CSCI 742 - Compiler Construction

Lecture 5

Automatic Construction of Lexers

Instructor: Hossein Hojjat

January 26, 2018

Lexical Analysis: Recap

Input:

i	f	(	x	=	=	0	)		x	=	x	+	1	;

Output:

IF , LPAREN , ID(x) , EQUALS , INTLIT(0) , RPAREN , ID(x) ,
EQSIGN , ID(x) , PLUS , INTLIT(1) , SEMICOLON

Regular Expressions: Recap

Regular expression over alphabet Σ:

1. ϵ is a RE denoting the set $\{\epsilon\}$
2. if $a \in \Sigma$, then a is a RE denoting $\{a\}$
3. if r and s are REs, denoting $L(r)$ and $L(s)$, then:
   - $r \mid s$ is a RE denoting $L(r) \cup L(s)$
   - $r \,.\, s$ is a RE denoting $L(r).L(s)$
   - $r*$ is a RE denoting $L(r)*$

Regular Expression Ambiguity

- **Ambiguity:** regular expressions can match input in multiple possible ways

Example

- keyword: `if`, `else`, `while`, `println`
- identifier: letter (letter | digit)*

Different ways to split the input string to tokens:

- motorcycle
  - ID(motorcycle)
  - ID(motor) , ID(cycle)
- elsevier
  - ID(elsevier)
  - ELSE , ID(vier)

Longest Match Rule (Maximal Munch)

- If multiple regular expressions match the input, the one matches the longest possible string takes precedence

  - keyword: `if`, `else`, `while`, `println`
  - identifier: letter (letter | digit)*

- motorcycle
  - ID(motorcycle) ✓
  - ID(motor) , ID(cycle)
- elsevier
  - ID(elsevier) ✓
  - ELSE , ID(vier)

Longest Match Rule (Maximal Munch)

- If multiple regular expressions match the input, the one matches the longest possible string takes precedence

> - keyword: `if`, `else`, `while`, `println`
> - identifier: letter (letter | digit)*

- motorcycle
  - ID(motorcycle) ✓
  - ID(motor) , ID(cycle)
- elsevier
  - ID(elsevier) ✓
  - ELSE , ID(vier)

What if two regular expressions match the same longest string?

Rule Priority

- If two regular expressions match the same longest string, the first declared regular expression takes precedence

  - keyword: `if`, `else`, `while`, `println`
  - identifier: letter (letter | digit)*

- else
  - ID(else)
  - ELSE ✓

Longest Match Rule (Exercise)

Exercise 1

- Consider the following specification of tokens
- Numbers gives the class of token described by the regular expression

① b(bc)* ② c*(ba)* ③ acb ④ a+

a	b	a	a	b	c	b	a	b	a	a	c	b	c	a	

Longest Match Rule (Exercise)

Exercise 1

- Consider the following specification of tokens
- Numbers gives the class of token described by the regular expression

① b(bc)* ② c*(ba)* ③ acb ④ a+

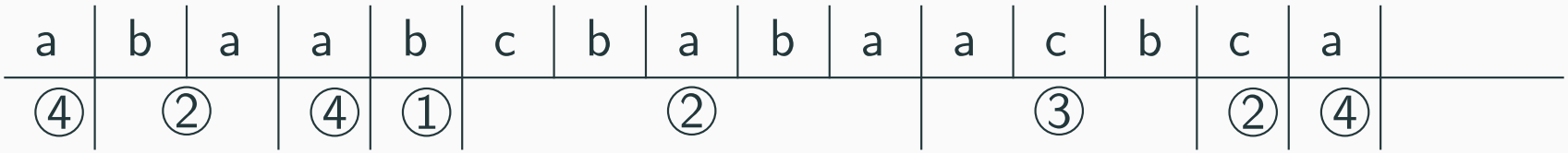
a	b	a	a	b	c	b	a	b	a	a	c	b	c	a	
④	②		④	①	②					③			②	④	

Longest Match Rule (Exercise)

Exercise 1

- Consider the following specification of tokens
- Numbers gives the class of token described by the regular expression

① b(bc)* ② c*(ba)* ③ acb ④ a+

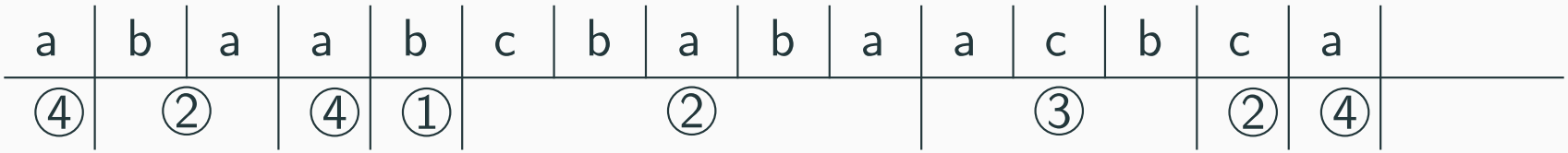


Longest Match Rule (Exercise)

Exercise 1

- Consider the following specification of tokens
- Numbers gives the class of token described by the regular expression

① b(bc)* ② c*(ba)* ③ acb ④ a+

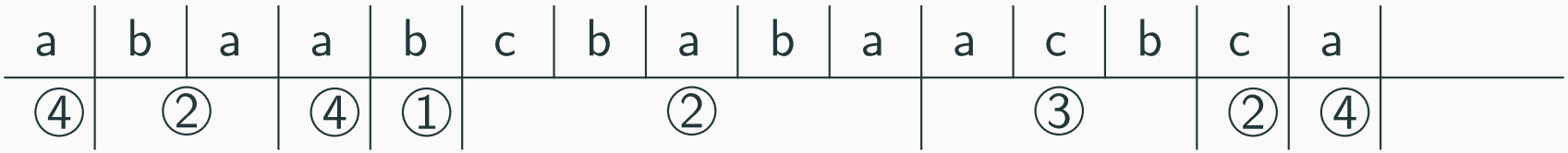


Longest Match Rule (Exercise)

Exercise 1

- Consider the following specification of tokens
- Numbers gives the class of token described by the regular expression

① b(bc)* ② c*(ba)* ③ acb ④ a+



Exercise 2

Give an example of a regular expression and an input string where

1. the regular expression is able to split the input strings into tokens
2. it is unable to do so if we use the maximal munch rule

Automatic Construction of Lexers

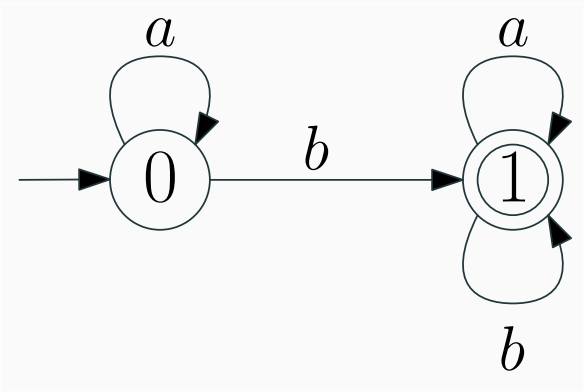
- Tools such as JFlex are able to convert regular-expression descriptions of tokens into lexers automatically
- JFlex Example:

```
Digit =  [0-9]
Letter = [a-zA-Z]
Whitespace = [ \t\n]+
{Whitespace}   {/* Do nothing! */}
{Digit}+       {return INT;}
{Letter}({Letter}|{Digit})* {return ID;}
```

Finite State Automaton

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

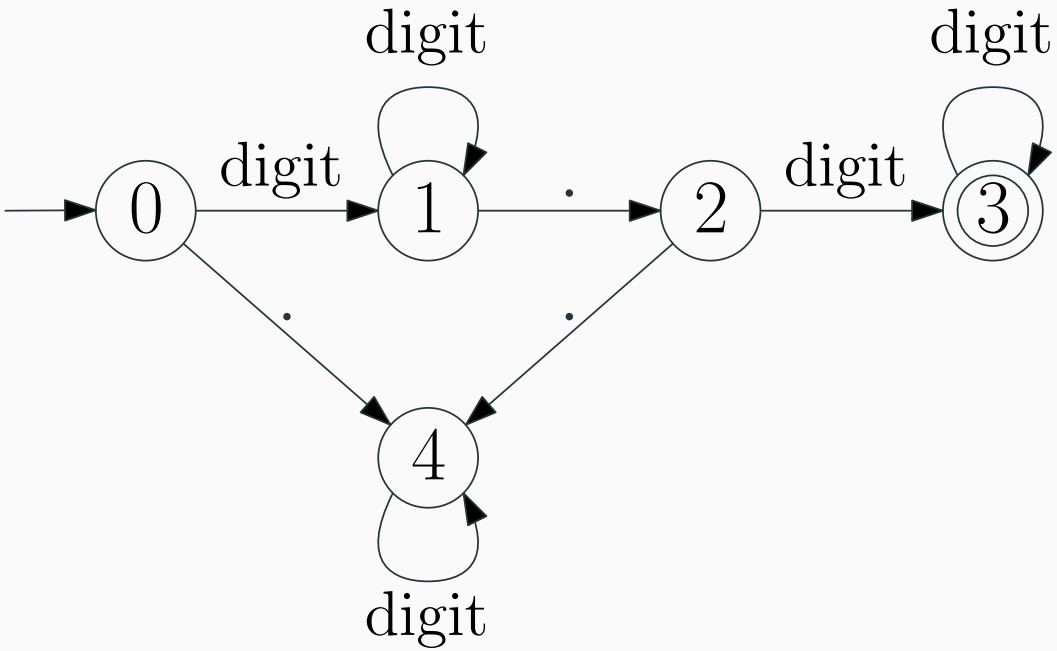
- Σ alphabet
- Q states (nodes in the graph)
- $q_0 \in Q$ initial state (with $\rightarrow$ sign in drawing)
- $\delta \subseteq Q \times \Sigma \times Q$ transitions (labeled edges in the graph)
- $F \subseteq Q$ final states (double circles)



$$\delta = \{(q_0, a, q_0), (q_0, b, q_1), (q_1, a, q_1), (q_1, b, q_1)\}$$

Finite State Automaton

digit digit* . digit digit*



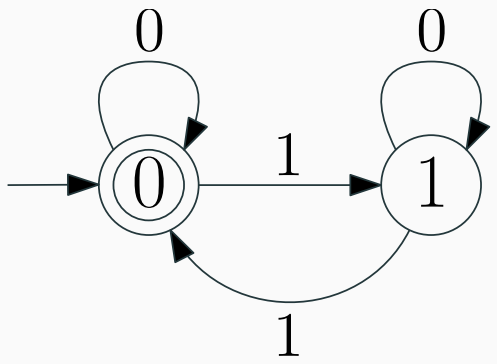
- What if the decimal part is optional?

Finite State Automaton (Exercise)

Draw a finite automaton recognizing strings over $\Sigma = \{0, 1\}$ with an even number of 1s

Finite State Automaton (Exercise)

Draw a finite automaton recognizing strings over $\Sigma = \{0, 1\}$ with an even number of 1s



Finite State Automaton (Exercise)

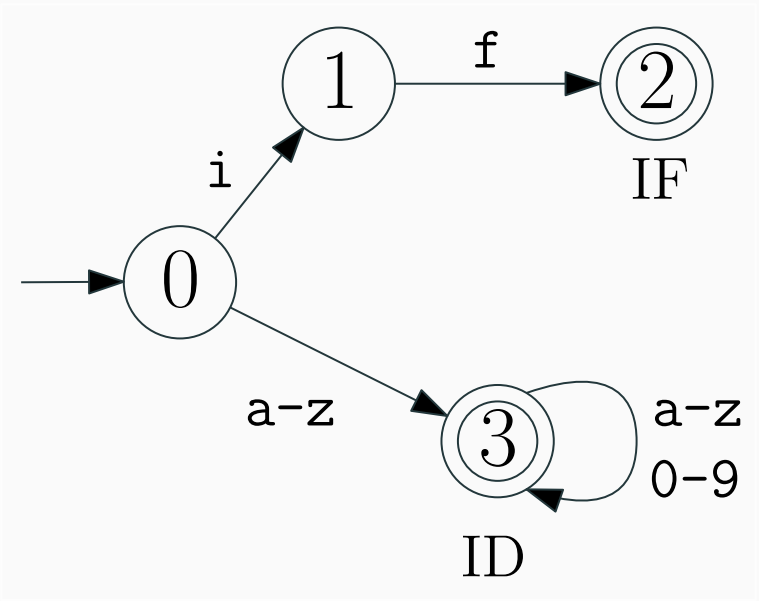
- For the alphabet $\Sigma = \{\mathtt{a}, \star, /\}$ design an automaton for C-style comments

Some test cases:

ACCEPTED	REJECTED
`/*a*/`	`/**`
`/**/`	`/**/a`
`/***/`	`aaa/**/`
`/*aaa*aaa*/`	`/*/`

Automaton for the Whole Language

- Combine the automata for individual tokens



- How can we implement the longest match rule in automata?