

目录

第一章 DFA 和 NFA	3
1.1 DFA 和 NFA	3
1.1.1 NFA	3
1.1.2 带空转移的 NFA	5
第二章 正则语言和正则表达式	7
2.1 正则表达式的定义	7
2.1.1 语言之间的运算	7
2.2 正则表达式的性质	9
2.2.1 正则表达式和 DFA 的等价性: 根据 DFA 构造正则表 达式	9
2.2.2 正则表达式和 DFA 的等价性: 根据正则表达式构造 DFA	10
2.3 正则表达式的代数性质	10
第三章 正则语言的性质	11
3.1 泵引理和泵引理的证明	11
3.2 运算的封闭性	13
3.3 判定正则语言	13
3.4 DFA 的最小化	13
3.4.1 填表算法:	13
3.5 Regular Expressions in Unix	14
第四章 上下文无关文法	15
4.1 上下文无关文法	15
4.1.1 归约和派生	16

4.1.2	文法的语言	17
4.2	语法分析树	18
4.3	文法和语言的歧义性	19
4.4	文法的化简和设计	19
4.4.1	化简	19
第五章	两种范式和下推自动机的定义	23
5.1	两种范式	23
5.2	下推自动机	24
5.2.1	下推自动机的定义	25

第一章 DFA 和 NFA

1.1 DFA 和 NFA

1.1.1 NFA

NFA 的性质

NFA, 可以允许多个现态, 以及一个现态可以有多个次态. 仅仅是状态转移函数的定义有着不同. 其定义如下:

$$\delta: Q \times \Sigma \rightarrow 2^Q \quad (1.1)$$

而 DFA 的状态转移函数长这个样子:

$$\delta: Q \times \Sigma \rightarrow Q \quad (1.2)$$

Example 1.1.1. 检测末尾为 01 的只由 0 和 1 组成的序列的 NFA. 状态转移函数如下¹:

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	\emptyset	$\{q_1\}$
*q_2	\emptyset	\emptyset

Definition 1.1.2 (状态转移函数的拓展). δ 可以扩展为 $\hat{\delta}: Q \times \Sigma^* \rightarrow 2^Q$, 定义是递归的, 如下:

$$\hat{\delta}(q, w) = \begin{cases} q & , \text{if } w = \epsilon \\ \bigcup_{p \in \hat{\delta}(q, x)} \delta(p, a) & , \text{if } w = xa \end{cases}$$

¹图就不画了, 真不会画

这个函数告诉了我们, 给定初态 q_0 , 以及一个输入的字符串 w 之后我们的现状 (集合) 是什么.

Theorem 1.1.3 (NFA 和 DFA 之间的等价性). 一个语言 L 能被 NFA 接收, 当且仅当能被 DFA 接收

证明. 我们需要证明的是: 给定一个 N , 存在一个 D 使得 $L(N) = L(D)$, 以及反过来的版本, 给定一个 D 存在一个 N 使得 $L(D) = L(N)$. 注意到 DFA 是一个 NFA 的一个特例, 于是我们能够知道: $\forall D \exists N (L(D) = L(N))$. 于是我们要证明的便是, 如果 L 能够被 NFA 接收, 那么 L 能够被 DFA 接收.

使用构造法. 给定一个 NFA, 能够构造出一个 DFA 接收 L . 这里使用的构造法称为子集构造法. 原理很简单: 对于一个 NFA, 其 ‘现态集合’ 虽然是一个集合, 但是可以视为一个 DFA 的现态. 因为状态是有限的, 所以说, 状态集合的幂集也是有限的, 若是能够将 NFA 看作是一个 DFA, 那么其现态就是一个 Σ 的子集, 当然次态也是一个 Σ 的子集.

我们给定 NFA $(Q, \Sigma, \delta, q_0, F)$, 构造 DFA $(Q', \Sigma, \delta', \{q_0\}, F')$, 其中

$$Q' = 2^\Sigma$$

$$F' = \{S \mid S \in Q', S \cap F \neq \emptyset\}$$

注意到, $\{q_0\} \in Q'$, 这是我们加上 $\{\}$ 的原因. 其中 δ' 的定义如下:

$$\delta'(S, a) = \bigcup_{q_i \in S} \delta(q_i, a)$$

我们需要证明 $\forall s \in \Sigma^* (\hat{\delta}'(q_0, s) = \hat{\delta}(q_0, s))$. 证明等会在说. □

Theorem 1.1.4. $\hat{\delta}(q_0, s) = \hat{\delta}'(q_0, s)$

证明. 对字符串的长度进行归纳: 当长度为 $n = 0$

$$\hat{\delta}'(\{q_0\}, \epsilon) = \{q_0\} = \hat{\delta}(q_0, \epsilon) \quad (1.3)$$

当长度为 $n = k$, 设假设成立.

当长度为 $n = k + 1$, 设 $w = xa$, 那么 $\hat{\delta}(q_0, w) = \bigcup_{p \in \hat{\delta}(q_0, x)} \delta(p, a)$. 而 $\hat{\delta}'(q_0, w) = \delta'(\delta'(q_0, x), a)$. 后者的定义为 $\bigcup_{p \in \delta'(q_0, x)} \delta(p, a)$. 因为 x 的长度为 k , 所以 $\hat{\delta}'(q_0, x) = \hat{\delta}(q_0, x)$, 因此 $\hat{\delta}'(q_0, w) = \hat{\delta}(q_0, w)$ 恒成立.

$$\delta'(\delta'(q_0, x)) = \bigcup_{p \in \delta'(q_0, x)} \delta(p, a) = \bigcup_{p \in \hat{\delta}(q_0, x)} \delta(p, a) = \hat{\delta}(q_0, w) \quad (1.4)$$

□

我们知道了 $\hat{\delta}(q_0, w) = \hat{\delta}'(q_0, w)$ 之后, 就能够证明 $L(N) = L(D)$ 了. 只需要注意到 $L(N)$ 的定义: 对于任意的 $s \in L(N)$, 都有 $\hat{\delta}'(q_0, s) \cap F \neq \emptyset$. 我们知晓了这个定义之后就能够证明 $L(N) = L(D)$ 了, 这是显然的.

NFA 转化为 DFA 的方法

使用上面的子集构造法就能够进行转化, 随后我们将不需要的状态进行删除即可.

1.1.2 带空转移的 NFA

空转移的意思便是, 在没有输入的时候状态可以自己发生转移, 这样的自动机记为 ϵ -NFA, 其中 ϵ 是空字符串的意思. 在本小节之后, 如果没有特别说明, NFA 包括 ϵ -NFA 和 NFA.

ϵ -NFA 的定义

仅有一处和 NFA 不同, 状态转移函数的定义不同:

$$\delta: Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q \quad (1.5)$$

并且, 我们需要理解什么是 ϵ : 对于一个输入的字符串 s , 其等价于 $s\epsilon$, 也等价于 $s\epsilon\epsilon$. 当然, ϵ 插在中间也是可以的.

Example 1.1.5.

Definition 1.1.6 (ϵ closure). 给定一个状态集合 S , 那么 S 的 ϵ closure 为 S 并上那些当前状态通过空转能够达到的状态, 记为 $\bar{\epsilon}(S)$. 记 $\bigcup_{s \in S} \delta(s, \epsilon)$ 为 $\epsilon_1(S)$, 并且记 $\bigcup_{s \in \epsilon_1(S)} \delta(s, \epsilon)$ 为 $\epsilon_2(S)$, 那么就能够得到 $\bar{\epsilon}(S)$ 的表达式:

$$\bar{\epsilon}(S) = S \cup \epsilon_1(S) \cup \epsilon_2(S) \cup \dots \quad (1.6)$$

当然可以写为

$$\bar{\epsilon}(S) = \left(\bigcup_{n=1}^{\infty} \epsilon_n(S) \right) \cup S \quad (1.7)$$

特别地, 可以称呼 S 为 $\epsilon_0(S)$. 表达式还可以再精简一点.

引入 ϵ closure 的概念是为了定义 $\hat{\delta}$, 以及下文之中 ϵ -NFA 向 DFA 转换.

Definition 1.1.7 (δ 函数的扩展). 同样地, δ' 的定义也是递归的. 定义如下:

$$\delta'(q_0, w) = \begin{cases} \bar{\epsilon}(q_0) & , \text{ if } w = \epsilon \\ \bar{\epsilon}(\bigcup_{p \in \hat{\delta}'(q_0, x)} \delta(p, a)) & , \text{ if } w = xa \end{cases} \quad (1.8)$$

空转 NFA 的性质

实际上 ϵ - NFA 也是和 NFA 等价的, 这就是说给定一个 ϵ - NFA , 我们能够构造出一个 NFA. 且有 $L(N) = L(N')$.

Theorem 1.1.8 (ϵ -NFA 和 DFA 等价性).

空转 NFA 转换为 DFA

方法是和前面类似的.

第二章 正则语言和正则表达式

2.1 正则表达式的定义

正则表达式 的递归定义.

Definition 2.1.1. 分为基础部分, 归纳部分. 基础部分:

- 1 空集是一个正则表达式, 匹配空语言.
- 2 ϵ 是一个正则表达式, 匹配 $\{\epsilon\}$
- 3 对于任意一个符号 a , 其也是一个正则表达式.

归纳部分: E_1, E_2 都是正则表达式, 那么 $E_1 + E_2$ 也是正则表达式, $+$ 表示或; $E_1 E_2$ 也是正则表达式; E_1^*, E_2^* 也是正则表达式; 括号用来表示运算的顺序.

2.1.1 语言之间的运算

我们给定了语言之间的运算, 实际上就是集合之间的运算, 我们根据这个规定, 能够对正则表达式进行运算.

语言之间的运算 给定两个语言 L, M , 语言是字符串的集合. 语言之间有乘法, 幂, 闭包等运算. 注意到乘法仅仅是一个称呼.

Definition 2.1.2 (运算). 下面列出四种运算.

$$L \cdot M \equiv \{w \mid w = xy, x \in L, y \in M\}$$

$$L + M \equiv L \cup M$$

$$L^2 = L \cdot L \text{ 特别地, } L^0 = \{\epsilon\}$$

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

运算和运算的优先级 对于三种运算, 我们可以将闭包看为是一个幂, 那么这些的运算顺序就如同我们平常的运算一样: 1. 考虑幂; 2. 考虑乘积; 3. 考虑加法.

Example 2.1.3. 下面正则表达式的语言是什么?

$$1 + 01^* \tag{2.1}$$

Example 2.1.4.

$$(a + b)^*(a + bb) \tag{2.2}$$

a, b 组成的, 以 a 或者 bb 结尾的字符串.

2.2 正则表达式的性质

2.2.1 正则表达式和 DFA 的等价性：根据 DFA 构造正则表达式

正如我们前面已经了解到的那样, 正则表达式和 DFA 的描述能力实际上是一样的, 也就是说 $L(E) = L(D)$, 我们将证明这种等价性. 我们分为两个方向, 给定一个正则表达式, 构造一个 DFA, 和给定一个 DFA 构造一个正则表达式. 有两种方法: 1. 递归法; 2. 状态消除法. 我们接下来就讲讲这两种方法.

递归法 递归法是一种算法, 其实是动态规划的思想. 给 DFA 进行编号 (从 1 开始编号, 注意这点), 定义 $R_{i,j}^k$, 其表示的是, 由 i 到 j 的, 中间经过的状态的编号不超过 k 的, 一个字符串的集合.

考虑其初始状态, $k = 0$ 的时候, 分 $i = j, i \neq j$ 讨论. 对于 $i = j$ 的时候, $R_{i,j}^k$ 为 i 到 i 状态的字符的闭包. 对于 $i \neq j$, 则, 考虑状态 i 到 j 的字符.

其后, 开始遍历, 对于 $k > 1$, 我们有

$$R_{i,j}^{(k)} = R_{i,j}^{(k-1)} + R_{i,k}^{(k-1)} (R_{k,k}^{(k-1)})^* R_{k,j}^{(k-1)} \quad (2.3)$$

其中 $R_{i,j}^{(k-1)}$ 是指不经过 k 的路径¹. 我们能够看出, 这个递归方法是眼熟的, 这不是 tm 的动态规划吗? 随后就好理解了. 甚至说, 我们能够写出对应的代码.

故, 我们和 DFA 等价的正则表达式是

$$\sum_{j \in F} R_{1,j}^{(n)} \quad (2.4)$$

Example 2.2.1. 见 ppt, 流程还是比较麻烦的.

状态消除法 我超, 这个有点难搞, 不会画图, 反正也不是很难, 要不就直接看 ppt 得了. 根据几个规则在图上化简, 看起来是这个学校喜欢的东西.

¹路径实际上是一个字符串!

2.2.2 正则表达式和 DFA 的等价性：根据正则表达式构造 DFA

我们根据正则表达式的定义出发, 可以对正则表达式构造规则, 建立起 DFA 的构造规则. 具体的符号还是见 ppt, 这里画不了图.

Example 2.2.2. 将 $0(0 + 1)^*$ 转化为 ϵ -NFA.

2.3 正则表达式的代数性质

比较简单, 自己看吧!

第三章 正则语言的性质

我们这边的题型是如何判定一个语言不是正则的. 既然他这么问了, 那么他大概率不是正则的.

3.1 泵引理和泵引理的证明

所用到的方法便是下面介绍的泵引理. 需要注意到, 符合泵引理的语言不一定是正则的, 但不满足的一定不是正则的, 也就是说, 泵引理实际上是正则的必要条件.

鸽巢原理 原理应该很简单, 之前在其他地方也有涉及¹, 其实是一个挺有意思的定理.

泵引理 泵的意思是, 正则语言之中的有些语句之中某些部分, 能够被泵出来. 若是 $w = xyz$ 是满足这个条件的, 就有 y 可以被泵出来, 也就是 $w^i = xy^iz$, $i = 1, 2, \dots$, 也有 $w \in L$. 所以形象地称呼其为泵引理.

Theorem 3.1.1. 对于正则语言 L , 存在 N 使得对于长度大于等于 N 的语句, i.e. $w \mid |w| \geq N$, 可以分为三个部分 xyz , i.e. $w = xyz$, 并且满足:

1. $y \neq \epsilon$, 这是说泵出的部分不能为空
2. $|xy| \leq N$, 这描述了 y 的位置.
3. $xy^*z \in L$, 这是说 y 可以被泵.

¹比如说, 线性代数之中有一题: 证明对于一个 n 阶方阵, 存在 $k \leq n$ 使得 $r(A^k) = r(A^{k+1}) = r(A^{k+2}) = \dots$

证明 证明需要用到前面的定理.

考虑这个语言对应的 DFA. 并且给定了 DFA 的状态编号. 并且设 N 为状态的数目.

设 $w = a_1 a_2 a_3 \dots a_m$, $m \geq N$, $q_i = \hat{\delta}(q_0, a_1 \dots a_i)$. q_i 代表的是, DFA 接收到 w 的前 i 字母的时候所处的状态. 这个证明的重点在于: 使用鸽巢原理知道, 至少存在一对 i, j , $i \neq j$, $0 \leq i, j \leq N$, 使得 $q_i = q_j$ ².

我们设 (q_n, q_m) 代表状态 q_n 到 q_m 需要的字符串集合, viz., $w \in (q_n, q_m) \iff \hat{\delta}(q_n, w) = q_m$, 那么我们有

$$(q_n, q_m) = (q_n, q_n)^*(q_n, q_m)$$

那么我们知道 w 之中 i, j 之间的字符串 $w_{i,j} \in (q_i, q_i)$, 那么

$$w_{i+1,j}^* w_{j+1,m} \in (q_i, q_m)$$

自然有 $w_{1,i} w_{i+1,j}^* w_{j+1,m} \in L = (q_0, q_m)$. 于是自然地, $|w_{i+1,j}| \neq 0$, 且 $|w_{1,j}| \leq N$.

通过泵引理证明某些语言并不是正则的. 我们称 w 是满足泵引理的, 如果存在对 w 的划分 xyz , 都有 $xy^*z \in L$. 那么我们可以重述泵引理.

$$L \text{ is regular} \implies \exists N \in \mathbb{N} \forall w \in L, |w| \geq N (w \text{ 满足泵引理})$$

取其逆否命题, 就有

$$\neg(\exists N \in \mathbb{N} \forall w \in L, |w| \geq N (w \text{ 满足泵引理})) \implies L \text{ is not regular}$$

但是我们有

$$\begin{aligned} & \neg(\exists N \in \mathbb{N} \forall w \in L, |w| \geq N (w \text{ 满足泵引理})) \\ &= \forall N \in \mathbb{N}, \exists w \in L (\forall x, y, z (xyz = w \rightarrow (\exists i \in \mathbb{N}, xy^i z \notin L))) \end{aligned}$$

如果说我们证明了这个命题, 那么我们就能够证明 L 不是正则的.

Example 3.1.2. 证明 $L_{01} = \{0^n 1^n \mid n \geq 0\}$ 不是正则的.

Example 3.1.3. 证明 $L = \{0^i 1^j \mid i > j\}$ 不是正则的.

Example 3.1.4. 证明 $L = \{a^{n!} \mid n \geq 0\}$ 不是正则的.

²当 $i = N$ 的时候, 经过的状态已经达到了 $N + 1$ 个.

3.2 运算的封闭性

正则语言再某些运算之后得到的新语言仍然是正则的. 称正则语言再这些运算下封闭.

并 补 交 差 反转 同态 逆同态

3.3 判定正则语言

三个判定问题:

w 是否属于描述的语言

语言是否为空 是否为无穷的.

语言的等价性问题.

对于非空的, 检查全部长度小于 n 串.

是否为无穷, 检查全部长度有 n 到 $2n - 1$ 的串.

3.4 DFA 的最小化

最小化

将冗余的状态消除

$$\hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$$

不等价

对于两个状态 p, q , 存在一个 w 使得, $\hat{\delta}(p, w), \hat{\delta}(q, w)$ 其中一个属于 F , 而另一个不是.

3.4.1 填表算法:

1. 基础: $p \in F, q \notin F$ then p, q 不等价.
 2. 对于 $a \in \Sigma$ if $r = \delta(p, a), s = \delta(q, a)$ 不等价, 那么 p, q 不等价.
- 和那个数字逻辑里面的差不多. 但是严谨一点.
- 我们这里有四个步骤.

- 1 首先考虑基础, 初始化

- 2 其次考虑经过 0 到达终态和非终态的状态对. 就是说, 将所有状态分为三类: 1. 输入为 0 不发生状态转移的; 2. 输入为 0 到达终态的; 3. 不能到达终态的. 考虑后面两类, $\{D, F\} \times \{A, B, E, G, H\}$, 其中 \times 是一个直积.
- 3 类似的, 考虑经过 1 到达终态和非终态的状态对.
- 4 最后进行对所有空余进行验证, 卧槽, 真几把麻烦.

Remark 3.4.1. 对于状态对, 比如说 (A, E) , 考虑 $(\delta(A, 0), \delta(E, 0) = (B, H))$ 这
是说, (A, E) 不等价取决于 (B, H) 是否等价.

考虑 (B, H) , $(\delta(B, 0), \delta(H, 0)) = (G, G)$, $(\delta(B, 1), \delta(H, 1)) = (C, C)$ 终
态相等, 也是等价的标志. 那么 B, H 等价.

于是 A, E 也是等价的.

我们进行总结, 如果说有两对, 或者以上的状态对, 其等价性相互依赖, 那
么可以断言这些状态对等价. □

3.5 Regular Expressions in Unix

第四章 上下文无关文法

Contents:

- 1 无关文法
- 2 语法分析树
- 3 歧义
- 4 文法的化简和范式

4.1 上下文无关文法

例 3.1.2 告诉我们 $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ 并不是一个正则语言. 但其为一个上下文无关语言. 并且很有意思的是 $L = \{w \mid w \text{ 是正则表达式}\}$ 并不是一个正则语言, 也就是说, 正则语言, 并不能描述所有的正则语言.

Example 4.1.1 (回文). if $w \in \Sigma^*, w = w^R$, then w 为回文

Example 4.1.2 (回文语言). if $L = \{w \in \Sigma^* \mid w = w^R\}$ then L 是回文语言.

我们可以使用递归方法定义回文语言, 正如我们定义一阶逻辑之中的命题一样:

1. 首先 $\epsilon, 0, 1$ 都是回文.
2. if w 是回文, 那么 $0w0, 1w1$ 都是回文.

Definition 4.1.3 (文法). 文法 G 是一个 4p 结构, $G = (V, T, P, S)$. 其中 V for variable, T for terminator, P for Production, S for start.

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

图 4.1: A context free grammar for palindromes

- 1 V 是变量的集合.
- 2 T 称为 terminators.
- 3 P 是生产规则, 形式为 $A \rightarrow \alpha \mid \beta$, 其中 $\alpha, \beta \in (V \cup T)^*$
- 4 $S \in V$, 表示的是初始变量.

Example 4.1.4 (0, 1 组成的回文). $G = (V, T, P, S)$, 其中 $V = \{A\}$, $T = \{0, 1\}$, $P = A \rightarrow \epsilon \mid 0 \mid 1 \mid 0A0 \mid 1A1$, $S = A$

Example 4.1.5 ($0^n 1^n$). $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ 和上面一个完全类似, 这里就懒得说了

Example 4.1.6. 算术表达式, 比如说 $a + b$, $a * b + a$, 或者是有 0 或者 1 的式子, 比如说 $1 + a * 0$ 等等. 这些式子的语言就不是正则的. 在文法的构造之中用到了两个变量 E, I . 其中 E for Expression; I .

4.1.1 归约和派生

Inference is the construction of a string using productions of the grammar. It is a construction **from body to head**. Which is to say that we are given some strings that are known to be the string in the grammar, and then use them to construct another string that is in the grammar.

Meanwhile, derivation is the construction **from head to body**, from the initial variable, we use productions in the grammar to substitute the variable with strings in $(V \cup T)^*$.

inference 规约
derivation 派生
多步派生
左派生和右派生

Definition 4.1.7 (派生). $\alpha, \beta, \gamma \in (V \cup T)^*$, if $A \rightarrow \gamma$, then $\alpha A \beta \xRightarrow{G} \alpha \gamma \beta$.

Definition 4.1.8 (文法派生的句型). $\alpha \in (V \cup T)^*$, α 是变元可能派生出的语句. 比如说 $A \rightarrow \alpha$, 那么 α 就是一个句型, 而后如果 α 继续派生的得到了 β , 那么 β 也是一个句型.

Definition 4.1.9 (归约). 以算术表达式为例, 如果说 w 是 I , 那么 $aw, bw, 0w, 1w$ 也是 I , 如果说 w', w'' 是 E , 那么 $w' \cdot w'', w' + w'', (w')$ 都是 E . 这种自底向上的构造法就是归约.

Example 4.1.10 (派生的例子).

Definition 4.1.11 (多步派生).

$$\alpha_1 \xRightarrow{G}^i \alpha_{1+i} \quad (4.1)$$

对于一般的多步派生, 记为 \xRightarrow{G}^* . 特别地, 还有零步派生 $\alpha \xRightarrow{G}^* \alpha$.

Definition 4.1.12 (最左派生, 最右派生). 对于一个句型 α , 我们考虑只对最左 (右) 的变量进行派生, 这就是最左 (右) 派生. 记为 $\xRightarrow{lm} (\xRightarrow{rm})$. 为了取消歧义.

Theorem 4.1.13 (派生的等价性). 对于任意一个派生, 都存在一个最左 (右) 派生与其对应

4.1.2 文法的语言

Definition 4.1.14 (语言).

$$L = \{w \mid 0, 1 \text{ 数量相等} \}$$

$$L = \{a^n, b^n \mid n \in \mathbb{N}\}$$

$$L(G) = \{w \mid w \in T^*, S \xRightarrow{G}^* w\} \quad (4.2)$$

Remark 4.1.15 (为什么称为是上下文无关文法). 我们参考 $\alpha A \beta \xRightarrow{G} \alpha \gamma \beta$, 这个派生是否成立, 是跟 α, β 无关的, 于是称为上下文无关文法.

Definition 4.1.16 (等价性). G_1, G_2 满足 $L(G_1) = L(G_2)$, 则 G_1 等价于 G_2

Definition 4.1.17 (句型, 左句型, 右句型). G 生成的句型定义如下:

$$\{w \mid w \in (V \cup T)^*, S \xRightarrow{G}^* w\} \quad (4.3)$$

也定义了左句型, 右句型.

Example 4.1.18. $L = \{a^n b^n \mid n \geq 1\}$

Example 4.1.19. $G = \{\}$

Example 4.1.20. $L = \{w \mid 0, 1 \text{ 数量相等}\}$, $G = (V, T, P, S)$, 其中 $V = \{S\}$, $T = \{0, 1\}$, P 是下面这个:

$$S \rightarrow 0S1S \mid 1S0S \mid \epsilon$$

如何证明其对应的语言确实是 L ? 我们需要验证, 对于 L 之中的任意一个字符串 w 确实能够如此划分, 使得 $w = 0w'1w''$, 其中 w', w'' 都是 L 的. 使用分类讨论即可证明这一点.

Example 4.1.21 (算术表达式).

4.2 语法分析树

sparse tree 分析树

Pump Lemma 泵引理

Sparse Tree 和 inference 和

derivation 之间的等价性

语法分析树是表示派生过程的一个树. 此后在 Context free language 的泵引理的证明之中要用到.

Definition 4.2.1 (分析树). 定义为

- 1 每个内节点是边缘符号
- 2 叶子节点是 $V \cup T \cup \{\epsilon\}$ 之中的符号
- 3 如果说内节点的标记是 A , 那么其子节点从左到右分别为

$$X_1, X_2, \dots, X_n$$

那么 $A \rightarrow X_1 X_2 \dots X_n$ 是一个产生式. □

如果说, 根节点是初始符号 s , 叶子节点是终结符, 那么该树是完成的, 该树的产物属于 $L(G)$.

Definition 4.2.2 (subtree). 在树 T 之中以内节点 A 为根节点的 subtree 为 T 的 subtree

Theorem 4.2.3 (the equivalence of tree). 一棵树和 $L(G)$ 之中的一个语句等价.

1. G 存在 A 为根节点的树, 其产物为 α , then $A \xRightarrow{*} \alpha$
2. 如果说 $A \xRightarrow{*} \alpha$ then 存在 A 为根的树, 其产物为 α

证明. □

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

图 4.2: A context free grammar for palindromes

4.3 文法和语言的歧义性

歧义性

算术表达式的语言中 L_{exp} , 比如说 $w = 1 * 1 + 1$, 有两个生成树. 此为歧义.

歧义性和算法

Example 4.3.1 (表示加法乘法的优先级). Fig 4.2 展示了 G_{exp} 的产生式, G_{exp} 是歧义的, 因为存在一个语句有两个 Sparse Tree.

Definition 4.3.2 (固有歧义). 对于一些语言, 其对应的文法都是有歧义的.

Example 4.3.3. 对于 $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$, L 是固有歧义的.

需要知道的是, 固有歧义证明非常繁琐, 甚至很难理解. 并且, 并不存在算法能够验证任意一个语言是固有歧义的.

4.4 文法的化简和设计

4.4.1 化简

Useless

1 消除无用符号

 ϵ -productions2 消除 ϵ 产生式 (ϵ production)

unit productions

3 消除单元产生式 (unit production) : $A \rightarrow B$

无用符号的消除

我们将一些无用的符号进行化简, 分为两类, 一类是不可达的, 一类是不可产生的, 这两种符号都是没有用的, 这是说, 对于一个语句的派生过程, 这些符号是不会出现的. 所以说是无用符号. 下面给出定义.

Definition 4.4.1 (Useful). 对于 $G = (V, T, P, S)$, 符号 $X \in (V \cup T)$.

1. 如果 $G \Rightarrow^* \alpha X \beta$ then X is **reachable**.
2. if $\alpha X \beta \Rightarrow^* w \in T^*$ then X is **generating**.
3. If X is reachable and generating, then X is **useful**. X is useless otherwise.

Remark 4.4.2. T is generating, while S is reachable. You can easily prove it.

Example 4.4.3. 对于 $P = \{S \rightarrow AB \mid a, A \rightarrow b\}$ 可达的符号机和产生的符号分别为什么?

证明. $T = \{a, b\}$, A, S 是产生的, S, A, B, a, b 是可达的, □

Remark 4.4.4. 能够看出, 我们要找到产生的, 则需要从后往前找, 若是要找到可达的, 则需要从前往后找.

ϵ 产生式

如果说 $A \Rightarrow \epsilon$, 称 A 是可空的. ϵ 产生式除了贡献 L 中的 ϵ 之外没有任何作用, 于是说, 当 L 之中没有 ϵ 的时候, 我们就可以将 G 之中的 ϵ 产生式消除掉. 这 (大概) 就是原理.

Definition 4.4.5 (可空变元).

Example 4.4.6. 消除 $G = (\{S, A, B\}, \{a, b\}, P, S)$ 的 ϵ 产生式.

$$\begin{aligned}
 P = \{ & S \rightarrow AB, \\
 & A \rightarrow AaA \mid \epsilon \\
 & B \rightarrow BbB \mid \epsilon \}
 \end{aligned} \tag{4.4}$$

Example 4.4.7. $G_2 = (\{S, A, B, C\}, \{a, b\}, P, S)$

$$\begin{aligned}
 P = \{ & S \rightarrow ABC, \\
 & A \rightarrow aA \mid \epsilon, \\
 & B \rightarrow bB \mid \epsilon, \\
 & C \rightarrow \epsilon \}
 \end{aligned} \tag{4.5}$$

单元产生式的消除

If $A \xRightarrow{*} B$ then A, B are equivalent. You may view A, B as the same variable and merge the relative productions.

Example 4.4.8.

$$\begin{aligned} P = \{ & S \rightarrow A \mid B \mid 0S1, \\ & A \rightarrow 0A \mid 0, \\ & B \rightarrow 1B \mid 1 \} \end{aligned} \quad (4.6)$$

证明. It is easy to note that S, A, B are equivalent. Then $P = \{S \rightarrow 0S1 \mid 0S \mid 0 \mid 1S \mid 1\}$ \square

化简步骤

- 1 消除 ϵ 产生式
- 2 reduce the unit productions
- 3 Reduce the not generating productions
- 4 Reduce the not reachable productions

第五章 两种范式和下推自动机的定义

5.1 两种范式

范式有两种范式, 一种是乔姆斯基范式 (CNF), 另一种是格雷巴赫范式 (GNF).
CNF GNF

Definition 5.1.1 (CNF). 每一个不包含 ϵ 的 G 的产生式都能够写为下面的形式:

$$A \rightarrow BC \quad A \rightarrow a$$

其中 A, B, C 是变量, 然后 a 是终结符.

证明. 能够知道这个定理实际上是显然的. 步骤为:

1. 将产生式的 terminate 消除
2. 让产生式一次只多一个变量.

具体流程为, 给定一个产生式

$$A \rightarrow X_1 X_2 X_3 \dots X_m$$

其中 $X_i \in (V \cup T)$. 我们对其中的终结符进行这样的处理: X_i 换为 V_i , V_i 是新引入的变量. 且有产生式 $V_i \rightarrow X_i$. 这就将 $A \rightarrow X_1 \dots X_m$ 的终结符处理了.

我们将表达式 $A \rightarrow X_i \dots$ 写为 $A \rightarrow V_1 \dots V_m$, V_i 是变量. 我们再次引入变量 B_i , 将 $A \rightarrow V_1 \dots V_m$, 拆解为多个产生式, 就是 $A \rightarrow V_1 B_1$, $B_1 \rightarrow V_2 B_2 \dots$ 于是, $A \rightarrow X_1 X_2 X_3 \dots X_m$ 就被被拆分为了多个产生式, 并且他们是属于乔姆斯基范式的. \square

Example 5.1.2. 给定 CFG¹ G , P 为

$$S \rightarrow bA \mid aB, \quad A \rightarrow bAA \mid aS \mid a, \quad B \rightarrow aBB \mid bS \mid b$$

流程为

1. 对于产生式之中有终结符的, 进行变量的替换, 比如说 $A \rightarrow bA$ 我们引入新的变量 C_b , 将这个式子换为

$$A \rightarrow C_b A, \quad C_b \rightarrow b$$

2. 对于形如 $A \rightarrow B_1 B_2 \dots B_m$ 的式子, 引入中间变量.

Example 5.1.3. 将 G 的产生式进行处理, 使得其为乔姆斯基范式.

$$S \rightarrow AS \mid BABC \quad A \rightarrow A1 \mid 0A1 \mid 01 \quad B \rightarrow 0B \mid 0 \quad C \rightarrow 1C \mid 1$$

Definition 5.1.4 (格雷巴赫范式). 每个产生式的形式为 $A \rightarrow a\alpha$ 其中 a 是一个终结符, 然后 α 是变元的串, 也就是 $\alpha \in V^*$. 简写为 GNF.

Example 5.1.5. 将 $S \rightarrow AB, A \rightarrow AaA \mid bB \mid b, B \rightarrow b$ 转化为 GNF.

虽然说, CNF 的转化是不要求掌握的.

就尼玛讲完了!

5.2 下推自动机

Contents:

1. 接受的语言
2. 其和文法的等价性
3. 确定型下推自动机.

¹CFG stands for context free grammar. 只是有点忘了

5.2.1 下推自动机的定义

Definition 5.2.1 (下推自动机). 下推自动机 (PDA)². 一个 PDA G 是一个七元组. PDA 相当于一个 NFA 加上一个栈. 进行每一次状态转移的时候, 会将栈顶元素弹出, 然后压入某些字符串. 初始阶段, 栈里面有一个初始元素 Z_0 . 比如说我们接受了 0, 而后弹出 Z_0 , 压入 $0Z_0$. 对于一个字符串, 压入顺序是从后往前的.

Example 5.2.2. 设计识别 $L_{01} = \{0^n 1^n \mid n \geq 1\}$ 的 PDA

Example 5.2.3. 设计识别 $L_{ww^R} = \{ww^R \mid w \in (0+1)^*\}$

²PDA for push down automata