目录

1 正则表达式的定义

正则表达式 的递归定义.

定义 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_1E_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.

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.

定义 1.2 (运算)**.** 下面列出四种运算.

$$
\begin{aligned}
&L \cdot M \equiv \{w \mid w = xy, x \in L, y \in M\} \\
&L + M \equiv L \cup M \\
&L^2 = L \cdot L \text{ 特别地}, L^0 = \{\epsilon\} \\
&L^* = \bigcup_{i=0}^{\infty} L^i
\end{aligned}
$$

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 1.3. *下面正则表达式的语言是什么?*

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

Example 1.4.

$$(a + b)^*(a + bb) \tag{2}$$

a, b 组成的, 以 a 或者 bb 结尾的字符串.

2 正则表达式的性质

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

$$R_{i,j}^{(k)} = R_{i,j}^{(k-1)} + R_{i,k}^{(k-1)} (R_{k,k}^{(k-1)})^* R_{k,j}^{(k-1)} \tag{3}$$

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

$$\sum_{j \in F} R_{1,j}^{(n)} \tag{4}$$

Example 2.1. *见 ppt , 流程还是比较麻烦的.*

[1]路径实际上是一个字符串!

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.

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. 将 $0(0+1)^*$ 转化为 $\epsilon - NFA$.