

编译原理复习

Garone Lombard

2023 年 12 月 7 日

摘要

编译原理烤漆复习手册

目录

1	文法和语言	4
1.1	文法的分类	4
1.2	(句型的) 短语	5
2	词法分析	6
2.1	有穷自动机	6
2.1.1	确定的有穷自动机 (DFA)	7
2.1.2	不确定的有穷自动机 (NFA)	9
2.1.3	从正则表达式到 DFA 的转换	10
2.1.4	DFA 的最小化	17
3	语法分析	20
3.1	自顶向下语法分析	20
3.2	文法转换	20
3.2.1	FIRST 集	21
3.2.2	FOLLOW 集	21
3.2.3	SELECT 集	21
3.3	自底向上语法分析	21

1 文法和语言

文法定义 四元组

$$G = (V_T, V_N, P, S) \quad (1)$$

1.1 文法的分类

0 型文法 无限制文法，只要求产生式的左部存在一个非终结符即可。

$$\forall \alpha \rightarrow \beta \in P \quad \alpha \text{中至少包含一个 } V_N \quad (2)$$

1 型文法 上下文有关文法，在0 型文法的基础上 进一步要求产生式的左部长度小于等于右部长度

$$\begin{aligned} \forall \alpha \rightarrow \beta \in P \quad |\alpha| \leq |\beta| \\ \text{产生式的一般形式} \quad \alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2 (\beta \neq \epsilon) \end{aligned} \quad (3)$$

2 型文法 上下文无关文法 (可以描述大部分程序设计语言的文法构造)，要求产生式的左部只能是一个非终结符

$$\forall \alpha \rightarrow \beta \in P \quad \alpha \in V_N \quad (4)$$

3 型文法 正则文法，只有两种形式，左线性文法 or 右线性文法 (注意是要求某一文法的所有产生式均符合左/右线性文法，而不只是 P1 满足左线性，P2 满足右线性)。

正则文法和正则表达式是等价的，对于任意一个正则文法 G，都存在定义同一语言的正则表达式 r，反之亦然。

- 左线性文法: $A \rightarrow \omega B$ or $A \rightarrow \omega$
- 右线性文法: $A \rightarrow B\omega$ or $A \rightarrow \omega$

例如

$$S \rightarrow a|b|c|d$$

$$S \rightarrow aT|bT|cT|dT$$

$$T \rightarrow a|b|c|d|0|1|2|3|4|5$$

$$T \rightarrow aT|bT|cT|dT|0T|1T|2T|3T|4T|5T$$

1.2 (句型的) 短语

已知文法

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow -E$$

$$E \rightarrow (E)$$

$$E \rightarrow idenfr$$

对于句型: $-(E + E)$, 可构造如下分析树

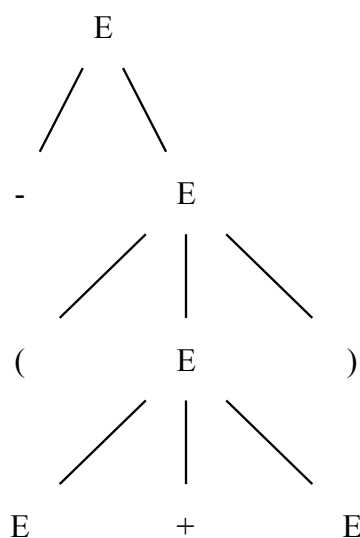


图 1: 分析树

- $E + E$ 是句型 $-(E + E)$ 相对于规则 $E \rightarrow E + E$ 的短语, 直接短语, 句柄 (子树层级为 1)
- $(E + E)$ 是句型 $-(E + E)$ 相对于规则 $E \rightarrow (E)$ 的短语
- $-(E + E)$ 是句型 $-(E + E)$ 相对于规则 $E \rightarrow -E$ 的短语

需要注意的是, 直接短语一定是某产生式的右部, 但某产生式的右部不一定是给定句型的直接短语

2 词法分析

2.1 有穷自动机

基本概念 省略...

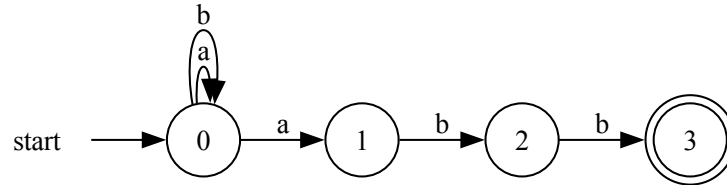


图 2: FA

最长前缀匹配原则 当输入串的多个前缀与一个或多个模式匹配时，总是选择最长的前缀匹配。也就是说在到达某个终态后，只要输入串上还有符号，FA 就会继续读入下一个符号，以寻求尽可能长度的匹配。

正则表达式和有穷自动机是等价的

2.1.1 确定的有穷自动机 (DFA)

定义 DFA 是一个五元组， $M = (S, \Sigma, \delta, s_0, F)$

- S : 有穷状态集
- Σ : 输入符号表
- δ : 状态转移函数， $\forall s \in S, a \in \Sigma, \delta(s, a)$ 表示从状态 s 出发，沿着标记为 a 的边所能到达的状态 (唯一)
- s_0 : 初始状态
- F : 终态集

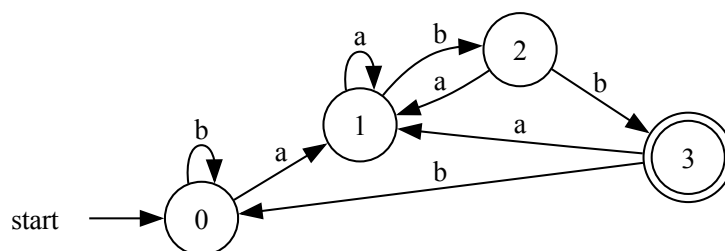


图 3: DFA

状态 \ 输入	a	b
0	1	0
1	1	2
2	1	3
3*	1	0

表 1: 转换表

DFA 的算法实现

- 输入：以文件结束符 eof 结尾的字符串 x ，DFA M 的开始状态 s_0 ，接受状态集合 F ，状态转换函数 $move(s, a)$
- 输出： M 接受则输出“yes”，拒绝则输出“no”
- 算法：

```

1 s=s0;
2 c=nextChar();
3 while(c!=eof){
4     s=move(s,c);
5     c=nextChar();
6 }
7 if(s in F) output("yes");

```



```
8 | else output("no");
```

2.1.2 不确定的有穷自动机 (NFA)

定义 NFA 是一个五元组, $M = (S, \Sigma, \delta, s_0, F)$

- S : 有穷状态集
- Σ : 输入符号表
- δ : 状态转移函数, $\forall s \in S, a \in \Sigma, \delta(s, a)$ 表示从状态 s 出发, 沿着标记为 a 的边所能到达的状态集合
- s_0 : 初始状态
- F : 终态集

NFA 和 DFA 的唯一区别就是状态转换函数的状态不唯一

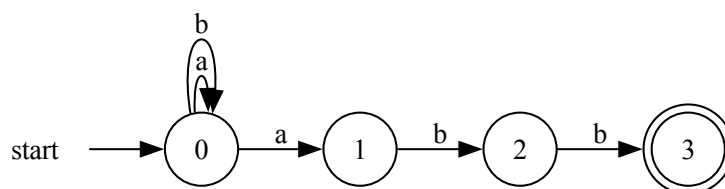


图 4: NFA

状态 \ 输入	a	b
0	{0,1}	{0}
1	ϕ	{2}
2	ϕ	{3}
3*	ϕ	ϕ

表 2: 转换表

DFA 和 NFA 具有等价性, 即对于任意一个 NFA , 都存在一个 DFA , 使得两者能够识别相同的语言, 反之亦然。

带有 ϵ 转换的 NFA ϵ - NFA , 是一种特殊的 NFA , 其状态转换函数 δ 中, $\delta(s, \epsilon)$ 表示从状态 s 出发, 不读入任何输入符号, 直接转移到下一个状态。

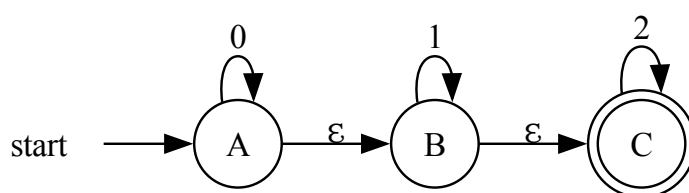


图 5: ϵ - NFA

可以证明, 对于任意一个 ϵ - NFA , 都存在一个 DFA , 使得两者能够识别相同的语言, 反之亦然。

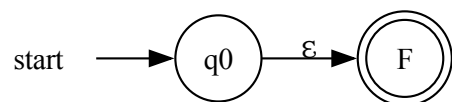
也就是说, DFA 、 NFA 、 ϵ - NFA 都具有等价性。

2.1.3 从正则表达式到 DFA 的转换

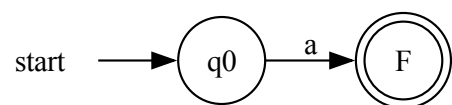
直接将正则表达式转换为 DFA 相当困难, 所以一般采取 $RE \rightarrow NFA \rightarrow DFA$ 的形式

正则表达式到 NFA 的转换 对应关系如下

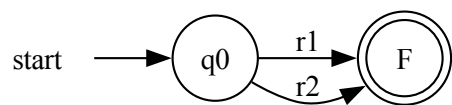
- ϵ 对应的 NFA



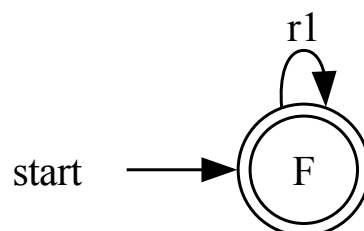
- 字母表 Σ 中符号 a 对应的 NFA



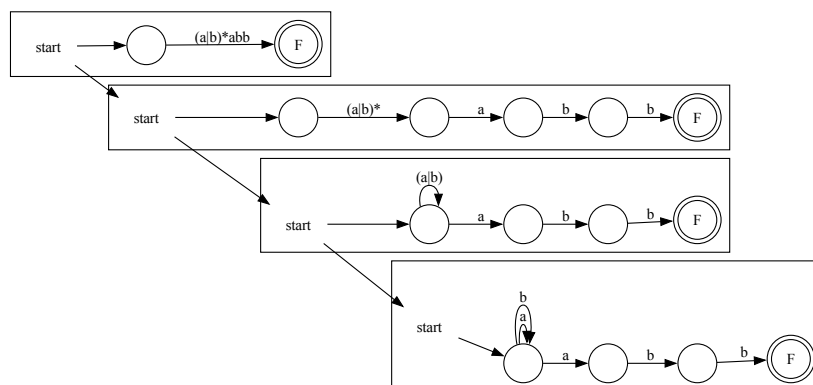
- $r = r_1 r_2$ 对应的 NFA



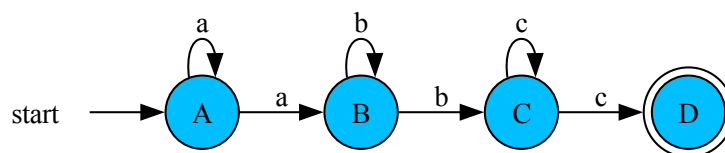
- $r = (r_1)^*$ 对应的 NFA



- $r = (a|b)^*abb$ 对应的 NFA



NFA 到 DFA 的转换 如下所示

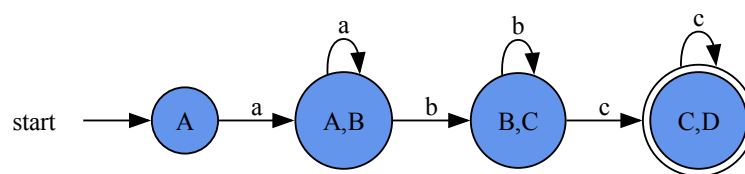


首先绘制状态转换表

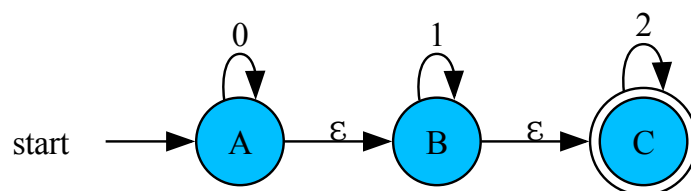
状态 \ 输入	a	b	c
A	{A,B}	ϕ	ϕ
B	ϕ	{B,C}	ϕ
C	ϕ	ϕ	{C,D}
D*	ϕ	ϕ	ϕ

表 3: 转换表

与 NFA 等价的 DFA 的每一个状态都是一个由 NFA 状态构成的集合



ϵ -NFA 到 DFA 的转换

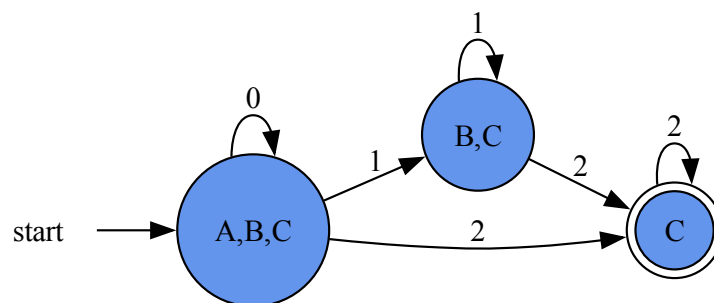


同样绘制状态转换表

输入 \ 状态	0	1	2
A	{A,B,C}	{B,C}	{C}
B	ϕ	{B,C}	{C}
C*	ϕ	ϕ	{C}

表 4: 转换表

需要注意的是，由于初始即可达 A,B,C，所以初始状态应该是 A,B,C 而不是 A



子集构造法

- 输入: NFA N
- 输出: DFA D
- 算法: 一开始, $\epsilon - \text{closure}(s_0)$ 是 Dstates 中唯一的, 且未加标记;

```

1 while(在Dstates中有一个未标记状态T){
2     tag(T);
3     for(每个输入符号a){
4         U=closure(move(T,a));
5         if(U not in Dstates){
6             add(Dstates,U);
7         }
8         Dtran[T,a]=U;
9     }
10 }

```

操作	描述
$\epsilon - \text{closure}(s)$	能够从 NFA 的开始状态只通过 ϵ 转换直接到达的 NFA 状态集合
$\epsilon - \text{closure}(T)$	能从集合 T 中的某个 NFA 状态只通过 ϵ 转换直接到达的 NFA 状态集合
$\text{move}(T, a)$	能从集合 T 中的某个 NFA 状态通过标号为 a 的转换到达的 NFA 状态的集合

NFA 的化简例题 ○○○○○○

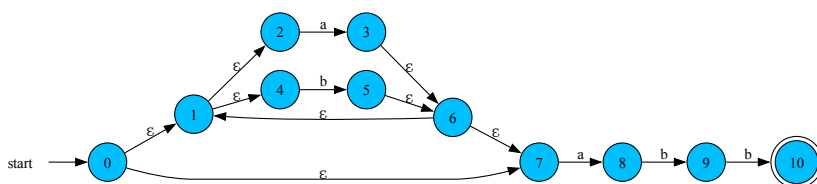


图 6: NFA

状态 \ 输入	a	b
0	{1,2,3,4,6,7,8}	{1,2,4,5,6,7}
1	{1,2,3,4,6,7}	{1,2,4,5,6,7}
2	{1,2,3,4,6,7}	ϕ
3	{1,2,3,4,6,7,8}	{1,2,4,5,6,7}
4	ϕ	{1,2,4,5,6,7}
5	{1,2,3,4,6,7,8}	{1,2,4,5,6,7}
6	{1,2,3,4,6,7,8}	{1,2,4,5,6,7}
7	{8}	ϕ
8	ϕ	{9}
9	ϕ	{10}
10*	ϕ	ϕ

表 5: 转换表 1(无效)

$\epsilon - closure(T_0)$	{0,1,2,4,7}	
状态 \ 输入	a	b
T0={0,1,2,4,7}	{1,2,3,4,6,7,8}	{1,2,4,5,6,7}
T1={1,2,3,4,6,7,8}	{1,2,3,4,6,7,8}	{1,2,4,5,6,7,9}
T2={1,2,4,5,6,7}	{1,2,3,4,6,7,8}	{1,2,4,5,6,7}
T3={1,2,4,5,6,7,9}	{1,2,3,4,6,7,8}	{1,2,4,5,6,7,10}
T4={1,2,4,5,6,7,10}	{1,2,3,4,6,7,8}	{1,2,4,5,6,7}

表 6: 转换表 2(正确)

状态 \ 输入	a	b
$T0=\{0,1,2,4,7\}$	T1	T2
$T1=\{1,2,3,4,6,7,8\}$	T1	T3
$T2=\{1,2,4,5,6,7\}$	T1	T2
$T3=\{1,2,4,5,6,7,9\}$	T1	T4
$T4^*=\{1,2,4,5,6,7,10\}$	T1	T2

表 7: 转换表 2(正确)

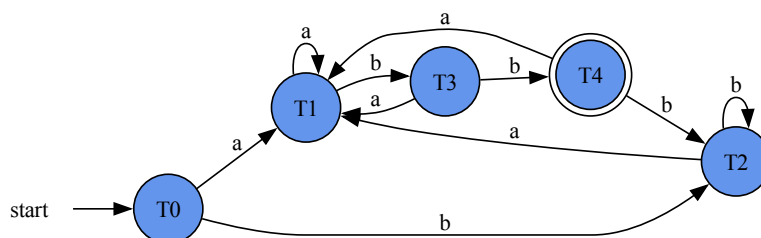


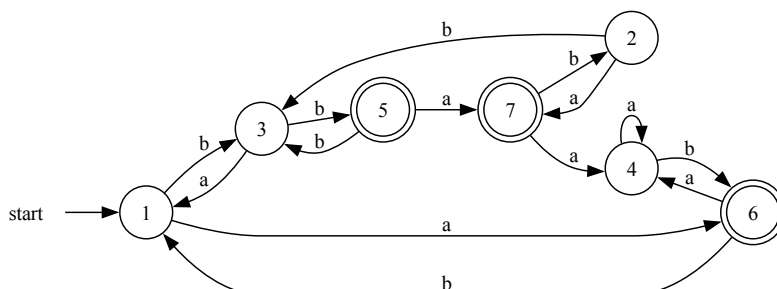
图 7: DFA

2.1.4 DFA 的最小化

概念 一个有穷自动机可以通过消除无用状态和合并等价状态来最小化

- 无用状态: 从该自动机的开始状态出发, 任何输入串都无法到达的状态 (从该状态出发, 没有通路抵达终态)
- 等价状态: 条件如下
 1. 一致性条件: 状态 s 和 t 必须同时为可接受状态或不可接受状态
 2. 蔓延性条件: 对于所有输入符号, 状态 s 和 t 必须转换到等价态

分割法 把一个 DFA(不含无用态) 的状态分成一些不相交的子集, 使得任何不同的两个子集的状态都是可区分的, 且同一个子集中的任何状态都是等价的



第一步都是固定的,把状态分为终态和非终态两个集合 $\{1,2,3,4\}, \{5,6,7\}$
 接下来考察 $\{1,2,3,4\}$ 是否可分

状态 \ 输入	a	b
1	6(E)	3(NE)
2	7(E)	3(NE)
3	1(NE)	5(E)
4	4(NE)	6(E)

因此可以将集合拆分为 $\{1,2\}, \{3,4\}, \{5,6,7\}$

状态 \ 输入	a	b
1	6(P3)	3(P2)
2	7(P3)	3(P2)

显然 $\{1,2\}$ 不可拆分

状态 \ 输入	a	b
3	1(P1)	5(P3)
4	4(P2)	6(P3)

$\{3,4\}$ 可拆分为 $\{3\},\{4\}$

此时集合为 $\{1,2\},\{3\},\{4\},\{5,6,7\}$

状态 \ 输入	a	b
5	7(P4)	3(P2)
6	4(P3)	1(P1)
7	4(P3)	2(P1)

$\{5,6,7\}$ 可拆分为 $\{5\},\{6,7\}$

因此最终得到的集合为 $\{1,2\},\{3\},\{4\},\{5\},\{6,7\}$

状态 \ 输入	a	b
1	6(P5)	3(P2)
2	7(P5)	3(P2)
3	1(P1)	5(P4)
4	4(P3)	6(P5)
5	7(P5)	3(P2)
6	4(P3)	1(P1)
7	4(P3)	2(P1)

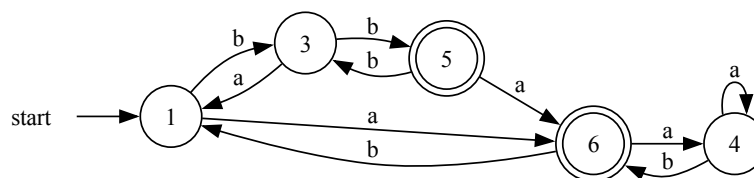


图 8: 最小化后的 DFA

3 语法分析

3.1 自顶向下语法分析

概念 从分析树的顶部向底部方向构造分析树，也就是从文法开始符号 S 从左向右推导句子 w 的过程

最左推导 总是选择每个句型的最左非终结符进行替换，其反过程称为最右规约

最右推导 总是选择每个句型的最右非终结符进行替换，其反过程称为最左规约

在自底向上的分析中，总是采用最左规约的方式，因此把最左规约成为规范规约，而把最右推导称为规范推导

最左推导和最右推导具备唯一性，因为对于每个句型而言，其最左/右终结符是唯一的

3.2 文法转换

左递归文法 如果一个文法中有一个非终结符 A 使得对某个串存在推导 $A \rightarrow^+ Aa$ ，那么这个文法就是左递归文法，这会使递归下降分析器陷入无限循环

解决办法如下 (可消除直接左递归, 其实是将其转化为了右递归)

$$\begin{aligned}
 A &\rightarrow A\alpha|\beta \\
 A &\rightarrow A\alpha \rightarrow A\alpha\alpha\alpha\alpha \rightarrow \beta\alpha\alpha\alpha\alpha \dots \\
 regex &= \beta\alpha^* \\
 A &\rightarrow \beta A' \\
 A' &\rightarrow \alpha A'|\epsilon
 \end{aligned} \tag{5}$$

同理，对于左递归推导 $E \rightarrow E + T|T$ ，消除左递归可得一下等价文法

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE'|\epsilon
 \end{aligned} \tag{6}$$

ϵ 产生式的使用时机 如果当前某终结符 A 与当前输入 a 不匹配时, 若存在 $A \rightarrow \epsilon$, 可以通过检查 a 是否可以出现在 A 的后面 (那不就是查看 A 的 FOLLOW 集吗?), 来决定是否可以使用产生式 $A \rightarrow \epsilon$

3.2.1 FIRST 集

概念 串首终结符, 给定一个文法符号串 a , a 的 $FIRST(a)$ 被定义为可以从 a 推导出的所有串首终结符的集合, 如果 $a \Rightarrow^* \epsilon$, 那么 ϵ 也在 $FIRST(a)$ 中

3.2.2 FOLLOW 集

概念 可能在某个句型中紧跟在 A 后边的终结符 a 的集合

如果 A 是某个句型的最右符号, 则将结束符 $\#$ 添加到 $FOLLOW(A)$ 中

3.2.3 SELECT 集

概念 产生式 $A \rightarrow \beta$ 的可选集是指可以选用该产生式进行推导时对应的输入符号的集合, 记为 $SELECT(A \rightarrow \beta)$

- $SELECT(A \rightarrow \alpha\beta) = \{\alpha\}$
- $SELECT(A \rightarrow \epsilon) = FOLLOW(A)$

如果每个具有相同左部的各个产生式的可选集互不相交的话, 就可以做出确定的分析

产生式 $A \rightarrow \alpha$ 的可选集 $SELECT$

- 如果 $\epsilon \notin FIRST(\alpha)$, 则 $SELECT(A \rightarrow \alpha) = FIRST(\alpha)$
- 如果 $\epsilon \in FIRST(\alpha)$, 则 $SELECT(A \rightarrow \alpha) = (FIRST(\alpha) - \{\epsilon\}) \cup FOLLOW(\alpha)$

3.3 自底向上语法分析