Subset Algorithm

→ we have 3 main steps
- get start state
- get states T', that are reachable by certain char → a & all its epsilons
- set a transition between these super Nodes.

حل بالك لو طلعت نفس الـ Node قبل كدا، متعملش node جديدة

→ get start state → we should implement a function that takes any node, and then append all its neighbours that can be achieved via epsilon else

→ def epsilon_closure (state) # DFS also, superNode
  if [visited[state]] # visited before.
    return
  vis[state] = true
  for trans in state.transitions:
    if (trans == 'ε')
      superNode.states.append(dest)
      epsilon_closure(dest, superNode)
→ vis[state] = False

Step2.

Create state T that is reachable via σ & a, where a is any character.

ABCPHI

def step2 (current SuperNode, a) → SuperNode:

1. new superNode
2. iterate over each state in the current superNode
   1. ~~SuperNode~~
   1. epsilon closure (state, new SuperNode)
   2. for trans in state.transitions:
      1. if (trans == a)
         new SuperNode. states. append (dest)
3. return new SuperNode

epsilon - closure (dest)

Step3: Add Transition (source SuperNode, dest SN, a)

source SuperNode. transitions. append ({a: dest SN})
a:

superNodes → states → list [state]
→ isTerminating → bool
→ Transition → map (key → letter, value → superNode)

Main logic

→ use know the initial state

1. initState = SuperNode (initial state, False, {})
2. epsilon closure (initial state, initState)
3. ~~for~~

Main logic:

→ we know the initial state → A

1. initial SuperNode = SuperNode(A, False, {})

→ get its closure

2. epsilon_closure(A,)
3. create list of superNodes = [initialSuperNode]
4. for sn in superNodes:
    1. for transition in transitions
        1. newSuper = step2(sn, transition)
        2. AddTransition(sn, newSuper, transition)
        3. superNodes.append(newSuper)

هنا هيبقى فيه بعده ان لو طلعت نفس ال super state كذا مرة محتاجين نـ handle الحوار ده