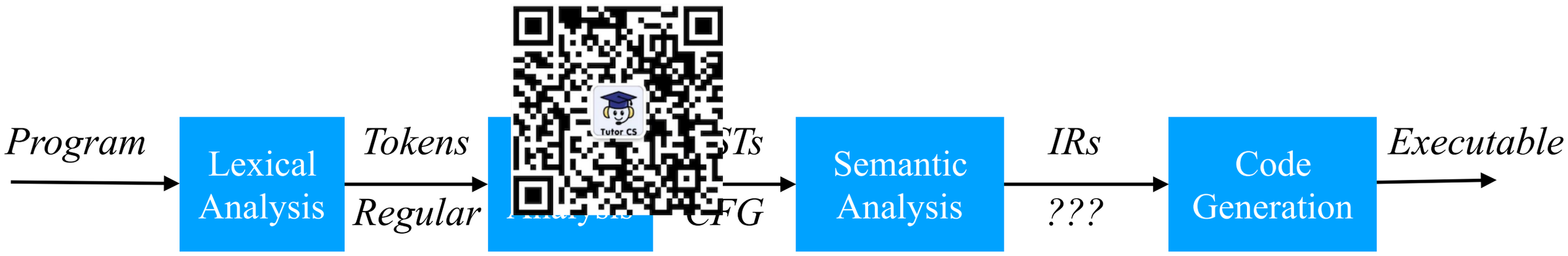
CS 160 Compilers

Lecture 5: Lexical Analysis

Yu Feng

Fall 2021

A typical flow of a compiler



A typical flow of a compiler

Source Code
(Character stream)

```
if (b == 0) { a = 1;
```

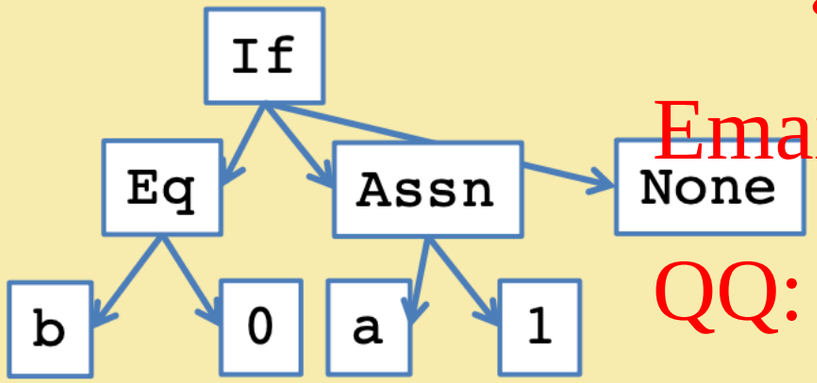
Token stream:

if	(	b	==	0	)	{	a	=	0	;	}

Lexical Analysis

Parsing

Abstract Syntax Tree:



Intermediate code:

```
l1:
  %cnd = icmp eq i64 %b,
0
  br i1 %cnd, label %l2,
label %l3
l2:
  store i64* %a, 1
  br label %l3
l3:
```


Analysis & Transformation

Backend

Assembly Code

```
l1:
  cmpq %eax, $0
  jeq l2
  jmp l3
l2:
```

Lexical analysis

- Main Question: How to apply structure to strings
- Analogy: Understanding an English sentence
  - First, we separate a string into words
  - Second, we understand sentence structure by diagramming the sentence
- Separating a string into words is called ***lexing***
- Note that lexing is not necessarily trivial

Lexical analysis

- Consider the followi ogram:

```
if x > y
then 10
else 8
```


- This program is just a string of characters

```
if x > y\nthen\t10\nelse\t8
```

- Goal: Portion the input string into substrings where the substrings are ***tokens***

What is a Token?

- Token is a syntactic category

- Example in English: noun, verbs, adjectives,...

- In a programming language: constants, identifiers, keywords, whitespaces…

Tokens in Patina

- Tokens correspond to sets of strings

- Identifier: strings of letters, digits and ’_’ starting with a letter

- Integer: a non-empty string of digits

- Keywords: “let”, “if”, ...

- Whitespace: a non-empty sequence of blanks, newlines, and tabs

What are tokens for?

- Classify program substrings according to their role

- Output of lexical analysis is a stream of tokens...

- ...which is input to the parser

- Parser relies on token distinction

  - An identifier is treated different than a keyword

Regular language/expressions

- We could specify tokens in many ways

- Regular Languages are the most popular

- Simple and useful theory

- Easy to understand

- Efficient to implement

Languages

- Definition: Let Σ be a set of characters, A language over Σ is a set of strings from characters drawn from Σ

- Alphabet: English characters => Language: English sentences

- Languages are sets of strings

- Need some notation for specifying which sets we want

- The standard notation for regular languages is regular expressions

Regular expressions

- Atomic Regular Ex

  - Single character: c = {“c”}

  - Epsilon: ε = {“”}

- Compound Regular Expressions

  - Union:A+B={s | s∈A or s∈B}

  - Concatenation: AB = {ab | a ∈ A and b ∈ B}

  - Iteration: $A^* = \bigcup_{i \geq 0} A^i$ where $A^i = A \ldots i$ times A

Regular expressions

- The regular expressions over Σ are the smallest set of expressions

- ε

- $'c'$ where $c \in \Sigma$

- $A + B$ where A, B are regular expressions over Σ

- AB where A, B are regular expressions over Σ

- A^* where A is a regular expression over Σ

- Regular expressions are simple, but very useful

Example: Integers

- Integer: non-empty string of digits.
- digit = ‘0’+ ‘1’+’2’+’3’+’4’+’5’+’6’+ ...

- integer = digit digit*

- Abbreviation: $A^{+} = AA^{*}$

Example: Identifier

- Identifier: strings of letters or digits, starting with a letter

- letter = ’A’+...+’Z’+’a’+...+’z’+’_’

- identifier = letter (letter + digit)*

- How about (letter* + digit*)?

Example: Whitespace

- Whitespace: a non-empty sequence of blanks, newlines and tabs

- Whitespace = (' ' + '\n' + '\t')$^+$

Last example: email

- Consider UCSB cs emails: anyone@cs.ucsb.edu format

- Σ= letters $\cup$ {. , @}

- name = letter$^+$

- address = name ’@’ name ’.’ name ’.’ name

TODOs by next lecture

- Come to the discussion or office hour if you have questions

- Continue with your good work on HW1