LEX9: Implementing DFAs

Lexical Analysis

CMPT 379: Compilers

Instructor: Anoop Sarkar

anoopsarkar.github.io/compilers-class

Building a Lexical Analyzer

- Token $\Rightarrow$ Pattern
- Pattern $\Rightarrow$ Regular Expression
- Regular Expression $\Rightarrow$ NFA
- NFA $\Rightarrow$ DFA
- DFA $\Rightarrow$ Table-driven implementation of DFA

Building a Lexical Analyzer

- Token $\Rightarrow$ Pattern
- Pattern $\Rightarrow$ Regular Expression
- Regular Expression $\Rightarrow$ NFA
- NFA $\Rightarrow$ DFA
- DFA $\Rightarrow$ Table-driven implementation of DFA

Building a Lexical Analyzer

- Token $\Rightarrow$ Pattern
- Pattern $\Rightarrow$ Regular Expression
- Regular Expression $\Rightarrow$ NFA
- NFA $\Rightarrow$ DFA

Implement NFAs

- DFA $\Rightarrow$ Table-driven implementation of DFA

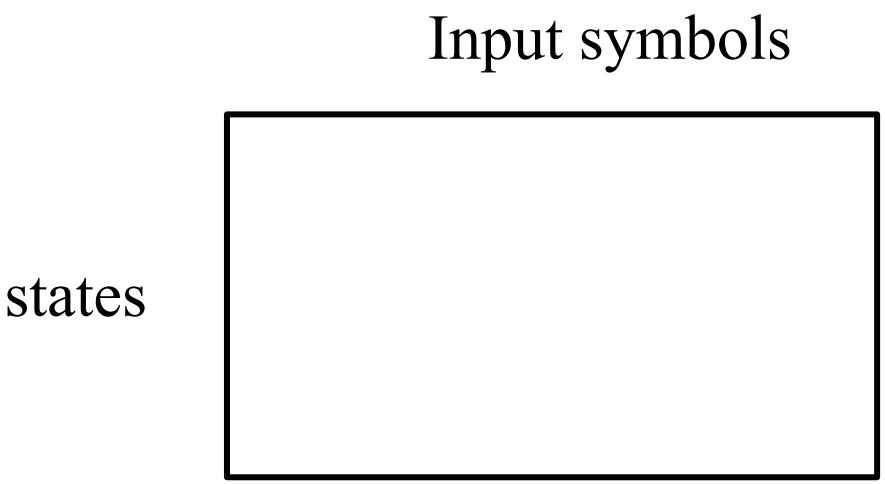
Building a Lexical Analyzer

- Token $\Rightarrow$ Pattern
- Pattern $\Rightarrow$ Regular Expression
- Regular Expression $\Rightarrow$ NFA
- NFA $\Rightarrow$ DFA
- DFA $\Rightarrow$ Table-driven implementation of DFA

Implement NFAs
Convert regexp to DFA

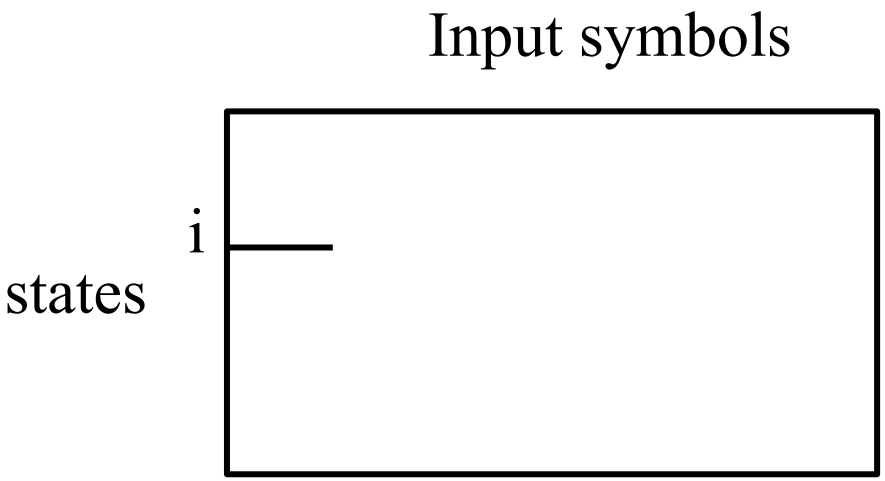
Implementing DFAs

- 2D array storing the transition table
  - One dimension is states
  - Other dimension is input symbols
  - For every transition $S_i \rightarrow^{a} S_k$ define T[i,a]=k



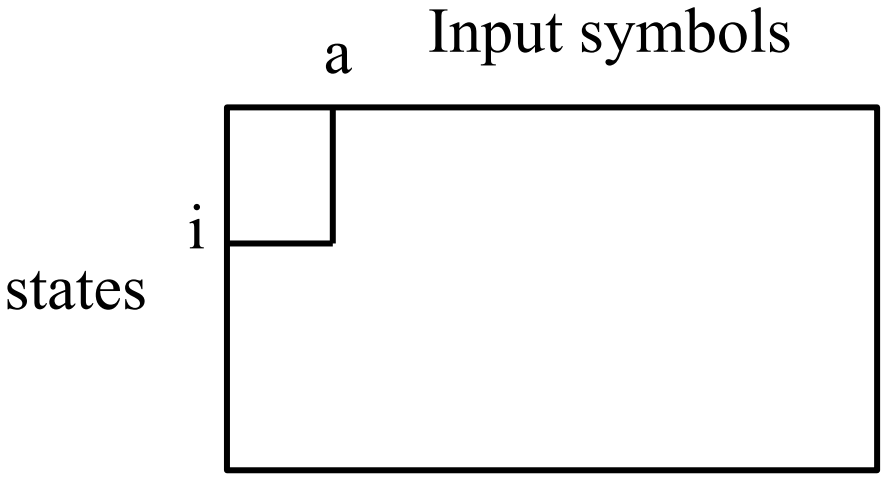
Implementing DFAs

- 2D array storing the transition table
  - One dimension is states
  - Other dimension is input symbols
  - For every transition $S_i \rightarrow^{a} S_k$ define T[i,a]=k



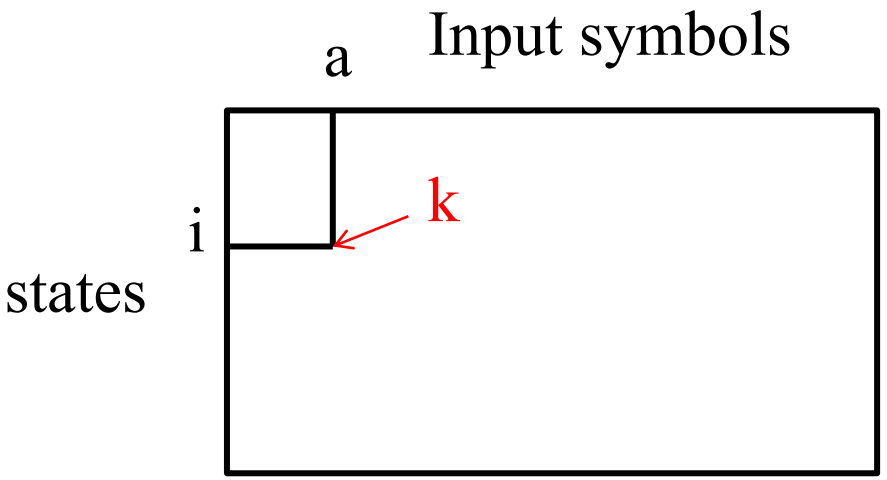
Implementing DFAs

- 2D array storing the transition table
  - One dimension is states
  - Other dimension is input symbols
  - For every transition $S_i \rightarrow^{a} S_k$ define T[i,a]=k

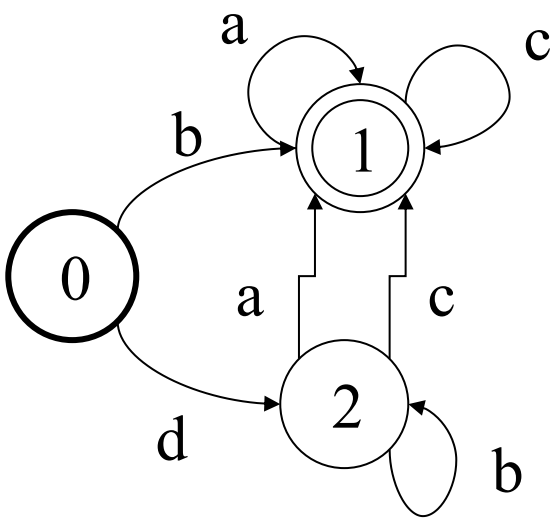


Implementing DFAs

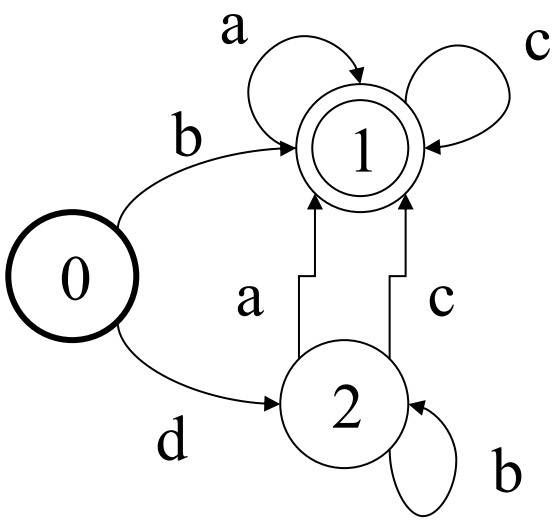
- 2D array storing the transition table
  - One dimension is states
  - Other dimension is input symbols
  - For every transition $S_i \rightarrow^a S_k$ define T[i,a]=k



Implementing DFAs

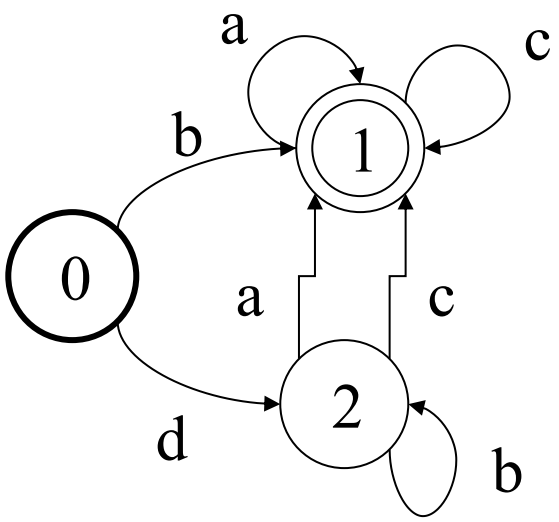


Implementing DFAs



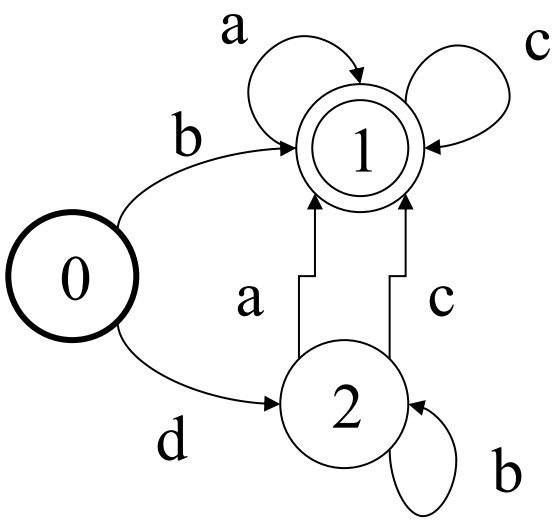
	a	b	c	d
0				
1				
2				

Implementing DFAs



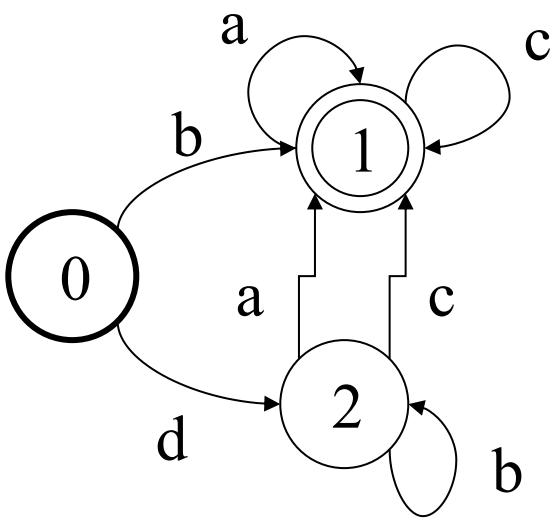
	a	b	c	d
0	-	1	-	2
1				
2				

Implementing DFAs



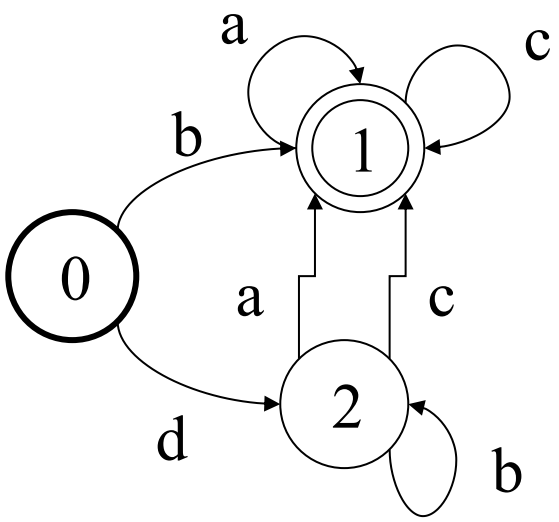
	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2				

Implementing DFAs



	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

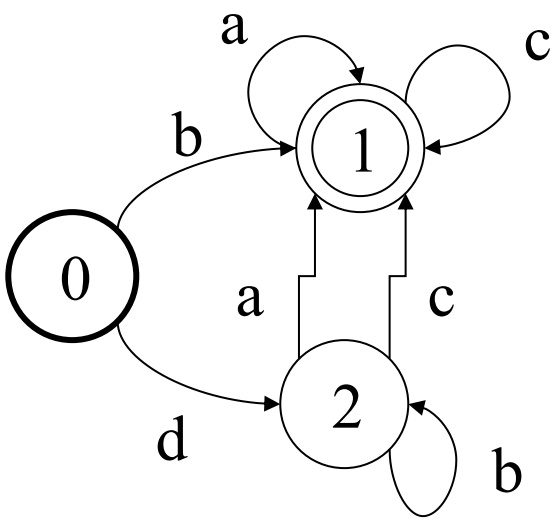
Implementing DFAs



	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

```
i = 0
state = 0
while (input[i]) {
    state = nextState(state, input[i])
    i = i + 1
}
```

Implementing DFAs



	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

```
i = 0
state = 0
while (input[i]) {
    state = nextState(state, input[i])
    i = i + 1
}
```

```
nextState(state, x) {
    return A[state][x]
}
```

Implementing DFAs

- 2D array storing the transition table
  - Too many states and duplicates

Implementing DFAs

- 2D array storing the transition table
  - Too many states and duplicates
- Adjacency list
  - more space efficient but slower

Implementing DFAs

- 2D array storing the transition table
  - Too many states and duplicates
- Adjacency list
  - more space efficient but slower
- Merge two ideas: array structures used for sparse tables like DFA transition tables

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

0	2
1	4
2	0

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

nextState(2,a)=

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

nextState(2,a)= next[0+0]

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

nextState(2,a)= next[0+0]

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

nextState(2,a)= next[0+0]

nextState(1,c)=

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

nextState(2,a)= next[0+0]

nextState(1,c)= next[4+2]

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

nextState(2,a)= next[0+0]

nextState(1,c)= next[4+2]

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

nextState(2,a)= next[0+0]

nextState(1,c)= next[4+2]

nextState(0,c)=

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

nextState(2,a)= next[0+0]

nextState(1,c)= next[4+2]

nextState(0,c)= next[2+2]

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

base

0	2
1	4
2	0

nextState(2,a)= next[0+0]

nextState(1,c)= next[4+2]

nextState(0,c)= next[2+2]

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

2	2	2	0	1	0	1	-

check

base

0	2
1	4
2	0

nextState(2,a)= next[0+0]

nextState(1,c)= next[4+2]

nextState(0,c)= next[2+2]

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

base

0	2
1	4
2	0

		-	1	-	2		
				1	-	1	-
1	2	1	-				
1	2	1	1	1	2	1	-
0	1	2	3	4	5	6	7

next

2	2	2	0	1	0	1	-

check

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L] **if** check[L] == *s*

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

-	1	-	2			
			1	-	1	-
-	2	-	-			
-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			
-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

base

0	1
1	3
2	0

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			
-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

base

0	1
1	3
2	0

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L]

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

base

0	1
1	3
2	0

	-	1	-	2		
			1	-	1	-
-	2	-	-			
-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L]

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

base

0	1
1	3
2	0

	-	1	-	2		
			1	-	1	-
-	2	-	-			
-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

nextState(*s*, *x*) :

L := base[*s*] + *x*

return next[L] **if** check[L] == *s*

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			
-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

nextState(*s*, *x*) :

L := base[*s*] + *x*

return next[L] **if** check[L] == *s*

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			
-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(*s*, *x*) :

L := base[*s*] + *x*

return next[L] **if** check[L] == *s*

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			

-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(*s*, *x*) :

L := base[*s*] + *x*

return next[L] **if** check[L] == *s*

else return *nextState*(default[*s*], *x*)

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			
-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(2,b)=

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L] **if** check[L] == *s*
  else return *nextState*(default[*s*], *x*)

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			

-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(2,b)=

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L] **if** check[L] == *s*
  else return *nextState*(default[*s*], *x*)

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			

-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(2,b)=

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L] **if** check[L] == *s*
  else return *nextState*(default[*s*], *x*)

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			

-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

nextState(2,b)=next[1]

base

0	1
1	3
2	0

-
-
1

default

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L] **if** check[L] == *s*
  else return *nextState*(default[*s*], *x*)

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			
-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(*s*, *x*) :

L := base[*s*] + *x*

return next[L] **if** check[L] == *s*

else return *nextState*(default[*s*], *x*)

nextState(2,a)=

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			

-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(*s*, *x*) :

L := base[*s*] + *x*

return next[L] **if** check[L] == *s*

else return *nextState*(default[*s*], *x*)

nextState(2,a)=

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			

-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(2,a)=

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L] **if** check[L] == *s*
  else return *nextState*(default[*s*], *x*)

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			

-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L] **if** check[L] == *s*
  else return *nextState*(default[*s*], *x*)

nextState(2,a)=
nextState(1,a)

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			

-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L] **if** check[L] == *s*
  else return *nextState*(default[*s*], *x*)

nextState(2,a)=
nextState(1,a)

Implementing DFAs

	a	b	c	d
0	-	1	-	2
1	1	-	1	-
2	1	2	1	-

	-	1	-	2		
			1	-	1	-
-	2	-	-			

-	2	1	1	2	1	-
0	1	2	3	4	5	6

next

-	2	0	1	0	1	-

check

base

0	1
1	3
2	0

-
-
1

default

nextState(*s*, *x*) :
  L := base[*s*] + *x*
  return next[L] **if** check[L] == *s*
  else return *nextState*(default[*s*], *x*)

nextState(2,a)=
nextState(1,a)= next[3]