Documentazione

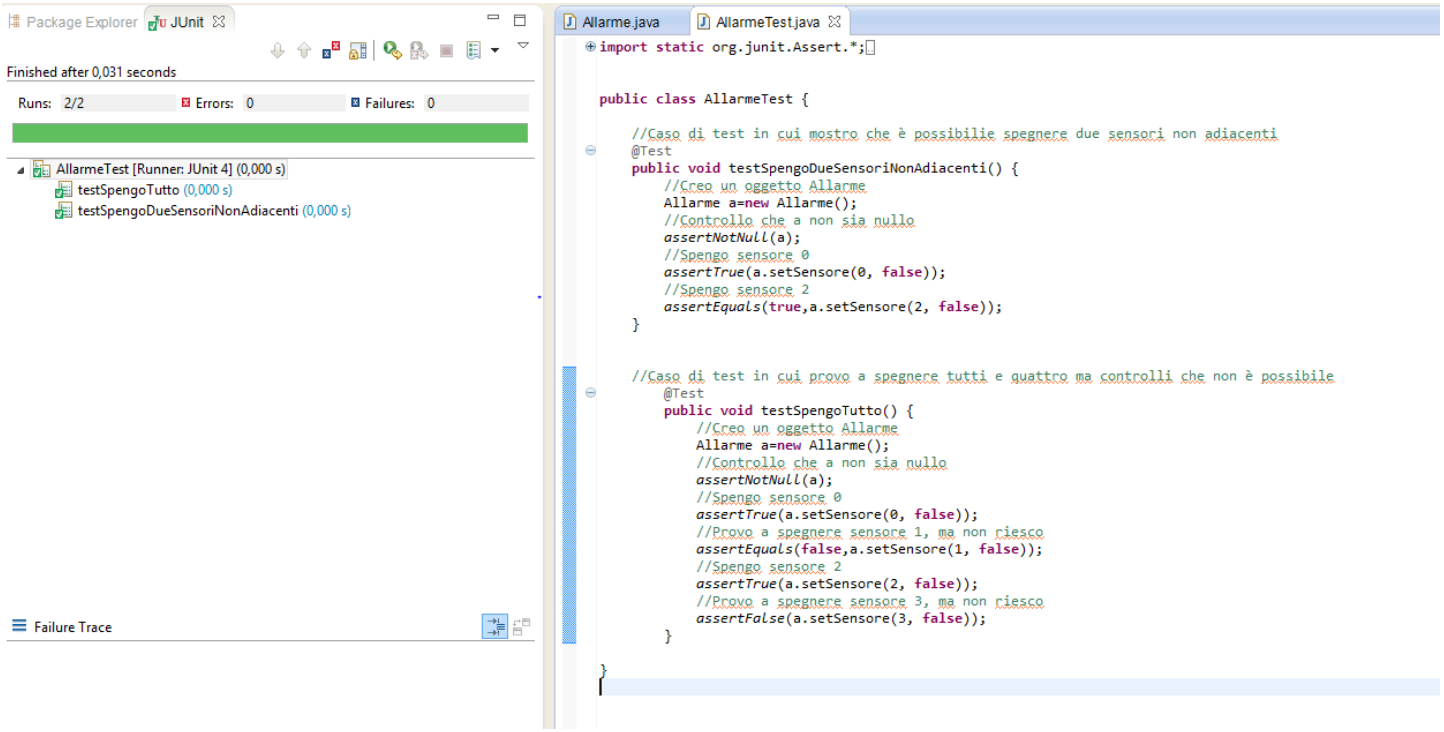
Sommario

Esercizio 1 - TESTING

L'esercizio 1 consisteva nella realizzazioni della classe … in grado di simulare funzionamento del … descritto nel tema d'esame e testarla attraverso il tool JUnit.

Scenario AllarmeTest

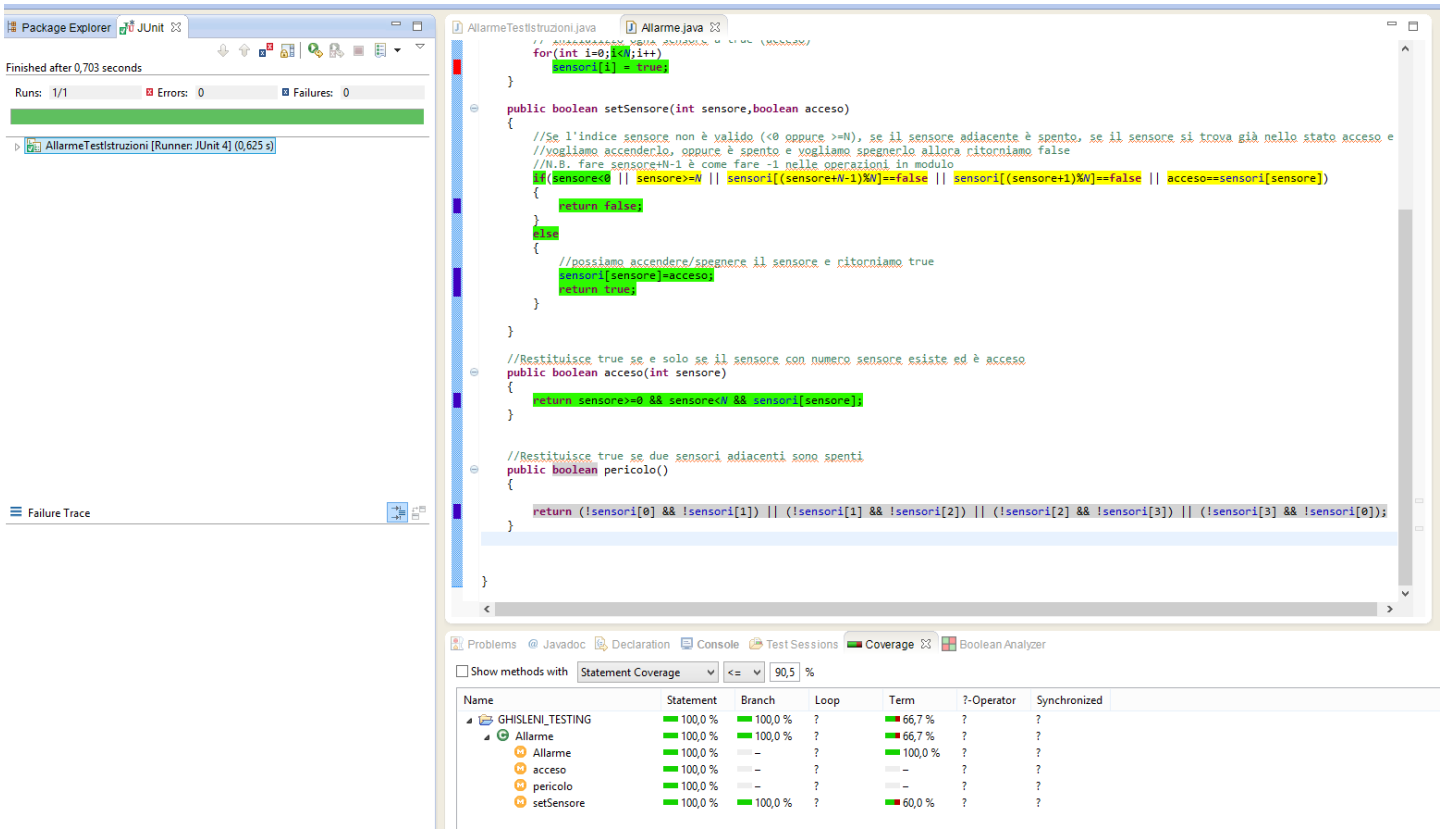
Scrivo un caso di test relativo allo scenario AllarmeTest, lo eseguo con JUnit e controllo il risultato.



Il risultato, come possiamo notare dalla barra verde, è positivo.

Copertura istruzioni (Statement coverage)

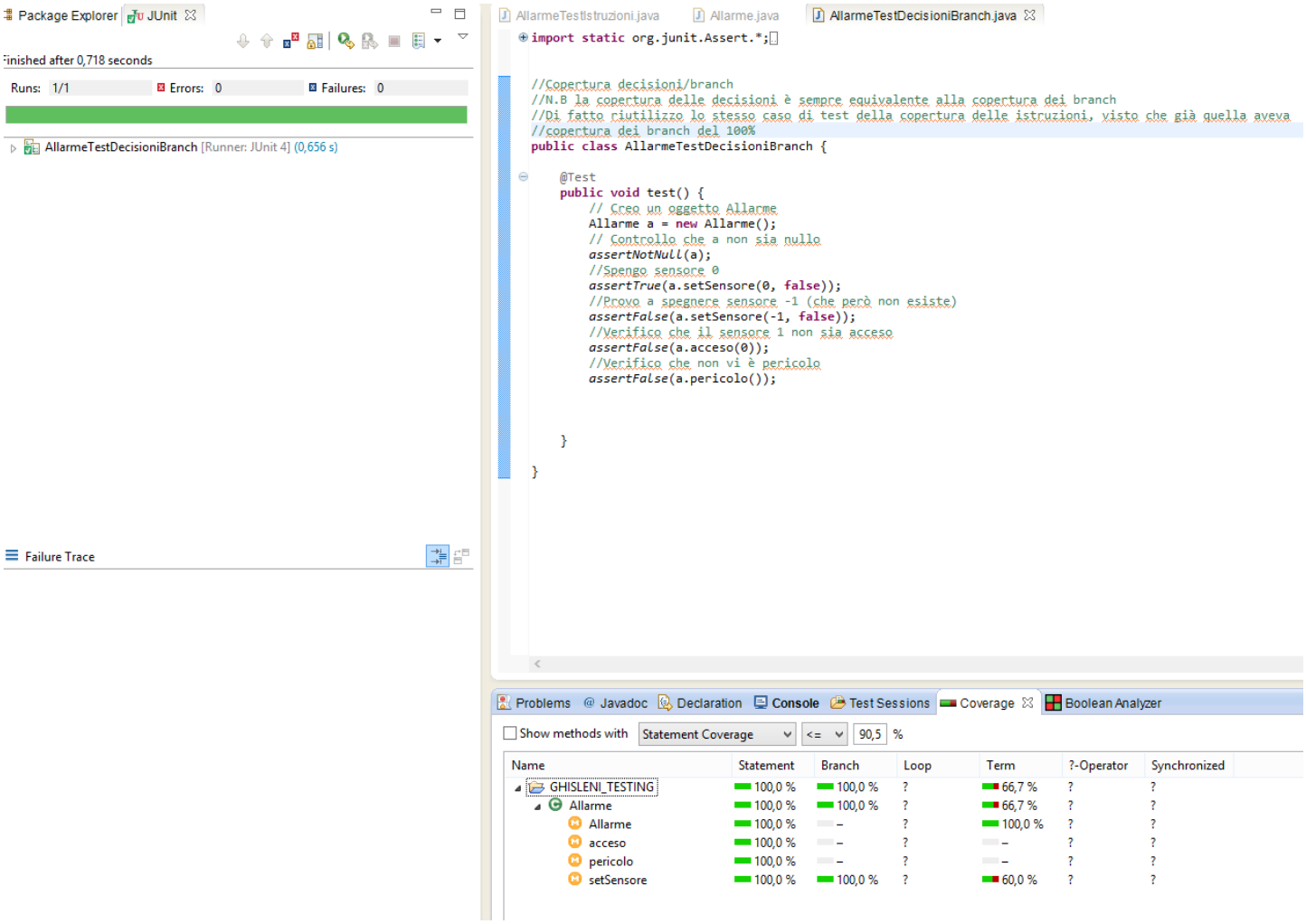
Scrivo un caso di test relativo alla copertura delle istruzioni e lo eseguo con CodeCover.



La copertura relativa alle istruzioni (chiamata Statement in CodeCover) è del 100%.

Copertura archi/decisioni (Branch coverage/Decision coverage)

Scrivo un caso di test relativo alla copertura delle decisioni/archi e lo eseguo con CodeCover.



La copertura relativa alle decisioni/archi (chiamata Branch in CodeCover è del 100%).

Copertura MCDC (Modified Condition/decision coverage)

Nel MCDC il test set deve essere preso in modo che ogni condizione all'interno di una decisione deve far variare in modo indipendente il valore finale della decisione.

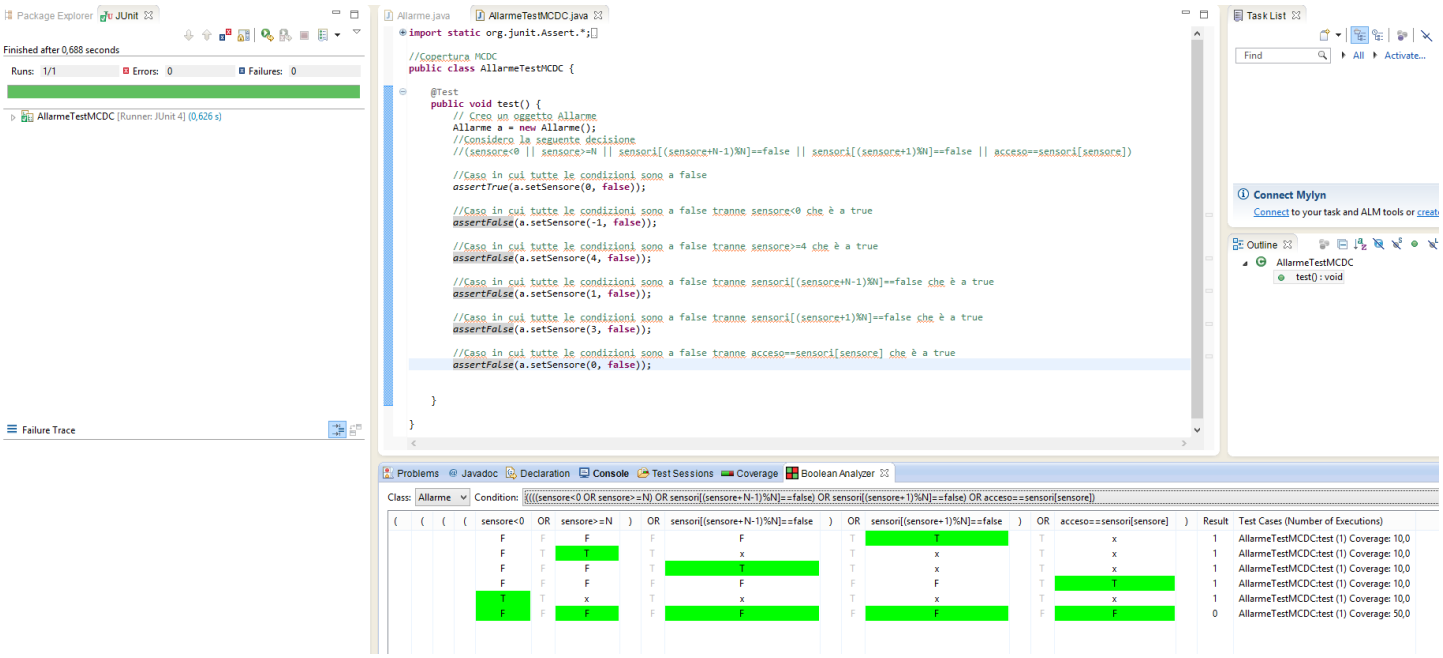
Scrivo un caso di test relativo alla copertura MCDC della decisione

(sensore<0 || sensore>=N || sensori[(sensore+N-1)%N]==false || sensori[(sensore+1)%N]==false || acceso==sensori[sensore])

e lo eseguo con CodeCover.

Osservando il Boolean Analyzer vediamo che vengono valutati:

- un caso in cui avendo tutte le condizioni false la decisione finale fa si che si abbia return true;
- gli altri casi in cui una singola condizione è vera e tutte le altre condizioni sono false (o non valutate a causa della short circuit evaluation), in questi casi si ottiene che la decisione finale fa si che si abbia return false.

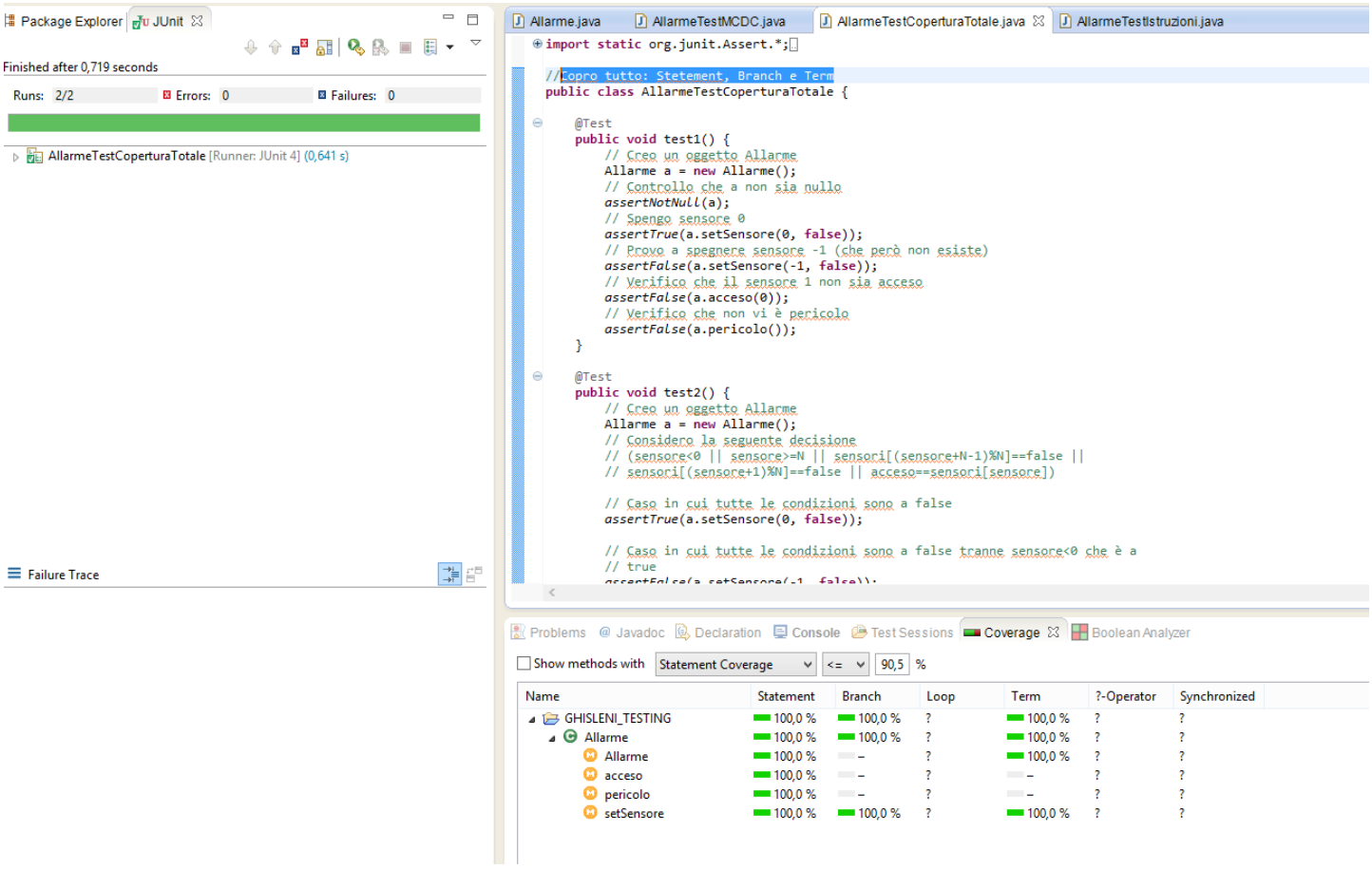


Quindi effettivamente abbiamo copertura al 100% di MCDC della decisione

(sensore<0 || sensore>=N || sensori[(sensore+N-1)%N]==false || sensori[(sensore+1)%N]==false || acceso==sensori[sensore]) . Inoltre si ha il minor numero di casi di test per la copertura MCDC pari a n+1, dove n è il numero di condizioni all'interno della decisione. In questo caso abbiamo 5 condizioni e 6 casi di test.

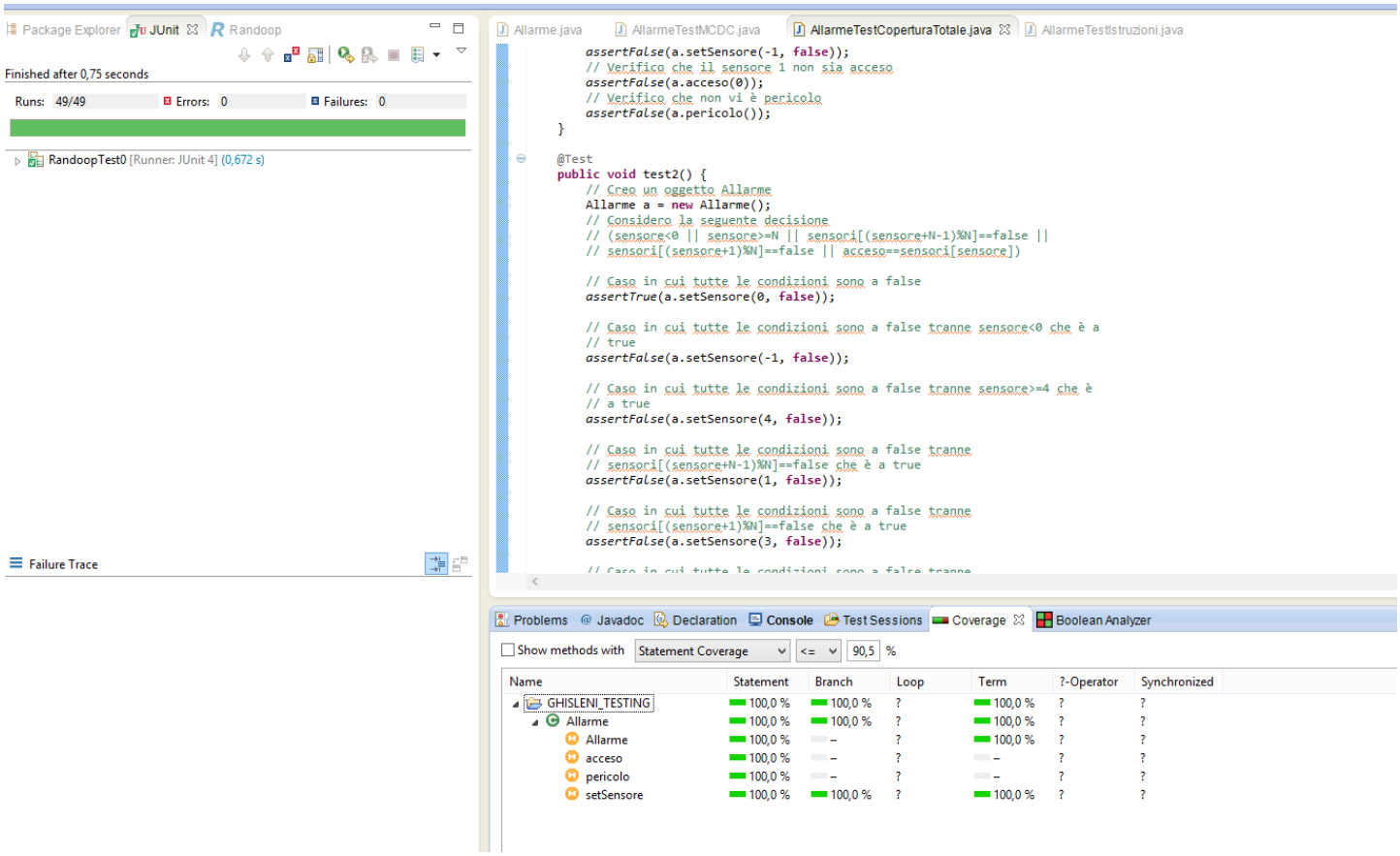
Copertura totale

Copro tutto: Stetement, Branch e Term.



Randoop

Genero in modo automatico con il tool randoop 49 casi di test. Controllo la copertura con CodeCover sui casi di test generati da randoop e osservo che la copertura risulta essere 100% degli Stetement, Branch e Term.



Esercizio 2 - JML

In questo esercizio introduciamo l'uso di contratti all'interno della nostra classe Java, per fare ciò usiamo JML.

Precondizioni (obblighi per il cliente)

Converto tutte le condizioni date dagli if all'interno dei metodi in precondizioni. Per comodità lascio anche gli if di controllo dentro i metodi (anche se non più necessarie, visto che adesso sono sostituite dalle precondizioni) per poter eseguire le violazioni di precondizioni senza generare eccezioni Java.

Guardare il codice per vedere le precondizioni aggiunte.

Postcondizioni (obblighi per il fornitore)

Guardare il codice per vedere le postcondizioni aggiunte.

spec_public

Ai campi privati della classe aggiungo /*@ spec_public @*/.

Invarianti (condizione che è sempre vera)

Guardare il codice per vedere gli invarianti aggiunti alla classe.

Dopo aver scritto i contratti scrivo una classe Main nella quale testare il corretto funzionamento della classe e provo a violare precondizioni, postcondizioni, invarianti.

Violazione di precondizioni

Per poter violare una precondizione chiamiamo dal main un metodo della classe con contratti passando come parametri dei valori che non rispettano la precondizione.

```
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Main.java:13: JML precondition is false
                a.setSensore(-1, false);
                             ^
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:37: Associated declaration: D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Main.java:13:
        //@ requires sensore>=0 && sensore<N && sensori[(sensore+N-1)%N]==true && sensori[(sensore+1)%N]==true && acceso!=sensori[sensore];
            ^
```

```
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Main.java:14: JML precondition is false
                a.acceso(-1); //Violo requires sensore>=0 && sensore<N;
                         ^
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:60: Associated declaration: D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Main.java:14:
        //@ requires sensore>=0 && sensore<N;
            ^
```

Violazione di postcondizioni

Per poter violare una postcondizione introduciamo un errore nel codice della classe, in modo tale che la postcondizione non sia più vera.

```
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:22: JML postcondition is false
        public Allarme()
               ^
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:21: Associated declaration: D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:22:
        //@ensures (\forall int i ; i >= 0 && i < N ; sensori[i] == true);
           ^
```

```
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:89: JML postcondition is false
        public boolean acceso(int sensore)
                       ^
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:88: Associated declaration: D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:89:
        //@ensures \result ==> (sensore>=0 && sensore<N && sensori[sensore]);
           ^
```

```
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:99: JML postcondition is false
        public boolean pericolo()
                       ^
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:98: Associated declaration: D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:99:
        //@ensures \result ==> (\forall int i; i >= 0 && i < N; !sensori[i] && !sensori[(i+1)%N]);
           ^
```

Violazione di invarianti

Per poter violare un invariante introduciamo un errore nel codice della classe, in modo tale che l'invariante non sia più vero.

```
D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:11: Associated declaration: D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Main.java:7:
        //@ public invariant sensori != null;
                ^


D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:17: Associated declaration: D:\TESTING\GHISLENI_STEFANO\GHISLENI_JML\src\Allarme.java:58:
        //@ public invariant (\exists int i;0 <= i && i < 4;sensori[i]);
                ^
```

Esercizio 3 - KEY

Dopo aver scritto i contratti necessari al corretto funzionamento del programma nell'esercizio precedente li verifichiamo in maniera formale, ossia dimostriamo la correttezza parziale dal punto di vista logico/matematico dei contratti in maniera automatica con il tool MonKey (non dimostriamo che i metodi terminano).

Aggiungo //@ diverges true; ai contratti per non costringerci a provare che i metodi terminano.

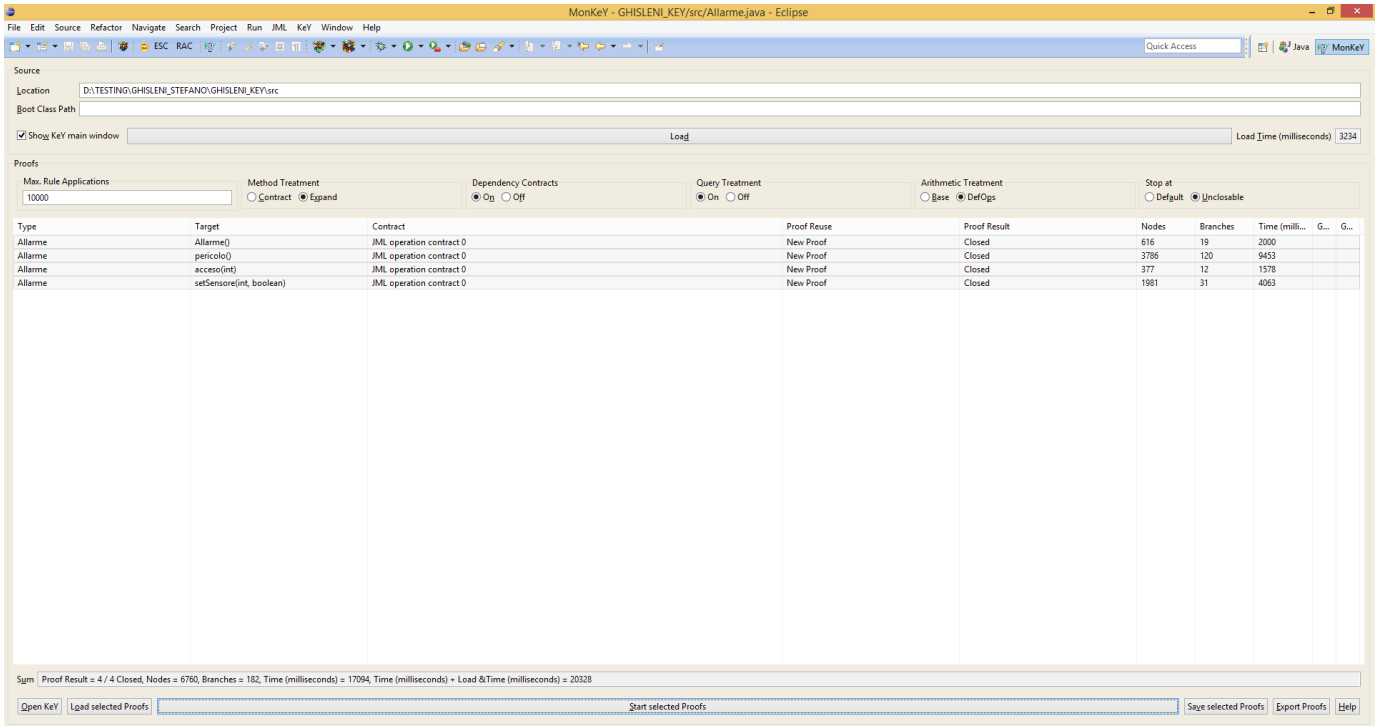
Per dimostrare i loop è stata usata la configurazione di Key con Loop treatment → Expand.

N.B. Per chiudere public boolean pericolo() ho modificato

//@ensures \result ==> (\forall int i; i >= 0 && i < N; !sensori[i] && !sensori[(i+1)%N]);

in

//@ensures \result ==> (!sensori[0] && !sensori[1]) || (!sensori[1] && !sensori[2]) || (!sensori[2] && !sensori[3]) || (!sensori[3] && !sensori[0]); Probabilmente Key fa fatica a gestire %



Il risultato ottenuto è molto buono dato che ogni metodo viene chiuso (dimostrato) correttamente dai contratti che ho scritto.

Esercizio 4 - NUSMV

In questo esercizio utilizziamo il tool NuSMV per descrivere il modello che rappresenta il funzionamento di Sensori Allarme.

Ho definito alcune specifiche LTL/CTL per poter eseguire la verifica formale di alcune proprietà che il modello deve avere (Model Checking).

P1

```
 -- specification AG !pericolo  is true
```

P2

```
-- specification !(EF (sensore0.statoSensore = FALSE & sensore2.statoSensore = FALSE))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  sensore0.statoSensore = TRUE
  sensore0.accendoSpengo = FALSE
  sensore1.statoSensore = TRUE
  sensore1.accendoSpengo = FALSE
  sensore2.statoSensore = TRUE
  sensore2.accendoSpengo = FALSE
  sensore3.statoSensore = TRUE
  sensore3.accendoSpengo = FALSE
  sceltaSensore = 0
  pericolo = FALSE
-> State: 1.2 <-
  sensore0.statoSensore = FALSE
  sceltaSensore = 2
-> State: 1.3 <-
  sensore2.statoSensore = FALSE
  sceltaSensore = 0
```

P3

```
-- specification AG (statoSensore = FALSE -> AF statoSensore) IN sensore3 is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 4.1 <-
  sensore0.statoSensore = TRUE
  sensore0.accendoSpengo = FALSE
  sensore1.statoSensore = TRUE
  sensore1.accendoSpengo = FALSE
  sensore2.statoSensore = TRUE
  sensore2.accendoSpengo = FALSE
  sensore3.statoSensore = TRUE
  sensore3.accendoSpengo = FALSE
  sceltaSensore = 0
  pericolo = FALSE
-> State: 4.2 <-
  sensore0.statoSensore = FALSE
  sensore0.accendoSpengo = TRUE
-> State: 4.3 <-
  sensore0.statoSensore = TRUE
  sensore0.accendoSpengo = FALSE
  sceltaSensore = 3
-- Loop starts here
-> State: 4.4 <-
  sensore3.statoSensore = FALSE
  sceltaSensore = 0
-> State: 4.5 <-
```

P4

```
-- specification AG (statoSensore = TRUE -> AF statoSensore = FALSE) IN sensore3 is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 4.1 <-
  sensore0.statoSensore = TRUE
  sensore0.accendoSpengo = FALSE
  sensore1.statoSensore = TRUE
  sensore1.accendoSpengo = FALSE
  sensore2.statoSensore = TRUE
  sensore2.accendoSpengo = FALSE
  sensore3.statoSensore = TRUE
  sensore3.accendoSpengo = FALSE
  sceltaSensore = 0
  pericolo = FALSE
-> State: 4.2 <-
  sensore0.statoSensore = FALSE
  sensore0.accendoSpengo = TRUE
-> State: 4.3 <-
  sensore0.statoSensore = TRUE
  sensore0.accendoSpengo = FALSE
```

P5

```
-- specification AG ((sensore0.statoSensore = TRUE & sensore1.statoSensore = FALSE) -> A [ sensore0.statoSensore = TRUE U sensore1.statoSensore = TRUE ] )  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  sensore0.statoSensore = TRUE
  sensore0.accendoSpengo = FALSE
  sensore1.statoSensore = TRUE
  sensore1.accendoSpengo = FALSE
  sensore2.statoSensore = TRUE
  sensore2.accendoSpengo = FALSE
  sensore3.statoSensore = TRUE
  sensore3.accendoSpengo = FALSE
  sceltaSensore = 0
  pericolo = FALSE
-> State: 1.2 <-
  sensore0.statoSensore = FALSE
  sensore0.accendoSpengo = TRUE
-> State: 1.3 <-
  sensore0.statoSensore = TRUE
  sensore0.accendoSpengo = FALSE
  sceltaSensore = 1
-- Loop starts here
-> State: 1.4 <-
  sensore1.statoSensore = FALSE
  sensore1.accendoSpengo = TRUE
  sceltaSensore = 0
-> State: 1.5 <-
```

Oltre alle specifiche richieste vengono inserite delle proprietà di safety e liveness.

```
-- Altre proprietà

-- Safety:
-- è sempre vero che sensore0 è acceso oppure sonsore1 è acceso
CTLSPEC AG (sensore0.statoSensore | sensore1.statoSensore); -- proprietà vera


-- Liveness
-- esiste uno stato futuro in cui sensore0 è spento
CTLSPEC EF sensore0.statoSensore=FALSE; -- proprietà vera
```

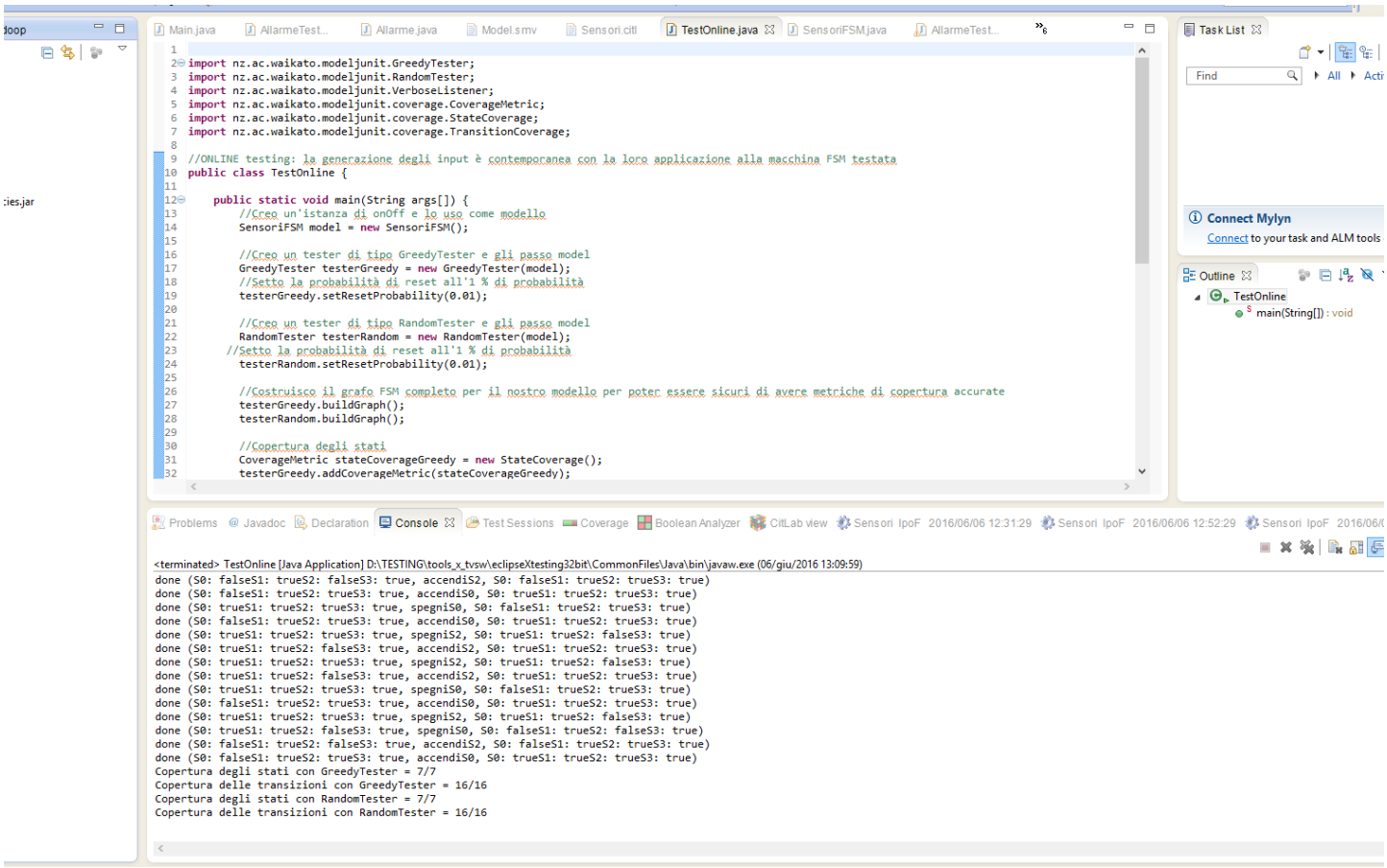
```
-- specification AG (sensore0.statoSensore | sensore1.statoSensore)  is true
-- specification EF sensore0.statoSensore = FALSE  is true
```

Esercizio 5 - FSM

Modello l'evoluzione del sistema con una FSM e la implemento con ModelJUnit.

ONLINE testing

Nell'Online testing la generazione degli input è contemporanea con la loro applicazione alla macchina FSM testata.



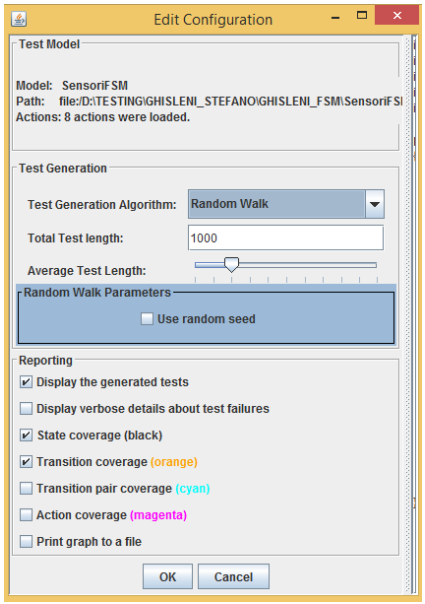
```
import nz.ac.waikato.modeljunit.GreedyTester;
import nz.ac.waikato.modeljunit.RandomTester;
import nz.ac.waikato.modeljunit.VerboseListener;
import nz.ac.waikato.modeljunit.coverage.CoverageMetric;
import nz.ac.waikato.modeljunit.coverage.StateCoverage;
import nz.ac.waikato.modeljunit.coverage.TransitionCoverage;

//ONLINE testing: la generazione degli input è contemporanea con la loro applicazione alla macchina FSM testata
public class TestOnline {

    public static void main(String args[]) {
        //Creo un'istanza di onOff e lo uso come modello
        SensoriFSM model = new SensoriFSM();

        //Creo un tester di tipo GreedyTester e gli passo model
        GreedyTester testerGreedy = new GreedyTester(model);
        //Setto la probabilità di reset all'1 % di probabilità
        testerGreedy.setResetProbability(0.01);

        //Creo un tester di tipo RandomTester e gli passo model
        RandomTester testerRandom = new RandomTester(model);
      //Setto la probabilità di reset all'1 % di probabilità
        testerRandom.setResetProbability(0.01);

        //Costruisco il grafo FSM completo per il nostro modello per poter essere sicuri di avere metriche di copertura accurate
        testerGreedy.buildGraph();
        testerRandom.buildGraph();

        //Copertura degli stati
        CoverageMetric stateCoverageGreedy = new StateCoverage();
        testerGreedy.addCoverageMetric(stateCoverageGreedy);
```

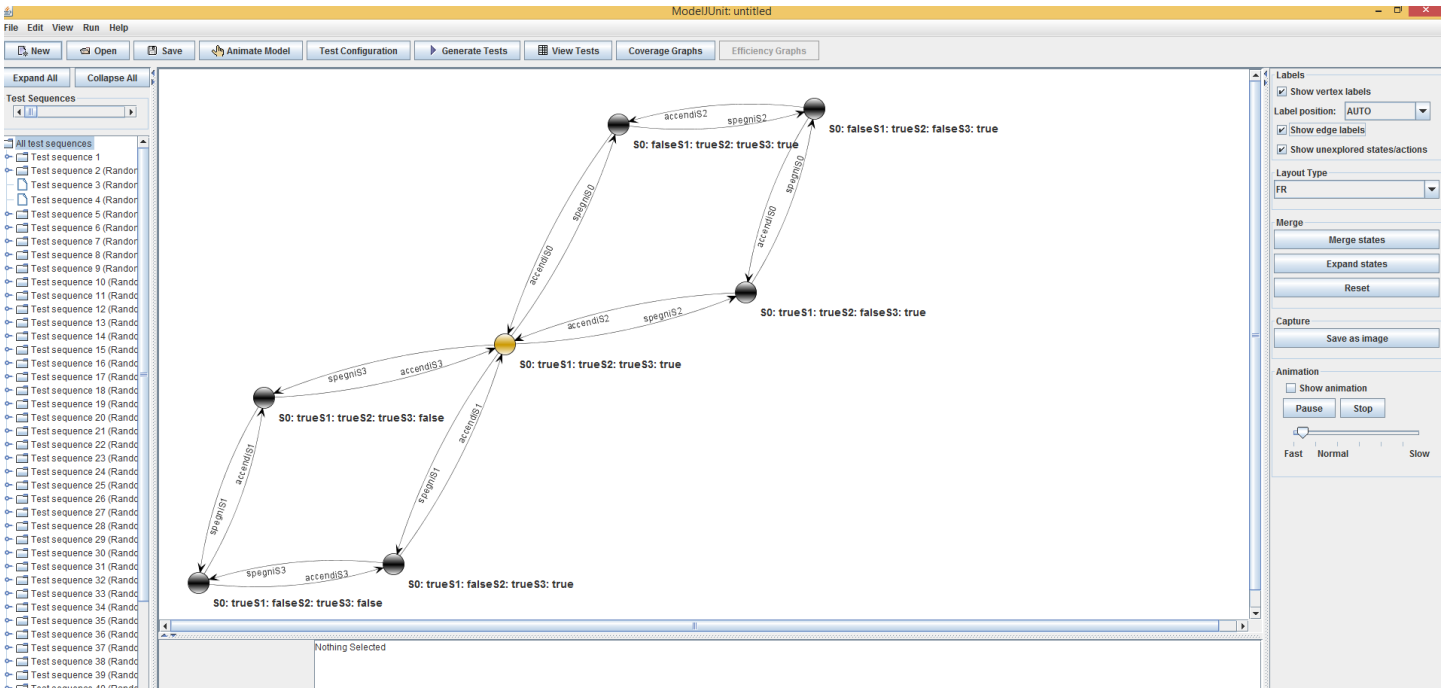
```
<terminated> TestOnline [Java Application] D:\TESTING\tools_x_tvsw\eclipseXtesting32bit\CommonFiles\Java\bin\javaw.exe (06/giu/2016 13:09:59)
done (S0: falseS1: trueS2: falseS3: true, accendiS2, S0: falseS1: trueS2: trueS3: true)
done (S0: falseS1: trueS2: trueS3: true, accendiS0, S0: trueS1: trueS2: trueS3: true)
done (S0: trueS1: trueS2: trueS3: true, spegniS0, S0: falseS1: trueS2: trueS3: true)
done (S0: falseS1: trueS2: trueS3: true, accendiS0, S0: trueS1: trueS2: trueS3: true)
done (S0: trueS1: trueS2: trueS3: true, spegniS2, S0: trueS1: trueS2: falseS3: true)
done (S0: trueS1: trueS2: falseS3: true, accendiS2, S0: trueS1: trueS2: trueS3: true)
done (S0: trueS1: trueS2: trueS3: true, spegniS2, S0: trueS1: trueS2: falseS3: true)
done (S0: trueS1: trueS2: falseS3: true, accendiS2, S0: trueS1: trueS2: trueS3: true)
done (S0: trueS1: trueS2: trueS3: true, spegniS0, S0: falseS1: trueS2: trueS3: true)
done (S0: falseS1: trueS2: trueS3: true, accendiS0, S0: trueS1: trueS2: trueS3: true)
done (S0: trueS1: trueS2: trueS3: true, spegniS2, S0: trueS1: trueS2: falseS3: true)
done (S0: trueS1: trueS2: falseS3: true, spegniS0, S0: falseS1: trueS2: falseS3: true)
done (S0: falseS1: trueS2: falseS3: true, accendiS2, S0: falseS1: trueS2: trueS3: true)
done (S0: falseS1: trueS2: trueS3: true, accendiS0, S0: trueS1: trueS2: trueS3: true)
Copertura degli stati con GreedyTester = 7/7
Copertura delle transizioni con GreedyTester = 16/16
Copertura degli stati con RandomTester = 7/7
Copertura delle transizioni con RandomTester = 16/16
```

Come possiamo notare otteniamo una copertura totale sia degli stati che delle transizioni, sia utilizzando un metodo di generazione di casi di test Greedy sia utilizzando metodo di generazione di casi di test Random.

OFFLINE testing (con l'interfaccia)

Usando il tool come interfaccia grafica possiamo esplorare il modello e generare le tracce offline.





Prendo queste transizioni e provo a tradurle in un caso di test Junit

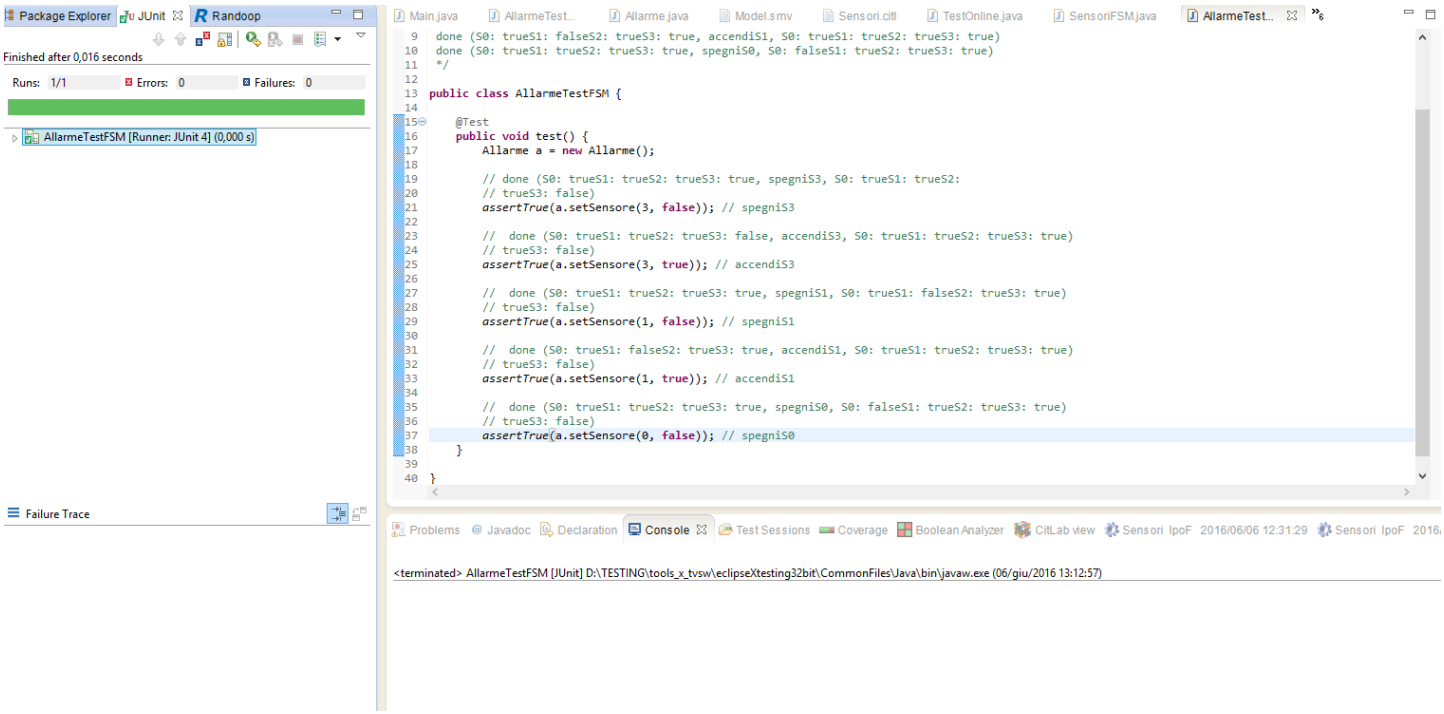
done (S0: trueS1: trueS2: trueS3: true, spegniS3, S0: trueS1: trueS2: trueS3: false)

done (S0: trueS1: trueS2: trueS3: false, accendiS3, S0: trueS1: trueS2: trueS3: true)

done (S0: trueS1: trueS2: trueS3: true, spegniS1, S0: trueS1: falseS2: trueS3: true)

done (S0: trueS1: falseS2: trueS3: true, accendiS1, S0: trueS1: trueS2: trueS3: true)

done (S0: trueS1: trueS2: trueS3: true, spegniS0, S0: falseS1: trueS2: trueS3: true)



Il test è superato, quindi abbiamo un po' di fiducia che il modello sia conforme alla classe Java scritta nell'esercizio Testing.

Esercizio 6 - COMBINATORIAL TESTING - CITLAB

Utilizzo il tool CitLab per effettuare il Combinatorial testing. Una volta definiti parametri e vincoli del modello, utilizzo l'estesione ACTS IpoF per generare i valori dei parametri. In particolar modo ci sono diversi criteri di scelta delle combinazioni di valori. Con il criterio Pair-Wise Coverage si ha che ogni valore di ogni parametro deve essere combinato con ogni valore di ogni altro parametro (combino i valori a coppie di parametri).

Copertura Pair-Wise:

AllarmeTestC... AllarmeTest... Allarme.java AllarmeTestD... Main.java AllarmeTest... Allarme.java Model.smv Sensori.citl

```
Model Sensori

//Parametri
Parameters:
    //Sensori
    Boolean s0;
    Boolean s1;
    Boolean s2;
    Boolean s3;
    //Situazione di pericolo
    Boolean pericolo;
end

//Vicoli
Constraints:
    //Se un sensore è spento allora i sensori adiacenti sono accesi
    # s0==false => (s3==true and s1==true) #
    # s1==false => (s0==true and s2==true) #
    # s2==false => (s1==true and s3==true) #
    # s3==false => (s2==true and s0==true) #
    //Vi è pericolo se e solo se vi sono due sensori adiacenti spenti
    # pericolo==true <=> ((s0==false and s1==false) or (s1==false and s2==false) or (s2==false and s3==false) or (s3==false and s0==false)) #
end
```

Problems Javadoc Declaration Console Test Sessions Coverage Boolean Analyzer CitLab view Sensori IpoF 2016/06/06 12:31:29

Test	s0	s1	s2	s3	pericolo
1	true	true	false	true	false
2	true	false	true	false	false
3	false	true	true	true	false
4	false	true	false	true	false
5	true	false	true	true	false
6	true	true	true	false	false

Name: IpoF

Time: 0.641

6

Export

Sensori IpoF 2016/06/06 12:31:29