CSCI 2320
Principles of Programming Languages

Lexical Analysis
Reading: Chapter 3 of Tucker-Noonan

MOHAMMAD T. IRFAN

Plan

Chomsky's Hierarchy

Lexical Analysis

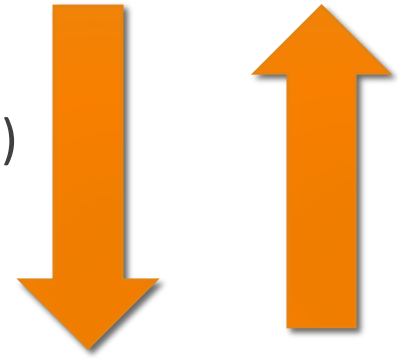
Chomsky's Hierarchy

Regular grammar

Context-free grammar (CFG/BNF)

Context-sensitive grammar

Unrestricted grammar



More expressive power

Faster computation (parsing)

Let's dive into these

Lexical Analysis

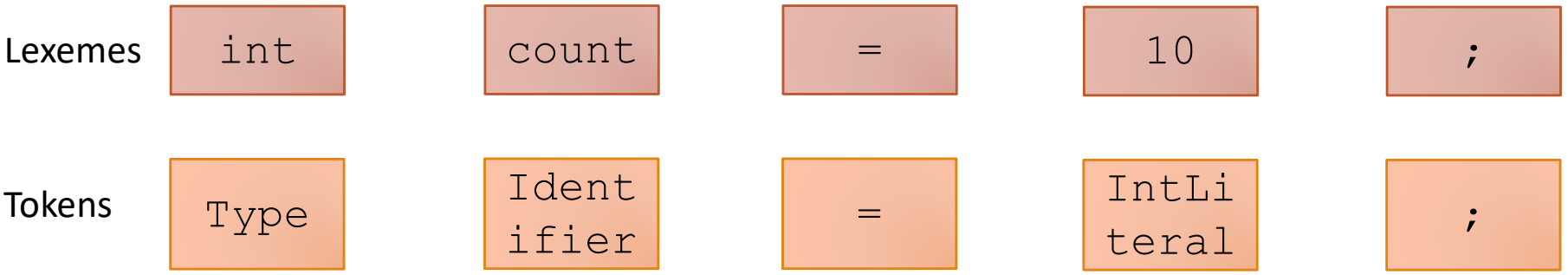
Lexical Analysis

Input: Lexeme(s) = sequence of input characters having a collective meaning

Output: Tokens (representing lexemes)

Discard: whitespace, comments

```
int count = 10;
```



Why do lexical analysis separately?

- 75% of compiling time spent in lexical analysis
- Simpler, faster grammar for parsing
  - Next: how?
- Take care of OS idiosyncrasies

Regular Expressions (regex) ↔ Regular Grammar

CONSTRUCTIVE PROOF LATER …

Recursive definition of a regex

Regex	Meaning	
x	A character x from an alphabet is a regex	(base case)
ϵ	The empty string ϵ is a regex	(base case)
MN	*Concatenation:* M followed by N is a regex	if M and N are regex
M|N	*Alteration:* M or N is a regex	if M and N are regex
M*	*Kleen star:* 0 or more occurrences of M is a regex	if M is a regex
M+	*Kleen plus:* 1 or more occurrences of M is a regex	if M is a regex
M?	0 or 1 occurrence of M is a regex	if M is a regex

Regex meta symbols

[set of char**]** Take exactly 1 character from a set of characters.

[aeiou**]** means a | e | i | o | u (that is, exactly one of the five vowels).

[0-9**]** means 0 | 1 | … | 9. Here, – is the range meta symbol.

{Z**}** A reference to a regex labeled Z (note the difference with EBNF's **{ }**).

(… **)** Grouping and additionally, in regex libraries, capturing.

. Means any single character.

^ Means not (or excluding); e.g., [a-zA-Z^aeiouAEIOU] means a consonant.

\ Escaped character; e.g., \. means dot, \(means the literal left parenthesis and not the meta symbol for grouping.

C Lite
Regular Definitions

CLite regular definition

Category	Definition	
AnyChar	[-~]	From space (ASCII 27) to tilde (126)
Letter	[a-zA-Z]	
Digit	[0-9]	
Whitespace	[\t]	Space and tab
Eol	\n	

Category	Definition
Keyword	bool | char | else | false | float | if | int | main | true | while
Identifier	{Letter}({Letter} | {Digit})*
IntegerLit	{Digit}+
FloatLit	{Digit}+\.{Digit}+
CharLit	'{AnyChar}'

Category	Definition
Operator	= | || | && | == | != | < | <= | > | + | - | * | / |! | [|]
Separator	; | . | { | } | (|)
Comment	// ({AnyChar} | {Whitespace})* {Eol}

CFG
vs
Regular grammar
vs
Regex

Describe the language:

1. $0(0|1)^{+}0$
2. $((\varepsilon|0)1^{*})^{*}$
3. $0^{*}10^{*}10^{*}10^{*}$
4. $(00|11)^{*}$

Write regular expression for:

1. All strings of lowercase letters, where letters appear in ascending order.

2. All strings of letters containing vowels in order.

3. Strings of digits (0, 1, 2) with no consecutively repeated digit.

Finite State Automata (FSA) --- DFA, NFA

BEHIND THE SCENE OF REGULAR EXPRESSIONS

READING:
TEXTBOOK AND HANDOUT (SCOTT)

Known algorithms

1. Regular expression → NFA
2. NFA → DFA

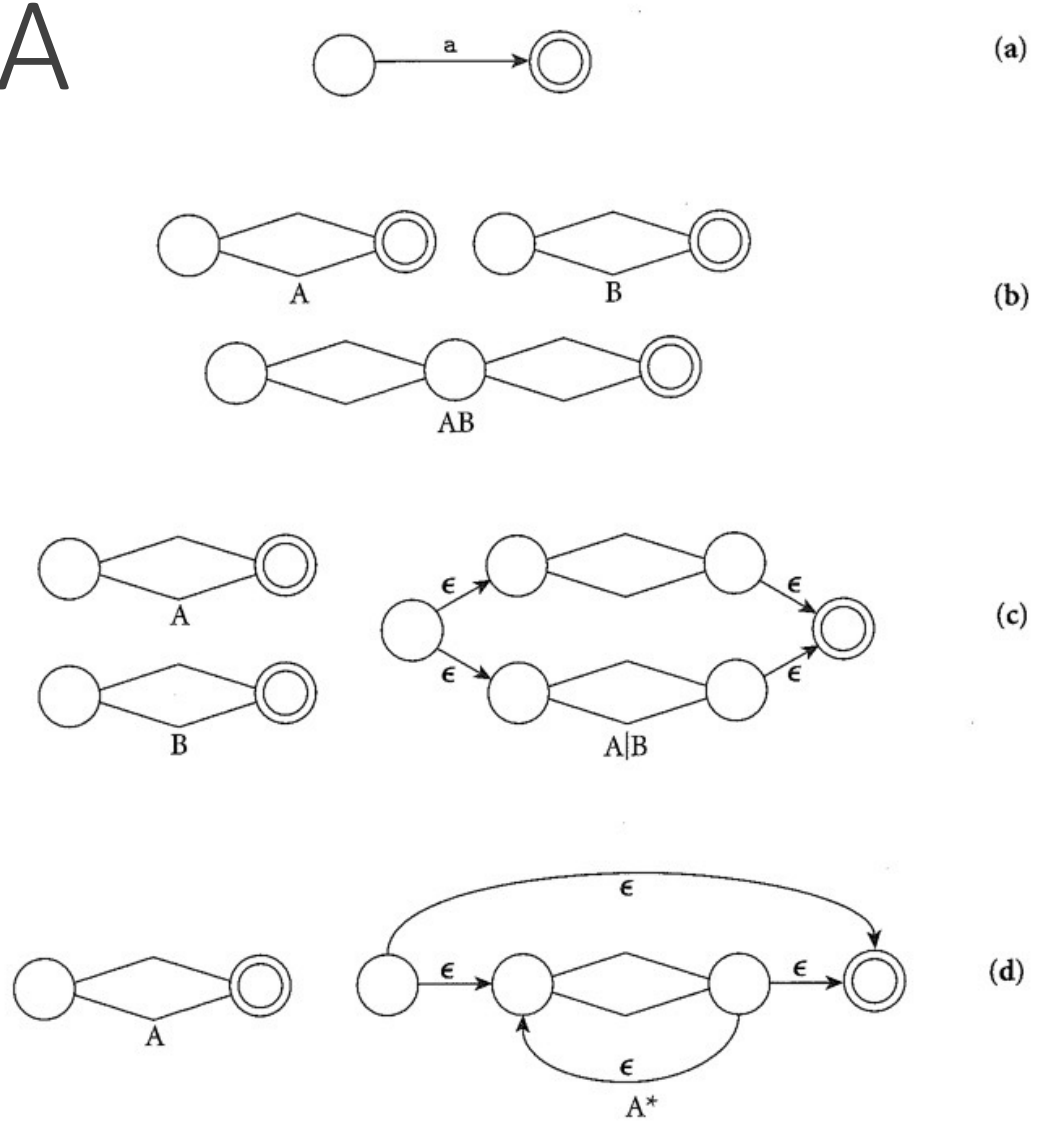
Language designer → implementation (parsing)

3. DFA → regular expression

State elimination

Regex → NFA → DFA → Regex
All 3 are equivalent!

Regex → NFA



Scott, Programming Languages (2000)

NFA → DFA

Tabular method

Regex ⟷ Regular Grammar

Regex → NFA (2 slides ago)
→ Regular grammar:

1. NFA state transition $Q_i \rightarrow Q_j$ on input a
   => Production rule $Q_i \rightarrow a\ Q_j$

2. NFA final states F
   => Production rule $F \rightarrow \epsilon$

Regex ⟷ Regular Grammar

Regular grammar

→NFA (state transitions follow production rules)

→DFA (tabular method)

→Regex (state elimination algorithm)

Lexical Analysis Using Python

Project 1

Python's re package

https://docs.python.org/3/library/re.html

```
import re #regex
re.split(...)#Use regex argument to split a string into parts
```

Common string-matching regex:

Symbol	Definition
\d	[0-9]
\D	[^0-9]
\w	[a-zA-Z0-9_]
\W	[^a-zA-Z0-9_]

Resources

Tucker-Noonan book chapter

Scott's book chapter (handout on Canvas)

Video lecture on NFA -> DFA: https://youtu.be/qfVTXwuxEOY