Tema esame inventato - 2

1. Testing di un programma

Dato il seguente programma in Java che restituisce true se la somma dei valori negativi e dispari contenuti nell'array *arr* è minore o uguale a *val* (con *val*=numero strettamente negativo):

```
// arr: può contenere sia valori positivi che negativi
// val: è un valore negativo (strettamente)
public boolean oddNegSumLessX(int[] arr, int val) {
	if(val>=0)
		return false;

	int sum=0;

	for(int x : arr)
		if(x%2!=0 && x<0)
		sum+=x;

	if(sum<=val)
		return true;
	else
		return false;
}
```

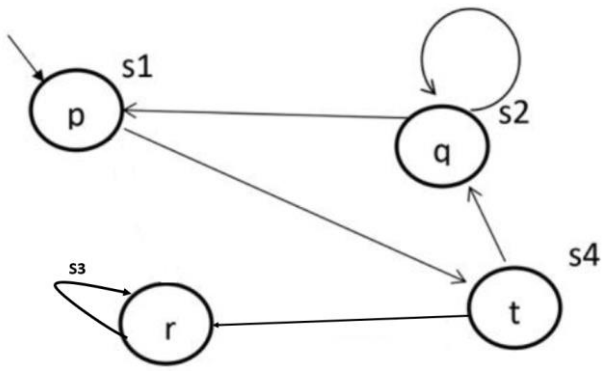
Scrivere i casi di test per:

1. Copertura delle istruzioni
2. Copertura dei branch
3. Copertura delle decisioni
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 |= AF(p or q)
- M,s |= AG(r)
- M,s |= EX(q or t)

3. Combinatorial testing

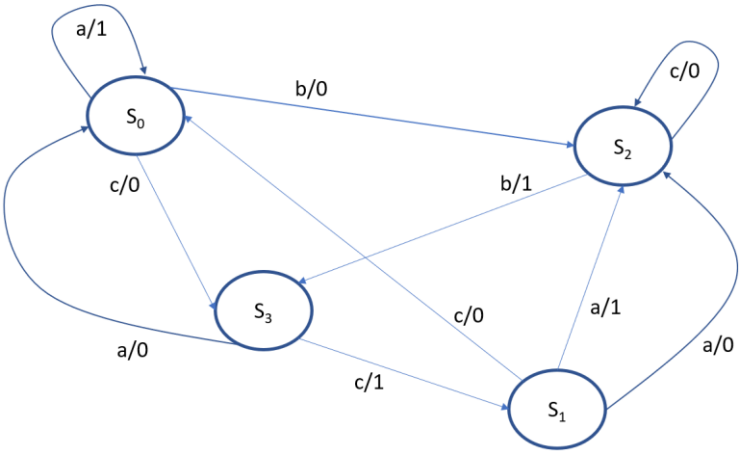
Date quattro variabili con i loro domini

D1: A,B **D2**: 6,7,8 **D3**:L,M **D4**:9,2,4

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

4. Conformance testing

Data la seguente FSM, tre input {a,b,c} e due output {1,0}.



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>)

6. Logica temporale con Asmeta

Un edificio ha 2 piani e un ascensore permette di visitarli mediante l'algoritmo *sabbath*. L'algoritmo prevede che l'ascensore si muova automaticamente dal primo al secondo piano e viceversa (1->2->1->2->..). Quando il bottone *signalPorta* viene premuto (valore *true*), l'ascensore apre le porte e rimane allo stesso piano fino a quando *signalPorta* torna al valore *false*. Tornato al valore *false*, l'ascensore chiude le porte e continua con l'algoritmo *sabbath*.

Prova le seguenti proprietà con LTL o CTL a tua scelta:

1. Se il bottone viene premuto, nel prossimo stato la porta sarà aperta.
2. Prima o poi la porta si aprirà.
3. Quando la porta è aperta, rimane aperta fino a quando *signalPorta* torna a *false* (**se torna a *false***).
4. Il piano nello stato successivo rimane lo stesso se *signalPorta* è *true* (scrivi una proprietà che valga sia per il piano 1 che per il piano 2)

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

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

- L’ascensore parte dal piano 1 con la porta chiusa. Nessun segnale viene attivato e l’ascensore raggiunge il piano 2 mantenendo le porte chiuse. L’ascensore raggiunge il piano 1: il segnale viene attivato e la porta si apre.
- L’ascensore parte dal piano 1 e raggiunge il piano 2. Raggiunto il piano 2, viene forzata l’apertura della porta mantenendo *signalPorta* a false. Nel prossimo stato l’ascensore chiude comunque le porte e va al piano 1.