编译原理笔记

陈鸿峥

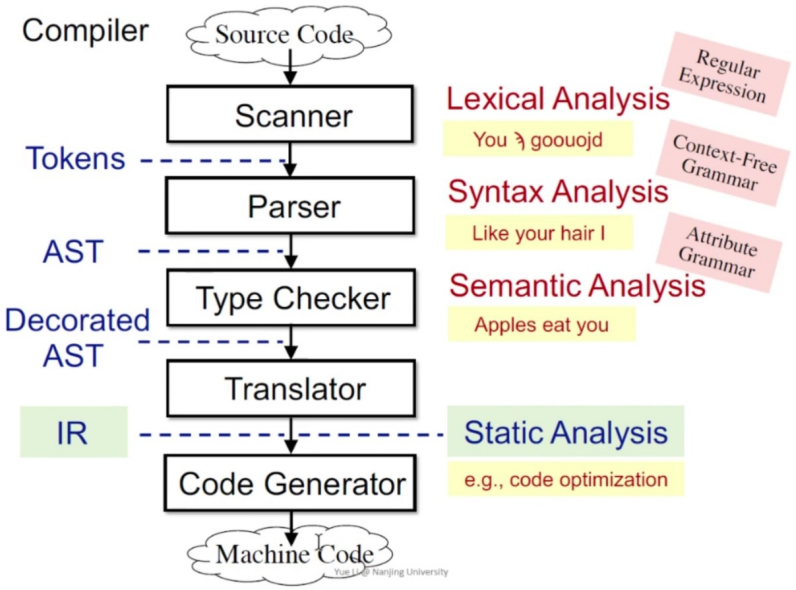
2020.05*

目录

本课程采用书目Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, *Compilers: Principles, Techniques & Tools (2nd ed)*，即大名鼎鼎的龙书。

1 简介

编译器的几个阶段如下，前端包括词法(lexical)、语法(syntax)、语义(semantic)分析，中端IR生成、优化，后端代码生成。



*Build 20200511

2 词法分析

分离词法分析和语法分析可以简化这两个任务，同时提升编译器的性能与兼容性。

2.1 基本定义

定义 1. 令牌*(token)*是一个令牌名字与可选属性值构成的对；模式*(pattern)*描述了每个词素*(lexeme)*要遵循什么规则；而词素（最小意义单位）则是源程序中一连串满足模式的字母，作为令牌的实例化。

例 1. 考虑*C*语句

```
printf("Total = %d\n", score);
```

其中`printf`和`score`是匹配*(match)*上令牌***id***模式的词素，而`"Total = %d\n"`是匹配上字面值***literal***的词素。

简单来讲，令牌是一个更大的概念，是同类词素的集合。比如一个令牌**comparison**的样例词素可以有`<=`和`!=`。

定义 2 (字母表与语言)**.** 字母表*(alphabet)*Σ是有限符号*(symbol)*的集合，如*ASCII*就是一个字母表。字符串*(string)*s是从字母表中抽取的有限符号的序列，$|s|$为字符串长度，ϵ为空串。语言*(language)*是字符串的可数集合。

例 2. 字母表$\Sigma = \{0, 1\}$，则$\{001, 1001\}$和$\{\}$都是定义在Σ上的语言。

定义 3 (字符串术语)**.** 前缀*(prefix)*和后缀*(suffix)*都可以包括ϵ。字串*(substring)*可通过删除任意前缀和任意后缀（包括**零个**）获得。真*(proper)*字串则不包含ϵ。子序列*(subsequence)*是删除**零个**或多个**不一定连续**的字母得到的字符串。

语言是一种集合，故集合运算也适用于语言。

并集(union)	$L \cup M$
连接(concatenation)/交集	LM
柯林闭包(Kleene closure)	$L^* = \cup_{i=0}^{\infty} L^i$
正闭包(positive)	$L^+ = \cup_{i=1}^{\infty} L^i$

2.2 正则表达式

定义 4 (正则表达式(regular expression, regex))**.** 正则表达式r定义了语言$L(r)$，以递归形式定义：

1. 奠基：
   - ϵ是正则表达式，即$L(\epsilon) = \{\epsilon\}$
   - $a \in \Sigma$是正则表达式，即$L(\mathbf{a}) = \{a\}$（这里用斜体代表符号，粗体代表符号对应的正则表达式）
2. 推论*(induction)*：若r和s都是正则表达式给出了语言$L(r)$和$L(s)$，则

- $(r)|(s)$是正则表达式，表示$L(r) \cup L(s)$
- $(r)(s)$是正则表达式，表示$L(r)L(s)$
- $(r)^*$是正则表达式，表示$(L(r))^*$
- (r)是正则表达式，表示$L(r)$

正则表达式表示的语言叫做正规集。

有以下运算规定：

- 一元运算符*有最高优先级，左结合
- 连接优先级次之，左结合
- |优先级**最低**，左结合

定义 5 (正则定义). $d_i \to r_i$，其中d_i都是名字，且各不相同。每个r_i是$\Sigma \cup \{d_1, \ldots, d_{i-1}\}$中符号上的正则表达式。

例 3. 比如C语言的标识符可记为

$$\begin{aligned} letter\_ &\to A|B|\cdots|Z|a|b|\cdots|z|\_ \\ digit &\to 0|1|\cdots|9 \\ id &\to letter\_(letter\_|digit)^* \end{aligned}$$

正则表达式的拓展[1]：

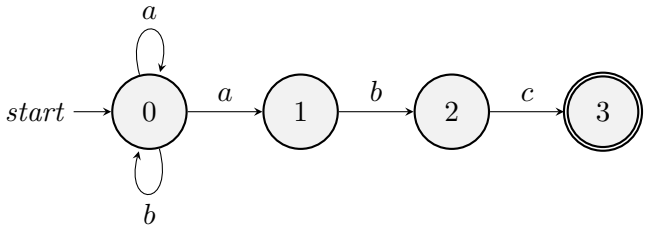
- r^+代表一个或多个
- $r?$代表零或一个
- $[a-z]$字母类

2.3 有限自动机

2.3.1 确定性/非确定性有限自动机

确定有限自动机(DFA)不可对ϵ进行移动，而且对于每一状态s，输入符号a，只有唯一一条出边标记为a；而非确定性有限自动机(NFA)可能有多种转换路径。有限状态集S，状态$s_0 \in S$为初始状态(start/initial)，$F \subset S$为终止状态(accepting/final)。

例 4. 识别语言$L((a|b)^*abb)$，下面为一个NFA



[1]更多可参见Regex101

判别字符串能否被DFA识别很简单，只需要读入字符按照状态转移表跳转，判断末态是不是终态即可。

Algorithm 1 基于DFA的识别算法

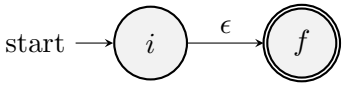
```
s = s_0
c = nextChar()
while (c!=eof) do
    s = move(s, c)
    c = nextChar()
if s ∈ F then
    return "yes"
elsereturn "no"
```

时间复杂度为$O(|str|)$。

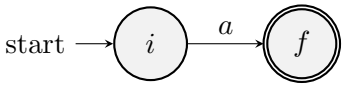
2.3.2 正则表达式转NFA

1. 奠基

   - 对于表达式ϵ，构建NFA

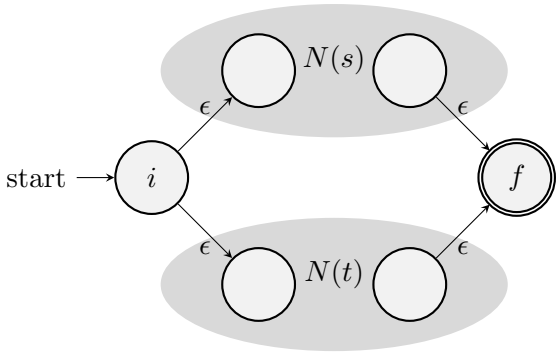


   - 对于任意子表达式$a \in \Sigma$，构建NFA

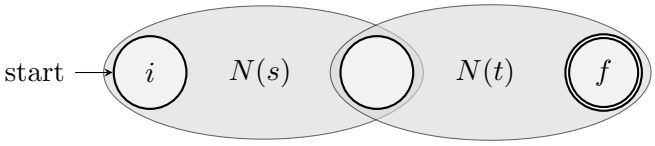


2. 推论

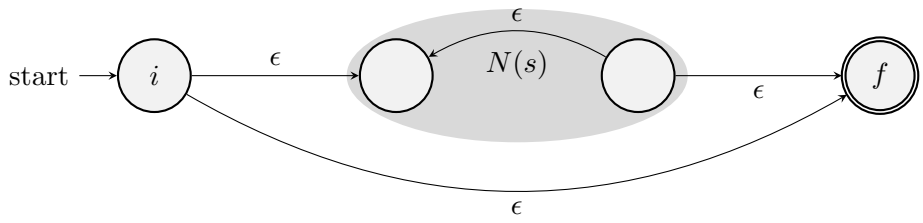
   - $r = s|t$，取并集



   - $r = st$，取连接



- $r = s^*$，Kleene闭包



2.3.3 NFA转DFA

定义 6 (ϵ闭包及$move$)**.** ϵ闭包是可通过NFA的ϵ边转换的状态。 $move(T, a)$为状态$s \in T$通过输入符号a可到达的新的状态。

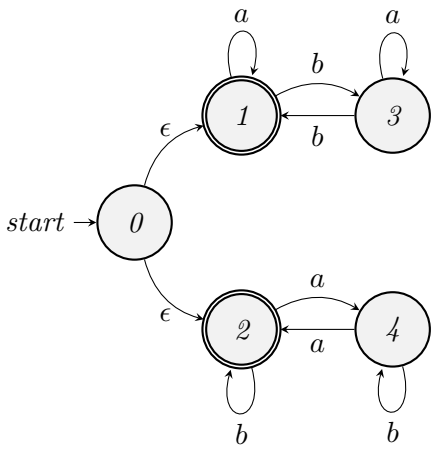
Algorithm 2 子集构造（NFA转DFA）

Require: NFA N

Ensure: DFA D（与N接受相同的语言）

1: ϵ-$closure(s_0)$是$Dstates$的唯一状态，且未被标记(unmarked)
2: **while** 在$Dstates$中还有未被标记的状态T **do**
3: 标记T
4: **for** 每一个输入符号a **do**
5: $U = \epsilon$-$closure(move(T, a))$
6: **if** $U \notin Dstates$ **then**
7: 将U作为未标记的状态加入$Dstates$
8: $Dtran[T, a] = U$

例 5. 考虑以下NFA:

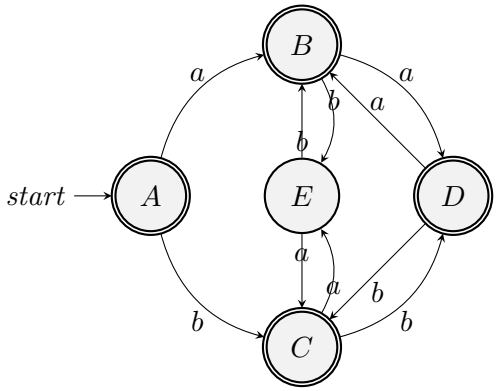


1. 这一NFA接受什么语言（用自然语言描述)?
2. 构造接受同一语言的DFA.

分析. 1. 含有偶数个a或偶数个b的由a、b构成的字符串，或者全是a或全是b

2. 由*subset construction*算法构造如下

NFA	*DFA*	a	b
$\{0, \underline{1}, \underline{2}\}$	A	$\{1, 4\}$	$\{2, 3\}$
$\{\underline{1}, 4\}$	B	$\{1, 2\}$	$\{3, 4\}$
$\{\underline{2}, 3\}$	C	$\{3, 4\}$	$\{1, 2\}$
$\{\underline{1}, \underline{2}\}$	D	$\{1, 4\}$	$\{2, 3\}$
$\{3, 4\}$	E	$\{2, 3\}$	$\{1, 4\}$



直接用NFA识别语言算法如下，需要每次算所有当前**可能状态**执行动作c后的ϵ闭包。

Algorithm 3 子集构造（NFA转DFA）

```
S = ε-closure(s_0)
c = nextChar()
while c!=eof do
    S = ε-closure(move(S, c))
    c = nextChar()
if S ∩ F!=∅ then
    return "yes"
else
    return "no"
```

定理 1. *DFA*，*NFA*和正则表达式三者的描述能力是一样的。