Modellazione ed Analisi di Sistemi - Risposte degli Esami

Esame del 4 Febbraio 2019

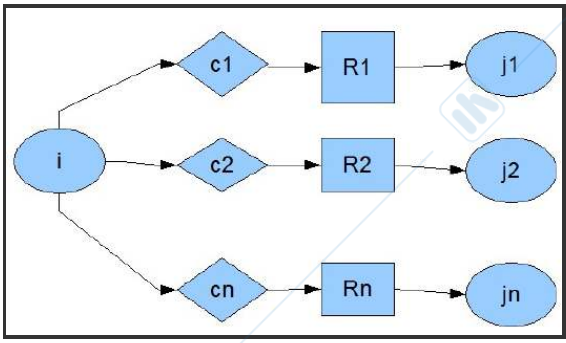
1. Descrivere i meccanismi in ASM che permettono di modellare il non determinismo. Portare degli esempi (almeno 1 per meccanismo).

Il meccanismo che consente di modellare il non-determinismo in ASM è il costrutto *choose* che permette, da una lista di elementi omogenei, di estrarre ed assegnare ad una variabile un valore casuale tra quelli della lista che rispettano una data condizione.
Un esempio di utilizzo della choose è il seguente:

```
choose $num in {1 : 100} with true do [term]
```

Seleziona casualmente un numero nel dominio {1 : 100} (qualsiasi numero è verificato dato il "with true"), lo assegna alla variabile $num ed esegue il term [term]. [Mi sembra che nell'ultima lezione abbia anche detto che il non determinismo sia dato anche dalle variabili monitorate.]

2. Dare lo schema ASM per una FSM.




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

dove:

- *ctl_state* è una variabile che rappresenta lo stato corrente e può prendere come valore uno stato di controllo
- *i, j1, j2, …, jn* sono stati interni di controllo (i valori di ctl_state)
- *ck(k=1, 2, …, n)* rappresentano le condizioni di input
- *Rk* le azioni della macchina

3. Descrivere la classificazione delle funzioni in una ASM.

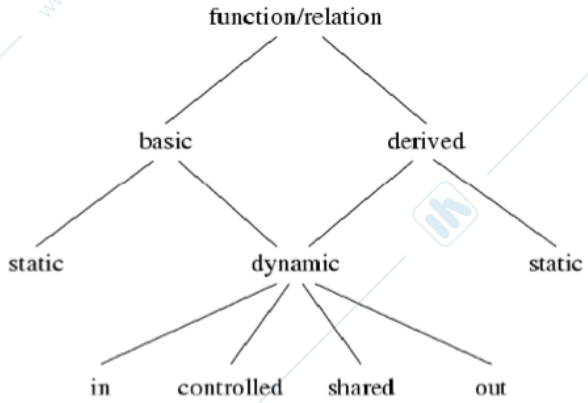
Le funzione in ASM possono essere:

- Controllate: I valori vengono letti ed assegnati dalla macchina ASM
- Monitorate: I valori vengono assegnati dall'ambiente e letti dalla macchina ASM
- Shared: I valori vengono letti ed assegnati sia dall'ambiente che dalla macchina ASM
- Out: I valori vengono assegnati dalla macchina ASM e letti dall'ambiente
- Derivate: Il valore viene derivato dai valori delle funzioni dello stato corrente della macchina ASM

Inoltre, le funzioni possono essere:

- Dinamiche: Il loro valore può variare nel tempo

- Statiche: Il loro valore è fissato all'inizio e non può variare



6. Date la definizione di automa di Kripke.

Un automa di Kripke M è composto dalla quadrupla (S, I, Δ, L), dove:

- S: Stati del sistema
- $I \subseteq S$: Stati iniziali del sistema
- Δ: Funzione di transizione tale che $\forall s \in S, \exists s' \in S \; t.c. \; s \rightarrow s'$
- $L : S \rightarrow 2^{PA}$: Funzione di etichettatura

Gli automi di Kripke sono rappresentabili graficamente tramite:

- S: Stati del sistema
- Cerchi per gli stati
- Archi orientati per le transizioni

Regole:

- Deve esistere almeno uno stato iniziale
- Gli stati possono essere etichettati tramite la funzione di labelling
- È un automa left-total, quindi si impone l'assenza di deadlock, cioè $\forall s \in S, \exists s' \in S \; t.c. \; s \rightarrow s'$
    - È possibile simulare il deadlock tramite un cappio sullo stato

- $AF\psi_1$:
  - Se uno stato s è etichettato con ψ_1 allora s viene etichettato con $AF\psi_1$.
  - Uno stato è etichettato con $AF\psi_1$ se tutti gli stati successori sono etichettati con $AF\psi_1$, fin quando è possibile (Figura 41).

- $E[\psi_1\ U\ \psi_2]$:
  - Se uno stato s è etichettato con ψ_2 allora viene etichettato con $E[\psi_1\ U\ \psi_2]$.
  - Uno stato è etichettato con $E[\psi_1\ U\ \psi_2]$ se è etichettato con ψ_1 e almeno uno dei suoi successori è etichettato con $E[\psi_1\ U\ \psi_2]$, fin quando possibile (Figura 42).

- $EX\psi_1$: Etichetta uno stato s con $EX\psi_1$ se uno dei suoi successori con ψ_1.

function $\mathtt{SAT}_{\mathtt{AF}}(\phi)$
/* determines the set of states satisfying AF ϕ */
local var X, Y
begin
  $X := S$;
  $Y := \mathtt{SAT}(\phi)$;
  repeat until $X = Y$
  begin
    $X := Y$;
    $Y := Y \cup \mathrm{pre}_\forall(Y)$ = $Y \cup \{s \mid \forall s$ con $s \to s'$ e $s' \in Y\}$
  end
  return Y
end

function $\mathtt{SAT}_{\mathtt{EU}}(\phi, \psi)$
/* determines the set of states satisfying E[ϕ U ψ] */
local var W, X, Y
begin
  $W := \mathtt{SAT}(\phi)$;
  $X := S$;
  $Y := \mathtt{SAT}(\psi)$;
  repeat until $X = Y$
  begin
    $X := Y$;
    $Y := Y \cup (W \cap \mathrm{pre}_\exists(Y))$ = $Y \cup (W \cap \{s \mid \exists s'$ con $s \to s'$ e $s' \in Y\})$
  end
  return Y
end

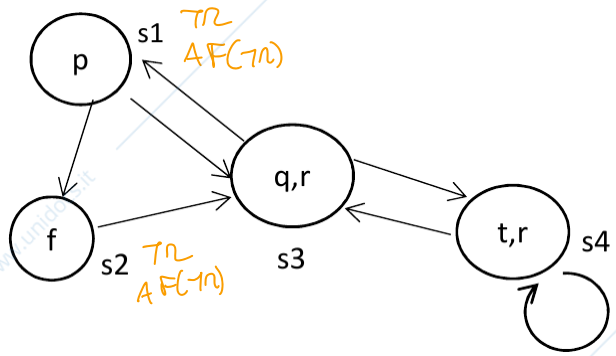
function $\mathtt{SAT}_{\mathtt{EX}}(\phi)$
/* determines the set of states satisfying EX ϕ */
local var X, Y
begin
  $X := \mathtt{SAT}(\phi)$;
  $Y := \mathrm{pre}_\exists(X)$; = $\{s_0 \in S \mid s_0 \to s_1$ per qualche $s_1 \in X\}$
  return Y
end

7. Formalizzare in logica CTL la seguente proprietà: *se l'evento p accade, allora nel futuro gli eventi s e t accadranno simultaneamente su tutti i cammini di computazione*.

$AG(p \rightarrow AF(s \wedge t))$

8. Dare la semantica in logica CTL dell'operatore $EG\varphi$. Tramite l'algoritmo di model checking verificare per quali stati s vale la proprietà $M, s \vDash EG(r)$ nell'automa in figura.

AF, EU, EX



a) $M, s \vDash EG(\varphi)$ sse ∃ un cammino $s_1, s_2, \ldots$ con $s_1 = s$, in cui ∀ s_i vale $M, s_i \vDash \varphi$

b)

$M, s \vDash EG(r)$

$EG(r) \equiv \neg AF(\neg r)$

EG(r) = ¬AF(¬r)

- 1: r
- 2: $\neg r$
- 3: $AF(\neg r)$
- 4: $\neg AF(\neg r)$

Etichettatura:

- s1: 2,3
- s2: 2,3
- s3: 1,4
- s4: 1,4

Output: {s3, s4}

Esame del 25 Maggio 2017

1. Descrivere i meccanismi in ASM che permettono di modellare il non-determinismo.

Risposta nella domanda 1 dell'esame del 4 Febbraio 2019

2. Dare la definizione di verifica e di validazione di un sistema. Indicare quali metodi possono essere applicati per l'espletamento delle due attività.

La validazione è il processo di valutare un sistema o una componente durante o alla fine del processo di sviluppo per determinare se esso soddisfa i requisiti specificati, essa è necessaria per controllare che il sistema soddisfa i requisiti richiesti.

La verifica invece è il processo di valutare un sistema o una componente per determinare se i prodotti di una data fase di sviluppo soddisfano le condizioni imposte allo stato di detta fase ed è necessaria per garantire le proprietà del sistema

La validazione può essere eseguita tramite formalismi operazionali, dato che facilitano la validazione. Uno dei formalismi operazionali più usati è quello di modellare il sistema tramite le Abstract State Machine e simularne l'esecuzione.
La verifica può essere eseguita tramite formalismi dichiarativi, dato che facilitano la verifica. Uno dei formalismi dichiarativi più utilizzati è la Logica Temporale (CTL) che permette di esprimere le formule logiche rispetto al tempo.

3. Dare la definizione di ground model nella modellazione ASM. Descrivere cosa vuol dire fare un passo di raffinamento del ground model.

Nella modellazione ASM, il ground model di un sistema è il primo modello corretto ma non necessariamente completo dei requisiti. Un ground model descrive i requisiti del sistema in modo evolutivo, consistente, non ambiguo, semplice, conciso e deve essere preciso rispetto al livello di astrazione prescelto e flessibile, in modo da poter essere facilmente modificato ed esteso (fase di raffinamento).
Fare un passo di raffinamento del ground model vuol dire evolvere il ground model tramite composizione (raffinamento orizzontale) o modificando i modelli aggiungendo sempre più dettagli (raffinamento verticale) in modo che la ASM risultante in ogni run raffinata (finita o infinita) simula una run astratta (finita o infinita) tra stati corrispondenti equivalenti

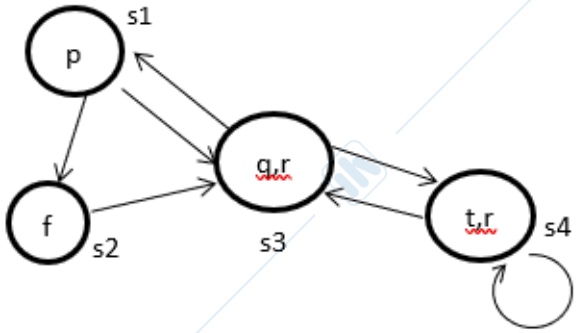
4. Specificare un modello ASM multi agente (luce e controllore) per modellare il comportamento del seguente sistema d'illuminazione dell'atrio di una casa. Se è giorno, la luce non viene accesa. Se è notte, la luce viene accesa per 10 minuti appena viene aperto il cancello di casa. In presenza di malfunzionamento, il sistema va in stato di errore.

17-05-25-Es4.asm

5. Dare la definizione e la formula in logica temporale per esprime le proprietà di *safety*, *liveness*, e *reachability*.

- Safety: Sotto certe condizioni, un evento non può mai accadere (cose cattive non si verificano mai)
  - $AG(\neg\varphi)$ - φ non si verifica mai
  - $A[\neg\varphi\ U\ \psi]$ - φ non si verifica finché non si verifica ψ
- Liveness: Sotto certe condizioni, un evento alla fine accadrà (cose buone prima o poi si verificheranno)
  - $AG(AF(\varphi))$ - In ogni nodo, tutti i percorsi contengono un nodo in cui φ è vera
  - $AG(EF(\varphi))$ - In ogni nodo esiste un percorso in cui φ è verificata
- Reachability: Determinate situazioni particolari possono essere raggiunte
  - $EF(\varphi)$ - Esiste un percorso in cui φ è verificato

6. Dare la definizione di automa di Kripke [pt. 2] e la definizione NuSMV dell'automa di Kripke in figura. [pt. 4]



Risposta nella domanda 2 dell'Esame del 4 Febbraio 2019

Modello NuSMV:

```
MODULE main
VAR
  state: {s1, s2, s3, s4};
DEFINE
f := state in {s2};
p := state in {s1};
q := state in {s3};
r := state in {s3, s4};
t := state in {s4};
  ASSIGN
    init(state) := {s1, s2, s3, s4}
    next(state) :=
      case
        state = s1 : {s2, s3};
        state = s2 : {s3};
        state = s3 : {s1, s4};
        state = s4 : {s3, s4};
      esac
```

1. Dare la definizione di una *locazione* ASM e di *aggiornamento di una locazione*. [pt. 2]

Una locazione in ASM è la coppia $(f, (a_1, ..., a_n))$ e rappresenta matematicamente il valore di $f(a_1, ..., a_n)$ in memoria.
Un aggiornamento è la coppia $((f, (a_1, ..., a_n)), b)$ e significa che l'interpretazione della funzione f in S viene modificata per gli argomenti $(a_1, .., a_n)$ con il valore b.
Un insieme di aggiornamenti è detto *update set*

2. Il modello computazionale ASM prevede diversi aspetti di parallelismo. Descrivili in maniera il più esaustiva possibile. [pt. 4]

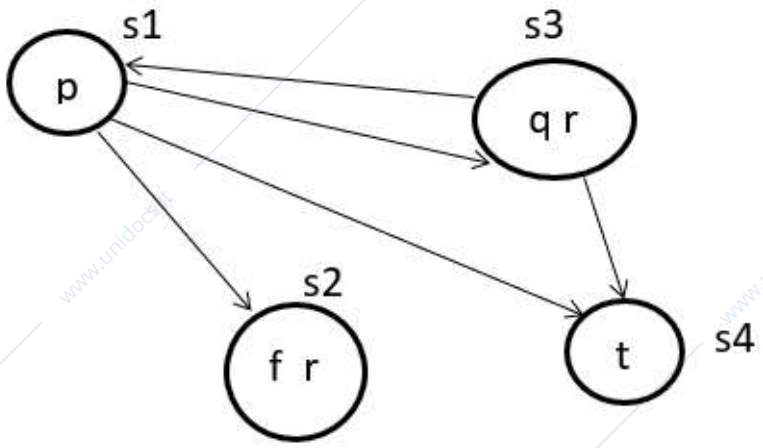
Il parallelismo in ASM è suddivisibile in parallelismo sincrono e asincrono.
Il parallelismo sincrono in ASM può essere rappresentato principalmente in 3 modi.
Il primo è tramite il costruttore *"par"* che permette di eseguire un insieme di azioni in modo parallelo. Il secondo è invece tramite il costrutto *"forall"* (*forall $d in Domain with cond do action*) che esegue l'azione

action in parallelo su ogni elemento *$d* in *Domain* che rispettino la condizione *cond* (parallelismo sincrono non limitato).
Il terzo è tramite l'utilizzo di un modello multi-agente sincrono, che consiste nel suddividere il modello in diverse entità che operano in modo parallelo, ma sincronizzate sullo stesso clock implicito (quasi-sequential run).
Mentre il parallelismo asincrono è rappresentato tramite l'uso di modelli multi-agente asincroni, questo permette ad ogni agente di eseguire il proprio ciclo in modo separato e non legato ad un clock unico.
Questo parallelismo rende molto difficile la gestione degli agenti in quanto non esiste il concetto di controllo dello stato generale (run parzialmente ordinata).

2. Definire l'esecuzione (run) di una ASM multi-agenti sincrona. [pt. 2]

Durante l'esecuzione di una ASM multi-agenti sincrona, gli agenti eseguono il loro programma in parallelo su un implicito clock globale del sistema. L'insieme opera negli stati globali, ottenuti dall'unione di tutti gli stati delle ASM componenti. La sequenza di eventi che determina un'esecuzione è la sequenza degli stati che rappresentano l'esecuzione della sync_ASM
Un'ASM multi-agenti sincrona ha una quasi-sequential run: Una sequenza di stati $S_0, S_1, ..., S_n$ dove ciascun stato S_i è ottenuto dal precedente S_{i-1} eseguendo in parallelo le regole di tutti gli agenti.

6. Dato l'automa di Kripke M in figura [pt. 10]
   a. Dare la definizione formale di automa;Dare una rappresentazione di M in ROBDD;
   b. Formalizzare in logica CTL la proprietà di raggiungibilità *t*.
   c. Formalizzare in logica CTL la proprietà: se vale *l'evento* p *allora nel futuro valgono gli eventi* q *e* f *in almeno un cammino di computazione*.
   d. Verificare con l'algoritmo di model checking se vale la proprietà $M, s \vDash (AG\, t)$



a) Un automa (o macchina) a stati finiti, abbreviata FSM (Finite State Machine), è una notazione formale che permette la rappresentazione astratta del comportamento di un sistema.
Una FSM è una tupla (S, I, δ)

- S: Insieme finito di stati
- I: Insieme finito di eventi di input
- $\delta : S \times I \to S$: Funzione di transizione

b) $f_{\to}(p,f,r,q,t) = ((p \wedge \neg f \wedge \neg r \wedge \neg q \wedge \neg t) \wedge (\neg p \wedge \neg f \wedge r \wedge q \wedge \neg t)) \vee$
$((p \wedge \neg f \wedge \neg r \wedge \neg q \wedge \neg t) \wedge (\neg p \wedge \neg f \wedge \neg r \wedge \neg q \wedge t)) \vee$
$((p \wedge \neg f \wedge \neg r \wedge \neg q \wedge \neg t) \wedge (\neg p \wedge f \wedge r \wedge \neg q \wedge \neg t)) \vee$
$((\neg p \wedge \neg f \wedge r \wedge q \wedge \neg t) \wedge (p \wedge \neg f \wedge \neg r \wedge \neg q \wedge \neg t)) \vee$
$((\neg p \wedge \neg f \wedge r \wedge q \wedge \neg t) \wedge (\neg p \wedge \neg f \wedge \neg r \wedge \neg q \wedge t))$

$S_1 = p \wedge \neg f \wedge \neg r \wedge \neg q \wedge \neg t$
$S_2 = \dots, S_3, S_4$
$S_1 \to S_2 \quad f_{S_1 \to S_2} = S_1 \wedge S_2$
$S_1 \to S_3 \dots$
$f_{\to} = (S_1 \to S_2) \vee (S_1 \to S_3) \vee \dots$

c) Reachability: $M, s \vDash EF(t)$

d) $M, s \vDash AG(p \to EF((q \wedge EF(f)) \vee (f \wedge EF(q))))$
In tutti gli stati, se vale p, allora esiste un cammino (incluso il presente) in cui vale q ed esiste un cammino per f, oppure vale f ed esiste un cammino per q → Se vale p, allora esiste un cammino in cui prima o poi vale q ed f (va bene anche il presente), non necessariamente lo stesso stato (quindi non $EF(q \wedge f)$), ma obbligatoriamente lo stesso cammino (quindi non $EF(q) \wedge EF(f)$).

e) $M, s \vDash (AG\, t) \equiv \neg EF(\neg t)$

- 1: t
- 2: $\neg t$
- 3: $EF(\neg t)$
- 4: $\neg EF(\neg t)$

Etichettatura:

- s1: 2,3
- s2: 2,3
- s3: 2,3

1. Dare la definizione di stato ASM e di update set. [pt. 3]

Fissato un vocabolario Σ, uno stato A del vocabolario Σ è un insieme non vuoto X, detto superuniverso di A, con le interpretazioni dei nomi delle funzioni di Σ

- Se f è una 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

L'update set è un insieme degli aggiornamenti delle locazioni. Un aggiornamento è una coppia $((f,(a_1,...,a_n)),\ b)$ che specifica che il valore contenuto nella locazione $f(a_1,...,a_n)$ deve assumere il valore b.

1. Definire il concetto di stato di una ASM e di aggiornamenti inconsistenti. Portare uno o più esempi di specifiche ASM che causano aggiornamenti inconsistenti tramite sole variabili controllate e tramite anche variabili monitorate. [pt. 5]

Fissato un vocabolario Σ, uno stato A del vocabolario Σ è un insieme non vuoto X, detto superuniverso di A, con le interpretazioni dei nomi delle funzioni di Σ

- Se f è una 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

Un aggiornamento $((f,\ (a_{,...,a_n})),b)$ è inconsistente se esiste, nello stesso update set, un altro aggiornamento $((f,(a_1,...,a_n)),c)$ tale che $b \neq c$.

2. Dare la definizione di run per una ASM di base a singolo agente. [pt. 2]

Una ASM di base a singolo agente ha una run sequenziale, cioè una sequenza di stati $S_0, S_1, ..., S_n$ dove ciascuno stato S_i è ottenuto dal precedente S_{i-1} eseguendo in parallelo le regole dell'ASM.

1. Descrivere come in ASM si può modellare il parallelismo limitato ed il non determinismo. Portare semplici esempi. [pt. 4]

Il parallelismo limitato può essere modellato in ASM tramite il costrutto "*par*", mentre il non determinismo può essere modellato con il costrutto "*choose*" e le funzioni monitorate.

Esempio di parallelismo con par	Esempio di non determinismo con $choose$ e monitorate

signature:
  ...
definitions:
  ...
  main rule r_Main = par
  r_Rule1[]
  r_Rule2[]
  endpar
	signature:
	  dynamic monitored state : Boolean
	  ...
	definitions:
	  ...
	  main rule r_Main =
	  choose $i in {1 : 10} with true do
	  if (state) then r_True[$i]

	else r_False[$i] endif

2. Dare la definizione di run per una ASM multi-agenti sincrona. [pt. 3]

Una ASM multi-agenti sincrona ha una quasi-sequential run, cioè una sequenza di stati $S_1, S_2, ..., S_n$ dove ciascuno stato S_i viene calcolato dallo stato precedente S_{i-1} eseguendo tutte le azioni di tutti gli agenti in parallelo.

```
signature:
        domain Producer subsetof Agent
        domain Consumer subsetof Agent
        dynamic abstract domain Products


        // se l'agente e' in attesa
        dynamic controlled waiting: Agent -> Boolean
        dynamic controlled inBuffer: Products // contenuto del buffer
        dynamic controlled consumedProducts: Consumer ->
Powerset(Products) // prodotti consumati dal consumer

        static producer: Producer
        static consumer: Consumer

main rule r_Main =
                par
                        program(producer)
                        program(consumer)
                endpar

default init s0:
        function waiting($a in Agent) = if($a=consumer) then true
                                        else false endif
        function consumedProducts($c in Consumer) = {} // all'inizio il
consumatore non ha consumato nulla

        agent Producer:
                r_produce[]

        agent Consumer:
                r_consume[]
```