## Why Don't We Use NFAs during Scanning?


i. How can we represent an NFA in computer memory?

## Why Don't We Use NFAs during Scanning?


i. How can we represent an NFA in computer memory?
An $n \times n$ Boolean matrix for $\lambda \mathrm{s}$, and a state $\times c \in \Sigma$ "transition table" whose cells contain what?

## Why Don't We Use NFAs during Scanning?


i. How can we represent an NFA in computer memory?
An $n \times n$ Boolean matrix for $\lambda \mathrm{s}$, and a state $\times c \in \Sigma$ "transition table" whose cells contain what?
ii. While matching a character sequence to an NFA, what type of data structure must be used to remember where in the NFA we are?

## Why Don't We Use NFAs during Scanning?


i. How can we represent an NFA in computer memory? An $n \times n$ Boolean matrix for $\lambda \mathrm{s}$, and a state $\times c \in \Sigma$ "transition table" whose cells contain what?
ii. While matching a character sequence to an NFA, what type of data structure must be used to remember where in the NFA we are?
iii. Can we represent a DFA more efficiently?
iv. What data structure is required to remember DFA matching state?

## Why Don't We Use NFAs during Scanning?

An example of the simple /* C/C++ comment */ RE converted to an NFA using automated tools (in fact, all of which you will build in this course!)...

c++comment-automated.pdf

## NFA to DFA Algorithm

 initialization$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

Stack $L$ <empty>

```
procedure NFAtoDFA( N}\mathrm{ 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 }\in\Sigma\mathrm{ .
let L be an empty stack
let A be the set of accepting states for N
let i be the starting state of N
```



# NFA to DFA Algorithm 

Transition Table $T$
FollowLambda

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

Stack $L$ <empty>

```
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\in\Sigma.
    let }L\mathrm{ be an empty stack
    let }A\mathrm{ be the set of accepting states for N
    let i be the starting state of N
    B\leftarrowFollowLamda( {i} )
```



# NFA to DFA Algorithm 

Transition Table $T$
FollowLambda

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

Stack $L$ <empty>

```
procedure FollowLambda( S a \subseteq of NFA N states )
returns the set of NFA states encountered by
recursively following only }\lambda\mathrm{ transitions
from states in S
Let }M\mathrm{ be an empty stack
foreach ( state t\inS) push t onto M
while ( }|M|>0) do 
    t\leftarrow pop }
    foreach ( \lambda transition from t to state q) do (
        if ( }q\not\inS)\mathrm{ then (
            add q to S
            push q onto M
        )
    )
)
return S
```



# NFA to DFA Algorithm <br> <br> FollowLambda 

 <br> <br> FollowLambda}

| is Start | is Accept | State | a | b | c | d | e | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ |  |  |  |  |  |  |

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

Stack $L<\{0,4\}>$

$$
B=\{0,4\}
$$

```
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\in\Sigma.
    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\leftarrowFollowLamda( {i} )
    initialize row T[B][·]
    mark T[B][\cdot] as the starting state of D
    if ( }A\capB\not=\emptyset) then 
        mark T[B][]] as an accepting state of D
    )
    push B onto L
```



NFA to DFA Algorithm discover new state sets

| is Start | is Accept | State | a | b | c | d | e | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ |  |  |  |  |  |  |

$i=0$
Stack $L$ <empty>
$S=\{0,4\}$
$c=\mathrm{a}$
repeat (
$S \leftarrow$ pop $L$
foreach ( $c \in \Sigma$ ) do (
$R \leftarrow$ FollowLambda(FollowChar $(S, c)$ )
$T[S][c] \leftarrow R$
if $(|R|>0$ AND $T[R][\cdot]$ does not exist) then initialize row $T[R][\cdot]$ if $(A \bigcap R \neq 0)$ then ( mark $T[R][\cdot]$ as an accepting state of $D$ ) push $R$ onto $L$ )
)
while $(|L|>0)$


NFA to DFA Algorithm

## FollowChar

$A=\{9,10\}$
$i=0$
Stack $L$ <empty>
$S=\{0,4\}$
$c=\mathrm{a}$
$\emptyset \leftarrow$ FollowChar $(S, c)$
procedure FollowChar ( $S$ a $\subseteq$ of NFA $N$ states, $c \in \Sigma$ )
returns the set of NFA states obtained from following
all $c$ transitions from states in $S$

```
Let }F\mathrm{ be an empty set
foreach ( state t\inS) do (
        foreach ( c transition from t to state q ) do (
        add }q\mathrm{ to }
        )
)
return F
```



# NFA to DFA Algorithm <br> <br> FollowLambda 

 <br> <br> FollowLambda}

Transition Table $T$

| is Start | is Accept | State | a | b | c | d | $\mathbf{e}$ | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ |  |  |  |  |  |  |

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

Stack $L$ <empty>
$S=\{0,4\}$
$c=\mathrm{a}$
$\emptyset \leftarrow$ FollowLambda (Ø)
procedure FollowLambda( $S$ a of NFA $N$ states )
returns the set of NFA states encountered by recursively following only $\boldsymbol{\lambda}$ transitions from states in $S$

```
Let }M\mathrm{ be an empty stack
foreach ( state t\inS) push t onto M
while ( }|M|>0) do 
    t\leftarrow pop M
    foreach ( \lambda transition from t to state q) do (
        if ( q\not\inS) then (
            add q to }
            push q onto M
        )
    )
)
return S
```



NFA to DFA Algorithm discover new state sets

| is Start | is Accept | State | $\mathbf{a}$ | b | c | $\mathbf{d}$ | $\mathbf{e}$ | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | $\emptyset$ |  |  |  |  |  |

$$
A=\{9,10\}
$$

$$
i=0
$$

Stack $L$ <empty>
$S=\{0,4\}$
$c=\mathrm{a}$
$R=\emptyset$

```
repeat (
    S\leftarrow pop L
    foreach ( c\in\Sigma) do (
        R\leftarrow FollowLambda(FollowChar (S,c))
        T[S][c]\leftarrowR
        if ( |R|>0 AND T[R][\cdot] does not exist ) then
            initialize row T[R][·]
            if ( A\bigcapR\not=\emptyset) then (
                mark T[R][\cdot] as an accepting state of D
            )
            push R onto L
        )
        )
        while ( }|||>0
```



NFA to DFA Algorithm discover new state sets

Transition Table $T$

| is Start | is Accept | State | a | b | c | d | e | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | 0 |  |  |  |  |  |

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

Stack $L$ <empty>
$S=\{0,4\}$
$c=\mathrm{b}$
$\{10\} \leftarrow$ FollowChar $(S, c)$
procedure FollowChar ( $S$ a $\subseteq$ of NFA $N$ states, $c \in \Sigma$ ) 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\inS) do (
    foreach ( }c\mathrm{ transition from t to state q ) do (
        add q}\mathrm{ to }
    )
)
return F
```



NFA to DFA Algorithm discover new state sets

Transition Table $T$

| is Start | is Accept | State | a | b | c | d | e | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | 0 |  |  |  |  |  |

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

Stack $L$ <empty>
$S=\{0,4\}$
$c=\mathrm{b}$

$$
\{10\} \leftarrow \text { FollowLambda }(\{10\})
$$

```
procedure FollowLambda( }S\mathrm{ a }\subseteq\mathrm{ of NFA }N\mathrm{ states)
returns the set of NFA states encountered by
recursively following only }\lambda\mathrm{ transitions
from states in S
Let }M\mathrm{ be an empty stack
foreach ( state t\inS) push t onto }
while ( }|M|>0) do 
    t\leftarrow pop }
    foreach ( }\lambda\mathrm{ transition from t to state q) do (
        if ( q\not\inS) then (
            add q}\mathrm{ to }
            push }q\mathrm{ onto }
        )
    )
)
return S
```



NFA to DFA Algorithm
discover new state sets
Transition Table $T$

| is Start | is Accept | State | a | b | c | d | e | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | 0 | $\{10\}$ |  |  |  |  |
| N | Y | $\{10\}$ |  |  |  |  |  |  |

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

$$
\text { Stack } L<\{10\}>
$$

$$
S=\{0,4\}
$$

$$
c=\mathrm{b}
$$

$$
R=\{10\}
$$

```
repeat (
    S\leftarrow pop L
    foreach ( c\in\Sigma) do
        R\leftarrow FollowLambda(FollowChar (S,c))
        T[S][c]\leftarrowR
        if ( |R|>0 AND T[R][\cdot] does not exist ) then
            initialize row T[R][·]
            if ( }A\capR\not=\emptyset) the
                mark T[R][·] as an accepting state of D
            )
            push R onto L
        )
        )
        while ( }|L|>0\mathrm{ )
```



NFA to DFA Algorithm discover new state sets

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0 \\
& \text { Stack } L<\{10\}> \\
& S=\{0,4\} \\
& c=\mathrm{c} \\
& \{1,4\} \leftarrow \text { FollowChar }(S, c)
\end{aligned}
$$

Transition Table $T$
procedure FollowChar ( $S$ a $\subseteq$ of NFA $N$ states, $c \in \Sigma$ ) 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\inS) do (
    foreach ( }c\mathrm{ transition from t to state q ) do (
        add q}\mathrm{ to }
        )
)
return F
```



NFA to DFA Algorithm discover new state sets

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0 \\
& \text { Stack } L<\{10\}> \\
& S=\{0,4\} \\
& c=\mathrm{c} \\
& \{1,4\} \leftarrow \text { FollowLambda }(\{1,4\}
\end{aligned}
$$

Transition Table $T$
procedure FollowLambda( $S$ a $\subseteq$ of NFA $N$ states )
returns the set of NFA states encountered by
recursively following only $\boldsymbol{\lambda}$ transitions
from states in $S$

```
Let }M\mathrm{ be an empty stack
```

Let }M\mathrm{ be an empty stack
foreach ( state t\inS) push t onto }
foreach ( state t\inS) push t onto }
while ( }|M|>0) do
while ( }|M|>0) do
t\leftarrow pop }
t\leftarrow pop }
foreach ( }\lambda\mathrm{ transition from t to state q) do (
foreach ( }\lambda\mathrm{ transition from t to state q) do (
if ( q\not\inS) then (
if ( q\not\inS) then (
add q}\mathrm{ to }
add q}\mathrm{ to }
push q onto M
push q onto M
)
)
)
)
)
)
return S
return S


| is Start | is Accept | State | a | b | c | d | e | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | 0 | $\{10\}$ |  |  |  |  |
| N | Y | $\{10\}$ |  |  |  |  |  |  |

NFA to DFA Algorithm discover new state sets

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

$$
\text { Stack } L<\{1,4\},\{10\}>
$$

$$
S=\{0,4\}
$$

$$
c=\mathrm{c}
$$

$$
R=\{1,4\}
$$

## repeat (

$$
S \leftarrow \text { pop } L
$$

$$
\text { foreach }(c \in \Sigma) \text { do }
$$

$$
R \leftarrow \text { FollowLambda(FollowChar }(S, c))
$$

$$
T[S][c] \leftarrow R
$$

$$
\text { if }(|R|>0 \text { and } T[R][\cdot] \text { does not exist }) \text { then }
$$

$$
\text { initialize row } T[R][\cdot]
$$

$$
\text { if }(A \cap R \neq 0) \text { then }
$$

$$
\operatorname{mark} T[R][\cdot] \text { as an accepting state of } D
$$

            )
            push \(R\) onto \(L\)
        )
        )
        while \((|L|>0)\)
    

NFA to DFA Algorithm

## discover new state sets

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0
\end{aligned}
$$

$$
\text { Stack } L<\{1,4\},\{10\}>
$$

$$
S=\{0,4\}
$$

$$
c=\mathrm{d}
$$

$$
\{5\} \leftarrow \operatorname{FollowChar}(S, c)
$$

$$
\text { procedure FollowChar ( } S \text { a } \subseteq \text { of NFA } N \text { states, } c \in \Sigma \text { ) }
$$

returns the set of NFA states obtained from following

$$
\text { all } c \text { transitions from states in } S
$$

```
Let F be an empty set
foreach ( state t\inS) do (
    foreach ( }c\mathrm{ transition from t to state q ) do (
        add q}\mathrm{ to }
        )
)
return F
```



NFA to DFA Algorithm discover new state sets

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0 \\
& \text { Stack } L<\{1,4\},\{10\}> \\
& S=\{0,4\} \\
& c=\mathrm{d} \\
& \{5,6,8\} \leftarrow \text { FollowLambda }(\{5\})
\end{aligned}
$$

Transition Table $T$

```
procedure FollowLambda( }S\mathrm{ a }\subseteq\mathrm{ of NFA N states )
returns the set of NFA states encountered by
recursively following only }\lambda\mathrm{ transitions
from states in S
Let }M\mathrm{ be an empty stack
foreach (state t\inS) push t onto }
while ( }|M|>0) do 
    t\leftarrow pop }
    foreach ( }\lambda\mathrm{ transition from t to state q) do (
        if ( q\not\inS) then (
            add q}\mathrm{ to }
            push q onto M
        )
    )
)
```



| is Start | is Accept | State | a | b | c | d | e | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | 0 | $\{10\}$ | $\{1,4\}$ |  |  |  |
| N | Y | $\{10\}$ |  |  |  |  |  |  |
| N | N | $\{1,4\}$ |  |  |  |  |  |  |

```
NFA to DFA Algorithm
    discover new state sets
Transition Table \(T\)
\begin{tabular}{ccccccccc} 
is Start & is Accept & State & a & b & c & d & e & \(\mathbf{f}\) \\
\hline \hline Y & N & \(\{0,4\}\) & 0 & \(\{10\}\) & \(\{1,4\}\) & \(\{5,6,8\}\) & \\
N & Y & \(\{10\}\) & & & & & \\
N & N & \(\{1,4\}\) & & & & & \\
N & N & \(\{5,6,8\}\) & & & & &
\end{tabular}
```

A={9,10}

```
A={9,10}
i=0
i=0
Stack L<{5,6,8},{1,4},{10}>
Stack L<{5,6,8},{1,4},{10}>
S={0,4}
S={0,4}
d=\textrm{d}
d=\textrm{d}
R={5,6,8}
R={5,6,8}
repeat (
        S\leftarrow pop L
        foreach ( c\in\Sigma) ) do (
        R\leftarrow FollowLambda(FollowChar (S,c))
        T[S][c]\leftarrowR
        if ( |R|>0 AND T[R][\cdot] does not exist) then
            initialize row T[R][·]
            if ( }A\capR\not=0)\mathrm{ then (
                mark T[R][·] as an accepting state of D
            )
            push R onto L
        )
        )
while ( }|L|>0
```


Transition Table $T$
NFA to DFA Algorithm characters e and fyield $\emptyset$

| is Start | is Accept | State | a | b | c | d | e | f |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | $\emptyset$ | $\{10\}$ | $\{1,4\}$ | $\{5,6,8\}$ | $\emptyset$ | $\emptyset$ |
| N | Y | $\{10\}$ |  |  |  |  |  |  |
| N | N | $\{1,4\}$ |  |  |  |  |  |  |
| N | N | $\{5,6,8\}$ |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

$$
S=\{0,4\}
$$

$$
c=\mathrm{f}
$$

$$
R=\emptyset
$$

repeat (
repeat (
S\leftarrow pop L
S\leftarrow pop L
foreach ( c\in\Sigma) do
foreach ( c\in\Sigma) do
R\leftarrow FollowLambda(FollowChar (S,c))
R\leftarrow FollowLambda(FollowChar (S,c))
T[S][c]\leftarrowR
T[S][c]\leftarrowR
if ( |R|>0 AND T[R][\cdot] does not exist ) then
if ( |R|>0 AND T[R][\cdot] does not exist ) then
initialize row T[R][·]
initialize row T[R][·]
if ( A\bigcapR\not=0) then
if ( A\bigcapR\not=0) then
mark T[R][·] as an accepting state of D
mark T[R][·] as an accepting state of D
)
)
push R onto L
push R onto L
)
)
)
)
while ( }|L|>0
while ( }|L|>0


| NFA to DFA Algorithm pop $L$ and do it all again | Transition Table $T$ |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |  |  |  |  |  |
|  | Y | N | \{0,4\} | 0 | \{10\} | \{1,4\} | \{5,6,8\} | $\emptyset$ | 0 |
| $A=\{9,10\}$ | N | Y | \{10\} |  |  |  |  |  |  |
| $i=0$ | N | N | \{1,4\} |  |  |  |  |  |  |
| Stack $L<\{9\},\{1,4\},\{10\}>$ | N | N | \{5,6,8\} | 0 | \{9\} | 0 | $\emptyset$ | 0 | \{9\} |
| $S=\{5,6,8\}$ | N | Y | \{9\} |  |  |  |  |  |  |

```
repeat (
    S\leftarrow pop L
    foreach ( c\in\Sigma) do (
        R\leftarrow FollowLambda(FollowChar (S,c))
        T[S][c]\leftarrowR
        if ( |R|>0 AND T[R][\cdot] does not exist ) then
            initialize row T[R][·]
            if ( }A\capR\not=\emptyset) the
                mark T[R][\cdot] as an accepting state of D
            )
            push R onto L
        )
    )
    while ( }|L|>0
```



| NFA to DFA Algorithm pop $L$ and do it all again | Transition Table $T$ |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | is Start | is Accept | State | a | b | c | d | e | f |
|  | Y | N | \{0,4\} | 0 | \{10\} | \{1,4\} | \{5,6,8\} | 0 | 0 |
| $A=\{9,10\}$ | N | Y | \{10\} |  |  |  |  |  |  |
| $i=0$ | N | N | $\{1,4\}$ |  |  |  |  |  |  |
| Stack $L<\{8\},\{1,4\},\{10\}>$ | N | N | \{5,6,8\} | 0 | \{9\} | 0 | 0 | 0 | \{9\} |
| $S=\{9\}$ | N | Y | \{9\} | 0 | 0 | 0 | \{9\} | \{8\} | 0 |
|  | N | N | \{8\} |  |  |  |  |  |  |

```
repeat (
    S\leftarrow pop L
    foreach ( c\in\Sigma) do (
        R\leftarrow FollowLambda(FollowChar (S,c))
        T[S][c]\leftarrowR
        if ( |R|>0 AND T[R][\cdot] does not exist ) then
            initialize row T[R][·]
            if ( A\bigcapR\not=0) then
                mark T[R][·] as an accepting state of D
            )
            push R onto L
        )
    )
    while ( |L|>0)
```



Transition Table $T$

## pop $L$ and do it all again

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0 \\
& \text { Stack } L<\{1,4\},\{10\}> \\
& S=\{8\}
\end{aligned}
$$

| is Start | is Accept | State | a | b | c | d | e | f |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | 0 | $\{10\}$ | $\{1,4\}$ | $\{5,6,8\}$ | 0 | 0 |
| N | Y | $\{10\}$ |  |  |  |  |  |  |
| N | N | $\{1,4\}$ |  |  |  |  |  |  |
| N | N | $\{5,6,8\}$ | 0 | $\{9\}$ | 0 | 0 | 0 | $\{9\}$ |
| N | Y | $\{9\}$ | 0 | 0 | 0 | $\{9\}$ | $\{8\}$ | 0 |
| N | N | $\{8\}$ | 0 | $\{9\}$ | 0 | 0 | 0 | 0 |

```
repeat (
    S\leftarrow pop L
    foreach ( c\in\Sigma) do (
        R\leftarrow FollowLambda(FollowChar (S,c))
        T[S][c]\leftarrowR
        if ( |R|>0 AND T[R][\cdot] does not exist ) then
            initialize row T[R][·]
            if ( A\bigcapR\not=0) then
                mark T[R][·] as an accepting state of D
            )
            push R onto L
        )
    )
    while ( }|L|>0
```



Transition Table $T$ pop $L$ and do it all again

| is Start | is Accept | State | a | b | c | d | e | $\mathbf{f}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | 0 | $\{10\}$ | $\{1,4\}$ | $\{5,6,8\}$ | 0 | 0 |
| N | Y | $\{10\}$ |  |  |  |  |  |  |
| N | N | $\{1,4\}$ | 0 | 0 | $\{4\}$ |  |  |  |
| N | N | $\{5,6,8\}$ | 0 | $\{9\}$ | 0 | 0 | 0 | $\{9\}$ |
| N | Y | $\{9\}$ | 0 | 0 | 0 | $\{9\}$ | $\{8\}$ | 0 |
| N | N | $\{8\}$ | 0 | $\{9\}$ | 0 | 0 | 0 | 0 |
| N | N | $\{4\}$ |  |  |  |  |  |  |

```
A={9,10}
i=0
Stack L<{2,6}, {4}, {10}>
S={1,4}
c=c
R={4}
repeat (
    S\leftarrow pop L
    foreach ( c\in\Sigma ) do (
        R\leftarrow FollowLambda(FollowChar (S,c))
        T[S][c]\leftarrowR
        if ( |R|>0 AND T[R][·] does not exist ) then
            initialize row T[R][·]
            if ( A\bigcapR\not=\emptyset) then (
                mark T[R][·] as an accepting state of D
            )
            push R onto L
        )
        )
while ( }|L|>0
```



Transition Table $T$

## pop $L$ and do it all again

$$
\begin{aligned}
& A=\{9,10\} \\
& i=0 \\
& \text { Stack } L<\{2,6\},\{4\},\{10\}> \\
& S=\{1,4\} \\
& c=\mathrm{d} \\
& R=\{5,6,8\}
\end{aligned}
$$

| is Start | is Accept | State | a | b | c | d | e | f |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | $\emptyset$ | $\{10\}$ | $\{1,4\}$ | $\{5,6,8\}$ | 0 | 0 |
| N | Y | $\{10\}$ |  |  |  |  |  |  |
| N | N | $\{1,4\}$ | $\emptyset$ | 0 | $\{4\}$ | $\{5,6,8\}$ |  |  |
| N | N | $\{5,6,8\}$ | 0 | $\{9\}$ | 0 | 0 | 0 | $\{9\}$ |
| N | Y | $\{9\}$ | 0 | 0 | 0 | $\{9\}$ | $\{8\}$ | 0 |
| N | N | $\{8\}$ | 0 | $\{9\}$ | 0 | 0 | 0 | 0 |
| N | N | $\{4\}$ |  |  |  |  |  |  |

```
repeat (
    S\leftarrow pop L
    foreach ( c\in\Sigma ) do (
        R\leftarrow FollowLambda(FollowChar (S,c))
        T[S][c]\leftarrowR
        if ( |R|>0 AND T[R][\cdot] does not exist ) then
            initialize row T[R][·]
            if ( }A\capR\not=0) then 
                mark T[R][·] as an accepting state of D
            )
            push R onto L
        )
        )
        while ( }|L|>0
```



Transition Table $T$
pop $L$ and do it all again

| is Start | is Accept | State | a | b | c | d | e | f |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Y | N | $\{0,4\}$ | $\emptyset$ | $\{10\}$ | $\{1,4\}$ | $\{5,6,8\}$ | $\emptyset$ | $\emptyset$ |
| N | Y | $\{10\}$ |  |  |  |  |  |  |
| N | N | $\{1,4\}$ | $\emptyset$ | $\emptyset$ | $\{4\}$ | $\{5,6,8\}$ | $\{2,6\}$ |  |
| N | N | $\{5,6,8\}$ | $\emptyset$ | $\{9\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{9\}$ |
| N | Y | $\{9\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{9\}$ | $\{8\}$ | $\emptyset$ |
| N | N | $\{8\}$ | $\emptyset$ | $\{9\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| N | N | $\{4\}$ |  |  |  |  |  |  |
| N | N | $\{2,6\}$ |  |  |  |  |  |  |

## repeat (

$S \leftarrow$ pop $L$
foreach ( $c \in \Sigma$ ) do (
$R \leftarrow$ FollowLambda(FollowChar $(S, c)$ )
$T[S][c] \leftarrow R$
if ( $|R|>0$ AND $T[R][\cdot]$ does not exist) then initialize row $T[R][\cdot]$ if $(A \cap R \neq \emptyset)$ then ( mark $T[R][\cdot]$ as an accepting state of $D$
)
push $R$ onto $L$
)
)
while $(|L|>0)$


NFA to DFA Algorithm
Transition Table $T$

| $A=\{9,10\}$ | N | Y | $\{10\}$ | $\{0,4\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{1\}$ | $\emptyset$ |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $i=0$ | N | N | $\{1,4\}$ | $\emptyset$ | $\emptyset$ | $\{4\}$ | $\{5,6,8\}$ | $\{2,6\}$ | $\emptyset$ |
| Stack $L$ <empty> | N | N | $\{5,6,8\}$ | $\emptyset$ | $\{9\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{9\}$ |
|  | N | Y | $\{9\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{9\}$ | $\{8\}$ | $\emptyset$ |
|  | N | N | $\{8\}$ | $\emptyset$ | $\{9\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
|  | N | N | $\{4\}$ | $\emptyset$ | $\emptyset$ | $\{4\}$ | $\{5,6,8\}$ | $\emptyset$ | $\emptyset$ |
|  | N | N | $\{2,6\}$ | $\{10\}$ | $\{3,10\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{9\}$ |
|  | N | Y | $\{3,10\}$ | $\{0,4\}$ | $\emptyset$ | $\{7\}$ | $\emptyset$ | $\{1\}$ | $\emptyset$ |
|  | N | N | $\{7\}$ | $\{9\}$ | $\emptyset$ | $\emptyset$ | $\{8\}$ | $\emptyset$ | $\emptyset$ |
|  | N | N | $\{1\}$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\{2,6\}$ | $\emptyset$ |

When $L$ is empty, the table $T$ holds a DFA derived from the original NFA.


