SilentSpace(bAutomaton)

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

isInitialState(state)

```
1: obs ← FALSE
2: transitionIn ← trIn[state]
3: while transitionIn ≠ NIL do
4:    if obs[transitionIn] ≠ NIL then
5:       obs ← TRUE
6:       break
7:    end if
8:    transitionIn ← 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 ← states[sAutomaton]
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
```

addTransitionsInSilentSpace(behavioralAutomaton,closureStatesHashMap,sAutomaton)

```
1: bTransitions ← tail(transitions[behavioralAutomaton])
2: while bTransitions ≠ NIL do
3:    if obs[bTransitions] ≠ NIL then
4:       lookup ← createLookupForHashMap(id[dest[bTransitions]])
5:       item ← hashMapSearch(closureStatesHashMap, lookup)
6:       dest ← value[item]
7:       sStates ← tail(states[sAutomaton])
8:       while sStates ≠ NIL do
9:          lookup ← createLookUpForHashMap(id[src[bTransitions]])
10:         item ← hashMapSearch(closureStatesHashMap, lookup)
11:         if item ≠ NIL then
12:            src ← value[item]
13:            connectTwoStates(sAutomaton, src, dest)
14:         end if
15:         sStates ← prev[sStates]
16:      end while
17:   end if
18:   bTransitions ← prev[bTransitions]
19: end while
```

getSilent(state)

```
1: {negli attributi value vengono salvati i puntatori agli stati omonimi che si trovano nella chiusura
   di uno stato e nello spazio comportamentale}
2: closure ← initializeAutomaton()
3: closeInitState ← createAndAddClosureState(state, closure)
4: initial[closure] ← closeInitState
5: silentVisitDfs(state, closure)
6: return closure
```

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 ← createClosureState(tranDest, closure)
10:         final[closeState] ← final[tranDest]
11:         silentVisitDfs(tranDest, closure)
12:      else
13:         connectTwoStates(closure, tranDest, nextState, rel[transitionOut])
14:      end if
15:   else
16:      nextState ← value[tranDest]
17:      exit ← TRUE
18:   end if
19: end while
```

createClosureState(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
```