```
getDiagnosticator(behavioralAutomaton)
1: {Ricordandoci che exp è una lista di transizioni: le transizioni rimaste dopo il calcolo di diagnosis.}

2: $sAutomaton \leftarrow silentSpace(bAutomaton)$
3: $sStates \leftarrow states[sAutomaton]$
4: while $sStates \neq NIL$ do
5:   $closure \leftarrow value[sStates]$
6:   $updateFinalClosureStates(closure)$
7:   $exp \leftarrow diagnosis(closure, TRUE)$
8:   $clRegexp \leftarrow NIL$
9:   $initialized \leftarrow FALSE$
10:  while $exp \neq NIL$ do
11:    $UnifyTransitionsWithSameExprScrState(sStates, exp)$
12:    $updateClRegex(sStates, exp, clRegex, inizialized)$
13:    $exp \leftarrow next[exp]$
14:  end while
15:  if $clRegexp \neq NIL$ then
16:    $delta[sState \leftarrow id[clRegexp]]$
17:  end if
18:  $sStates \leftarrow next[sStates]$
19: end while
20: return $sAutomaton$
```

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

UnifyTransitionsWithSameExprScrState(sStates,exp)

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

updateFinalClosureStates(closure)

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

SilentSpace(bAutomaton)

```
1: $sAutomaton \leftarrow addClosuresInSilentSpace(bAutomaton)$
2: $sHashMap \leftarrow hashMapCreate()$
3: $fillSHashMapWithClosureStates(sHashMap)$
4: $addTransitionsInSilentSpace(behavioralAutomaton, sHashMap, sAutomaton)$
5: return $sAutomaton$
```

addClosuresInSilentSpace(behavioralAutomaton)

```
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}
2: $silentAutomaton \leftarrow initializeAutomaton()$
3: $bStates \leftarrow tail(states[behaviorlaAutomaton])$
4: while $bStates \neq NIL$ do
5:    if $isInitialState(bStates)$ then
6:       $closure \leftarrow getSilent(bStates)$
7:       $newState \leftarrow initializeState()$
8:       $id[newState] \leftarrow id[bStates]$
9:       $addState(silentAutomaton, newState)$
10:      if $initial = TRUE$ then
11:         $initial[sAuromaton] \leftarrow newState$
12:      end if
13:      $value[newState] \leftarrow closure$
14:   end if
15:   $bStates \leftarrow prev[bStates]$
16: end while
17: return $silentAutomaton$
```

isInitialState(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 $bStates = initial[behavioralAutomaton]$)
```

fillSHashMapWithClosureStates(sHashMap)

```
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 dove è situato lo stato}
2: $sStates \leftarrow states[sAutomaton]$
3: while $sStates \neq NIL$ do
4:    $closure \leftarrow value[sStates]$
5:    $closureStates \leftarrow states[closure]$
6:    while $closureStates \neq NIL$ do
7:       $lookup \leftarrow createLookUpForHashMap(id[closureStates])$
8:       $itemForMap \leftarrow createItem(lookup, sStates)$
9:       $hashMapInsert(sHashMap, itemForMap)$
10:      $closureStates \leftarrow next[closureStates]$
11:   end while
12:   $sStates \leftarrow next[sStates]$
13: end while
```

addTransitionsInSilentSpace(behavioralAutomaton,closureStatesHashMap,sAutomaton)

```
1: {Le transizioni della rete comportamentale aggiunte sono quelle tra una chiusura silenziosa ed un altra: nell'attributo value della transizione aggiunta troviamo lo stato sorgente della transizione selezionata}
2: $bTransitions \leftarrow tail(transitions[behavioralAutomaton])$
3: while $bTransitions \neq NIL$ do
4:    if $obs[bTransitions] \neq NIL$ then
5:       $lookup \leftarrow createLookupForHashMap(id[dest[bTransitions]])$
6:       $item \leftarrow hashMapSearch(closureStatesHashMap, lookup)$
7:       $dest \leftarrow value[item]$
8:       $sStates \leftarrow tail(states[sAutomaton])$
9:       while $sStates \neq NIL$ do
10:         $lookup \leftarrow createLookUpForHashMap(id[src[bTransitions]])$
11:         $item \leftarrow hashMapSearch(closureStatesHashMap, lookup)$
12:         if $item \neq NIL$ then
13:            $src \leftarrow value[item]$
14:            $newTransition \leftarrow connectTwoStates(sAutomaton, src, dest)$
15:            $value[newTransition] \leftarrow src[bTransitions]$
16:         end if
17:         $sStates \leftarrow prev[sStates]$
18:      end while
19:   end if
20:   $bTransitions \leftarrow prev[bTransitions]$
21: end while
```

getSilent(state)

```
{negli attributi value vengono salvati i puntatori agli stati omonimi che si trovano nella chiusura di uno stato e nello spazio comportamentale}
$closure \leftarrow initializeAutomaton()$
$closeInitState \leftarrow createAndAddClosureState(state, closure)$
$initial[closure] \leftarrow closeInitState$
$silentVisitDfs(state, closure)$
return $closure$
```

silentVisitDfs(state,closure)

```
{exit verrà usato successivamente per il diagnosticatore}
$color[state] \leftarrow GRAY$
$transitionOut \leftarrow trOut[state]$
while $transitionOut \neq NIL$ do
    if $obs[transitionsOut] \neq NIL$ then
        $tranDest \leftarrow dest[transitionOut]$
        if $color[tranDest] = WHITE$ then
            $connectoTwoStates(closure, tranDest, nextState, rel[transitionOut])$
            $closeState \leftarrow createClosureState(tranDest, closure)$
            $final[closeState] \leftarrow final[tranDest]$
            $silentVisitDfs(tranDest, closure)$
        else
            $connectTwoStates(closure, tranDest, nextState, rel[transitionOut])$
        end if
    else
        $nextState \leftarrow value[tranDest]$
        $exit \leftarrow TRUE$
    end if
end while
```

createClosureState(state,closure)

```
$closeState \leftarrow initializeState()$
$id[closeState] \leftarrow id[state]$
$addState(closure, nextState)$
$value[state] \leftarrow cState$
$value[closeState] \leftarrow state$
return $closeState$
```