Università degli studi di Roma 'Sapienza'

Laurea Specialistica

Ingegneria Informatica

Corso di Metodi Formali nell'Ingegneria del Software

JASMINE 2.0

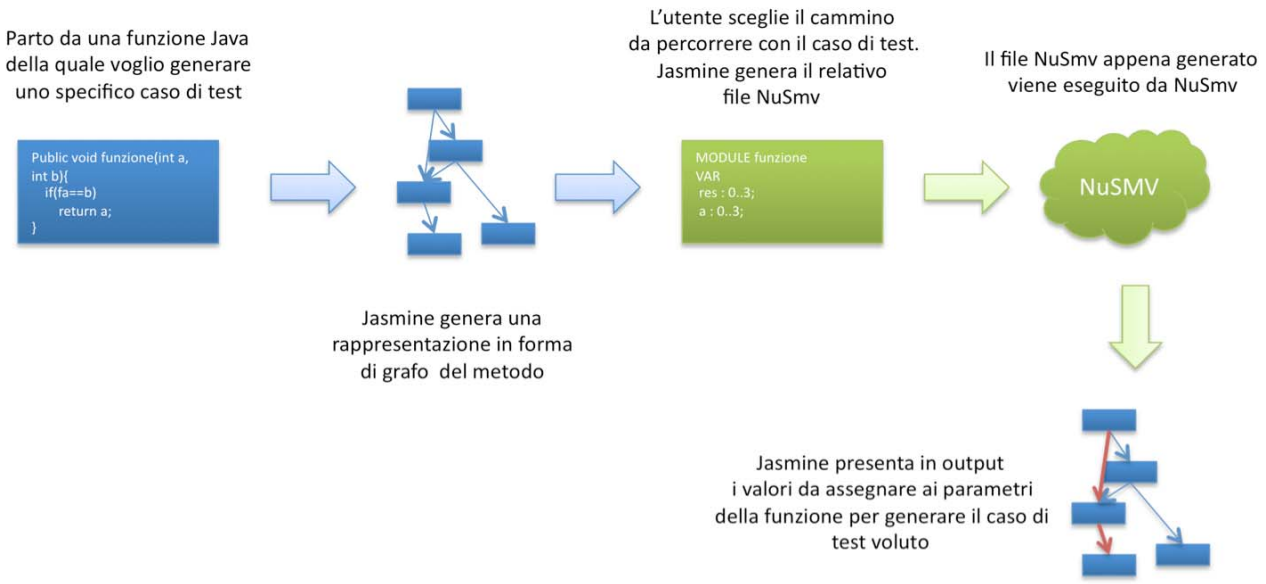
Andrea De Angelis

RELAZIONE JASMINE 2.0

TABLE OF CONTENTS

1. DOVE ERAVAMO: JASMINE 1.0

Il lavoro svolto in questa tesina è il naturale proseguimento del progetto Jasmine 1.0 degli studenti Luca Porrini, Constantin Moldovanu ed Emanuele Tatti. Jasmine 1.0 è un programma che effettua il parsing di una classe Java e ne crea un'opportuna rappresentazione con grafo di flusso dei metodi in essa definiti. Successivamente l'utente può scegliere un cammino da percorrere all'interno di uno dei metodi, e sarà compito dell'applicazione convertire il grafo in codice smv, che sarà successivamente passato al programma NuSmv, il quale a sua volta restituirà come output un caso di test che sia in grado di percorrere il cammino scelto dall'utente. Questo tool nasce quindi come strumento per automatizzare la generazione di casi di test a scatola bianca di funzioni scritte in Java.



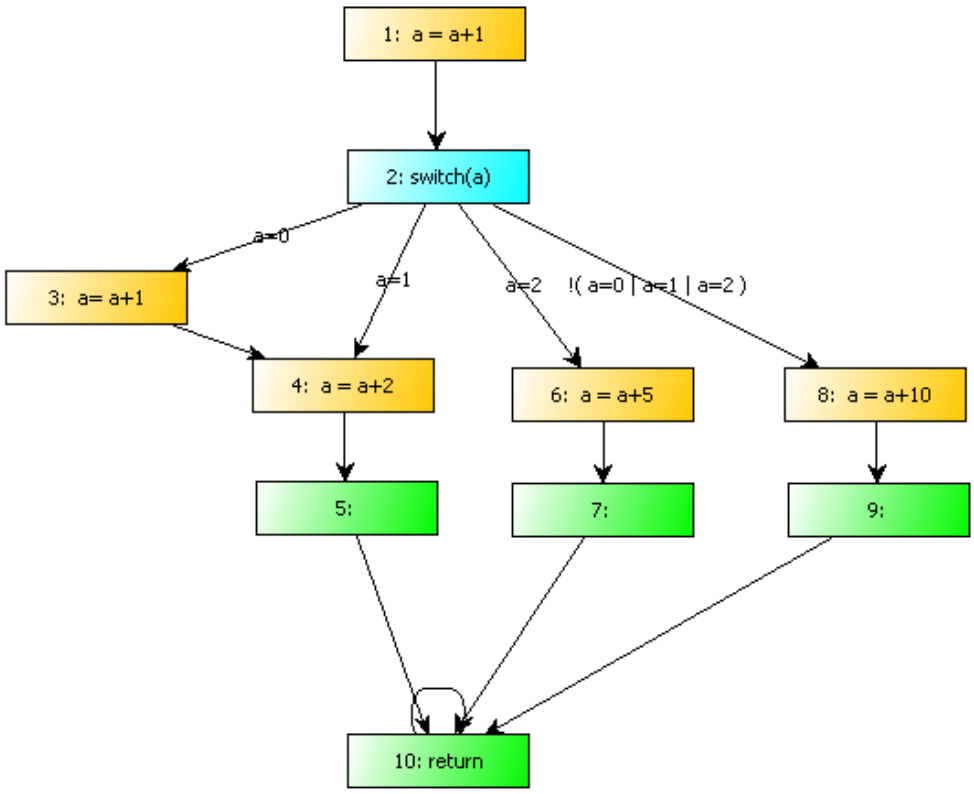
2. COSA VOGLIAMO REALIZZARE: JASMINE 2.0

Attualmente in Jasmine 1.0 è l'utente che deve indicare il cammino del quale generare il caso di test, quindi è l'utente stesso che deve cercare dei cammini che siano coerenti per un test a scatola bianca fatto con cognizione di causa. Vogliamo automatizzare questa operazione ed introdurre la generazione automatica di una base di cammini indipendenti che coprano l'intero grafo delle funzione. Per non snaturare il progetto vogliamo utilizzare come strumento di ricerca lo stesso NuSmv utilizzato per la generazione dei casi di test ed in particolare vogliamo mantenere il più possibile dell'attuale modello utilizzato da Jasmine 1.0.
Generare questa base di cammini in NuSmv però non è affatto ovvio: il problema principale è quello di riuscire a verificare l'indipendenza dei cammini trovati:

Prendiamo come esempio la semplice funzione switch(int a):int

```
public int Switch(int a) {
        a=a+1;
        switch (a) {
                case 0:a++;
                case 1:{
                        a=a+2;
                        break;
                }
                case 2: {
                        a = a+5;
                        break;
                }
                default: {
                        a= a+10;
                        break;
                }
        }
        return a;
}
```

Questa funzione, meramente di esempio, restituisce a secondo del valore in input un corrispondente intero in output. Vediamo il grafo che viene generato da Jasmine:



Ed il corrispondente codice NuSmv che viene utilizzato per la generazione di casi di test:

```
MODULE TestSwitch_Switch
--questo modulo descrive la funzione Java

VAR
 a : 0..3;
 PC : 1..10;

ASSIGN

DEFINE
 TERM := PC = 10;
 PRE := 1;
 POST := 1;

TRANS
 case
   PC = 1 : next(PC) = 2 & next(a)  = a+1 ;
   PC = 2 &  a=0 : next(PC) = 3 & next(a) = a ;
   PC = 2 &  a=1 : next(PC) = 4 & next(a) = a ;
   PC = 2 &  a=2 : next(PC) = 6 & next(a) = a ;
```

```
  PC = 2 & !( a=0 | a=1 | a=2 ) : next(PC) = 8 & next(a) = a ;
  PC = 3 : next(PC) = 4 & next(a) = a+1 ;
  PC = 4 : next(PC) = 5 & next(a)  = a+2 ;
  PC = 5 : next(PC) = 10 & next(a) = a ;
  PC = 6 : next(PC) = 7 & next(a)  = a+5 ;
  PC = 7 : next(PC) = 10 & next(a) = a ;
  PC = 8 : next(PC) = 9 & next(a)  = a+10 ;
  PC = 9 : next(PC) = 10 & next(a) = a ;
  PC = 10 : next(PC) = 10 & next(a) = a ;
  1: next(PC)=PC & next(a) = a ;
esac
--end MODULE TestSwitch_Switch

MODULE main
VAR
 t : TestSwitch_Switch;

ASSIGN
 init(t.PC):=1;
--end MODULE main

LTLSPEC
--TERMINAZIONE
--t.PRE -> F(t.TERM);
--CORRETTEZZA PARZIALE
--t.PRE -> G(t.TERM ->t.POST);
--CORRETTEZZA TOTALE
--t.PRE -> (F(t.TERM) & G(t.TERM ->t.POST));
--CAMMINO
```

(Non abbiamo considerato la condizione che viene scritta da jasmine per indurre NuSmv a generare il caso di test perchè dal nostro punto di vista è priva di interesse in questo momento.)

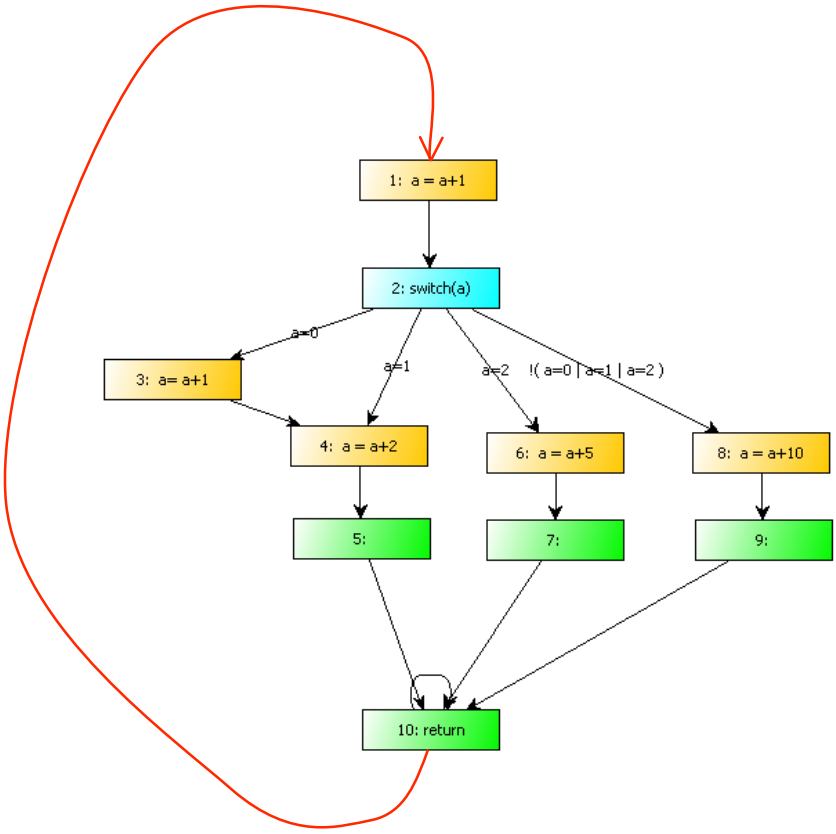
Il nostro intento è di modificare questo modello in modo che sia possibile scrivere una condizione che permetta a NuSmv di mostrarci una base di cammini di dimensione n fissata.
Il primo problema che risulta evidente per raggiungere questo risultato è la difficoltà che abbiamo per esprimere la condizione di indipendenza di un cammino sull'altro.
Necessitiamo di una soluzione che permetta di dichiarare che quello che stiamo cercando è un'insieme di cammini che rappresentano una copertura degli archi del grafo ed ogni cammino ha almeno un arco di differenza da ogni altro.
La mia soluzione è quella di aggiungere al grafo originale un arco di ritorno che congiunge l'ultimo nodo del grafo al primo in modo da permettere di percorrere ciclicamente il grafo. Quante volte lo percorriamo? Un numero di volte pari alla dimensione della base che stiamo cercando.
In poche parole introduciamo una variabile cammino che prende valori interi da 1 a n (con n dimensione della base). Idealmente il flusso del programma viene percorso la prima volta fino all'ultimo nodo del grafo ed il valore del cammino sarà uguale ad 1, ogni volta che si arriva in fondo si percorre l'arco di ritorno appena introdotto e si incrementa la variabile cammino di una unità. In questo modo il prossimo percorso all'interno del grafo sarà relativo ad un nuovo cammino, corrispondente appunto alla variabile 'cammino'. In questa maniera abbiamo l'opportunità di discriminare qual è il cammino che stiamo cercando correntemente e quali sono gli archi che formano il cammino precedente, quindi possiamo esprimere le condizioni di indipendenza.

A questo punto risulta infatti possibile affermare ad esempio che se il cammino 1 è formato dall'arco (1,2) e (2,3) (e basta) allora non esiste un cammino 2 che percorra almeno un arco diverso. In questo modo NuSmv troverà un contro esempio che sarà proprio il cammino 2 che ha un arco diverso.
Questo permette di ottenere una base di cammini indipendenti con una sola esecuzione di NuSMV.

Pro: **(+)** richiede un'unica esecuzione di NuSMV
(+) didatticamente interessante

Contro: **(-)** constringe NuSMV a cercare attraverso cammini molto lunghi. In particolare, se è possibile navigare il grafo attraversando m archi, con questo metodo NuSMV per ricercare una base di dimensione n è costretto a trovare una soluzione di lunghezza n*m
(-) è necessario indicare a priori la dimensione della base da cercare.

Vediamo sull'esempio come ottenere una base di dimensione n=4:



Aggiungiamo al grafo un arco di ritorno dall'ultimo nodo al primo nodo.
Introduciamo la variabile 'cammino' che avrà il compito di contare il numero di cammini eseguiti

Come scrivere a questo punto la condizione che ci permette di ottenere la base in NuSmv?
Deve avere la struttura seguente:

!(indipendenza primocammino & indipendenzasecondocammino & indipendenzaterzocammino & indipendenzaquartocammino)

vediamo nel dettaglio le singole condizioni di indipendenza per ogni cammino:

indipendenzaprimocammino: deve esprimere il fatto che deve esistere almeno un arco percorso nel primo cammino che non è percorso in uno degli altri cammini.

(camminoUnoVisitaArco1 & !camminoDueVisitaArco1 & !camminoTreVisitaArco1 & !camminoQuattroVisitaArco1)
|
(camminoUnoVisitaArco2 & !camminoDueVisitaArco2 & !camminoTreVisitaArco2 & !camminoQuattroVisitaArco2)
|
(camminoUnoVisitaArco3 & !camminoDueVisitaArco3 & !camminoTreVisitaArco3 & !camminoQuattroVisitaArco3)
...
...
...
(camminoUnoVisitaArco12 & !camminoDueVisitaArco12 & !camminoTreVisitaArco12 & !camminQuattroVisitaArco12)

Indipendenzasecondocammino:

(camminoDueVisitaArco1 & !camminoUnoVisitaArco1 & !camminoTreVisitaArco1 & !camminoQuattroVisitaArco1)
|
(camminoDueVisitaArco2 & !camminoUnoVisitaArco2 & !camminoTreVisitaArco2 & !camminoQuattroVisitaArco2)
|
(camminoDueVisitaArco3 & !camminoUnoVisitaArco3 & !camminoTreVisitaArco3 & !camminoQuattroVisitaArco3)
...
...
...
(camminoDueVisitaArco12 & !camminoUnoVisitaArco12 & !camminoTreVisitaArco12 & !camminQuattroVisitaArco12)

Indipendenzaterzocammino:

(camminoTreVisitaArco1 & !camminoDueVisitaArco1 & !camminoUnoVisitaArco1 & !camminoQuattroVisitaArco1)
|
(camminoTreVisitaArco2 & !camminoDueVisitaArco2 & !camminoUnoVisitaArco2 & !camminoQuattroVisitaArco1)
|
(camminoTreVisitaArco3 & !camminoDueVisitaArco3 & !camminoUnoVisitaArco3 & !camminoQuattroVisitaArco3)
...
...
...
(camminoTreVisitaArco12 & !camminoDueVisitaArco12 & !camminoUnoVisitaArco12 & !camminQuattroVisitaArco12)

Indipendenzaquartocammino:

(camminoQuattroVisitaArco1 & !camminoDueVisitaArco1 & !camminoTreVisitaArco1 & !camminoUnoVisitaArco1)
|
(camminoQuattroVisitaArco2 & !camminoDueVisitaArco2 & !camminoTreVisitaArco2 & !camminoUnoVisitaArco2)
|
(camminoQuattroVisitaArco3 & !camminoDueVisitaArco3 & !camminoTreVisitaArco3 & !camminoUnoVisitaArco3)
...
...
...
(camminoQuattroVisitaArco12 & !camminoDueVisitaArco12 & !camminoTreVisitaArco12 & !camminUnoVisitaArco12)

Esprimiamo queste condizioni in linguaggio proprio per NuSmv:

indipendenzaprimocammino in NuSmv:

F(m.cammino=1 & m.PC=1 & X(m.PC=2)) & !(F(m.cammino=2 & m.PC=1 & X(m.PC=2))) & !(F(m.cammino=3 & m.PC=1 & X(m.PC=2))) & !(F(m.cammino=4 & m.PC=1 & X(m.PC=2)))

|

F(m.cammino=1 & m.PC=2 & X(m.PC=3)) & !(F(m.cammino=2 & m.PC=2 & X(m.PC=3))) & !(F(m.cammino=3 & m.PC=2 & X(m.PC=3))) & !(F(m.cammino=4 & m.PC=2 & X(m.PC=3)))

...

...

...

F(m.cammino=1 & m.PC=9 & X(m.PC=10)) & !(F(m.cammino=1 & m.PC=9 & X(m.PC=10))) & !(F(m.cammino=3 & m.PC=9 & X(m.PC=10))) & !(F(m.cammino=4 & m.PC=9 & X(m.PC=10)))

indipendenzasecondocammino in NuSmv:

F(m.cammino=2 & m.PC=1 & X(m.PC=2)) & !(F(m.cammino=1 & m.PC=1 & X(m.PC=2))) & !(F(m.cammino=3 & m.PC=1 & X(m.PC=2))) & !(F(m.cammino=4 & m.PC=1 & X(m.PC=2)))

|

F(m.cammino=2 & m.PC=2 & X(m.PC=3)) & !(F(m.cammino=1 & m.PC=2 & X(m.PC=3))) & !(F(m.cammino=3 & m.PC=2 & X(m.PC=3))) & !(F(m.cammino=4 & m.PC=2 & X(m.PC=3)))

...

...

...

F(m.cammino=2 & m.PC=9 & X(m.PC=10)) & !(F(m.cammino=1 & m.PC=9 & X(m.PC=10))) & !(F(m.cammino=3 & m.PC=9 & X(m.PC=10))) & !(F(m.cammino=4 & m.PC=9 & X(m.PC=10)))

indipendenzaterzocammino in NuSmv:

F(m.cammino=3 & m.PC=1 & X(m.PC=2)) & !(F(m.cammino=2 & m.PC=1 & X(m.PC=2))) & !(F(m.cammino=1 & m.PC=1 & X(m.PC=2))) & !(F(m.cammino=4 & m.PC=1 & X(m.PC=2)))

|

F(m.cammino=3 & m.PC=2 & X(m.PC=3)) & !(F(m.cammino=2 & m.PC=2 & X(m.PC=3))) & !(F(m.cammino=1 & m.PC=2 & X(m.PC=3))) & !(F(m.cammino=4 & m.PC=2 & X(m.PC=3)))

...

...

...

F(m.cammino=3 & m.PC=9 & X(m.PC=10)) & !(F(m.cammino=2 & m.PC=9 & X(m.PC=10))) & !(F(m.cammino=1 & m.PC=9 & X(m.PC=10))) & !(F(m.cammino=4 & m.PC=9 & X(m.PC=10)))

indipendenzaquartocammino in NuSmv:

F(m.cammino=4 & m.PC=1 & X(m.PC=2)) & !(F(m.cammino=2 & m.PC=1 & X(m.PC=2))) & !(F(m.cammino=3 & m.PC=1 & X(m.PC=2))) & !(F(m.cammino=1 & m.PC=1 & X(m.PC=2)))

|

F(m.cammino=4 & m.PC=2 & X(m.PC=3)) & !(F(m.cammino=2 & m.PC=2 & X(m.PC=3))) & !(F(m.cammino=3 & m.PC=2 & X(m.PC=3))) & !(F(m.cammino=1 & m.PC=2 & X(m.PC=3)))

...

...

...

F(m.cammino=4 & m.PC=9 & X(m.PC=10)) & !(F(m.cammino=2 & m.PC=9 & X(m.PC=10))) & !(F(m.cammino=3 & m.PC=9 & X(m.PC=10))) & !(F(m.cammino=1 & m.PC=9 & X(m.PC=10)))

Vediamo come viene modificato il codice NuSmv per ottenere questo risultato:
In particolare esaminiamo il caso della ricerca di una base di dimensione n=4 per il grafo della funzione switch vista in precedenza:

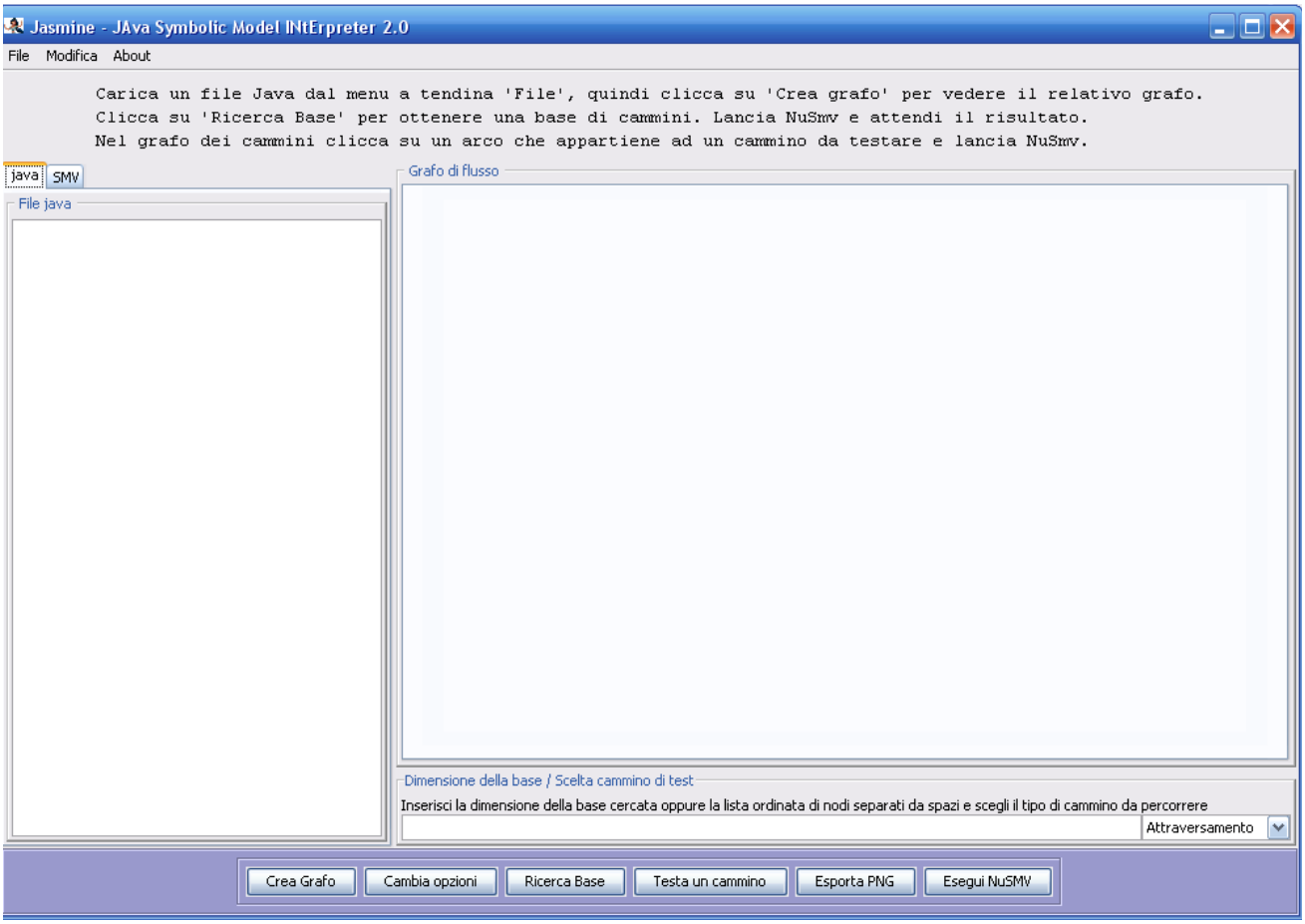
```
MODULE TestSwitch_Switch
--questo modulo descrive la funzione Java

VAR
 a : 0..3;
 cammino : 1..4;      introdotta per indicare il cammino nel quale ci troviamo correntemente

 PC : 1..10;

ASSIGN

DEFINE
TERM := PC = 10 & cammino=MAXCAMMINI;     adesso la condizione di terminazione non è semplicemente espressa
                                          dal fatto che ci troviamo sull'ultimo nodo ma dobbiamo anche aver
                                          calcolato tutti i cammini
 MAXCAMMINI:=4;          numero di cammini da trovare
 PRE := 1;
 POST := 1;

TRANS
 case
   PC = 1 : next(PC) = 2 & next(cammino)=cammino & next(a)  = a+1 ;          in ogni passaggio in un arco che non sia
   PC = 2 &  a=0 : next(PC) = 3 & next(cammino)=cammino & next(a) = a ;      quello di ritorno la variabile cammino
   PC = 2 &  a=1 : next(PC) = 4 & next(cammino)=cammino & next(a) = a ;      non può variare
   PC = 2 &  a=2 : next(PC) = 6 & next(cammino)=cammino & next(a) = a ;
   PC = 2 & !( a=0 | a=1 | a=2 ) : next(PC) = 8 & next(cammino)=cammino & next(a) = a ;
   PC = 3 : next(PC) = 4 & next(cammino)=cammino & next(a) = a+1 ;
   PC = 4 : next(PC) = 5 & next(cammino)=cammino & next(a)  = a+2 ;
   PC = 5 : next(PC) = 10 & next(cammino)=cammino & next(a) = a ;
   PC = 6 : next(PC) = 7 & next(cammino)=cammino & next(a)  = a+5 ;
   PC = 7 : next(PC) = 10 & next(cammino)=cammino & next(a) = a ;
   PC = 8 : next(PC) = 9 & next(cammino)=cammino & next(a)  = a+10 ;
   PC = 9 : next(PC) = 10 & next(cammino)=cammino & next(a) = a ;
```

```
  PC = 10 & cammino=4 : next(PC) = 10 & next(cammino)=cammino & next(a) = a ;     ho terminato
  PC = 10 & cammino!=4: next(PC)=1 & next(cammino)=cammino +1 ;          percorrendo l'arco di ritorno la variabile
1: next(PC)=PC & next(a) = a & next(cammino)=cammino;                    cammino viene incrementata di uno
esac
--end MODULE TestSwitch_Switch

MODULE main
VAR
 t : TestSwitch_Switch;

ASSIGN
 init(t.PC):=1;
--end MODULE main

LTLSPEC
--TERMINAZIONE
--t.PRE -> F(t.TERM);
--CORRETTEZZA PARZIALE
--t.PRE -> G(t.TERM ->t.POST);
--CORRETTEZZA TOTALE
--t.PRE -> (F(t.TERM) & G(t.TERM ->t.POST));
--CAMMINO
!(
(
(F(t.cammino=1 & t.PC=1 & X t.PC=2) & !F(t.cammino=2 & t.PC=1 & X t.PC=2) & !F(t.cammino=3 & t.PC=1 & X t.PC=2)
& !F(t.cammino=4 & t.PC=1 & X t.PC=2)) | (F(t.cammino=1 & t.PC=2 & X t.PC=3) & !F(t.cammino=2 & t.PC=2 & X
t.PC=3) & !F(t.cammino=3 & t.PC=2 & X t.PC=3) & !F(t.cammino=4 & t.PC=2 & X t.PC=3)) | (F(t.cammino=1 & t.PC=2 &
X t.PC=4) & !F(t.cammino=2 & t.PC=2 & X t.PC=4) & !F(t.cammino=3 & t.PC=2 & X t.PC=4) & !F(t.cammino=4 & t.PC=2
& X t.PC=4)) | (F(t.cammino=1 & t.PC=2 & X t.PC=6) & !F(t.cammino=2 & t.PC=2 & X t.PC=6) & !F(t.cammino=3 &
t.PC=2 & X t.PC=6) & !F(t.cammino=4 & t.PC=2 & X t.PC=6)) | (F(t.cammino=1 & t.PC=2 & X t.PC=8) & !F(t.cammino=2
& t.PC=2 & X t.PC=8) & !F(t.cammino=3 & t.PC=2 & X t.PC=8) & !F(t.cammino=4 & t.PC=2 & X t.PC=8)) |
(F(t.cammino=1 & t.PC=3 & X t.PC=4) & !F(t.cammino=2 & t.PC=3 & X t.PC=4) & !F(t.cammino=3 & t.PC=3 & X t.PC=4)
& !F(t.cammino=4 & t.PC=3 & X t.PC=4)) | (F(t.cammino=1 & t.PC=4 & X t.PC=5) & !F(t.cammino=2 & t.PC=4 & X
t.PC=5) & !F(t.cammino=3 & t.PC=4 & X t.PC=5) & !F(t.cammino=4 & t.PC=4 & X t.PC=5)) | (F(t.cammino=1 & t.PC=5 &
X t.PC=10) & !F(t.cammino=2 & t.PC=5 & X t.PC=10) & !F(t.cammino=3 & t.PC=5 & X t.PC=10) & !F(t.cammino=4 &
t.PC=5 & X t.PC=10)) | (F(t.cammino=1 & t.PC=6 & X t.PC=7) & !F(t.cammino=2 & t.PC=6 & X t.PC=7) & !F(t.cammino=3
& t.PC=6 & X t.PC=7) & !F(t.cammino=4 & t.PC=6 & X t.PC=7)) | (F(t.cammino=1 & t.PC=7 & X t.PC=10) &
!F(t.cammino=2 & t.PC=7 & X t.PC=10) & !F(t.cammino=3 & t.PC=7 & X t.PC=10) & !F(t.cammino=4 & t.PC=7 & X
t.PC=10)) | (F(t.cammino=1 & t.PC=8 & X t.PC=9) & !F(t.cammino=2 & t.PC=8 & X t.PC=9) & !F(t.cammino=3 & t.PC=8 &
X t.PC=9) & !F(t.cammino=4 & t.PC=8 & X t.PC=9)) | (F(t.cammino=1 & t.PC=9 & X t.PC=10) & !F(t.cammino=2 & t.PC=9
& X t.PC=10) & !F(t.cammino=3 & t.PC=9 & X t.PC=10) & !F(t.cammino=4 & t.PC=9 & X t.PC=10)) | (F(t.cammino=1 &
t.PC=10 & X t.PC=10) & !F(t.cammino=2 & t.PC=10 & X t.PC=10) & !F(t.cammino=3 & t.PC=10 & X t.PC=10) &
!F(t.cammino=4 & t.PC=10 & X t.PC=10))) & ((F(t.cammino=2 & t.PC=1 & X t.PC=2) & !F(t.cammino=1 & t.PC=1 & X
t.PC=2) & !F(t.cammino=3 & t.PC=1 & X t.PC=2) & !F(t.cammino=4 & t.PC=1 & X t.PC=2)) | (F(t.cammino=2 & t.PC=2 &
X t.PC=3) & !F(t.cammino=1 & t.PC=2 & X t.PC=3) & !F(t.cammino=3 & t.PC=2 & X t.PC=3) & !F(t.cammino=4 & t.PC=2
& X t.PC=3)) | (F(t.cammino=2 & t.PC=2 & X t.PC=4) & !F(t.cammino=1 & t.PC=2 & X t.PC=4) & !F(t.cammino=3 &
t.PC=2 & X t.PC=4) & !F(t.cammino=4 & t.PC=2 & X t.PC=4)) | (F(t.cammino=2 & t.PC=2 & X t.PC=6) & !F(t.cammino=1
& t.PC=2 & X t.PC=6) & !F(t.cammino=3 & t.PC=2 & X t.PC=6) & !F(t.cammino=4 & t.PC=2 & X t.PC=6)) |
(F(t.cammino=2 & t.PC=2 & X t.PC=8) & !F(t.cammino=1 & t.PC=2 & X t.PC=8) & !F(t.cammino=3 & t.PC=2 & X t.PC=8)
& !F(t.cammino=4 & t.PC=2 & X t.PC=8)) | (F(t.cammino=2 & t.PC=3 & X t.PC=4) & !F(t.cammino=1 & t.PC=3 & X
t.PC=4) & !F(t.cammino=3 & t.PC=3 & X t.PC=4) & !F(t.cammino=4 & t.PC=3 & X t.PC=4)) | (F(t.cammino=2 & t.PC=4 &
```

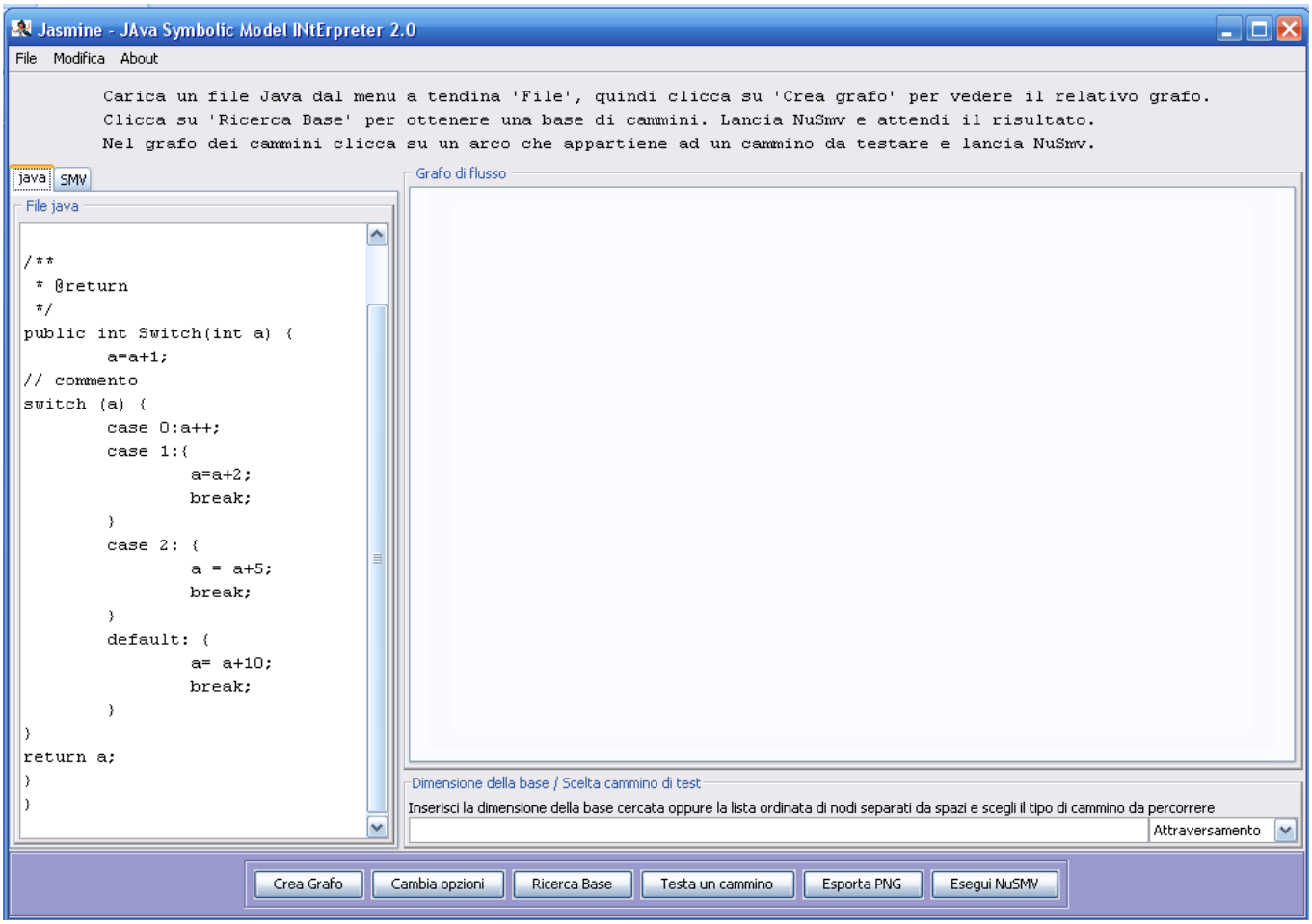
X t.PC=5) & !F(t.cammino=1 & t.PC=4 & X t.PC=5) & !F(t.cammino=3 & t.PC=4 & X t.PC=5) & !F(t.cammino=4 & t.PC=4 & X t.PC=5)) | (F(t.cammino=2 & t.PC=5 & X t.PC=10) & !F(t.cammino=1 & t.PC=5 & X t.PC=10) & !F(t.cammino=3 & t.PC=5 & X t.PC=10) & !F(t.cammino=4 & t.PC=5 & X t.PC=10)) | (F(t.cammino=2 & t.PC=6 & X t.PC=7) & !F(t.cammino=1 & t.PC=6 & X t.PC=7) & !F(t.cammino=3 & t.PC=6 & X t.PC=7) & !F(t.cammino=4 & t.PC=6 & X t.PC=7)) | (F(t.cammino=2 & t.PC=7 & X t.PC=10) & !F(t.cammino=1 & t.PC=7 & X t.PC=10) & !F(t.cammino=3 & t.PC=7 & X t.PC=10) & !F(t.cammino=4 & t.PC=7 & X t.PC=10)) | (F(t.cammino=2 & t.PC=8 & X t.PC=9) & !F(t.cammino=1 & t.PC=8 & X t.PC=9) & !F(t.cammino=3 & t.PC=8 & X t.PC=9) & !F(t.cammino=4 & t.PC=8 & X t.PC=9)) | (F(t.cammino=2 & t.PC=9 & X t.PC=10) & !F(t.cammino=1 & t.PC=9 & X t.PC=10) & !F(t.cammino=3 & t.PC=9 & X t.PC=10) & !F(t.cammino=4 & t.PC=9 & X t.PC=10)) | (F(t.cammino=2 & t.PC=10 & X t.PC=10) & !F(t.cammino=1 & t.PC=10 & X t.PC=10) & !F(t.cammino=3 & t.PC=10 & X t.PC=10) & !F(t.cammino=4 & t.PC=10 & X t.PC=10))) & ((F(t.cammino=3 & t.PC=1 & X t.PC=2) & !F(t.cammino=1 & t.PC=1 & X t.PC=2) & !F(t.cammino=2 & t.PC=1 & X t.PC=2) & !F(t.cammino=4 & t.PC=1 & X t.PC=2)) | (F(t.cammino=3 & t.PC=2 & X t.PC=3) & !F(t.cammino=1 & t.PC=2 & X t.PC=3) & !F(t.cammino=2 & t.PC=2 & X t.PC=3) & !F(t.cammino=4 & t.PC=2 & X t.PC=3)) | (F(t.cammino=3 & t.PC=2 & X t.PC=4) & !F(t.cammino=1 & t.PC=2 & X t.PC=4) & !F(t.cammino=2 & t.PC=2 & X t.PC=4) & !F(t.cammino=4 & t.PC=2 & X t.PC=4)) | (F(t.cammino=3 & t.PC=2 & X t.PC=6) & !F(t.cammino=1 & t.PC=2 & X t.PC=6) & !F(t.cammino=2 & t.PC=2 & X t.PC=6) & !F(t.cammino=4 & t.PC=2 & X t.PC=6)) | (F(t.cammino=3 & t.PC=2 & X t.PC=8) & !F(t.cammino=1 & t.PC=2 & X t.PC=8) & !F(t.cammino=2 & t.PC=2 & X t.PC=8) & !F(t.cammino=4 & t.PC=2 & X t.PC=8)) | (F(t.cammino=3 & t.PC=3 & X t.PC=4) & !F(t.cammino=1 & t.PC=3 & X t.PC=4) & !F(t.cammino=2 & t.PC=3 & X t.PC=4) & !F(t.cammino=4 & t.PC=3 & X t.PC=4)) | (F(t.cammino=3 & t.PC=4 & X t.PC=5) & !F(t.cammino=1 & t.PC=4 & X t.PC=5) & !F(t.cammino=2 & t.PC=4 & X t.PC=5) & !F(t.cammino=4 & t.PC=4 & X t.PC=5)) | (F(t.cammino=3 & t.PC=5 & X t.PC=10) & !F(t.cammino=1 & t.PC=5 & X t.PC=10) & !F(t.cammino=2 & t.PC=5 & X t.PC=10) & !F(t.cammino=4 & t.PC=5 & X t.PC=10)) | (F(t.cammino=3 & t.PC=6 & X t.PC=7) & !F(t.cammino=1 & t.PC=6 & X t.PC=7) & !F(t.cammino=2 & t.PC=6 & X t.PC=7) & !F(t.cammino=4 & t.PC=6 & X t.PC=7)) | (F(t.cammino=3 & t.PC=7 & X t.PC=10) & !F(t.cammino=1 & t.PC=7 & X t.PC=10) & !F(t.cammino=2 & t.PC=7 & X t.PC=10) & !F(t.cammino=4 & t.PC=7 & X t.PC=10)) | (F(t.cammino=3 & t.PC=8 & X t.PC=9) & !F(t.cammino=1 & t.PC=8 & X t.PC=9) & !F(t.cammino=2 & t.PC=8 & X t.PC=9) & !F(t.cammino=4 & t.PC=8 & X t.PC=9)) | (F(t.cammino=3 & t.PC=9 & X t.PC=10) & !F(t.cammino=1 & t.PC=9 & X t.PC=10) & !F(t.cammino=2 & t.PC=9 & X t.PC=10) & !F(t.cammino=4 & t.PC=9 & X t.PC=10)) | (F(t.cammino=3 & t.PC=10 & X t.PC=10) & !F(t.cammino=1 & t.PC=10 & X t.PC=10) & !F(t.cammino=2 & t.PC=10 & X t.PC=10) & !F(t.cammino=4 & t.PC=10 & X t.PC=10))) & ((F(t.cammino=4 & t.PC=1 & X t.PC=2) & !F(t.cammino=1 & t.PC=1 & X t.PC=2) & !F(t.cammino=2 & t.PC=1 & X t.PC=2) & !F(t.cammino=3 & t.PC=1 & X t.PC=2)) | (F(t.cammino=4 & t.PC=2 & X t.PC=3) & !F(t.cammino=1 & t.PC=2 & X t.PC=3) & !F(t.cammino=2 & t.PC=2 & X t.PC=3) & !F(t.cammino=3 & t.PC=2 & X t.PC=3)) | (F(t.cammino=4 & t.PC=2 & X t.PC=4) & !F(t.cammino=1 & t.PC=2 & X t.PC=4) & !F(t.cammino=2 & t.PC=2 & X t.PC=4) & !F(t.cammino=3 & t.PC=2 & X t.PC=4)) | (F(t.cammino=4 & t.PC=2 & X t.PC=6) & !F(t.cammino=1 & t.PC=2 & X t.PC=6) & !F(t.cammino=2 & t.PC=2 & X t.PC=6) & !F(t.cammino=3 & t.PC=2 & X t.PC=6)) | (F(t.cammino=4 & t.PC=2 & X t.PC=8) & !F(t.cammino=1 & t.PC=2 & X t.PC=8) & !F(t.cammino=2 & t.PC=2 & X t.PC=8) & !F(t.cammino=3 & t.PC=2 & X t.PC=8)) | (F(t.cammino=4 & t.PC=3 & X t.PC=4) & !F(t.cammino=1 & t.PC=3 & X t.PC=4) & !F(t.cammino=2 & t.PC=3 & X t.PC=4) & !F(t.cammino=3 & t.PC=3 & X t.PC=4)) | (F(t.cammino=4 & t.PC=4 & X t.PC=5) & !F(t.cammino=1 & t.PC=4 & X t.PC=5) & !F(t.cammino=2 & t.PC=4 & X t.PC=5) & !F(t.cammino=3 & t.PC=4 & X t.PC=5)) | (F(t.cammino=4 & t.PC=5 & X t.PC=10) & !F(t.cammino=1 & t.PC=5 & X t.PC=10) & !F(t.cammino=2 & t.PC=5 & X t.PC=10) & !F(t.cammino=3 & t.PC=5 & X t.PC=10)) | (F(t.cammino=4 & t.PC=6 & X t.PC=7) & !F(t.cammino=1 & t.PC=6 & X t.PC=7) & !F(t.cammino=2 & t.PC=6 & X t.PC=7) & !F(t.cammino=3 & t.PC=6 & X t.PC=7)) | (F(t.cammino=4 & t.PC=7 & X t.PC=10) & !F(t.cammino=1 & t.PC=7 & X t.PC=10) & !F(t.cammino=2 & t.PC=7 & X t.PC=10) & !F(t.cammino=3 & t.PC=7 & X t.PC=10)) | (F(t.cammino=4 & t.PC=8 & X t.PC=9) & !F(t.cammino=1 & t.PC=8 & X t.PC=9) & !F(t.cammino=2 & t.PC=8 & X t.PC=9) & !F(t.cammino=3 & t.PC=8 & X t.PC=9)) | (F(t.cammino=4 & t.PC=9 & X t.PC=10) & !F(t.cammino=1 & t.PC=9 & X t.PC=10) & !F(t.cammino=2 & t.PC=9 & X t.PC=10) & !F(t.cammino=3 & t.PC=9 & X t.PC=10)) | (F(t.cammino=4 & t.PC=10 & X t.PC=10) & !F(t.cammino=1 & t.PC=10 & X t.PC=10) & !F(t.cammino=2 & t.PC=10 & X t.PC=10) & !F(t.cammino=3 & t.PC=10 & X t.PC=10))))

3. FUNZIONAMENTO DEL PROGRAMMA

La schermata principale di Jasmine 2.0



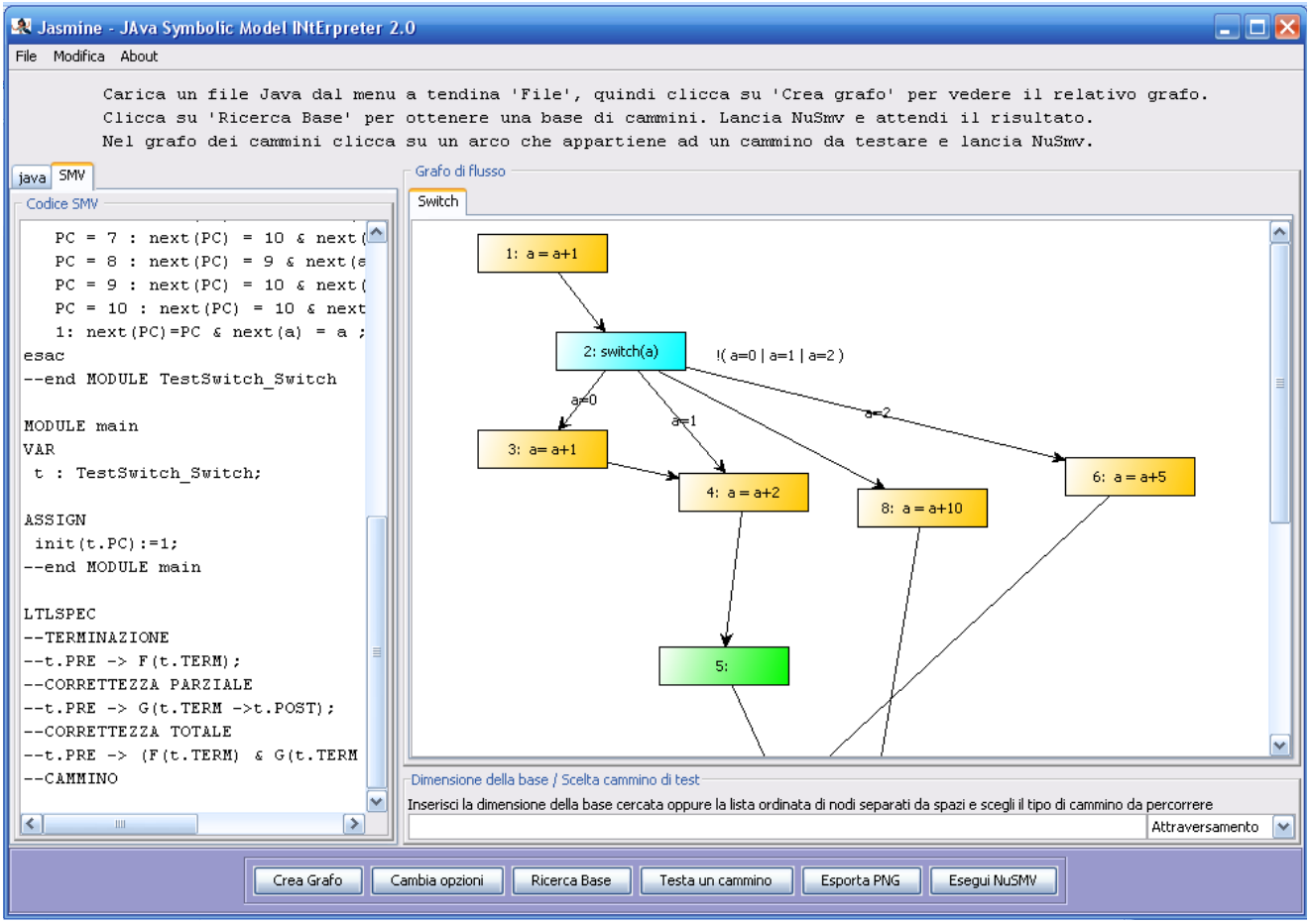
Clicchiamo su File -> Apri e scegliamo il file java che contiene la funzione da testare.
In questo caso apriamo il file TestSwitch.java



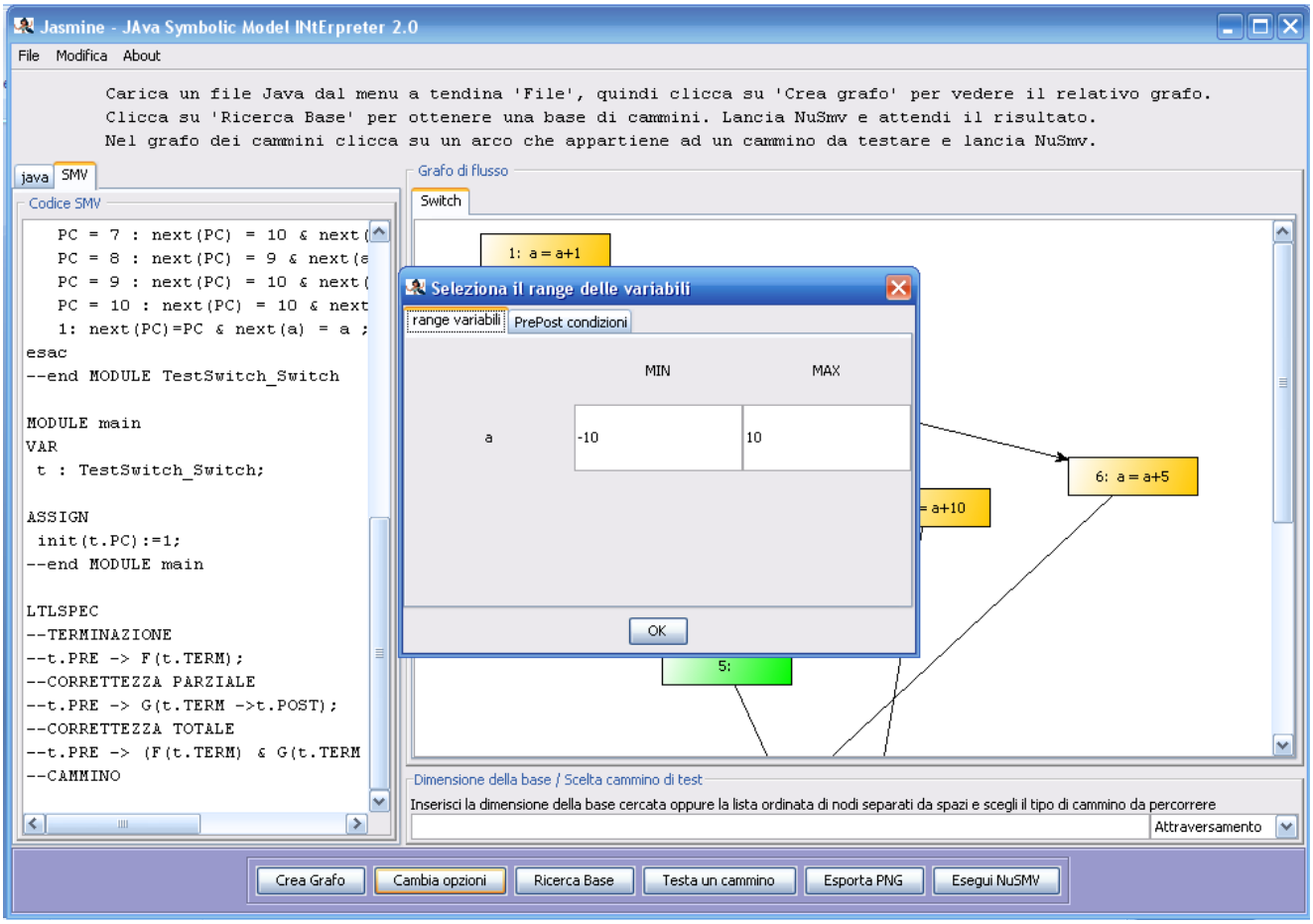
Come possiamo vedere nel pannello ‘java’ a sinistra compare il codice della funzione appena aperta.

Il passo successivo è quello di ottenere il grafo relativo alla funzione in questione.

Otteniamo questo risultato cliccando sul tasto ‘Crea Grafo’ in basso a sinistra.

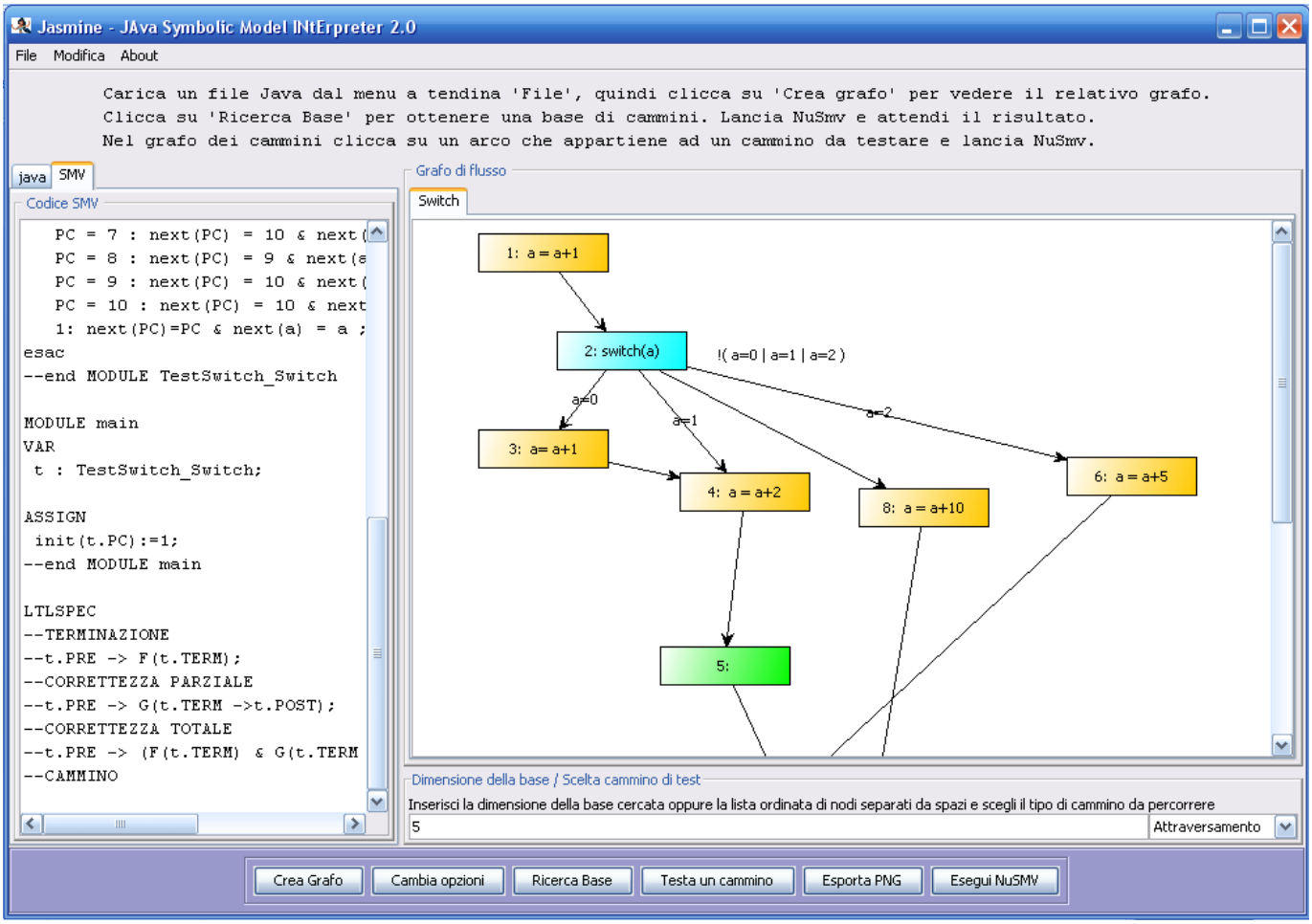


Jasmine genera il grafo del metodo e lo mostra nel pannello centrale mentre notiamo che la finestra a sinistra che prima conteneva il codice java è stata sostituita da un pannello con la codifica NuSmv del grafo appena creato. Possiamo sempre recuperare la vista della funzione java cliccando sulla linguetta Java nella parte alta del pannello. A questo punto scegliamo il range delle variabili rispetto al quale avverrà la ricerca in NuSmv, cliccando su 'cambia opzioni'

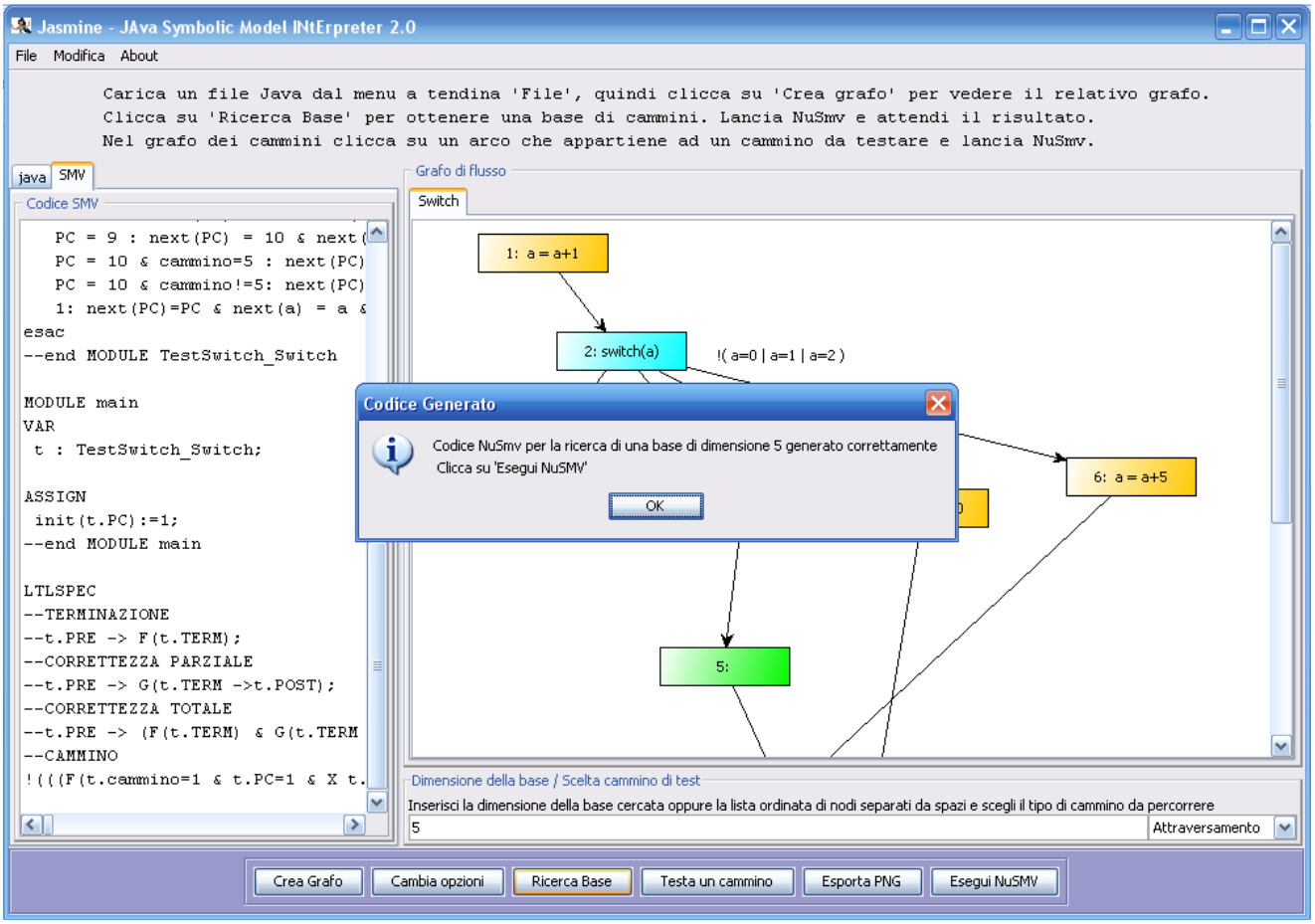


In questo modo compare una finestra che ci permette di scegliere il range di ogni variabile (in questo caso la sola variabile a). Questa scelta influenza la corretta esecuzione della ricerca di una base perchè un range di variabili sbagliato non permette di eseguire tutti i cammini e di conseguenza non sarà possibile trovare una base adeguata. In questo caso la variabile a è inserita all'interno di un case che discrimina i valori 0,1,2 e default quindi sarebbe sufficiente far variare la a tra 0 e 3 in modo da coprire tutti i rami dello switch. In realtà però all'interno del programma viene assegnato ad a il valore 10 quindi il valore più opportuno per a è -2..10

Un ulteriore passo opzionale è quello di cliccare sulla linguetta 'PrePost condizioni' e impostare appunto le pre e le post condizioni della funzione. In questo caso tiriamo dritto cliccando su OK (verranno impostate PRE=1 e POST=1).



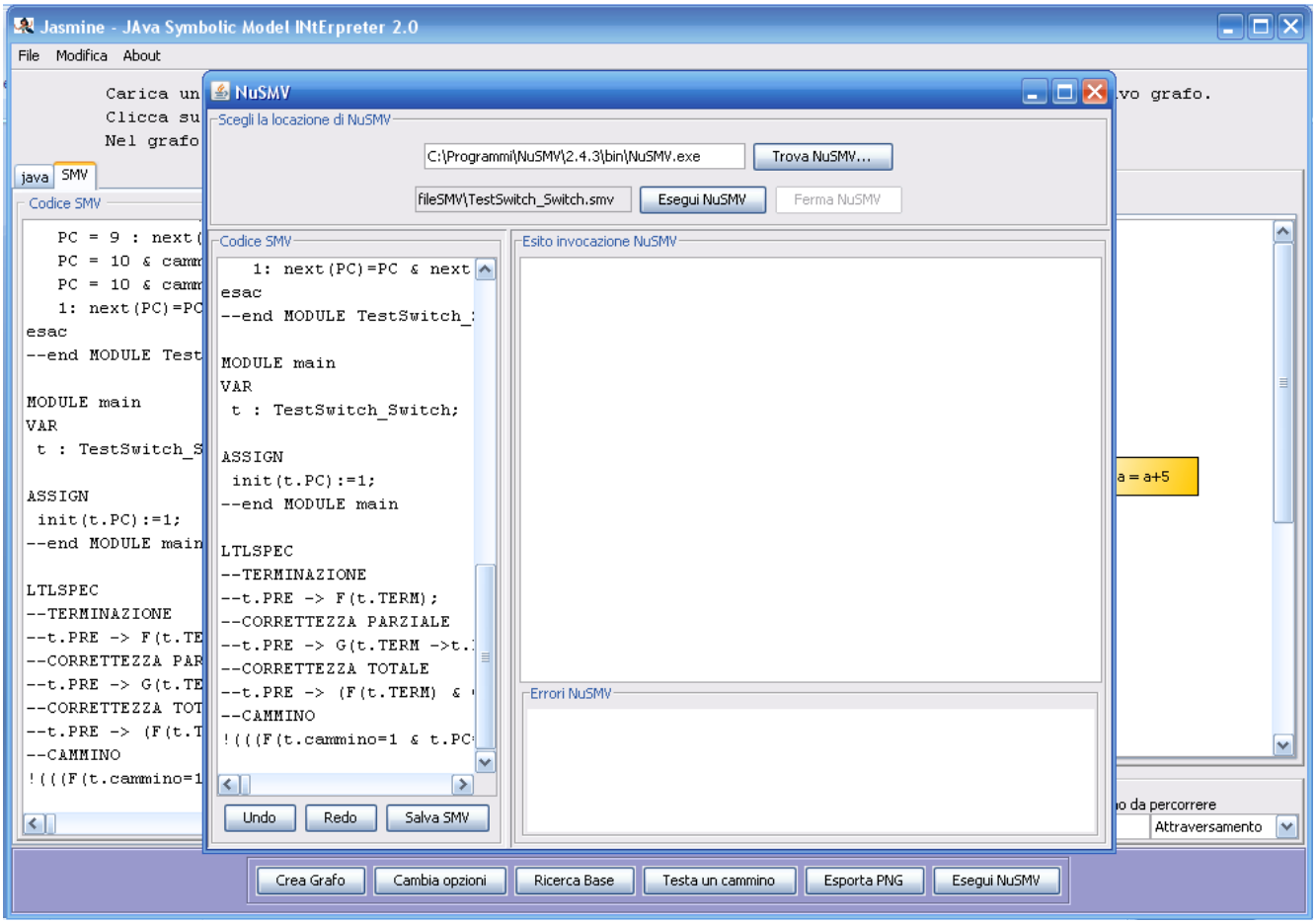
A questo punto inseriamo nell'area di testo appena sotto il grafo la dimensione della base voluta. Il grafo in questione ha una complessità ciclomatica pari a 4 ma noi chiediamo di cercare una base di dimensione 5 (sapendo che, essendo 5 maggiore della complessità ciclomatica, non verrà mai trovata) per dimostrare che il programma se non trova una base della dimensione voluta automaticamente ripete la ricerca abbassando di una unità la dimensione della base. Se la complessita ciclomatica è n, il cammino percorribile massimo può andare da n cammini a 0 perché non tutti o addirittura nessuno, potrebbero essere percorribili. E' per questo che Jasmine prende la dimensione della base inserita come limite superiore di ricerca. Una volta inserito il numero 5 clicchiamo su 'Ricerca Base'.



Il programma ci avverte che il codice NuSmv relativo alla ricerca della base di dimensione 5 è stato correttamente generato ed aggiunto quindi al codice del pannello NuSmv sulla sinistra.

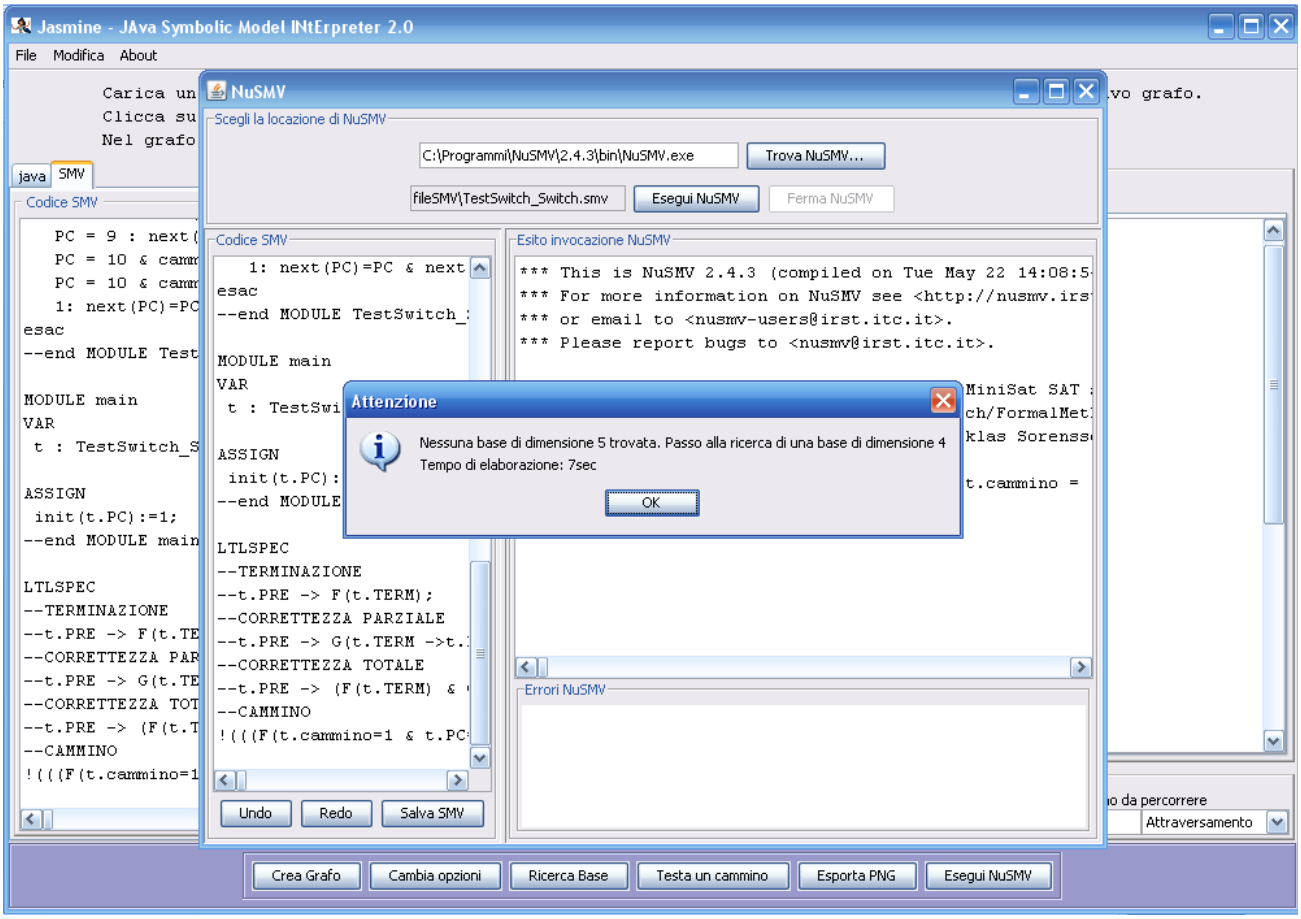
Adesso non ci resta che eseguirlo.

Clicchiamo quindi su ‘Esegui NuSmv’



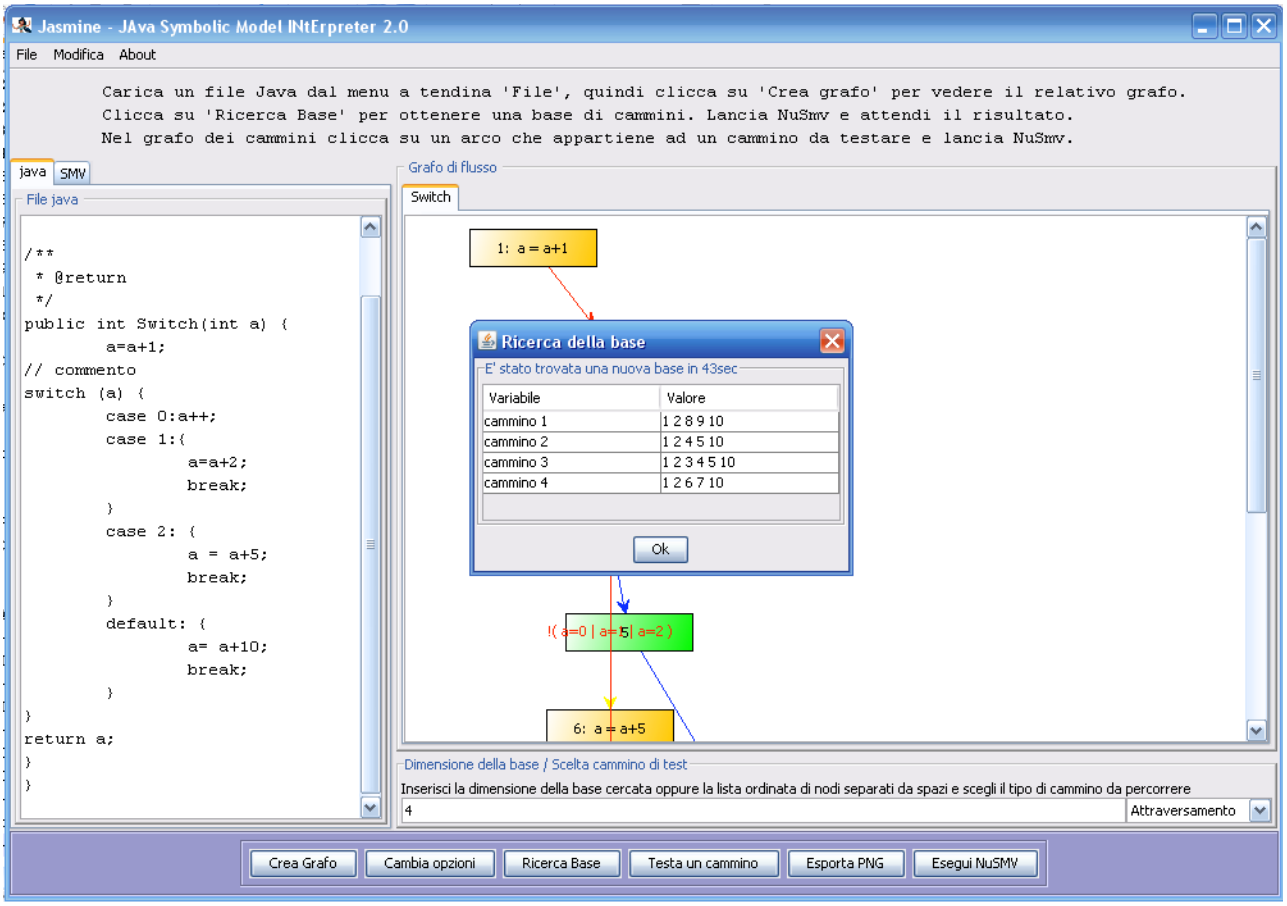
Compare allora la finestra che permette di eseguire il codice in NuSmv.

A sinistra c'è il codice NuSmv corrente, ogni modifica deve essere memorizzata cliccando su 'Salva SMV' che salva il codice su un file fisico in memoria secondaria. Nell'aria testuale in alto compare l'indirizzo corrente di NuSmv, se non è corretto clicchiamo su 'Trova NuSmv' e cerchiamo il percorso corretto. Una volta finito eseguiamo NuSmv cliccando su 'Esegui NuSmv'

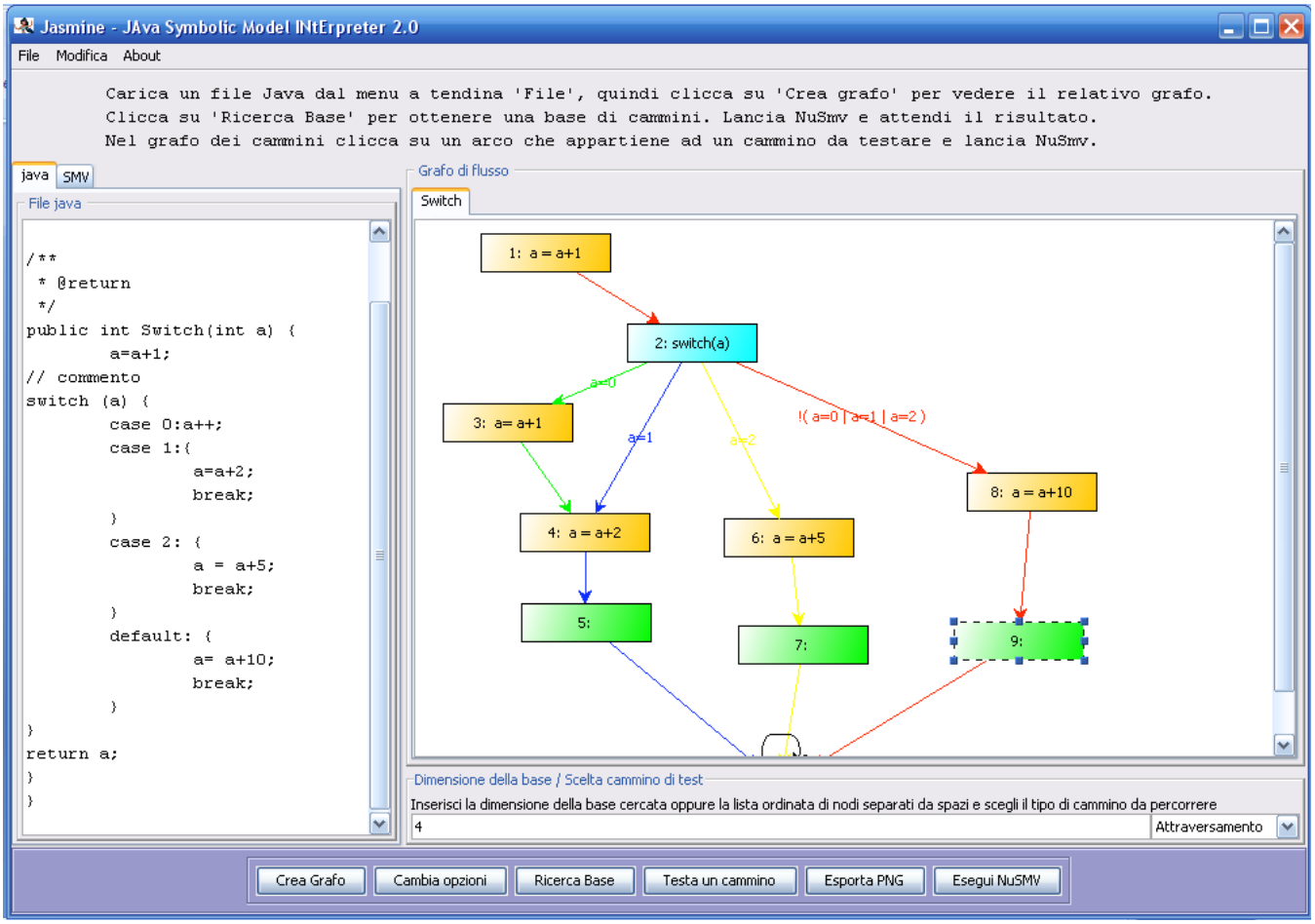


Ovviamente(stiamo cercando una base di 5 su un grafo con complessità ciclomatica pari a 4) NuSmv non riesce a trovare una base di dimensione 5 quindi Jasmine 2 ci avverte che la computazione riprenderà con la ricerca di una base di dimensione n=5-1=4.

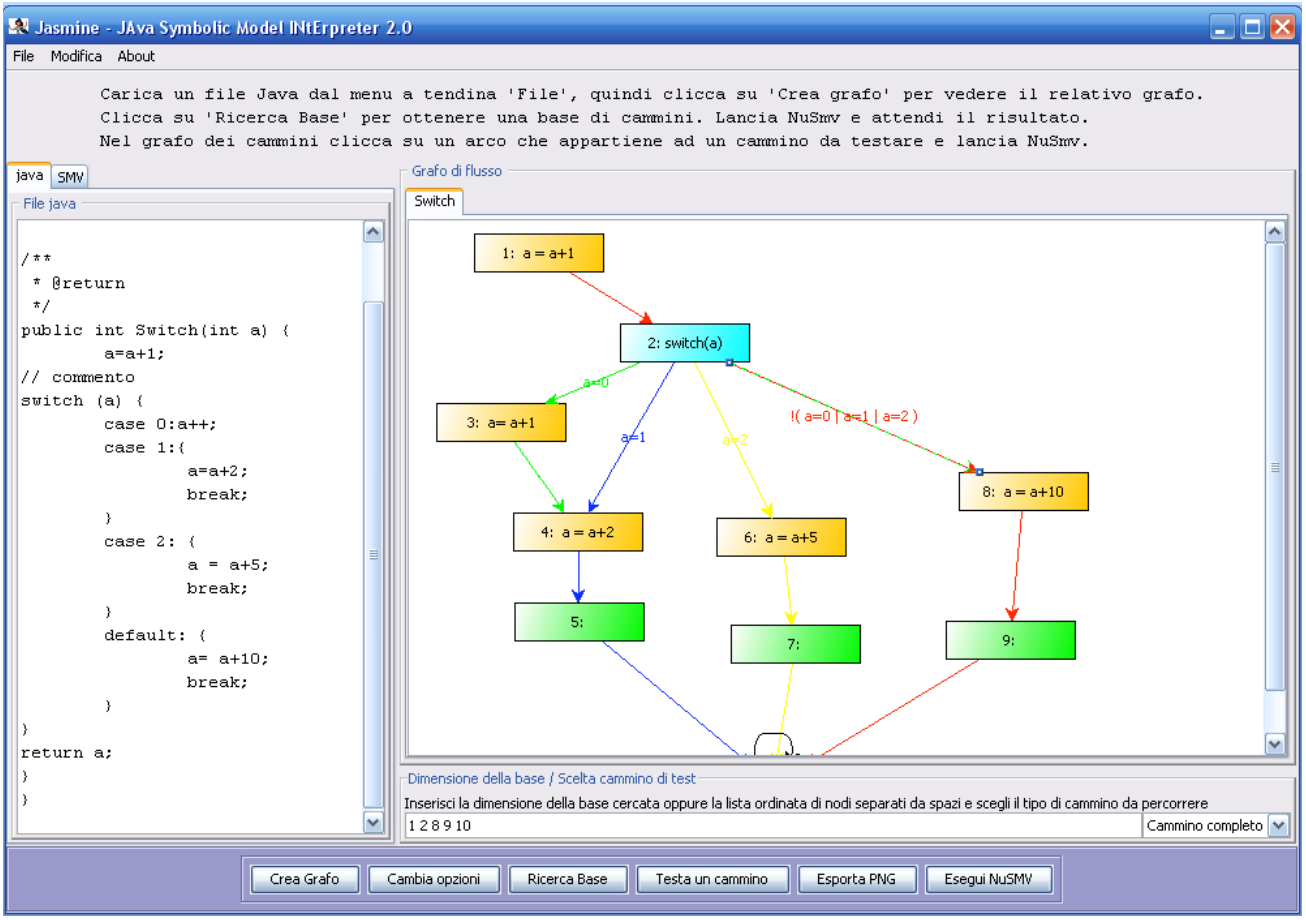
Clicchiamo su ok e aspettiamo i risultati di questa computazione.



Una volta che NuSmv ha restituito dei risultati, scompare la finestra relativa a NuSmv e vengono mostrati i risultati ottenuti. Clicchiamo su Ok per tornare al programma



Ci accorgiamo che gli archi del grafo sono colorati in base al cammino di appartenenza. Per non introdurre archi multipli però non sono colorati i cammini per intero ma solo la parte del cammino che differisce dal cammino precedente. Per esempio: il primo cammino è colorato di rosso ed essendo il primo è colorato nella sua interezza. Del secondo cammino, colorato in blu vengono mostrati solo gli archi che si differenziano dal cammino rosso. Quindi anche se l'arco (1,2) non è colorato anche di blu ma solo di rosso fa parte anche del cammino blu e così via.



Adesso abbiamo quindi raggiunto lo scopo di questa tesina: generare una base di cammini indipendenti per un grafo che rappresenta una funzione java qualsiasi. Se adesso clicchiamo su un arco, ad esempio l'arco (2,8) che appartiene al primo cammino (rosso), possiamo notare che nell'area di testo sottostante compare l'intero cammino rosso e la combobox a destra si autoimposta su 'cammino completo'.

Questo vuol dire che cliccando su un arco possiamo generare un caso di test per il cammino corrispondente semplicemente cliccando sul bottone 'Esegui NuSmv' utilizzando Jasmine come veniva utilizzato fino alla versione precedente

4. NULL PROBLEM

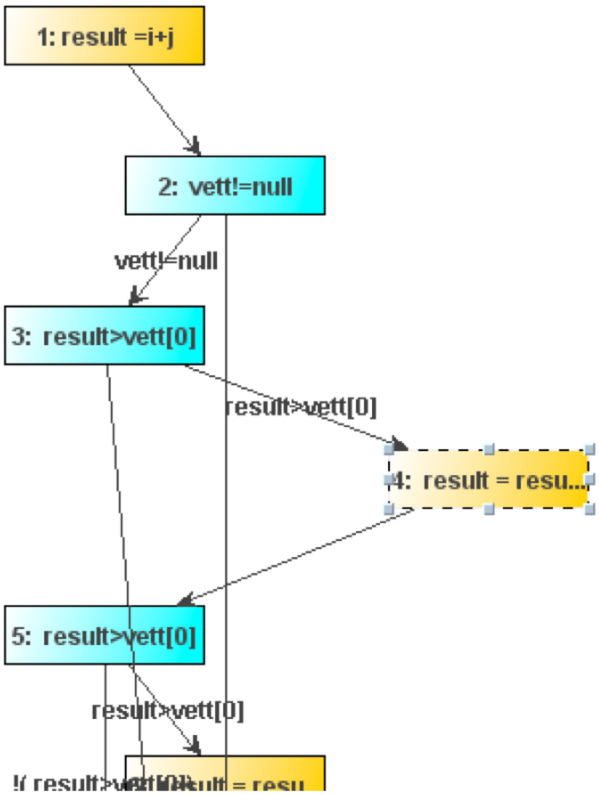
Prima di ottenere un prodotto usabile in pratica ho dovuto fronteggiare molti bug che affliggevano Jasmine 1.0. Molti di questi sono stati corretti e la versione attuale del programma è stata utilizzata con successo con molte funzioni non banali. Ovviamente il tempo dedicato al debugging della precedente versione si è limitato a raggiungere un livello accettabile ma, per motivi di tempo, non può garantire che il programma sia esente da errori ereditati da Jasmine 1.0. Mi sono impegnato a cercare di non introdurne di nuovi.

Uno dei problemi concettualmente rilevanti che ho ritenuto di riportare in questa relazione è quello che ho definito il null problem.
Andiamo a vedere nel dettaglio quando si verificava:

prendiamo la funzione java:

```
public int casoCritico3_2(int i, int j, int[] vett){
        int result=i+j;
        if(vett!=null){
                if(result>vett[0]){
                        result=result-vett[0];
                        if(result>vett[0]){
                                result=result-vett[0];
                        }else{
                                Int z=i-j;
                                If(result>z){
                                        Result=result-z;
                                }
                        }
                }else{
                        System.out.println("E' stato passato un vettore vuoto");
                }
                Return result;
        }
}
```

Abbiamo evidenziato in rosso la parte di istruzione che genera l'errore. Nessuna sorpresa, il Null Problem è generato da una istruzione che contiene il valore null.
Mostriamo di seguito il relativo grafo e codice NuSmv generato da Jasmine 1.0:



questo viene tradotto in NuSMV con:

TRANS
case
PC = 1 : next(PC) = 2 & next(result) =i+j &
next(z) = z & next(i) = i & next(j) = j &
next(vett[0]) = vett[0] & next(vett[1]) = vett[1]
& next(vett[2]) = vett[2] ;
PC = 2 & vett!=null : next(PC) = 3 & next(result)
= result & next(z) = z & next(i) = i &
next(j) = j & next(vett[0]) = vett[0] &
next(vett[1]) = vett[1] & next(vett[2]) = vett[2] ;
PC = 2 & !(vett!=null) : next(PC) = 14 &
next(result) = result & next(z) = z & next(i) = i &
next(j) = j & next(vett[0]) = vett[0] &
next(vett[1]) = vett[1] & next(vett[2]) = vett[2] ;
PC = 3 & result>vett[0] : next(PC) = 4 &
next(result) = result & next(z) = z & next(i) = i &
next(j) = j & next(vett[0]) = vett[0] &
next(vett[1]) = vett[1] & next(vett[2]) = vett[2]
…

La condizione presente nel codice Java che contiene il valore null è riportata da Jasmine 1.0 in NuSmv come una variabile 'null' che nel modello Smv non esiste. E' l'evidenziazione di quello che, di fatto, è una sorta di 'impedance mismatch' tra il codice java, che conosce la semantica del valore null e NuSmv che non ha nessun costrutto per trattarlo.

SOLUZIONE ADOTTATA

Ogni volta che nel codice compare un confronto tra una variabile e null, ad esempio:
var1!=null;
var2==null;

mi comporto nel seguente modo:
introduco nel codice NuSmv le variabili var1null e var2null
assegnerò a var1null il valore 1 se la var1 è a null, 0 altrimenti. Lo stesso per var2null

ogni volta che nella sezione TRANS compare la stringa var1!=null (var2!=null) la sostituisco con var1null!=1 (var2null!=1) che vista la semantica della variabile var1null (var2null) assume lo stesso significato.

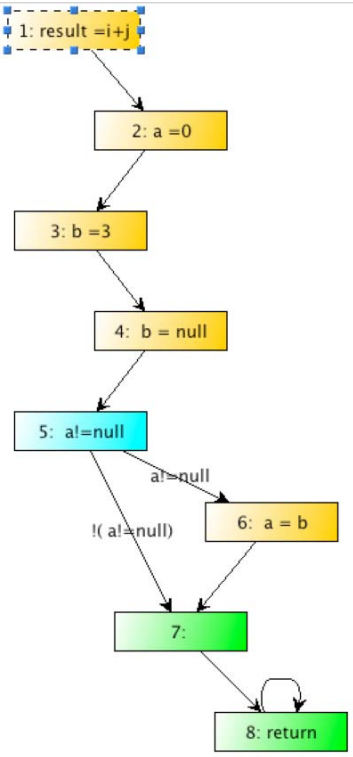
A questo punto trattiamo queste variabili introdotte come trattiamo le altre, dobbiamo imporre cioè che i valori assegnati loro possano cambiare solo percorrendo l'arco di ritorno a meno di alcuni casi particolari:

1. se una istruzione assegna un valore a var1 che non è una variabile, evidentemente quest'ultima non può avere un valore di var1null diverso da 0. Quindi la imposterò a tale valore
2. se una istruzione assegna un valore null a var1 allora dovrò assegnare a var1null il valore 1
3. se una istruzione assegna il valore di un'altra variabile a var1, del tipo: var1=a; allora dobbiamo verificare se a compaia tra le variabili che possiedono una versione anull. In questo caso dobbiamo assegnare var1null=anull. Nel caso contrario var1null deve essere impostato a 1.

Vediamo qui di seguito un esempio di come il programma tratta questo caso:

Prendiamo in considerazione un programma java che contiene un po' tutti i casi particolari evidenziati nel paragrafo precedente e il rispettivo grafo generato da Jasmine:

```
package filejava;


public class ProvaNull{

   public int prova(int i, int j){

      int result=i + j;
      int a=0;            (3)
      int b=3;            (4)
      b=null;             (5)
      if(a!=null){        (1)
         a=b;             (2)
      }
   }
}
```



Questo codice contiene tutti i tratti essenziali che vogliamo mettere in evidenza:

la variabile a nell'istruzione (1) è confrontata con null quindi dobbiamo creare la variabile anull 0..1

nell'istruzione (2) ad una variabile poi confrontata con null viene assegnata la variabile b, quindi dobbiamo creare anche bnull 0..1 perché dobbiamo assegnare next(anull)=bnull

l'istruzione 3 contiene una variabile per la quale esiste la corrispettiva variabile null alla quale viene assegnata un valore uguale a 0 quindi dobbiamo impostare next(anull)=0 in modo da riflettere questo cambiamento

nell'istruzione 4 la variabile b (che ha la corrispettiva variabile bnull) viene assegnata a 3 quindi dobbiamo specificare che next(bnull)=0

mentre nell'istruzione 5 viene assegnata al valore null quindi dobbiamo rispecchiare questo cambiamento con next(bnull)=1

Mostriamo quindi il codice NuSMV generato con i tratti discussi evidenziati in rosso:

```
VAR
 result : 0..3;
 a : 0..3;
 b : 0..3;
 bnull: 0..1;
 anull: 0..1;
 i : 0..3;
 j : 0..3;
 PC : 1..8;

ASSIGN
```

```
DEFINE
 TERM := PC = 8;
 PRE := 1;
 POST := 1;

TRANS
 case
   PC = 1 : next(PC) = 2 & next(bnull)=bnull & next(anull)=anull & next(result)  =i+j & next(a) = a & next(b) = b &
next(i) = i & next(j) = j ;
   PC = 2 : next(PC) = 3 & next(anull)=0 & next(bnull)=bnull & next(result) = result & next(a)  =0 & next(b) = b &
next(i) = i & next(j) = j ;
   PC = 3 : next(PC) = 4 & next(bnull)=0 & next(anull)=anull & next(result) = result & next(a) = a & next(b)  =3 &
next(i) = i & next(j) = j ;
   PC = 4 : next(PC) = 5 & next(bnull)=0 & next(anull)=anull & next(result) = result & next(a) = a & next(i) = i &
next(j) = j ;
   PC = 5 &  anull!=1 : next(PC) = 6 & next(bnull)=bnull & next(anull)=anull & next(result) = result & next(a) = a &
next(b) = b & next(i) = i & next(j) = j ;
   PC = 5 & !( anull!=1) : next(PC) = 7 & next(bnull)=bnull & next(anull)=anull & next(result) = result & next(a) = a &
next(b) = b & next(i) = i & next(j) = j ;
   PC = 6 : next(PC) = 7 & next(anull)=0 & next(bnull)=bnull & next(result) = result & next(a)  = b & next(b) = b &
next(i) = i & next(j) = j ;
   PC = 7 : next(PC) = 8 & next(result) = result & next(a) = a & next(b) = b & next(i) = i & next(j) = j ;
   PC = 8 : next(PC) = 8 & next(result) = result & next(a) = a & next(b) = b & next(i) = i & next(j) = j ;
   1: next(PC)=PC & next(i) = i & next(j) = j & next(result) = result & next(a) = a & next(b) = b ;
esac
```

5. RISULTATI

Riportiamo di seguito I risultati in termini di prestazioni che Jasmine 2.0 ottiene con alcune significative funzioni già testate anche nella tesi di Santelli (metti riferimento)

Precisazione sulla complessità ciclomatica: nella determinazione della complessità ciclomatica di un grafo di flusso si applica comunemente la formula V(G)=E-N+2 dove E è il numero degli archi e N è il numero dei nodi del grafo. Tuttavia questa formula si riferisce ad un grafo costruito in maniera tale che ogni nodo contenga una istruzione di decisione semplice. Significa che, ad esempio, se un programma contiene l'istruzione if(a>=5 || a<0 || b>=5 || b<0), un grafo che permette di calcolare correttamente la sua complessità ciclomatica con la formula suddetta deve dividere questa macro-istruzione di decisione in decisioni semplici del tipo if(a>=5) if(a<0) if(b>=5) e if(b<0).
Questo non è il caso dei grafi costruiti da Jasmine che, di conseguenza non possono essere utilizzati per calcolare la loro complessità ciclomatica.
Utilizzeremo quindi un' altra strada che è quella di contare all'interno del codice del metodo il numero di decisioni semplici (if con una singola condizione, for, while ecc.) ed eggiungere 1.

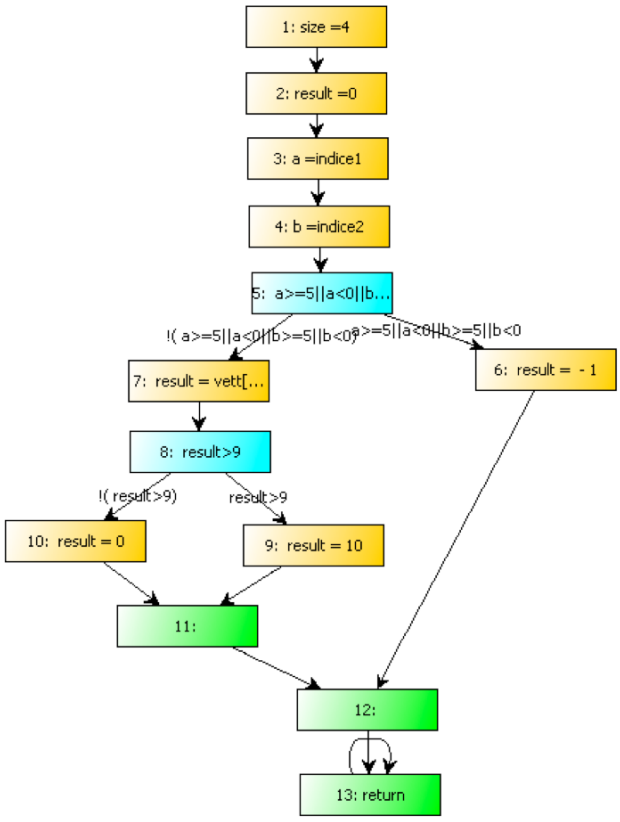
Dopo questa precisazione passiamo all'analisi dei risultati:

SOMMADUEELEMENTIVETTORE.JAVA

INPUT: un array di interi e due interi
OUTPUT: un valore intero
COMPLESSITA' CICLOMATICA: 6

Questa funzione prende in ingresso un array di interi e due indici (int), e somma il contenuto dell'array nelle due celle indicate dagli indici, restituendo 10 se questo valore supera 9 oppure 0 nel caso contrario. Se gli indici inseriti non sono validi restituisce -1.

```
public int sommaDueElementiVettore(int[] vett,int
indice1,int indice2) {
                int size=4;
                int result=0;
                int a=indice1;
                int b=indice2;
                if(a>=5 || a<0 || b>=5 || b<0)
                        result= -1;
                else{
                        result=vett[a]+vett[b];
                        if (result>9)
                                result= 10;
                        else
                                result= 0;
                }
                return result;
        }
```



IMPOSTAZIONI DEI RANGE

Variabile	Range
Size	4..4
Result	-1..15
A	-1..5
B	-1..5
Vett	0..5
Vett.length	4
Indice1	-1..5
Indice2	-1..5

RISULTATI

Ricerca base di 4:	non trovata in 2 min
Ricerca base di 3:	trovata in 3 min

MASSIMO.JAVA

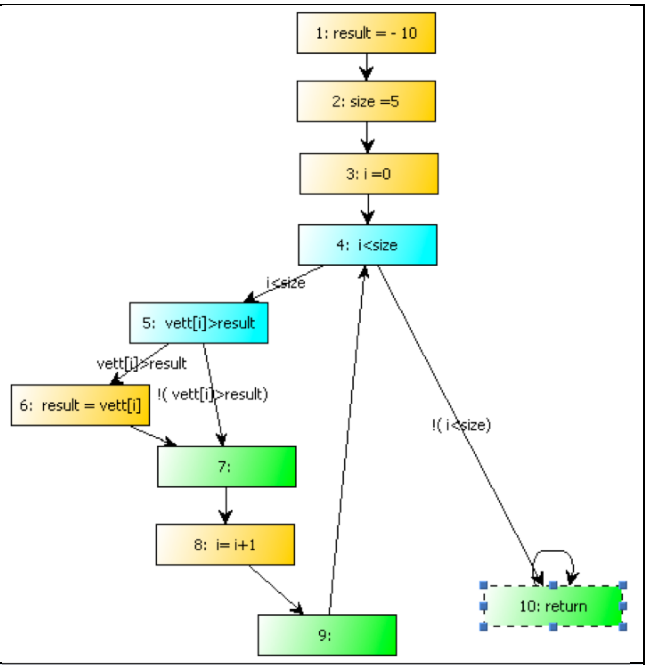
INPUT: un array di interi

OUTPUT: un intero che è il max valore contenuto nell'array in input

COMPLESSITA' CICLOMATICA: 3

Questa funzione prende in ingresso un array di interi e restituisce il valore massimo contenuto nell'array.

```
package filejava;

import java.io.*;

public class Massimo {

	public static int massimo(int[] vett){
		int result=-10;
		int size=5;
		int i=0;
		while(i<size){
			if(vett[i]>result)
				result=vett[i];
			i++;
		}
		return result;
	}
}
```



IMPOSTAZIONI DEI RANGE

Variabile	Range
Result	-10..10
Size	5..5
I	0..5
Vett	-10..10
Vett.length	5

RISULTATI

Ricerca base di 4:	non trovata in 5 min
Ricerca base di 3:	non trovata in 2 min
Ricerca base di 2:	base trovata in 1 min

BUBBLESORT.JAVA

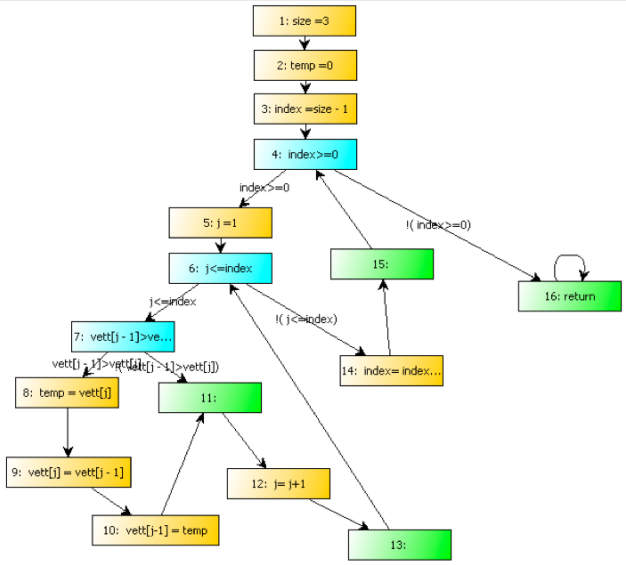
INPUT: un array di interi

OUTPUT: un nessuno

COMPLESSITA' CICLOMATICA: 4

Questa funzione prende non restituisce nessun valore ma ordina gli elementi del vettore in ingresso.

```
package filejava;

import java.io.*;

public class BubbleSort {

public static void bubbleSort(int[] vett) {
        int size=3;
        int temp=0;
        int index=size-1;
        while (index>=0){
                int j=1;
                while (j<=index){
                        if (vett[j-1]>vett[j]){
                                temp=vett[j];
                                vett[j]=vett[j-1];
                                vett[j-1]=temp;
                        }
                        j++;
                }
                index--;
        }
}
}
```



IMPOSTAZIONI DEI RANGE	
Variabile	**range**
Size	3..3
Temp	0..10
Index	-1..3
J	1..3
Vett	0..10
Vett.length	3

RISULTATI	
Ricerca base di 4:	**non trovata in 8 min**
Ricerca base di 3:	**base trovata in 4 min**

SELECTIONSORT.JAVA

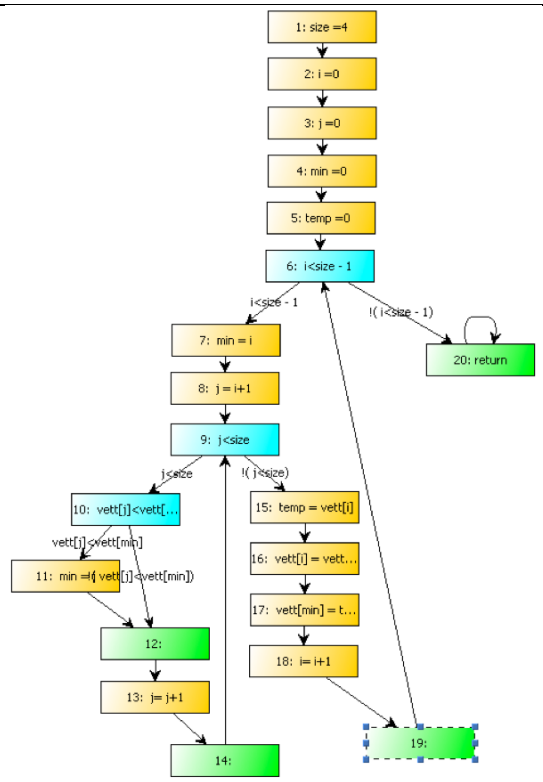
INPUT: un array di interi

OUTPUT: nessuno

COMPLESSITA' CICLOMATICA: 4

Questa funzione prende non restituisce nessun valore ma ordina gli elementi del vettore in ingresso.

```
import java.io.*;

public class SelectionSort {
	public static void selectionSort(int[] vett) {
		int size=4;
		int i=0;
		int j=0;
		int min=0;
		int temp=0;
		while (i<size-1){
			min=i;
			j=i+1;
			while (j<size){
				if (vett[j]<vett[min])
					min=j;
				j++;
			}
			temp=vett[i];
			vett[i]=vett[min];
			vett[min]=temp;
			i++;
		}
	}
}
```



IMPOSTAZIONI DEI RANGE

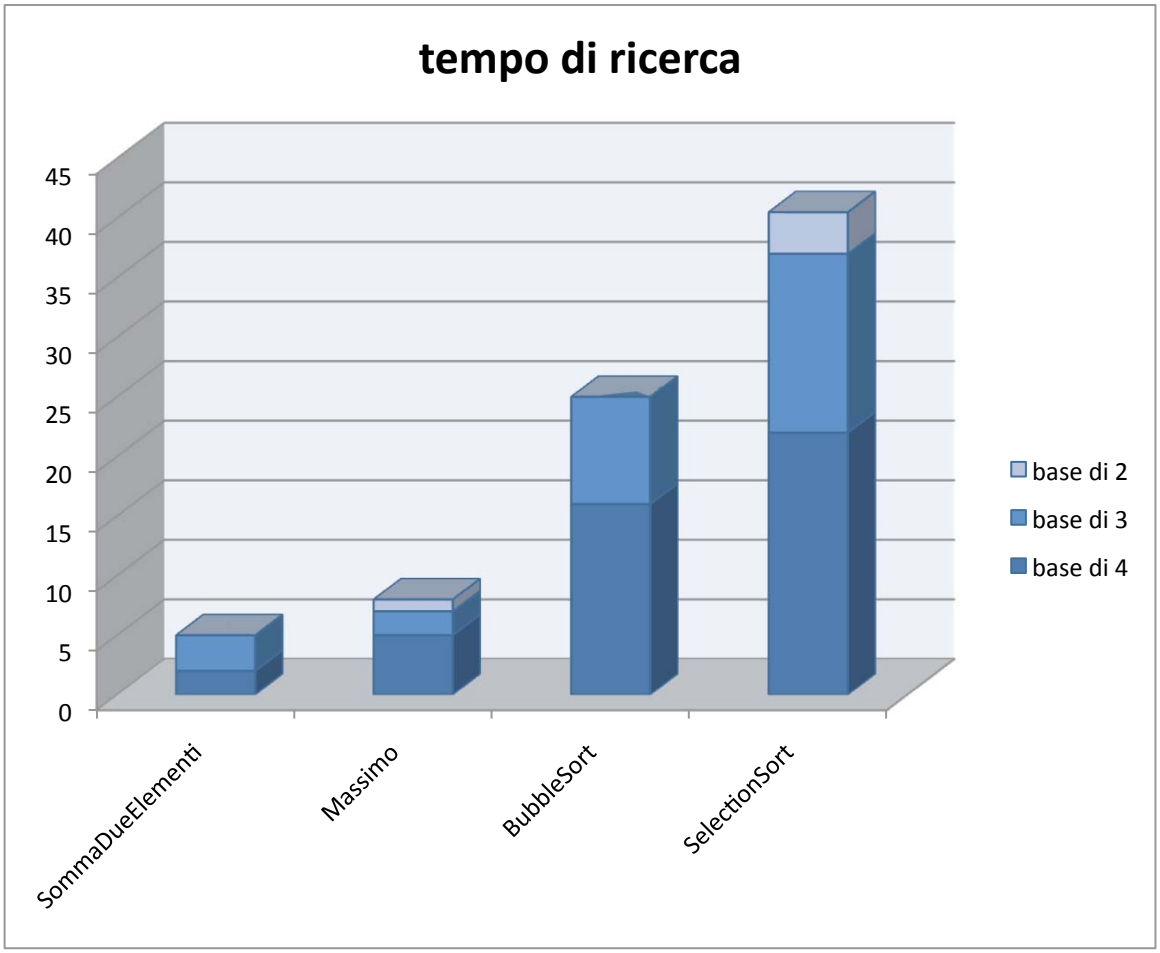
Variabile	range
Size	4..4
I	0..5
J	0..5
Min	0..5
Temp	0..5
Vett	0..5
Vett.length	4

RISULTATI

Ricerca base di 4:	non trovata in 22 min
Ricerca base di 3:	non trovata in 15 min
Ricerca base di 2:	base trovata in 3.5 min

RIEPILOGO DEI RISULTATI

Questo grafico mostra i tempi di ricerca di una base in Jasmine 2.0 supponendo che la dimensione iniziale di ricerca sia pari a 4. Per ogni funzione i diversi colori mostrano il tempo di ricerca della base di dimensione corrispondente. Se per una funzione non è presente un colore, significa che è stata trovata una base pari al colore precedente e quindi la ricerca di quella dimensione non è mai avvenuta.



chiaramente, come si evince dal grafico, con il crescere della complessità della funzione, intesa come complessità del grafo ottenuto, il tempo di ricerca della base diventa molto maggiore. Questo dipende dal fatto che con l'implementazione con arco di ritorno la lunghezza del cammino viene moltiplicata per la dimensione della base che si sta cercando.

6. GESTIONE METODI MULTIPLI

Una delle evoluzioni naturali di Jasmine è quella del supporto a metodi multipli. Parliamo cioè di permettere al programma di funzionare anche con funzioni che al loro interno richiamano altre funzioni.
Di seguito traccio due possibili approcci al problema in modo da facilitare il compito di chi vorrà proseguire su questa strada.

SIMULAZIONE DELLO STACK

Questo metodo prevede di arricchire il modello NuSmv creato dotandolo dell'infrastruttura logica che gli permette di simulare le chiamate a metodo e di gestire una vera e propria simulazione dello stack.

Dobbiamo:

1. Fissare un bound sulla profondita' max dello stack (MaxDepth) e sul num max di variabili (MaxVars) locali da memorizzare in ogni record di attivazione (PC + param. attuali + var. locali)

2. Modellare lo stack come un array di MaxDepth elementi, ognuno dei quali è:
   - un array di MaxVars interi (param attuali + var locali della funzione in esecuzione)
   - il PC corrente per tale funzione
   - un ID per tale funzione

3. Il record i-esimo dell'array `stack' modella l'i-esimo record di attivazione.

   Questo puo' essere fatto con un array di MaxDepth elementi ognuno dei quali e' un array di MaxVars+2 elementi, oppure con 3 array separati (forse piu' leggibile):
   - `stackFunctions' array di MaxDepth elementi di interi (ID di funzioni)
   - `params' array di MaxDepth elementi di array di MaxVars elementi (parametri attuali e var. locali)
   - `PC' array di MaxDepth elementi di interi (PC)

4. Mantenere una var smv `top' che indirizzi l'elemento affiorante nello stack (e, conseguentemente, nei diversi array che lo implementano)

5. Creare un MODULE per ogni funzione, aggiungendo nella sez. VAR un riferimento a tutti i MODULE chiamati (questo potrebbe essere un problema, e quindi puo' anche rivelarsi necessario mettere tutto nello stesso module `allProcedures')

6. La sez. TRANS di ogni module deve avere, per ogni arco del grafo di flusso, tanti `case', uno per ogni possibile posizione che questa funzione, se in esecuzione, possa avere nell'array `stack' (uno per ogni valore di `top'). Ad es., per l'arco PC=2 --> PC=3 della funzione FUNZ() (contenente l'istruzione a=b, dove `a' è la `prima' var. locale, e `b' la seconda):

```
<?php
per ogni valore `t' di `top', genera:
top=t & stackFunctions[t]=<ID_DI_FUNZ> & PC[t]=2 :
next(params[top][0]) = params[top][1] &  -- <<<-------  a=b
next(params[top][altri]) = `invariati' &
next(params[altri][tutti]) = `invariati' &
```

```
next(top) = top &
next(PC[top]) = 3 &
next(PC[altri elem]) = `invariati';
next(stackFunctions[tutti]) = `invariati';

...

--istruzione di return:
per ogni valore `t' di `top', genera:
top=t & stackFunctions[t]=<ID_DI_FUNZ()> & PC[t]=<ISTR_RETURN> :
next(top) = top-1 &
--passaggio del valore ritornato nella var. locale del metodo chiamante (nello stack)
next(params[top-1][id_parametro_p]) = params[top][id_parametro_result di FUNZ()] &    <<----   istr del metodo
chiamante: p = FUNZ();
..

--istruzione di chiamata a procedura
...
top = top+1 & .....


php?>
```

E' l'approccio più generale possibile al problema della gestione dei metodi multipli ma aggiunge un overhead di complessità notevole al lavoro di NuSmv. Per cui la domanda è: è davvero necessario gestire lo stack di attivazione?

APPROCCIO UNIT TESTING

La risposta alla domanda che conclude il paragrafo sulla simulazione dello stack può trovare risposta quì:
In questo approccio consideriamo i metodi chiamati dalla funzione della quale vogliamo generare i casi di test come delle scatole nere.
Se la mia funzione contiene una chiamata a metodo del tipo: faisomma(a,b)
Semplicemente io wrappo la funzione faisomma(a,b) in un metodo java che la esegue con ogni parametro in input (entro un intervallo prefissato) e memorizza una tabella con il corrispondente risultato della funzione faisomma(a,b) in corrispondenza dei valori a e b.
In NuSmv si tratterà semplicemente di cablare la tabella ottenuta in modo che se ad esempio nel percorrere un cammino alla ricerca del caso di test ci sarà bisogno di faisomma(3,2) allora NuSmv saprà sostituire il valore corretto che quella funzione avrebbe restituito.

Esempio:

Supponiamo di voler testare la funzione:

```
Public static bool dispari(int a){
        return !isPari(a);
}
```

mentre costruiamo il modello smv della funzione dispari(int a) ci accorgiamo che il metodo isPari(int a) non è un'istruzione semplice ma è altresì una chiamata a metodo. A questo punto supponiamo che l'utente avesse inserito come range della variabile a l'intervallo 0..3.
Dobbiamo quindi costruire una tabella che indichi ogni risultato dell'esecuzione del metodo isPari(int a) con ogni possibile valore in ingresso:

Esecuzione di isPari(int a)

Valore di a	Risultato
0	True
1	False
2	True
3	False

Questa tabella viene compilata incapsulando la funzione isPari ed eseguendo il metodo variando gli argomenti in base al range delle variabili.

Una volta costruita la tabella si torna a scrivere il modello nusmv, cablando nel codice i risultati della tabella

In questo caso, quella che sarebbe stata in nusmv la parte di codice:

```
TRANS
 case
   PC = 1 : next(PC) = 2 & next(a)=a & next(result) =isPari(a)  ;
```

diventa:

```
TRANS
 case
   PC = 1 & a=0 : next(PC) = 2 & next(a)=a & next(result) =true;
   PC = 1 & a=1 : next(PC) = 2 & next(a)=a & next(result) =false;
   PC = 1 & a=2 : next(PC) = 2 & next(a)=a & next(result) =true;
   PC = 1 & a=3 : next(PC) = 2 & next(a)=a & next(result) =false;
```