Tema esame inventato - 1

1. Testing di un programma

Dato il seguente programma in Java:

```
boolean conditionChecker(int a[], int c1, int c2) {
        boolean res = true;
        for (int i = 0; i < a.length; i++) {
                if (!(c1 <= a[i] && a[i] <= c2)) {
                        res = false;
                        break;
                }
        }
        return res;
}
```

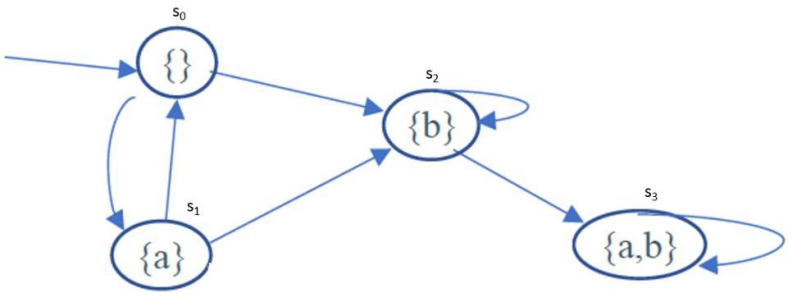
Scrivere i casi di test per:

1. Copertura delle istruzioni
2. Copertura dei branch
3. Copertura delle condizioni
4. Copertura MCDC

Cerca di **minimizzare** i casi di test necessari per avere queste coperture. Non è necessario che tu scriva i test come JUnit (però deve essere possibile scrivere i test JUnit dai tuoi test).

2. Algoritmo di model checking

Data la seguente macchina M



Mediante l'algoritmo di model checking, dire in quali stati s valgono le proprietà:

- M,s |= AG(a and b)
- M,s |= EX(a and b) [ATTENZIONE]
- M,s |= AF(a or b)

3. Combinatorial testing

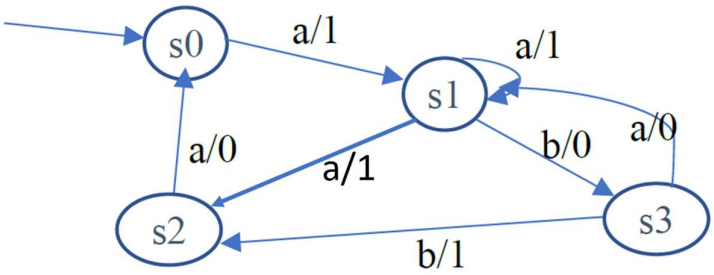
Date tre variabili con i loro domini

S: 1,2,3 **T**: a,b **V**:A,B,C,D

Costruisci la test suite combinatoriale pairwise usando l'algoritmo IPO.

4. Conformance testing

Data la seguente FSM, due input a e b e due output 1 e 0.



La macchina è correttamente definita? (giustifica la risposta)
Scrivi due test sequences, una per la copertura degli stati e una per la copertura delle transizioni.
Fa due esempi di errori (di quelli visti) e controlla se riesci a scoprirli con i test che hai trovato tu (può succedere che tu non riesca a scoprirli?) Giustifica la risposta

5. JML e Key

Riscrivi il programma dell'esercizio 1 in cui metti gli opportuni contratti e provi la correttezza. Puoi semplificare il codice se i contratti che metti rendono superflui alcuni controlli. Puoi riscrivere il codice a tuo piacimento per portare a termine la dimostrazione. Ricordati la sintassi dei quantificatori:

(\forall <dominio>;<range_valori>;<condizione>)
(\exists <dominio>;<range_valori>;<condizione>)

ATTENZIONE: verificare anche che c1 sia minore uguale di c2 (all'uguaglianza tutti i numeri dell'array devono essere uguali a c1 e c2).

6. Logica temporale con Asmeta

Dato il seguente modello Asmeta:

```
asm Semaforo

import StandardLibrary
import CTLlibrary
import StandardLibrary

signature:

enum domain ColoreSemaforo = {ROSSO | VERDE | GIALLO}
enum domain LuceSemaforo = { LIGHT1 | LIGHT2}

// controllata: le luci dei dei due semafori
controlled light: LuceSemaforo -> ColoreSemaforo
// monitorata: quale semaforo deve cambiare
monitored changeLight: LuceSemaforo
// data una luce mi restituisce l'altra
static otherLight: LuceSemaforo -> LuceSemaforo

definitions:

function otherLight($l in LuceSemaforo) =
                                if $l = LIGHT1 then LIGHT2 else LIGHT1 endif

// setta a verde
macro rule r_verde = if light(changeLight) = ROSSO and
            light(otherLight(changeLight)) = ROSSO then light(changeLight):= VERDE endif

main rule r_changeLight =
        par
                r_verde[]
         if light(changeLight) = VERDE then light(changeLight):= GIALLO endif
         if light(changeLight) = GIALLO then light(changeLight):= ROSSO endif
```

```
        endpar

default init s0:
function light($l in LuceSemaforo) = ROSSO
```

Verificare le seguenti proprietà:

1. non accade mai che i semafori siano entrambi verdi
2. la luce 2 può diventare sempre verde (non solo allo stato iniziale)
3. se un semaforo è verde allora l'altro è rosso
4. se light1 è rosso e viene scelto e light2 è rosso, allora nello stato successivo diventa verde
5. la luce 1 non può mai essere verde
6. i semafori sono sempre tutti rossi

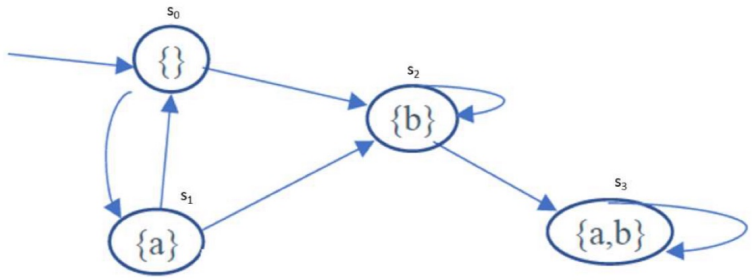
Se c'è qualche proprietà che invece è giustamente falsa e il cui controesempio ti aiuta a capire come funziona, spiegalo.

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

- I due semafori sono inizialmente rossi. Dai il segnale per settare il semaforo 1 a verde e assicurati che l'altro rimanga rosso. Assicurati che il semaforo 1 passi al giallo e poi al rosso di nuovo e il semaforo 2 nel mentre rimanga sempre rosso.
- Setta il segnale changeLight a LIGHT1 e fai diventare il semaforo giallo. Poi setta forzatamente il semaforo 1 a verde prima che diventi nuovamente rosso. Assicurati che dopo essere diventato verde, il semaforo 1 passi ancora a giallo. Assicurati che il semaforo 2 rimanga sempre rosso nel mentre.

2. Algoritmo di model checking

Data la seguente macchina M



Mediante l'algoritmo di model checking, dire in quali stati s valgono le proprietà:

- M,s |= AG(a and b)
- M,s |= EX(a and b) [ATTENZIONE]
- M,s |= AF(a or b)

1) $AG(a \wedge b) = \neg E[true \; U \; \neg(a \wedge b)]$

	S_0	S_1	S_2	S_3
a		x		x
b			x	x
$a \wedge b$				x
$\neg(a \wedge b)$	x	x	x	
$E[true \; U \; \neg(a \wedge b)]$	x	x	x	
$\neg E[..]$				x

2) $EX(a \wedge b)$

	S_0	S_1	S_2	S_3
$a \wedge b$				x
$EX(a \wedge b)$			x	x

3) AF(a ∨ b)

	S_0	S_1	S_2	S_3
a		x		x
b			x	x
a ∨ b		x	x	x
AF(a ∨ b)	x	x	x	x

3. Combinatorial testing

Date tre variabili con i loro domini

S: 1,2,3 **T**: a,b **V**:A,B,C,D

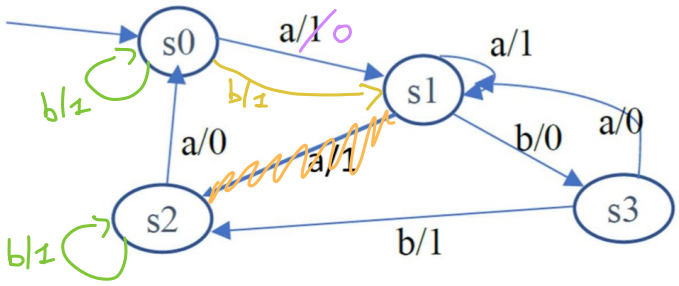
Costruisci la test suite combinatoriale pairwise usando l'algoritmo IPO.

S	T	V
1	a	A
2	a	B
3	a	C
1	b	B
2	b	A
3	b	D
1	b	C
2	a	C
3	a	A
1	a	D
2	a	D
3	a	B

1) S-T ✓
2) S-V ✓
T-V ✓

4. Conformance testing

Data la seguente FSM, due input a e b e due output 1 e 0.



La macchina è correttamente definita? (giustifica la risposta)
Scrivi due test sequences, una per la copertura degli stati e una per la copertura delle transizioni.
Fa due esempi di errori (di quelli visti) e controlla se riesci a scoprirli con i test che hai trovato tu (può succedere che tu non riesca a scoprirli?) Giustifica la risposta

☑ completamente definita ☑ deterministica

- Cop. stati : a b b a = TS1
  => output atteso : 1010
- Cop. transizioni : a a b a b b b a b = TS2
  => output atteso : 110011101
- Output error : S_0 -> a/0 -> S_1
  => trovo sia con TS1 che con TS2,
  infatti output : cop. stati = 0010 ≠ TS1
  cop. transizioni = 0... ≠ TS2
- Transfer error : S_0 -> b/1 -> S_1
  => non trovo né con TS1, né con TS2, serve status