扫描/词法分析

词法分析（lexical analysis）又称扫描（scanning）

1、scanner的功能

词法分析程序（scanner）从源代码中读取字符形成若干个记号（token）。

scanner的任务

从源程序中读取字符并将其分成一个个的逻辑单元token，每个token是源程序中表示信息单元的字符序列。

scanner的输出形式

词法分析程序的输出形式是一个二元式：`(token类型,token的属性值)`。

2、token的分类

token { keywords, identifiers, special symbols

关键字（keywords）

例如`if`和`while`，它们是固定的字母串

标识符（identifiers）

由用户定义的串，通常由字母和数字组成且由字母开头

特殊符号（special symbols）

如算术符号`+`和`*`；一些多字符符号，`>=`

token

token有若干种，在程序语言中通常定义为枚举类型。比如在C语言中：

```
typedef enum{
    IF,THEN,ELSE,PLUS,MINUS,NUM,ID,...
} TokenType;
```

保留字（reversed words）

比如`IF`、`THEN`，代表了`if`和`then`

特殊符号（special words）

比如算术符号`PLUS`和`MINUS`，代表`+`和`-`

可表示多个字符串的token

can represent multiple strings，比如 `NUM` 和 `ID`，分别代表数字和标识符（identifiers）

token的attribute

token的string value

token必须和它所代表的字符串区分开来，比如 `IF` 必须和它代表的 `if` 区分开。

为了使这个区别更明显，由token代表的字符串有时被称作串值（**string value**）或它的词义（**lexeme**）。

某些token只有一个lexeme，比如reserved words；

有些token有多个词义：比如identifiers，多个字符串（根据用户定义得出）都是 `ID`，编译器会把这些标识符都记录在**symbol table**中。

token的attribute

任何与token相关联的值都被叫做**token的attribute**，string value就是attribute的一种。

token还可以有其他的attribute：

比如 `NUM`，它除了串值 `32767`，还有一个由其串值得到的数字值 `32767`

比如 `PLUS`，它除了串值 `+`，还有一个真正的算术操作 `+`

实际上，可以将token看作其attributes的结构体。

scanner计算attribute

*scanner*必须计算每一个token的若干必要的attribute。

比如 `NUM` 的串值比如被计算，而其数字值不必要被计算，因为其可由串值计算得到；另一方面，如果其数字值被计算出，那其串值可能被丢弃。

token record

既然需要计算token的attribute，那将其attribute收集起来则是十分有用的，这种数据结构叫做**token record**，其在C语言中定义可以如下：

```
typedef struct{
    TokenType tokenval;
    char* stringval;
    int numval;
} TokenRecord;
```

scanner的工作模式

scanner很少一次性将整个源程序转换为多一个个token。

实际上，scanner是在分析程序（**parser**，语法分析）的控制下进行操作的，它通过一个函数从输入中返回有关命令的下一个token，该函数有如下声明：

```
TokenType getToken(void);
```

该函数没有参数，参数保存在缓冲区或由系统输入设备提供。

3. 正则表达式（regular expressions）

定义

正则表达式表示字符串的格式，用来匹配相应的字符串，它完全由其所匹配的字符串来定义，一般用r来表示

$L(r)$

$L(r)$：r匹配的字符串的集合，称为**正则表达式r生成的语言**（language generated by the regular expression）

该语言与程序设计语言无关（至少在此是这样），其依赖于适用的字符集（character set），通常是ASCII字符的集合或其子集。

Σ

$L(r)$的合法字符集称为**字母表**（**alphabet**），记作Σ（**Sigma**），其中的元素称为**符号**（**symbol**）

patterns

在正则表达式中，所有的symbol都象征（indicate）着**patterns**。

元字符

元字符（**metacharacters** or **metasymbols**）有着特殊的含义，它可能不是Σ中的字符，或者我们区分不出它是作为元字符适用还是作为Σ中的普通字符适用（为了区分其作用，可以用**转义字符**（**escape character**）实现）

基本正则表达式（3个）

基本正则表达式是字母表中的单个字符且与自身匹配

- 若$a \in \Sigma$，则$L(a) = \{a\}$
- **空串**ε，$L(\varepsilon) = \{\varepsilon\}$
- **空集**$\emptyset$，$L(\emptyset) = \{\}$

正则表达式的三个基本运算及优先级

三个基本运算的优先级顺序为：`*` 优先级最高，连接 其次，`|` 优先级最低，可以用()改变这个优先级顺序

在该定义中，$\emptyset$、ε、|、*、(、)这6个符号都有元字符的含义。

- **选择**（**choice**）：$r|s$
  - $L(r|s) = L(r) \cup L(s)$，例如$L(a|b) = L(a) \cup L(b) = \{a, b\}$
- **连接**（**concatenation**）：rs
  - $L(rs) = L(r)L(s)$，例如$L((a|b)c) = L(a|b)L(r) = \{a, b\}\{c\} = \{ab, bc\}$
- **重复**（**repetition** or **closure**）：r^*
  - 有时称为Kleene闭包、克林闭包（Kleene closure），写作r^*，r的0次或多次连接
  - 例如$S^* = \{\varepsilon\} \cup S \cup SS \cup SSS \cup \ldots$

- $L(r^*) = L(r)^*$

正则表达式的名字

我们可以给一个较长的正则表达式一个名字，以便使用

例如：$digit = 0|1|2|\ldots|9$

正则表达式的两个事实

不同的正则表达式可以生成相同的语言（在实际中从未尝试着证实已找到了最简单的，这有两个原因）

并非用简单术语描述的所有串都可由正则表达式产生，我们将正则表达式可匹配的串的集合称为**正则集合**（regular set）

正则表达式的扩展

- 一个或多个重复：r^+
- 任意字符：.
- 字符范围：$[a-z], [0-9], [a-zA-Z]$
- 非：$\sim r$
- 可选的子表达式（即0个或1个重复）：$r?$

最长子串定理

当串可以是单个token也可以是多个token的序列时，则通常解释为单个token。

principle of longest substring：可组成单个token的字符的最长串在任何时候都是假设为下一个记号，*待考究*

有限自动机（finite automata）

它是对由正则表达式给出的串格式的识别算法

几个概念

- **状态（state）**
  - 在图中用圆圈表示
- **转换（transition）**
  - 在图中有带有箭头的线表示由一个状态向另一个状态的转换
- **开始状态（start state）**
  - 识别过程开始的状态，开始状态表示为一个不来自任何地方且指向它的箭头线无标识
- **接受状态（accepting state）**
  - 接受状态指匹配成功
- **出错转换（error transition）**
  - 来自出错状态的所有转换都要回到其本身
  - 出错状态是非接受的，因此一旦发生一个出错，则无法从这个出错状态逃出，而且再也不能接受串了

确定有限自动机（deterministic finite automata）

4. DFA定义和复组成(部分)

DFA：下一个状态由当前状态和当前输入字符唯一确定的自动机

一个确定有限自动机M由5部分组成：

- 字母表Σ
- 状态集合S
- 转换函数$T: S \times \Sigma \to S$
  - $S \times \Sigma$指的是S和Σ的**笛卡尔积**或叉积：集合对(s, c)，其中$s \in S, c \in \Sigma$。
  - 例如：$T(s, c) = s'$
- 初始状态s_0
  - $s_0 \in S$
- 接受状态集合A
  - $A \subset S$

5. ## ε-转换

$\varepsilon - transition$是无需考虑输入串就有可能发生的转换，它可以看作一个空串的"匹配"。

有两个好处：①方便合并DFA ②描述空串的匹配

- 使最初的确定性有限自动机保持完整并且只添加一个新的初始状态就可以将几个确定性有限自动机合并
- 有了它就可以清晰地描述出空串的匹配

6. ## 非确定性有限自动机（nondeterministic finite automata）

NFA

NFA和DFA有两点不同：①字母表ε ②转移函数T

- 字母表Σ扩充为$\Sigma \cup \varepsilon$
- 转移函数$T: S \times (\Sigma \cup \{\varepsilon\}) \to P(S)$，其中$P(S)$为$S$的幂集（power set）

任意个ε都可能在任一状态上引入到串中，并与NFA中ε转换的数量相对应。因此，NFA并不表示算法，但是却可以通过在每个非确定性选择中回溯的算法来模拟它

从正则表达式到DFA

将正则表达式翻译成DFA可以这样做：

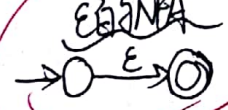
1. 将正则表达式翻译成NFA
2. 将NFA翻译成DFA

选择、重复都只会增加两个状态，并置不会增加状态，只是简单连接原NFA的状态和能力

7. ## 从正则表达式到NFA

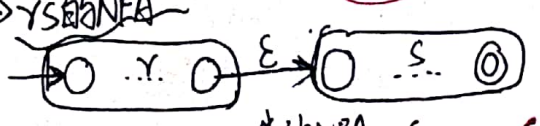
可以使用**Thompson构造（Thompson's construction）**，其利用ε转换将正则表达式对应的自动机连接起来形成整个正则表达式的自动机。

基本正则表达式的NFA

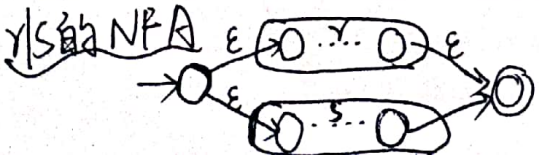


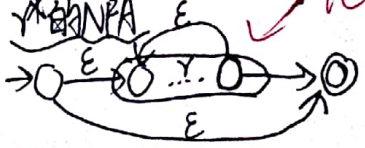
连接（concatenation）的NFA

选择（choice）对应的NFA

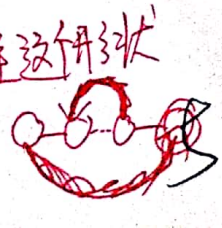


重复（repetition）的NFA





记住这种形状



8. 从NFA到DFA

状态s的ε-闭包 ($\varepsilon-closure$)

我们将单个状态s的$\varepsilon-closure$定义为：状态s可经过0个或多个$\varepsilon-closure$达到的状态的集合，并将这个集合记作$\bar{s}$。

一个状态的ε总是包含着该状态本身

状态集合的$\varepsilon-closure$

我们将一个状态集合的$\varepsilon-closure$定义为集合中每个状态的$\varepsilon-closure$的并。

记状态集合为S，则$\bar{S}=\bigcup_{s\ in\ S}\bar{s}$

子集构造 (subset construction)

设有NFA。M，则其DFA为$\bar{M}$。

①求M开始状态的ε闭包即$\bar{M}$的开始状态

②求每个状态在每个字符a上的转换，即求$\bar{S}\Rightarrow\bar{S_a}$

(旁注：循环直至没有新状态和转换)

A. 求S_a。若S中有状态t的转换($s\to t$)，则$t\in S_a$。

B. 求$\bar{S_a}$。求S_a的ε闭包。

$\bar{M}$中包含M接受状态的状态为$\bar{M}$的接受状态

注意 检查集合间的重复 一直计算到无新集合

9. DFA状态数最小化

定理：对于任意的DFA，都存在一个含有最少量的状态的等价DFA，且这个最小状态的DFA是唯一的。

状态在同一个集合内代表着状态等价

方法：

①将DFA状态分为两个集合：非接受状态集合和接受状态集合

②对于字符集上的每个字符a，

检查每个集合中是否有a区分状态s和t，若有，则拆开，并重复步骤2

若无，则定义了该集合到其转换集合的a转换

区分的定义：若a区分s和t，

即s和t在a上有不同的转换，有2种情况。

①转换至不同集合

②存在错误转换和正确转换

1.词法分析的功能任务

2.token的分类(3类)

3.正则表达式(regular expression)　　odd 奇数个　even 偶数个

作用

正则表达式r生成的语言

字母表(alphabet)Σ(sigma)

符号(symbol)

元字符(metacharacter|metasymbol) "" \

基本正则表达式(3个) a ε φ　　3个基本运算 rs r|s r*　及优先级 r* > rs > r|s

正则表达式的名字

正则表达式的两个事实

正则表达式的扩展:5个: r+ . [a-z] ~r r?

最长子串原理 → 消除二义性

4.DFA定义和组成 →(5部分) 开始状态、接受状态、状态集、转换函数、字母表

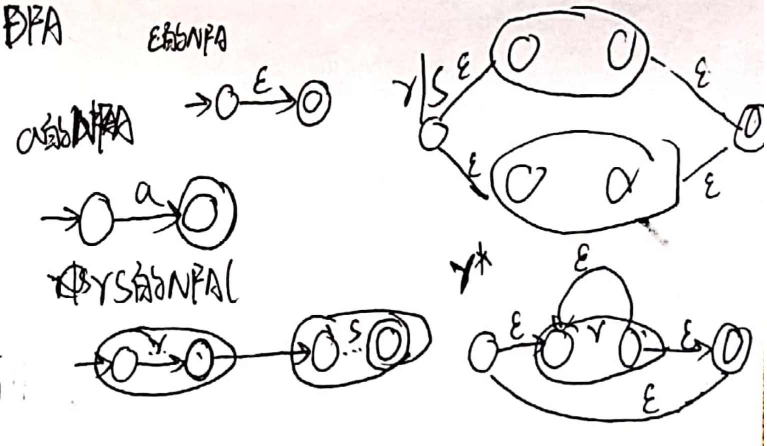
5.ε转换及其好处:①匹配空串ε ②合并FA

6.NFA与DFA的两点不同

7.从正则表达式到NFA

基本正则表达式的NFA

连接、选择、重复的NFA



8.NFA→DFA

{状态S的ε闭包 S̄ : 包括自身

{状态集S的ε闭包 S̄ : 元素ε闭包的并

子集构造法 2步骤: →遍历状态集和字符

① M̄　求开始状态s的ε闭包作为开始状态: Ā　　①求M̄的开始状态

② ②① 求 S'a → 写在一起　　②求每个状态集在每个字符a上的转换

②② 求 S̄'a 得 S →a S̄'a　/bar表示ε闭包　　即 S →a S'a　A.求S'a　B.求S̄'a

9.DFA状态数最小化(分割等价类, step1, step2)

定理: 唯一性

方法: ①先分成可接受状态集合和不可接受状态集合

②遍历状态集和字符寻找区分; 若无区分,则定义在a上的转换