

# Implementation of Lexical Analysis

## Lecture 4

Prof. Aiken CS 143 Lecture 4

1

## Written Assignments

- WA1 assigned today
- Due in one week
  - By 5pm
  - Turn in
    - In class
    - In box outside 411 Gates
    - Electronically

Prof. Aiken CS 143 Lecture 4

2

## Tips on Building Large Systems

- KISS (Keep It Simple, Stupid!)
- Don't optimize prematurely
- Design systems that can be tested
- It is easier to modify a working system than to get a system working

Prof. Aiken CS 143 Lecture 4

3

## Outline

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

Prof. Aiken CS 143 Lecture 4

4

## Notation

- There is variation in regular expression notation
- Union:  $A \mid B \equiv A + B$
- Option:  $A + \epsilon \equiv A?$
- Range:  $'a'+ 'b'+ \dots + 'z' \equiv [a-z]$
- Excluded range:  
complement of  $[a-z] \equiv [^a-z]$

Prof. Aiken CS 143 Lecture 4

5

## Regular Expressions in Lexical Specification

- Last lecture: a specification for the predicate  
 $s \in L(R)$
- But a yes/no answer is not enough!
- Instead: partition the input into tokens
- We adapt regular expressions to this goal

Prof. Aiken CS 143 Lecture 4

6

### Regular Expressions => Lexical Spec. (1)

1. Write a rexp for the lexemes of each token
  - Number =  $\text{digit}^+$
  - Keyword = 'if' + 'else' + ...
  - Identifier =  $\text{letter}(\text{letter} + \text{digit})^*$
  - OpenPar = '('
  - ...

Prof. Aiken CS 143 Lecture 4

7

### Regular Expressions => Lexical Spec. (2)

2. Construct  $R$ , matching all lexemes for all tokens

$$R = \text{Keyword} + \text{Identifier} + \text{Number} + \dots \\ = R_1 + R_2 + \dots$$

Prof. Aiken CS 143 Lecture 4

8

### Regular Expressions => Lexical Spec. (3)

3. Let input be  $x_1 \dots x_n$   
For  $1 \leq i \leq n$  check  
 $x_1 \dots x_i \in L(R)$
4. If success, then we know 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)

Prof. Aiken CS 143 Lecture 4

9

### Ambiguities (1)

- There are ambiguities in the algorithm
- 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”

Prof. Aiken CS 143 Lecture 4

10

### Ambiguities (2)

- 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

Prof. Aiken CS 143 Lecture 4

11

### Error Handling

- What if  
No rule matches a prefix of input ?
- Problem: Can't just get stuck ...
- Solution:
  - Write a rule matching all “bad” strings
  - Put it last (lowest priority)

Prof. Aiken CS 143 Lecture 4

12

## 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)

Prof. Aiken CS 143 Lecture 4

13

## Finite Automata

- Regular expressions = specification
- Finite automata = implementation
- A finite automaton consists of
  - An input alphabet  $\Sigma$
  - A set of states  $S$
  - A start state  $n$
  - A set of accepting states  $F \subseteq S$
  - A set of transitions  $state \xrightarrow{input} state$

Prof. Aiken CS 143 Lecture 4

14

## Finite Automata

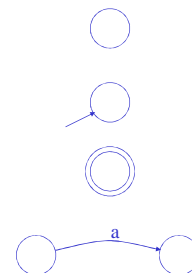
- Transition
$$s_1 \xrightarrow{a} s_2$$
- Is read  
In state  $s_1$  on input "a" go to state  $s_2$
- If end of input and in accepting state => accept
- Otherwise => reject

Prof. Aiken CS 143 Lecture 4

15

## Finite Automata State Graphs

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

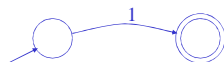


Prof. Aiken CS 143 Lecture 4

16

## A Simple Example

- A finite automaton that accepts only "1"

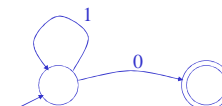


Prof. Aiken CS 143 Lecture 4

17

## Another Simple Example

- A finite automaton accepting any number of 1's followed by a single 0
- Alphabet:  $\{0,1\}$

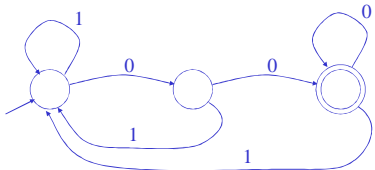


Prof. Aiken CS 143 Lecture 4

18

### And Another Example

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

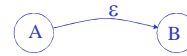


Prof. Aiken CS 143 Lecture 4

19

### Epsilon Moves

- Another kind of transition:  $\epsilon$ -moves



- Machine can move from state A to state B without reading input

Prof. Aiken CS 143 Lecture 4

20

### Deterministic and Nondeterministic Automata

- Deterministic Finite Automata (DFA)
  - One transition per input per state
  - No  $\epsilon$ -moves
- Nondeterministic Finite Automata (NFA)
  - Can have multiple transitions for one input in a given state
  - Can have  $\epsilon$ -moves

Prof. Aiken CS 143 Lecture 4

21

### Execution of Finite Automata

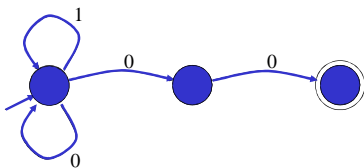
- A DFA can take only one path through the state graph
  - Completely determined by input
- NFAs can choose
  - Whether to make  $\epsilon$ -moves
  - Which of multiple transitions for a single input to take

Prof. Aiken CS 143 Lecture 4

22

### Acceptance of NFAs

- An NFA can get into multiple states



- Input: 1 0 0

Rule: NFA accepts if it can get to a final state

Prof. Aiken CS 143 Lecture 4

23

### NFA vs. DFA (1)

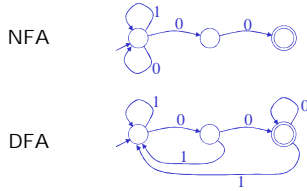
- NFAs and DFAs recognize the same set of languages (regular languages)
- DFAs are faster to execute
  - There are no choices to consider

Prof. Aiken CS 143 Lecture 4

24

## NFA vs. DFA (2)

- For a given language NFA can be simpler than DFA



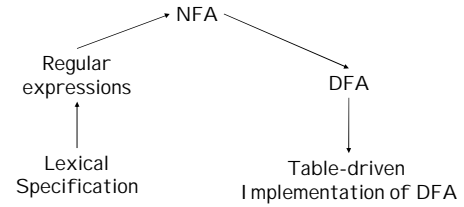
- DFA can be exponentially larger than NFA

Prof. Aiken CS 143 Lecture 4

25

## Regular Expressions to Finite Automata

- High-level sketch



Prof. Aiken CS 143 Lecture 4

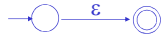
26

## Regular Expressions to NFA (1)

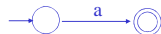
- For each kind of rexp, define an NFA
  - Notation: NFA for rexp  $M$



- For  $\epsilon$



- For input  $a$



Prof. Aiken CS 143 Lecture 4

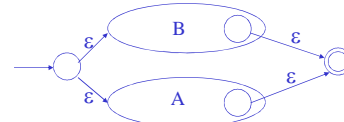
27

## Regular Expressions to NFA (2)

- For  $AB$



- For  $A + B$

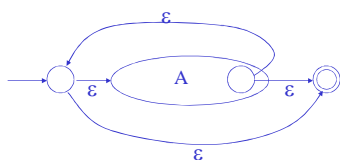


Prof. Aiken CS 143 Lecture 4

28

## Regular Expressions to NFA (3)

- For  $A^*$

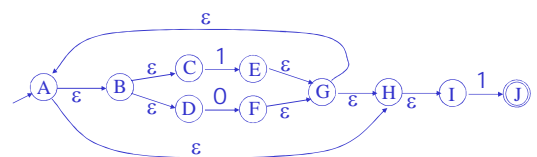


Prof. Aiken CS 143 Lecture 4

29

## Example of RegExp -> NFA conversion

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



Prof. Aiken CS 143 Lecture 4

30

### NFA to DFA: *The Trick*

- Simulate the NFA
- 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 any state in  $S$  after seeing the input  $a$ , considering  $\epsilon$ -moves as well

Prof. Aiken CS 143 Lecture 4

31

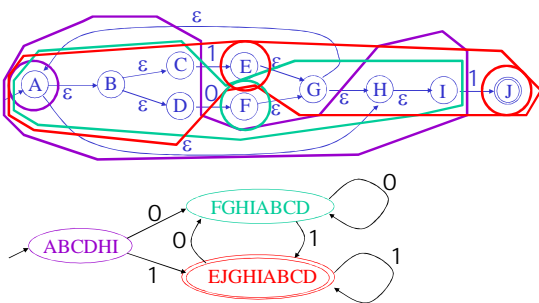
### NFA to DFA. Remark

- 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

Prof. Aiken CS 143 Lecture 4

32

### NFA $\rightarrow$ DFA Example



Prof. Aiken CS 143 Lecture 4

33

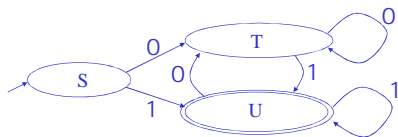
### Implementation

- A DFA can be implemented by a 2D table  $T$ 
  - One dimension is "states"
  - Other dimension is "input symbol"
  - For every transition  $S_i \xrightarrow{a} S_k$  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

Prof. Aiken CS 143 Lecture 4

34

### Table Implementation of a DFA



	0	1
S	T	U
T	T	U
U	T	U

Prof. Aiken CS 143 Lecture 4

35

### Implementation (Cont.)

- NFA  $\rightarrow$  DFA conversion is at the heart of tools such as flex
- But, DFAs can be huge
- In practice, flex-like tools trade off speed for space in the choice of NFA and DFA representations

Prof. Aiken CS 143 Lecture 4

36