Abstract State Machines
Info 3 2008/09
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 asupporto
  - asmeta.sf.net

Materiale

- Queste slides
- manuali d'uso di asmeta
- Sito web di egon boerger

Idee guida

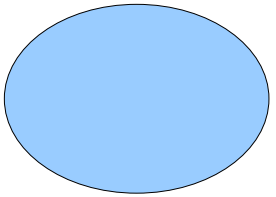
ASM = FSM con stati generalizzati

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

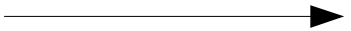
- Ground Model (descrizioni formali)
- Raffinamenti

FSM

- Insieme (finito) di stati



- Collegati da transizioni

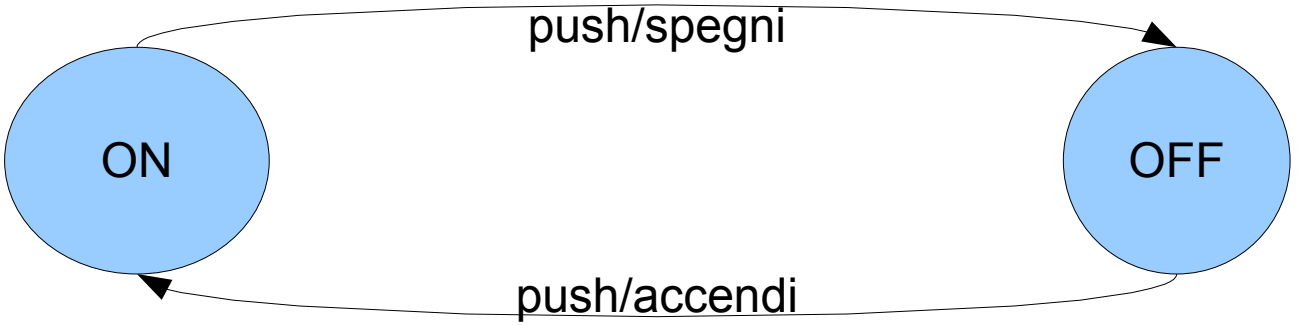


- Con condizioni e azioni sulle transizioni
  - Insiem finito di input e di azioni

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)

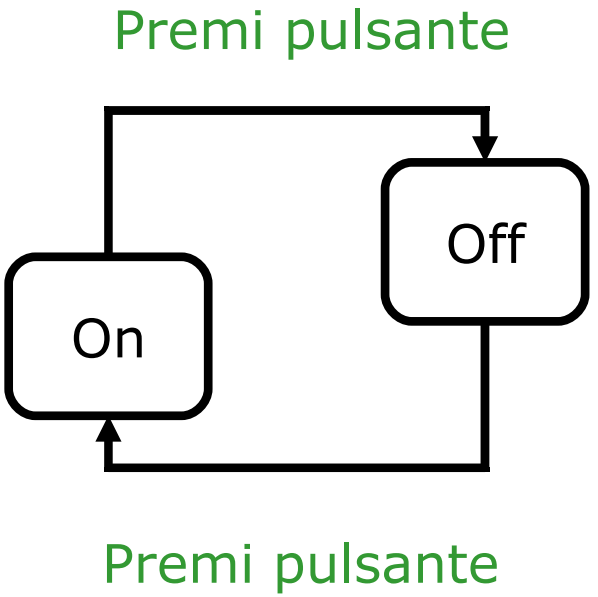
Una macchina a stati finiti, abbreviata FSM (Finite State Machine), è una notazione formale che permette la rappresentazione astratta del comportamento di un sistema

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

Macchina a Stati Finiti (2)

Dall'esempio si evince che le FSM modellano le configurazioni istantanee di un sistema tramite stati (*nodi* del diagramma) e le operazioni del sistema tramite transizioni (*archi* del diagramma)

Le operazioni possono ricevere input e produrre output

Modellare con FSMs

Sono utilizzate per modellare

- GUIs
- Protocolli di rete
- Pacemakers e teller machines
- Applicazioni WEB
- Software di sicurezza
- Sistemi embedded

Modellare con FSMs: limiti

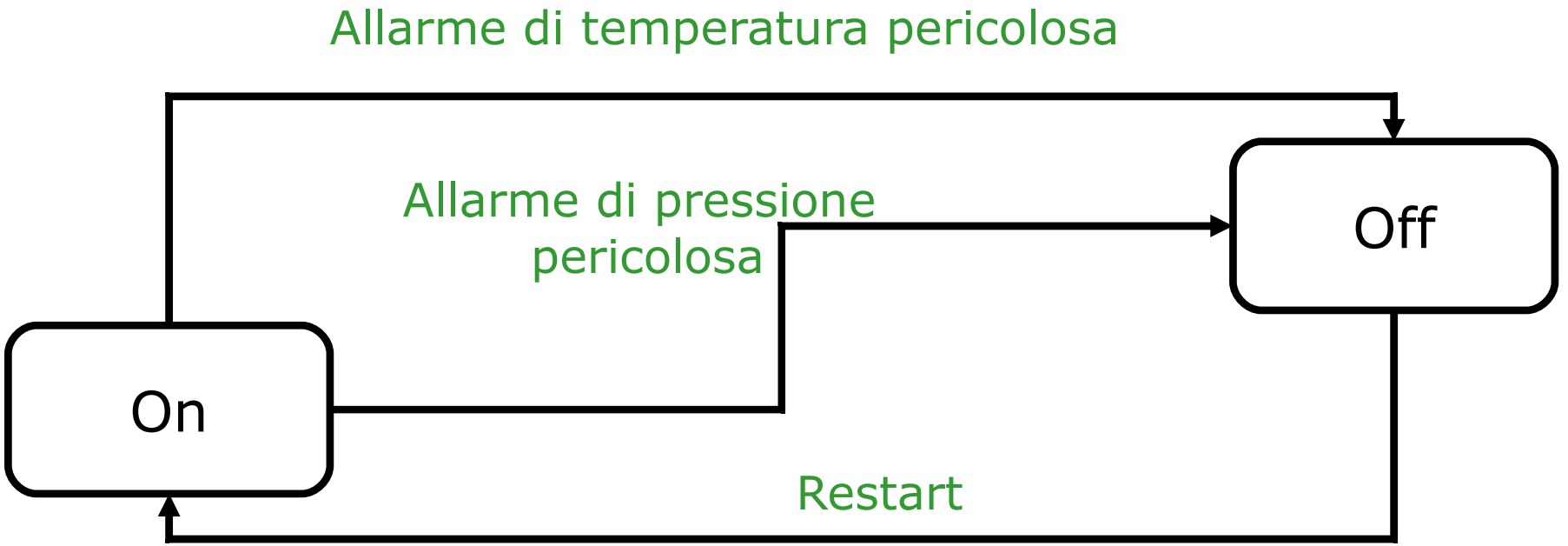
Non tutti i requisiti possono essere specificati con una FSM

Tra i requisiti non specificabili in FSM:

- requisiti real time
- requisiti riguardanti performance
- requisiti riguardanti tipi di computazioni

Esempio di una FSM

Sistema di controllo di impianto chimico



Modellare con FSM

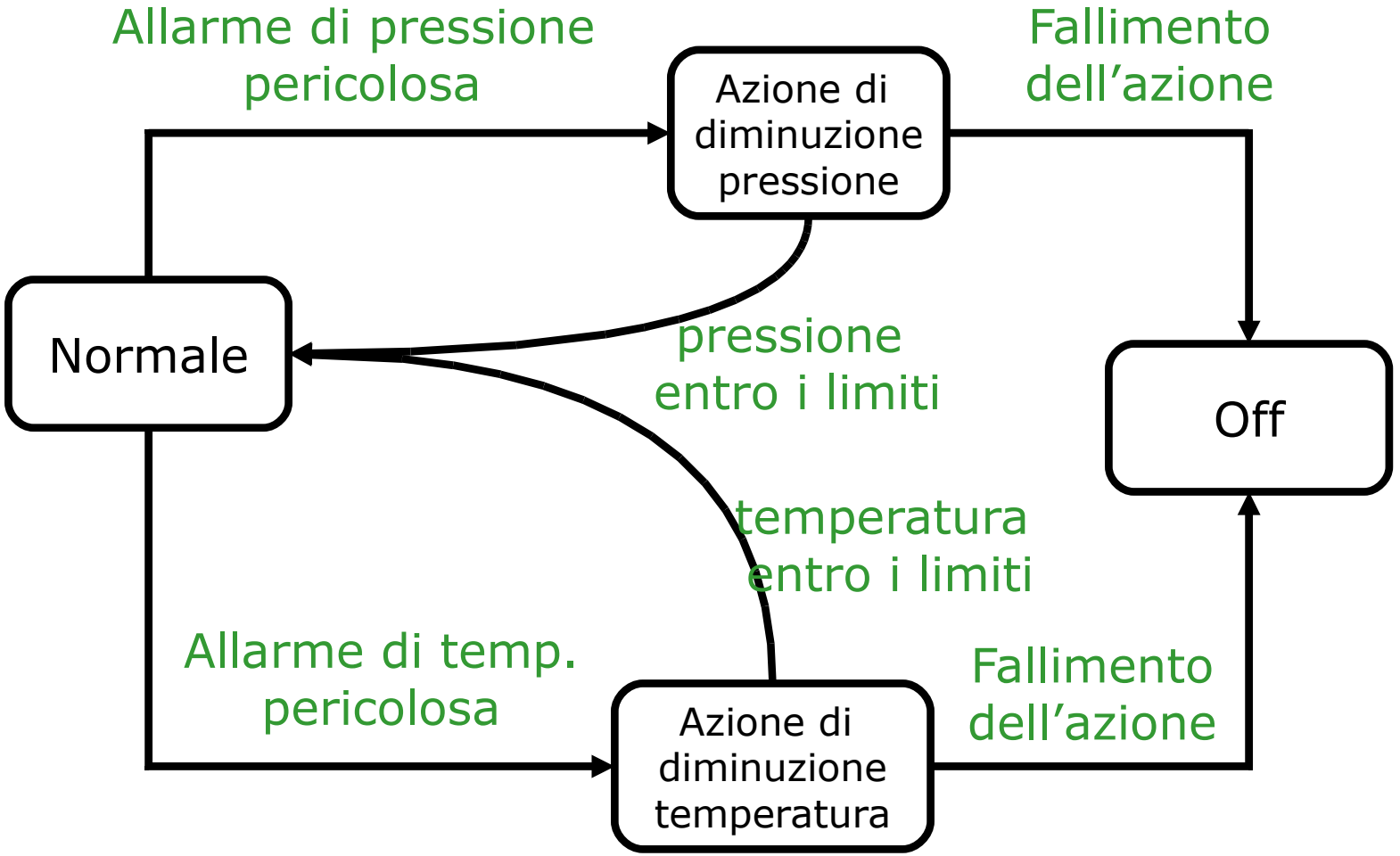
Il modello FSM di un sistema non è unico

Modelli diversi in base al livello di astrazione/dettaglio considerato

- Confronta il modello FSM del sistema di controllo di un impianto chimico con il modello successivo che considera un diverso livello di astrazione

Esempio di una FSM

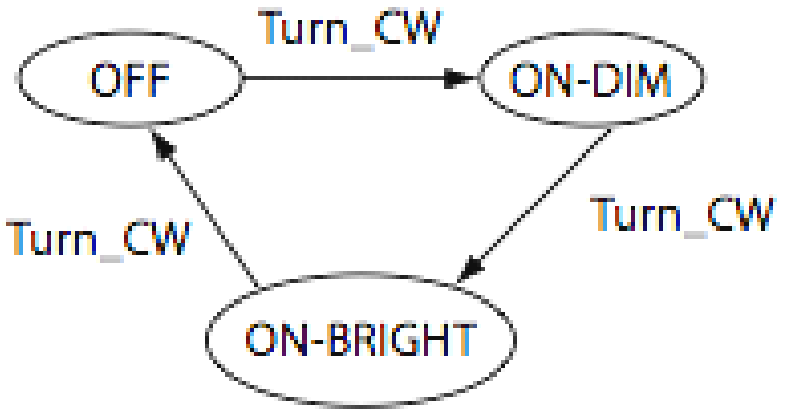
Sistema di controllo di impianto chimico: un livello di astrazione più basso



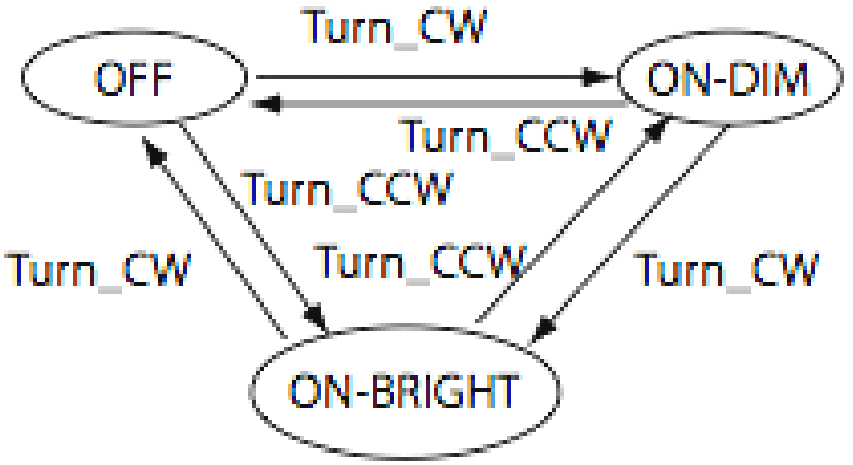
Esempio di una FSM

Lampada a tre stati:

(a) l'interruttore può essere girato in senso orario
(b) l'interruttore può essere girato in senso orario ed antiorario



(a)



(b)

Finite State Machine: definizione

Una FSM è una tupla (S, I, δ)

- S : insieme finito di stati
- I : insieme finito di eventi di input

$\forall \delta$: S x I ->S : funzione di transizione

FSM: rappresentazione grafica

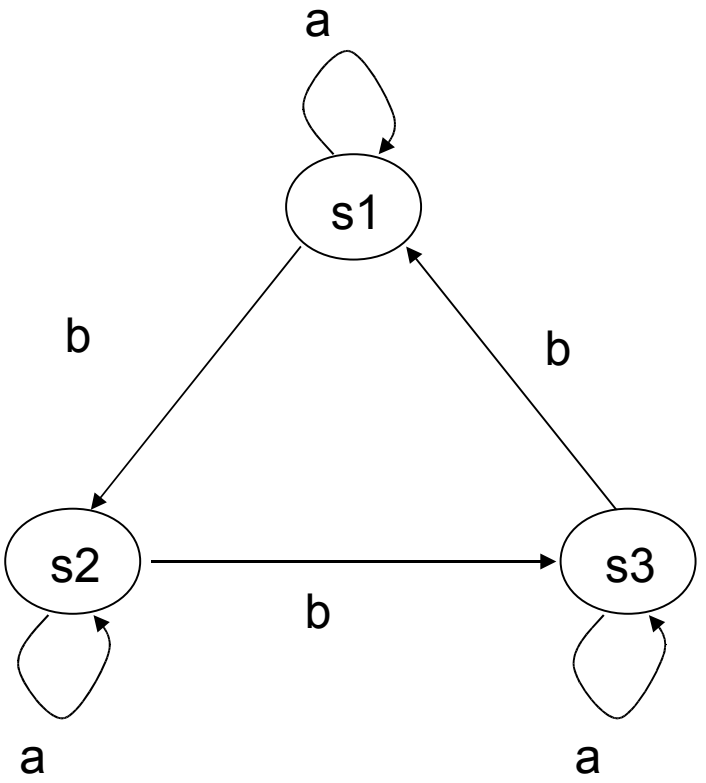
Un diagramma di stato è un grafo direzionato i cui nodi rappresentano gli stati ed i cui archi, etichettati dagli input, rappresentano le transizioni di stato

Esempio:

S = {s1, s2, s3}

I = {a, b}

δ= { (s1,a,s1), (s1,b,s2),
(s2,a,s2), (s2,b,s3),
(s3,a,s3), (s3,b,s1)}



Esempio

Sistema di controllo di impianto chimico (caso raffinato)

S = {Normale, Off, Azione di diminuzione pressione, Azione di diminuzione temperatura}

I = {Allarme di pressione pericolosa, Allarme di temp. pericolosa, Fallimento dell'azione_P, Fallimento dell'azione_T, pressione entro i limiti, temperatura entro i limiti}

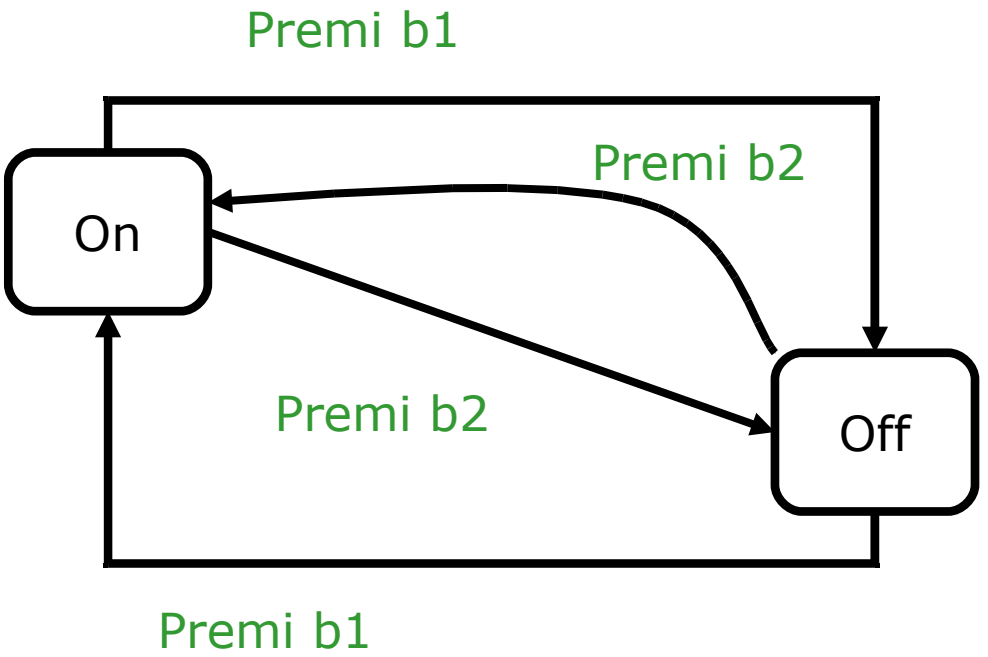
δ = {(Normale, Allarme di pressione pericolosa, Azione di diminuzione pressione), (Azione di diminuzione pressione, Fallimento dell'azione_P, Off), (Azione di diminuzione pressione, pressione entro i limiti, Normale), ecc.}

Esercizio 1

Usando una FM descrivere un sistema di una lampada e due bottoni avente il seguente comportamento: se la lampada è spenta, il pigiare uno dei due tasti causa l'accensione della lampada.

Soluzione esercizio 1

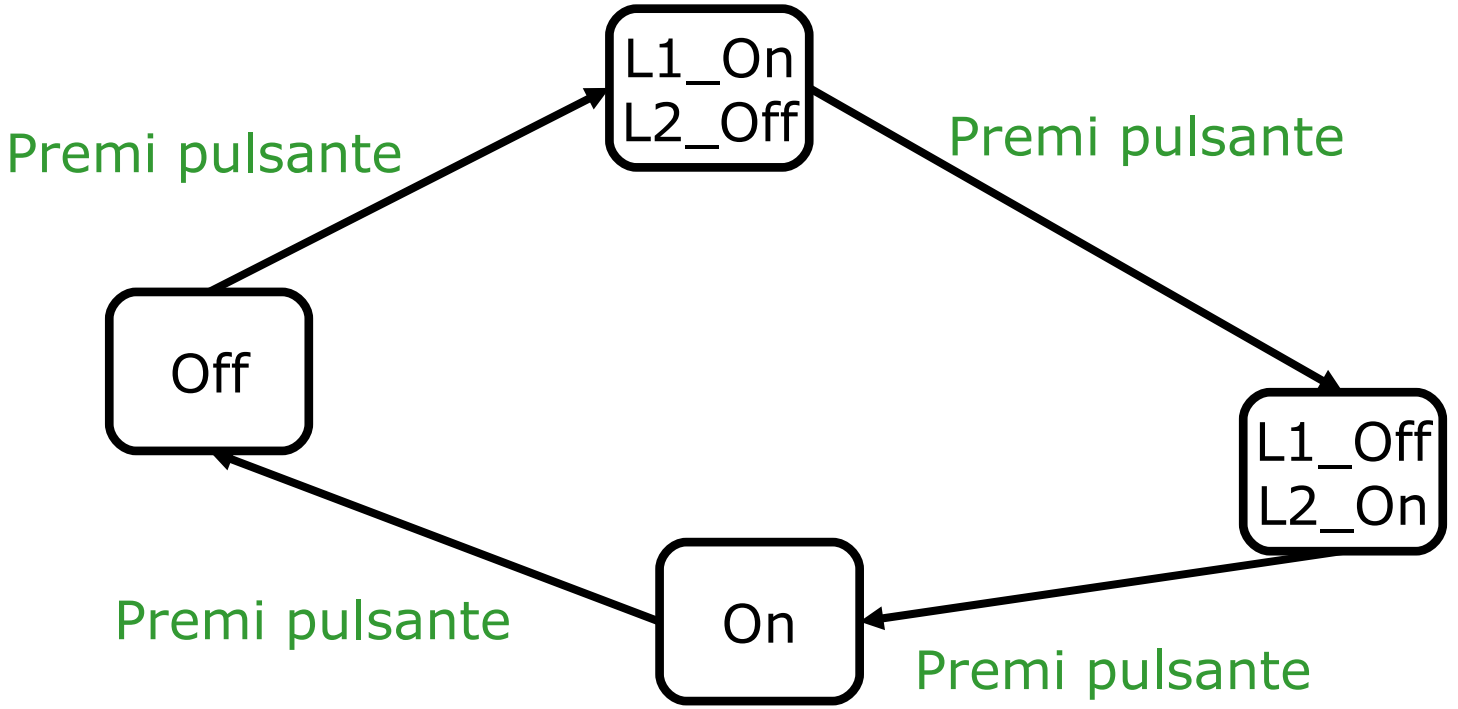
Tasti: b1 e b2



Esercizio 2

Descrivere un sistema con 2 lampade ed un bottone. Quando la luce è off, pigiare il bottone causa l' accensione della prima lampadina. Pigiare nuovamente il bottone causa l' accensione della seconda lampadina e lo spegnimento della prima. Pigiare nuovamente il bottone causa l' accensione di entrambe le lampadine, pigiare nuovamente causa lo spegnimento di entrambe le lampadine.

Soluzione esercizio 2



On= L1_On ed L2_On

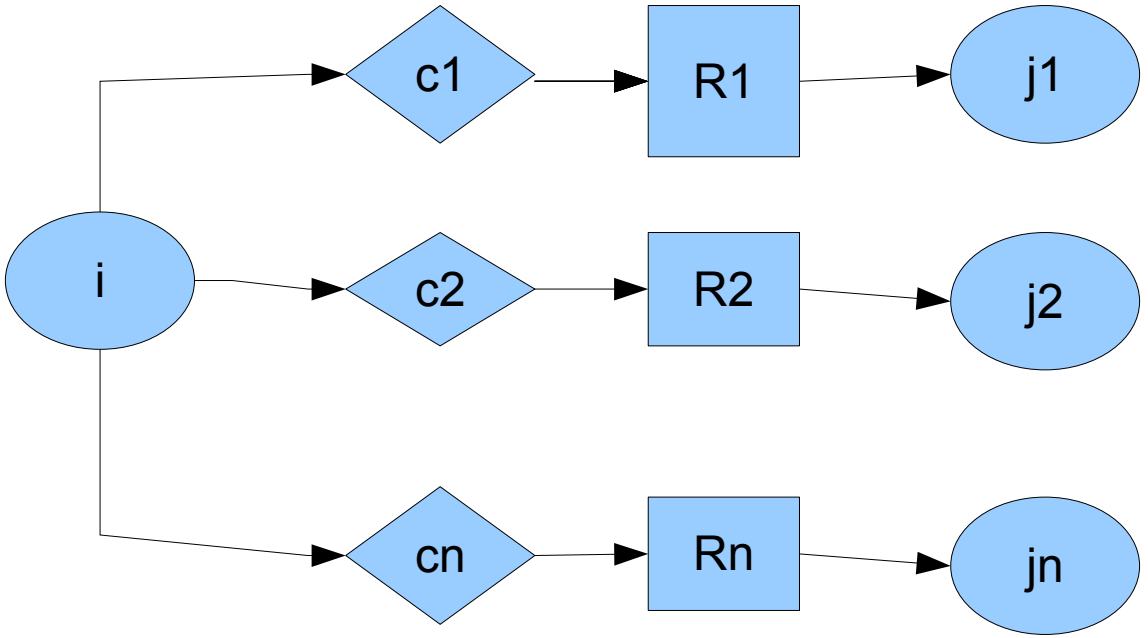
Estensioni di FSM

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***

Finite State Machine → ASM

Una FSM può essere definita da programmi della forma (uno per ogni stato)




```
if ctl_state = i then
  if c1  then
     R1
     ctl_state := j1
....
   if cn then
     Rn
     ctl_state := jn
```

- dove

– i, j1, j2, …, jn sono stati interni di controllo

  ctl_state una variabile che può prendere quei valori

– ck (k=1, 2, …, n) rappresentano le condizioni di input

– Rk le azioni di output

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

Abstract State Machines

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

Modello Computazionale

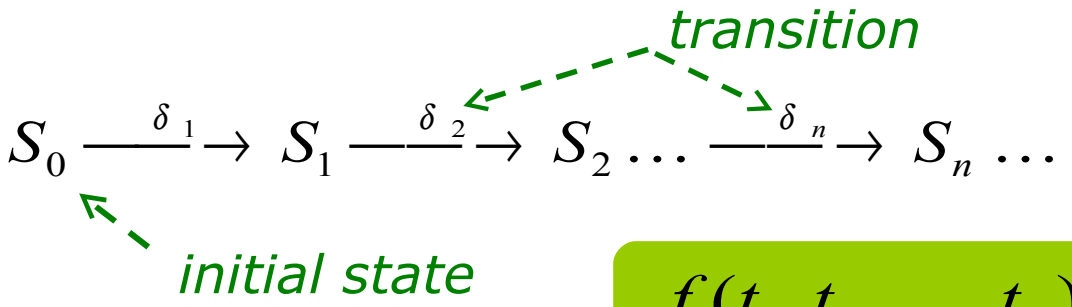
Vocabolario o segnatura

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

Azioni ⇨ transizioni di stato

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

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



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

Programmi (istruzioni di aggiornamento)

ASM STATO

ASM Stati

- Nelle ASM invece gli stati sono associati a un insieme di valori di qualsiasi tipo, memorizzate in apposite locazioni
  - Variabili (0-arie) x, y , …
  - Mappe (array) name(1), name(2), ...

• Conseguenza

- Le transizioni di stato delle FSM corrispondono alle transizioni di stato delle ASM con aggiornamenti dei valori contenuti nelle locazioni

  · Assegnamenti della forma loc(x1, x2,…,xn) := val

ASM state

- Gli stati di una ASM sono delle strutture algebriche, dette (semplicemente) *algebre*

- **DEF.** Un **vocabolario** Σ è una collezione finita di nomi di funzioni

- Le **funzioni** possono essere dinamiche o statiche a seconda che l'interpretazione del nome della funzione cambia o no da uno stato al successivo

  - Funzioni in senso matematico non informatico come "procedure"

  - Funzioni statiche di arietà zero sono dette costanti

  - Funzioni dinamiche di arietà zero sono le comuni variabili dei linguaggi di programmazione

ASM state formale

DEF. Uno s*tato* A del vocabolario *Σ è un insieme non vuoto X*, il *superuniverso di* A, con le *interpretazioni dei nomi delle funzioni di Σ*.

Se *f è* un nome di funzione *n*-aria di *Σ*, allora la sua interpretazione f^A è una funzione da X^n *a X*

Se *c* è un nome di costante di *Σ*, allora la sua interpretazione c^A è un elemento di *X*

Costanti

- Le costanti 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

Funzioni statiche

- 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)

ASM state esempio

Esempio di vocabolario

che contiene solo funzioni statiche

Il vocabolario Σ bool dell'algebra booleana contiene due costanti 0 e 1, una funzione unaria di nome '-' e due funzioni binarie di nomi '+' (per l'OR) e '*' (per l'AND)

ASM state

Esempio.

Due stati A e B per il vocabolario Σ_{bool} :

Il superuniverso dello stato A è {0, 1}.

Le funzioni sono interpretate come (dove *a*, *b* sono 0 o 1):

$0^A := 0$ (zero)

$1^A := 1$ (uno)

$-^A a := 1 - a$ (complemento logico)

$a +^A b := \max(a, b)$ (or logico)

$a *^A b := \min(a, b)$ (and logico)

Concetto di funzione dinamica

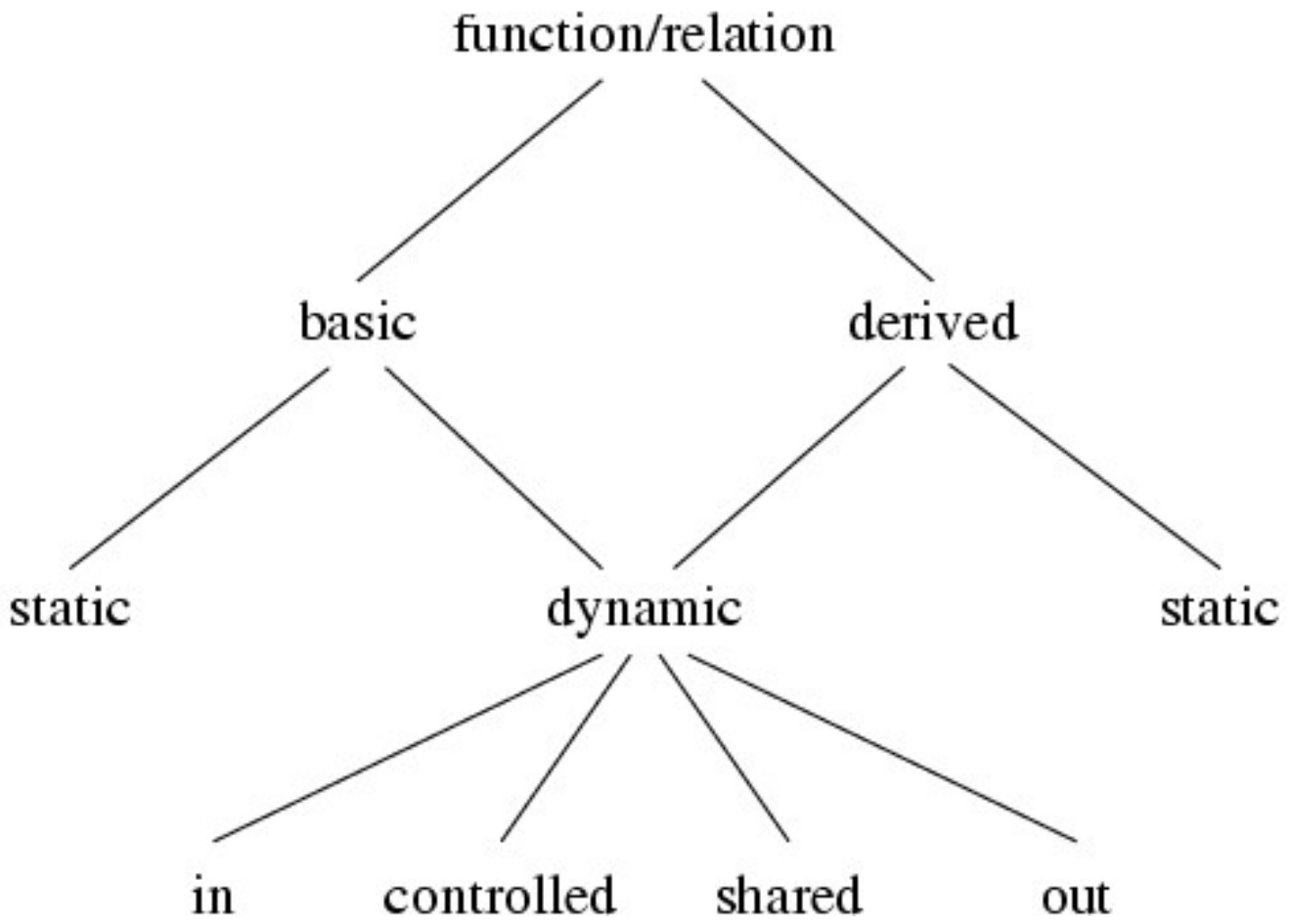
Alcune funzioni possono cambiare “valore”

Funzioni dinamiche

- primo esempio: le normali variabili dei programmi

  X, y, ...

ASM Function Classification



Realizza l'incapsulamento/information hiding dei linguaggi OO

ASM Function Classification

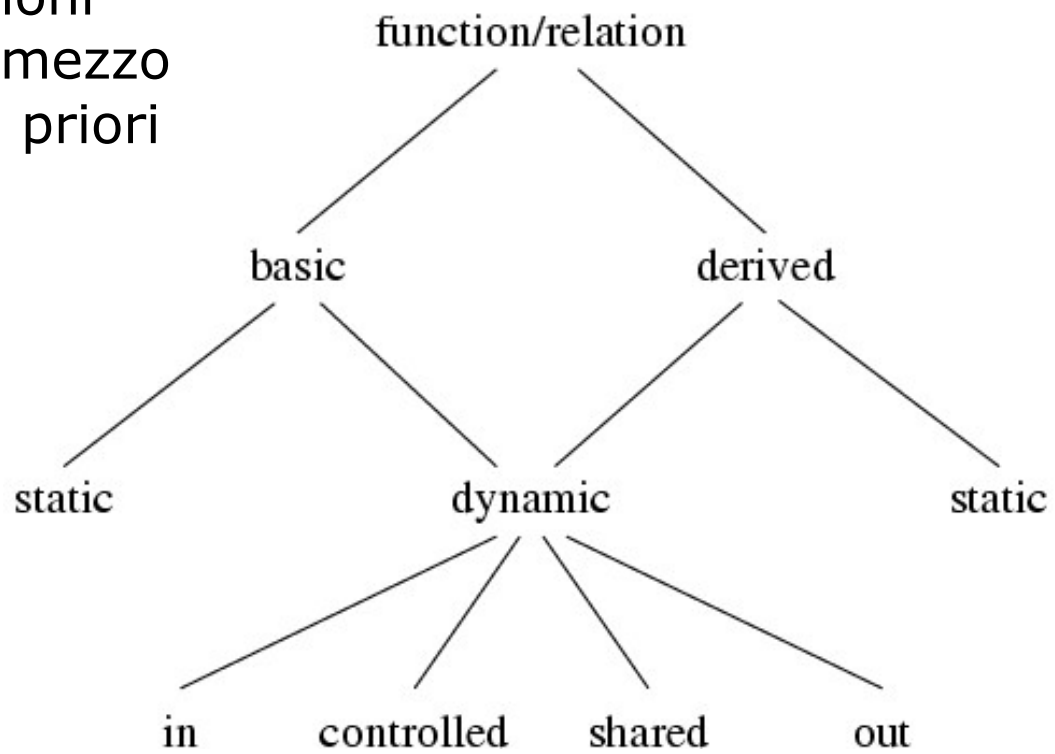
Detta ***M*** l'ASM corrente e ***env*** l'ambiente di ***M***:

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



Classification of Locations/Functions for given ASM M

- Dynamic: values depend on states of M
  - in (monitored): only read (not updated) by M, written only by env of M
  - out: only written (not read) by M, only read by env
  - controlled: read and written by M
  - shared: read and written by M and by the env of M (so that a protocol is needed to guarantee consistency of updates)
- Derived: values computed from monitored and static functions by some fixed scheme

Illustrating the ASM Function Classification

A real time CLOCK:

- Monitored: CurrTime: Real (supposed to be increasing)
- Controlled: DisplayTime: Nat x Nat

  Static: Delta: Real (system dependent time granularity), +, conversion to convert Real values into elements of Nat

If DisplayTime + Delta = CurrTime

Then DisplayTime := conversion(CurrTime)

With the following derived fct

- ClockTick = 1 iff (CurrTime = DisplayTime + Delta)

expressing a standard computing procedure, the rule becomes

If ClockTick = 1 Then DisplayTime := CurrTime

Concetto di funzione dinamica 2

Alcune funzioni 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

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

locazione

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

ASM Domini

Lo stato di una ASM è solitamente usato per modellare domini eterogeni

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 vedremo)

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

ASM Domini

- I soliti domini predefiniti cono disponibili
  - Interi, String
- L'utente può definirne altri
  - Da niente, come tipi astratti, come enumerativi
  - A partire da altri domini (strutturati)

Esempi

- → asmetaL
- Definizione dello stato – funzioni e domini semplici

ASM termini - espressioni

DEF. I *termini* di Σ sono espressioni sintattiche così costruite:

1. Variabili v_0, v_1, v_2, ...sono termini

2. Costanti c of Σ sono termini.

3. Se f è un nome di funzione n-aria di Σ e $t1, \ldots, tn$ asono termini, allora $f(t1, \ldots, tn)$ è un termine.

Tipicamemente i termini sono denotati dalle lettere r, s,t; le variabili dalle lettere x,y,z.

Un termine che non contiene variabili è detto *chiuso*.

ASM termine

Eesempio. Esempi di termini del vocabolario Σ_{bool}:

$+(v_0, v_1), +(1, *(v_7, 0))$

Solitamente, vengono scritti in notazione infissa come $v_0 + v_1$ e $1 + (v_7 * 0)$.

Essendo oggetti sintattici, i termini non hanno un significato.

Un termine può essere valutato in uno stato, se gli elementi del superuniverso so assegnati alle variabili del termine, cioè i *termini sono interpretati* in *A*

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

Costruttori di regole (alcuni)

- *Skip Rule:*

  skip

  Significato: non fare niente

- *Update Rule:*

  f **(*t*1, . . . , *tn*) := *s***

  dove:

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

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

  Significato: Nello stato successivo, il valore di f per gli argomenti $t_1, \ldots, t_n$ è aggiornato a s. Se f è 0-aria, cioè una variabile, l'aggiornamento ha la forma ***c* := *s***

Costruttori di regole (alcuni)

- *Conditional Rule:*

  if φ **then** R **else** S **endif**
  if φ then R endif

  Significato: se φ è vera, allora esegui *R,* altrimenti esegui *S*

- *Block Rule:*

  par R S **endpar**

  Significato: *R* e *S sono eseguite* in parallelo

- *Seq Rule:*

  seq R S **endseq**

  Significato: *R* e *S sono eseguite* in sequenza (gli update causati in R sono già visibili in S)

- *Let Rule:*

  let $x = t$ **in** *R*

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

Rule constructors

- *Forall Rule:*

  forall *x* **with** φ **do** *R*

  Significato: Esegui *R* in parallelo per ogni *x* che soddisfa φ

  Implementa il concetto di parallelismo sincrono

- *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!*

Rule constructors macro

- *(Macro) Call Rule:*

  $r\ [t_1, \ldots, t_n]$

  Significato: Chiama *r (regola con parametri) con* argomenti *t*1*, . . . , tn*

  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*)

Sorting ASMs con call rule

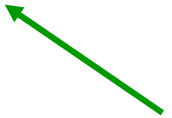
Problema: ordinare un array *a*

Soluzione:

- iterare localmente con swap:

```
choose i,j in dom(a) s.t. (i < j & a(i) > a(j))
         swap(a(i),a(j))

swap(x,y) =  x := y
             y := x
```



Call rule

ASMs

Una *abstract state machine M* è una terna (Σ,A,R)

- un vocabolario Σ,
- uno stato iniziale A per Σ,
- un insieme R di nomi di regole con
  - un nome di regola di arietà zero, la cosidetta *main rule (*l' "entry point" per l'esecuzione della macchina)
  - una definizione di regola per ogni nome di regola

La *semantica delle regole di transizione* è data dall'insieme di tutti gli aggiornamenti.

La Riserva di una ASM

- *Extend Rule:*

Per estendere un sub-universo del superuniverso con nuovi elementi si usa la notazione:

import *x* **do** *R*

Il significato di tale costrutto è: scegli un elemento *x* da *Reserve* (dominio predefinito di simboli "freschi"), cancellalo da *Reserve,* aggiungi *x* al superuniverso e esegui *R*

- Spesso si preferisce la forma seguente:

extend *U* **with** *x* [**do]** *R*

come abbreviazione per **import** *x* **do**

U(*x*):= true **seq** *R*

Aggiornamenti e locazioni

- Un *aggiornamento* per A è una tripla

$$(f, (a1, \ldots, an), b)$$

dove f è un nome di funzione dinamica n-aria, e $a1, \ldots, an$ e b sono elementi di $|A|$

- Il *significato* dell'aggiornamento è che l'interpretazione della funzione f in A viene modificata per gli argomenti $a1, \ldots, an$ con il valore b

- La coppia $<f, (a1, \ldots, an)>$, cioè le prime due componenti di un update, è detta *locazione*

- Un *update set* è un insieme di aggiornamenti

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 $b = c$

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.

All one needs to know about (sequential) ASMs (1)

Una ASM è un insieme finito di regole della forma

If Condition **Then** Updates

- dove Updates è un insieme finito di aggiornamenti di funzioni del tipo f(t1, …,tn):=t,
- Condition è espressione booleana (1st order formula)

All one needs to know about (sequential) ASMs (2)

Nello stato corrente (struttura) S:

- determina tutte le regole che possono scattare (fireable) in S (t.c. Cond è true in S)
- computa tutte le exprs ti,t che occorrono negli aggiornamenti f(t1,…,tn):=t
- esegui simultaneamente tutti questi aggiornamenti di funzioni

L'aggiornamento globale produce lo stato successivo S'

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)

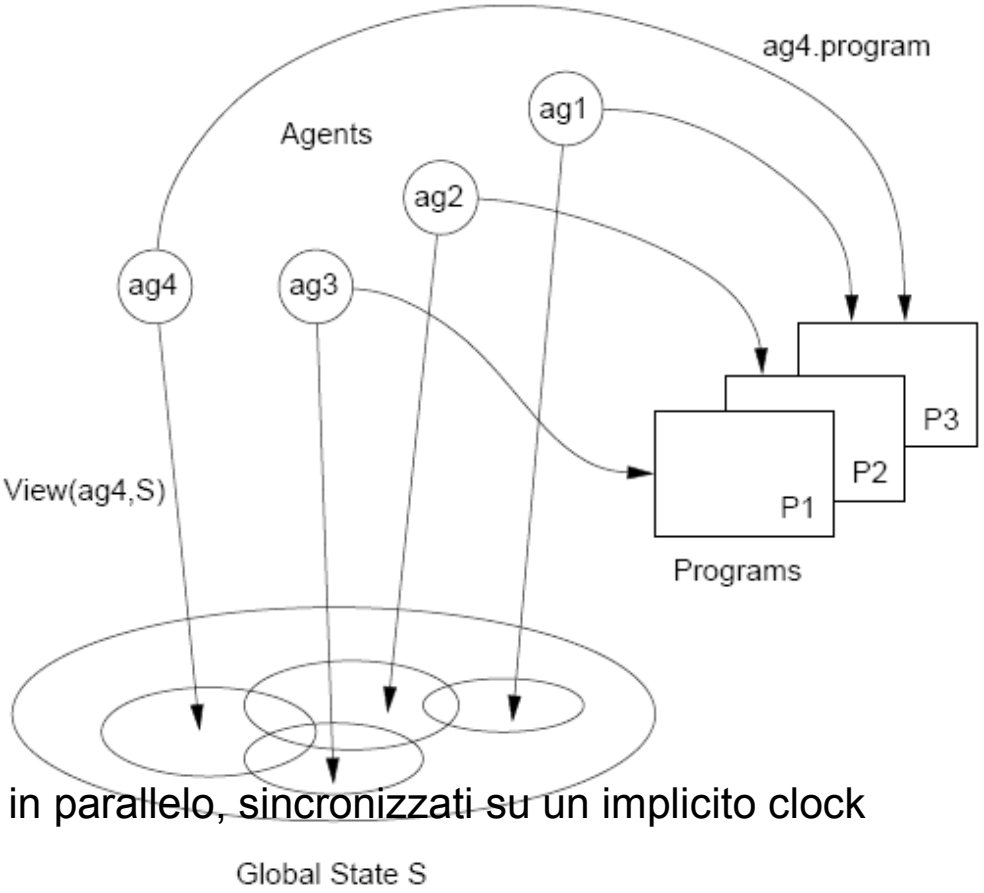
- Agenti computazionali
- Stati globali condivisi tra gli agenti
- Movimenti concorrenti sincroni/asincroni

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