```
procedure MergeStates( accepts DFA D defined by
    transition table T[·][·] )
returns a potentially new T[·][·]

T[row][·] uniquely identifies one state of D, and
each T[r][c] identifies the unique transition from
state r to state T[r][c] on input character c ∈ Σ.

let M be an empty set
let L be an empty stack
push ({accepting states of D},Σ) onto L
push ({non-accepting states of D},Σ) onto L
repeat (
  S,C ← pop L
  remove an element c from C
  Partition states s in S by T[s][c] into sets
    X_1,X_2,X_3,...,X_k
  foreach ( X_i of X_1,X_2,X_3,...,X_k with |X_i| > 1 ) do (
    if ( C = ∅ ) then (
      add X_i to M
    ) else (
      push (X_i,C) onto L
    )
  )
) while ( |L| > 0 )
foreach ( S ∈ M ) do (
  merge rows of T[·][·] identified by states in S,
    fixing up transitions to these states as well!
  if ( starting state of D ∈ S ) then (
    mark the newly merged row as the
    starting state of D
  )
)
return T[·][·]
```

```
Let DFA D be defined by transition table T[·][·].
T[row][·] uniquely identifies one state of D, and
each T[r][c] identifies the unique transition from
state r to state T[r][c] on input character c ∈ Σ.
repeat (
  T′ ← MergeStates( T )
  if ( |T| = |T′| ) then (
    break loop
  ) else (
    T ← T′
  )
)
T′[·][·] is now a well (near?) optimized DFA equivalent
  to D with a reasonable number of effective states.
(Dead or unreachable states may still exist.)
```