Lexical Analysis

Uday Khedker

(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

January 2025

Outline

- Introduction
- Specifying scanners
- Tokenizing input using DFAs
- Constructing DFAs
- Representing DFAs using four-arrays
- Minimizing DFAs

Introduction

Introduction

Prof. Sanyal's slides (scanning-slides-sanyal-part1.pdf)

Specifying Scanners

Introduction

Prof. Sanyal's slides (scanning-slides-sanyal-part2.pdf)

Tokenizing the Input Using DFAs

An Example for Scanning: Specifications

Let L and D denote the set of all letters and digits, respectively

Pattern	Token
int	INT
L(L|D)*	ID
D^+	NUM
=	=
;	;

We will scan the input string int int32=5;↩

Example for Scanning: DFA for the Patterns

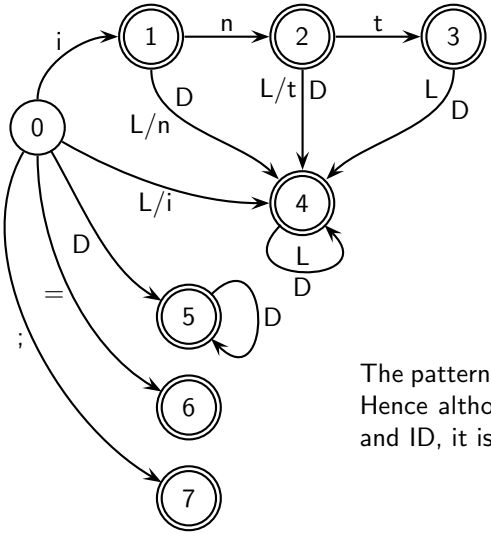
Formally, a Deterministic Finite Automaton (DFA) is a five tuple

$$(\Sigma, S, s_0, \delta, F)$$

where

- Σ is the input alphabet
- S is the set of states
- $s_o \in S$ is a unique start state
- $\delta : S \times \Sigma \rightarrow S$ is a transition function
- $F \subseteq S$ is a set of final states

Example for Scanning: DFA for the Patterns



States	Action
3	Found INT
1, 2, 4	Found ID
5	Found NUM
6	Found =
7	Found ;

The patterns for INT precedes the pattern for ID
Hence although state 3 could accept both INT and ID, it is made to accept only INT

A Format to Show A Trace of Scanning

Step No	State	MatchedString	Buffer	NextChar	LastFinalState	MarkedPos	Action

- State (S). Current State
- MatchedString (MS). Prefix of the buffer matched to identify a lexeme
- Buffer.
- NextChar (NC). The next character in the input; it will be shifted to the buffer if there is a valid transition in the DFA
- LastFinalState (LFS). The last final state seen
- MarkedPos (MP). The position of the character (in the buffer) just after the last seen lexeme
- Action.

A Format to Show A Trace of Scanning

Step No	State	MatchedString	Buffer	NextChar	LastFinalState	MarkedPos	Action

- State (S). Current State
- M
- B
- N
  th
- La
- M
  se
- Action.

When there is no transition on Nextchar,

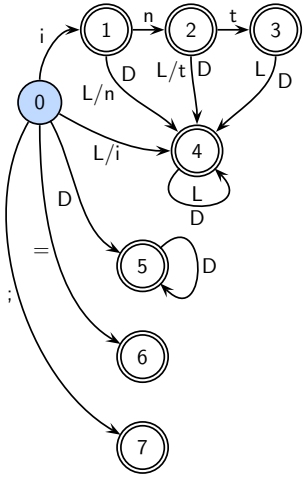
- if MarkedPos is -1, no final state is seen, the first character in the buffer is discarded, and the second character becomes NextChar,
- otherwise, the lexeme upto MarkedPos (excluding it) is returned, the character at MarkedPos becomes NextChar

In either case, the LastFinalState is set to -1 and the state is set to 0

if

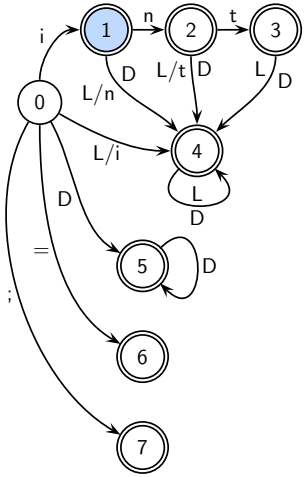
st

Scanning the Input "int int32=5;↩"



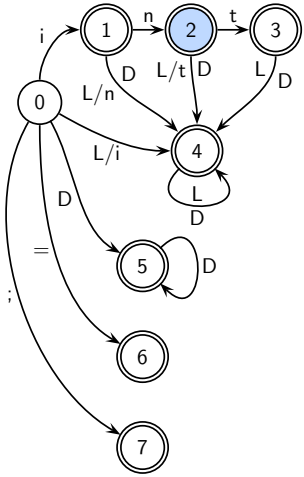
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			

Scanning the Input "int int32=5;↩"



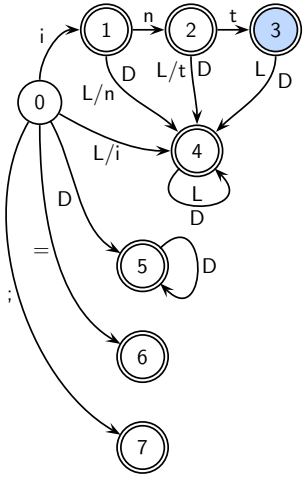
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	

Scanning the Input "int int32=5;↩"



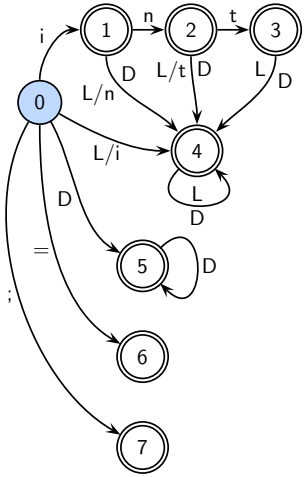
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	

Scanning the Input "int int32=5;↩"



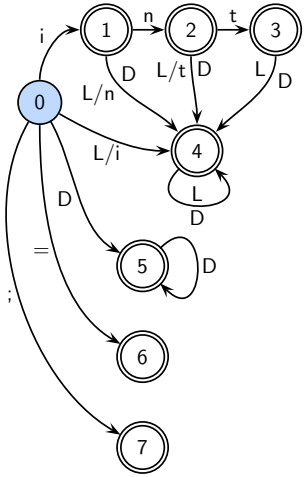
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT

Scanning the Input "int int32=5;↩"



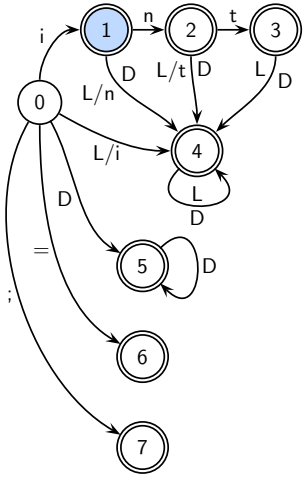
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣

Scanning the Input "int int32=5;↩"



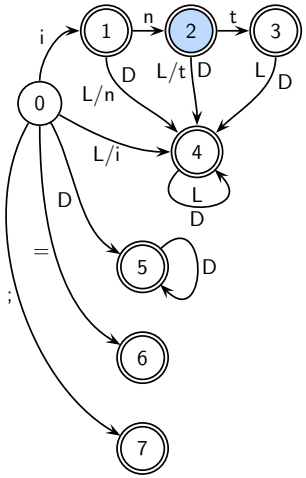
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			

Scanning the Input "int int32=5;↩"



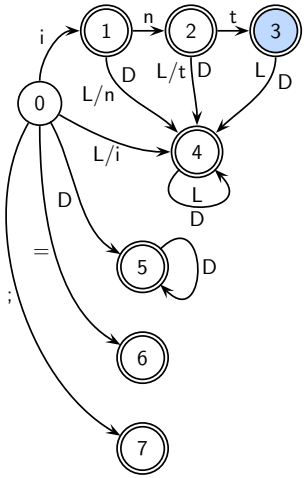
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	

Scanning the Input "int int32=5;↩"



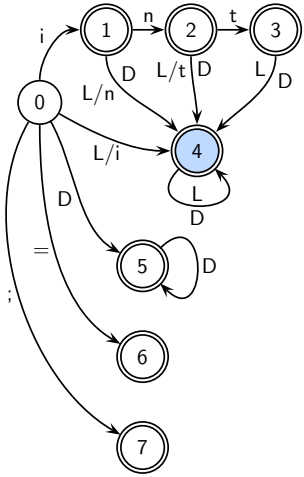
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	

Scanning the Input "int int32=5;↩"



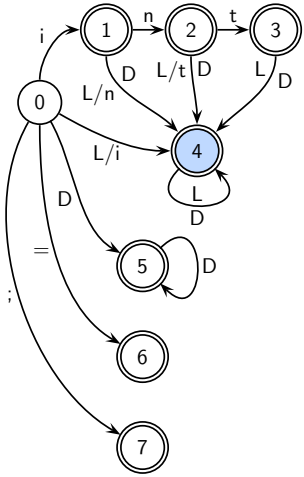
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	
9	3	int	int32=5;↩	3	3	3	

Scanning the Input "int int32=5;↩"



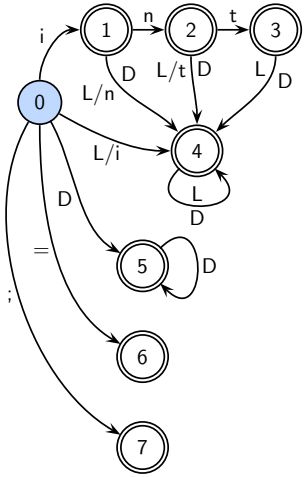
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	
9	3	int	int32=5;↩	3	3	3	
10	4	int3	int32=5;↩	2	4	4	

Scanning the Input "int int32=5;↩"



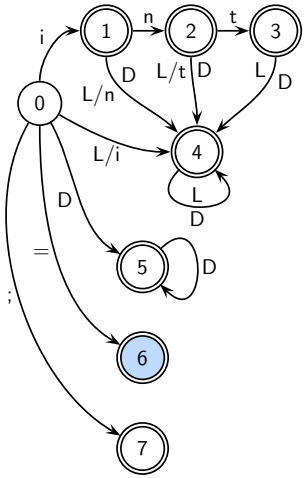
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	
9	3	int	int32=5;↩	3	3	3	
10	4	int3	int32=5;↩	2	4	4	
11	4	int32	int32=5;↩	=	4	5	Found ID

Scanning the Input "int int32=5;↩"



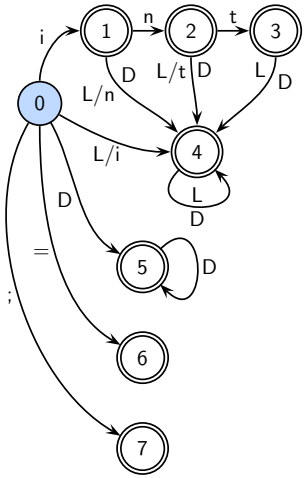
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	
9	3	int	int32=5;↩	3	3	3	
10	4	int3	int32=5;↩	2	4	4	
11	4	int32	int32=5;↩	=	4	5	Found ID
12	0		=5;↩	=			

Scanning the Input "int int32=5;↩"



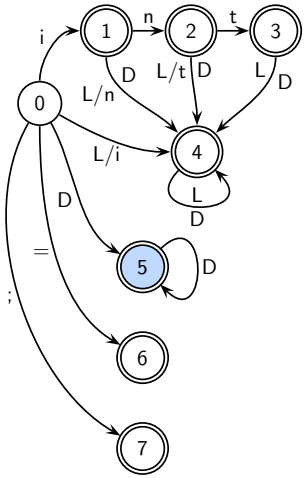
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	
9	3	int	int32=5;↩	3	3	3	
10	4	int3	int32=5;↩	2	4	4	
11	4	int32	int32=5;↩	=	4	5	Found ID
12	0		=5;↩	=			
13	6	=	=5;↩	5	6	1	Found =

Scanning the Input "int int32=5;↩"



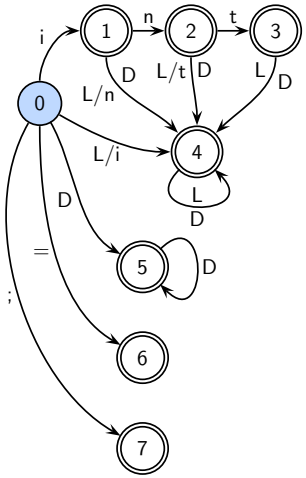
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	
9	3	int	int32=5;↩	3	3	3	
10	4	int3	int32=5;↩	2	4	4	
11	4	int32	int32=5;↩	=	4	5	Found ID
12	0		=5;↩	=			
13	6	=	=5;↩	5	6	1	Found =
14	0		5;↩	5			

Scanning the Input "int int32=5;↩"



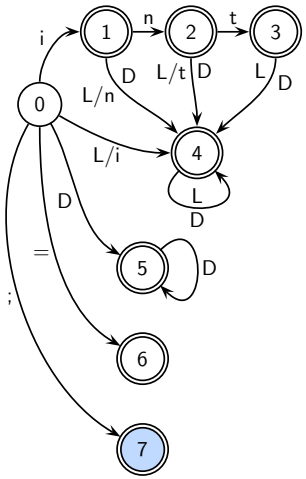
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	
9	3	int	int32=5;↩	3	3	3	
10	4	int3	int32=5;↩	2	4	4	
11	4	int32	int32=5;↩	=	4	5	Found ID
12	0		=5;↩	=			
13	6	=	=5;↩	5	6	1	Found =
14	0		5;↩	5			
15	5	5	5;↩	;	5	1	Found NUM

Scanning the Input "int int32=5;↩"



SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	
9	3	int	int32=5;↩	3	3	3	
10	4	int3	int32=5;↩	2	4	4	
11	4	int32	int32=5;↩	=	4	5	Found ID
12	0		=5;↩	=			
13	6	=	=5;↩	5	6	1	Found =
14	0		5;↩	5			
15	5	5	5;↩	;	5	1	Found NUM
16	0		;↩	;			

Scanning the Input "int int32=5;↩"



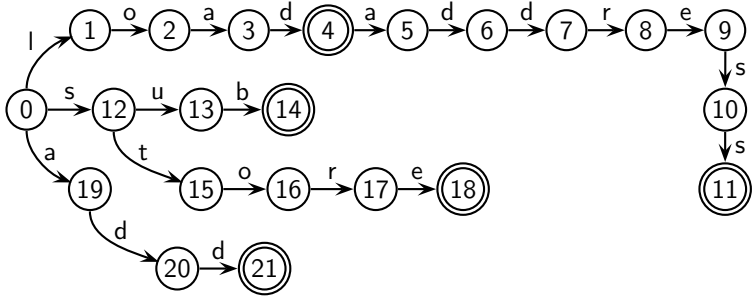
SNo	S	MS	Buffer	NC	LFS	MP	Action
1	0		int␣int32=5;↩	i			
2	1	i	int␣int32=5;↩	n	1	1	
3	2	in	int␣int32=5;↩	t	2	2	
4	3	int	int␣int32=5;↩	␣	3	3	Found INT
5	0		␣int32=5;↩	␣			Discard ␣
6	0		int32=5;↩	i			
7	1	i	int32=5;↩	n	1	1	
8	2	in	int32=5;↩	t	2	2	
9	3	int	int32=5;↩	3	3	3	
10	4	int3	int32=5;↩	2	4	4	
11	4	int32	int32=5;↩	=	4	5	Found ID
12	0		=5;↩	=			
13	6	=	=5;↩	5	6	1	Found =
14	0		5;↩	5			
15	5	5	5;↩	;	5	1	Found NUM
16	0		;↩	;			
17	7	;	↩	↩	7		Found ;

Tutorial Problem On Scanning

- Find the occurrences of following substrings in a given input string

  load, loadaddress, add, sub, store

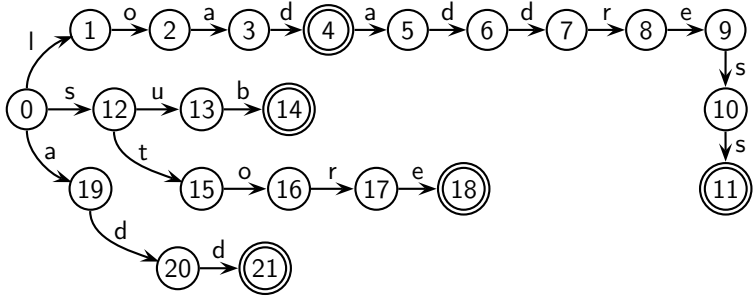
- Use the following automata



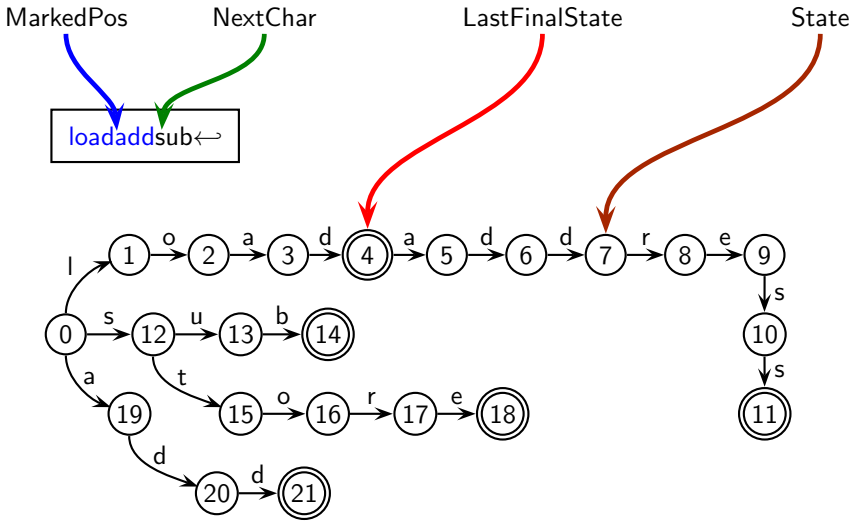
- Scan two input strings loadsubadd↩ and loadaddsub↩

The Role of MarkedPos

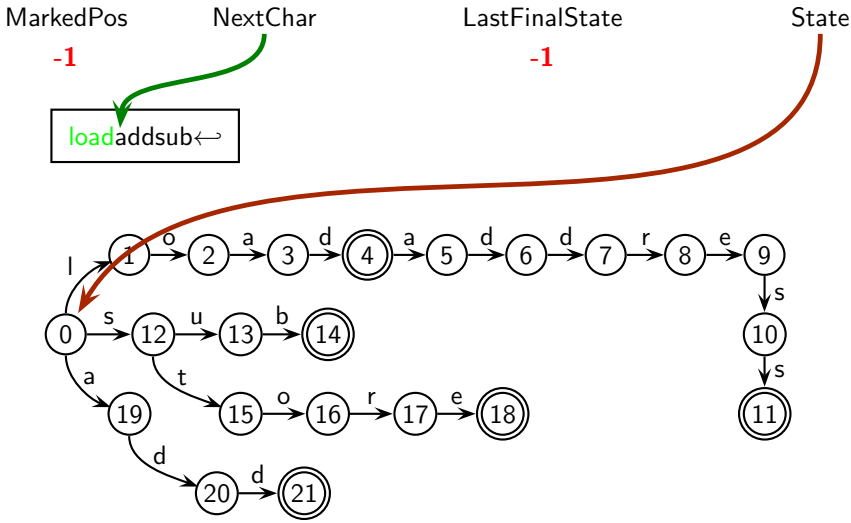
Observe the role of MarkedPos for the input loadaddsub⟲



The Role of MarkedPos



The Role of MarkedPos



Demo of Scan Trace

Constructing DFAs

Constructing DFA for Multiple Patterns

- Join multiple DFAs/NFAs using ϵ transition

  Transition without consuming any input symbol
- This creates an NFA (Non-deterministic Finite Automaton)
  - Possible transition without consuming any input symbol
  - Possibly multiple transitions on the same input symbol
- Make the NFA deterministic by subset construction
  - Each state in the resulting DFA is a set of "similar" states of the NFA
  - The start state of the DFA is a union of all original start states (of multiple patterns)
  - Subsequent states are identified by finding out the sets of states of the NFA for each possible input symbol

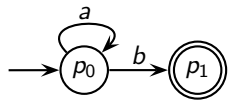
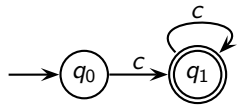
Constructing NFA for a Regular Expression

Consider a regular expression R. Apply steps 1 to 4 to construct an NFA for R inductively:

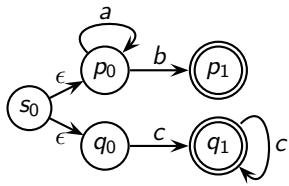
1. If R is a letter in the alphabet Σ, create a two state NFA that accepts the letter (single transition from the start state to a single final state on the letter)
2. If R is $R_1 \cdot R_2$, create an NFA by joining the two NFAs N_1 and N_2 by adding an epsilon transition from every final state of N_1 to the start state of N_2.
3. If the R is $R_1 \mid R_2$, create an NFA by joining the two NFAs N_1 and N_2 by creating a new start state s_0 and a new final state s_f. Add an epsilon transition from s_0 the start state of R_1 and similarly for R_2. Add an epsilon transition from every final state of N_1 to s_f and similarly for N_2.
4. If R is R_1*, create an NFA by adding an epslion transition from every final state of R_1 to the start state of R_1

   Alternatively, we can create a new start state s_0 with an epsilon transition to the start state of R_1 and a new final state s_f with epsilon transitions from the final states of R_1, and then add an espilon transition from s_f to s_0.

Constructing DFA for Multiple Patterns: Example 1

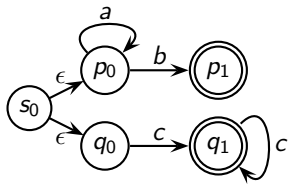
Pattern	DFA
a^*b	
c^+	

Constructing DFA for Multiple Patterns: Example 1



State	*Transition*		
	a	*b*	*c*

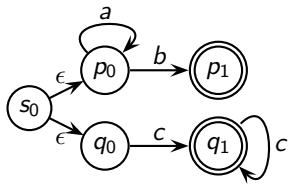
Constructing DFA for Multiple Patterns: Example 1



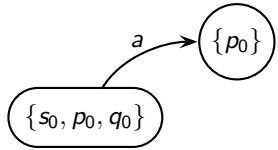
State	*Transition*		
	a	*b*	*c*
$\{s_0, p_0, q_0\}$			

$\{s_0, p_0, q_0\}$

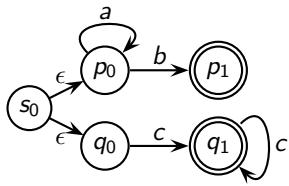
Constructing DFA for Multiple Patterns: Example 1



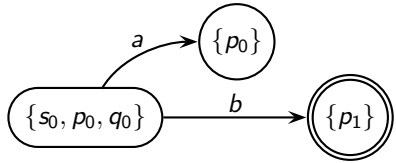
State	Transition		
	a	b	c
$\{s_0, p_0, q_0\}$	$\{p_0\}$		
$\{p_0\}$			



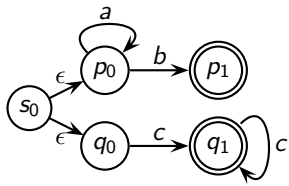
Constructing DFA for Multiple Patterns: Example 1



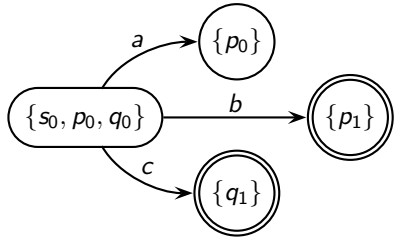
State	Transition		
	a	b	c
$\{s_0, p_0, q_0\}$	$\{p_0\}$	$\{p_1\}$	
$\{p_0\}$			
$\{p_1\}$			



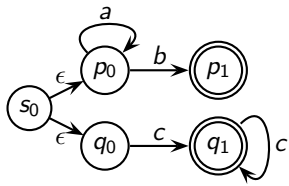
Constructing DFA for Multiple Patterns: Example 1



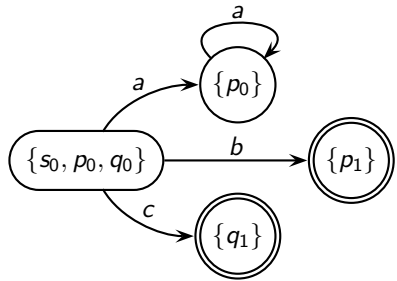
State	*Transition*		
	a	*b*	*c*
$\{s_0, p_0, q_0\}$	$\{p_0\}$	$\{p_1\}$	$\{q_1\}$
$\{p_0\}$			
$\{p_1\}$			
$\{q_1\}$			



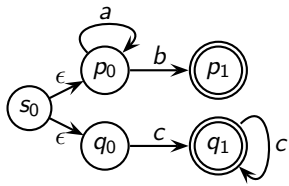
Constructing DFA for Multiple Patterns: Example 1



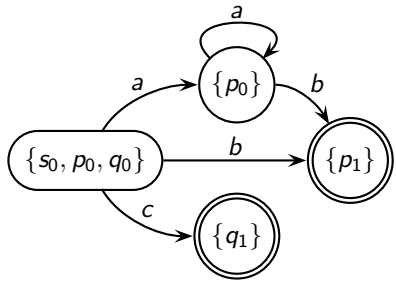
State	*Transition*		
	a	*b*	*c*
$\{s_0, p_0, q_0\}$	$\{p_0\}$	$\{p_1\}$	$\{q_1\}$
$\{p_0\}$	$\{p_0\}$		
$\{p_1\}$			
$\{q_1\}$			



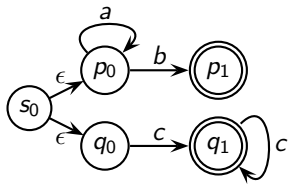
Constructing DFA for Multiple Patterns: Example 1



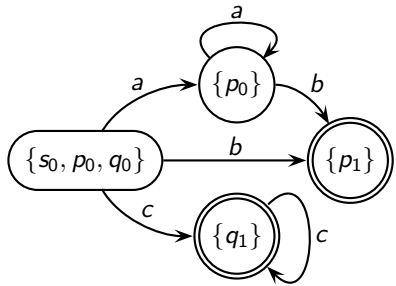
State	*Transition*		
	a	*b*	*c*
$\{s_0, p_0, q_0\}$	$\{p_0\}$	$\{p_1\}$	$\{q_1\}$
$\{p_0\}$	$\{p_0\}$	$\{p_1\}$	
$\{p_1\}$			
$\{q_1\}$			



Constructing DFA for Multiple Patterns: Example 1



State	Transition		
	a	b	c
$\{s_0, p_0, q_0\}$	$\{p_0\}$	$\{p_1\}$	$\{q_1\}$
$\{p_0\}$	$\{p_0\}$	$\{p_1\}$	
$\{p_1\}$			
$\{q_1\}$			$\{q_1\}$



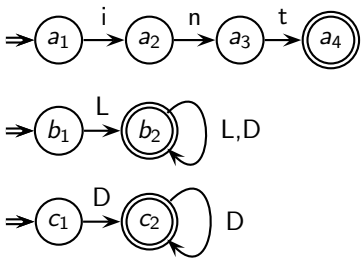
Constructing DFA for Multiple Patterns: Example 2

Let L and D denote the set of all letters and digits, respectively

Pattern	Token
int	INT
$L(L\|D)^*$	ID
D^+	NUM
=	=
;	;

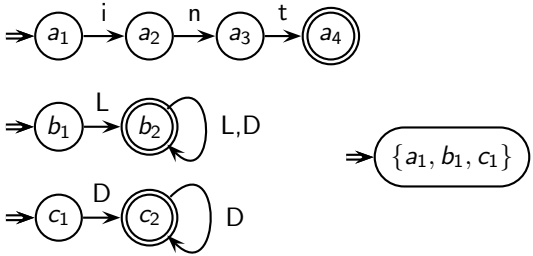
For convenience, we will ignore the last two patterns that are completely independent

Constructing DFA for Multiple Patterns: Example 2



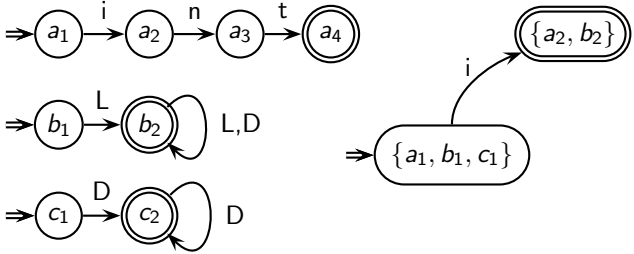
State	i	n	t	$L-\{i,n,t\}$	D

Constructing DFA for Multiple Patterns: Example 2



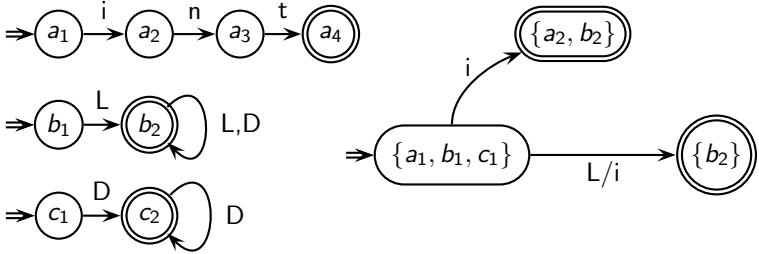
State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$					

Constructing DFA for Multiple Patterns: Example 2



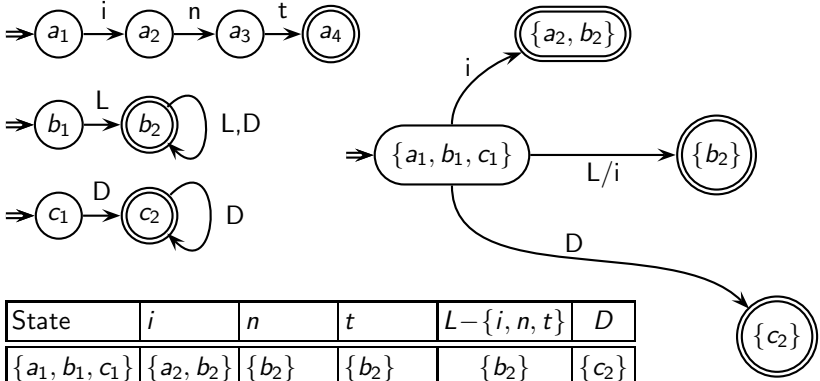
State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$				
$\{a_2, b_2\}$					

Constructing DFA for Multiple Patterns: Example 2



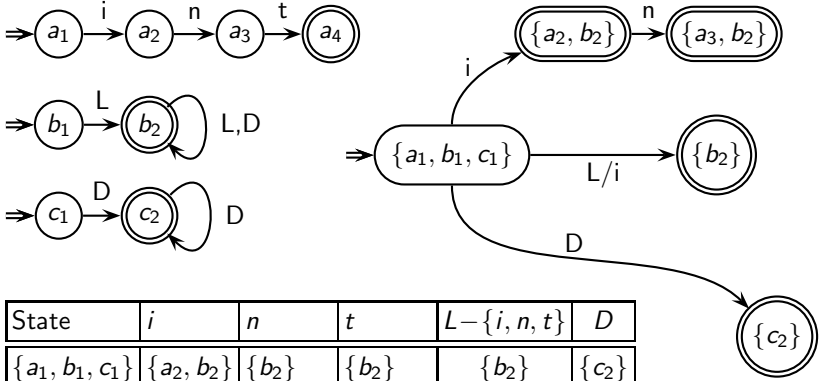
State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	
$\{a_2, b_2\}$					
$\{b_2\}$					

Constructing DFA for Multiple Patterns: Example 2



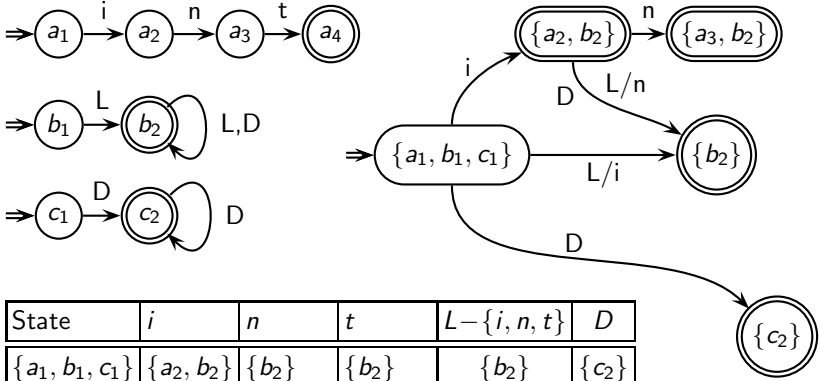
State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{c_2\}$
$\{a_2, b_2\}$					
$\{b_2\}$					
$\{c_2\}$					

Constructing DFA for Multiple Patterns: Example 2



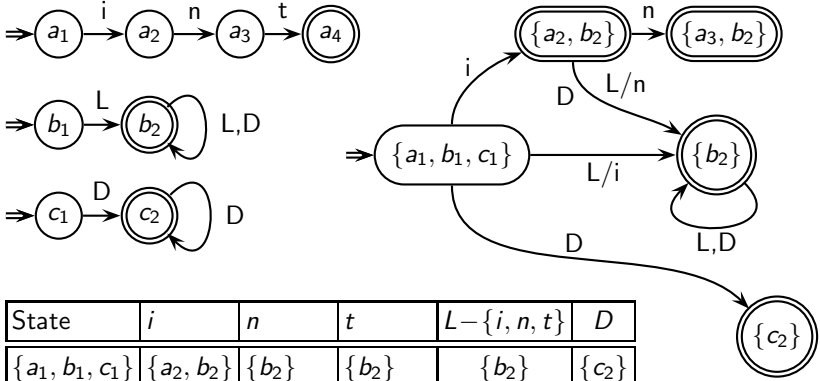
State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{c_2\}$
$\{a_2, b_2\}$		$\{a_3, b_2\}$			
$\{b_2\}$					
$\{c_2\}$					

Constructing DFA for Multiple Patterns: Example 2



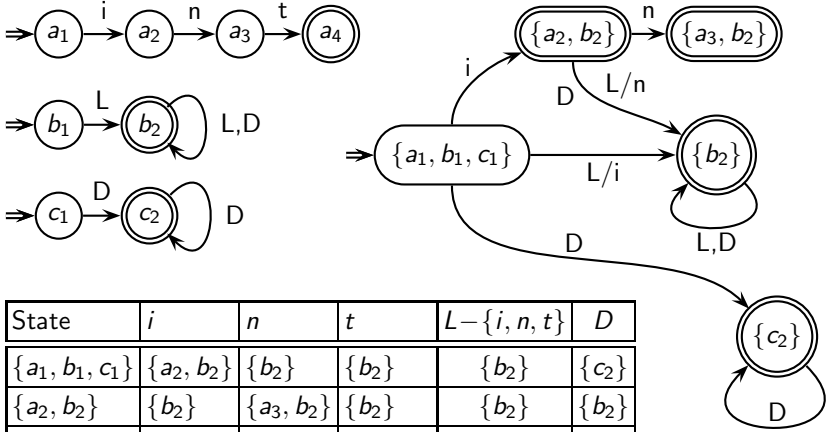
State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{c_2\}$
$\{a_2, b_2\}$	$\{b_2\}$	$\{a_3, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{b_2\}$					
$\{c_2\}$					
$\{a_3, b_2\}$					

Constructing DFA for Multiple Patterns: Example 2



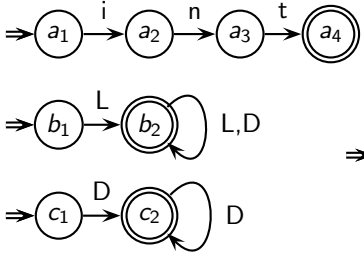
State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{c_2\}$
$\{a_2, b_2\}$	$\{b_2\}$	$\{a_3, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{c_2\}$					
$\{a_3, b_2\}$					

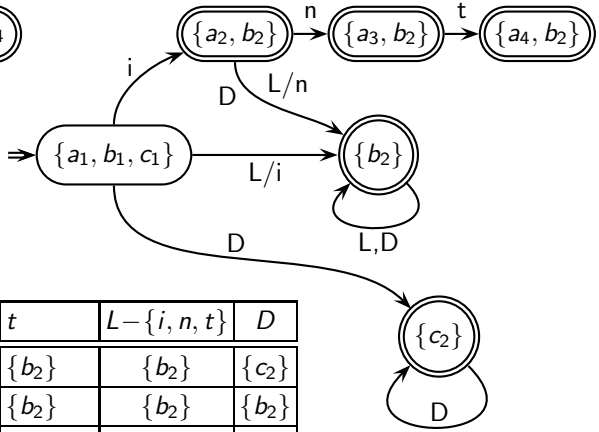
Constructing DFA for Multiple Patterns: Example 2



State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{c_2\}$
$\{a_2, b_2\}$	$\{b_2\}$	$\{a_3, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{c_2\}$					$\{c_2\}$
$\{a_3, b_2\}$					

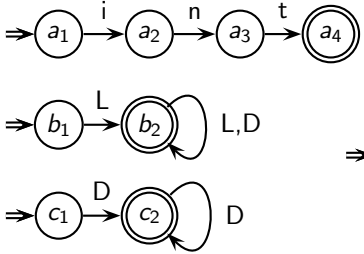
Constructing DFA for Multiple Patterns: Example 2

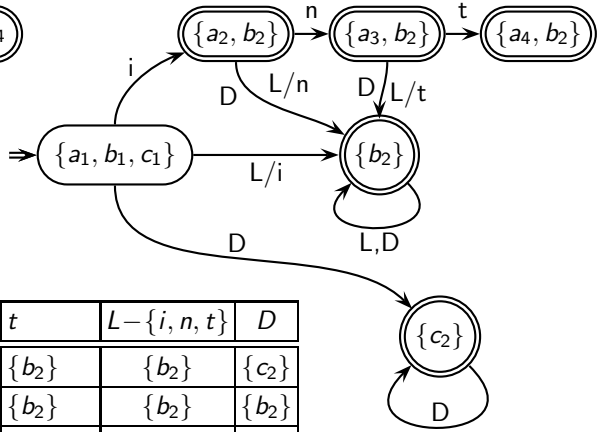




State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{c_2\}$
$\{a_2, b_2\}$	$\{b_2\}$	$\{a_3, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{c_2\}$					$\{c_2\}$
$\{a_3, b_2\}$			$\{a_4, b_2\}$		
$\{a_4, b_2\}$					

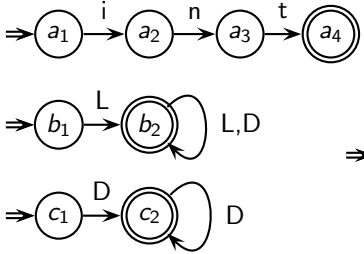
Constructing DFA for Multiple Patterns: Example 2

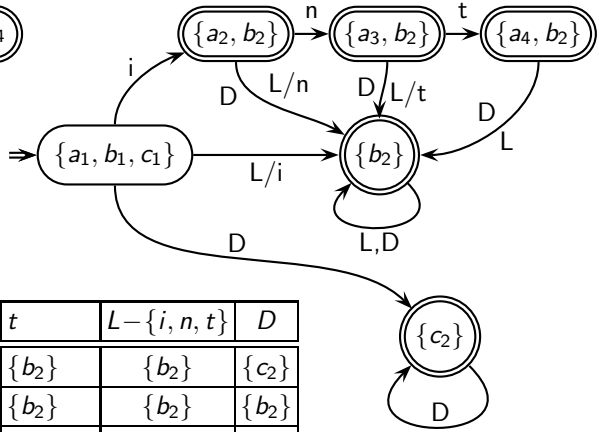




State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{c_2\}$
$\{a_2, b_2\}$	$\{b_2\}$	$\{a_3, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{c_2\}$					$\{c_2\}$
$\{a_3, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{a_4, b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{a_4, b_2\}$					

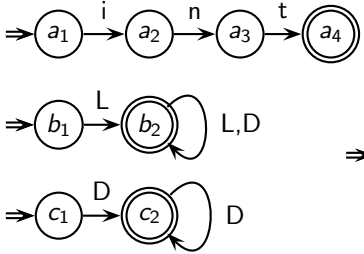
Constructing DFA for Multiple Patterns: Example 2

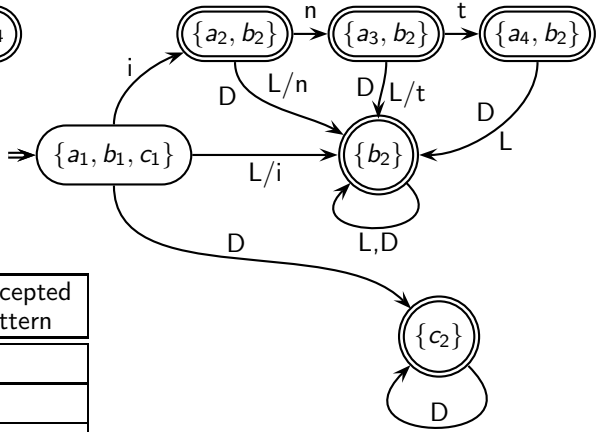




State	i	n	t	$L-\{i,n,t\}$	D
$\{a_1, b_1, c_1\}$	$\{a_2, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{c_2\}$
$\{a_2, b_2\}$	$\{b_2\}$	$\{a_3, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{c_2\}$					$\{c_2\}$
$\{a_3, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{a_4, b_2\}$	$\{b_2\}$	$\{b_2\}$
$\{a_4, b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$

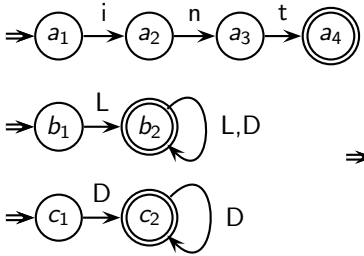
Constructing DFA for Multiple Patterns: Example 2

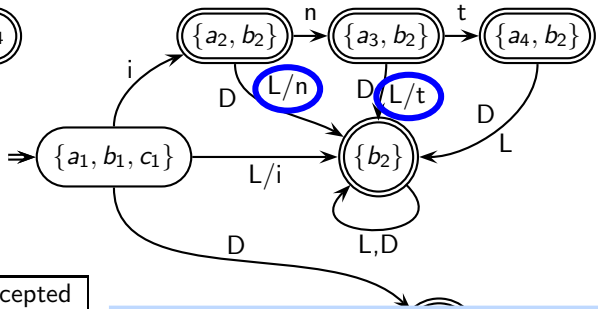




State	Possible Patterns	Accepted Pattern
$\{a_1, b_1, c_1\}$		
$\{a_2, b_2\}$	ID	ID
$\{b_2\}$	ID	ID
$\{c_2\}$	NUM	NUM
$\{a_3, b_2\}$	ID	ID
$\{a_4, b_2\}$	INT, ID	INT

Constructing DFA for Multiple Patterns: Example 2





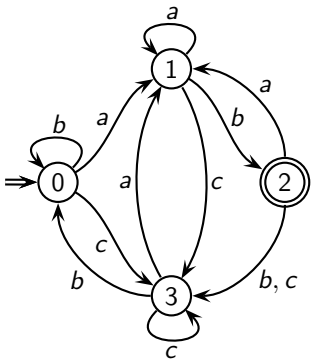
State	Possible Patterns	Accepted Pattern
$\{a_1, b_1, c_1\}$		
$\{a_2, b_2\}$	ID	ID
$\{b_2\}$	ID	ID
$\{c_2\}$	NUM	NUM
$\{a_3, b_2\}$	ID	ID
$\{a_4, b_2\}$	INT, ID	INT

Longest match. Lexeme "int" reaches state $\{a_4, b_2\}$ whereas lexeme "integer" reaches the state $\{b_2\}$

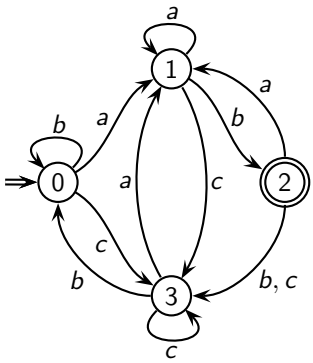
First maching rule preferred. Transitions L/n and L/t to state $\{b_2\}$ ensure that INT is prefered over ID for the lexeme "int"

Representing DFAs Using Four Arrays

DFA to be Represented Using Four Arrays: Example 1



DFA to be Represented Using Four Arrays: Example 1

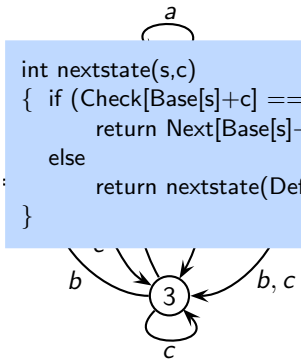


	a	b	c
0	1	0	3
1	1	2	3
2	1	3	3
3	1	0	3

DFA to be Represented Using Four Arrays: Example 1

```
int nextstate(s,c)
{  if (Check[Base[s]+c] == s)
        return Next[Base[s]+c];
   else
        return nextstate(Default[s],c);
}
```

States 0 and 3 have identical transitions. Transitions in states 1 and 2 differ from them only on b.



Char	Code
a	0
b	1
c	2

	a	b	c
0	1	0	3
1	1	2	3
2	1	3	3
3	1	0	3

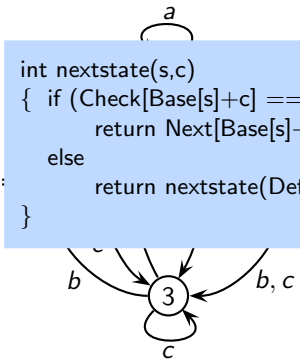
State	Default	Base
0		
1		
2		
3		

	Next	Check
0		
1		
2		
3		
4		
5		

DFA to be Represented Using Four Arrays: Example 1

```
int nextstate(s,c)
{  if (Check[Base[s]+c] == s)
        return Next[Base[s]+c];
   else
        return nextstate(Default[s],c);
}
```

We choose to fill the entries for state 0 first (state 3 could also have been used)



Char	Code
a	0
b	1
c	2

	a	b	c
0	1	0	3
1	1	2	3
2	1	3	3
3	1	0	3

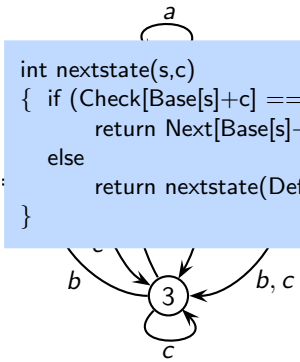
State	Default	Base
0		
1		
2		
3		

	Next	Check
0		
1		
2		
3		
4		
5		

DFA to be Represented Using Four Arrays: Example 1

```
int nextstate(s,c)
{  if (Check[Base[s]+c] == s)
        return Next[Base[s]+c];
   else
        return nextstate(Default[s],c);
}
```

The Check array contains 0 to confirm that the corresponding entries in the next array are for state 0



Char	Code
a	0
b	1
c	2

	a	b	c
0	1	0	3
1	1	2	3
2	1	3	3
3	1	0	3

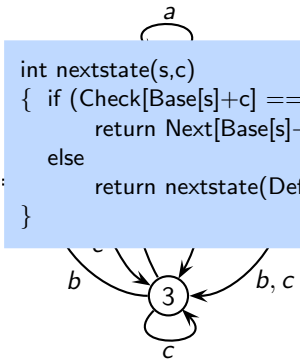
State	Default	Base
0		
1		
2		
3		

	Next	Check
0	1	0
1	0	0
2	3	0
3		
4		
5		

DFA to be Represented Using Four Arrays: Example 1

```
int nextstate(s,c)
{  if (Check[Base[s]+c] == s)
        return Next[Base[s]+c];
   else
        return nextstate(Default[s],c);
}
```

The Check array contains 0 to confirm that the corresponding entries in the next array are for state 0



Char	Code
a	0
b	1
c	2

	a	b	c
0	1	0	3
1	1	2	3
2	1	3	3
3	1	0	3

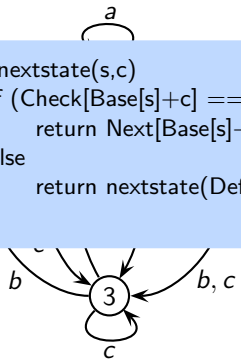
State	Default	Base
0		
1		
2		
3		

	Next	Check
0	1	0
1	0	0
2	3	0
3		
4		
5		

DFA to be Represented Using Four Arrays: Example 1

```
int nextstate(s,c)
{  if (Check[Base[s]+c] ==
        return Next[Base[s]+
   else
        return nextstate(Default[s],c);
}
```

For state 1, we reuse the transitions on a and c from state 0 but need to enter transition on b explicitly. We do this using the next free entry (index 3) in the next array and back calculating the base of state 1.



Char	Code
a	0
b	1
c	2

	a	b	c
0	1	0	3
1	1	2	3
2	1	3	3
3	1	0	3

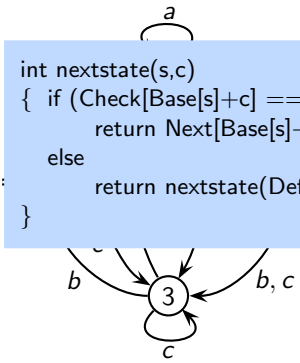
State	Default	Base
0		
1	0	
2		
3		

	Next	Check
0	1	0
1	0	0
2	3	0
3	2	1
4		
5		

DFA to be Represented Using Four Arrays: Example 1

```
int nextstate(s,c)
{  if (Check[Base[s]+c] == s
        return Next[Base[s]+c
   else
        return nextstate(Default[s],c);
}
```

The variation in state 2 is similar to that for state 1. We reuse the transitions on a and c from state 0 but enter transition on b explicitly in the next free entry (index 4) in the next array and back-calculate the base of state 2.



Char	Code
a	0
b	1
c	2

	a	b	c
0	1	0	3
1	1	2	3
2	1	3	3
3	1	0	3

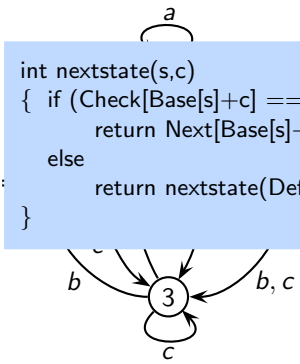
State	Default	Base
0		
1	0	
2	0	
3		

	Next	Check
0	1	0
1	0	0
2	3	0
3	2	1
4	3	2
5		

DFA to be Represented Using Four Arrays: Example 1

```
int nextstate(s,c)
{  if (Check[Base[s]+c] == s
        return Next[Base[s]+c
   else
        return nextstate(Default[s],c);
}
```

State 3 is identical to state 0. We have shown here its base as same as for state 0.
(In practice, lex begins the entries from index 1 and keeps index 0 free for such entries. We have ignored it because it is a matter of details.)



Char	Code
a	0
b	1
c	2

	a	b	c
0	1	0	3
1	1	2	3
2	1	3	3
3	1	0	3

State	Default	Base
0		
1	0	
2	0	
3	0	

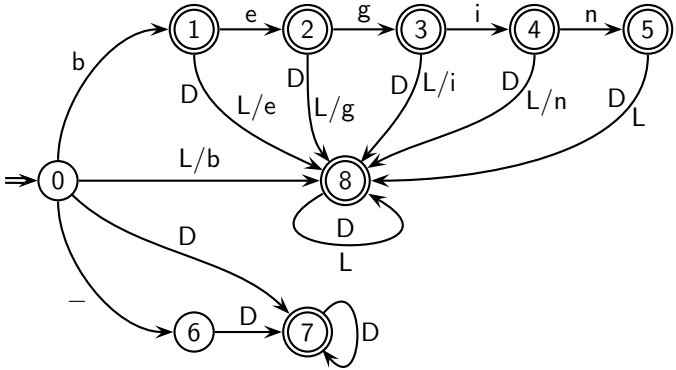
	Next	Check
0	1	0
1	0	0
2	3	0
3	2	1
4	3	2
5		

The Intuition Behind Four Array Representation

How to find the appropriate space in Next array for a state?

- View the entries (in the row of the state) that are required to be stored as "pins" separated by the entries that are not required to be stored
- View the positions in the Next array that do not contain a transition as "holes"
- Try to match the pattern (i.e. separation) of pins with that of the available holes

DFA to be Represented Using Four Arrays: Example 2



Set	Characters
L	a to z
D	0 to 9

Pattern	Token
begin	BEGIN
L(L|D)*	ID
(-|ϵ)D$^+$	NUM

Table Representation for Example 2

In the following, L denotes any letter from a to z other than b, e, g, i, n because these letters are listed seperately

	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	

Character	Code
a - z	0 - 25
0 - 9	26 - 35
-	36

Choice of Default States for Example 2

- States 8 and 6 are represented independently
- State 6 is the default state for state 7
- State 8 is the default state for all other states

Four Arrays Representation for Example 2

S: State
D: Default
B: Base
N: Next
C: Check

S	D	B
0		
1		
2		
3		
4		
5		
6		
7		
8		

	N	C
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		

	N	C
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		

	N	C
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		

	N	C
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		

Four Arrays Representation for Example 2

S: State
D: Default
B: Base
N: Next
C: Check

S	D	B
0		
1		
2		
3		
4		
5		
6		
7		
8		

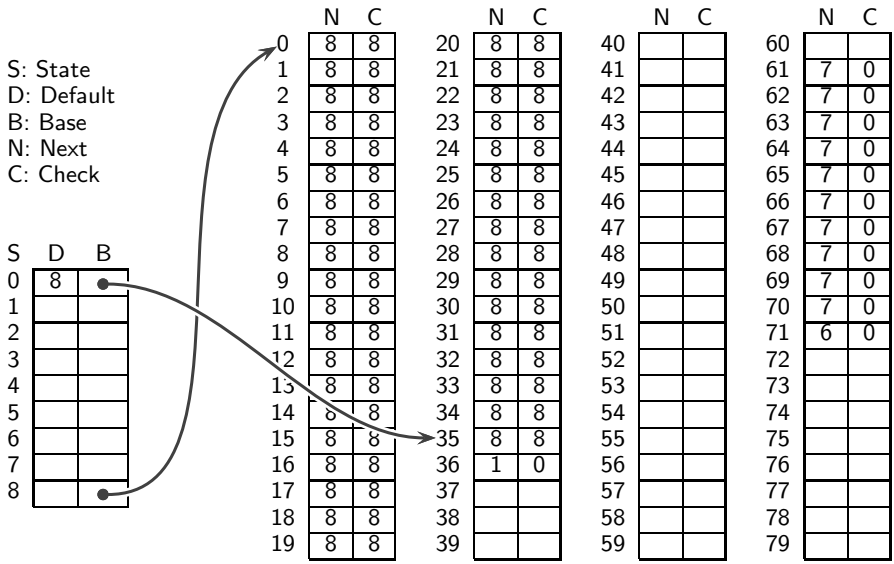
	N	C
0	8	8
1	8	8
2	8	8
3	8	8
4	8	8
5	8	8
6	8	8
7	8	8
8	8	8
9	8	8
10	8	8
11	8	8
12	8	8
13	8	8
14	8	8
15	8	8
16	8	8
17	8	8
18	8	8
19	8	8

	N	C
20	8	8
21	8	8
22	8	8
23	8	8
24	8	8
25	8	8
26	8	8
27	8	8
28	8	8
29	8	8
30	8	8
31	8	8
32	8	8
33	8	8
34	8	8
35	8	8
36		
37		
38		
39		

	N	C
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		

	N	C
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		

Four Arrays Representation for Example 2



Four Arrays Representation for Example 2

S: State
D: Default
B: Base
N: Next
C: Check

S	D	B
0	8	
1	8	
2		
3		
4		
5		
6		
7		
8		

	N	C
0	8	8
1	8	8
2	8	8
3	8	8
4	8	8
5	8	8
6	8	8
7	8	8
8	8	8
9	8	8
10	8	8
11	8	8
12	8	8
13	8	8
14	8	8
15	8	8
16	8	8
17	8	8
18	8	8
19	8	8

	N	C
20	8	8
21	8	8
22	8	8
23	8	8
24	8	8
25	8	8
26	8	8
27	8	8
28	8	8
29	8	8
30	8	8
31	8	8
32	8	8
33	8	8
34	8	8
35	8	8
36	1	0
37	2	1
38		
39		

	N	C
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		

	N	C
60		
61	7	0
62	7	0
63	7	0
64	7	0
65	7	0
66	7	0
67	7	0
68	7	0
69	7	0
70	7	0
71	6	0
72		
73		
74		
75		
76		
77		
78		
79		

Four Arrays Representation for Example 2

S: State
D: Default
B: Base
N: Next
C: Check

S	D	B
0	8	
1	8	
2	8	
3		
4		
5		
6		
7		
8		

	N	C
0	8	8
1	8	8
2	8	8
3	8	8
4	8	8
5	8	8
6	8	8
7	8	8
8	8	8
9	8	8
10	8	8
11	8	8
12	8	8
13	8	8
14	8	8
15	8	8
16	8	8
17	8	8
18	8	8
19	8	8

	N	C
20	8	8
21	8	8
22	8	8
23	8	8
24	8	8
25	8	8
26	8	8
27	8	8
28	8	8
29	8	8
30	8	8
31	8	8
32	8	8
33	8	8
34	8	8
35	8	8
36	1	0
37	2	1
38	3	2
39		

	N	C
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		

	N	C
60		
61	7	0
62	7	0
63	7	0
64	7	0
65	7	0
66	7	0
67	7	0
68	7	0
69	7	0
70	7	0
71	6	0
72		
73		
74		
75		
76		
77		
78		
79		

Four Arrays Representation for Example 2

S: State
D: Default
B: Base
N: Next
C: Check

S	D	B
0	8	
1	8	
2	8	
3	8	
4		
5		
6		
7		
8		

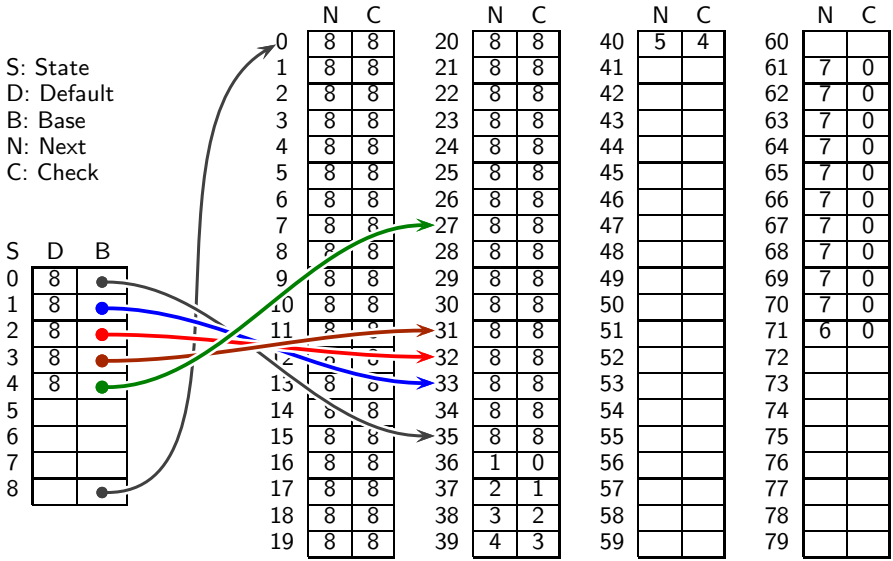
	N	C
0	8	8
1	8	8
2	8	8
3	8	8
4	8	8
5	8	8
6	8	8
7	8	8
8	8	8
9	8	8
10	8	8
11	8	8
12	8	8
13	8	8
14	8	8
15	8	8
16	8	8
17	8	8
18	8	8
19	8	8

	N	C
20	8	8
21	8	8
22	8	8
23	8	8
24	8	8
25	8	8
26	8	8
27	8	8
28	8	8
29	8	8
30	8	8
31	8	8
32	8	8
33	8	8
34	8	8
35	8	8
36	1	0
37	2	1
38	3	2
39	4	3

	N	C
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		

	N	C
60		
61	7	0
62	7	0
63	7	0
64	7	0
65	7	0
66	7	0
67	7	0
68	7	0
69	7	0
70	7	0
71	6	0
72		
73		
74		
75		
76		
77		
78		
79		

Four Arrays Representation for Example 2



Four Arrays Representation for Example 2

S: State
D: Default
B: Base
N: Next
C: Check

S	D	B
0	8	
1	8	
2	8	
3	8	
4	8	
5	8	
6		
7		
8		

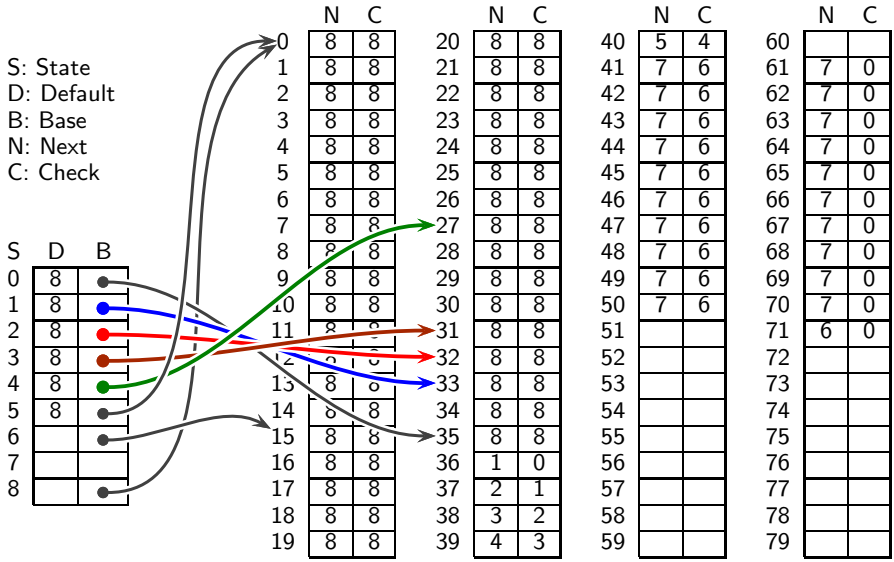
	N	C
0	8	8
1	8	8
2	8	8
3	8	8
4	8	8
5	8	8
6	8	8
7	8	8
8	8	8
9	8	8
10	8	8
11	8	8
12	8	8
13	8	8
14	8	8
15	8	8
16	8	8
17	8	8
18	8	8
19	8	8

	N	C
20	8	8
21	8	8
22	8	8
23	8	8
24	8	8
25	8	8
26	8	8
27	8	8
28	8	8
29	8	8
30	8	8
31	8	8
32	8	8
33	8	8
34	8	8
35	8	8
36	1	0
37	2	1
38	3	2
39	4	3

	N	C
40	5	4
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		

	N	C
60		
61	7	0
62	7	0
63	7	0
64	7	0
65	7	0
66	7	0
67	7	0
68	7	0
69	7	0
70	7	0
71	6	0
72		
73		
74		
75		
76		
77		
78		
79		

Four Arrays Representation for Example 2



Four Arrays Representation for Example 2

S: State
D: Default
B: Base
N: Next
C: Check

S	D	B
0	8	
1	8	
2	8	
3	8	
4	8	
5	8	
6		
7	6	
8		

	N	C
0	8	8
1	8	8
2	8	8
3	8	8
4	8	8
5	8	8
6	8	8
7	8	8
8	8	8
9	8	8
10	8	8
11	8	8
12	8	8
13	8	8
14	8	8
15	8	8
16	8	8
17	8	8
18	8	8
19	8	8

	N	C
20	8	8
21	8	8
22	8	8
23	8	8
24	8	8
25	8	8
26	8	8
27	8	8
28	8	8
29	8	8
30	8	8
31	8	8
32	8	8
33	8	8
34	8	8
35	8	8
36	1	0
37	2	1
38	3	2
39	4	3

	N	C
40	5	4
41	7	6
42	7	6
43	7	6
44	7	6
45	7	6
46	7	6
47	7	6
48	7	6
49	7	6
50	7	6
51		
52		
53		
54		
55		
56		
57		
58		
59		

	N	C
60		
61	7	0
62	7	0
63	7	0
64	7	0
65	7	0
66	7	0
67	7	0
68	7	0
69	7	0
70	7	0
71	6	0
72		
73		
74		
75		
76		
77		
78		
79		

Size Comparison for Example 2

- Space for a 2 dimensional table

$$\text{rows} \times \text{columns} = 9 \times 36 = 324$$

- Space for four arrays representation

Array	Size
Next	71
Check	71
Default	9
Base	9
Total	160

- If a large graph seen as adjacency matrix is stored using four arrays, it would have the need of pointers and dynamic memory allocation

  This would imply good cache behaviour

Further Compression Using Equivalence Classes

- The four arrays handle similarity in the rows of the 2-D table
- Several columns could have a lot of similarity too
- We can define equivalence classes of characters that have identical transitions
  Identical columns are collapsed into a single column
- The equivalence classes are given contiguous codes thereby eliminating several "holes" in the Next and Check arrays

Further Compression Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	

```
int nextstate(s,c)
{  if (Check[Base[s]+c] == s)
        return Next[Base[s]+c];
   else
        return nextstate(Default[s],c);
}
```

Now c represents the class of a character instead of the character

Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	

S: State
D: Default
B: Base
N: Next
C: Check

S	D	B
0		
1		
2		
3		
4		
5		
6		
7		
8		

	N	C
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	

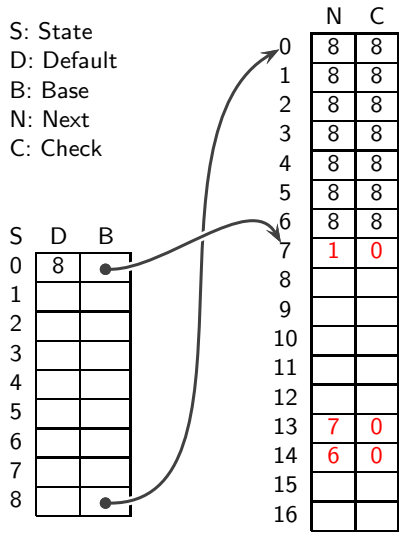
S: State
D: Default
B: Base
N: Next
C: Check

S	D	B
0		
1		
2		
3		
4		
5		
6		
7		
8		

	N	C
0	8	8
1	8	8
2	8	8
3	8	8
4	8	8
5	8	8
6	8	8
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

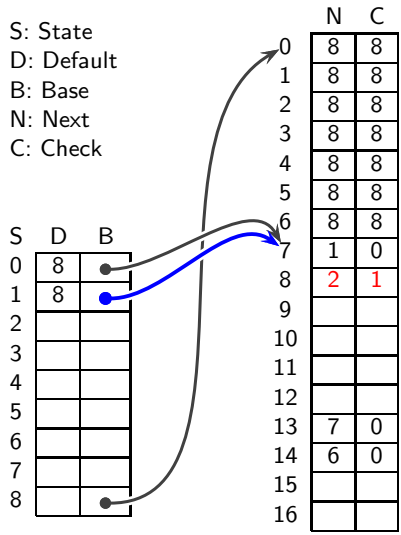
Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	



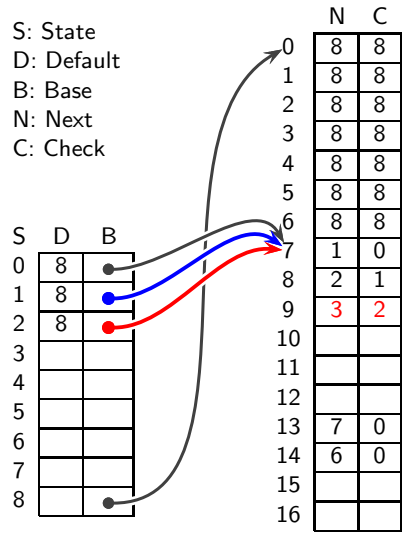
Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	



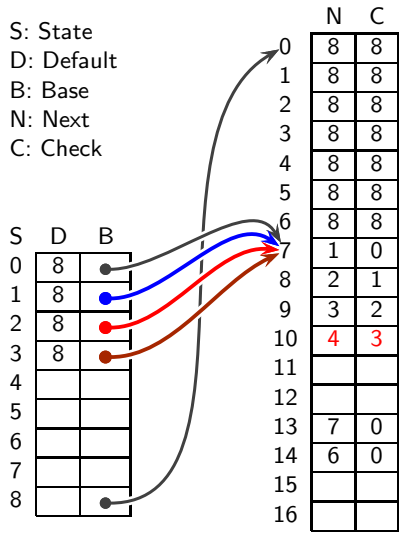
Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	



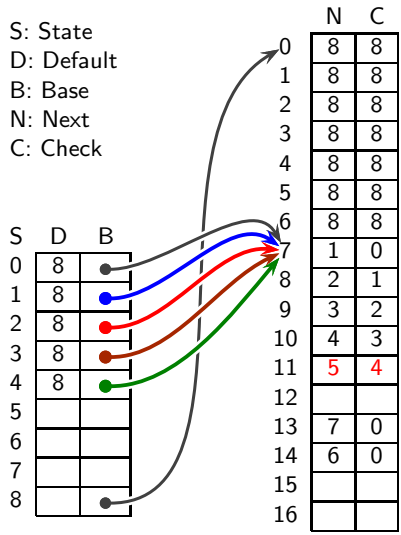
Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	



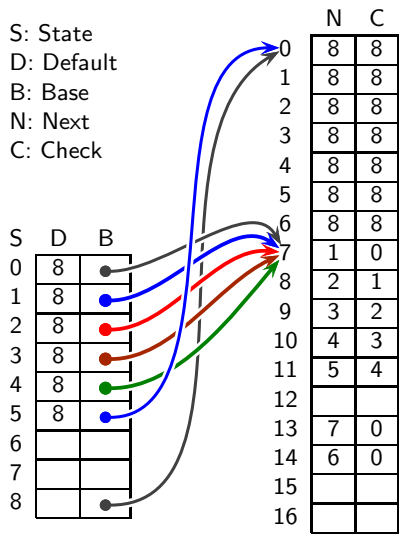
Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	



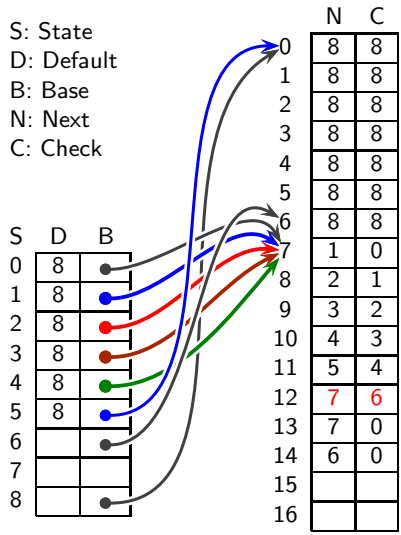
Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	



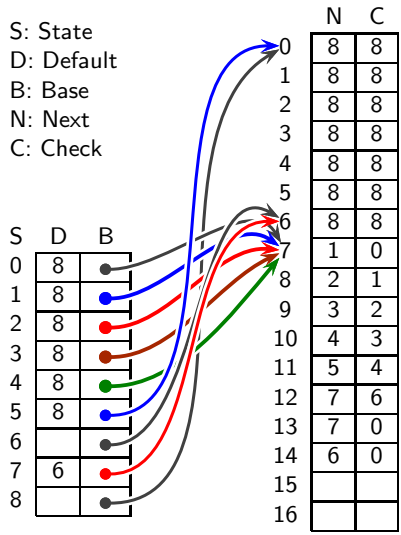
Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	



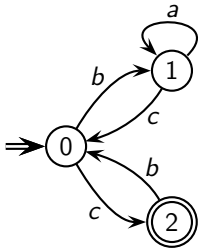
Four Arrays Representation Using Equivalence Classes for Example 2

EC →	0	1	2	3	4	5	6	7
	b	e	g	i	n	L	D	-
0	1	8	8	8	8	8	7	6
1	8	2	8	8	8	8	8	
2	8	8	3	8	8	8	8	
3	8	8	8	4	8	8	8	
4	8	8	8	8	5	8	8	
5	8	8	8	8	8	8	8	
6							7	
7							7	
8	8	8	8	8	8	8	8	



Tutorial Problem

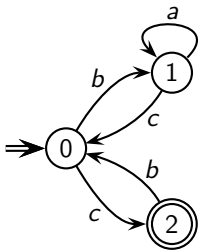
Represent the following DFA using 4-arrays notation as compactly as possible



Character	Code
a	0
b	1
c	2

Tutorial Problem

Represent the following DFA using 4-arrays notation as compactly as possible



Character	Code
a	0
b	1
c	2

State	Base	Default
0	2	
1	0	
2	0	

	Next	Check
0	1	1
1	0	2
2	0	1
3	1	0
4	2	0
5		

Minimizing DFAs

Minimizing DFAs

Prof. Sanyal's slides (scanning-slides-sanyal-part3.pdf)