```
procedure NFAtoDFA( N an NFA )
Let T[row][col] be an empty transition table defining
D. T[row][·] is uniquely identified by a set of
states from N, each T[·][col] uniquely identifies
a character c ∈ Σ.

  let L be an empty stack
  let A be the set of accepting states for N
  let i be the starting state of N
  B ← FollowLamda( {i} )
  initialize row T[B][·]
  mark T[B][·] as the starting state of D
  if ( A ⋂ B ≠ ∅ ) then (
    mark T[B][·] as an accepting state of D
  )
  push B onto L
  repeat (
    S ← pop L
    foreach ( c ∈ Σ ) do (
      R ← FollowLambda(FollowChar(S,c))
      T[S][c] ← R
      if ( |R| > 0 AND T[R][·] does not exist ) then (
        initialize row T[R][·]
        if ( A ⋂ R ≠ ∅ ) then (
          mark T[R][·] as an accepting state of D
        )
        push R onto L
      )
    )
  ) while ( |L| > 0 )

  T now defines a DFA D equivalent to N
```

```
procedure FollowLambda( S a ⊆ of NFA N states )
returns the set of NFA states encountered by
recursively following only λ transitions
from states in S

  Let M be an empty stack
  foreach ( state t ∈ S ) push t onto M
  while ( |M| > 0 ) do (
    t ← pop M
    foreach ( λ transition from t to state q ) do (
      if ( q ∉ S ) then (
        add q to S
        push q onto M
      )
    )
  )
  return S
```

```
procedure FollowChar( S a ⊆ of NFA N states, c ∈ Σ )
returns the set of NFA states obtained from following
all c transitions from states in S

  Let F be an empty set
  foreach ( state t ∈ S ) do (
    foreach ( c transition from t to state q ) do (
      add q to F
    )
  )
  return F
```