getDiagnosticator(behavioralAutomaton)

```
1: {Ricordandoci che exp è una lista di transizioni: le transizioni rimaste dopo il calcolo di diagnosis.
   Regex contiene l'espressione regolare della chiusura presa in esame in quel momento}
2: silentAutomaton ← generateSilentSpace(bAutomaton)
3: sStates ← states[silentAutomaton]
4: while sStates ≠ NIL do
5:    closure ← value[sStates]
6:    updateFinalClosureStates(closure)
7:    exp ← diagnosis(closure, TRUE)
8:    clRegexp ← NIL
9:    initialized ← FALSE
10:   while exp ≠ NIL do
11:      concatenateTransitionsWithSameExprScrState(sStates, exp)
12:      updateClRegex(sStates, exp, clRegex, inizialized)
13:      exp ← next[exp]
14:   end while
15:   if clRegexp ≠ NIL then
16:      delta[sState ← id[clRegexp]]
17:   end if
18:   sStates ← next[sStates]
19: end while
20: return silentAutomaton
```

updateClRegex(sStates,exp,clRegex,inizialized)

```
1: srcState ← value[exp]
2: if final[srcState] = TRUE then
3:    final[sStates] ← TRUE
4:    if initialized = TRUE then
5:       lab ← alternateLabel(clRegexp, rel[exp])
6:       clRegexp ← lab
7:    else
8:       clRegexp ← rel[exp]
9:       return initialized ← TRUE
10:   end if
11: end if
```

generateSilentSpace(bAutomaton)

1: {Lo spazio delle chiusure silenziose sarà implementato attraverso la struttura dati automaton. L'attributo value in state per la chiusura silenzionsa punterà alla chiusura relativa a quello stato. Nell'sHashMap troviamo associati gli stati iniziali di ogni chiusura con gli stati appartenenti alla chiusura: questo ci permette di collegare le chiusure con le transizioni opportune}
2: $silentAutomaton \leftarrow initializeAutomaton()$
3: $bStates \leftarrow states[behaviorlaAutomaton]$
4: **while** $bStates \neq NIL$ **do**
5: **if** $checkIfGenerateClosure(bStates)$ **then**
6: $addNewClosure(bStates, silentAutomaton, behavioralAutomaton)$
7: **end if**
8: $bStates \leftarrow next[bStates]$
9: **end while**
10: $sHashMap \leftarrow hashMapCreate()$
11: $fillSHashMapWithClosureStates(sHashMap)$
12: $bTransitions \leftarrow transitions[behavioralAutomaton]$
13: **while** $bTransitions \neq NIL$ **do**
14: **if** $obs[bTransitions] \neq NIL$ **then**
15: $lookup \leftarrow createLookupForHashMap(id[dest[bTransitions]])$
16: $item \leftarrow hashMapSearch(closureStatesHashMap, lookup)$
17: $dest \leftarrow value[item]$
18: $connectClosuresWithSameDest(silentAutomaton, closure, dest, closureHashMap)$
19: **end if**
20: $bTransitions \leftarrow next[bTransitions]$
21: **end while**
22: **return** $silentAutomaton$

checkIfGenerateClosure(state)

1: $obs \leftarrow FALSE$
2: $transitionIn \leftarrow trIn[state]$
3: **while** $transitionIn \neq NIL$ **do**
4: **if** $obs[transitionIn] \neq NIL$ **then**
5: $obs \leftarrow TRUE$
6: **break**
7: **end if**
8: $transitionIn \leftarrow next[transitionIn]$
9: **end while**
10: **return** $(obs = TRUE$ **or** $bStates = initial[behavioralAutomaton])$

addNewClosure(bStates,silentAutomaton,behavioralAutomaton)

1: $newState \leftarrow initializeState()$
2: $id[newState] \leftarrow id[bStates]$
3: $closure \leftarrow initializeAutomaton()$
4: $closeInitState \leftarrow createStateForClosure(bState, closure)$
5: $final[closeInitState] \leftarrow final[bState]$
6: $initial[closure] \leftarrow closeInitState$
7: $silentVisitDfs(bState, closure)$
8: $value[newState] \leftarrow closure$
9: $addState(silentAutomaton, newState)$
10: **if** $bStates = initial[behavioralAutomaton]$ **then**
11: $initial[silentAutomaton] \leftarrow newState$
12: **end if**

fillSHashMapWithClosureStates(sHashMap,silentAutomaton)

```
1: {closureStates contiene gli stati presenti nella chiusura silenziosa. Il lookup si crea con l'id dello
   stato nella chiusura, questo lookup punterà allo stato iniziale della chiusura}
2: sStates ← states[silentAutomaton]
3: while sStates ≠ NIL do
4:    closure ← value[sStates]
5:    closureStates ← states[closure]
6:    while closureStates ≠ NIL do
7:       lookup ← createLookUpForHashMap(id[closureStates])
8:       itemForMap ← createItem(lookup, sStates)
9:       hashMapInsert(sHashMap, itemForMap)
10:      closureStates ← next[closureStates]
11:   end while
12:   sStates ← next[sStates]
13: end while
```

concatenateTransitionsWithSameExprScrState(sStates,exp)

```
1: transitionsOut ← trOut[sStates]
2: while transitionsOut ≠ NIL do
3:    if value[exp] = value[transitionsOut] then
4:       rel[transitionsOut] ← concatenateLabels(rel[exp], rel[transitionsOut])
5:    end if
6:    tranitionsOut ← next[transitionsOut]
7: end while
```

updateFinalClosureStates(closure)

```
1: closureStates ← states[closure]
2: while cStates ≠ NIL do
3:    if exit[closureStates] = TRUE then
4:       final[closureStates] ← TRUE
5:    end if
6:    closureStates ← next[closureStates]
7: end while
```

silentVisitDfs(state,closure)

```
1: {exit verrà usato successivamente per il diagnosticatore}
2: color[state] ← GRAY
3: transitionOut ← trOut[state]
4: while transitionOut ≠ NIL do
5:    if obs[transitionsOut] ≠ NIL then
6:       tranDest ← dest[transitionOut]
7:       if color[tranDest] = WHITE then
8:          connectoTwoStates(closure, tranDest, nextState, rel[transitionOut])
9:          closeState ← createStateForClosure(tranDest, closure)
10:         final[closeState] ← final[tranDest]
11:         silentVisitDfs(tranDest, closure)
12:      else
13:         connectTwoStates(closure, tranDest, nextState, rel[transitionOut])
14:      else
15:         nextState ← value[state]
16:         exit[nextState] ← TRUE
17:      end if
18:   end if
19:   transitionOut ← next[transitionOut]
20: end while
```

connectClosuresWithSameDest(silentAutomaton,closure,dest,closureHashMap)

```
1: sStates ← states[silentAutomaton]
2: while sStates ≠ NIL do
3:    lookup ← createLookUpForHashMap(id[src[bTransitions]])
4:    item ← hashMapSearch(closureStatesHashMap, lookup)
5:    if item ≠ NIL then
6:       src ← value[item]
7:       newTransition ← connectTwoStates(silentAutomaton, src, dest)
8:       value[newTransition] ← src[bTransitions]
9:    end if
10:   sStates ← next[sStates]
11: end while
```

createStateForClosure(state,closure)

```
1: closeState ← initializeState()
2: id[closeState] ← id[state]
3: addState(closure, nextState)
4: value[state] ← cState
5: value[closeState] ← state
6: return closeState
```