step(currentBsState)

```
1: currentContext ← context[currentBsState]
2: for each state in states[currentContext] do
3:    tran ← trOut[state]
4:    while tran ≠ NIL do
5:       if isEventPresent(currentContext, tran) and isBufferFree(currentContext, tran) then
6:          newContext ← createNewContext(currentContex, tran)
7:          newTransition ← createNewTransition(currentContext, tran)
8:          item ← contextSearch(newContext, ctHashMap)
9:          if item ≠ NIL then
10:            destinationBsState ← subValue[item]
11:            dest[newTransition] ← destinationBsState
12:         else
13:            createNewState(newContext, tran)
14:            step(destinationBsState)
15:         end if
16:      end if
17:   end while
18: end for
```

takeEventFromBuffer(context,action)

```
1: l ← link[action]
2: pos ← index[l]
3: return buffer[context][pos]
```

isEventPresent(context,transition)

```
1: actionRequest ← actIn[transition]
2: eventBuffer ← takeEventFromBuffer(context, actionRequest)
3: eventRequest ← event[actionRequest]
4: return (actionRequest = NIL or eventBuffer = eventRequest)
```

isBufferFree(context,transition)

```
1: actionProduced ← actOut(transition)
2: while actionProduced ≠ NIL do
3:    eventBuffer ← takeEventFromBuffer(context, actionProduced)
4:    if eventBuffer ≠ NIL then
5:       return FALSE
6:    end if
7:    actionProduced ← next[actionProduced]
8: end while
9: return TRUE
```

createNewContext(context,transition)

```
1: newContext ← initializeContext()
2: state ← dest[transition]
3: actionRequest ← actIn[transition]
4: eventRequest ← event[actionRequest]
5: if eventRequest ≠ NIL then
6:    eventBuffer ← NIL
7: end if
8: actionProduced ← actOut[tran]
9: while actionProduced ≠ NIL do
10:   l2 ← link[actionProduced]
11:   pos2 ← index[l2]
12:   buffer[newContext][pos2] ← actionProduced
13:   actionProduced ← next[actionProduced]
14: end while
15: return newContext
```

createNewTransition(context,transition)

```
1: newTransition ← initializeTransition()
2: obs[newTransition] ← obs[transition]
3: rel[newTransition] ← rel[transition]
4: src[newTransition] ← createSource(context)
5: netBs ← addTransition(newTransition)
6: return newTransition
```

createNewState(context,transition)

```
1: destinationBsState ← initializeState()
2: context[destinationBsState] ← context
3: if isFinal(context) then
4:    final[destinationBsState] ← TRUE
5: else
6:    finale[destinationBsState] ← FALSE
7: end if
8: netBs ← addState(destinationBsState)
9: dest[transition] ← destinationBsState
10: addContextToHashMap(context)
```

dfsVisit(state)

```
1: color[source] ← GRAY
2: transitionsIncoming ← trIn[state]
3: while transitionsIncoming ≠ NIL do
4:    stateSource ← scr[transitionsIncoming]
5:    if color[stateSource] = WHITE then
6:       dfsVisit[stateSource]
7:    end if
8:    transitionsIncoming ← nect[transitionsIncoming]
9: end while
10: color[state] ← BLACK
```

prune(network)

```
1: {consideriamo solo reti con un solo automa, infatti il pruning avviene solo sulle
2: network generate dal sistema}
3: autom ← automatons[network]
4: totalStat ← states[autom] {La lista autom avrà un solo elemento}
5: while totalStat ≠ NIL do
6:    if final[totalState] then
7:       dfsVisit(totalState)
8:    end if
9:    totalState ← next[totalState]
10: end while
11: while totalStat ≠ NIL do
12:    if color[totalState] = WHITE then
13:       removeTheState(network, totalState)
14:    end if
15:    totalState ← next[totalState]
16: end while
```