

## Abstract State Machine – Concetti di Base

## Idee guida

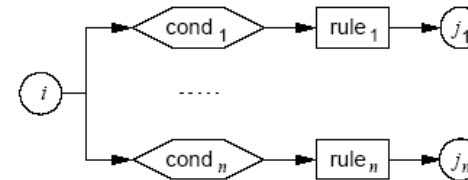
- ASM = FSM con stati generalizzati
  - Le ASM rappresentano la forma matematica di **Macchine Astratte** che estendono la nozione di **Finite State Machine**
- Ground Model (descrizioni formali)
- Raffinamenti

## Finite State Machine (1)

- Un **automa a stati finiti** è definito da una 5-pla:  $FSM = \langle Q, \Sigma, \delta, q_0, F \rangle$ , dove:
  - $Q$  è l'insieme finito e non vuoto degli **stati dell'automa**, ciascuno caratterizzato da una particolare configurazione di valori delle variabili di stato
  - $\Sigma$  è l'**alfabeto di input**
  - $\delta$  è la funzione **transizione di stato**  $\delta : (Q \times \Sigma) \rightarrow Q$  (automi deterministici)
  - $q_0$  è lo **stato iniziale**
  - $F \subseteq Q$  è l'insieme di **stati finali**

## Finite State Machine (2)

- Una FSM può essere definita da un programma della forma



```
if ctl_state = i then
  if cond1 then
    rule1
    ctl_state := j1
    ...
  if condn then
    rulen
    ctl_state := jn
```

## Finite State Machine (3)

- dove
  - `ctl_state` rappresenta lo stato, i cui valori appartengono a un insieme finito
  - `i, j1, j2, ..., jn`, sono stati interni (i valori di `ctl_state`)
  - `condk` ( $k=1, 2, \dots, n$ ) rappresentano le condizioni di input
  - `rulek` le azioni di output

## Finite State Machine (4)

- Definizione in forma testuale (**alternativa**)  
 $FSM(i, \text{if } cond \text{ then } rule, j) =$   
**if** `ctl_state = i` **and** `cond` **then** {`rule`,  
`ctl_state:=j`}

## Da FSM a ASM (1)

- 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 di un certo tipo
  - le condizioni di input e le azioni di output
    - alfabeto finito
  - la potenza computazionale
    - le ASM soddisfano la Tesi di Church-Turing

## Da FSM a ASM (2)

- Nelle ASM invece gli stati sono associati a un **insieme di valori di qualsiasi tipo**, memorizzate in apposite **locazioni**
  - Una locazione rappresenta il concetto astratto di unità di memoria, indipendente dal particolare meccanismo di indirizzamento
  - L'astrazione al livello opportuno è ottenuta mediante parametrizzazione delle locazioni

## Da FSM a ASM (3)

- Più precisamente: gli stati delle ASM sono strutture matematiche in cui **i dati sono oggetti astratti**, e cioè elementi di insiemi a cui è possibile applicare operazioni e predicati
- Conseguenza
  - Le transizioni di stato delle FSM corrispondono alle transizioni di stato delle ASM, ma in più **con aggiornamenti dei valori** contenuti nelle locazioni
    - Assegnamenti della forma  $\text{loc}(x_1, x_2, \dots, x_n) := \text{val}$

## Da FSM a ASM (4)

- La differenza tra il concetto di stato di FSM e ASM porta a macchine i cui stati possono essere domini **di qualsiasi oggetto**
- Analogamente all'estensione degli stati FSM (non strutturati) in stati ASM (strutturati), le condizioni di input di FSM sono estese ad arbitrarie espressioni sugli stati nelle ASM
  - Queste espressioni sono sentinelle (guard), in quanto determinano l'istruzione che deve essere eseguita.

## Definizione di ASM

- Insieme di istruzioni (**regole ASM**) della forma
    - if cond then Updates**
- dove:
- Updates è un insieme di aggiornamenti di funzioni  $f(t_1, t_2, \dots, t_n) := t$

## Nota

- Si parla di “**regole ASM**” per evidenziare la distinzione tra
  - il **modello di esecuzione parallela** per le ASM
  - e il **modello di esecuzione a singola istruzione** della programmazione tradizionale

## Concetto di funzione

- Il termine funzione deve essere inteso in senso **matematico**, non informatico
  - Per esprimere dal punto di vista matematico 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

## Abstract State Machine – Definizioni Formali

## Definizioni (1)

- **Signature**: insieme di nomi di funzioni dell'ASM
  - Detta anche **vocabolario** della ASM, è indicata con  $\Sigma$
- **Superuniverso**: è la collezione di elementi di uno stato

## Definizioni (2)

- **Stato** su una signature  $\Sigma$  è costituito dal superuniverso di quello stato e dall'interpretazione dei nomi di funzioni in  $\Sigma$
- L'**interpretazione** di una funzione  $f$  (di arietà  $n$ ) su un universo  $X$  è una funzione da  $X^n$  a  $X$ 
  - L'interpretazione di una costante è un elemento di  $X$
  - Le funzioni parziali sono totalizzate ponendo undef l'interpretazione per gli argomenti per cui la funzione non è definita

## Definizioni (3)

- Le **Locations** sono coppie  $(f, (v_1, v_2, \dots, v_n))$  con  $f$  nome di una funzione e  $v_1, v_2, \dots, v_n$  lista di valori di  $X$

## Regole di Transizione (1)

- Una ASM è un sistema che comprende un numero finito di **regole di transizione** della forma

if Condition then Updates

- che determinano transizioni di stato nella macchina

## Regole di Transizione (2)

- **Condition** (o sentinella) è una qualsiasi formula del primo ordine, senza variabili libere, la cui interpretazione può essere true o false
  - Funge da sentinella, in quanto solo il suo verificarsi determina l'applicazione della regola
- **Updates** è l'insieme finito di aggiornamenti di funzione, la cui esecuzione determina la definizione o il cambiamento (in **parallelo**) dei valori di una funzione
  - Un update corrisponde alla coppia  $\langle \text{location}, \text{value} \rangle$

## Regole di Transizione (3)

- In un dato stato:
  - vengono valutati tutti i parametri
  - si cambia il valore della funzione
- Un update  $\langle \text{loc}, \text{val} \rangle$  rappresenta l'unità del cambiamento di stato, determinato dal cambiamento del valore della locazione  $\text{loc}$ , che assume il valore  $\text{val}$

## Stati

- La nozione di stato nelle ASM corrisponde alla nozione di **struttura dati astratta**
  - I dati sono elementi di insiemi (**domini**) a cui è possibile applicare
    - operazioni di base
    - predicati (attributi/relazioni)
- L'esecuzione di una ASM è un'**istanza della computazione** di sistemi a transizione

## Consistenza di update

- Un insieme di update è consistente se tutti fanno riferimento a locazioni diverse
- Se una coppia di update fa riferimento alla stessa locazione, l'intero insieme è inconsistente

## Computazione (1)

- Un passo computazionale di una ASM in un dato stato consiste nell'eseguire **simultaneamente** tutti gli update di tutte le regole di transizione la cui sentinella è vera per quello stato
- Condizione necessaria è che gli update siano tutti **consistenti**. In tal caso il risultato della computazione determina la transizione di stato

## Computazione (2)

- Se esiste **inconsistenza** tra una coppia di update, allora la computazione non porta a un nuovo stato, ma si ha un errore
- La computazione di una ASM procede in modo iterativo, ripetendo successivamente ogni passo computazionale

## Computazione (3)

- Nel caso particolare di esecuzioni che terminano, è possibile definire qualche criterio di terminazione
  - Non è più applicabile alcuna regola
  - Viene eseguito un update vuoto
  - Lo stato non cambia più (stato pozzo)
  - ...

## Vantaggi della esecuzione simultanea (1)

- L'esecuzione simultanea fornisce un utile strumento per la progettazione di alto livello per descrivere localmente un cambiamento di stato globale
  - Cioè, quello ottenuto in un unico step, applicando un insieme di update
  - Unica limitazione: la consistenza

## Vantaggi della esecuzione simultanea (2)

- Favorisce l'astrazione dalla sequenzialità
  - Facilita la progettazione di sistemi distribuiti/paralleli

## Esecuzione simultanea di una regola (1)

- Notazione per esprimere la esecuzione simultanea di una regola R per ogni x che soddisfa la condizione C:

forall x with C

R

- Analogamente è possibile esprimere il non determinismo nel modo seguente

choose x with C

R

## Esecuzione simultanea di una regola (2)

- In entrambi i casi
  - C è un'espressione a valori booleani
  - R è una regola

## Classificazioni (1)

- Due tipologie di funzioni
  - **Basic**: sono le funzioni elementari, che costituiscono la signature di una ASM
  - **Derived**: sono funzioni il cui valore in ogni stato è calcolato a partire dalle funzioni di base

## Classificazioni (2)

- Le funzioni basic possono essere
  - **Static**: non cambiano mai valore durante l'esecuzione,
    - il loro valore non dipende dallo stato corrente
    - funzioni static di arietà 0 sono le costanti
  - **Dynamic**: sono funzioni il cui valore dipende dallo stato corrente della ASM
    - il loro valore dipende dagli update della ASM o dell'ambiente esterno
    - funzioni dynamic di arietà 0 corrispondono alle variabili dei tradizionali linguaggi di programmazione

## Classificazioni (3)

- Le funzioni dynamic possono essere
  - **Monitored** (o **in**): possono essere lette ma non modificate dalla ASM
    - Sono modificate solo dall'ambiente esterno o da altre ASM, nel caso di sistemi multiagente
    - In ogni stato **deve** essere specificato il valore di **tutte** le funzioni monitored
  - **Controlled**: sono modificate solo dalla ASM, mediante l'esecuzione delle regole di transizione
    - non dall'ambiente né da altre ASM



## Classificazioni (4)

- **Shared** (o **interaction**): possono essere modificate da più ASM e dall'ambiente
- **Out**: possono essere modificate ma non lette dalla ASM
  - in genere sono monitored per l'ambiente e per le altre ASM

## Classificazioni (5)

- Classificazioni analoghe valgono per le location e per le relazioni

## Abstract State Machine – Composizione

## Composizione di ASM

- Modellizzare un sistema mediante ASM presenta il vantaggio di favorire la composizione di diverse ASM, ottenendo ASM più complesse
- Idea di base:
  - Definire costrutti per la composizione di ASM analoghi a quelli di composizione di istruzioni nei linguaggi di programmazione

## Costrutti per la Composizione di ASM

- Per comporre ASM sono necessari i costrutti di
  - sequenza
  - iterazione

## Costrutto di Sequenza

- La composizione per sequenzialità deve essere esplicitamente costruita
  - Costruire meccanismi che permettono di eseguire quanto specificato da una ASM dopo aver eseguito quanto specificato da un'altra
  - trattare l'esecuzione sequenziale  $P \text{ seq } Q$  delle due regole  $P$  e  $Q$  come un'azione atomica

## Costrutto di Sequenza - Esempio (1)

- Si vuole costruire per composizione di ASM una ASM che svolga la funzione di `pop_back` su una lista
  - Cancellazione dell'ultimo elemento di una lista
- Siano disponibili le regole
  - `move_last`: setta il puntatore sull'ultimo elemento della lista
  - `delete`: cancella l'elemento della lista correntemente puntato

## Costrutto di Sequenza - Esempio (2)

```
pop_back =  
if mode = Move then  
    move_last  
    mode:= Delete  
if mode = Delete then  
    delete  
    mode:= Move
```

**ERRATO**

## Costrutto di Sequenza - Esempio (3)

- L'utente della `pop_back` deve sapere che l'attività computazionale viene svolta in due step
- L'ordinamento sequenziale è definito dalla mode
  - Che si suppone inizializzata a `Move`
  - Spesso è difficile da gestire
- In generale, problemi potrebbero sorgere quando si raffina una ASM con regole sequenzializzate

## Costrutto di Sequenza - Esempio (4)



L'esecuzione simultanea  
delle regole R e S porta  
dallo stato 1 a 2

R è raffinata da  $R_1 - R_2 - R_3$

S è raffinata da  $S_1 - S_2$

- Non è possibile eseguire una regola  $R_i$  in modalità interleaved con una regola  $S_i$
- Non si può usare mode

## Costrutto di Sequenza

- È necessario introdurre il costruttore di sequenza che permette di inserire una regola  $P \text{ seq } Q$  in una seconda ASM
- Se la regola  $P \text{ seq } Q$  deve essere eseguita in parallelo con altre regole, allora  $P \text{ seq } Q$  deve essere considerata come azione atomica

## Costrutto di Sequenza

- La composizione sequenziale  $P \text{ seq } Q$  di due regole P e Q è una regola, la cui semantica è quella di
  - eseguire la regola P
  - raggiungere così uno stato intermedio
  - eseguire la regola Q a partire dallo stato intermedio

## Merging di update (1)

- L'esecuzione di  $P \text{ seq } Q$  richiede che avvenga il **merge** dei due insiemi di update per poter evitare problemi di inconsistenza
- Sia  $U$  l'update determinato da  $P$  e  $V$  quello determinato da  $Q$  in  $P \text{ seq } Q$ 
  - Se riguardano la stessa location allora  $V$  sovrascrive  $U$

## Merging di update (2)

- È possibile fare il merge dei due update solo se  $U$  è consistente
- $$U \oplus V =$$
- $$\{(loc, val) \mid (loc, val) \in U \wedge loc \notin \text{Locs}(V)\} \cup V$$
- se  $U$  è consistente
- $U$  altrimenti
- Se la regola  $P$  non è consistente allora
    - $P \text{ seq } Q = P$

## Proprietà del costrutto di sequenza

- Il costruttore di sequenza per ASM  $\text{seq}$ 
  - Ha un elemento neutro sinistro e uno destro
    - $\text{skip seq } R = R \text{ seq skip} = R$
  - È associativo
    - $P \text{ seq } (Q \text{ seq } R) = (P \text{ seq } Q) \text{ seq } R$

## Costrutto di iterazione (1)

- Avendo definito l'operatore di sequenza  $\text{seq}$  tra ASM, esso può essere applicato ripetutamente per ottenere l'iterazione di una regola
- In questo modo è possibile ripetere un'azione atomica un numero di volte
  - finito
  - non conosciuto a priori

## Costrutto di iterazione (2)

- Definiamo  $R^n$ :
  - skip se  $n=0$
  - $R^{n-1}$  seq  $R$  se  $n>0$
- L'iterazione si conclude quando l'insieme di update
  - diventa vuoto (terminazione con successo)
  - diventa inconsistente

## While

- Un ciclo While ripete l'esecuzione del suo corpo fintanto che una data condizione è vera,
  - cioè
- while (cond) R = iterate (if cond then R)

## ASM di Boehm-Jacopini

- Ogni ASM che può essere definita con i costrutti di sequenza e iterazione è detta ASM di Boehm-Jacopini
- Ogni funzione computabile può essere calcolata da una ASM di Boehm-Jacopini
- Le ASM di Boehm-Jacopini hanno la stessa capacità computazionale delle Macchine di Turing

## ASM Parametrizzate

- Si può strutturare una ASM complessa utilizzando macro
- Meccanismi di parametrizzazione permettono il passaggio di argomenti tra la ASM chiamante e la ASM chiamata
- Analogamente, è possibile definire meccanismi che permettono alla ASM chiamata di restituire un valore alla ASM chiamante

## Astrazione degli Stati mediante Predicati – State Predicate Abstraction

## Precisazione

- La discussione sull'astrazione degli stati mediante predicati è stata introdotta in [BPV15] e non è (ancora) definita nel corpo delle conoscenze delle ASM

[BPV15] Bianchi A., Pizzutilo S., Vessio, G., "Applying Predicate Abstraction to Abstract State Machines", in K. Gaaloul, R. Schmidt, S. Nurcan, S. Guerreiro, and Q. Ma, (Editors) Enterprise, Business-Process and Information Systems Modeling - Proc. of the 20th International Conference on Exploring Modeling Methods for Systems Analysis and Design – EMMSAD – Held at CAiSE2015, Stockholm, Sweden, June 2015, Lecture Notes on Business Information Processing - LNBP/214, pp.283-292  
Abstract State Machine –  
Concetti di Base

## Problema

- I modelli computazionali classici basati su stati rappresentano lo stato corrente come una sequenza di simboli
  - La rappresentazione degli stati è quindi limitata alla specifica struttura dati adottata
- Le ASM superano tale limitazione permettendo qualsiasi struttura dati
  - Grande quantità di dati per specificare lo stato
  - Maggior difficoltà nell'analisi del sistema

## Soluzione

- Definizione di un'astrazione in grado di catturare la semantica degli stati ASM
- L'insieme delle location viene partizionato in sottoinsiemi da cui estrarre le location effettivamente utili nell'analisi di uno specifico problema

## Approccio Seguito

- **Predicate Abstraction**: è una tecnica ampiamente usata per l'analisi dei programmi
  - Prevede di generare un modello astratto (più semplice) così da semplificare l'analisi
  - Gli stati del sistema sono mappati in stati del modello
  - Il modello presenta lo stesso flusso di controllo ma riguarda solo i predicati relativi agli stati

## Definizione

- Un predicato  $\phi$  su uno stato  $s$  di una ASM è una formula del primo ordine definita sulle locazioni in  $s$  tale per cui  $s \models \phi$
- Il predicato è soddisfatto nello stato, cioè quando il sistema si trova in  $s$ ,  $\phi$  è soddisfatto
  - Il predicato è **sempre** soddisfatto nello stato  $s$
  - Il predicato può non essere soddisfatto in altri stati diversi da  $s$

## Commenti

- I predicati sugli stati rappresentano così la semantica di ogni stato
- In una ASM è possibile definire un insieme  $\Phi = \{\phi_1, \phi_2, \dots, \phi_n\}$  di predicati t.c. nello stato corrente ogni  $\phi_i$  può essere verificato o meno

## Conseguenze

- L'uso dei predicati sugli stati permette di rappresentare le proprietà locali di ogni stato
- Le proprietà globali possono essere indagate componendo le proprietà locali
- Uno stesso predicato può valere in più stati
- In un dato stato possono essere soddisfatti più predicati