



中山大學
SUN YAT-SEN UNIVERSITY



国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

DCS290

Compilation Principle 编译原理

第三章 词法分析 (1)

郑馥丹

zhengfd5@mail.sysu.edu.cn

CONTENTS

目录

01

概述

Introduction

02

词法规范

Lexical
Specification

03

有穷自动机

Finite
Automata

04

转换和等价

Transformation
and Equivalence

05

词法分析实践

Lexical Analysis
in Practice

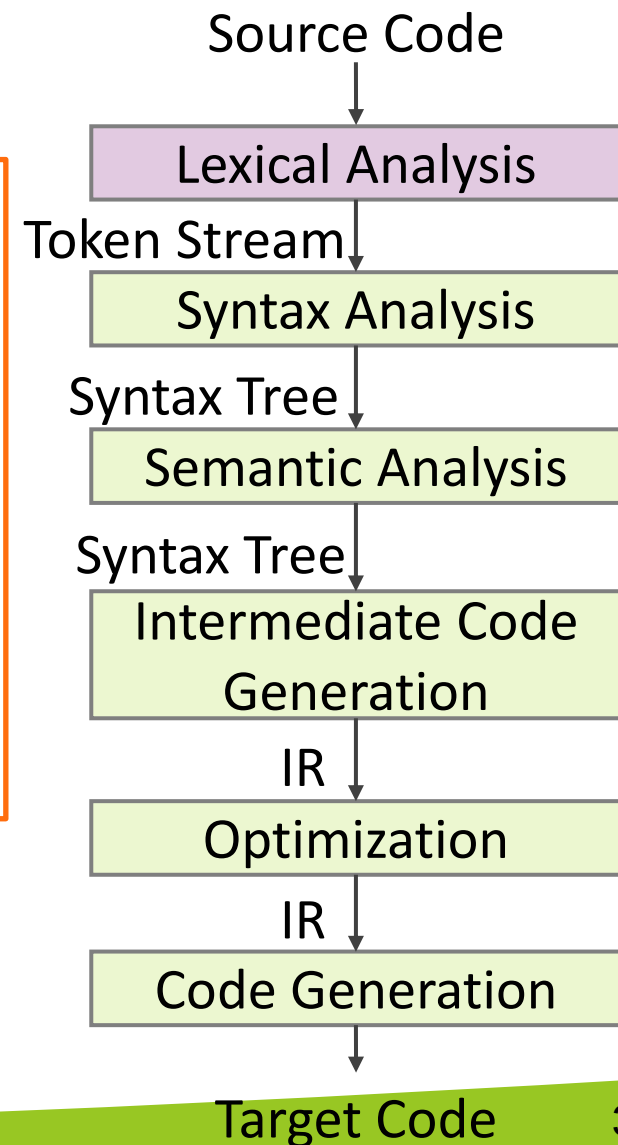
1. 什么是词法分析[Lexical Analysis]?

- 扫描源程序字符流，识别并分解出有词法意义的单词或符号 (**token**)

- 输入：源程序，输出：token序列
- token表示：<类别, 属性值>
 - ✓ 保留字、标识符、常量、运算符等
- token是否符合**词法规则**?
 - ✓ 0var, \$num

```
void main()
{
    int arr[10], i, x = 1;
    for (i = 0; i < 10; i++)
        arr[i] = x * 5;
}
```

keyword(for)	id(arr)
symbol(()	symbol([)
id(i)	id(i)
symbol(=)	symbol(])
num(0)	symbol(=)
symbol(;)	id(x)
id(i)	symbol(*)
symbol(<)	num(5)
num(10)	symbol(;)
symbol(;)	
id(i)	
symbol(++)	
symbol())	



1. 什么是词法分析[Lexical Analysis]?

- 扫描源程序字符流，识别并分解出有词法意义的单词或符号 (**token**)

- 例：

- 输入：

- ✓ 字符串“if (i == j)\n\tz = 0; \nelse\n\tz = 1; \n”

- 目标：将字符串划分为一组**tokens**

- 步骤：

- ① 移除注释：/* simple example */

- ② 识别tokens：‘if’ ‘(’ ‘i’ ‘==’ ‘j’

- ③ 识别tokens所属的类别：(**keyword**, ‘if’), (**LPAR**, ‘(’), (**id**, ‘i’)

- 输出：(**keyword**, ‘if’), (**LPAR**, ‘(’), (**id**, ‘i’)

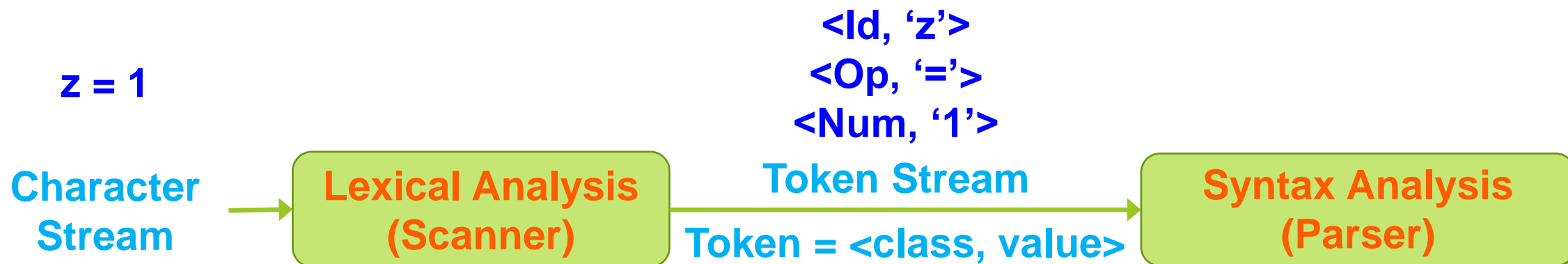
```
/* simple example */  
if (i == j)  
    z = 0;  
else  
    z = 1;
```

2. 什么是词[token]?

- Token: 词, **最小意义单元**
 - 英语中的token类别:
 - ✓ 名词noun, 动词verb, 形容词adjective, ...
 - 编程语言中的token类别:
 - ✓ 数字number, 关键词keyword, 空白whitespace, 标识符identifier, ...
- 每个token类别对应一个字符串集合[a set of strings]
 - number: 非空的数字字符串
 - keyword: 一组固定的**保留字**(“for”, “if”, “else”, ...)
 - whitespace: 由空格blank、制表符tab、换行符newline组成的非空序列
 - identifier: 用户自定义的要标识的实体名称

3. 词法分析：分词[Tokenization/Scanner]

- 词法分析器也称**标记化Tokenization**或**扫描器Scanner**
 - 将输入字符串划分为token序列
 - 对token进行分类——token类别
 - ✓ **词素[lexeme]**：token类的实例，例如：‘z’，‘=’，‘1’
 - ✓ 事先定义好token类别：例如keyword, identifier, whitespace, integer等
- 得到的token会被送入**语法分析器[Syntax Analyzer]**（也称**Parser**）
 - parser依赖token类来识别角色（例如，关键字与标识符的处理方式不同）。



3. 词法分析：设计[Design]

- 定义token类别

- 描述所有感兴趣的项
- 依赖于语言、parser的设计

✓ `“if (i == j)\n\tz = 0; \nelse\n\tz = 1; \n”`

Keyword, identifier, whitespace, integer

- 识别哪些字符串属于哪个token类别

```
if (i == j)
    z = 0;
else
    z = 1;
```

‘==’ or ‘=’? → 最长匹配原则

keyword or identifier? → 查询keyword表

5种最常见的编程语言token类别:

- Identifiers: `getBalance`, `weight`, ...
- Reserved words: `if`, `else`, `while`, ...
- Constants: `10`, `3.14`, `“abc”`, `‘a’`, ...
- Operators: `+`, `-`, `*`, `/`, `<<`, ...
- Punctuation: `(`, `)`, `;`, `:`, ...

3. 词法分析：实现[Implementation]

- 实现时需完成2件事情：
 - 识别字符串的token分类
 - 返回token的值或词素
- 一个token是一个二元组(class, lexeme)
- 丢弃无意义词
 - 例如：whitespace, comments等
- 如果token类别是无二义性的，则可以在对输入字符串进行一次从左到右的扫描中识别标记
- 但token类别可能有歧义[ambiguous]

3. 词法分析：实现[Implementation]

- 歧义举例

- C++ 模板语法：Foo<Bar>

- C++ 流语法：cin >> var

- 二义性

- ✓ Foo<Bar<Bar>> **疑问：‘>>’应当成是流操作符，还是两个连续的右尖括号呢？**
- ✓ cin >> var

- “**向前看[look ahead]**”可以展望消除歧义

- 要查看更大的上下文或结构 cin >> var

- 有时词法分析需要parser的反馈：“解析模板嵌套，这应该是两个独立的>”

- 如果token没有歧义

- tokenizing可在无parser反馈的情况下一次完成，可明确区分词法/语法分析

```
Template <typename T>
T getMax(T x, T y) {
    return (x > y) ? x : y;
}
```

```
int main (int argc, char* argv[]) {
    getMax<int>(3, 7);
    getMax<double>(3.0, 2.0);
    getMax<char>('g', 'e');

    return 0;
}
```

3. 词法分析：总结

- 词法分析

- 将输入符号串分解成token
- 识别每个token的类别

- 从左到右扫描

- 有时候需要 “向前看[look ahead]”
- 应该尽量减少 “向前看[look ahead]” 的数量（过多会增加复杂性）

```
/* simple example */  
if (i == j)  
    z = 0;  
else  
    z = 1;
```

```
if (i == j)  
    z = 0;  
else  
    z = 1;
```

```
'if' '(' 'i' '==' 'j' ')' '\n' '\t' 'z' '=' '0' ';' '\n'  
'\n' 'else' '\n' '\t' 'z' '=' '1'
```

```
<keyword, if> <LPAR, (>, <id, i>, <op, ==>
```

```
... ..
```

CONTENTS

目录

01

概述

Introduction

02

词法规范

Lexical
Specification

03

有穷自动机

Finite
Automata

04

转换和等价

Transformation
and Equivalence

05

词法分析实践

Lexical Analysis
in Practice

1. 词法规范

• 三种词法规范

– 表达式[Expressions]

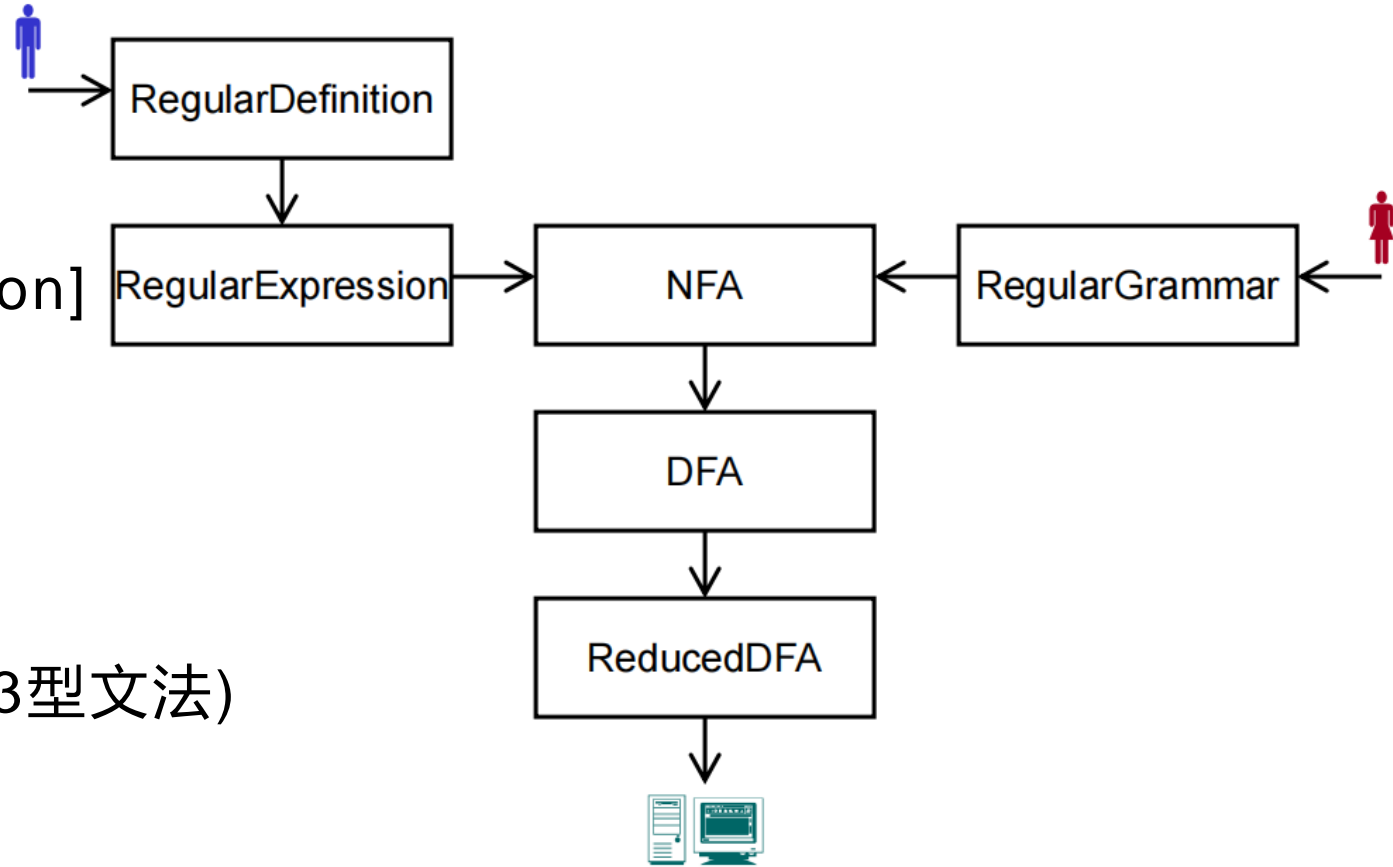
- ✓ 正则表达式[Regular expression]
- ✓ 正则定义[Regular definition]

– 语法

- ✓ 正规文法[Regular grammar](3型文法)

– 有穷自动机

- ✓ 确定的有穷自动机[Deterministic Finite Automata (DFA)]
- ✓ 不确定的有穷自动机[Nondeterministic Finite Automata (NFA)]



2. 正则表达式[Regular expressions, REs]

- 正则表达式

- 也称：**正规式**
- 可用于定义token class
- **简单但功能强大**（能够表达模式）
- 可以从规范**自动生成**Tokenizer实现（使用翻译工具）
- 最终实现是**有效的**

2. 正则表达式[Regular expressions, REs]

• 正规集：正规式所能描述的语言集合

- ① ε 和 Φ 是 Σ 上的正规式，正规集 $L(\varepsilon)=\{\varepsilon\}$ ， $L(\Phi)=\Phi$ ；
- ② 对于任何 $a \in \Sigma$ ， a 是 Σ 上的正规式，它所表示的正规集为 $L(a)=\{a\}$ ；
- ③ 假定 r 和 s 都是 Σ 上的正规式，他们所表示的正规集分别为 $L(r)$ 和 $L(s)$ ，那么，以下也都是正规式和它们所表示的正规集；

正规式	正规集
(r)	$L(r) = L((r)) = L(r)$
$r s$	$L(r s) = L(r) \cup L(s)$
$r \cdot s$	$L(r \cdot s) = L(r) L(s)$
r^*	$L(r^*) = (L(r))^*$

算符的优先顺序: $'()' > '*' > '\cdot' > '|'$

其中: $'\cdot'$ 和 $'|'$ 都是左结合

$ab^*c|d$ $a(b^*)c|d$
 $(a(b^*))c|d$
 $((a(b^*))c)|d$

- ④ 仅由有限次使用上述三步定义的表达式才是 Σ 上的正规式，仅由这些正规式所表示的字集才是 Σ 上的正规集。

2. 正则表达式[Regular expressions, REs]

- 例子

- 令 $\Sigma = \{a, b\}$, Σ 上的正规式和相应的正规集有:

正规式

正规集

a

{a}

a | b

{a,b}

ab

{ab}

(a | b)(a | b)

{aa,ab,ba,bb}

a*

{ ϵ , a, aa, ... 任意个a串}

(a | b)*

{ ϵ , a, b, aa, ab 所有由a和b组成的串}

2. 正则表达式[Regular expressions, REs]

• 正则表达式的代数性质

$$- r \mid s = s \mid r$$

“ \mid ” 满足交换律

$$- r \mid (s \mid t) = (r \mid s) \mid t$$

“ \mid ” 的结合律

$$- (r s) t = r (s t)$$

“ \cdot (连接)” 的结合律

$$- r(s \mid t) = r s \mid r t$$

$$- (r \mid s) t = r t \mid s t$$

分配律

$$- \varepsilon r = r \varepsilon = r$$

ε 是 “ \cdot ” 的恒等元素

$$- r \mid r = r$$

“ \mid ” 的抽取律 $r^* = \varepsilon \mid r \mid rr \mid \dots$

2. 正则表达式[Regular expressions, REs]

- 常用表达

- At least one: $A^+ \equiv AA^*$
- Option: $A? \equiv A | \epsilon$
- Characters: $[a_1a_2\dots a_n] \equiv a_1|a_2|\dots|a_n$
- Range: $'a' + 'b' + \dots + 'z' \equiv [a-z]$
- Excluded range: **complement of $[a-z] \equiv [^a-z]$**

2. 正则表达式[Regular expressions, REs]

• 例子

Regular Expression	Explanation
a^*	0 or more a's (ϵ , a, aa, aaa, aaaa, ...)
a^+	1 or more a's (a, aa, aaa, aaaa, ...)
$(a b)(a b)$	(aa, ab, ba, bb)
$(a b)^*$	all strings of a's and b's (including ϵ)
$(aa ab ba bb)^*$	all strings of a's and b's of even偶数 length
$[a-zA-Z]$	shorthand简写 for "a b ...z A B ... Z"
$[0-9]$	shorthand for "0 1 2 ... 9"
$0([0-9])^*0$	numbers that start and end with 0
$1^*(0 \epsilon)1^*$	binary strings that contain at most one zero
$(0 1)^*00(0 1)^*$	all binary strings that contain '00' as substring

2. 正则表达式[Regular expressions, REs]

- 思考: $(a|b)^*$ 和 $(a^*b^*)^*$ 是否等价? **等价**

$$(a|b)^* = ?$$

$$\begin{aligned} (L(a|b))^* &= (L(a) \cup L(b))^* = (\{a\} \cup \{b\})^* = \{a, b\}^* \\ &= \{a, b\}^0 + \{a, b\}^1 + \{a, b\}^2 + \dots \\ &= \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \dots\} \quad \text{(所有由 a 和 b 组成的字符串)} \end{aligned}$$

$$(a^*b^*)^* = ?$$

$$\begin{aligned} (L(a^*b^*))^* &= (L(a^*)L(b^*))^* \\ &= (\{\varepsilon, a, aa, \dots\}\{\varepsilon, b, bb, \dots\})^* \\ &= L(\{\varepsilon, a, b, aa, ab, bb, \dots\})^* \\ &= \varepsilon + \{\varepsilon, a, b, aa, ab, bb, \dots\} + \{\varepsilon, a, b, aa, ab, bb, \dots\}^2 + \{\varepsilon, a, b, \\ &aa, ab, bb, \dots\}^3 + \dots \\ &= \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \dots\} \quad \text{(所有由 a 和 b 组成的字符串)} \end{aligned}$$

2. 正则表达式[Regular expressions, REs]

- 更多例子

- Keywords: 'if' 或 'else' 或 'then' 或 'for' ...
 - ✓ RE = 'i' 'f' + 'e' 'l' 's' 'e' + ... = 'if' + 'else' + 'then' + ...
- Numbers: 非空数字字符串
 - ✓ digit = '0' + '1' + '2' + '3' + '4' + '5' + '6' + '7' + '8' + '9'
 - ✓ 若定义integer = digit digit*, 请问'000' 是否为integer?
- Identifier: 以字母开头的字母或数字的字符串
 - ✓ letter = 'a' + 'b' + ... 'z' + 'A' + 'B' + ... + 'Z' = [a-zA-Z]
 - ✓ 若定义RE = letter(letter + digit)*, 请问RE符合C语言中标识符的定义吗?
- Whitespace: 由空格、换行符、制表符组成的非空序列
 - ✓ (' ' + '\n' + '\t')+

2. 正则表达式[Regular expressions, REs]

- 编程语言中的REs <https://docs.python.org/3/howto/regex.html>

Symbol	Meaning		
\d	Any decimal digit, i.e. [0-9]		
\D	Any non-digit char, i.e., [^0-9]		
\s	Any whitespace char, i.e., [\t\n\r\f\v]		
\S	Any non-whitespace char, i.e., [^ \t\n\r\f\v]		
\w	Any alphanumeric char, i.e., [a-zA-Z0-9_]		
\W	Any non-alphanumeric char, i.e., [^a-zA-Z0-9_]		
.	Any char	\.	Matching “.”
[a-f]	Char range	[^a-f]	Exclude range
^	Matching string start	\$	Matching string end
(...)	Capture matches		

2. 正则表达式[Regular expressions, REs]

- 正则表达式vs.正规文法[Regular grammar]

	正则表达式	正规文法
标识符集合	$l(l d)^*$ 其中: l 为a-z的字母, d 为0-9的数字	<ul style="list-style-type: none"> $\langle \text{标识符} \rangle \rightarrow l l \langle \text{字母数字} \rangle$ $\langle \text{字母数字} \rangle \rightarrow l d l \langle \text{字母数字} \rangle d \langle \text{字母数字} \rangle$
无符号整数	dd^*	$\langle \text{无符号整数} \rangle \rightarrow d d \langle \text{无符号整数} \rangle$

正规式比正规文法更容易让人理解单词是按怎样的规律构成的, 且可以从某个正规式**自动地**构造识别程序。

随堂练习(1)

- 写出C语言中标识符的正则表达式

`(_|letter)(_|letter|digit)*`

- 写出C语言中十进制数字的正则表达式

`0|([1-9][0-9]*)`

- 写出2的倍数的二进制表示的正则表达式

`(0|1)*0`