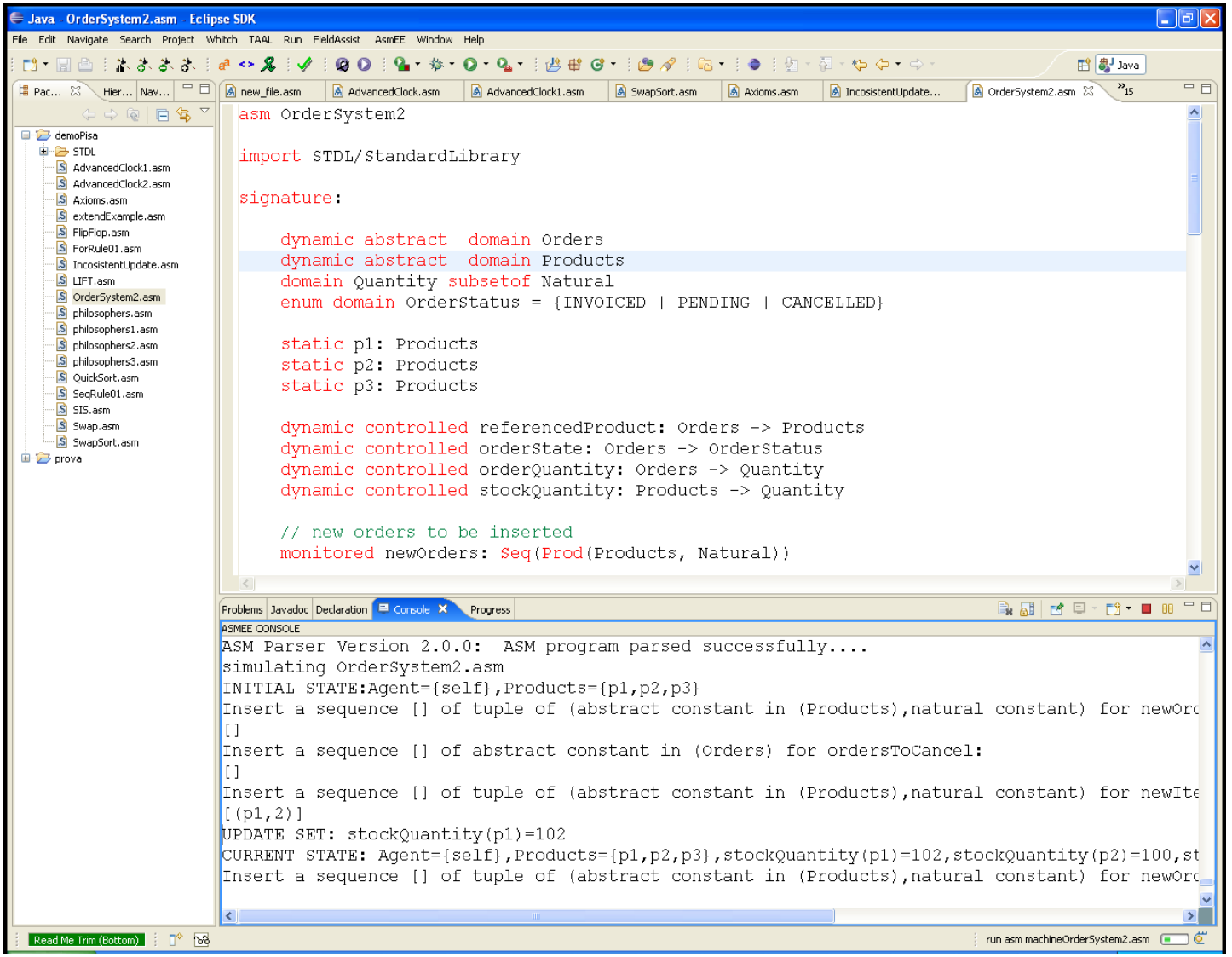
- Ogni utente può personalizzare il suo log (per scopi di debugging di specifiche ASM e per l'interprete)

JUnit per lo unit testing

Emma per la valutazione della copertura del codice "code coverage"

L'IDE ASMEE

ASMETA **E**clipse 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(t1,t2,..,tn)*)
  - ***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

Body section

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

Initialization section

Inizializzazioni per domini e funzioni dinamici

asm flip_flop
import StandardLibrary
signature: ....

definitions: ....

default init initial_state:
.....

Nome deve coincidere con il nome del file

Qui ci va la definizione di stato

Qui ci va la definizione delle regole

Il Linguaggio delle definizioni

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 OO sono i campi dell'oggetto
- in ASM si chiamano **funzioni**
- distinguiamo la cardinalità delle funzioni
  - Variabili e costanti (0-arie) x, y , …
  - Mappe/array/funzioni n-arie
    - Es: name(1), name(2), ...

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 (come le funzioni in scala)

  - Funzioni statiche di arietà zero sono dette costanti
  - Funzioni dinamiche di arietà 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

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

Domini

ASM Domini

- I soliti domini predefiniti sono disponibili
  - Interi, String
    - Da importare dalla StandardLibrary
- 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 >=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

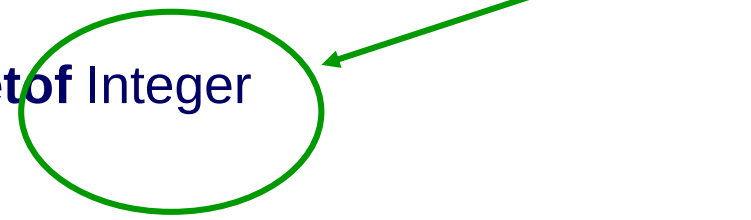
[**dynamic**] **domain** D **subsetof** td
dove:

- 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** Integer

definitions:
domain State = {0,1}



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 sostituscono 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
- Le stringhe sono costanti, esempio “pippo”

ASM Function Classification

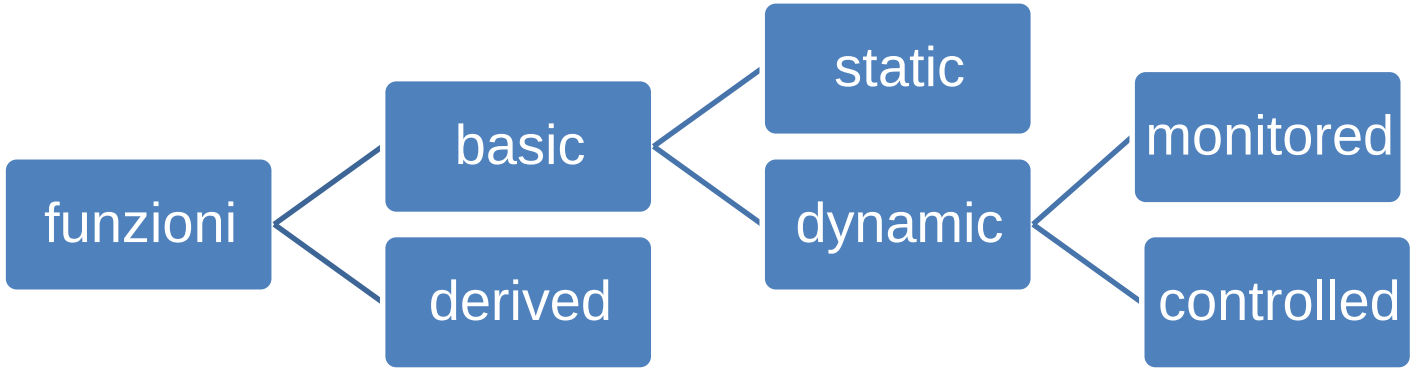
Le funzioni dinamiche unarie corrispondono alle variabili

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

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

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

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



Realizza l'incapsulamento/information hiding dei linguaggi OO

Controlled e monitored

- **Controlled**: rappresentano quelle che la macchina modifica
- **Monitored**: quelle modificati dall'ambiente esterno (in simulazione andranno domandate all'utente o lette da un file esterno)

- ASM: particolarmente adatte per sistemi reattivi

Definizione delle funzioni in AsmetaL

Dichiarazioni di funzioni 0-arie

Funzioni **statiche**	static **f: C**
Funzioni **dinamiche**	[dynamic] monitored **f**: **C** [dynamic] controlled **f**: **C** [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 di parla di location si può pensare all'indicizzazione di una cella della tabella

Funzione dinamica n-aria

- Esempio
- voto: Studenti-> interi

Funzione:
Intera tebella

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**
Funzioni dinamiche	[dynamic] monitored **f**: [**D** ->] **C** [dynamic] controlled **f**: [**D** ->] **C** [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 una 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 gesitice più ascensori
- Ogni ascensore può avere due direzioni, e può essere ferma o in movimento, e può essere in un certo piano

```
abstract domain Lift
enum domain Direction = {UP | DOWN }
enum domain State = {MOVING | HALTED }
dynamic controlled direction : Lift -> Direction
dynamic controlled state : 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:
  - mymax(x,y)= if x > y then return x else return y
- In AsmetaL dobbiamo definire la fuzione 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

Idem le funzioni derivate

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
      - exist $x in Student with voto($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	[t1,...,tn] **con** ti **termini della stessa natura** [] **per la sequenza vuota**
Set	{t1,...,tn} **con** ti **termini della stessa natura** {} **per l'insieme vuoto**
Bag	<t1,...,tn> **con** ti **termini della stessa natura** <> **per il bag vuoto**
Map	{t1->s1,...,tn->sn} **con** ti **termini della stessa natura, e** si **termini della stessa natura pure** {->} **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] ≡ [1,2,3,4]
Set	{[],[1,2],[1]} {'a','b'} {1..2,0.5}≡{1.0,1.5,2.00}
Bag	<1,2,1> <'a','b','a','b'> <1..10,2> ≡ <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	[v1 in S1,...,vn in Sn | Gv1,...,vn : tv1,...,vn]
set	{v1 in D1,...,vn in Dn | Gv1,...,vn : tv1,...,vn}
	<v1 in B1,...,vn in Bn | Gv1,...,vn : tv1,...,vn>
	{v1 in D1,...,vn in Dn | Gv1,...,vn : tv1,...,vn ->sv1,...,vn}

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 ASMdefiniscono la sintassi di un programma ASM
Sia *Σ un* vocabolario. Le *regole di transizione di una ASM sono espressioni sintattiche generate come segue* attraverso l'uso di costruttori di regole

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 **=** 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, . . . , *tn*) := t**

dove:

– *f* è un nome di funzione **dinamica** *n*-aria di *Σ*

– $t_1, \ldots, t_n$ *e s* sono termini di *Σ*

- t è un termine (un valore

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

***f* (*t*1, . . . , *tn*)**

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(t1,..,tn) 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 (v1 **=** t1**, ..., ** vn **=** tn) **in**
 Rv1,...,vn
endlet

dove v1,...,vn **sono variabili (logiche),** t1,...,tn **sono termini, e** Rv1,...,vn **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 R1,R2,...,Rn **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 $(f, (a1, \ldots, an), b) \in U$ *and* $(f, (a1, \ldots, an), c) \in U$,
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 v1 **in** D1**, ..., ** vn **in** Dn
with Gv1,...,vn **do** Rv1,...,vn

dove vi sono variabili, Di termini che rappresentano domini, Gv1,...,vn termine booleano che rappresenta la condizione, e Rv1,...,vn è una regola

Esempio:

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

Macro call rule

Per chiamare un'altra regola

r [t1,...,tn]

dove r **è il nome della regola e** ti **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*

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

dove *R* è una regola di transizione.
In una call rule $r\ [t_1, \ldots, t_n]$ 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** φ **do** *R*
  Significato: Esegui *R* in parallelo per un *x* che soddisfa φ
  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`

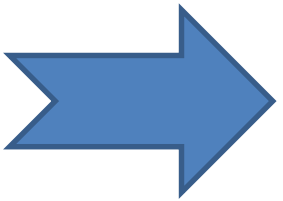
Esempio:

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 porduce 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**

Stato corrente

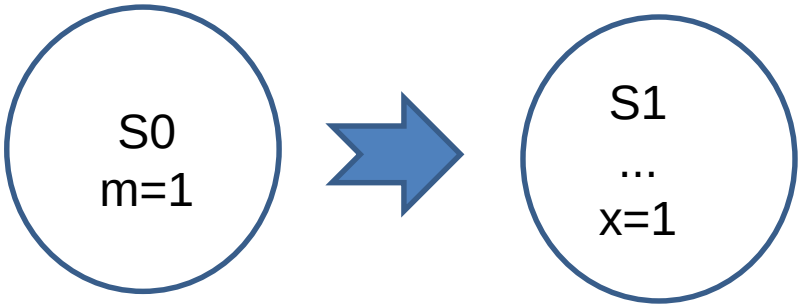


Prossimo stato

Esempio

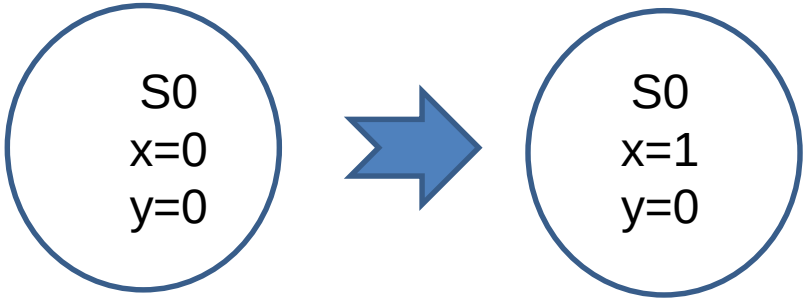
- **Le variabili controllate vengono aggiornate nel prossimo stato**

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



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

```
main rule r_main =
     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*

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)

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

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)

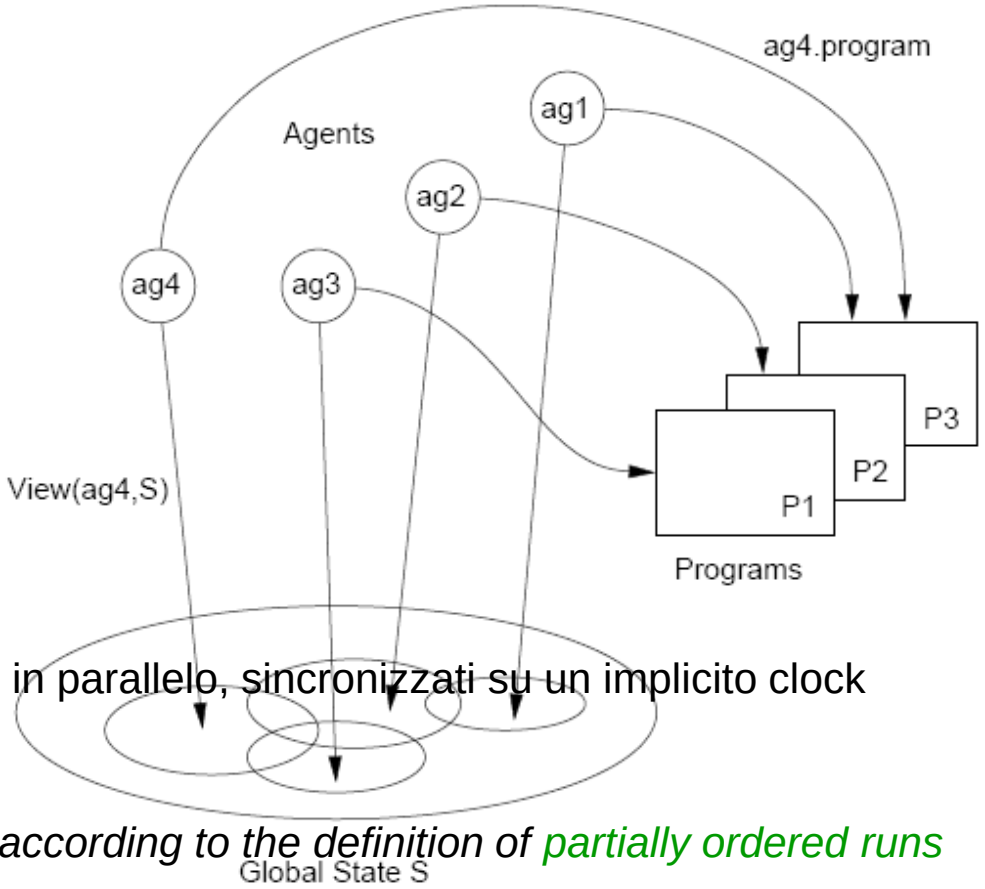
… ?

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 ∈ Agent

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

Riferimenti bibliografici

- 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