## CH2 词法分析

### 2.1 扫描处理

1.某些记号只有一个词义:保留字;某些记号有无限多个语义:标识符 ID 表示.

### 2.2 正则表达式

1.一些 notation

| | |
|---|---|
| a | An ordinary character stands for itself. |
| ϵ | The empty string.<br>Another way to write the empty string. |
| M \| N | Alternation, choosing from *M* or *N*. |
| M · N | Concatenation, an *M* followed by an *N*. |
| MN | Another way to write concatenation. |
| M* | Repetition (zero or more times). |
| M+ | Repetition, one or more times. |
| M? | Optional, zero or one occurrence of *M*. |
| [a - zA - Z] | Character set alternation. |
| . | A period stands for any single character except newline. |
| "a.+*" | Quotation, a string in quotes stands for itself literally. |

2.RE 匹配优先匹配保留字;最长字串优先

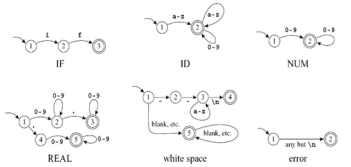
### 2.3 有穷自动机

1.DFA:M 由字母表Σ、状态集 S、转换函数 T:S×Σ→S、初始状态 $S_0 \in S$ 以及接受状态 A⊂S.

2.错误状态默认不画,但是存在;错误状态下的任何转移均回到自身,永远无法进入接受.

3.NFA:M 由字母表Σ、状态集 S、转换函数 $T:S\times(\Sigma\cup\{\varepsilon\})\to P(S)$、初始状态 $S_0$ 及接受状态 A 的集合.

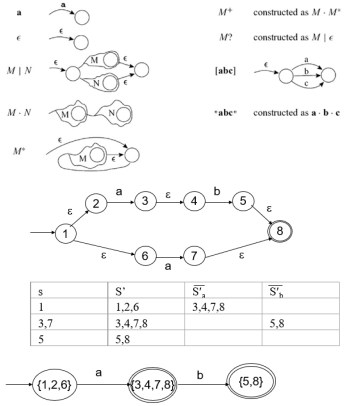
一些给出的基本 FA 图:




### 2.4 正则表达式到 DFA

**1.子集构造的过程**:首先列出所有状态的ε闭包;然后将初始状态的ε闭包作为新的初始状态;然后计算在每个新状态下在各个字符上的转移的闭包作为新的状态,转移自然成为新的转移;包含原接受状态的所有新状态都是接受状.

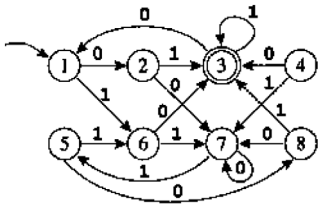
PS:ε闭包首先包含自身.




| s | S' | $S'_a$ | $S'_b$ |
|---|---|---|---|
| 1 | 1,2,6 | 3,4,7,8 | |
| 3,7 | 3,4,7,8 | | 5,8 |
| 5 | 5,8 | | |




4.DFA 状态数最小化:一种正向思路:从 DFA 的终态出发**逆着找**,看出边、终点一致的就是等价态.以 2.6 为例:{4, 6} {2, 8} {1, 5}等价.



5. NFA-DFA 的步骤:给出一个表头如下:新状态编号 新加字符 新加旧状态 新状态代号;

## CH3 上下文无关文法分析

### 3.1 CFG

1.左递归:定义 A 的推导式的右边第一个出现的是 A;右递归:定义 A 的推导式右边最后一个出现的是 A;

### 3.2 分析数和抽象语法树

**1.二义性(ambiguous)**:同一个串存在多个推导即多个分析树

2. 分析树 (concrete sytax tree)是一个作了标记 labeled 的树,内部节点是非终结符,树叶是终结符;对一个内部节点运用推导时,推导结果从左到右依次成为该内部节点的子节点

3.最左推导和前序编号对应,最右推导后序

4.AST(syntax tree)去除了终结符和非终结符信息,仅保留了语义信息;一般用左孩子右兄弟

### 3.3 Ambiguity 二义性

**1.定义**:带有两个不同的分析树的串的文法

**2.解决方法**①设置消歧规则 disambiguating rule,在每个二义性情况下指出哪个是对的.无需对文法进行修改,但是语法结构就不是单纯依赖文法了,还需要规则②修改文法.

4.修改文法时需要同时保证优先级和结合律 precedence and associativity

5.在语法树中,越接近根,越高,优先级越低;左递归导致左结合,右递归会右结合

6.相同优先级的运算符组叫 precedence cascade 优先级联

7.期中错题: A grammar is ambiguous if it has only one parse trees for all sentences.
**Answer: T**

## 自顶向下分析: LL(k)

第一个 L 是从左到右处理,第二个 L 是最左推导,1 代表仅使用 1 个符号预测分析方向

**递归下降**:1.将一个非终结符 A 的文法规则看作将识别 A 的一个过程的定义.递归下降需要使用 EBNF;将可选[]翻译成 if,将重复{}翻译成 while 循环

**LL(1)1. 动作**:① 生成**(generate)**,利用文法将栈顶的 N 替换成串,串反向进栈 ②**匹配(match)**:将栈顶的记号和下一个输入记号匹配 ③**接受(accept)**:接受字符串 ④**错误(error)**

2. LL(1)文法是无二义性的,对任意规则 $A\to\alpha1|\alpha2$, First(α1)∩First(α2)为空,否则不是 LL(1).

3. LL(1)面对重复和选择的解决方法:**消除左递归** left recursion removal 和**提取左因子** left factoring.

4. 简单直接左递归:$A\to A\alpha|\beta,\alpha\beta$是 N,且β不以 A 开头.$A\to\beta A',A'\to\alpha A'|\varepsilon$

5. 普遍直接左递归:$A\to A\alpha_1|A\alpha_2|\dots|A\alpha_n|\beta_1|\beta_2|\dots|\beta_m$
$A\to\beta_1A'|\beta_2A'|\dots|\beta_mA'$
$A\to\alpha_1A'|\alpha_2A'|\dots|\alpha_nA'|\varepsilon$

6. 提取左因子:$A\to\alpha\beta|\alpha\gamma.A\to\alpha A',A'\to\beta|\gamma$

**first follow sets**:

1.Fisrt 定义:令 X 为一个 T 或 N 或ε,Fisrt(X)由 T 或ε组成.① 若 X 为 T 或ε,Fisrt(X)={X}②若 X 为 N,对于每个产生式 $X\to X_1X_2\dots X_n$,First(X)都包含了 $First(X_1)-\{\varepsilon\}$.
若对于某个 i<n,所有的 $Fisrt(X_1),\dots,First(X_i)$ 都含有ε,则 First(X)也包括了 $First(X_{i+1})-\{\varepsilon\}$.若所有 $Fisrt(X_1),\dots,First(X_n)$ 都含有ε,则 First(X)也包含ε.

2.定理:A non-terminal A is nullable if and only if First(A) contains ε

3.Follow 定义:若 A 是一个 N,那么 Follow(A)由 T 和$组成.①若 A 是$,直接进入 Follow(A) ②若存在产生式 $B\to\alpha A\gamma$,则 First(γ)-{ε}在 Follow(A)中 ③若存在产生式 $B\to\alpha A\gamma$,且ε在 First(γ)中,则 Follow(A) 包括 Follow(B)

PS:③更常见的情况是 $B\to\alpha A$,那么 Follow(A) 包括 Follow(B)

4.First 关注点在产生式左边,Follow 在右边

**期中错题:Given production $A\to B\alpha C$, we have:Follow(A)⊂Follow(C),First(B)⊂First(A)**

5.LL(1)分析表 M[N,T]的构造算法:为每个非终结符 A 和产生式 $A\to\alpha$ 重复以下:
①对于 First(α)中的每个记号 a,都将 $A\to\alpha$ 添加到项目 M[A,a] ② 若 α 可空(nullable),则 Follow(A)中的每个元素 a (包括$),都将 $A\to\alpha$ 添加到 M[A,a]

例:下列文法构造 LL(1)

$Z\to d$ $Z\to XYZ$ $Y\to$ $Y\to c$ $X\to Y$ $X\to a$

1) 检查有无左递归左因子

2)nullable|First|Follow

| | nullable | FIRST | FOLLOW |
|---|---|---|---|
| X | yes | a c | a c d |
| Y | yes | c | a c d |
| Z | no | a c d | |

3) 按照算法填写规则进表

| | a | c | d |
|---|---|---|---|
| X | $X\to a$<br>$X\to Y$ | $X\to Y$ | $X\to Y$ |
| Y | $Y\to$ | $Y\to$<br>$Y\to c$ | $Y\to$ |
| Z | $Z\to XYZ$ | $Z\to XYZ$ | $Z\to d$<br>$Z\to XYZ$ |

6.LL(1)判别:A grammar in BNF is LL(1) if the following conditions are satisfied. ① For every production $A_i\to\alpha_1|\alpha_2|\dots|\alpha_n$, $First(\alpha_i)\cap First(\alpha_j)$ is empty for all $i$ and $j$, $1\le i,j\le n,i\ne j$ ② For every non-terminal A such that First(A) contains ε, First(A)∩Follow(A)empty.

## 自底向上分析: LR(k)

**Yacc 基于 LALR(1).**

1.动作为①shift,将 T 从输入开头移到栈顶②reduce 使用产生式 $A\to\alpha$ 将栈顶的α规约成 A③accept 分析栈为开始符号,输入栈为空时的动作④error ⑤**转移(goto)**: 吃 non-terminal

2.加一个 S′:新的开始符号

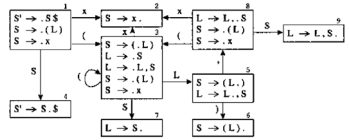
3.LR 分析器需要构造和使用一张 LR 分析表;而 LR 分析表的构造需要一个生成器.对于 start symbol 而言,需要新增 S′→S$规则来引入.

4.LR(0)的状态中包含带'.'的规则,'.'左边表示已读,右边表示未读;如果圆点'.'右边的第一个非终结符有规则,那么也写到这一状态中.

例如:如下文法构造 LR(0)

0 $S'\to S\$$
1 $S\to(L)$
2 $S\to x$
3 $L\to S$
4 $L\to L,S$

状态图如下:



由上面的状态图构造表如下:

| | ( | ) | x | . | $ | S | L |
|---|---|---|---|---|---|---|---|
| 1 | s3 | | s2 | | | g4 | |
| 2 | r2 | r2 | r2 | r2 | r2 | | |
| 3 | s3 | | s2 | | | g7 | g5 |
| 4 | | | | | a | | |
| 5 | | s6 | | s8 | | | |
| 6 | r1 | r1 | r1 | r1 | r1 | | |
| 7 | r3 | r3 | r3 | r3 | r3 | | |
| 8 | s3 | | s2 | | | g9 | |
| 9 | r4 | r4 | r4 | r4 | r4 | | |

5.s-r conflict:表中同时出现了 sn 和 rn,即构成 s-r 冲突

6.r-r conflict:表中同时出现两个及以上的 rn,则构成 r-r

7.LR(0)文法不可能二义的

8.A grammar is LR(0) if and only if ①Each state is a shift state (a state containing only shift items) or a reduce state (containing a single complete item).

这里的 r1 都是用 S'→S·规约,应该写成 accept

**SLR:**

**1.SLR 算法定义:**移进规则不变;规约时要求输入必须在属于 follow(A)的终结符的项中,而不是全放.就是比 LR(0)改进

$R \leftarrow \{\}$
**for each state** $I$ in $T$
  **for each item** $A \to \alpha.$ in $I$
    **for each token** $X$ in FOLLOW($A$)
      $R \leftarrow R \cup \{(I, X, A \to \alpha)\}$

2.SLR 不可能是二义性

3.自底向上右递归可能引起栈溢出,需要避免

4.SLR 中的两种冲突,sr 冲突使用消岐规则:优先移进;rr 冲突基本是设计出问题

**LR(1) and LALR(1)**

1.LR(1) items:[ A→α·β,a]前面是 LR(0)项,后面是 lookahead token

2.LR(1)的起始状态[S'→·S,$]的闭包

3.LR(1)中如何获得规则状态的闭包?难点在找 look ahead symbol.使用如下算法:

**Closure(*I*) =**
**repeat**
  **for *I* 中任意项 (A→*α*.*Xβ*,*z*)**
    **for 任意产生式 *X*→*γ***
      **for 任意 *w* ∈ First(*βz*)**
        ***I*:=*I* ∪ {(*X*→.*γ*, *w*)}**
**until I 没有变化**

| | | |
|---|---|---|
| $S' \to .\, S\,\&$ | ? |
| $S \to .\, V = E$ | $ |
| $S \to .\, E$ | $ |
| $E \to .\, V$ | $ |
| $V \to .\, x$ | $ |
| $V \to .\, * E$ | $ |
| $V \to .\, x$ | = |
| $V \to .\, * E$ | = |

| | | |
|---|---|---|
| $S' \to .\, S\,\&$ | ? |
| $S \to .\, V = E$ | $ |
| $S \to .\, E$ | $ |
| $E \to .\, V$ | $ |
| $V \to .\, x$ | $, = |
| $V \to .\, * E$ | $, = |

4.LR(1)表的构造如下:在圆点到末尾的位置发生规约,对应项中填入 rn(n 是规则编号);读入终结符则在对应项中写入 sn;读入非终结符则在对应项中写入 gn;读入 S 后在标记符位置填 a 表示 accept

5.LR(1)文法不可能二义性

6. LR(1)分析表示例

| | x | * | = | $ | S | E | V |
|---|---|---|---|---|---|---|---|
| 1 | s8 | s6 | | | g2 | g5 | g3 |
| 2 | | | | a | | | |
| 3 | | | s4 | r3 | | | |
| 4 | s11 | s13 | | | | g9 | g7 |
| 5 | | | | r2 | | | |
| 6 | s8 | s6 | | | | g10 | g12 |
| 7 | | | | r3 | | | |
| 8 | | | r4 | r4 | | | |
| 9 | | | | r1 | | | |
| 10 | | | r5 | r5 | | | |
| 11 | | | | r4 | | | |
| 12 | | | r3 | r3 | | | |
| 13 | s11 | s13 | | | | g14 | g7 |
| 14 | | | | r5 | | | |

8.LALR(1)将 look ahead symbol 进行合并:除了 look ahead symbol 不同以外全都相同的规则合并成一个,目的在于减少表项.

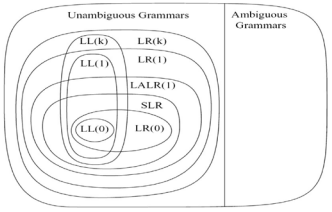
9.A grammar is an LALR(1) grammar if no parsing conflicts arise in the LALR(1) parsing algorithm.

10.如果文法是 LR(1),那么 LALR(1)中必然没有 sr 冲突,但是可能有 rr 冲突.

$[A \to \alpha.\ a] \quad [B \to \alpha.\ b] \Rightarrow [A \to \alpha.\ a/b] \quad [B \to \alpha.\ a/b]$

11.如果文法是 SLR(1),那么必然是 LALR(1).

12. 各类文法的层次如下:




13. 悬挂 else(dangling else)的处理:悬挂 else 会导致 s-r 冲突,冲突文法如下:

$S \to$ **if** $E$ **then** $S$ **else** $S$
$S \to$ **if** $E$ **then** $S$
$S \to$ **other**

解决办法:引入新非终结符用于 if-else 的匹配问题.M 是匹配的 else;U 是未匹配的 else

$S \to M$
$S \to U$
$M \to$ **if** $E$ **then** $M$ **else** $M$
$M \to$ **other**
$U \to$ **if** $E$ **then** $S$
$U \to$ **if** $E$ **then** $M$ **else** $U$

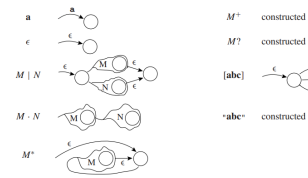



FIGURE 2.6. Translation of regular expressions to NFAs.

找出下面文法的 nullable、FIRST 和 FOLLOW 集合,然后构造 LL(1) 分析表。

```
0  S' -> S $          5  X -> B E E
1  S ->               6  X -> ( S )
2  S -> X S           7  X -> WORD
3  B -> \ begin { WORD }   8  X -> begin
4  E -> \ end { WORD }     9  X -> end
10 X -> \ WORD
```

nullable、FIRST 和 FOLLOW:

| | nullable | FIRST | FOLLOW |
|---|---|---|---|
| S' | False | \, [, $, WORD, begin, end | |
| S | True | \, [, WORD, begin, end | $, ], \ |
| B | False | \ | \, [, WORD, begin, end |
| E | False | \ | \, [, WORD, begin, end, ], $ |
| X | False | \, [, WORD, begin, end | \, [, WORD, begin, end, ], $ |

LL(1) 分析表:

| | \ | begin | end | WORD | [ | ] | $ |
|---|---|---|---|---|---|---|---|
| S' | S' → S $ | S' → S $ | S' → S $ | S' → S $ | S' → S $ | | S' → S $ |
| S | S → ε, S → X S | S → X S | S → X S | S → X S | S → X S | S → ε | S → ε |
| B | B → \ begin { WORD } | | | | | | |
| E | E → \ end { WORD } | | | | | | |
| X | X → B S E, X → \ WORD | X → begin | X → end | X → WORD | X → [ S ] | | |

计算下面文法的 nullable、FIRST、FOLLOW 集合:

```
S -> u B D z          B -> B v
B -> w                D -> E F
E -> y                E ->
F -> x                F ->
```

构造 LL(1) 分析表,给出证据说明该文法不是 LL(1) 文法。尽可能少地修改文法使他成为一个接收相同语言的 LL(1) 文法。

nullable、FIRST 和 FOLLOW:

| | nullable | FIRST | FOLLOW |
|---|---|---|---|
| S | False | u | |
| B | False | w | v, y, z, x |
| D | True | y, x | z |
| E | True | y | z, x |
| F | True | x | z |

LL(1) 分析表:

| | u | v | w | x | y | z |
|---|---|---|---|---|---|---|
| S | S → u B D z | | | | | |
| B | | | B → B v, B → w | | | |
| D | | | | D → E F | D → E F | D → E F |
| E | | | | E → ε | E → y | E → ε |
| F | | | | F → x | | F → ε |

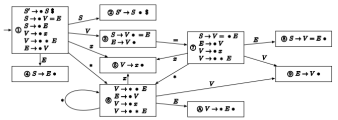
因为在上面的 LL(1) 分析表中有两个规则包含在同一个表项 (B, w) 中,所以不是 LL(1) 文法。

修改目标是将 B → B v 与 B → w 分层。将这两个规则替换为 B → w B', B' → v B' | ε 即可。

画出下面文法的 LR(0) 状态图,构造 SLR 分析表并指出冲突

```
0  S' -> S $          3  E -> V
1  S -> V = E         4  V -> x
2  S -> E             5  V -> * E
```

该文法的 LR(0) DFA 状态图如下:



构造 SLR 分析表还需要计算非终结符的 Follow 集:Follow($S$) = {$} / Follow($V$) = {=, $} / Follow($E$) = {=, $}。构造 SLR 分析表如下(前四列 Action,后三列 GOTO):

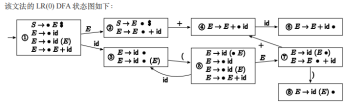
| | z | * | = | $ | S | V | E |
|---|---|---|---|---|---|---|---|
| 1 | s5 | s6 | | | g2 | g3 | g4 |
| 2 | | | | accept | | | |
| 3 | | | s7, r3 | r3 | | | |
| 4 | | | | r2 | | | |
| 5 | | | r4 | r4 | | | |
| 6 | s5 | s6 | | | | g9 | gA |
| 7 | s5 | s6 | | | | g9 | g8 |
| 8 | | | | r1 | | | |
| 9 | | | r3 | r3 | | | |
| A | | | r5 | r5 | | | |

因为单元格 (3, =) 中包含了 s7, r3 两个 action,出现冲突。

构造下面文法的 LR(0) DFA,并分析是否是 LR(0) / SLR / LR(1) 文法,给出证据。

```
0  S -> E $           2  E -> id ( E )
1  E -> id            3  E -> E + id
```

该文法的 LR(0) DFA 状态图如下:



构造 LR(0) 分析表:

| | id | ( | ) | + | $ | S | E |
|---|---|---|---|---|---|---|---|
| 1 | s3 | | | | | | g2 |
| 2 | | | | s4 | accept | | |
| 3 | r1 | s5, r1 | r1 | r1 | r1 | | |
| 4 | s6 | | | | | | |
| 5 | s3 | | | | | | g8 |
| 6 | r3 | r3 | | r3 | r3 | | |
| 7 | | | s8 | s4 | | | |
| 8 | r2 | r2 | | r2 | r2 | | |

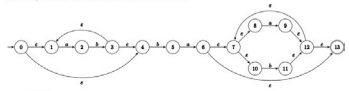
因为状态 2 中同时存在 shift 和 accept,并且 3,( 单元格中 s5 r1 冲突,所以不是 LR(0) 文法。

因为 Follow($E$) = {$, +, )},所以在 SLR 分析表中 3,( 单元格中不会填入 r1,没有冲突,而且 SLR(1) 可以 lookahead 一个 token,所以状态 2 中可以通过 token 区分 s4 和 accept,所以是 SLR 文法。同时因为 LR(1) 强于 SLR(1),所以也是 LR(1) 文法。

写出 (ab)*ba(a|b)* 的 NFA,并写出它的 minimal DFA。

**根据正则表达式转化为 NFA**

通过 Thompson 算法,可以直接将该正则表达式转化为如下 NFA:



**NFA 转化为 DFA**

利用子集构造法,从初始状态开始构造:

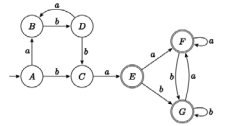
- $A = \varepsilon\text{-closure}(0) = \{0, 1, 4\}$
- $A$ 接收 $a$ 转移得到 $B = \varepsilon\text{-closure}(2) = \{2\}$
- $A$ 接收 $b$ 转移得到 $C = \varepsilon\text{-closure}(5) = \{5\}$
- $B$ 接收 $b$ 转移得到 $D = \varepsilon\text{-closure}(3) = \{1, 3, 4\}$
- $C$ 接收 $a$ 转移得到 $E = \varepsilon\text{-closure}(6) = \{6, 7, 8, 10, 13\}$
- $E$ 接收 $a$ 转移得到 $F = \varepsilon\text{-closure}(9) = \{7, 8, 9, 10, 12, 13\}$
- $F$ 接收 $a$ 转移得到 $G = \varepsilon\text{-closure}(11) = \{7, 8, 10, 11, 12, 13\}$
- $D$ 接收 $a$ 转移得到 $B$,接收 $b$ 转移得到 $C$,$F$ 和 $G$ 接收 $a$ 和 $b$ 均转移到 $F$ 和 $G$

得到如下状态转移表:

| 状态 | 接收 $a$ | 接收 $b$ |
|---|---|---|
| $A$ | $B$ | $C$ |
| $B$ | / | $D$ |
| $C$ | $E$ | / |
| $D$ | $B$ | $C$ |

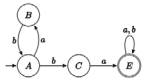
| 状态 | 接收 $a$ | 接收 $b$ |
|---|---|---|
| $E$ | $F$ | $G$ |
| $F$ | $F$ | $G$ |
| $G$ | $F$ | $G$ |

根据该表得到 DFA 如下(不包含 dead state):

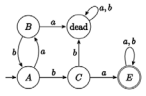


**最小化 DFA**

初始化划分为 $\{A, B, C, D\}, \{E, F, G\}$。根据加上状态转移表可知 $A, D$ 间和 $E, F, G$ 间均不可区分,最终的划分为 $\{A, D\}, \{B\}, \{C\}, \{E, F, G\}$。每组内选一代表状态,得到最小化 DFA 如下:



其中 $B$ 不可接收 $a$,$C$ 不可接收 $b$,更严谨地可以引入一个 dead state,得到如下 DFA:



1. Given a legal string of tokens for a CFG, there must be a unique parsing tree to derivate the string.

判断题 (4.0 分) (难易度:中)
A. True
B. False
正确答案: B
答案解释: 暂无

2. Left-recursive CFG cannot be LL(1).

判断题 (4.0 分) (难易度:中)
A. True
B. False
正确答案: A
答案解释: 暂无

3. The language L={ $a^n b^n | n > 1$ } can't be generated by any CFG.

判断题 (4.0 分) (难易度:中)
A. True
B. False
正确答案: B
答案解释: 暂无

4. There is only one parse tree for a string of an unambiguous grammar.

判断题 (4.0 分) (难易度:中)
A. True
B. False
正确答案: A
答案解释: 暂无

5. Both DFA and NFA can recognize regular set.

判断题 (4.0 分) (难易度:中)
A. True
B. False
正确答案: A
答案解释: 暂无

6. Which of the following string can be defined by theregular expression (a|b)*ab*c(a|b)c)*

单选题 (5.0 分) (难易度:中)
A. ccbbaa
B. ccaabb
C. ccbbaa
D. aabbcc
正确答案: D

7. Which action is notin a LL(1) parsing table?

单选题 (5.0 分) (难易度:中)
A. Match
B. Generate
C. Accept
D. Reduce
正确答案: D
答案解释: 暂无

8. The output of the parser is_____.

单选题 (5.0 分) (难易度:中)
A. token
B. syntax tree
C. target code
D. intermediate code
正确答案: B
答案解释: 暂无

9. _____reflects the derivation steps of a string.

单选题 (5.0 分) (难易度:中)
A. Syntax tree
B. Parse tree
C. Binary tree
D. Quadtree
正确答案: B
答案解释: 暂无

10. _____is commonly used to make operations left associative.

单选题 (5.0 分) (难易度:中)
A. Right recursion
B. Indirect recursion
C. Left recursion
D. Left factoring
正确答案: C
答案解释: 暂无