南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Chapter 1: Regular Expressions & Lexical Analysis

Yepang Liu

liuyp1@sustech.edu.cn

The chapter numbering in lecture notes does not follow that in the textbook.

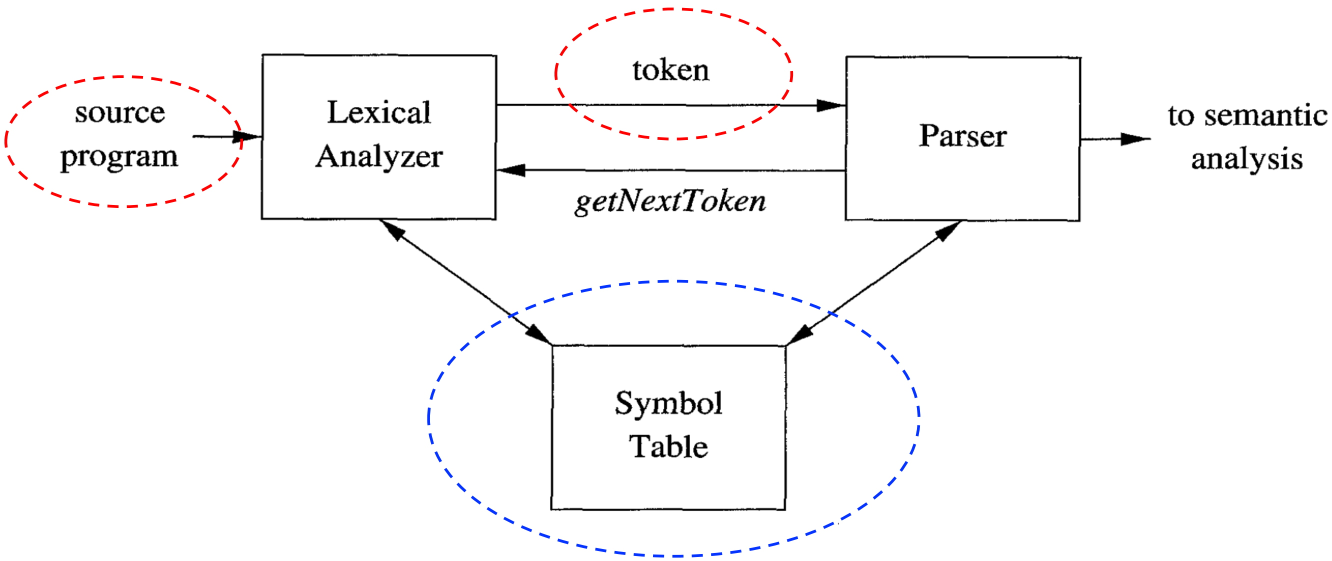
Outline

- The Role of Lexers: Recognizing Tokens
- Regular Expressions (for specifying tokens)
- Finite Automata (for recognizing patterns)

The Role of Lexical Analyzer

- Read the input characters of the source program, group them into lexemes, and produces a sequence of tokens (to be used by parser)

- Add lexemes into the symbol table when necessary



Recall the Earlier Example

```
position = initial + rate * 60
```

Lexical Analyzer

```
<id, 1>
<=>
<id, 2>
<+>
<id, 3>
<*>
<60>
```

SYMBOL TABLE

1	position	...
2	initial	...
3	rate	...

Tokens, Patterns, and Lexemes

- A ***lexeme*** is a string of characters that is a lowest-level syntactic unit in programming languages

- A ***token*** is a syntactic category representing a class of lexemes. Formally, it is a pair <token name, attribute value>
  - Token name: an abstract symbol representing the kind of the token
  - Attribute value (optional) points to the symbol table

- Each token has a particular ***pattern***: a description of the form that the lexemes of the token may take

Examples

Patterns

TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

Attributes for Tokens

- Typically, for each token, the lexical analyzer also provides additional *attribute values* to the subsequent compiler phases
  - Token names influence parsing decisions
  - Attribute values influence semantic analysis, code generation etc.

- For example, an **id** token is often associated with: (1) its lexeme, (2) data type, and (3) the location at which it is first found.

- Token attributes are stored in the symbol table.

```
                  <id, pointer to symbol-table entry for A>
                  <assign_op>
A = B * 2  ───▶   <id, pointer to symbol-table entry for B>
                  <mult_op> <number, integer value 2>
```

Lexical Errors

- When the prefix of the remaining input does not match any token patterns
- Example: `int @3 = 5;`

Outline

- The Role of Lexers: Recognizing Tokens
- Regular Expressions (for specifying tokens)
- Finite Automata (for recognizing patterns)

Specification of Tokens

- **Regular expression (正则表达式, regexp for short)** is an important notation for specifying token patterns
- Content of this part
  - Strings and Languages (串和语言)
  - Operations on Languages (语言上的运算)
  - Regular Expressions
  - Regular Definitions (正则定义)
  - Extensions of Regular Expressions

Strings and Languages

- **Alphabet (字母表)**: any finite set of symbols
  - Examples of symbols: letters, digits, and punctuations
  - Examples of alphabets: {1, 0}, ASCII, Unicode

- A **string (串)** over an alphabet is a finite sequence of symbols drawn from the alphabet
  - The length of a string *s*, denoted |***s***|, is the number of symbols in *s* (i.e., cardinality)
  - Empty string (空串): the string of length 0, ϵ

Terms (using banana for illustration)

- **Prefix (前缀) of string *s*:** any string obtained by removing 0 or more symbols from the end of *s* (ban, banana, ϵ)
- **Proper prefix (真前缀):** a prefix that is not ϵ and not *s* itself (ban)
- **Suffix (后缀):** any string obtained by removing 0 or more symbols from the beginning of *s* (nana, banana, ϵ).
- **Proper suffix (真后缀):** a suffix that is not ϵ and not equal to *s* itself (nana)

Terms Cont.

- **Substring (子串) of s:** any string obtained by removing any prefix and any suffix from s (banana, nan, ϵ)
- **Proper substring (真子串):** a substring that is not ϵ and not equal to s itself (nan)
- **Subsequence (子序列):** any string formed by removing 0 or more not necessarily consecutive symbols from s (bnn)

Think about this after class: How many substrings and subsequences does banana have?

(Two substrings are different if they have different start/end index)

String Operations (串的运算)

- **Concatenation (连接)**: the concatenation of two strings x and y, denoted xy, is the string formed by appending y to x
  - x = I, y = ♡, z = compilers, xyz = I♡compilers

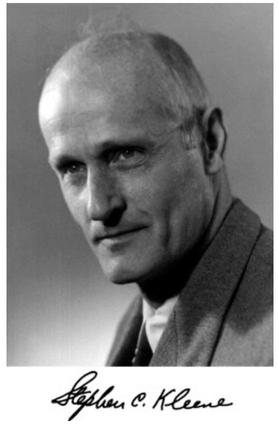
- **Exponentiation (幂/指数运算):** $s^0 = \epsilon$, $s^1 = s$, $s^i = s^{i-1}s$
  - $x = 6$, $x^0 = \epsilon$, $x^1 = 6$, $x^3 = 666$

Language (语言)

- A **language** is any **countable set**[1] of strings over some fixed alphabet
  - The set containing only the empty string, that is $\{\epsilon\}$, is a language, denoted ∅
  - The set of all grammatically correct English sentences
  - The set of all syntactically well-formed C programs

[1] In mathematics, a countable set is a set with the same cardinality (number of elements) as some subset of the set of natural numbers. A countable set is either a finite set or a countably infinite set.

Operations on Languages (语言的运算)



- Union (并), Concatenation (连接)
- Kleene Closure (Kleene闭包)
- Positive Clousre (正闭包)

OPERATION	DEFINITION AND NOTATION
Union of L and M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L	$L^* = \cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+ = \cup_{i=1}^{\infty} L^i$

The exponentiation of L can be defined using concatenation. L^n means concatenating L n times.

https://en.wikipedia.org/wiki/Stephen_Cole_Kleene

Examples

- L = {A, B, …, Z, a, b, …, z} .... 52 English letters
- D = {0, 1, …, 9} .... 10 digits

$L \cup D$	{A, B, …, Z, a, b, …, z, 0, 1, …,9}
LD	the set of 520 strings of length two, each consisting of one letter followed by one digit
L^4	the set of all 4-letter strings
L^*	the set of all strings of letters, including ϵ
$L(L \cup D)^*$	?
D^+	?

Note: L, D might seem to be the alphabets of letters and digits. We define them to be languages: all strings happen to be of length one.

Regular Expressions - For Describing Languages/Patterns

Rules that define regexps over an alphabet Σ:

- **BASIS**: two rules form the basis:
  - ϵ is a regexp, $L(\epsilon) = \{\epsilon\}$
  - If a is a symbol in Σ, then a is a regexp, and $L(a) = \{a\}$
- **INDUCTION:** Suppose r and s are regexps denoting the languages $L(r)$ and $L(s)$
  - $(r)|(s)$ is a regexp denoting the language $L(r) \cup L(s)$
  - $(r)(s)$ is a regexp denoting the language $L(r)L(s)$
  - $(r)^*$ is a regexp denoting $(L(r))^*$
  - (r) is a regexp denoting $L(r)$, that is, additional parentheses do not change the language an expression denotes.

Regular Expressions Cont.

- Following the rules, regexps often contain unnecessary pairs of parentheses. We may drop some if we adopt the conventions:
  - **Precedence (优先级):** closure * > concatenation > union |
  - **Associativity (结合性):** All three operators are left associative, meaning that operations are grouped from the left.
    - For example, a | b | c would be interpreted as (a | b) | c
- Example: $(a) \mid ((b)^*(c))$ can be simplified as $a \mid b^*c$

Regular Expressions Examples

- Let $\Sigma = \{a, b\}$
  - a|b denotes the language {a, b}
  - (a|b)(a|b) denotes {aa, ab, ba, bb}
  - a^* denotes {ϵ, a, aa, aaa, …}
  - $(a|b)^*$ denotes the set of all strings consisting of 0 or more *a*'s or *b*'s: {ϵ, a, b, aa, ab, ba, bb, aaa, …}
  - $a|a^*b$ denotes the string *a* and all strings consisting of 0 or more *a*'s and ending in *b*: {a, b, ab, aab, aaab, …}

Regular Language (正则语言)

- A **regular language** is a language that can be defined by a regexp
- If two regexps r and s denote the same language, they are *equivalent*, written as $r = s$

$$(\mathbf{a} \mid \mathbf{b})(\mathbf{a} \mid \mathbf{b})$$

$$=$$

$$\mathbf{aa} \mid \mathbf{ab} \mid \mathbf{ba} \mid \mathbf{bb}$$

?

Algebraic Laws

- Each law below asserts that expressions of two different forms are equivalent

LAW	DESCRIPTION
$r \mid s = s \mid r$	$\mid$ is commutative
$r \mid (s \mid t) = (r \mid s) \mid t$	$\mid$ is associative
$r(st) = (rs)t$	Concatenation is associative
$r(s \mid t) = rs \mid rt;\ (s \mid t)r = sr \mid tr$	Concatenation distributes over $\mid$
$\epsilon r = r\epsilon = r$	ϵ is the identity for concatenation
$r^* = (r \mid \epsilon)^*$	ϵ is guaranteed in a closure
$r^{**} = r^*$	* is idempotent

| can be viewed as + in arithmetics, concatenation can be viewed as ×, * can be viewed as the power operator.

Regular Definitions (正则定义)

- For notational convenience, we can give names (e.g., d_i below) to certain regexps and use those names in subsequent expressions

If Σ is an alphabet of basic symbols, then a ***regular definition*** is a sequence of definitions of the form:

$$d_1 \rightarrow r_1$$
$$d_2 \rightarrow r_2$$
$$\dots$$
$$d_n \rightarrow r_n$$

where:

- Each d_i is a new symbol not in Σ and not the same as the other d's
- Each r_i is a regexp over the alphabet $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$

Each new symbol denotes a regular language. The second rule means that you may reuse previously-defined symbols.

Examples

- Regular definition for C identifiers

$$\begin{array}{rcl} letter\_ & \rightarrow & \texttt{A} \mid \texttt{B} \mid \cdots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \mid \texttt{\_} \\ digit & \rightarrow & \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\ id & \rightarrow & letter\_ \; (\; letter\_ \mid digit \;)^* \end{array}$$

`_hello` valid?

`3times` valid?

- Regexp for C identifiers

(A|B|...|Z|a|b|...|z|_)((A|B|...|Z|a|b|...|z|_)|(0|1|...|9))*