

## Implementation of Lexical Analysis (Scanning)

Adapted from material by:

Prof. Alex Aiken and Prof. George Necula (UCB)

Prof. Saman Amarasinghe (MIT)

## Outline

- Specifying lexical structure using regular expressions
- Recognizing tokens using finite automata
  - Deterministic Finite Automata (DFAs)
  - Non-deterministic Finite Automata (NFAs)
- Implementation of regular expressions  
 RegExp  $\Rightarrow$  NFA  $\Rightarrow$  DFA  $\Rightarrow$  Tables

## Regular Expressions $\Rightarrow$ Lexical Spec.

1. Write a regular expression for the lexemes of each token
  - Number = digit + (Kleene plus)
  - Keyword = 'if' | 'else' | ... (Union)
  - Identifier = letter (letter | digit)\*
  - OpenPar = '('
  - ...
2. Construct  $R$ , matching all lexemes for all tokens  
 $R = \text{Keyword} \mid \text{Identifier} \mid \text{Number} \mid \dots$   
 $= R_1 \mid R_2 \mid \dots$

## (cont'd)

3. Let input be the sequence of characters  
 $x_1 \dots x_n$ 
  - For each  $1 \leq i \leq n$ , check if  
 $x_1 \dots x_i \in L(R)$
4. It must be that  
 $x_1 \dots x_i \in L(R_j)$  for some  $j$
5. Remove  $x_1 \dots x_i$  from input and go to (3)

**Problem:**

There are ambiguities in the algorithm.

## Ambiguities



- How much input is used? What if
  - $x_1 \dots x_i \in L(R)$  and also
  - $x_1 \dots x_k \in L(R)$
  - **Rule: Pick longest possible string in L(R)**
    - ❖ The “maximal munch” << vs <, != vs !
- Which token is used? What if
  - $x_1 \dots x_i \in L(R_j)$  and also
  - $x_1 \dots x_i \in L(R_k)$
  - **Rule: use rule listed first (j if j < k)**
    - ❖ Treats “if” as a keyword not an identifier

CS780(Prasad)

L4Lexer

5

## Error Handling



- What if
  - No rule matches a prefix of input ?
- *Problem:* Can get stuck ...
- *Solution:*
  - Write a rule matching all “bad” strings
  - Put it last (*catch-all clause*)

CS780(Prasad)

L4Lexer

6

## Summary



- Regular expressions provide a concise notation for string patterns.
- Use in lexical analysis requires small extensions:
  - To resolve ambiguities
  - To handle errors
- Good algorithms known
  - Require only single pass over the input
  - Few operations per character (table lookup)

CS780(Prasad)

L4Lexer

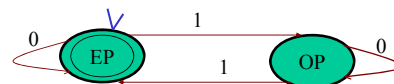
7

## Parity Problem



$\Sigma = \{0,1\} \quad \omega \in \Sigma^*$   
 $parity: \Sigma^* \rightarrow boolean$   
 $parity(\omega) \Leftrightarrow \omega$  contains even number of 1s.

Finite automaton = Recognizer



CS780(Prasad)

L4Lexer

8

### Basic Features

- Consumes the entire input string.
- Remembers the parity of the bit string by abstracting from the number of 1s in the string.
- Finite amount of memory required for this purpose.
  - Observe that counting requires unbounded memory, while computing the parity requires very small and fixed amount of memory.
- Accepts/Rejects the input in a deterministic fashion.

### Deterministic Finite State Automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

$Q$ : Finite set of states

$\Sigma$ : Finite Alphabet

$\delta$ : Transition function

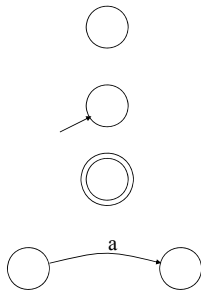
total function from  $Q \times \Sigma$  to  $Q$

$q_0$ : Initial/Start State

$F$ : Set of final/accepting state

### Finite Automata State Graphs

- State
- The start state
- An accepting state
- A transition



### Operation of the machine



- Read the current letter of input under the tape head.
- Transit to a new state depending on the current input and the current state, as dictated by the transition function.
- Halt after consuming the entire input.

### Associating Language with DFA

- Machine configuration:

$$[q, \omega] \quad \text{where } q \in Q, \omega \in \Sigma^*$$

- Yields relation:

$$[q, a\omega] \mapsto_M [\delta(q, a), \omega]$$

- Language:

$$\{\omega \in \Sigma^* \mid \underbrace{[q_0, \omega] \mapsto_M^* [q, \varepsilon]} \wedge q \in F\}$$

CS780(Prasad)

L4Lexer

13

### Example

- Set of strings over  $\{a,b\}$  that contain  $bb$

$$(a|b)^*bb(a|b)^*$$

- Design states by partitioning  $\Sigma^*$ .

□ Strings containing  $bb$   $q2$

□ Strings not containing  $bb$

✦ Strings that end in  $b$   $q1$

✦ Strings that do not end in  $b$   $q0$

➤ Initial state:  $q0$

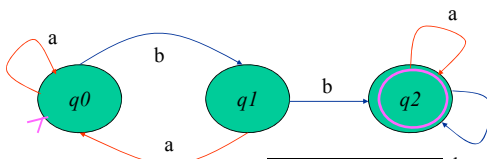
➤ Final state:  $q2$

CS780(Prasad)

L4Lexer

14

### State Diagram and Table



$$Q = \{q0, q1, q2\}$$

$$\Sigma = \{a, b\}$$

$$F = \{q2\}$$

$$[q0, aab] \mapsto^* [q1, \varepsilon]$$

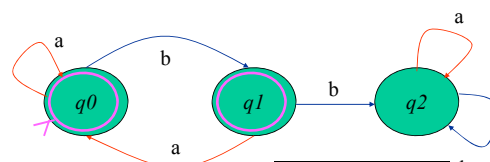
$\delta$	a	b
q0	q0	q1
q1	q0	q2
q2	q2	q2

CS780(Prasad)

L4Lexer

15

### Strings over $\{a,b\}$ that *do not* contain $bb$



$$Q = \{q0, q1, q2\}$$

$$\Sigma = \{a, b\}$$

$$F = \{q0, q1\}$$

$$[q0, ba] \mapsto^* [q0, \varepsilon]$$

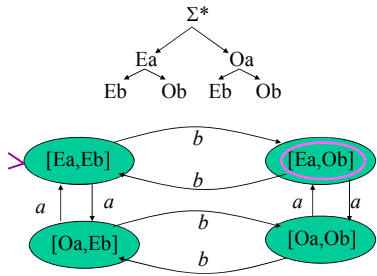
$\delta$	a	b
q0	q0	q1
q1	q0	q2
q2	q2	q2

CS780(Prasad)

L4Lexer

16

Strings over  $\{a,b\}$  containing even number of  $a$ 's and odd number of  $b$ 's.



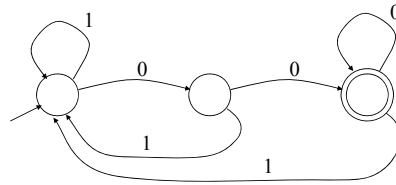
CS780(Prasad)

L4Lexer

17

**Example**

- Alphabet  $\{0,1\}$
- What language does this recognize?

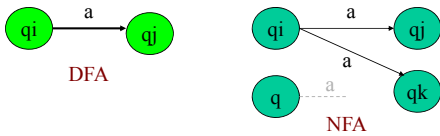


CS780(Prasad)

L4Lexer

18

Nondeterministic Finite Automata



$\delta_{DFA} : Q \times \Sigma \rightarrow Q$  total function

$\delta_{NFA} : Q \times \Sigma \rightarrow Pow(Q)$  total function

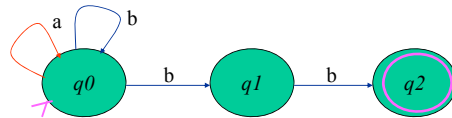
$\delta_{NFA} \subseteq Q \times \Sigma \times Q$  subset relation

CS780(Prasad)

L4Lexer

19

NFA State Diagram  
(Strings over  $\{a,b\}$  ending in  $bb$ )



$Q = \{q0, q1, q2\}$

$\Sigma = \{a, b\}$

$F = \{q2\}$


$(a|b)^*bb$

$\delta$	a	b
q0	{q0}	{q0, q1}
q1	$\phi$	{q2}
q2	$\phi$	$\phi$

CS780(Prasad)

L4Lexer

20



$[q_0, abb] \mapsto [q_0, bb] \mapsto [q_0, b] \mapsto [q_0, \varepsilon]$

$[q_0, abb] \mapsto [q_0, bb] \mapsto [q_0, b] \mapsto [q_1, \varepsilon]$


Halts in non-accepting state after consuming the input.

$[q_0, abb] \mapsto [q_0, bb] \mapsto [q_1, b] \mapsto [q_2, \varepsilon]$

Halts in accepting state after consuming the input.

$abb \in L(NFA)$

CS780(Prasad) L4Lexer 21




Introducing  $\varepsilon$ -transitions into NFA

$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q)$

- A  $\varepsilon$ -transition causes the machine to change its state non-deterministically, without consuming any input.

$L(DFAs) \subseteq L(NFAs)$


CS780(Prasad) L4Lexer 22



Deterministic and Nondeterministic Automata

- Deterministic Finite Automata (DFA)
  - One transition per input per state
  - No  $\varepsilon$ -moves
- Nondeterministic Finite Automata (NFA)
  - Can have multiple transitions for one input in a given state
  - Can have  $\varepsilon$ -moves
- Finite automata have finite memory
  - Need only to encode the current state

CS780(Prasad) L4Lexer 23



How do we associate a language with NFA?

- A DFA can take only one path through the state graph
- NFAs can choose
  - Whether to make  $\varepsilon$ -moves
  - Or one of the multiple transitions for a single input to take
    - ☐ Accept if *there exists* a accepting computation.
    - ☐ Reject if *all* computations are *non-accepting*.

CS780(Prasad) L4Lexer 24

- Every DFA is an NFA. However, does non-determinism make NFAs strictly more expressive (powerful) than DFAs?

$$L(NFAs) \supseteq L(DFAs)$$

- For type 0 languages (Turing Machines) and type 3 languages (regular languages), non-determinism does not add expressive power.
- For context-free languages and context-sensitive languages, non-determinism does enhance the expressive power.

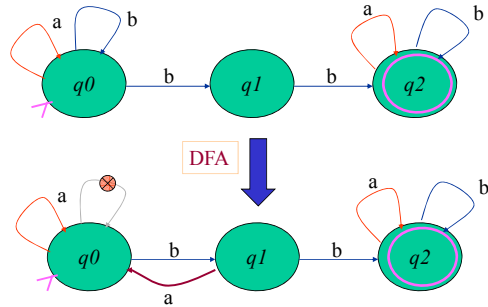
CS780(Prasad)

L4Lexter

25

### NFA State Diagram

$$(a | b)^* bb (a | b)^*$$



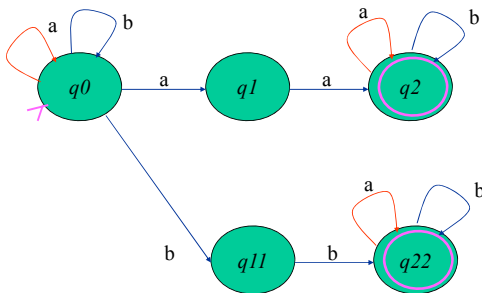
CS780(Prasad)

L4Lexter

26

### NFA for

$$(a | b)^* (aa | bb) (a | b)^*$$



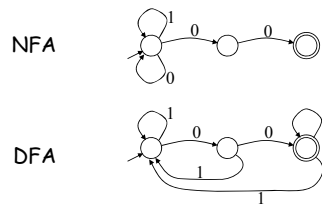
CS780(Prasad)

L4Lexter

27

### NFA vs. DFA

- Equivalent machines



CS780(Prasad)

L4Lexter

28

## NFA vs. DFA

- NFAs and DFAs recognize the same set of languages (regular languages).
- DFAs are faster to execute
  - There are no choices to consider.
- For a given language, NFA can be simpler than DFA
- DFA can be exponentially larger than NFA.

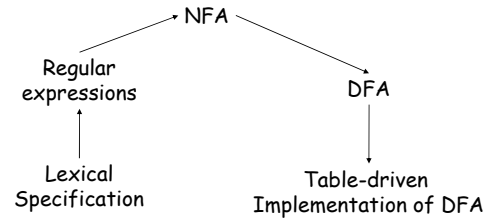
CS780(Prasad)

L4Lexer

29

## Reg. Expr. to Finite Automata

- High-level sketch



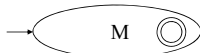
CS780(Prasad)

L4Lexer

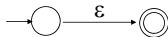
30

## Regular Expressions to NFA (1)

- For each kind of regular expr., define an NFA
  - Notation: NFA for rexp M



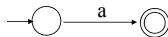
- For  $\epsilon$



- For  $\phi$



- For input a



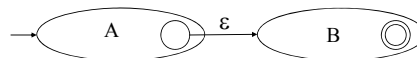
CS780(Prasad)

L4Lexer

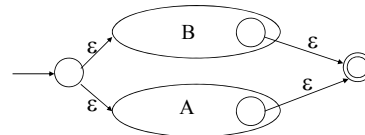
31

## Regular Expressions to NFA (2)

- For AB



- For  $A \mid B$



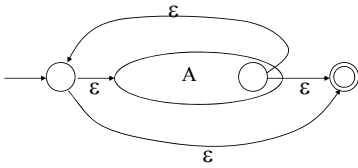
CS780(Prasad)

L4Lexer

32

### Regular Expressions to NFA (3)

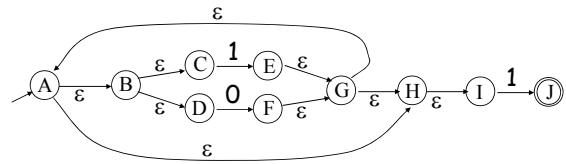
- For  $A^*$



Thompson's Construction

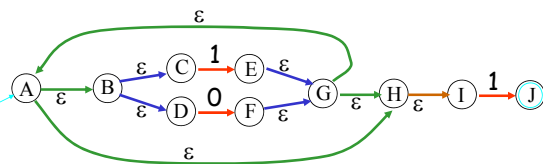
### Reg. Expr. $\rightarrow$ NFA conversion

- Consider the regular expression  $(1|0)^*1$
- The corresponding NFA is



### Regular Expression $\rightarrow$ NFA conversion

$(1|0)^*1$



### NFA to DFA: *The Trick*

- Simulate the NFA, *in parallel*
  - Michael Rabin and Dana Scott's work
- Each state of DFA = a non-empty subset of states of the NFA
- Start state = the set of NFA states reachable through  $\epsilon$ -moves from NFA start state
- Add a transition  $S \xrightarrow{a} S'$  to DFA *iff*  $S'$  is the set of NFA states reachable from some state in  $S$  after seeing the input 'a', considering  $\epsilon$ -moves as well

## NFA to DFA: Remark

- An NFA may be in many states at any time.
- How many different states ?
  - If there are N states, the NFA must be in some subset of those N states
- How many subsets are there?
  - $2^N - 1 =$  finitely many
- NFA  $\rightarrow$  DFA conversion is at the heart of tools such as `flex`. In practice, `flex`-like tools trade off speed for space in the choice of NFA and DFA representations. (*DFA Minimization*)

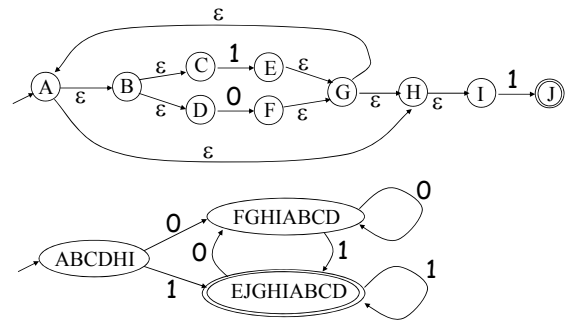
☐ Myhill-Nerode's work

CS780(Prasad)

L4Lexer

37

## NFA $\rightarrow$ DFA Example



CS780(Prasad)

L4Lexer

38

## Implementation

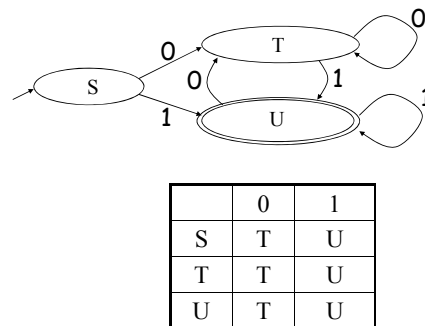
- A DFA can be implemented by a 2D table T
  - One dimension is "states"
  - Other dimension is "input symbol"
  - For every transition  $S_i \rightarrow^a S_k$  define  $T[i,a] = k$
- DFA "execution"
  - If in state  $S_i$  and input 'a', read  $T[i,a] = k$  and skip to state  $S_k$ 
    - ❖ Very efficient

CS780(Prasad)

L4Lexer

39

## Table Implementation of a DFA



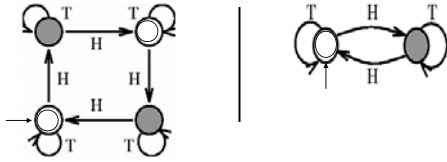
CS780(Prasad)

L4Lexer

40

## Finite State Machine Minimization

Language over  $\{H, T\}$  : Strings with even number of **H**



CS780(Prasad)

L4Lexer

41

## Minimizing DFA: *The Idea*

- Two states  $s$  and  $t$  should be merged unless there is some way to tell them apart.
  - Initially, assume that all states are equivalent until proven otherwise.
- How can we tell if two states  $s$  and  $t$  are different?
  - If one is accepting and the other is not.
  - If, on input  $c$ ,  $s \rightarrow x$  and  $t \rightarrow y$ , and we already know  $x$  is different from  $y$ .

CS780(Prasad)

L4Lexer

42

## Minimizing DFA: *The Algorithm*

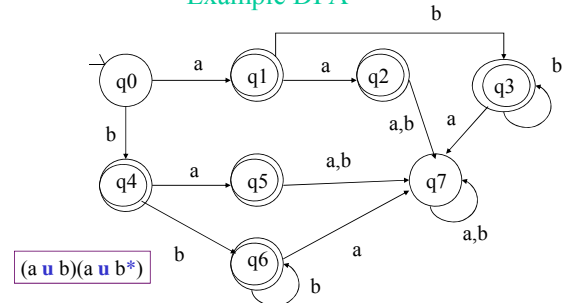
- Overall, it partitions the set of states.
- **Initial partition** :
  - (Accepting States, Non-accepting states)
- **Refinement** :
  - In each pass, find a new partition for each state such that
    - $s$  and  $t$  are in the same new partition *if and only if* states  $s$  and  $t$  were in the same old partition, *and*, on each input  $c$ , states  $s$  and  $t$  go to states in the same partition.

CS780(Prasad)

L4Lexer

43

## Example DFA



CS780(Prasad)

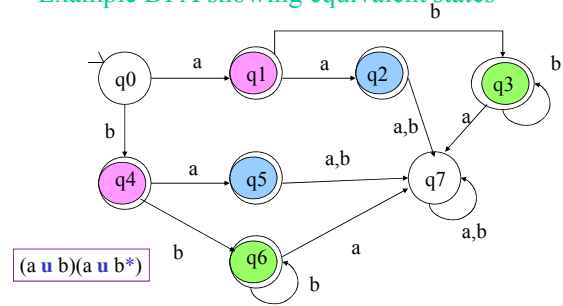
L4Lexer

44

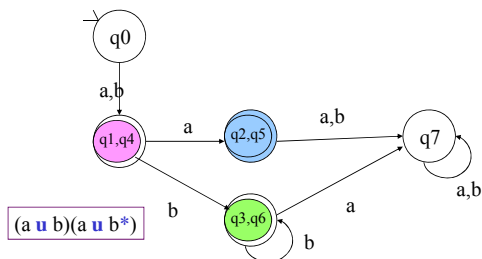
### Refinement of State Partitions

- $\{ \{q_0, q_7\}, \{q_1, q_2, q_3, q_4, q_5, q_6\} \}$
- $\{ \{q_0\}, \{q_7\}, \{q_1, q_2, q_3, q_4, q_5, q_6\} \}$ 
  - On any transition
- $\{ \{q_0\}, \{q_7\}, \{q_1, q_2, q_3, q_4, q_5, q_6\} \}$
- $\{ \{q_0\}, \{q_7\}, \{q_1, q_4\}, \{q_2, q_3, q_5, q_6\} \}$ 
  - On "a" transition
- $\{ \{q_0\}, \{q_7\}, \{q_1, q_4\}, \{q_2, q_5\}, \{q_3, q_6\} \}$ 
  - On "b" transition

### Example DFA showing equivalent states

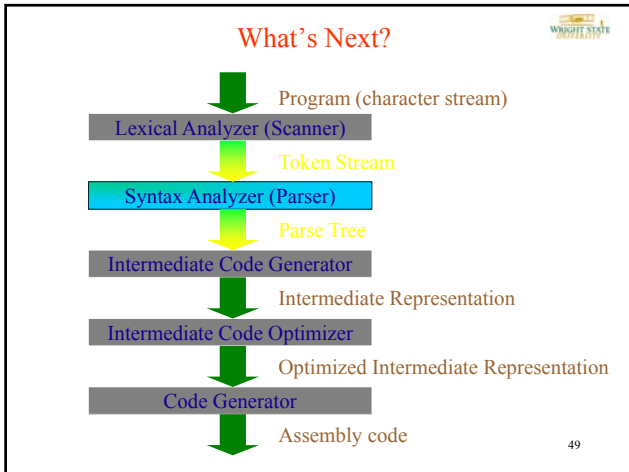


### Example Minimum DFA



### Summary

- Lexer creates tokens out of a text stream.
- Tokens are defined using regular expressions.
- Regular expressions can be mapped to Nondeterministic Finite Automata (NFA)
- NFA is transformed to a DFA
  - By removing non-determinism, and
  - By minimizing states.
- Executing a DFA is straightforward.
- Common scanner generator tools
  - lex, flex in C
  - jlex in Java



## Animating Lexical Analysis

CS780(Prasad)
L4L.exer
50

### Lexical Analyzer in Action

f	o	r		v	a	r	1		=		1	0			v	a	r	1		<	=	
---	---	---	--	---	---	---	---	--	---	--	---	---	--	--	---	---	---	---	--	---	---	--

**for**    ID("var1")   eq\_op   Num(10)   ID("var1")   leq\_op

CS780(Prasad)
L4L.exer
51

### Lexical Analyzer in Action

f	o	r		v	a	r	1		=		1	0			v	a	r	1		<	=	
---	---	---	--	---	---	---	---	--	---	--	---	---	--	--	---	---	---	---	--	---	---	--

**for**    ID("var1")   eq\_op   Num(10)   ID("var1")   leq\_op

CS780(Prasad)
L4L.exer
52

## Lexical Analyzer in Action

f o r   v a r l   =   1 0   v a r l   < =

for   ID("varl")   eq\_op   Num(10)   ID("varl")   leq\_op

- Partition input program text into subsequence of characters corresponding to tokens
- Attach the corresponding attributes to the tokens
- Eliminate white space and comments

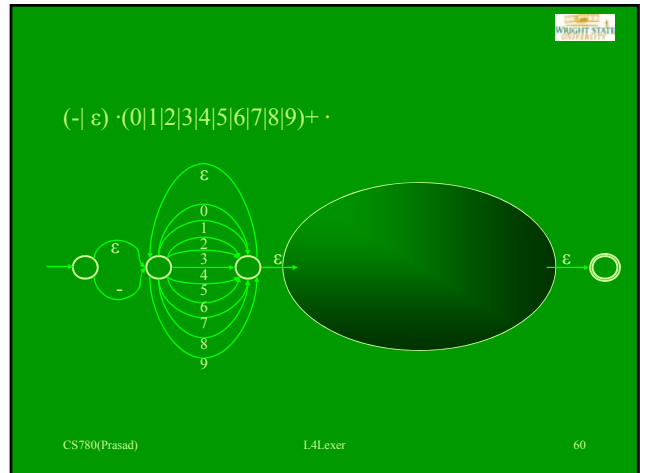
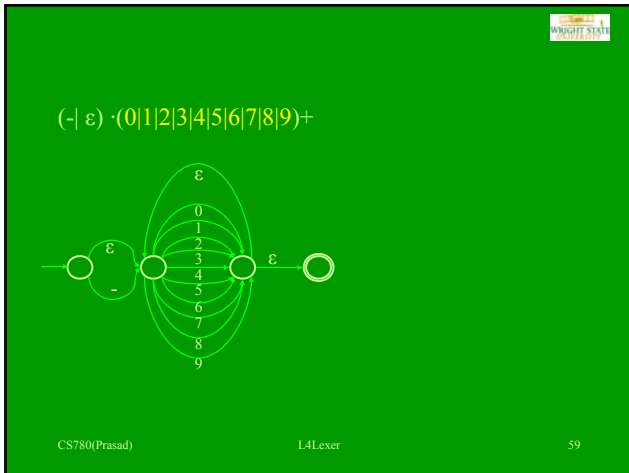
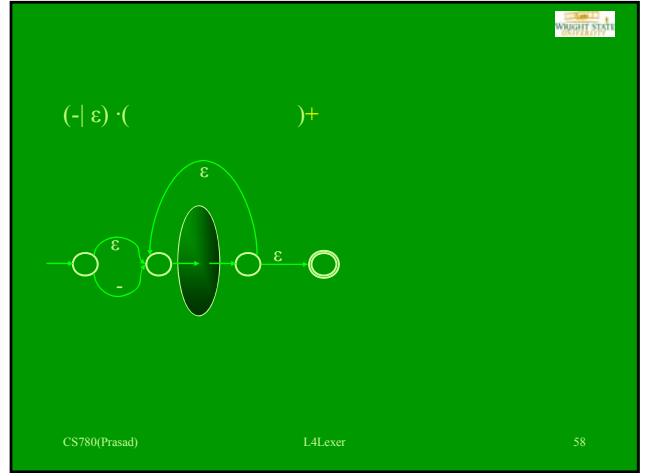
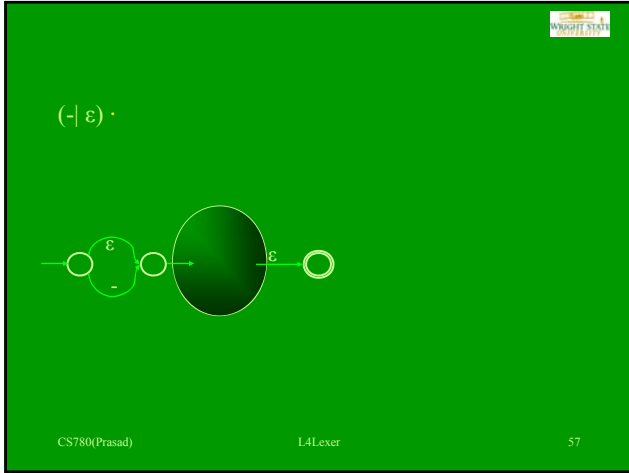
## Animating NFA construction

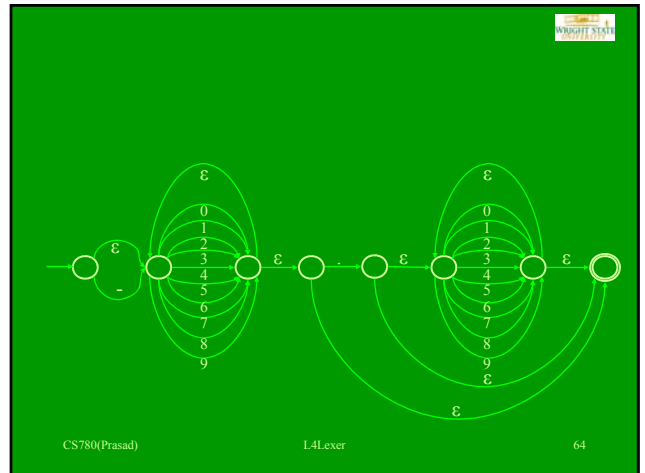
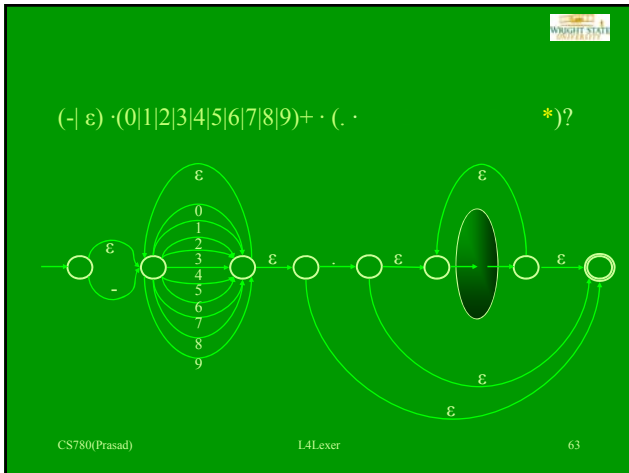
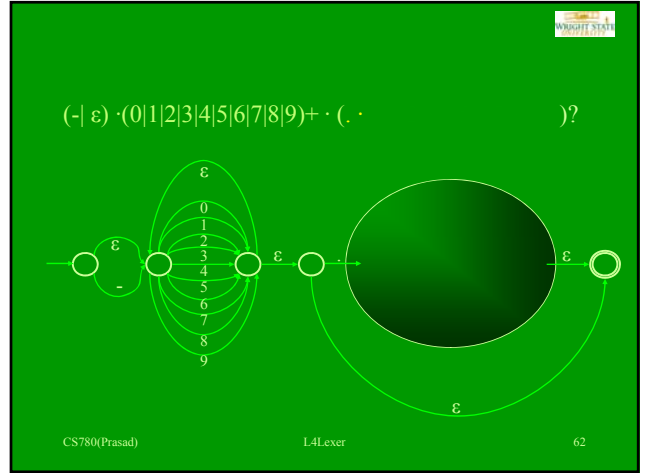
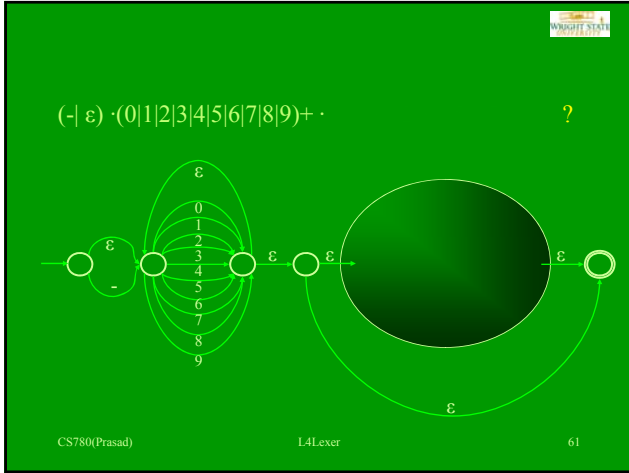
24

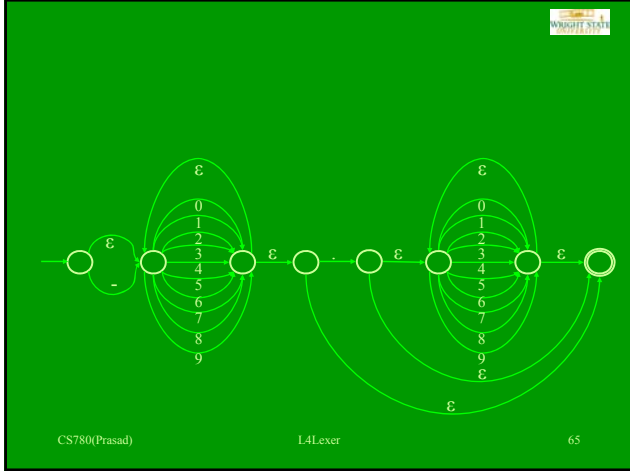
$(-|\epsilon)(0|1|2|3|4|5|6|7|8|9)^+ \cdot (.(0|1|2|3|4|5|6|7|8|9)^*)?$

$(-|\epsilon)$



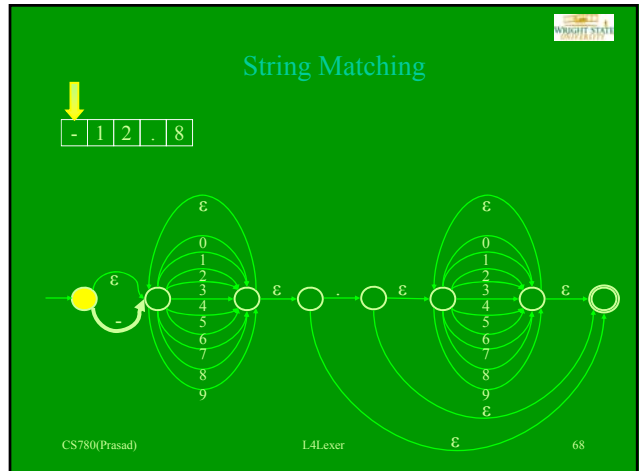
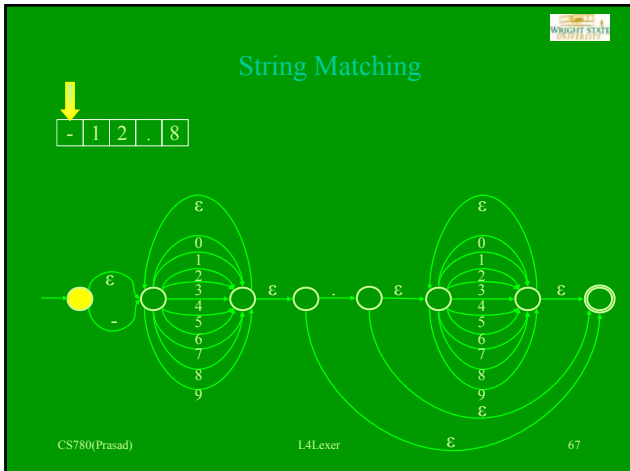


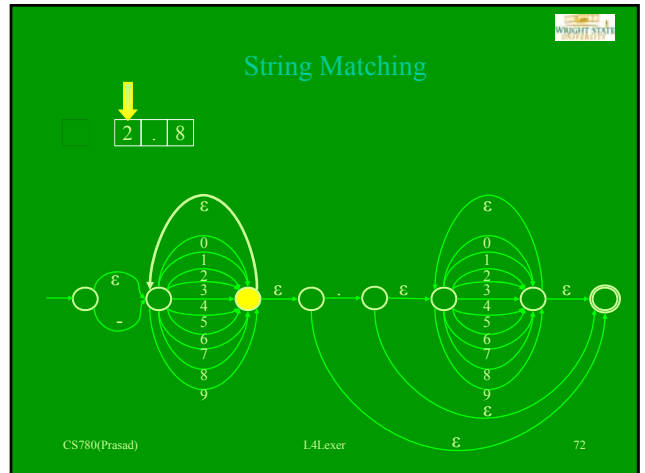
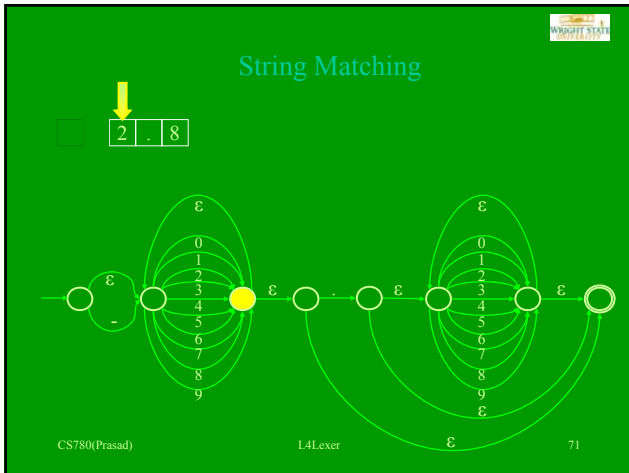
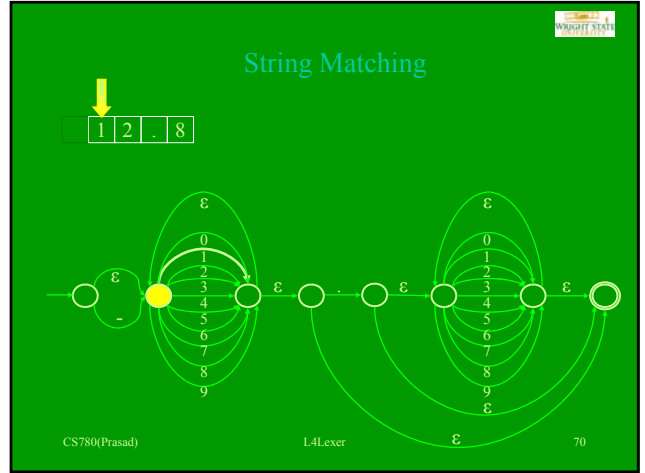
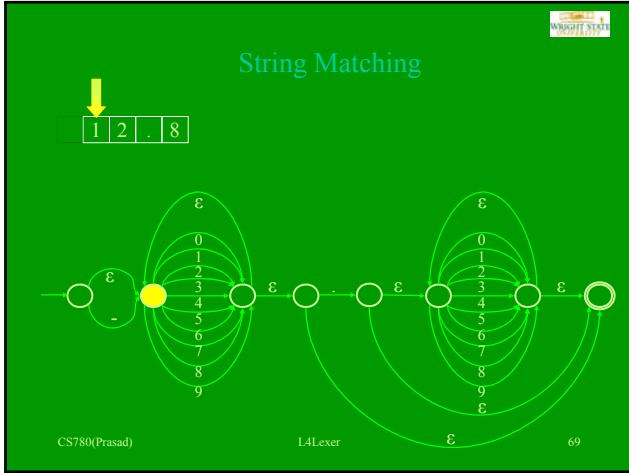


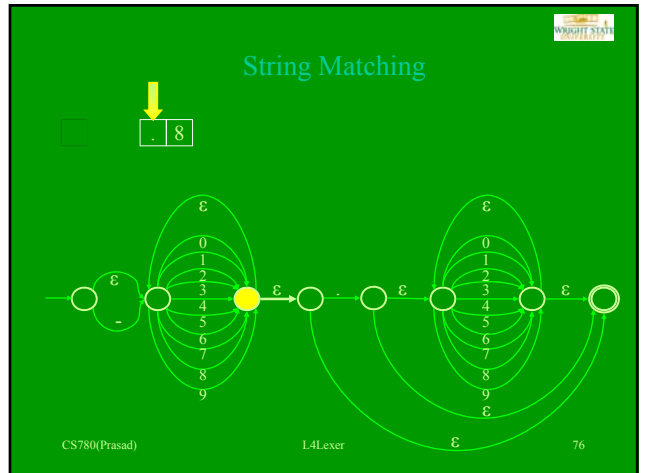
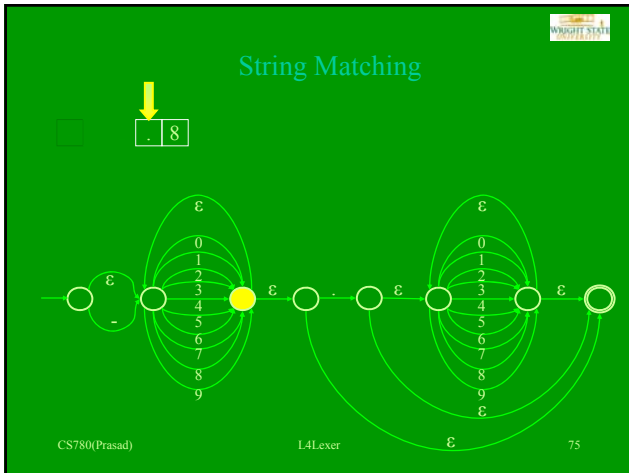
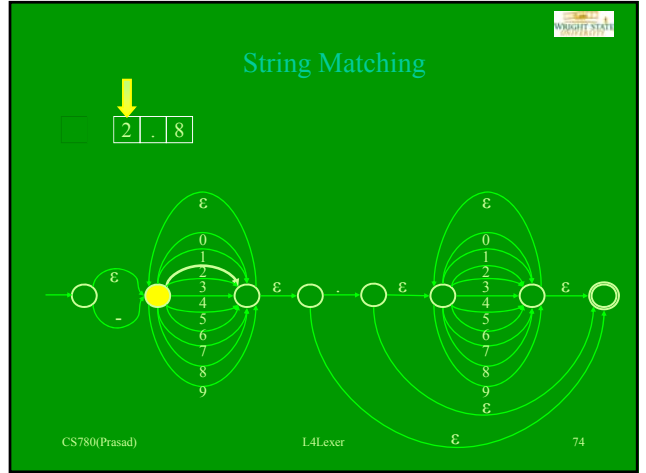
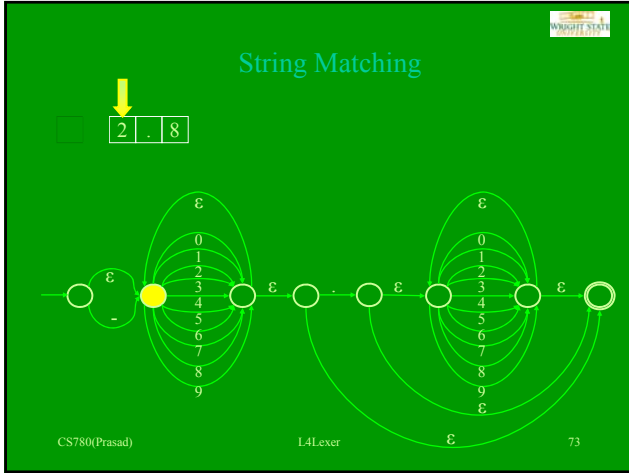


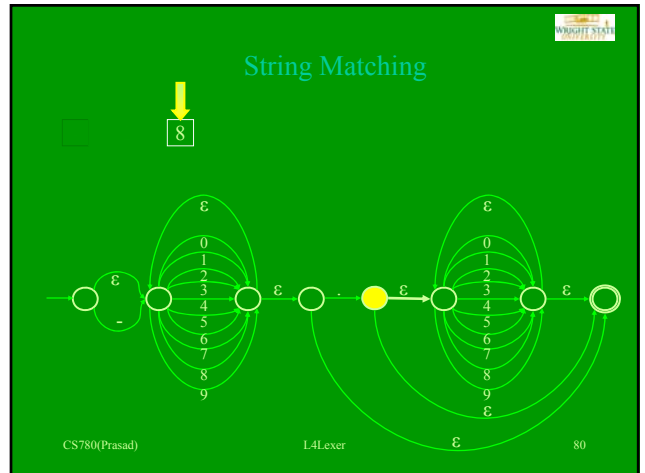
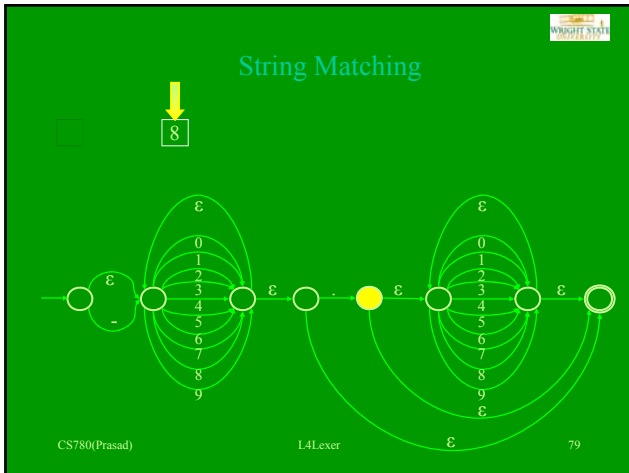
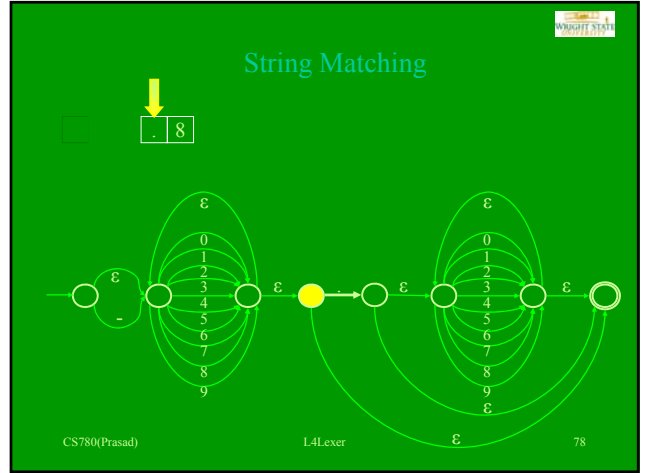
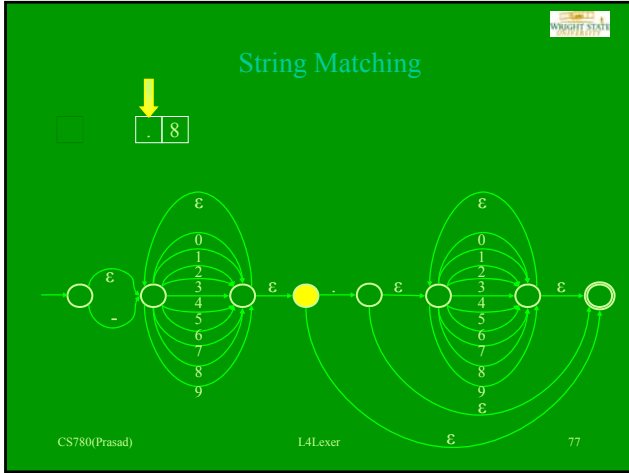
Animating Token Recognition

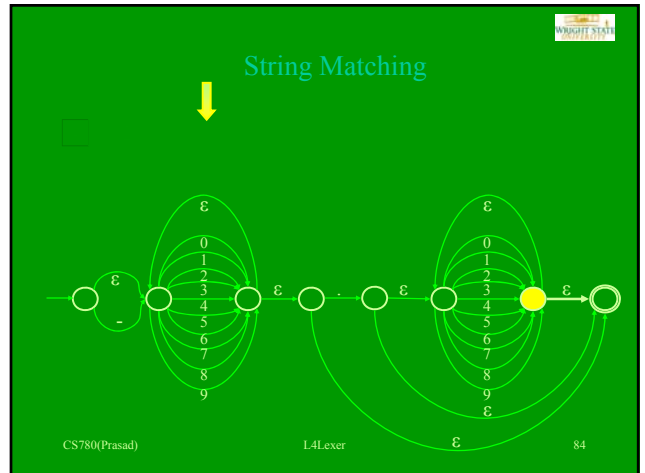
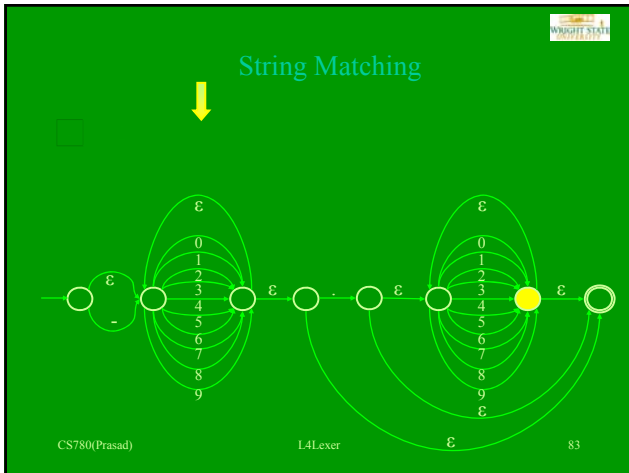
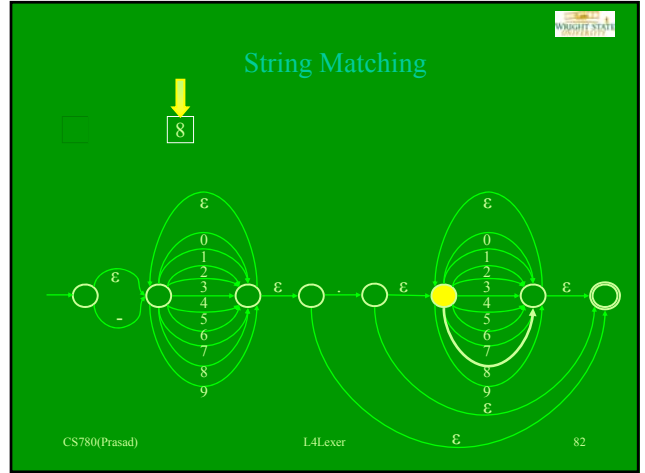
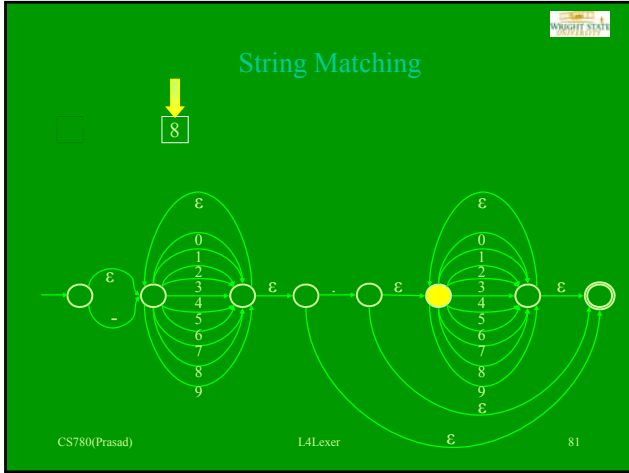
CS780(Prasad) L4Lexer 66

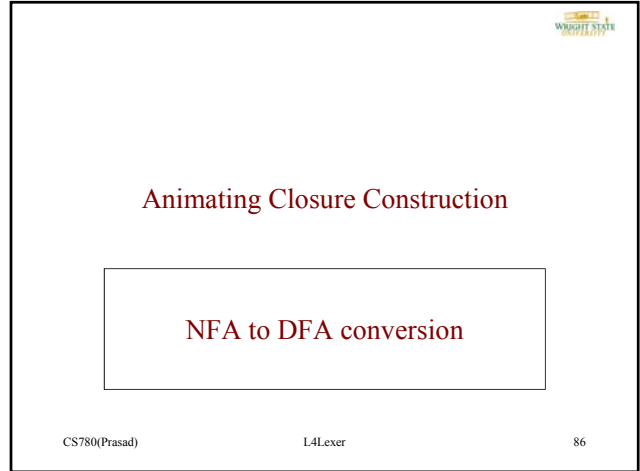
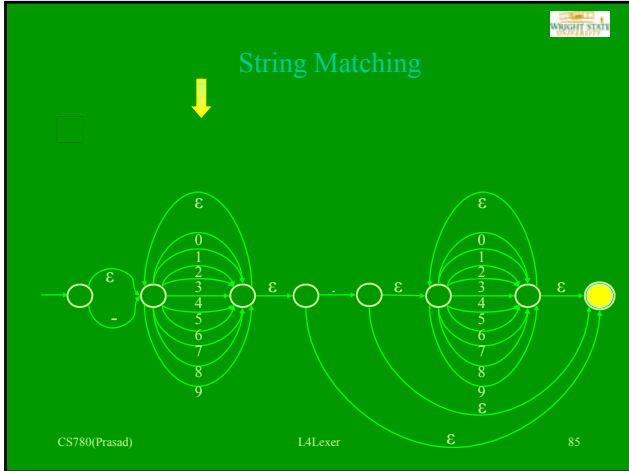












1. Closure

- The *closure* of a state is the set of states that can be reached from that state without consuming any of the input
  - Closure(S) is the smallest set T such that

$$T = S \cup \left( \bigcup_{s \in T} \text{Edge}(s, \epsilon) \right)$$

- Algorithm**

```

T ← S
repeat
    T' ← T
    T ← T ∪ ( ∪_{s ∈ T'} Edge(s, ε) )
until T = T'
```

87

(-) ε · (0|1|2|3|4|5|6|7|8|9)+ · ( · (0|1|2|3|4|5|6|7|8|9)\* )?

**S = {1}**  
**T = {}**  
**T' = {}**

→

```

T ← S
repeat
    T' ← T
    T ← T ∪ ( ∪_{s ∈ T'} Edge(s, ε) )
until T = T'
```

L4Lexter 88

$S = \{1\}$   
 $T = \{1\}$   
 $T' = \{\}$

$T \leftarrow S$   
**repeat**  
 $T' \leftarrow T$   
 $T \leftarrow T \cup \left( \bigcup_{s \in T'} \text{edge}(s, \varepsilon) \right)$   
**until**  $T = T'$

89

$S = \{1\}$   
 $T = \{1\}$   
 $T' = \{1\}$

$T \leftarrow S$   
**repeat**  
 $T' \leftarrow T$   
 $T \leftarrow T \cup \left( \bigcup_{s \in T'} \text{edge}(s, \varepsilon) \right)$   
**until**  $T = T'$

90

$S = \{1\}$   
 $T = \{1, 2\}$   
 $T' = \{1\}$

$T \leftarrow S$   
**repeat**  
 $T' \leftarrow T$   
 $T \leftarrow T \cup \left( \bigcup_{s \in T'} \text{edge}(s, \varepsilon) \right)$   
**until**  $T = T'$

91

$S = \{1\}$   
 $T = \{1, 2\}$   
 $T' = \{1\}$

$T \leftarrow S$   
**repeat**  
 $T' \leftarrow T$   
 $T \leftarrow T \cup \left( \bigcup_{s \in T'} \text{edge}(s, \varepsilon) \right)$   
**until**  $T = T'$

92

$S = \{1\}$   
 $T = \{1, 2\}$   
 $T' = \{1, 2\}$

$T \leftarrow S$   
**repeat**  
 $T' \leftarrow T$   
 $T \leftarrow T \cup \left( \bigcup_{s \in T'} \text{edge}(s, \varepsilon) \right)$   
**until**  $T = T'$

93

$S = \{1\}$   
 $T = \{1, 2\}$   
 $T' = \{1, 2\}$

$T \leftarrow S$   
**repeat**  
 $T' \leftarrow T$   
 $T \leftarrow T \cup \left( \bigcup_{s \in T'} \text{edge}(s, \varepsilon) \right)$   
**until**  $T = T'$

94

$S = \{1\}$   
 $T = \{1, 2\}$   
 $T' = \{1, 2\}$

$T \leftarrow S$   
**repeat**  
 $T' \leftarrow T$   
 $T \leftarrow T \cup \left( \bigcup_{s \in T'} \text{edge}(s, \varepsilon) \right)$   
**until**  $T = T'$

95

$S = \{1\}$   
 $T = \{1, 2\}$   
 $T' = \{1, 2\}$

$T \leftarrow S$   
**repeat**  
 $T' \leftarrow T$   
 $T \leftarrow T \cup \left( \bigcup_{s \in T'} \text{edge}(s, \varepsilon) \right)$   
**until**  $T = T'$

96

Question: What is closure(3)?

$S = \{3\}$   
 $T = ??$

CS780(Prasad) L4Lexer 97

Question: What is closure(3)?

$S = \{3\}$   
 $T = \{2, 3, 4, 8\}$

CS780(Prasad) L4Lexer 98

Animating DFAEdge Construction

NFA to DFA conversion

CS780(Prasad) L4Lexer 99

## 2. DFAedge

- Given a symbol and a state, what states can you reach?

CS780(Prasad) L4Lexer 100

What is  $\text{DFAedge}(\{1\}, 3)$ ?

$d = \{1\}$

$\text{DFAedge}(\{1\}, 3) = ??$

CS780(Prasad) L4Lxer 101

What is  $\text{DFAedge}(\{1\}, 3)$ ?

$d = \{1\}$

$\text{edge}(\{1\}, 3) = \{\}$

$\text{DFAedge}(\{1\}, 3) = \{\}$

CS780(Prasad) L4Lxer 102

What is  $\text{DFAedge}(\{1\}, 3)$ ?

$d = \{1\}$

$\text{closure}(\{1\}) = \{1, 2\}$

$\text{DFAedge}(\{1\}, 3) = \{\}$

CS780(Prasad) L4Lxer 103

What is  $\text{DFAedge}(\{1\}, 3)$ ?

$d = \{1, 2\}$

$\text{edge}(\{2\}, 3) = \{3\}$

$\text{DFAedge}(\{1\}, 3) = \{3\}$

CS780(Prasad) L4Lxer 104

What is  $\text{DFAedge}(\{1\}, 3)$ ?

$d = \{1, 2\}$   
 $\text{closure}(\{3\}) = \{2, 3, 4, 8\}$

$\text{DFAedge}(\{1\}, 3) = \{2, 3, 4, 8\}$

CS780(Prasad) L4Lxer 105

Question: What is  $\text{DFAedge}(\{3\}, .)$ ?

$d = \{3\}$

$\text{DFAedge}(\{3\}, .) = ??$

CS780(Prasad) L4Lxer 106

Question: What is  $\text{DFAedge}(\{3\}, .)$ ?

$d = \{3\}$

$\text{DFAedge}(\{3\}, .) = \{5, 6, 8\}$

CS780(Prasad) L4Lxer 107

Animating DFA Construction

NFA to DFA conversion

CS780(Prasad) L4Lxer 108

