Generazione automatica di codice nuSmv per la verifica di metodi Java

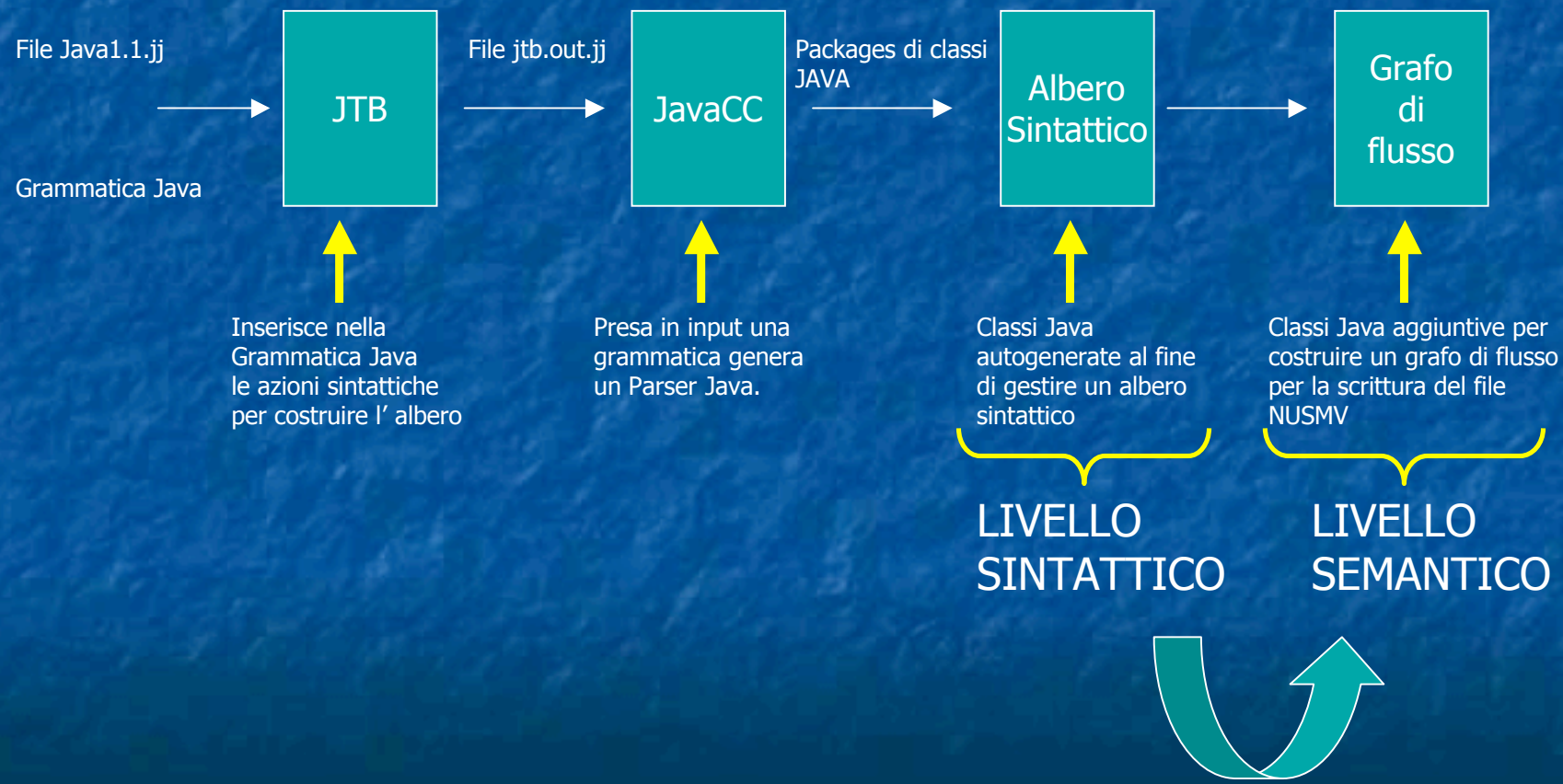
Sapienza Università di Roma
Laurea specialistica in Ing. Informatica
Corso di Metodi Formali nell'Ingegneria del SW
Prof. Toni Mancini

A cura di
Daniele Ippoliti
Marco Leone

Introduzione

- In questo progetto abbiamo realizzato una estensione del tool *JTB (Java Tree Builder).*
- **INPUT**: codifica di un metodo Java.
- **OUTPUT di JTB std:** se il codice compila, un syntaxTree.
- **Output di JTB esteso:** codifica .smv del metodo java, su cui poi poter fare verifica di proprietà con NuSmv.

Processo di generazione del JAVA PARSER

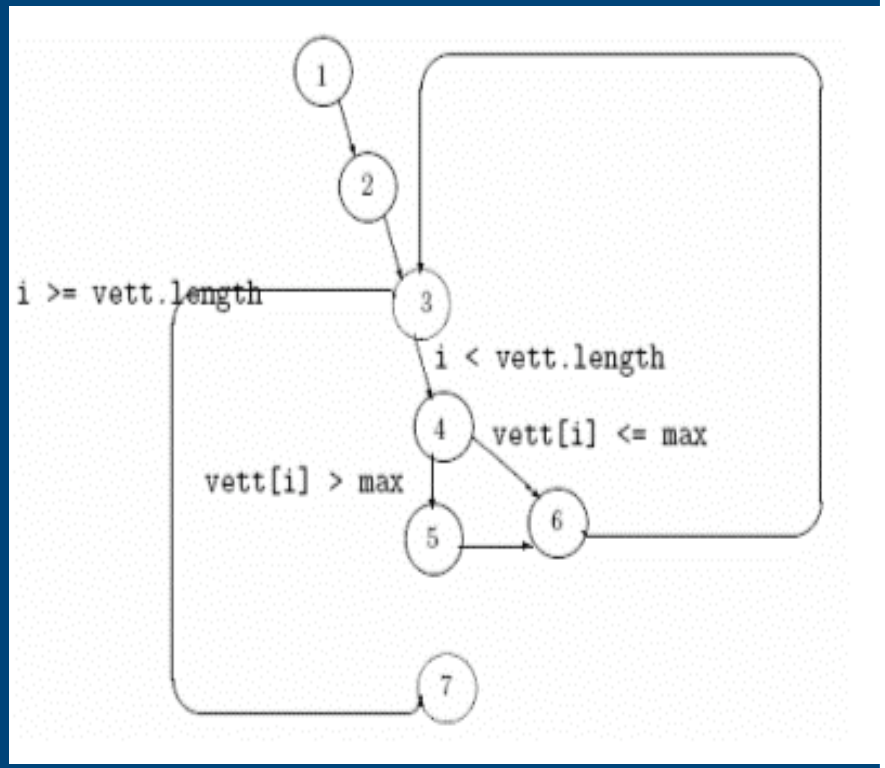


Esempio: Massimo.java

```
public static int massimo(int[] vett) {

        int result = 0;                          /* 1 */
        int i = 0;                               /* 2 */
        while(i < vett.length) {                 /* 3 */
                if (vett[i] > result)            /* 4 */
                        result = vett[i];                /* 5
*/
                i++;                             /* 6 */
        }
        return result;                           /* 7 */

}
```

Massimo.smv

```
MODULE massimo

VAR
 result : 0..3;
 i : 0..3;
 vett : array 0..2 of -10..10;
 PC : 1..7;

ASSIGN
 next(vett[0]) := vett[0] ;
 next(vett[1]) := vett[1] ;
 next(vett[2]) := vett[2] ;

DEFINE
 TERM := PC = 7;

TRANS
 case
   PC = 1 : next(PC) = 2 & next(result)  =0 & next(i) = i ;
   PC = 2 : next(PC) = 3 & next(result) = result & next(i)  =0 ;
   PC = 3 &  i<vett.length : next(PC) = 4 & next(result) = result & next(i) = i ;
   PC = 3 & ! i<vett.length : next(PC) = 7 & next(result) = result & next(i) = i ;
   PC = 4 &  vett[i]>result : next(PC) = 5 & next(result) = result & next(i) = i ;
   PC = 4 & ! vett[i]>result : next(PC) = 6 & next(result) = result & next(i) = i ;
   PC = 5 : next(PC) = 6 & next(result)  =vett[i] & next(i) = i ;
   PC = 6 : next(PC) = 3 & next(result) = result & next(i) =  ++ ;
   PC = 7 : next(PC) = 7 & next(result) = result & next(i) = i ;
esac
--end MODULE massimo
```



La sezione VAR:

Prima visita del syntaxTree:

- memorizzazione nome e tipo delle variabili locali/parametri formali (int o int[]).
- conteggio istruzioni (PC).

VAR
result : 0..3;
i : 0..3;
vett : array 0..2 of -10..10;
PC : 1..7;

La sezione ASSIGN:

Side-effect:

- variabili locali.
- parametri formali di tipo int.

```
ASSIGN
 next(vett[0]) := vett[0] ;
 next(vett[1]) := vett[1] ;
 next(vett[2]) := vett[2] ;
```

La sezione DEFINE:

Prima visita del syntaxTree:

- memorizzazione istruzioni return.

```
DEFINE
 TERM := PC = 7;
```

La sezione TRANS :

ALBERO SINTATTICO

albero ad Oggetti che permette una agevole esplorazione tramite visite ricorsive dell'informazione sintattica.

GRAFO DI FLUSSO

grafo che rappresenta in modo intuitivo l'informazione semantica del flusso d'esecuzione del metodo.

```
TRANS
 case
   PC =
   PC =
   PC =
   PC =
   PC =
   PC =
   PC =
   PC = 6 : next(PC) = 3 & next(result) = result & next(i) =  ++ ;
   PC = 7 : next(PC) = 7 & next(result) = result & next(i) = i ;
esac
```

ni
entuale
lizionale
lle var.
ram.
formali che possano
subire side-effect.

I nostri sforzi si sono concentrati prevalentemente sull'ottenere e memorizzare in opportune strutture dati, l'informazione semantica necessaria ad editare correttamente ed automaticamente la sezione TRANS, tramite più visite del syntaxTree.

Oggetti Nodo:

- Per memorizzare l'informazione sulla semantica di transizione da riproporre nelle righe della sezione TRANS abbiamo previsto delle strutture dati Nodo con:

INIT_PC	CONDITION	DEST_PC	EXPRESSION	VAR_NAME

1. Il valore del PC di partenza(obbligatorio) per la transizione;
2. La condizione di etichettatura (opzionale) dell'arco del grafo di flusso relativo alla transizione;
3. Il valore del PC di destinazione della transizione (obbligatorio);
4. L'espressione(opzionale) di side-effect, a seguito della transizione;
5. Il nome(opzionale) del parametro formale o della variabile locale che abbia subito il side-effect.

Problematiche affrontate:

SOLUZIONI :

- Implementazione di soluzione algoritmiche su 3 Visitors sequenziali.
- Algoritmi offline (correttori d'errore)

calcolo del corretto valore del PC di partenza

INIT_PC	CONDITION	DEST_PC	EXPRESSION	VAR_NAME

calcolo del corretto valore del PC di destinazione

IfStatement del syntaxTree

GRAMMAR PRODUCTION:

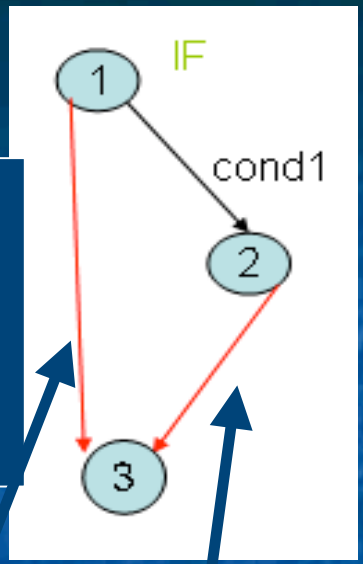
* f0 -> "if"
* f1 -> "("
* f2 -> Expression()
* f3 -> ")"
* f4 -> Statement()
* f5 -> ["else" Statement()]

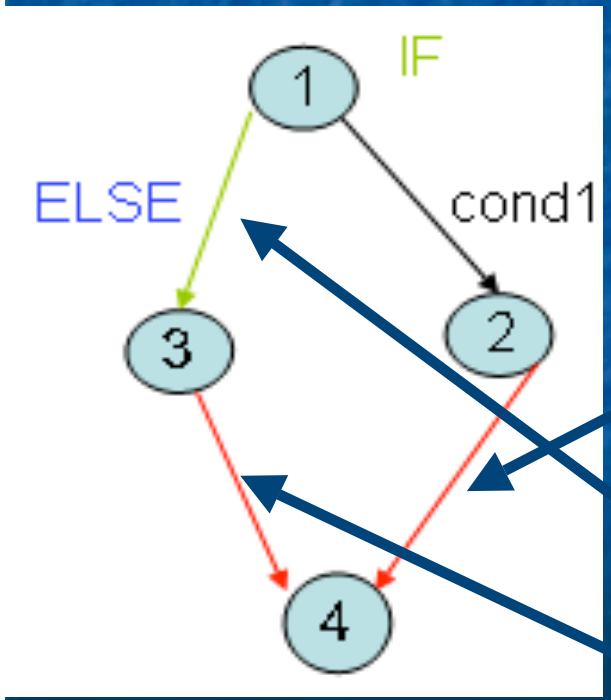
INIT_PC	CONDITION	DEST_PC	EXPRESSION	VAR_NAME

Pattern per l'ifStatement

```
if(cond1)        //1
    istr;        //2
else istr;       //3
istr;            //4
```

```
if(cond1)        //1
    istr;        //2
istr;            //3
```





FORMULE DEL PATTERN :

- **PC_if + If_+.length → PC_if + If.length**

(senza else):

PC_if → PC_if + If.length

(con else):

- **PC_if → PC_if + If_+.length + 1**
- **PC_if + if.length- 1 → Pc_if + If.length.**

Statement Condizionali-L'Oggetto WhileStatement del syntaxTree

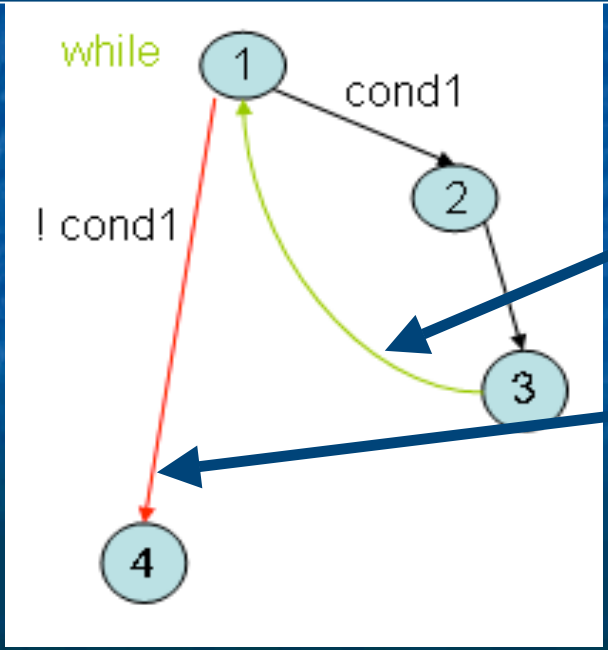
GRAMMAR PRODUCTION:

INIT_PC	CONDITION	DEST_PC	EXPRESSION	VAR_NAME

* f0 -> "while"
 * f1 -> "("
 * f2 -> Expression()
 * f3 -> ")"
 * f4 -> Statement()

Pattern per il WhileStatement

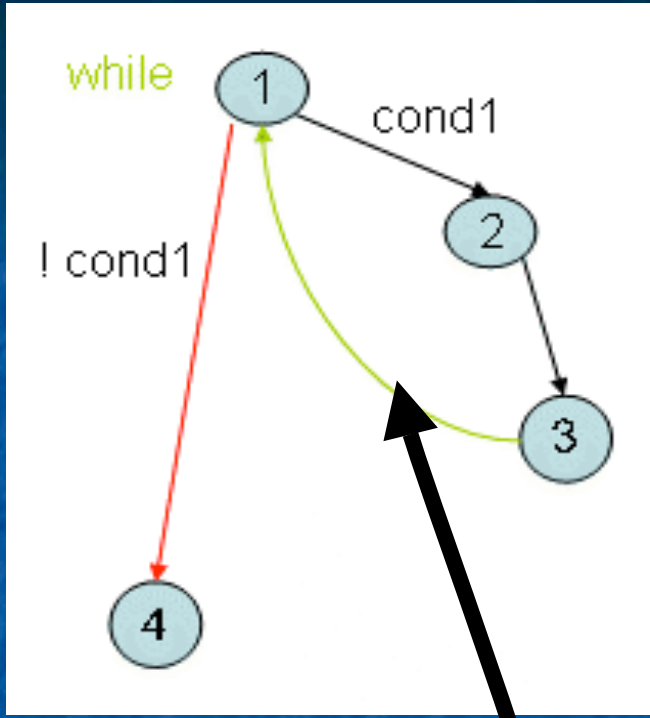
```
while(cond1){      //1
      istruzione;  //2
      istruzione;  //3
}
istruzione;        //4
```



FORMULE DEL PATTERN:

- **Pc_while + while$_+$.length → PC_while**
- **PC_while → PC_while + while.length**

BackEdgeTable:



- **Nel primo Visitor popoliamo una struttura dati (BackEdgeTable) con una BackEdgeEntry per ogni WhileStatement visitato;**

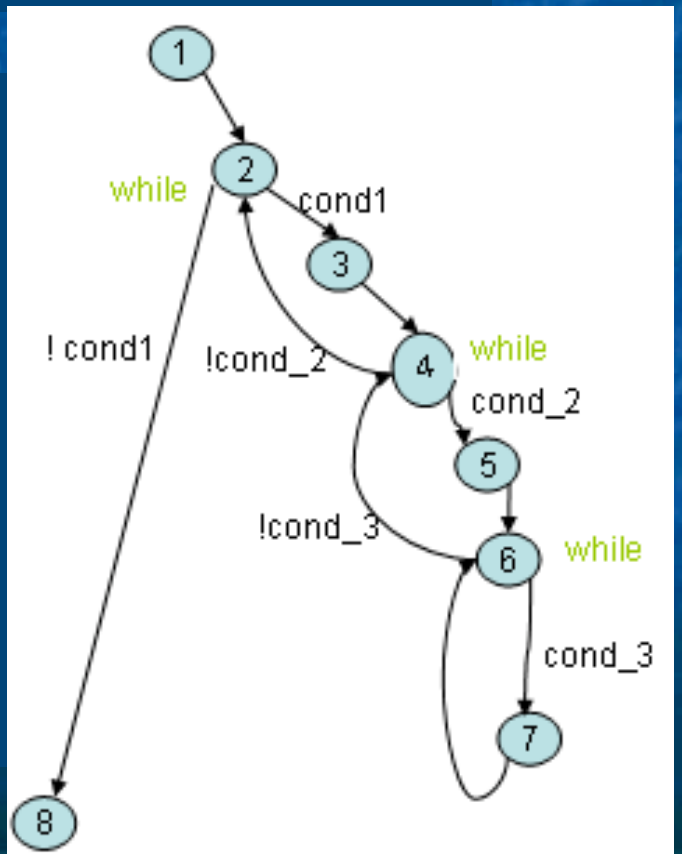
- **Ogni riga mantiene le informazioni sui valori de l' init_PC (*IstructionJump)* e dest_PC(*IstructionPc)* per la *transizione dell' arco all'indietro* (regola pattern: Pc_while + $while_+$.length → PC_while).**

InstructionJUMP (Pc_while + $while_+$.length)	***InstructionPC*** (PC_while)
3	1
…	…
…	…

Casi critici dei While annidati(1/3)

- Possibilità di commettere errori di calcolo applicando la regola del pattern per le configurazioni dei *casi critici di While annidati* annidati, con lo stesso valore per l'*InstructionJUMP*;

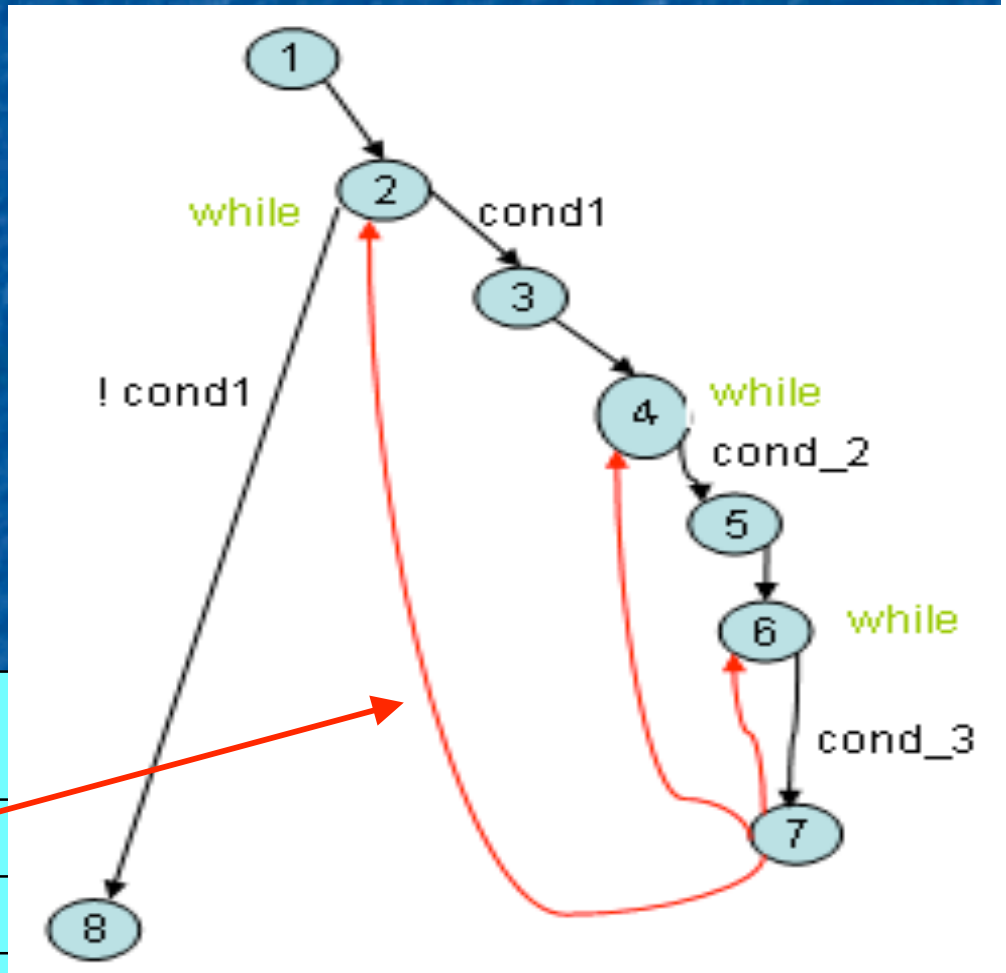
```
int result = i*j;                    //1
    while(result>50){                //2
        result=result-5;             //3

        while(result>100){           //4
            result=result-10;        //5

            while(result>200){       //6
                result=result-20;    //7
            }
        }
    }
    return result;                   //8
```



Casi critici dei While annidati(2/3)

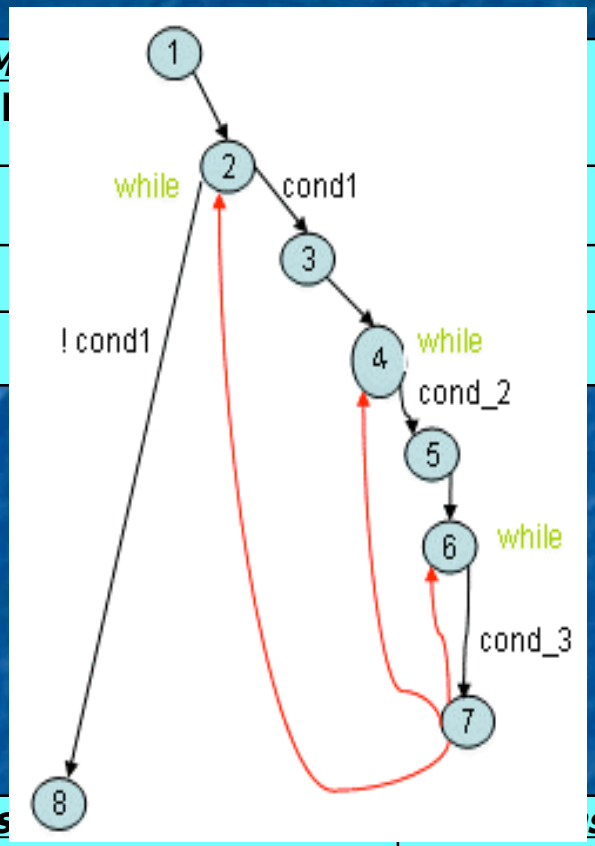
Visitor Uno

InstructionJUMP **(Pc_while + while$_+$.length)**
7
7
7

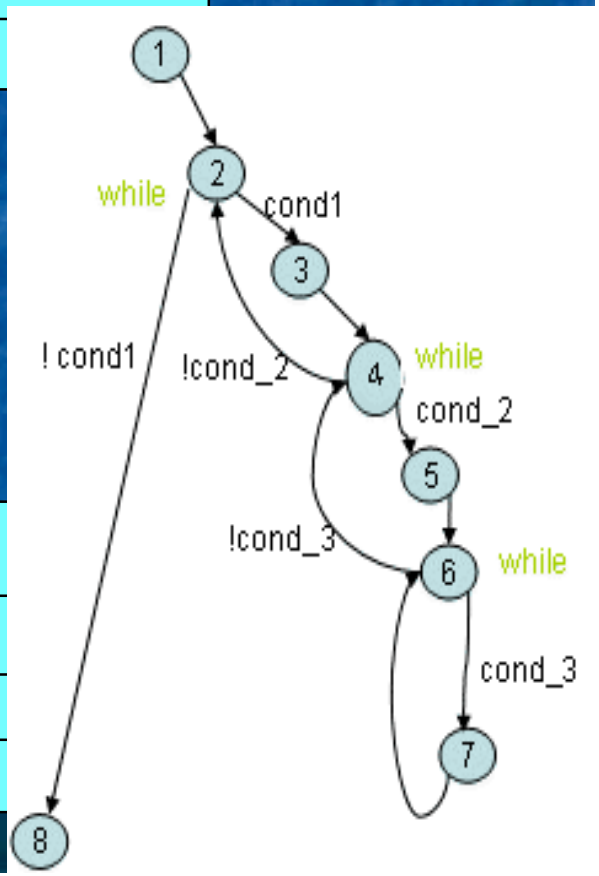


Casi critici dei While annidati:normalizzazione offline(3/3)

InstructionJUM (Pc_while + while$_+$.l	***tructionPC*** ***hile)***
7	2
7	4
7	6



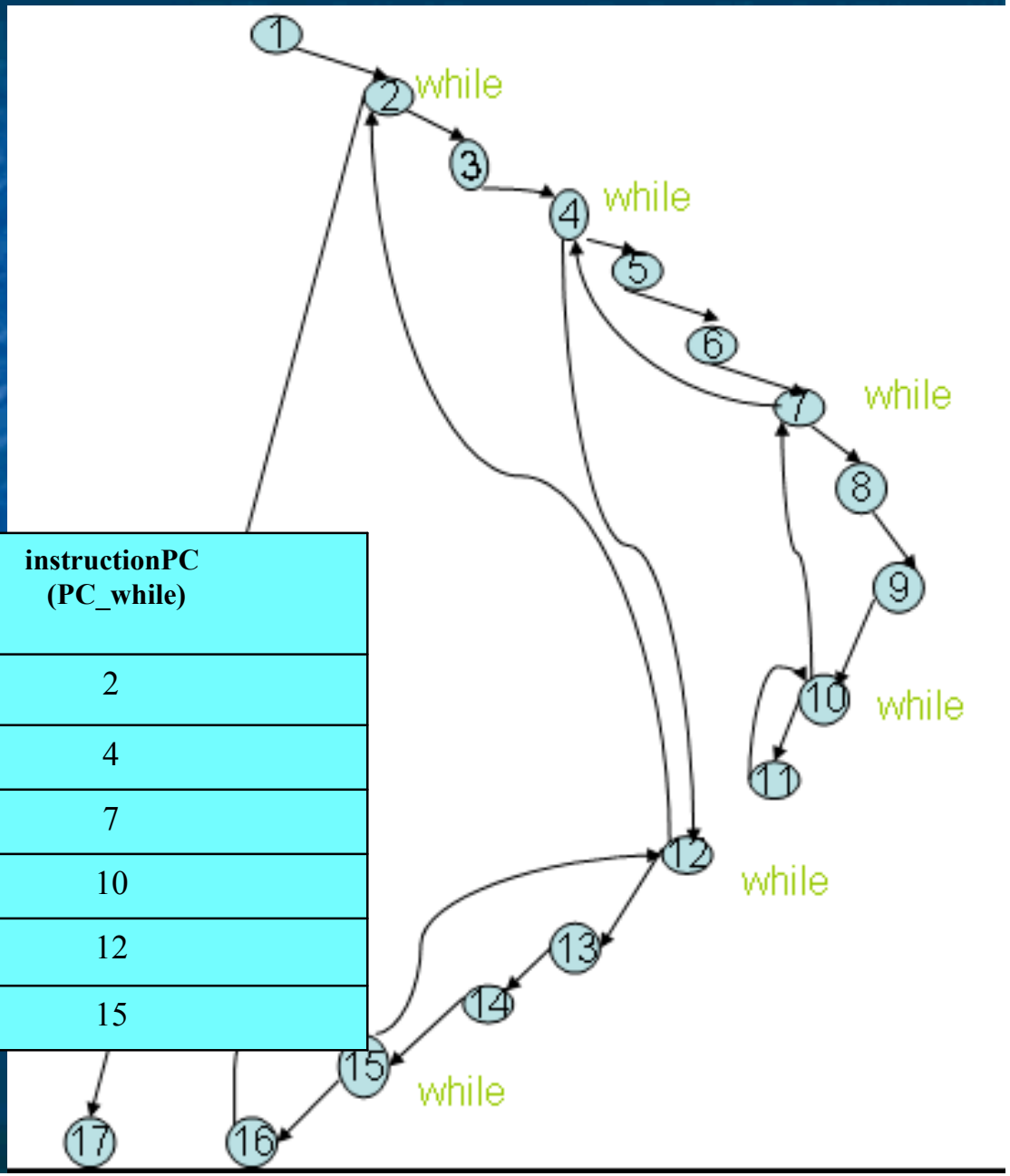
Ins (Pc_while + while$_+$.length)	***structionPC*** (PC_while)
4	2
6	4
7	6



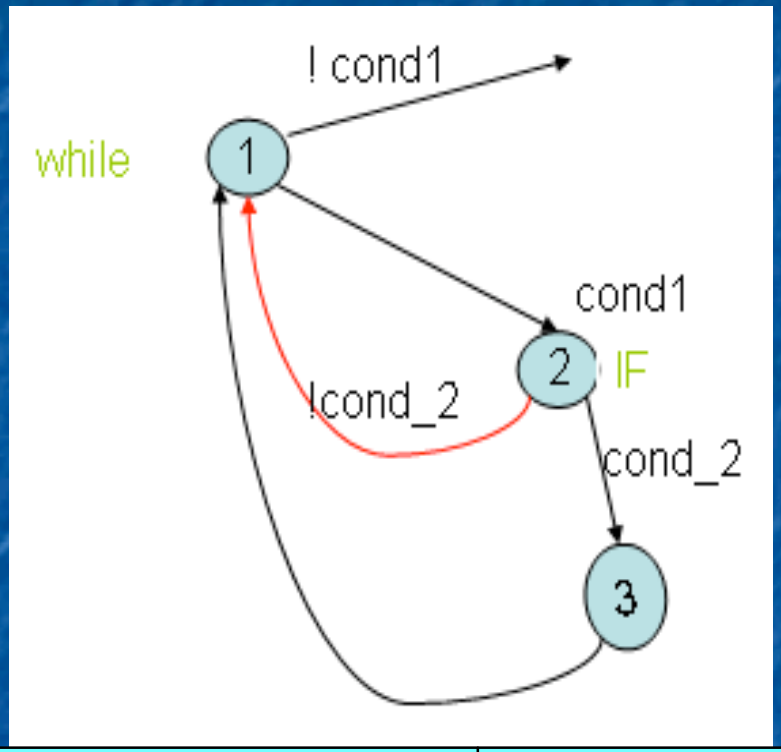
Generalità dell'algoritmo offline:

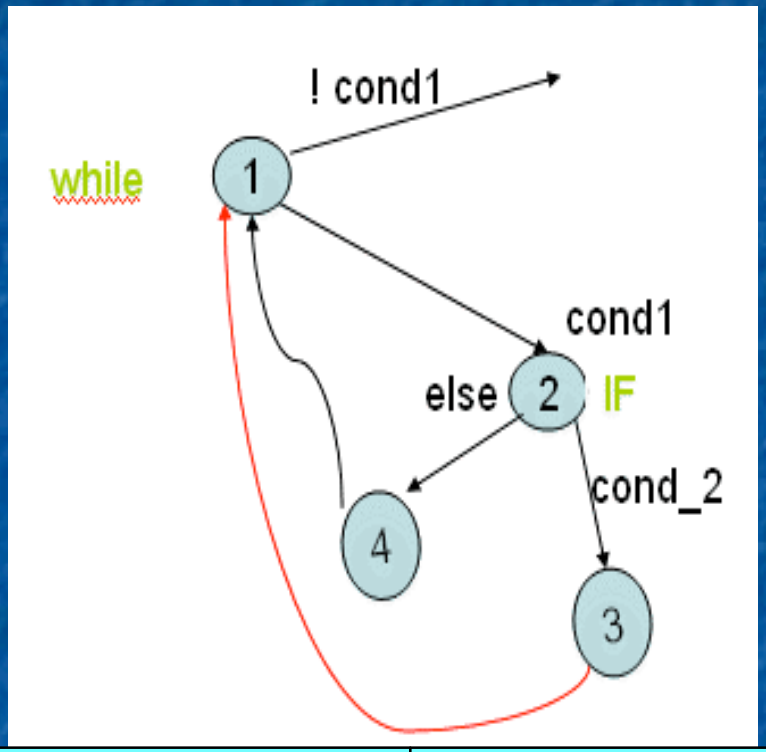
- **L'algoritmo offline normalizzatore della BackEdgeTable, corregge configurazioni di annidamenti comunque complessi.**

instructionJUMP (Pc_while + while.length -1)	instructionPC (PC_while)
~~16~~ 12	2
~~11~~ 7	4
~~11~~ 10	7
11	10
~~16~~ 15	12
16	15



Ulteriori archi all'indietro: If annidati nei While





InstructionJUMP (Pc_while + while$_+$.length)	InstructionPC (PC_while)
3	1

InstructionJUMP (Pc_while + while$_+$.length)	InstructionPC (PC_while)
4	1

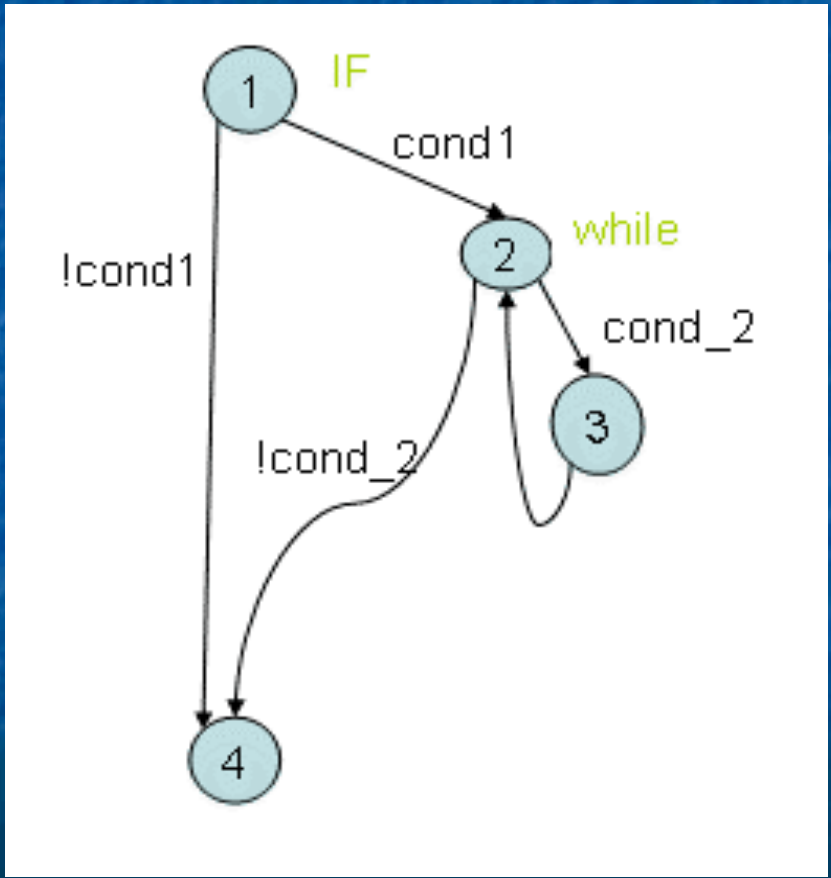
(PC_if + if.length -1) ==(PC_While +While$_+$.length)

Riconoscimento di if annidato nel while (1/2)

BASTA VERIFICARE CHE
(PC_if + if.length -1)==(PC_While +While$_+$.length)
??

<u>CONDIZIONE NECESSARIA</u>
<u>MA NON SUFFICIENTE!!</u>

→→→ NO ARCO ALL'INDIETRO!!

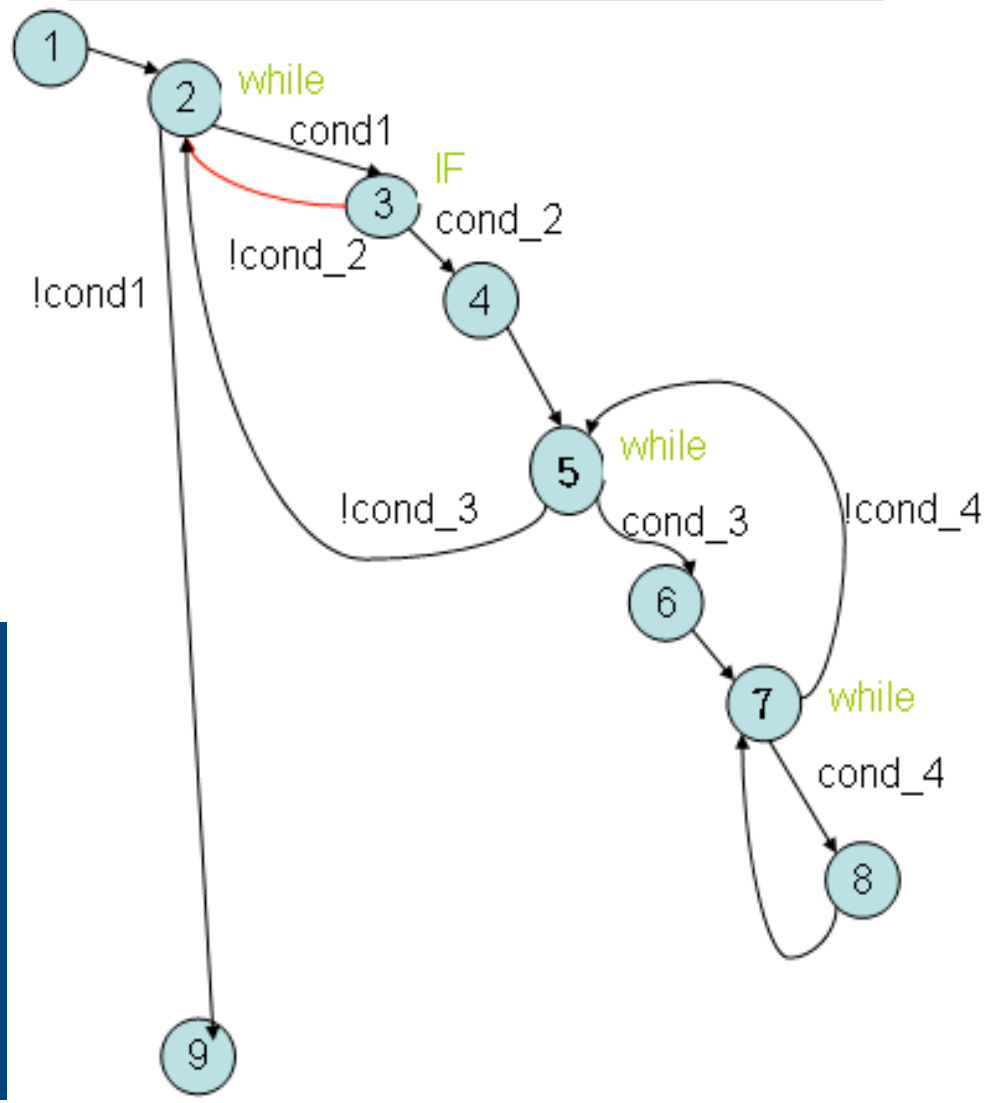


Riconoscimento di if annidato nel while (2/2)

(PC_if + if.length -1)==(PC_While +While$_+$.length)

```
int result=k;                          //1
while(result>0){                       //2

    if(result> i){                     //3
        result=result-i;               //4

        while(result>i){               //5
            result= result -1 ;        //6

            while(result >i*2){        //7
                result = result-5;     //8
            }
        }
    }
}
return result;                         //9
```

CASO DI ANNIDAMENTO MISTO



SOLUZIONE:

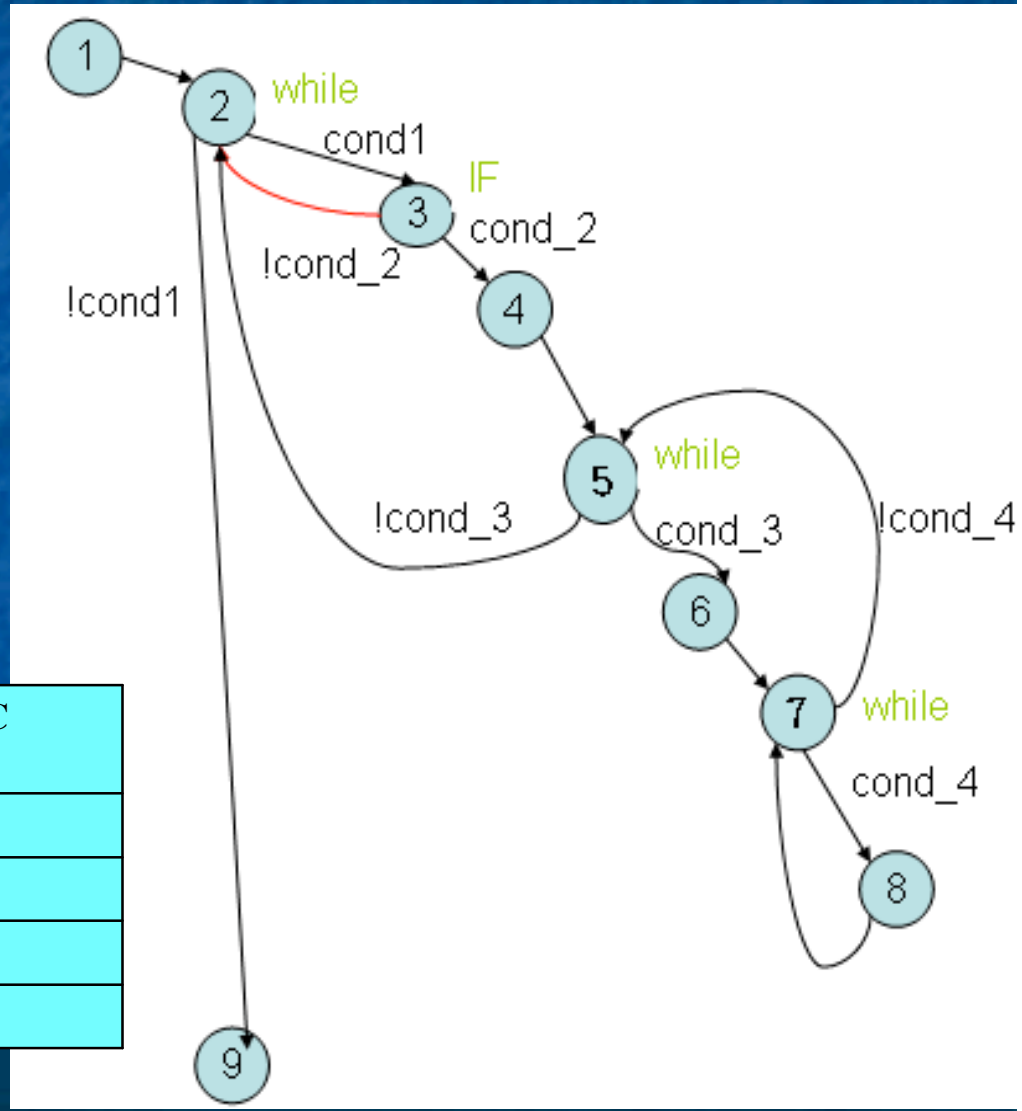
Nel secondo Visitor interrogare iterativamente la BackEdgeTable, per trovare il minimo cammino di origine in <u>Pc_if+if.length+1</u> che conduca ad un valore <u>PC* < PC_if;</u>

→→→ dest_PC=PC* .

E L'init_PC dell'arco all'indietro ?
Caso banale:<u>if senza else</u>

init_PC = PC_if

come da pattern



instructionJUMP	instructionPC
5	2
7	5
8	7
3	2

E L'init_PC dell'arco all'indietro ? Caso dell' <u>if con else</u>

PC_if+ if_+.length

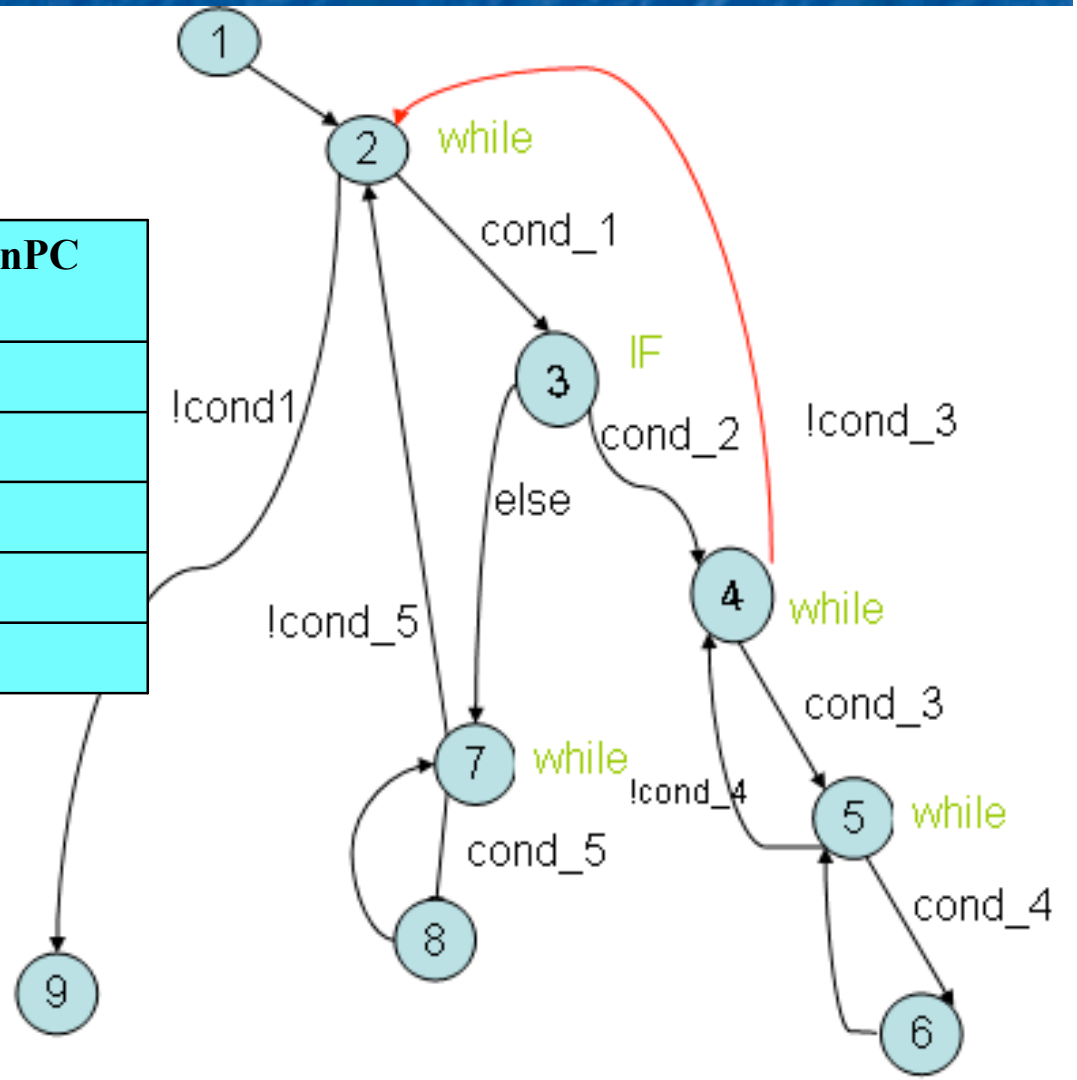
del pattern NON BASTA!

instructionJUMP	instructionPC
7	2
5	4
6	5
8	7
4	2

S
I
B
l'
r
di origine in

<u>PC_if+ if_+.length</u>

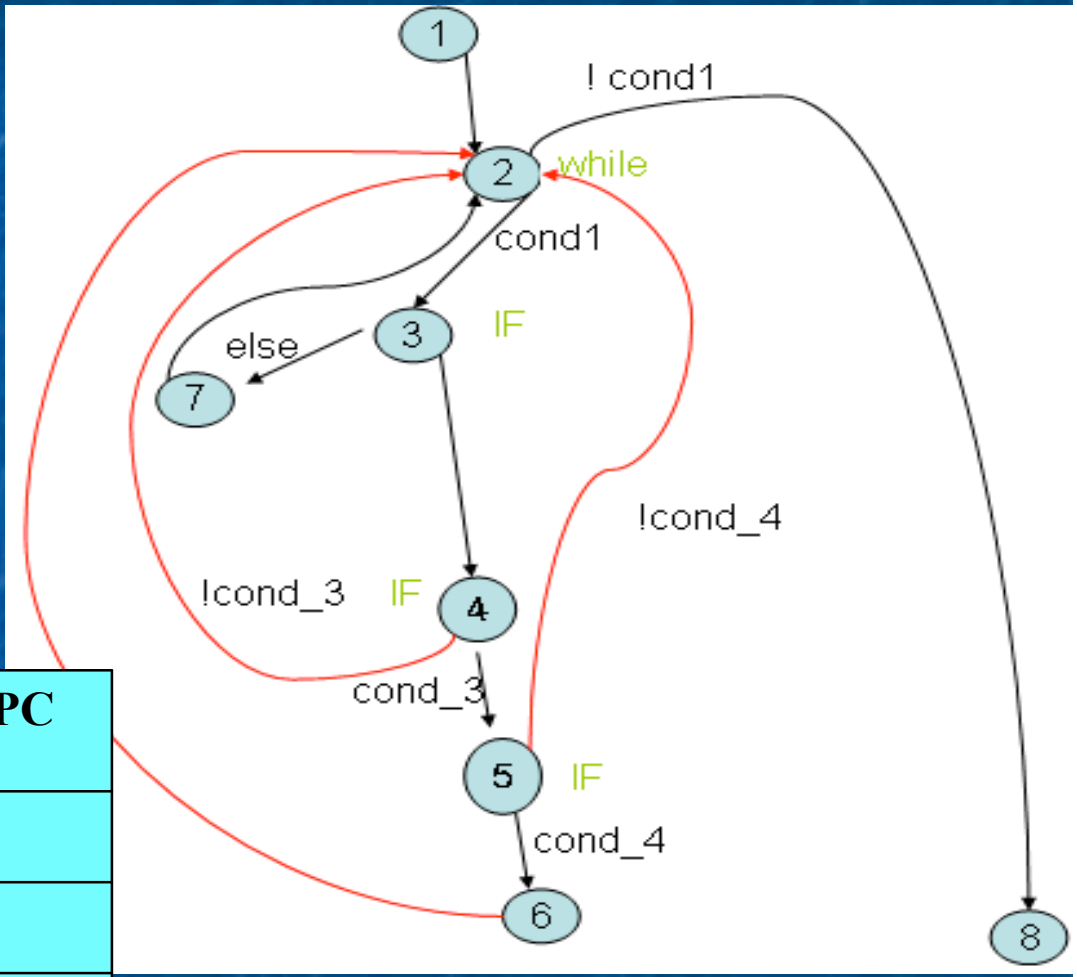
→→→ init_PC=PC* .



Generalità dell'algoritmo del secondo Visitor:

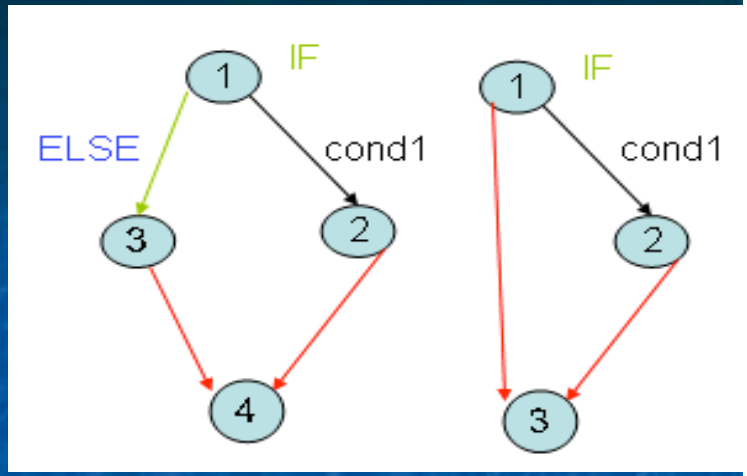
(PC_if + if.length -1)==(PC_While +While$_+$.length)

- **L'algoritmo di aggiunta archi all'indietro mancanti per i casi di If annidati nei While, gestisce correttamente configurazioni comunque complesse.**



InstructionJUMP	InstructionPC
7	2
6	2
4	2
5	2

IfFrontEdgeTable:



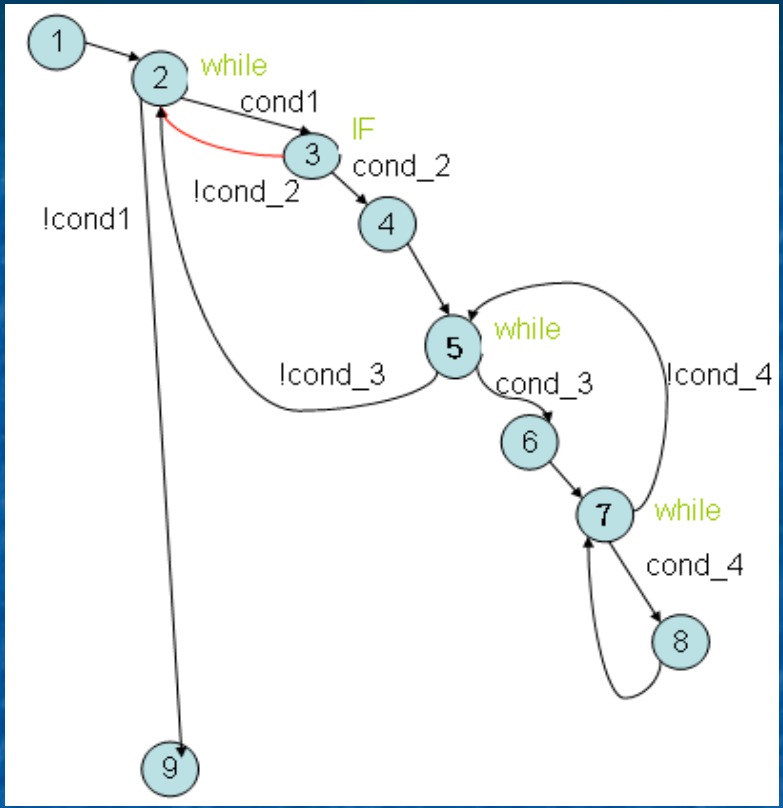
- **Terzo Visitor, e nuova struttura dati per gli IfStatement, per memorizzare le informazioni sulle transizioni di uscita con archi in avanti (no annidamento nel While)**

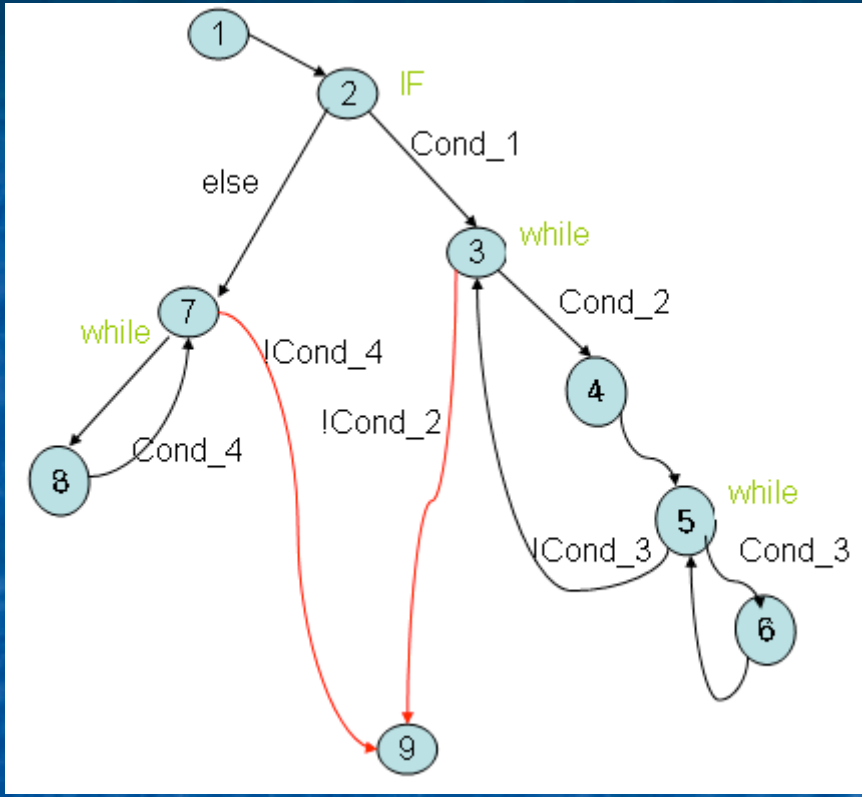
PC_if	*OpzionalElseJumpPC*	*InitPC_1*	*InitPC_2*	*Exit_Pc*

- **Ogni IfFrontEdgeEntry memorizza:**

1. **valore del Pc_If;**
2. ***OpzionalElseJumpPc*, per il destPc della transizione opzionale verso la ramificazione else, del valore di PC_If+ If$_+$.length+1 o -1 (se senza else);**
3. ***InitPC_1*, del pc di partenza della prima transizione di uscita;**
4. ***InitPC_2*, del pc di partenza della seconda transizione di uscita;**
5. ***ExitPc*, col valore PC_If +if.length, del pc di destinazione di uscita.**

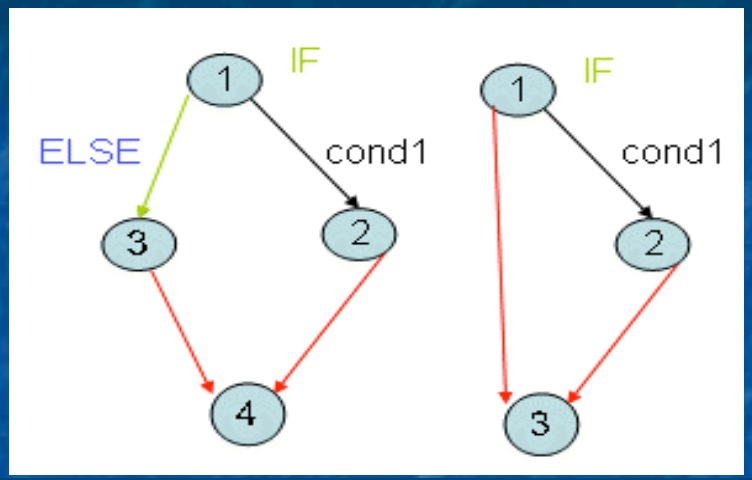
Primo Check: if con archi in avanti o all'indietro??

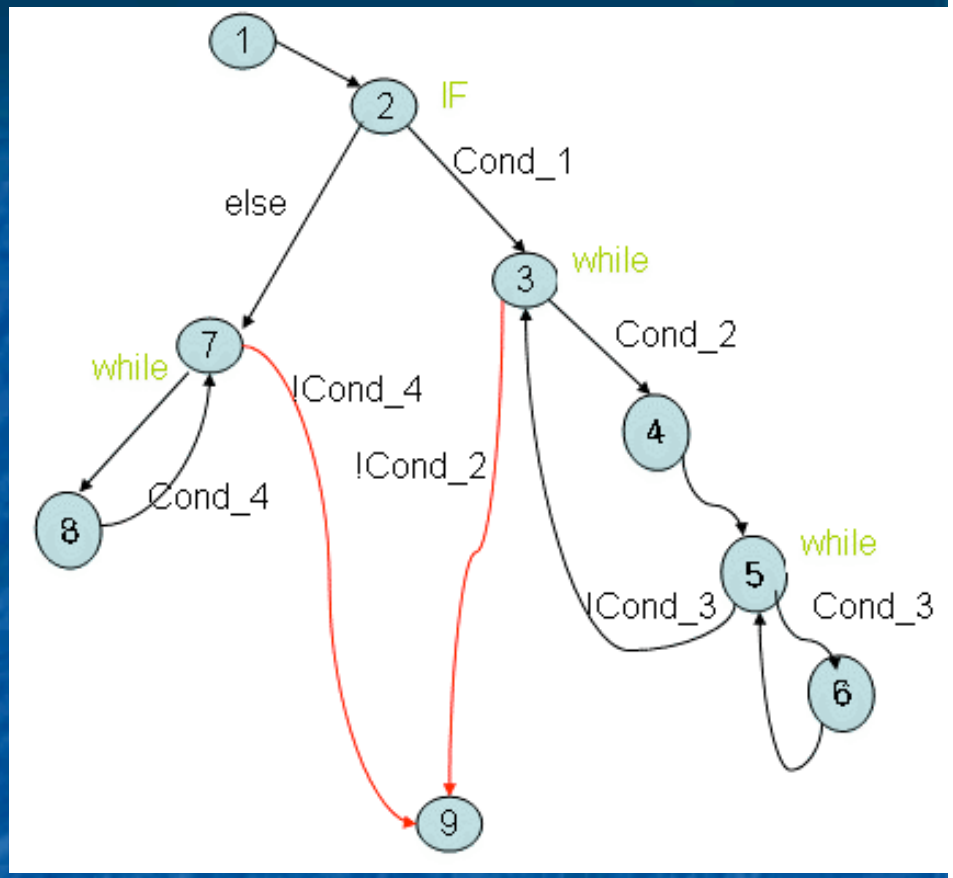




Il Terzo Visitor deve riconoscere gli ifStatement con transizioni di uscita di in avanti, piuttosto che all' indietro:
interroga iterativamente la BackEdgesTable e verificare se esiste un cammino di archi all'indietro di origine in <u>Pc_if+if.length-1</u>, che possa raggiungere un valore PC* < *Pc_if* (caso if con archi all'indietro),
Altrimenti può popolare la IfFrontEdgeTable con una nuova IfFrontEdgeEntry.
(caso if con archi in avanti)

Assegnare i campi InitPC_1 e InitPC_2





Con la sola applicazione del pattern :

InitPC_1 = PC_if + if_+.length
e
InitPC_2 = PC_if+if.length-1

Caso più generale: n while annidati nell'if;
→→ assegnazione più sofisticata della sola applicazione del pattern:

Interrogare iterativamente la BackEdgesTable e assegnare a InitPC_1 ed InitPC_2 gli ultimi valori PC_1* e PC_2* raggiungibili con i due cammini di archi all'indietro di origine in <u>PC_if + if_+.length</u> e <u>PC_if + if.length-1</u>

Casi critici degli if annidati(1/3)

- Per assegnare il campo *Exit_PC* della ifFrontEdgeEntry il terzo Visitor usa la formula del pattern PC_If +if.length.
- Possibilità di errori di calcolo:

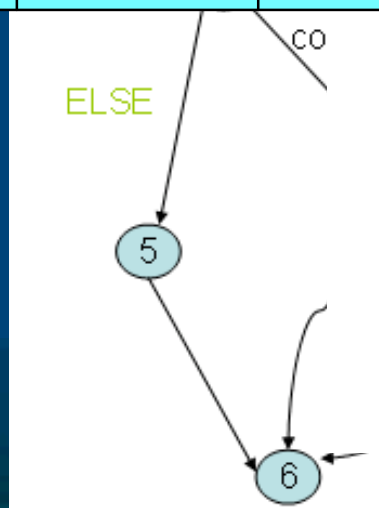
Casi critici di if annidati in if con ramificazione else, quando PC_If +if.length dell'if figlio = *OpzElseJumpPC* dell'if padre.

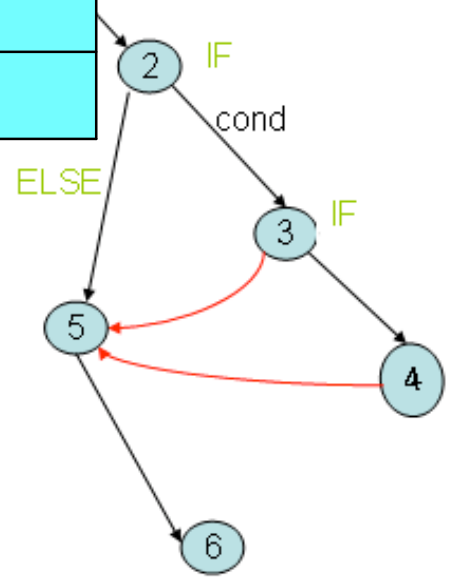
PC_IF	OPZ_ELSE_JUMP_PC (PC_If + f4.length+ 1)	EXIT_PC (PC_if + If.leng	INITPC_1	INITPC_2
2	5	6	4	5
3	-1	5	4	3

Grafo di flusso CORRET

ERRATO!

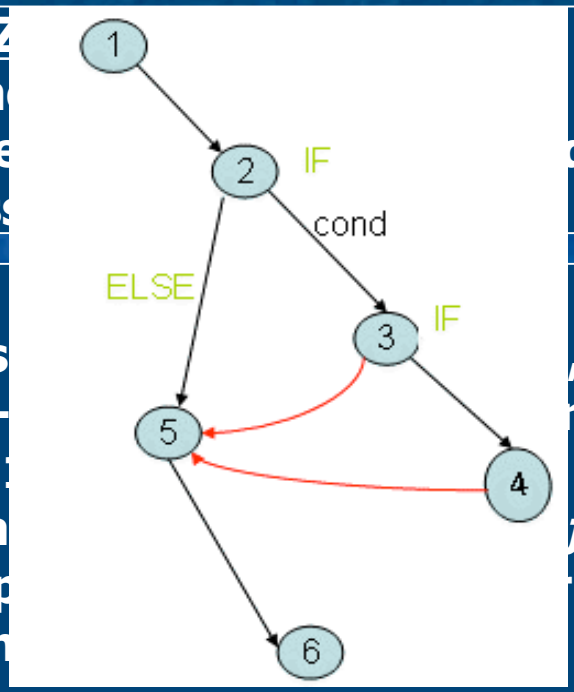
```
}
else{
        result= result+vett[0];        //5
}
return result;                         //6
```

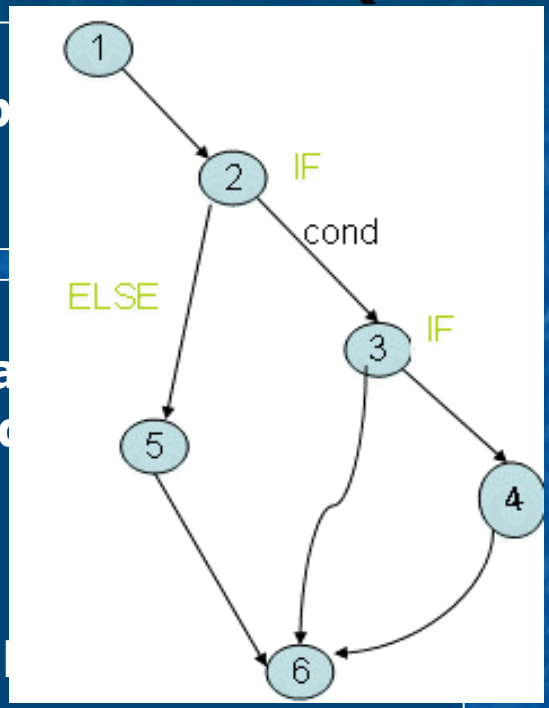




Casi critici degli if annidati(2/3)

RISOLUZ
algoritm **ifFrontEdgesTab**
per corre **di calcolo**
commess



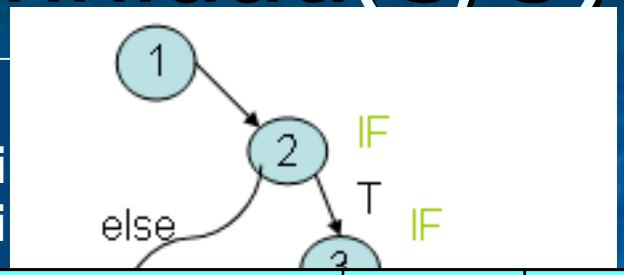


- **Scandis** **na**
- **Per la i-** **re le righe preced**
  per j=i-1
- **Se trova** ***j = ExitPc_i*** **,**
  deve app **riga i:**
  → assegn **E va avanti con**

	PC_IF	OPZ_ELSE_JUMP_PC (PC_If + f4.length+ 1)	EXIT_PC (PC_if + If.length)	INITPC_1	INITPC_2
j →	2	5	6	4	5
i →	3	-1	6	4	3

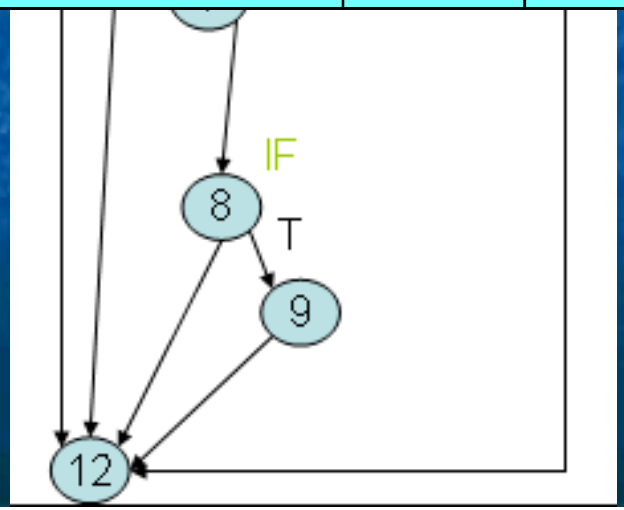
Casi critici degli if annidati(3/3)

- **<u>Algoritmo normalizzatore generalizzato:</u>**
  efficace nel riconoscimento e correggere errori nella IfFrontEdgesTable, per esempi di qualsiasi



	PC_IF	OPZ_ELSE_JUMP_PC (PC_If + f4.length+ 1)	EXIT_PC (PC_if + If.length)	…	…
j	2	11	12		
j i	3	10	12		
j	5	7	12		
	8	-1	12		

CORRETTA SEMANTICA DI TRANSIZIONE!



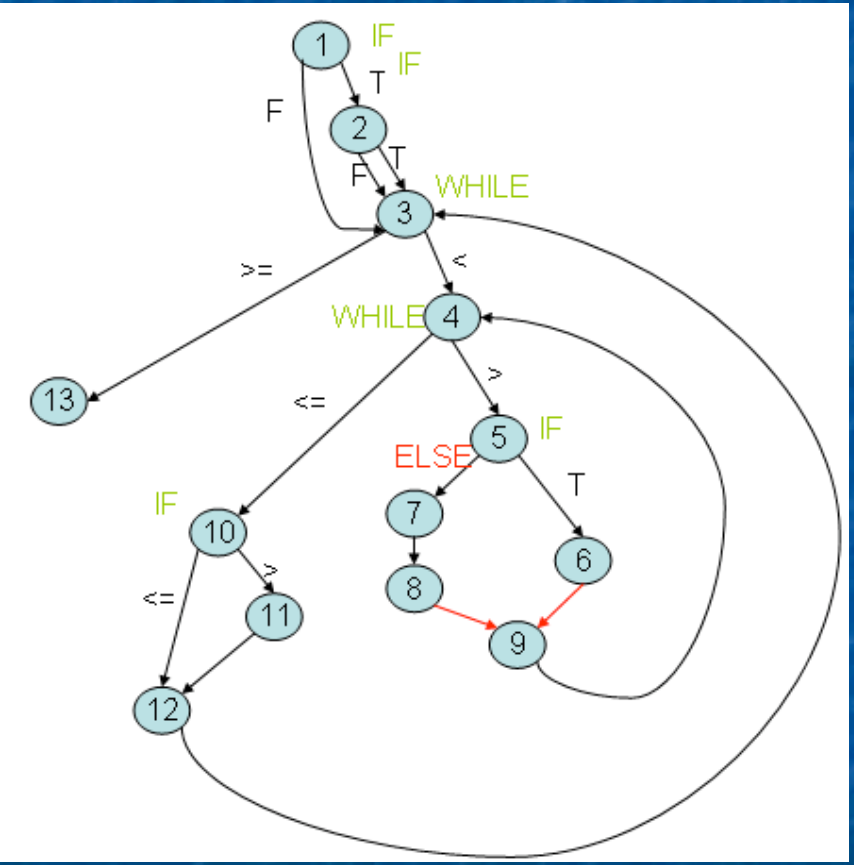
Last Visitor

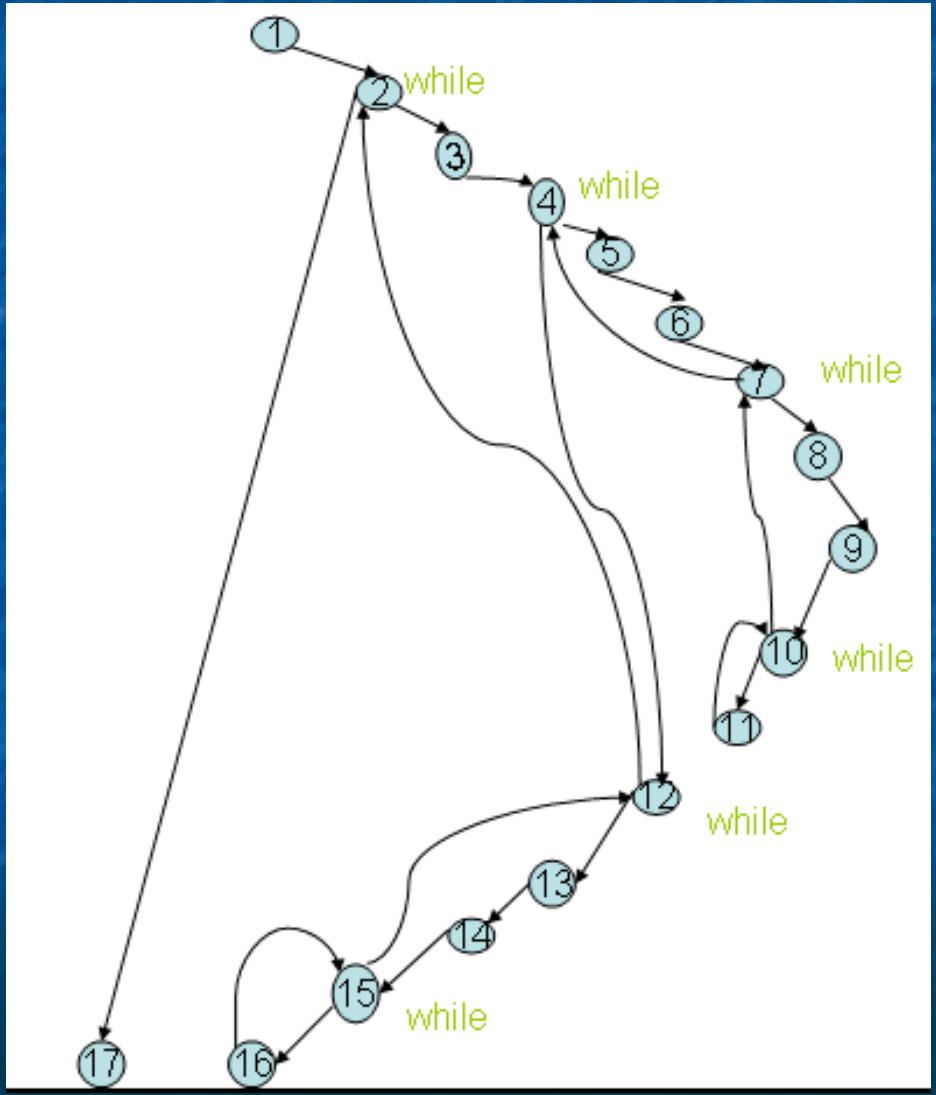
INIT_PC	CONDITION	DEST_PC	EXPRESSION	VAR_NAME

- Quarto ed ultimo Visitor:
  visita nuovamente il syntaxTree, e,
  sfrutta le BackEdgesTable e ifFrontEdgeTable,
  per ricavare per ogni Oggetto Nodo
  le corrette informazioni sui valori da assegnare ai campi
  init_PC e dest_PC,
  rispecchiando la semantica del grafo di flusso;

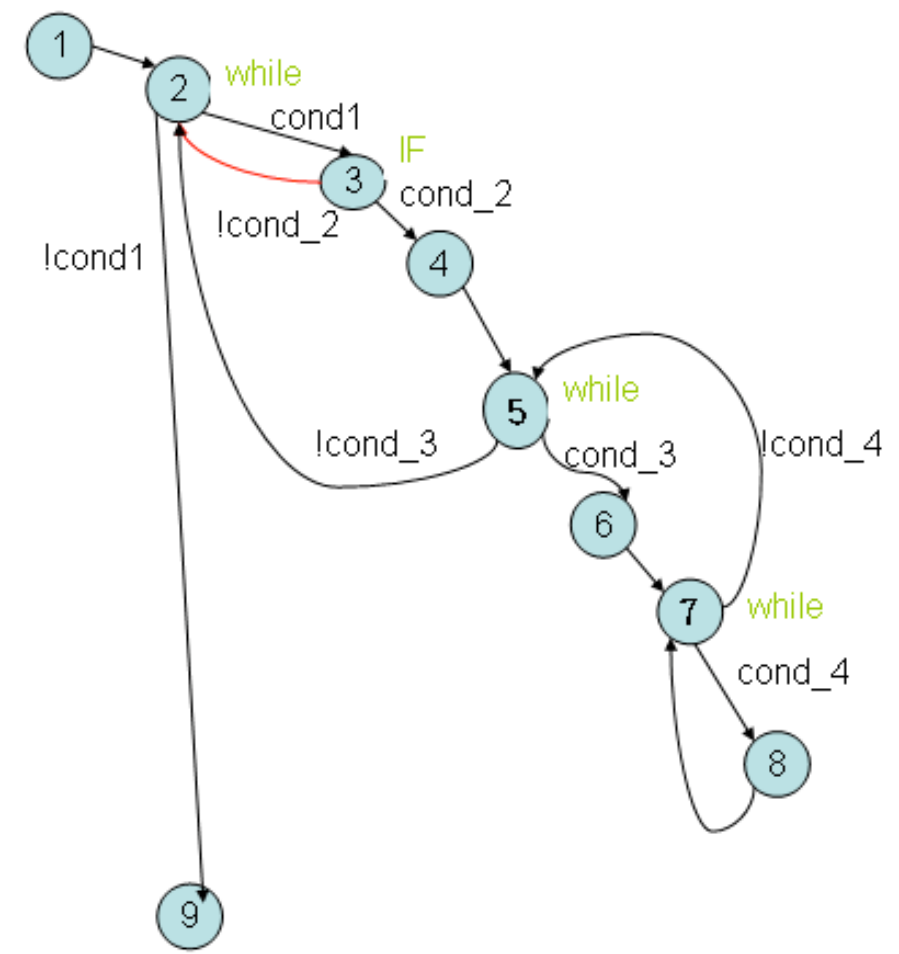
- Per quelle istruzione che non hanno alcuna transizione
  particolare memorizzata nelle due tabelle,
  può considerare banalmente
  init_PC=PC e dest_PC = PC+1;

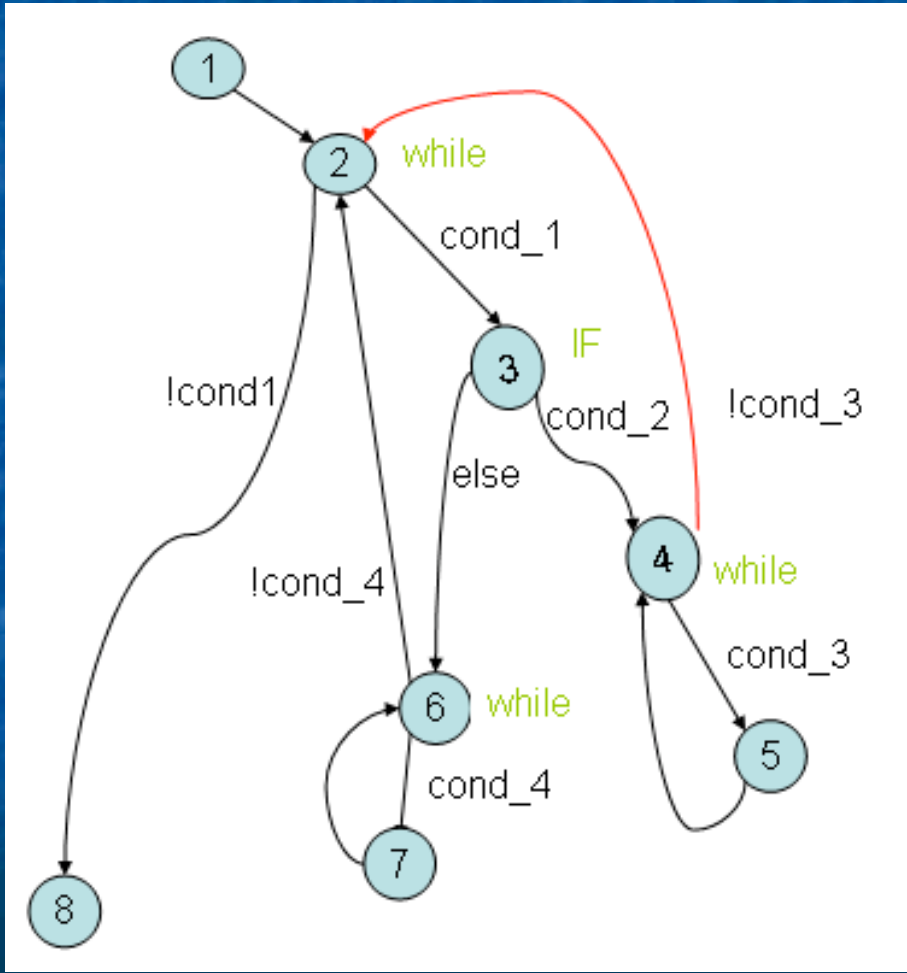
Alcuni Esempi :



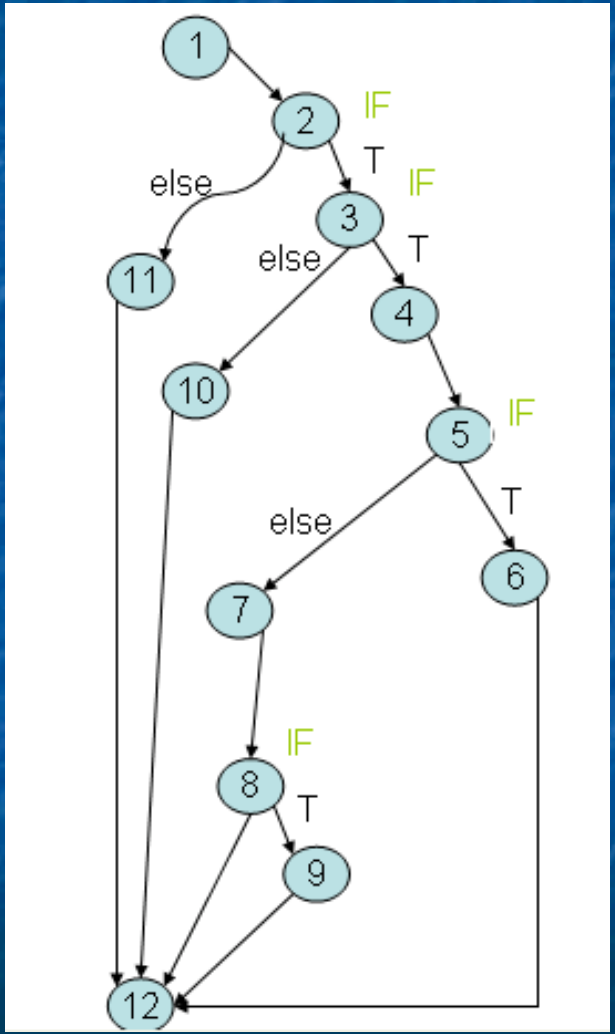


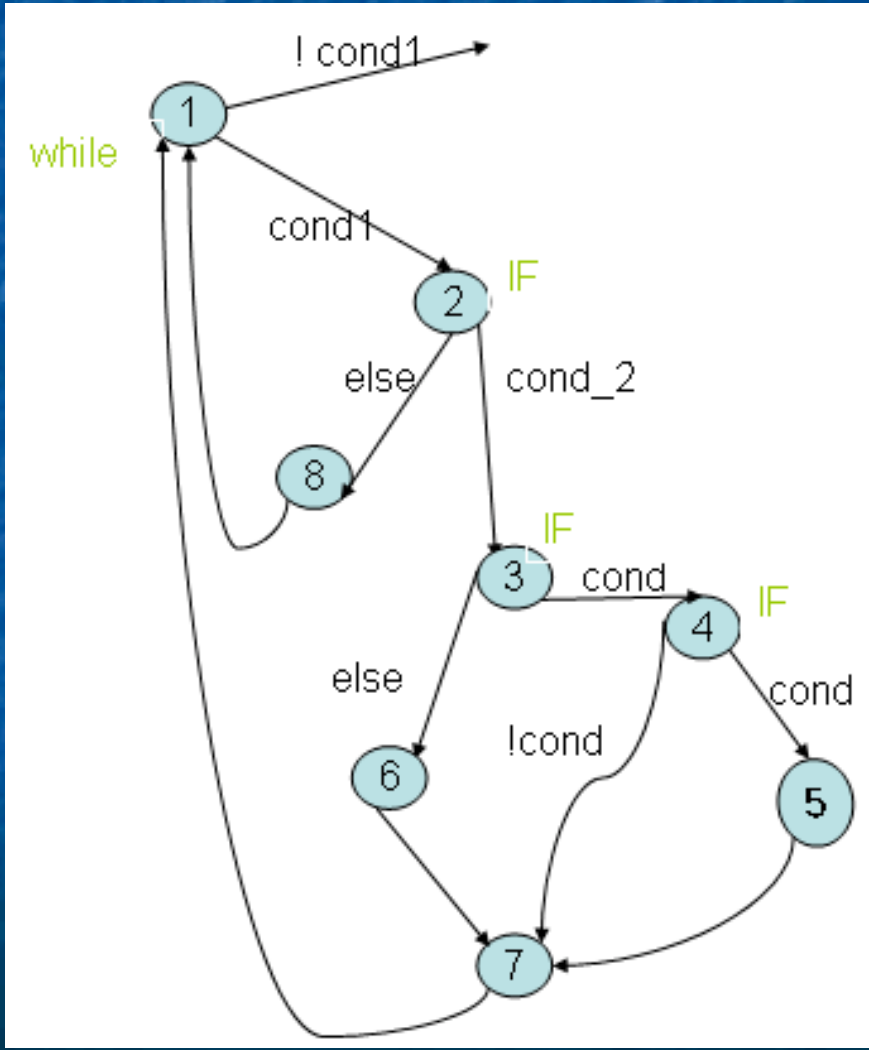
Alcuni Esempi :



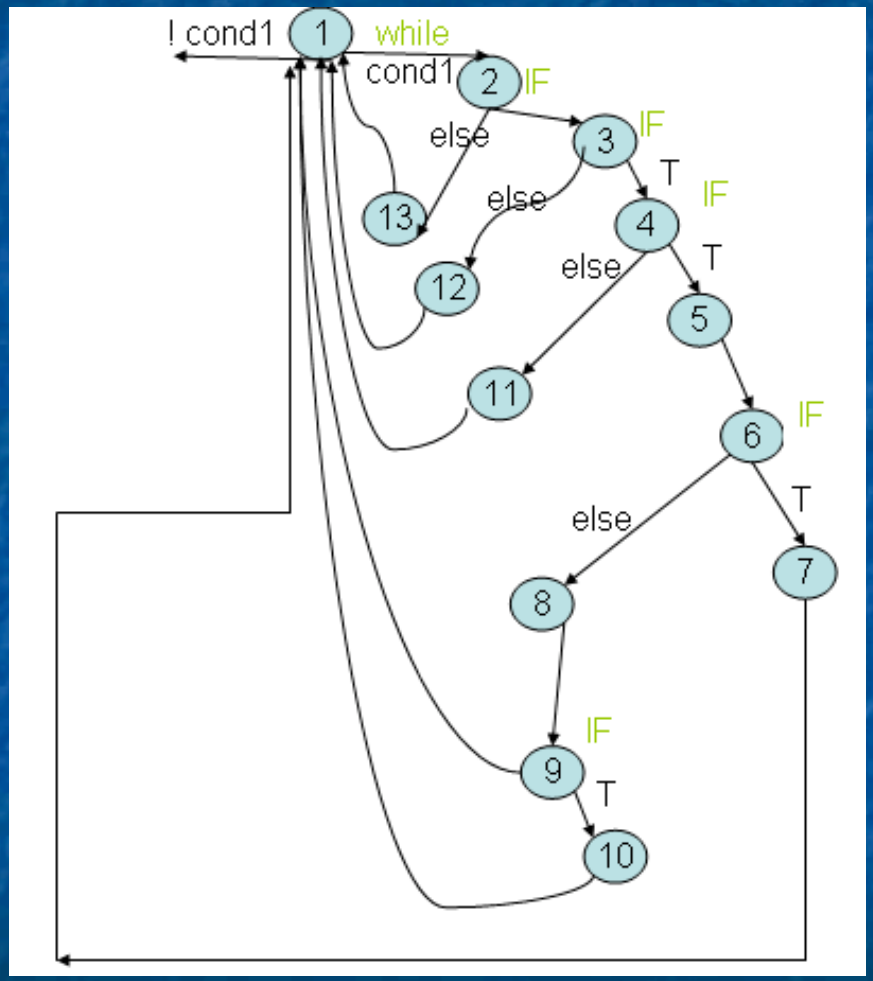


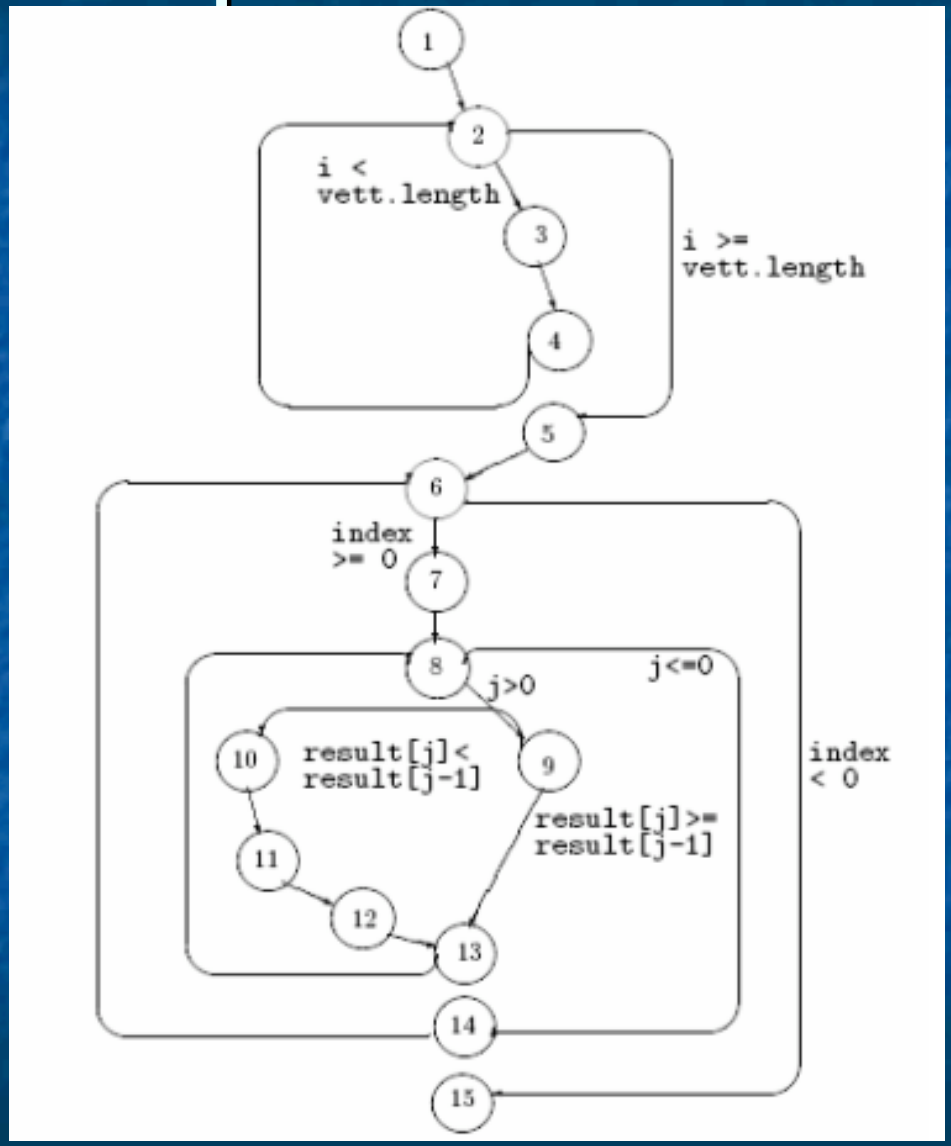
Alcuni Esempi :





Alcuni Esempi :





BubbleSort.java

Alcuni Esempi (TRANS di BubbleSort):

```
TRANS
 case
   PC = 1 : next(PC) = 2 & next(result)  =newint[vett.length] & next(i) = i & next(index) = index & next(j) = j &
next(tmp) = tmp ;
   PC = 2 : next(PC) = 3 & next(result) = result & next(i)  =0 & next(index) = index & next(j) = j & next(tmp) =
tmp ;
   PC = 3 &  i<vett.length : next(PC) = 4 & next(result) = result & next(i) = i & next(index) = index & next(j) = j &
next(tmp) = tmp ;
   PC = 3 & ! i<vett.length : next(PC) = 6 & next(result) = result & next(i) = i & next(index) = index & next(j) = j
& next(tmp) = tmp ;
   PC = 4 : next(PC) = 5 & next(result) = result & next(i) = i & next(index) = index & next(j) = j & next(tmp) =
tmp ;
   PC = 5 : next(PC) = 3 & next(result) = result & next(i) =  ++ & next(index) = index & next(j) = j & next(tmp) =
tmp ;
   PC = 6 : next(PC) = 7 & next(result) = result & next(i) = i & next(index)  =vett.length-1 & next(j) = j &
next(tmp) = tmp ;
   PC = 7 &  index>=0 : next(PC) = 8 & next(result) = result & next(i) = i & next(index) = index & next(j) = j &
next(tmp) = tmp ;
   PC = 7 & ! index>=0 : next(PC) = 16 & next(result) = result & next(i) = i & next(index) = index & next(j) = j &
next(tmp) = tmp ;
   PC = 8 : next(PC) = 9 & next(result) = result & next(i) = i & next(index) = index & next(j)  =vett.length-1 &
next(tmp) = tmp ;
   PC = 9 &  j>0 : next(PC) = 10 & next(result) = result & next(i) = i & next(index) = index & next(j) = j &
next(tmp) = tmp ;
   PC = 9 & ! j>0 : next(PC) = 15 & next(result) = result & next(i) = i & next(index) = index & next(j) = j &
next(tmp) = tmp ;

...
...
...
```

Scelte Progettuali:Pro e Contro

Abbiamo previsto di non contare le parentesi graffe chiuse "}" ;
→ non hanno una proprio dignità di PC:

PRO:

-Il grafo di flusso espresso nella sezione TRANS generata è simile a quello che potrebbe disegnare un progettista umano, ed ha una forma minimale, giacchè non prevedendo righe per codificare transizioni da/verso le parentesi graffe;

- La verifica di proprietà in NuSmv potrebbe giovare della minore grandezza della sezione TRANS, a livello di performance temporali.

CONTRO:

- Complessità implementativa maggiore per le visite del syntaxTree, per calcolare le corrette informazioni da memorizzare negli Oggetti "Nodo" (uso dei 4 algoritmi Visitor, e 2 algoritmi offline) .

- Maggior tempo di generazione del file .smv;
Se contassimo le parentesi graffe avremmo un numero minore di Visitor.

Problematiche aperte:Future Works

- **Limitazioni espressive di .SMV per le espressioni condizionali che coinvolgono indici di array; (non possono essere codificate in una unica riga di TRANS, ma ce ne deve essere una particolare per ogni possibile valore di i (es.: l'istruzione //5 del metodo MASSIMO: "result = vett[i];").**

- **Trattamento di PRE e POST-condizioni, ed ASSERTION specificate tramite commenti intermedi nel codice.**

- **Trattamento di ulteriori statement condizionali quali ForStatement, SwitchStatement , DoStatement…**

- **Studio comparativo della nostra soluzione del tool jtb-esteso, col tool Bandera:**
  Input file .java e Output codifica in SPIN e SMV (articolo scientifico allegato).

Any Question?

THANK YOU!