diagnosis(netBs,split)

1: {considerando netBs come una rete comportamentale}
2: {Definiamo gli stati "One To One" come quelli con una sola transizione entrante ed uscente e consideriamo tutti gli altri come "Many To Many", due transizioni vengono definite parallele se hanno in comune lo stato di partenza e di arrivo}
3: {la variabile *split* è booleana e serve ad inciare se l'algoritmo che stiamo usando è EspressioneRegolare o EspressioniRegolari}
4: $automaton \leftarrow sutomatons[netBs]$
5: $replaceInitialState(netBs)$
6: $finalState \leftarrow replaceEndStates(netBS)$
7: $tran \leftarrow transitions[automaton]$
8: $states \leftarrow states[automaton]$
9: **while** $(split = FALSE$ **and** $next[tran] \neq NIL)$ **or** $(split = FALSE$ **and** $(next[next[states]] \neq NIL$ **or** $multipleTr(automaton = TRUE)))$ **do**
10: $replaceOneToOneStates(netBs)$
11: $unifyParallelTransitions(netBs)$
12: $replaceManyToManyStates(netBs, finalState)$
13: **if** $tran = NIL$ **then**
14: $error()$
15: **end if**
16: **end while**
17: **return** *tran*

connectTwoStates(network,source,destination,relevanceLabel)

1: $transition \leftarrow initialiseTranstition()$
2: $src[transition] \leftarrow source$
3: $dest[tranEnd] \leftarrow destination$
4: $rel[transition] \leftarrow relevanceLabel$
5: $addTransition(transition, network)$
6: **return** *transition*

replaceInitialState(network)

1: $automaton \leftarrow automatons[network]$
2: $initState \leftarrow initialiseState()$
3: $addState(init, network)$
4: $stateToStart \leftarrow initial[automaton]$
5: $connectTwoStates(network, stateToStart, initState, NIL)$
6: $initial[automaton] \leftarrow initState$

replaceEndStates(network,split)

```
1: automaton ← automatons[network]
2: endState ← initialiseState()
3: totalState ← states[automaton]
4: while totalState ≠ NIL do
5:    if final[totalState] = TRUE then
6:       connectTwoStates(network, totalState, endState, NIL)
7:       final[totalState] ← FALSE
8:    end if
9:    totalState ← next[totalState]
10: end while
11: final[endState] ← TRUE
12: return endState
```

createNewRelevanceLabel(newId)

```
1: newLabel ← initialiseLabel()
2: id[newLabel] ← newId
3: labelType[newLabel] ← RELEVANCE
4: return newLabel
```

replaceOneToOneStates(network)

```
1: automaton ← automatons[network]
2: totalState ← states[network]
3: while totalState ≠ NIL do
4:    transitionIn ← trIn[totalState]
5:    transitionOut ← trOut[totalState]
6:    if transitionIn ≠ NIL and next[transitionIn] = NIL and transitionOut ≠ NIL and
      next[transitionOut] = NIL then
7:       labelIn ← rel[transitionIn]
8:       labelOut ← rel[transitionOut]
9:       newId ← oneToOneRelation(id[LabelIn], id[LabelOut])
10:      newLabel ← createNewRelevanceLabel(newId)
11:      tran ← connectTwoStates(network, transitionIn, transitionOut, newLabel)
12:      if split = TRUE then
13:         if final[totalState] = TRUE then
14:            value[tran] ← value[totalState]
15:         else
               value[tran] ← value[transitionOut]
16:         end if
17:      end if
18:      removeTheState(automaton, totalState)
19:   end if
20:   totalState ← next[totalState]
21: end while
```

unifyParallelTransitions(network)

1: {lookup contiene la chiave usata per mappare la transizione all'interno dell'hashmap, una stringa contenente l'identificativo dello stato sorgente e lo stato di destinazione}
2: $automaton \leftarrow automatons[network]$
3: $trHashMap \leftarrow createHashmap()$
4: $ids \leftarrow createList()$
5: $tran \leftarrow transitions[automaton]$
6: **while** $tran \neq NIL$ **do**
7: $lookup \leftarrow createLookUpForHashMap(trHashMap, lookup, value[tran])$
8: $item \leftarrow hashmapSearch(trHashmap, lookup)$
9: **if** $item = NIL$ **then**
10: $itemForMap \leftarrow createItem(lookup, tran)$
11: $hashMapInsert(trHashmap, itemForMap)$
12: **else**
13: $parallelTransition \leftarrow value[item]$
14: $label1 \leftarrow rel[parallelTransition]$
15: $label2 \leftarrow rel[tran]$
16: **if** $split \neq TRUE$ **then**
17: $newId \leftarrow parallelRelation(id[label1], id[label2])$
18: **else**
$newId \leftarrow parallelRelationWithP(id[label1], id[label2], value[tran])$
19: **end if**
20: $rel[parallelTransition] \leftarrow createNewRelevanceLabel(newId)$
21: $removeTransition(automaton, tran)$
22: **end if**
23: $tran \leftarrow next[tran]$
24: **end while**

replaceManyToManyStates(network)

1: $automaton \leftarrow automatons[network]$
2: $totalState \leftarrow states[automaton]$
3: **while** $totalState \neq NIL$ **do**
4: **if** $initial[aut] \neq totalState$ **and** $final[automaton] \neq totalState$ **then**
5: $autoTransitionRel \leftarrow removeAutoTansition(totalState)$
6: $unifyAllTransitionsInState(totalState, autoTransitionRel)$
7: **end if**
8: $totalState \leftarrow next[totalState]$
9: **end while**

unifyAllTransitionsInState(state)

```
1:  transitionIn ← trIn[state]
2:  transitionOut ← trOut[state]
3:  while transitionIn ≠ NIL do
4:     while transitionOut ≠ NIL do
5:        labelIn ← rel[transitionIn]
6:        labelOut ← rel[transitionOut]
7:        newId ← manyToManyRel(id[labelIn], id[labelOut], autoTransitionRel)
8:        newLabel ← createNewRelevanceLabel(newId)
9:        newTran ← connectTwoStates(network, src[transitionIn], dest[transitionOut], newLabel)
10:       if split = TRUE then
11:          if final[state] and dest[newTran] = finalState and value[transitionOut] ≠ NIL then
12:             value[newTran] ← value[state]
13:          else
                value[tran] ← value[transitionOut]
14:          end if
15:       end if
16:       transitionOut ← next[transitionOut]
17:       removeTheState(automaton, totalState)
18:    end while
19:    transitionIn ← next[transitionIn]
20: end while
```

removeAutoTransition(state)

```
1:  transitionIn ← trIn[totalState]
2:  autoTransitionRel ← NIL
3:  while transitionIn ≠ NIL do
4:     if src[transitionIn] = dest[transitionIn] and rel[transitionIn] = NIL then
5:        labelRel ← rel[transitionIn]
6:        autoTransitionRel ← id[labelRel]
7:        removeTransition(automaton, transitionIn)
8:        return autoTransitionRel
9:     end if
10:    transitionIn ← next[transitionIn]
11: end while
```

removeTheState(automaton,state)

1: {Prima di eliminare lo stato dalla lista dell'automa bisogna liberarsi di tutte le transizioni in entrate e in uscita dallo stato}
2: $transitionIn \leftarrow trIn[state]$
3: **while** $transitionIn \neq NIL$ **do**
4: $\quad tran \leftarrow transitionIn$
5: $\quad removeTransition(automaton, tran)$
6: $\quad transitionIn \leftarrow next[transitionIn]$
7: **end while**
8: $transitionOut \leftarrow trOut[state]$
9: **while** $transitionOut \neq NIL$ **do**
10: $\quad tran \leftarrow transitionOut$
11: $\quad removeTransition(automaton, tran)$
12: $\quad transitionOut \leftarrow next[transitionOut]$
13: **end while**
14: $sttrHashMap \leftarrow sttrHashMap[automaton]$
15: $item \leftarrow statePuntactorSearch(state, sttrHashMap)$
16: $listRemove(states[automaton], item)$

removeTransition(automaton,transition)

1: {Per eliminare la transizione bisogna rimuoverla dalla lista dello stato in entrata e in uscita e dall'automa dove risiede}
2: $sttrHashMap \leftarrow sttrHashMap[automaton]$
3: $item \leftarrow transitionOutPuntactorSearch(transition, sttrHashMap)$
4: $srcState \leftarrow src[transition]$
5: $listRemove(trOut[scrState], item)$
6: $item \leftarrow transitionInPuntactorSearch(transition, sttrHashMap)$
7: $destState \leftarrow dest[transition]$
8: $listRemove(trIn[destState], item)$
9: $item \leftarrow transitionPuntactorSearch(transition, sttrHashMap)$
10: $listRemove(transitions[automaton], item)$