

Soluzione al compito d'esame

Ringrazio gli studenti da cui ho preso larghe porzioni di codice.

Implementazione Java

Cerco di scriverla riusabile al massimo. Introduco le classi per track e per treno.

La configurazione data nell'esempio è un caso particolare di disposizione di track.

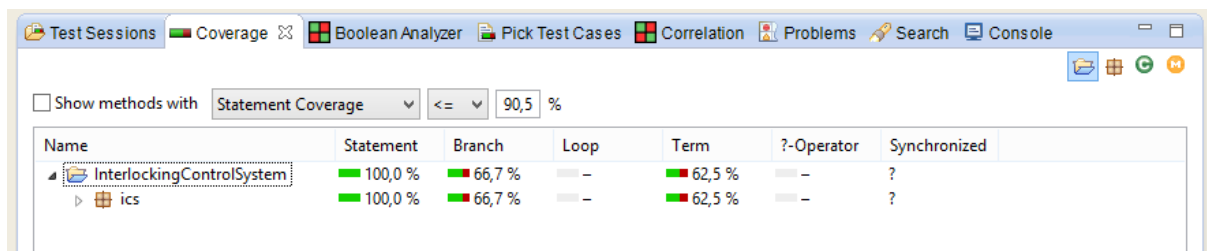
1. Testing

Con

Copertura_Istruzioni

Simulo lo spostamento doppio di uno stesso treno (uno effettivo e uno no perché è rosso il semaforo). Aggiungo molti assert per controllare che i semafori siano corretti.

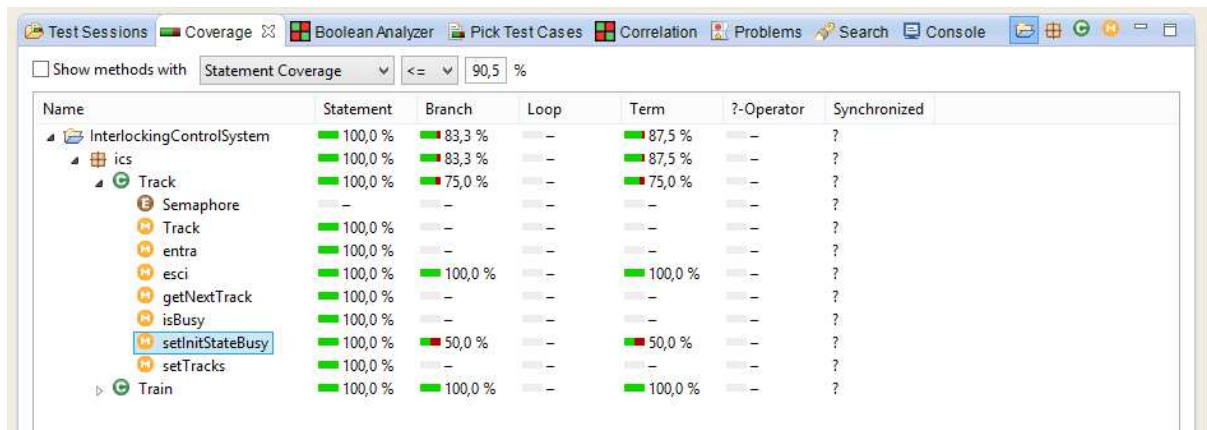
Copro tutte le istruzioni



| Name | Statement | Branch | Loop | Term | ?-Operator | Synchronized |
|---------------------------|-----------|--------|------|--------|------------|--------------|
| InterlockingControlSystem | 100,0 % | 66,7 % | — | 62,5 % | — | ? |
| ics | 100,0 % | 66,7 % | — | 62,5 % | — | ? |

Però sono un po' basso con le condizioni e i branch.

Aggiungo un caso di test più complesso:



| Name | Statement | Branch | Loop | Term | ?-Operator | Synchronized |
|---------------------------|-----------|---------|------|---------|------------|--------------|
| InterlockingControlSystem | 100,0 % | 83,3 % | — | 87,5 % | — | ? |
| ics | 100,0 % | 83,3 % | — | 87,5 % | — | ? |
| Track | 100,0 % | 75,0 % | — | 75,0 % | — | ? |
| Semaphore | — | — | — | — | — | ? |
| Track | 100,0 % | — | — | — | — | ? |
| entra | 100,0 % | — | — | — | — | ? |
| esci | 100,0 % | 100,0 % | — | 100,0 % | — | ? |
| getNextTrack | 100,0 % | — | — | — | — | ? |
| isBusy | 100,0 % | — | — | — | — | ? |
| setInitStateBusy | 100,0 % | 50,0 % | — | 50,0 % | — | ? |
| setTracks | 100,0 % | — | — | — | — | ? |
| Train | 100,0 % | 100,0 % | — | 100,0 % | — | ? |

C'è un caso branch che non riesco a coprire per via della configurazione iniziale (non ho due treni adiacenti).

Controllo anche MCDC. Ho solo un metodo con una condizione complessa:

```
(track.getNextTrack().s == Semaphore.GREEN || track.getNextTrack().s == Semaphore.YELLOW) {
```

| TestSessions Coverage Boolean Analyzer Pick Test Cases Correlation Problems Search Console | | | | |
|--|--|--|--------|---|
| Class: Train | Condition: (track.getNextTrack().s == Semaphore.GREEN OR track.getNextTrack().s == Semaphore.YELLOW) | | | |
| (track.getNextTrack().s == Semaphore.GREEN | OR | track.getNextTrack().s == Semaphore.YELLOW | Result | Test Cases (Number of Executions) |
| F | T | T | 1 | ics.anello.CoperturaDecisioni:test (1) Coverage |
| F | F | F | 0 | ics.anello.CoperturaDecisioni:test (1) Coverage |
| T | T | x | 1 | ics.anello.CoperturaDecisioni:test (1) Coverage |

La copertura soddisfa anche l'MCDC. Infatti essendo un OR devo:

| <code>track.getNextTrack().s == Semaphore.GREEN</code> | OR | <code>track.getNextTrack().s == Semaphore.YELLOW</code> |
|--|----|---|
| F | | T |
| F | | F |
| T | | F |
| F | | F (già fatta) |

2 JML

Contratti:

Ho messo degli invarianti per dire che i semafori sono correttamente settati e il treno va al prossimo track solo se libero.

Ho messo anche che due treni non possono stare sullo stesso track.

Ho dovuto aumentare la visibilità di alcuni.

Violazioni dei contratti:

So sostituisco la condizione di ingresso

```
if (track.getNextTrack().s != Semaphore.RED) {
```

con true, ottengo violazioni

```
Z:\AgDocuments\Dropbox\esami\testingeverifica\preappallo14\angelogargantini\InterlockingControlSystemJML\src\ics\Track.java:58: JML precondition is false
  @requires isBusy() && ! nextT.busy;
```

```
Z:\AgDocuments\Dropbox\esami\testingeverifica\preappallo14\angelogargantini\InterlockingControlSystemJML\src\ics\Track.java:58: JML precondition is false
  @requires isBusy() && ! nextT.busy;
```

Se mi scordo di mettere il semaforo a red:

```
//nextT.s = Semaphore.RED;
```

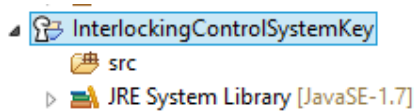
ho molti invarianti falsi tra cui:

```
Z:\AgDocuments\Dropbox\esami\testingeverifica\preappallo14\angelogargantini\InterlockingControlSystemJML\src\ics\Track.java:8: Associated declaration:
```

```
Z:\AgDocuments\Dropbox\esami\testingeverifica\preappallo14\angelogargantini\InterlockingControlSystemJML\src\ics\Track.java:35:
    //@public invariant busy <==> s==Semaphore.RED;
```

2b KEY

Ho creato un nuovo progetto di key:



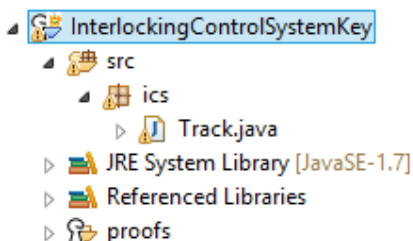
Ci ho copiato alcune classi e ho provato a lanciare key.

Ho anche semplificato un po' il codice e aggiunto dei contratti.

Ne ho dimostrato anche uno:

| Proofs | | | | | | | | | |
|------------------------|--------------------|--|-------------|---|-------|---|---------|---|---|
| Max. Rule Applications | | Method Treatment | | Dependency Contracts | | Query Treatment | | Arithmetic Treatment | |
| 10000 | | <input type="radio"/> Contract <input checked="" type="radio"/> Expand | | <input checked="" type="radio"/> On <input type="radio"/> Off | | <input checked="" type="radio"/> On <input type="radio"/> Off | | <input type="radio"/> Base <input checked="" type="radio"/> DefOps | |
| | | | | | | | | Stop at | |
| | | | | | | | | <input type="radio"/> Default <input checked="" type="radio"/> Unclosable | |
| Type | Target | Contract | Proof Reuse | Proof Result | Nodes | Bran... | Time... | G | G |
| ics.Track | Track() | JML normal_behavior operation contract 0 | New Proof | Open | 114 | 3 | 1292 | N | Y |
| ics.Track | goToNext() | JML normal_behavior operation contract 0 | New Proof | Open | 166 | 9 | 855 | Y | Y |
| ics.Track | setInitStateBusy() | JML operation contract 0 | New Proof | Closed | 144 | 7 | 1133 | | |

I proofs sono salvati:



3 NUSMV

Ho modellato i treni e le tracce. Ho usato i moduli (si può fare bene anche senza)

Provato **diverse** proprietà- commenti nel file .smv

La più importante:

```
-- due treni non si troveranno mai nella stessa posizione
CTLSPEC AG !(treno1=treno2);
```

Ho testato anche delle proprietà false. Ad esempio:

```
-- il treno1 non raggiungerà mai la posizione 4
LTLSPEC G (treno1!=4);
```

Dal controesempio:

```
Trace Type: Counterexample
-> State: 1.1 <-
```

```

treno1 = 1
treno2 = 3
track1.semaforo = ros
track2.semaforo = gial
track3.semaforo = ros
track4.semaforo = gial
-> State: 1.2 <-
  treno1 = 2
  treno2 = 4
  track1.semaforo = gial
  track2.semaforo = ros
  track3.semaforo = gial
  track4.semaforo = ros
-> State: 1.3 <-
  treno1 = 3
  treno2 = 1
  track1.semaforo = ros
  track2.semaforo = gial
  track3.semaforo = ros
  track4.semaforo = gial
-> State: 1.4 <-
  treno1 = 4
  treno2 = 2
  track1.semaforo = gial
  track2.semaforo = ros
  track3.semaforo = gial
  track4.semaforo = ros

```

Si capisce che il treno2 può raggiungere la 4 posizione ammesso che anche il treno2 vada avanti (come atteso).