diagnosis(automaton,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: replaceInitialState(automaton)
5: finalState ← replaceEndStates(automaton)
6: tran ← transitions[automaton]
7: states ← states[automaton]
8: while (split = FALSE and next[tran] ≠ NIL) or (split = TRUE and (next[next[states]] ≠ NIL or multipleTr(automaton = TRUE))) do
9:    replaceOneToOneStates(automaton, split)
10:   unifyParallelTransitions(automaton, split)
11:   replaceManyToManyStates(automaton, finalState)
12:   if tran = NIL then
13:      error()
14:   end if
15: end while
16: return tran
```

multipleTr(automaton)

```
1: subHashMap ← initializeHashMap()
2: transitions ← transitions[automaton]
3: emptyTr ← FALSE
4: while transitions ≠ NIL do
5:    if value[transitions ≠ NIL] then
6:       if emptyTr = TRUE then
7:          return TRUE
8:       else
9:          emptyTr ← TRUE
10:      end if
11:   else
12:      state ← value[transitions]
13:      lookup ← createLookUpForHashMap(subHashMap, id[state])
14:      item ← hashmapSearch(subHashMap, lookup)
15:      if item ≠ NIL then
16:         return TRUE
17:      else
18:         hashmapInsert(subHashMap, id[state])
19:      end if
20:   end if
21:   transitions ← next[transitions]
22: end while
```

connectTwoStates(automaton,source,destination,relevanceLabel)

```
1: transition ← initialiseTranstition()
2: src[transition] ← source
3: dest[tranEnd] ← destination
4: rel[transition] ← relevanceLabel
5: addTransition(transition, automaton)
6: return transition
```

replaceInitialState(automaton)

```
1: initState ← initialiseState()
2: addState(init, automaton)
3: stateToStart ← initial[automaton]
4: connectTwoStates(automaton, stateToStart, initState, NIL)
5: initial[automaton] ← initState
```

replaceEndStates(automaton,split)

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

createNewRelevanceLabel(newId)

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

replaceOneToOneStates(automaton,split)

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

```
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: trHashMap ← createHashmap()
3: ids ← createList()
4: tran ← transitions[automaton]
5: while tran ≠ NIL do
6:    lookup ← createLookUpForHashMap(trHashMap, lookup, value[tran])
7:    item ← hashmapSearch(trHashmap, lookup)
8:    if item = NIL then
9:       itemForMap ← createItem(lookup, tran)
10:      hashMapInsert(trHashmap, itemForMap)
11:   else
12:      parallelTransition ← value[item]
13:      label1 ← rel[parallelTransition]
14:      label2 ← rel[tran]
15:      if split ≠ TRUE then
16:         newId ← parallelRelation(id[label1], id[label2])
17:      else
18:         newId ← parallelRelationWithP(id[label1], id[label2], value[tran])
19:      end if
20:      rel[parallelTransition] ← createNewRelevanceLabel(newId)
21:      removeTransition(automaton, tran)
22:   end if
23:   tran ← next[tran]
24: end while
```

replaceManyToManyStates(automaton,split)

```
1: totalState ← states[automaton]
2: while totalState ≠ NIL do
3:    if initial[aut] ≠ totalState and final[automaton] ≠ totalState then
4:       autoTransitionRel ← removeAutoTansition(totalState)
5:       unifyAllTransitionsInState(totalState, autoTransitionRel, split)
6:    end if
7:    totalState ← next[totalState]
8: end while
```

unifyAllTransitionsInState(state,autoTransitionRel,split)

```
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(automaton, 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
14:            value[tran] ← value[transitionOut]
15:         end if
16:      end if
17:      transitionOut ← next[transitionOut]
18:      removeTheState(automaton, totalState)
19:   end while
20:   transitionIn ← next[transitionIn]
21: 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
```