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: $automaton \leftarrow sutomatons[netBs]$
4: $replaceInitialState(netBs)$
5: $replaceEndStates(netBS)$
6: $tran \leftarrow transitions[automaton]$
7: $states \leftarrow states[automaton]$
8: **while** $(split = FALSE$ **and** $next[tran] \neq NIL)$ **or** $(split = FALSE$ **and** $(next[next[states]] \neq NIL)$ **or** $multipleTr(automaton = TRUE))$ **do**
9: $replaceOneToOneStates(netBs)$
10: $unifyParallelTransitions(netBs)$
11: $replaceManyToManyStates(netBs)$
12: **if** $tran = NIL$ **then**
13: $error()$
14: **end if**
15: **end while**
16: **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: $addTransition(transition, network)$

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)

```
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
```

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:      connectTwoStates(network, transitionIn, transitionOut, newLabel)
12:      removeTheState(automaton, totalState)
13:   end if
14:   totalState ← next[totalState]
15: 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 ← automatons[network]
3: transitionHashMap ← createHashmap()
4: ids ← createList()
5: tran ← transitions[automaton]
6: while tran ≠ NIL do
7:    lookup ← createLookUpForHashmap(tran)
8:    item ← hashmapSearch(transitionHashmap, lookup)
9:    if item = NIL then
10:      itemForMap ← createItem(lookup, tran)
11:      hashMapInsert(transitionHashmap, itemForMap)
12:   else
13:      parallelTransition ← value[item]
14:      label1 ← rel[parallelTransition]
15:      label2 ← rel[tran]
16:      newId ← parallelRelation(id[label1], id[label2])
17:      rel[parallelTransition] ← createNewRelevanceLabel(newId)
18:      removeTransition(automaton, tran)
19:   end if
20:   tran ← next[tran]
21: end while
```

replaceManyToManyStates(network)

```
1: {questa funzione riassume le righe 21-31 dello pseudocodice nella consegna}
2: automaton ← automatons[network]
3: totalState ← states[automaton]
4: while totalState ≠ NIL do
5:    if initial[aut] ≠ totalState and final[automaton] ≠ totalState then
6:       autoTransitionRel ← removeAutoTansition(totalState)
7:       unifyAllTransitionsInState(totalState, autoTransitionRel)
8:    end if
9:    totalState ← next[totalState]
10: 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:       connectTwoStates(network, src[transitionIn], dest[transitionOut], newLabel)
10:      transitionOut ← next[transitionOut]
11:      removeTheState(automaton, totalState)
12:   end while
13:   transitionIn ← next[transitionIn]
14: 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: automaton
3: transitionIn ← trIn[state]
4: while transitionIn ≠ NIL do
5:    tran ← transitionIn
6:    removeTransition(tran)
7:    transitionIn ← next[transitionIn]
8: end while
9: transitionOut ← trOut[state]
10: while transitionOut ≠ NIL do
11:   tran ← transitionOut
12:   removeTransition(tran)
13:   transitionOut ← next[transitionOut]
14: end while
15: sttrHashMap ← sttrHashMap[automaton]
16: item ← statePuntactorSearch(state, sttrHashMap)
17: listRemove(states[automaton], item)
```

removeTransition(transition,state)

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