Esercitazione guidata ASM

Esercizio Tema d'esame Giugno 2021

Un forno può essere in standby (chiuso e spento), con la porta aperta (ma spento) oppure acceso (la porta deve essere chiusa). Modella questi stati e le opportune azioni con ASMETA. Prova le seguenti proprietà con LTL o CTL a tua scelta:

- Quando è acceso la porta è sempre chiusa.
- Prima o poi si può accendere in qualsiasi momento in futuro.
- La porta può essere aperta dopo che viene acceso.
- Quando è acceso, la porta rimane chiusa fino quando rimane acceso (usa Until).

C'è qualche proprietà che invece è giustamente falsa e il cui controesempio ti aiuta a capire come funziona il forno?

Scrivi poi i seguenti scenari, controllando che il valore delle funzioni sia quello atteso:

- Il forno inizialmente è spento. L'utente apre la porta e poi la chiude. Il forno viene acceso (controlla che la porta sia chiusa quando viene acceso)
- Il forno inizialmente è spento. Il forno viene acceso e l'utente prova ad aprire la porta. Controllare che la porta non sia aperta.

Definizione delle funzioni controllate e monitorate

- Monitorate: lette (non aggiornate) dalla macchina, scritte dall'ambiente
  - comandoStatoPorta: stato della porta, controllato dall'utente che può aprirla o chiuderla, booleano
  - accendi: comando di accensione da parte dell'utente, booleano
  - spegni: comando di spegnimento da parte dell'utente, booleano
- Controllate: lette e scritte dalla macchina
  - statoForno: stato del forno (ACCESO, SPENTO)
  - statoPorta: stato del forno (APERTA, CHIUSA)

Definizione dei domini

Utilizziamo due domini enumerativi per gestire le due variabili collegate:

- Stato = {ACCESO | SPENTO}
- StatoPorta = {APERTA | CHIUSA}

Modello ASMETA

```
asm Forno

import StandardLibrary
import CTLlibrary
import LTLlibrary


signature:
        // DOMAINS
        enum domain Stato = {ACCESO | SPENTO}
        enum domain StatoPorta = {APERTA | CHIUSA}
        // FUNCTIONS
```

```
        controlled statoForno: Stato
        controlled statoPorta: StatoPorta
        monitored comandoStatoPorta: Boolean
        monitored accendi: Boolean
        monitored spegni: Boolean

definitions:
        // DOMAIN DEFINITIONS

        // FUNCTION DEFINITIONS

        // RULE DEFINITIONS
        rule r_accendiForno =
                statoForno := ACCESO

        rule r_spegniForno =
                statoForno := SPENTO

        // INVARIANTS

        // PROPERTIES

        // MAIN RULE
        main rule r_Main =
                par
                        // Se il forno è spento e con la porta chiusa, e ricevo
                        // il comando di accensione, posso accenderlo
                        if statoForno = SPENTO and statoPorta = CHIUSA and accendi then
                                r_accendiForno[]
                        else
                                // Se il forno è acceso e ricevo il comando di
                                // spegnimento
                                if statoForno = ACCESO and spegni then
                                        r_spegniForno[]
                                endif
                        endif

                        // Se ricevo il comando di apertura porta, la apro solo
                        // se il forno è spento
                        if comandoStatoPorta then
                                if statoForno != ACCESO and not accendi then
                                        statoPorta := APERTA
                                endif
                        else
                                statoPorta := CHIUSA
                        endif

                endpar

// INITIAL STATE
default init s0:
        function statoForno = SPENTO
        function statoPorta = CHIUSA
```

In alternativa, potremmo fare anche un modello in cui la porta può essere aperta anche se il forno è acceso. In questo caso, quando la porta viene aperta andiamo a spegnere il forno. La main rule, nella parte che gestisce l'apertura del forno, andrà modificata nel modo seguente:

```
// Se ricevo il comando di apertura porta, la apro solo
// se il forno è spento
if comandoStatoPorta then
        if statoForno != ACCESO and not accendi then
                statoPorta := APERTA
                r_spegniForno[]
        endif
else
        statoPorta := CHIUSA
endif
```

Proprietà

DESCRIZIONE	PROPRIETA'	SIGNIFICATO
Quando è acceso la porta è sempre chiusa	**CTLSPEC** ag(statoForno = ACCESO **implies** statoPorta = CHIUSA)	Lungo tutti i cammini, globalmente, se il forno è acceso, allora la porta è chiusa
Prima o poi si può accendere in qualsiasi momento in futuro	**CTLSPEC** ef (statoForno = ACCESO)	Esiste almeno un cammino in cui, nel futuro, il forno è acceso
	CTLSPEC af (statoForno = ACCESO)	Lungo tutti i cammini, nel futuro, il forno è acceso. Questa proprietà è **falsa** perché se non do mai il comando di accensione non può accendersi.
La porta può essere aperta dopo che viene acceso	**CTLSPEC** eg (statoForno = ACCESO **implies** ef (statoPorta = APERTA))	Esiste un cammino in cui, dopo aver acceso il forno si ha uno stato futuro in cui la porta è aperta. Attenzione che questo richiede che in quel momento il forno sia spento. Infatti, la proprietà **CTLSPEC** eg (statoForno = ACCESO **implies** ef (statoPorta = APERTA **and** statoForno = SPENTO)) è falsa
Quando è acceso, la porta rimane chiusa fino quando rimane acceso (usa Until)	**CTLSPEC** ag (statoForno = ACCESO **implies** a(statoPorta = CHIUSA, statoForno = ACCESO))	Lungo tutti i cammini, se il forno è acceso, allora la porta rimane chiusa fino a quando il forno è acceso. Nota che "a" nella CTLlibrary è l'equivalente di "au"

Scenari

Scenario 1

```
scenario Forno1

load Forno.asm
```

```
// Inizialmente il forno è spento
check statoForno = SPENTO;
// Apro la porta (e imposto a false gli input di accensione
// e spegnimento)
set comandoStatoPorta := true;
set spegni := false;
set accendi := false;

step

// Controllo che il forno sia sempre spento ma che la porta
// ora sia aperta
check statoForno = SPENTO;
check statoPorta = APERTA;
// Chiudo la porta (e imposto a false gli input di accensione
// e spegnimento)
set comandoStatoPorta := false;
set spegni := false;
set accendi := false;

step

// Controllo che il forno sia sempre spento e che ora anche
// la porta sia chiusa
check statoForno = SPENTO;
check statoPorta = CHIUSA;
// Accendo il forno
set accendi := true;
set spegni := false;

step

// Controllo che il forno si sia acceso e che la porta sia
// ancora chiusa
check statoForno = ACCESO;
check statoPorta = CHIUSA;
```

Scenario 2

```
scenario Forno2

load Forno.asm

// Inizialmente il forno è spento
check statoForno = SPENTO;
// Accendo il forno
set accendi := true;
set spegni := false;
set comandoStatoPorta := false;

step
```

```
// Controllo che il forno sia acceso e che la porta sia rimasta
// chiusa
check statoForno = ACCESO;
check statoPorta = CHIUSA;
// Provo ad aprire la porta
set comandoStatoPorta := true;
set spegni := false;
set accendi := false;

step

// Controllo che il forno sia sempre acceso e che la porta non
// si sia aperta
check statoForno = ACCESO;
check statoPorta = CHIUSA;
```

Esercizio Tema d'esame Settembre 2021

Scrivi un modello ASM per un semaforo che però diventa verde solo su richiesta (usa ad esempio con una monitorata boolean). Il semaforo fa il ciclo regolare: ROSSO -> VERDE -> GIALLO ->ROSSO.

Prova ad animare il sistema e fai uno screenshot del comportamento.

Prova le seguenti proprietà sia come LTL che come CTL:

- Non può mai passare a VERDE direttamente da ROSSO
- Quando è ROSSO rimarrà sempre ROSSO a meno che ci sia una richiesta
- Se c'è una richiesta allora prima o poi diventa VERDE.
- In qualsiasi istante, prima o poi potrebbe diventare VERDE.

Se qualcuna è falsa spiega perché. Scrivi e spiega anche una tua proprietà vera e una falsa il cui controesempio ti aiuta a capire come funziona il semaforo.

Scrivi poi i seguenti scenari, controllando che il valore delle funzioni sia quello atteso:

- Il semaforo è rosso, per uno stato rimane ancora rosso e non si ricevono richieste di passaggio. Arriva poi una richiesta di passaggio ed il semaforo passa sul verde. Lo stato successivo scatta il timer e si passa sul giallo.
- Il semaforo è rosso, poi si riceve una richiesta di passaggio e passa sul verde. Forzare manualmente lo stato del semaforo di nuovo sul rosso e controllare che sia avvenuto il passaggio corretto.

Definizione delle funzioni controllate e monitorate

- Monitorate: lette (non aggiornate) dalla macchina, scritte dall'ambiente
  - richiestaPassaggio: comando azionato dall'utente per richiedere il passaggio al verde, booleano.
  - timerPassed: timer che indica se è scaduto il tempo per cui si deve passare allo stato successivo, booleano.
- Controllate: lette e scritte dalla macchina
  - statoSemaforo: stato interno del semaforo (ROSSO, VERDE, GIALLO)

Definizione dei domini

Utilizziamo un dominio enumerativi per gestire lo stato del semaforo

- Stato = {ROSSO | VERDE | GIALLO}

Modello ASMETA

```
asm Semaforo

import StandardLibrary
import CTLlibrary
import LTLlibrary

signature:
	// DOMAINS
	enum domain Stato = {VERDE | GIALLO | ROSSO}

	// FUNCTIONS
	controlled statoSemaforo: Stato
	monitored richiestaPassaggio: Boolean
	monitored timerPassed: Boolean

definitions:
	// DOMAIN DEFINITIONS

	// FUNCTION DEFINITIONS

	// RULE DEFINITIONS

	// INVARIANTS

	// MAIN RULE
	main rule r_Main =
		// Se la luce è ROSSA e arriva la richiesta dell'utente
		if statoSemaforo = ROSSO and richiestaPassaggio then
			statoSemaforo := VERDE
		else
			// Se la luce è VERDE ed il tempo è passato
			if statoSemaforo = VERDE and timerPassed then
				statoSemaforo := GIALLO
			else
				// Se la luce è GIALLA ed il tempo è passato
				if statoSemaforo = GIALLO and timerPassed then
					statoSemaforo := ROSSO
				endif
			endif
		endif

// INITIAL STATE
default init s0:
	function statoSemaforo = ROSSO
```

Animazione

	Type	Functions		State 0	State 1	State 2	State 3	State 4	State 5	State 6	State 7
	M	richiestaPassaggio		false	true	false	false	false	false	false	
	M	timerPassed		false	false	false	true	false	true	false	
	C	statoSemaforo		ROSSO	ROSSO	VERDE	VERDE	GIALLO	GIALLO	ROSSO	ROSSO

Proprietà

DESCRIZIONE	PROPRIETA'	SIGNIFICATO
Non può mai passare a VERDE direttamente da ROSSO	**CTLSPEC** ag (statoSemaforo = ROSSO **implies** ax (statoSemaforo != VERDE))	Lungo tutti i cammini, se lo stato è ROSSO, allora non è possibile (lungo nessun cammino) che nello stato successivo sia VERDE. Questa è **falsa** per come è definito il semaforo. Il testo originale dell'esame era sbagliato (aveva giallo dopo il rosso) quindi nel testo originale sarebbe stata provata correttamente.
	CTLSPEC ag (statoSemaforo = VERDE **implies** ax (statoSemaforo != ROSSO))	Lungo tutti i cammini, se lo stato è VERDE, allora non è possibile (lungo nessun cammino) che nello stato successivo sia ROSSO
Quando è ROSSO rimarrà sempre ROSSO a meno che ci sia una richiesta	**CTLSPEC** ag (statoSemaforo = ROSSO **implies** aw (statoSemaforo = ROSSO, richiestaPassaggio))	Lungo tutti i cammini, se il semaforo è ROSSO, rimane ROSSO almeno fino a quando si riceve una richiesta di passaggio. Nota che usiamo "aw" perché la richiesta di passaggio potrebbe non arrivare mai.
Se c'è una richiesta allora prima o poi diventa VERDE	**CTLSPEC** ag (richiestaPassaggio **implies** ef (statoSemaforo = VERDE))	Lungo tutti i cammini, se viene richiesto il passaggio, in futuro il semaforo può diventare prima o poi VERDE
	CTLSPEC ag (richiestaPassaggio **implies** af (statoSemaforo = VERDE))	Lungo tutti i cammini, se viene richiesto il passaggio, in futuro il semaforo diventa prima o poi VERDE. Questa è **falsa** perché il comando viene considerato solo se siamo con il semaforo nella luce ROSSA. Per averla vera dovremmo fare: **CTLSPEC** ag ((richiestaPassaggio and statoSemaforo = ROSSO) **implies** af (statoSemaforo = VERDE))
In qualsiasi istante, prima o poi potrebbe diventare VERDE	**CTLSPEC** ef (statoSemaforo = VERDE)	Esiste almeno un cammino in cui, nel futuro, il semaforo è VERDE

Scenari

Scenario 1

```
scenario Semaforo1

load Semaforo.asm

// Controllo lo stato iniziale
check statoSemaforo = ROSSO;
set timerPassed := false;
set richiestaPassaggio := false;

step

// Controllo che sia rimasto rosso
check statoSemaforo = ROSSO;
set richiestaPassaggio := true;
set timerPassed := false;

step

// controllo il passaggio sul verde
check statoSemaforo = VERDE;
set timerPassed := false;
set richiestaPassaggio := false;

step

// Arriva lo scattare del timer
set timerPassed := true;
set richiestaPassaggio := false;

step

// controllo il passaggio sul giallo
check statoSemaforo = GIALLO;
```

Scenario 2

```
scenario Semaforo2

load Semaforo.asm

// Controllo lo stato iniziale
check statoSemaforo = ROSSO;
// Ricevo una richiesta di passaggio
set richiestaPassaggio := true;
set timerPassed := false;

step

// Controllo il passaggio sul verde
check statoSemaforo = VERDE;
```

```
// Forzo il passaggio sul rosso
exec statoSemaforo := ROSSO;
set timerPassed := false;
set richiestaPassaggio := false;

step

// Controllo che sia effettivamente tornato rosso
check statoSemaforo = ROSSO;
```