

Abstract State Machines
Testing e verifica del sw 18/19
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

- Info 3 +
 - Model checker
 - Validatore basato su scenari
 - Model advisor

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

Modello computazionale - brief

ASM = Abstract State + *virtual* Machine

[Gurevich'95/'99]

Modello Computazionale

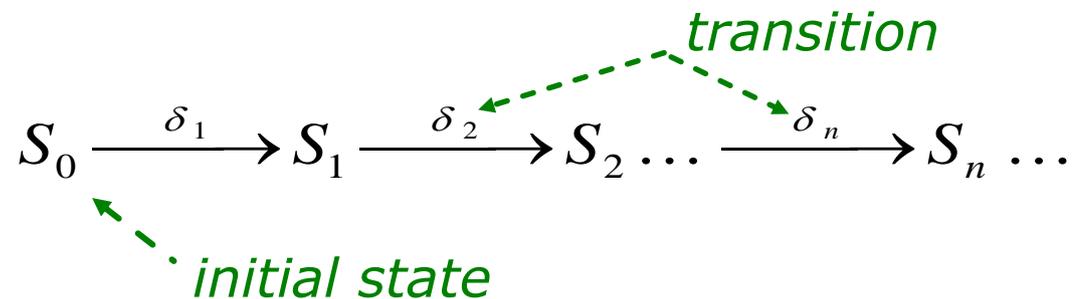
Vocabolario o segnatura

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

Azioni: transizioni di stato

if Cond then Updates

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



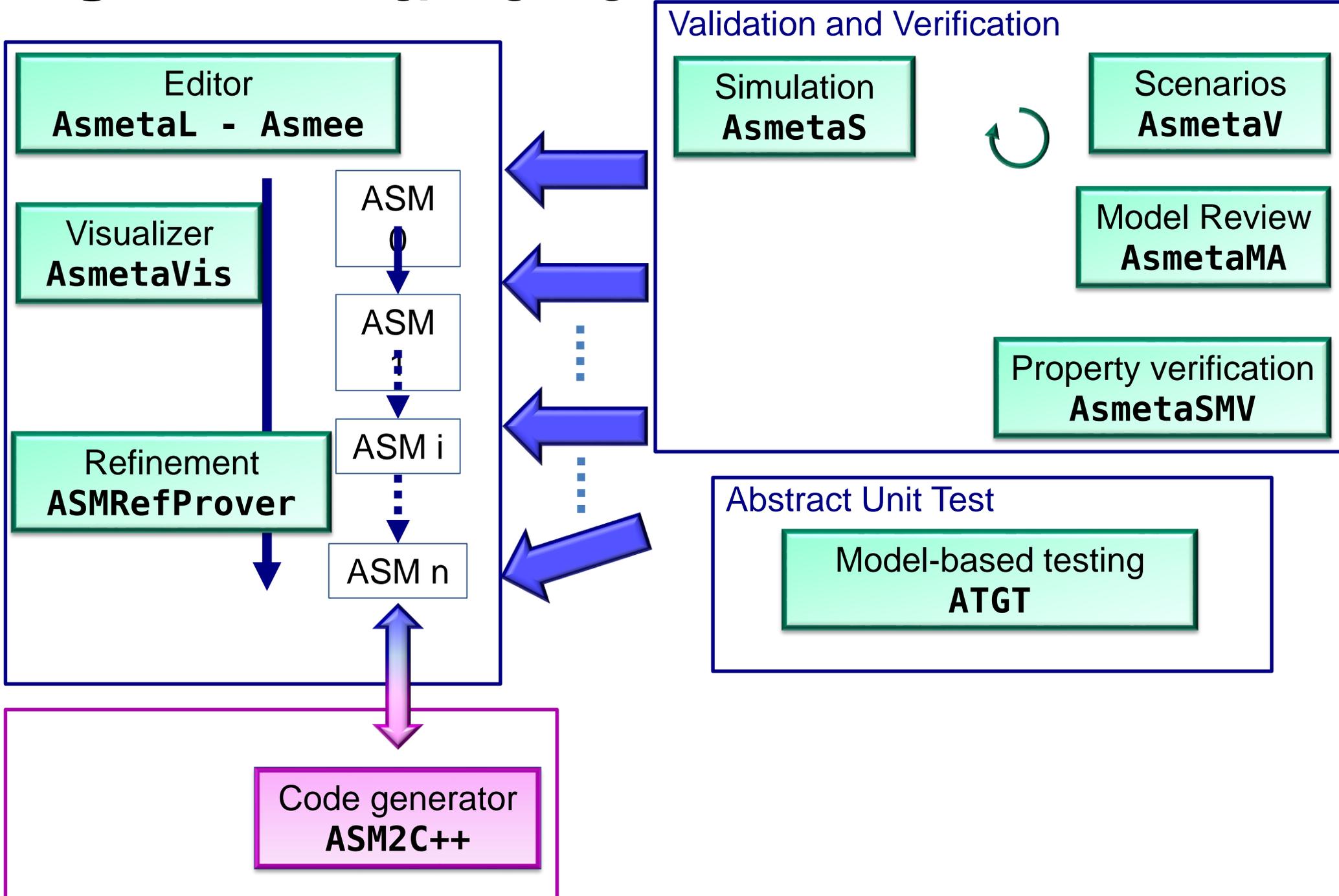
Programmi (istruzioni di aggiornamento)

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

Asmeta

- Useremo Asmeta come tool per le ASM:
- Ha un linguaggio/editor
 - AsmetaL
- Un simulatore
 - AsmetaS
- Un ambiente eclipse
 - Asmee
- Un model checker
 - AsmetaSMV
- Scenario validation – Avalla
- Model Advisor

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

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**
- 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**)

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 "pippo"

ASM Function Classification

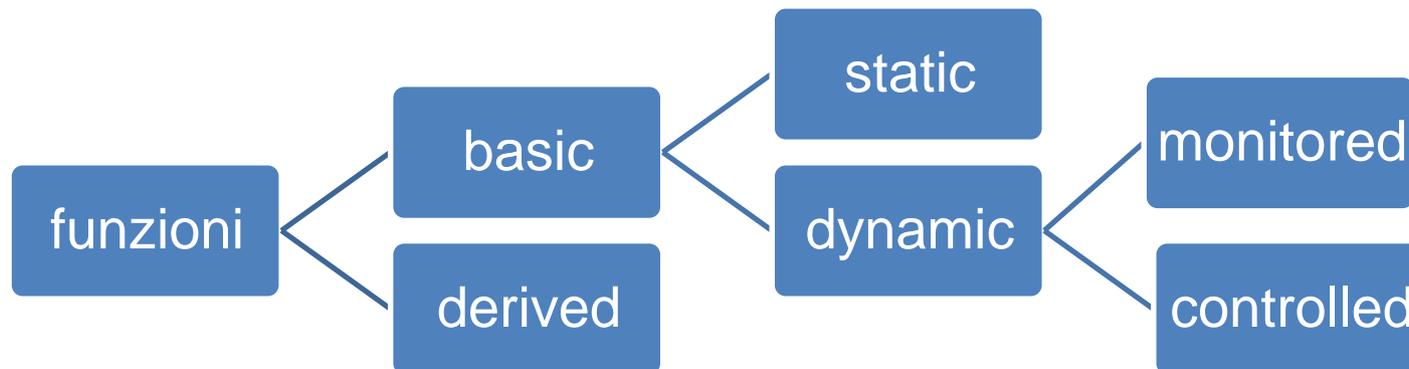
Le funzioni dinamiche unarie corrispondono alle variabili

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

Non fanno parte dello stato che cambia (interpretazione è fissa)

Esempio di funzioni statiche sono le usuali operazioni tra numeri

$+$, $-$, \dots

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

Studente	Voto
verdi	18
...	...
Rossi	30

locazione

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	<code>static f: [D ->] C</code>
Funzioni dinamiche	<code>[dynamic] monitored f: [D ->] C</code> <code>[dynamic] controlled f: [D ->] C</code> <code>[dynamic] shared f: [D ->] C</code> <code>[dynamic] out f: [D ->] C</code> <code>[dynamic] local f: [D ->] C</code>

- 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, 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 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

Variabili 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 restituisce 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)
```

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
```

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 Σ 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 Σ
- t_1, \dots, 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, \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 (v1 = t1, ..., vn = tn) in  
  Rv1,...,vn  
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 $r_1 = \dots$

rule $r_2 = \text{if } c \text{ then } r_1[] \text{ endif}$

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** φ **do** R

Significato: Esegui R in parallelo per un x che soddisfa φ
Implementa il concetto di **non-determinismo**

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

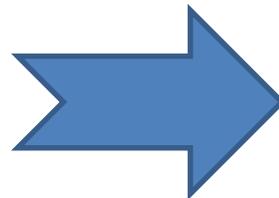
Esempio:

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**

Stato corrente

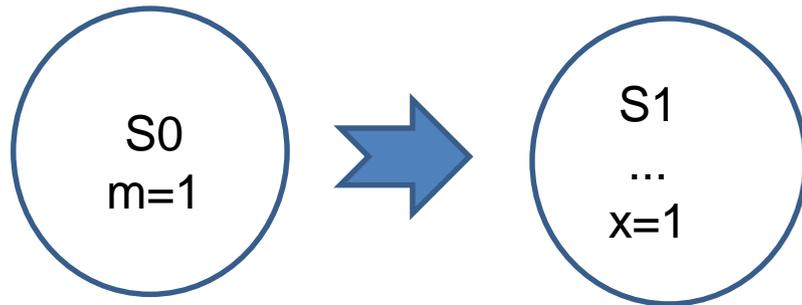


Prossimo stato

Esempio

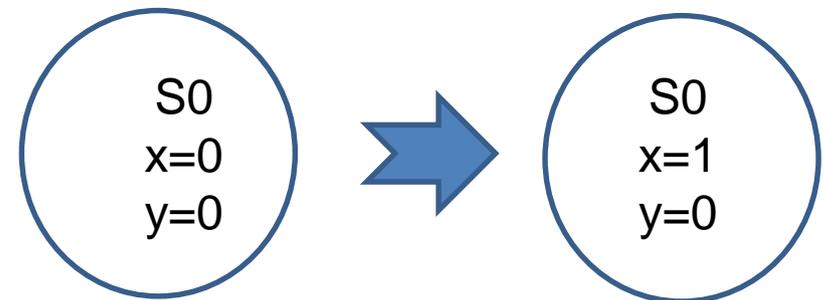
- **Le variabili controllate vengono aggiornate nel prossimo stato**

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



- La variabile 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

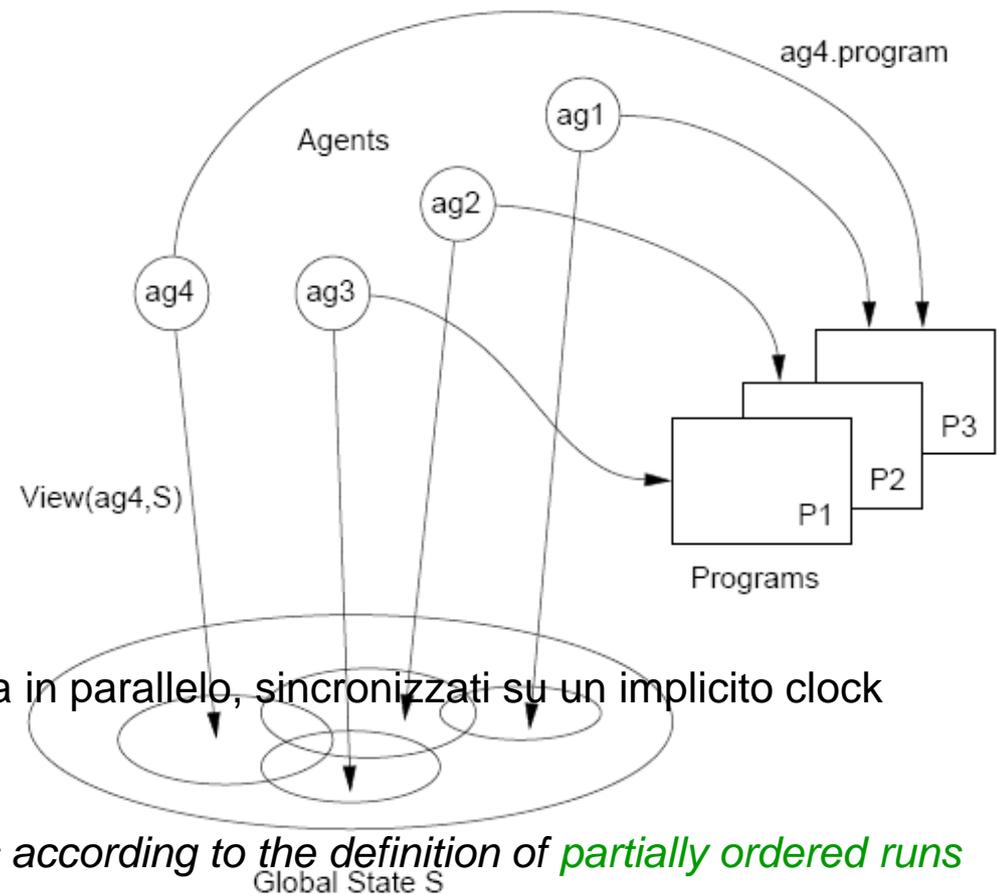
Una *sync/async ASM* è una famiglia di coppie $(a, ASM(a))$ di agenti $a \in Agent$

- (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>