diagnosis(netBs)

```
$automaton \leftarrow sutomatons[netBs]$
{considerando netBs come una rete comportamentale}
{automaton è l'unico elemento della lista}
{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}
{riporta l'unica transizione rimasta nell'automa}
$replaceInitialState(netBs)$
$replaceEndStates(netBS)$
$tran \leftarrow transitions[automaton]$
while $next[tran] \neq NIL$ do
  $replaceOneToOneStates(netBs)$
  $unifyParallelTransitions(netBs)$
  $replaceManyToManyStates(netBs)$
  if $tran = NIL$ then
    $error()$
  end if
end while
return $tran$
```

connectTwoStates(network,source,destination,relevanceLabel)

```
$transition \leftarrow initialiseTranstition()$
$src[transition] \leftarrow source$
$dest[tranEnd] \leftarrow destination$
$rel[transition] \leftarrow relevanceLabel$
$addTransition(transition, network)$
```

replaceInitialState(network)

```
$automaton \leftarrow automatons[network]$
$initState \leftarrow initialiseState()$
$addState(init, network)$
$stateToStart \leftarrow initial[automaton]$
$connectTwoStates(network, stateToStart, initState, NIL)$
$initial[automaton] \leftarrow initState$
```

replaceEndStates(network)

```
$automaton \leftarrow automatons[network]$
$endState \leftarrow initialiseState()$
$totalState \leftarrow states[automaton]$
while $totalState \neq NIL$ do
    if $final[totalState] = TRUE$ then
        $connectTwoStates(network, totalState, endState, NIL)$
        $final[totalState] \leftarrow FALSE$
    end if
    $totalState \leftarrow next[totalState]$
end while
$final[endState] \leftarrow TRUE$
```

replaceOneToOneStates(network)

```
{questa funzione fa riferimento alle righe 16-17 dello pseudocodice nella consegna}
$automaton \leftarrow automatons[network]$
$totalState \leftarrow states[network]$
while $totalState \neq NIL$ do
    $transitionIn \leftarrow trIn[totalState]$
    $transitionOut \leftarrow trOut[totalState]$
    if $transitionIn \neq NIL$ and $next[transitionIn] = NIL$ and $transitionOut \neq NIL$ and $next[transitionOut] = NIL$ then
        $labelIn \leftarrow rel[transitionIn]$
        $labelOut \leftarrow rel[transitionOut]$
        $newId \leftarrow oneToOneRelation(id[LabelIn], id[LabelOut])$
        $newLabel \leftarrow initialiseLabel()$
        $id[newLabel] \leftarrow newId$
        $labelType[newLabel] \leftarrow RELEVANCE$
        $connectTwoStates(network, transitionIn, transitionOut, newLabel)$
        $removeTheState(network, totalState)$
    end if
    $totalState \leftarrow next[totalState]$
end while
```

unifyParallelTransitions(network)

```
{questa funzione riassume le righe 18-19 dello pseudocodice nella consegna . 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}
$automaton \leftarrow automatons[network]$
$transitionHashMap \leftarrow createHashmap()$
$ids \leftarrow createList()$
$tran \leftarrow transitions[automaton]$
while $tran \neq NIL$ do
    $lookup \leftarrow createLookUpForHashmap(tran)$
    $item \leftarrow hashmapSearch(transitionHashmap, lookup)$
    if $item = NIL$ then
        $itemForMap \leftarrow createItem(lookup, tran)$
        $hashMapInsert(transitionHashmap, itemForMap)$
    else
        $parallelTransition \leftarrow value[item]$
        $label1 \leftarrow rel[parallelTransition]$
        $label2 \leftarrow rel[tran]$
        $newId \leftarrow parallelRelation(id[label1], id[label2])$
        $id[label1] \leftarrow newId$
        $rel[parallelTransition] \leftarrow label1$
        $removeTransition(network, tran)$
    end if
    $tran \leftarrow next[tran]$
end while
```

replaceManyToManyStates(network)

```
{questa funzione riassume le righe 21-31 dello pseudocodice nella consegna}
$automaton \leftarrow automatons[network]$
$totalState \leftarrow states[automaton]$
while $totalState \neq NIL$ do
    if $initial[aut] \neq totalState$ and $final[automaton] \neq totalState$ then
        $autoTransitionRel \leftarrow removeAutoTansition(totalState)$
        $unifyAllTransitionsInState(totalState, autoTransitionRel)$
    end if
    $totalState \leftarrow next[totalState]$
end while
```

unifyAllTransitionsInState(state)

```
$transitionIn \leftarrow trIn[state]$
$transitionOut \leftarrow trOut[state]$
while $transitionIn \neq NIL$ do
    while $transitionOut \neq NIL$ do
        $labelIn \leftarrow rel[transitionIn]$
        $labelOut \leftarrow rel[transitionOut]$
        $newId \leftarrow manyToManyRel(id[labelIn], id[labelOut], autoTransitionRel)$
        $newLabel \leftarrow labelInitialise()$
        $id[newLabel] \leftarrow newId$
        $connectTwoStates(network, src[transitionIn], dest[transitionOut], newLabel)$
        $removeTheState(networl, totalState)$
        $transitionOut \leftarrow next[transitionOut]$
    end while
    $transitionIn \leftarrow next[transitionIn]$
end while
$removeState(network, state)$
```

removeAutoTransition(state)

```
$transitionIn \leftarrow trIn[totalState]$
$autoTransitionRel \leftarrow NIL$
while $transitionIn \neq NIL$ do
    if $src[transitionIn] = dest[transitionIn]$ and $rel[transitionIn] = NIL$ then
        $labelRel \leftarrow rel[transitionIn]$
        $autoTransitionRel \leftarrow id[labelRel]$
        $removeTransition(network, transitionIn)$
        return $autoTransitionRel$
    end if
    $transitionIn \leftarrow next[transitionIn]$
end while
```