形 式 语 言 与 自 动 机 理 论

课 程 讲 义

王春宇

计 算 机 科 学 与 技 术 学 院

软 件 基 础 教 研 室

哈 尔 滨 工 业 大 学

目录

Chapter 1

课程简介与基础知识

1.1 课程简介

作为计算机科学的核心, 计算理论是关于计算知识的有系统的整体, 本是数学的一个研究领域, 诞生于数理逻辑学家对计算本质的探索. 这里的计算 (*Computation*) 并不是指纯粹的算术 (*Calculation*), 而是指一种以 "机械而有效的" 方式获取问题答案的过程.

随着计算理论的发展和相关技术的进步, 最终促使了计算机的发明. 此后, 计算理论的重心也逐渐从数学转移到了计算机科学. 计算理论, 乃至计算机科学, 所关心的核心问题是:

计算机的基本能力和限制究竟是什么?

这个问题中包含了两个内容, 分别对应计算理论的两个研究方向: 可计算性理论和计算复杂性理论, 而形式语言与自动机理论正是这两个重要的研究方向的理论基础.

(1) 可计算性理论的核心问题是: 究竟哪些问题, 可以通过计算解决?

计算作为一种能力, 是否有边界? 是不是任何问题都可以通过计算来解决? 如果是, 它会是什么样的? 如果不是, 有哪些问题可以, 而哪些问题不可以? 为什么不可以?

为了能够严谨的研究这种机械而有效的计算过程, 我们需要严格定义的概念去描述它, 需要严谨的计算模型去分析它. 这些模型呢, 就是我们将要学习的自动机理论; 而这个概念, 其实就是已经被我们大家所熟知的—算法 (*Algorithm*).

所谓算法, 可以追溯到公元前 3 世纪, 那时的人们, 就已经有了算法的直觉概念, 并且试图寻找解决各种数学问题的算法, 其中最著名的就是求两整数最大公约数的欧几里得算法, 也称为辗转相除法. 如果一个问题, 有了具体的算法以后, 解决起来就不再需要太多的人的智慧, 只要按照算法的步骤机械的计算, 就可以得到答案. 比如, 寻找两个整数的最大公约数, 使用欧几里得算法, 一个小学生和一个数学家可以做的一样好.

数学家们也一直在寻找更为通用的算法, 试图解决更基本的问题. 20 世纪初, 数学家希尔

伯特曾经很乐观的试图寻找“整数上一阶谓词演算中判断任何公式是否为真”的算法. 如果这样的算法存在, 那么, 数学家就可以一劳永逸地解决几乎所有的数学命题. 但是, 由于当时对计算和算法本质的认识还不够深刻, 还不能使用数学工具去分析计算的能力. 所以这样的企图失败了, 但在这个过程当中, 积累了有关算法的重要知识.

到 20 世纪 30 年代, 数理逻辑学家在研究可计算的整数函数时, 通过图灵机和 λ 演算等计算模型, 首次将算法的概念形式化. 从那以后, 人们才可以利用数理逻辑方法研究计算的本质, 并且发现计算也不是万能的, 确实存在一些问题是不可能通过计算解决的, 或者说, 这些问题是不存在算法的. 我们在课程的最后部分中, 会给出这样的问题和相关的证明.

(2) 计算复杂性理论的核心问题是: *解决可计算的问题，究竟需要多少资源?*

也就是计算一个问题时, 需要消耗的时间和占用的存储空间, 会达到什么程度? 如果一个问题, 无论使用什么算法, 求解过程都需要相当多的资源, 那么其中的原因是什么? 究竟是什么, 造成了一些问题很难计算, 而另一些问题却很容易?

虽然, 其中的原因仍然是未知的, 但在分析各种有效计算模型的过程中, 人们发现了一个按照难度给问题分类的完美体系. 如同元素周期表对化学元素性质的分类. 有了这个体系, 我们就可以将未知的问题分类, 根据难易程度选择用什么样的对策去解决. 目前, 计算复杂性理论仍然是计算机科学领域重要的研究热点, 但超出了形式语言与自动机课程的内容, 我们不会涉及太多, 但是这些内容却是通往复杂性理论研究的必经之路.

因为, 为了可计算性和计算复杂性的研究, 需要使用和构造什么样的计算模型, 就是形式语言与自动机理论的主要内容. 这些模型都是高度抽象化的计算装置, 简单明确但功能强大, 不但便于在理论分析中进行推导和证明, 在很多实际问题中也有非常直接的应用.

计算理论在一系列不同的领域中都有先驱者. 研究神经细胞网络模型的生物学家, 发展开关理论用于硬件设计的机电工程师, 从事数理逻辑研究的数学家, 研究自然语言文法的语言学家等. 这些研究中逐渐发展起来的一些模型, 对于理论计算机科学的研究是至关重要的.

什么是自动机理论?

自动机理论: 研究抽象机器及其所能解决问题的理论.

- 图灵机
- 有限状态机
- 文法, 下推自动机

其中最重要的就是图灵机, 它具有现如今各种计算机的所有能力, 是计算机的理论模型, 它区分了哪些问题是可计算的哪些是不可计算的. 而其他一些稍简单的模型, 比如有穷自动机, 在数字电路, 通讯协议和游戏 AI 的设计等实际问题上有重要的应用; 下推自动机和文法在计算机程序设计语言的设计和编译器实现上发挥了重要的作用.

什么是形式语言?

形式语言: 经数学定义的语言.

		自然语言		形式语言	
		English	中文	化学分子式	C 语言
语言	字符	A,a,B,b,...	天, 地,...	A-Z,a-z,0-9...	`A-Z,a-z,0-9...`
	单词	apple	苹果	H_2O	`char`
	句子	How're you?	早上好!	$2H_2+O_2=2H_2O$	`char a=10;`
	语法	Grammar	语法规则	精确定义的规则	

如果自动机是研究计算的模型, 那么语言就是研究计算的问题或实例. 而所谓*形式语言*, 就是指经数学定义的语言. 我们要以数学的方法从解决问题的角度研究计算, 那么我们首先需要以数学的方法来描述问题, 这种描述就是形式语言. 使用语言这个概念, 似乎有些奇怪, 但和我们的常识其实是一致的. 我们可以以这样的观点, 理解语言的构成, 由字符, 单词, 句子, 和语法构成了语言. 比如自然语言中的英文, 中文等. 只要有严谨的数学定义, 就可以称为形式语言, 比如化学分子式, 程序设计语言等. 定义一个语言, 我们首先要确定基本字符有哪些, 再确定如何构成单词和句子的基本规则, 单词和句子在形式语言中我们都认为是字符串, 最关键的是如何描述这个基本规则, 在形式语言与自动机理论中, 这种描述实际上就是自动机. 所以形式语言与自动机是密不可分的, 一方面自动机以语言为处理对象, 另一方面语言是以自动机为形式定义的.

参考书

- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* 《自动机理论、语言和计算导论》机械工业出版社
- Michael Sipser. *Introduction to the Theory of Computation.* 《计算理论导引》机械工业出版社

1.2 基础知识

1.2.1 基本概念

1. *字母表*: 符号 (或字符) 的非空有穷集.

$$\Sigma_1 = \{0, 1\},$$

$$\Sigma_2 = \{a, b, \ldots, z\},$$
$$\Sigma_3 = \{x \mid x \text{ 是一个汉字}\}.$$

这里的“符号”是一个抽象的实体, 我们不再去形式的定义它, 如同几何学中对“点”和“线”的概念不加定义一样. 字母和数字是经常使用的一些符号的例子.

2. 字符串: 由某字母表中符号组成的有穷序列.

   若 $\Sigma_1 = \{0, 1\}$, 那么 $0, 1, 00, 111001$ 为 Σ_1 上的字符串;

   若 $\Sigma_2 = \{a, b, \ldots, z\}$, 那么 $ab, xkcd$ 为 Σ_2 上的字符串.

3. 空串: 记为 ε, 有 0 个字符的串.

   字母表 Σ 可以是任意的, 但都有 $\varepsilon \notin \Sigma$.

4. 字符串的长度: 字符串中符号所占位置的个数, 记为 $|\sqcup|$.

   若字母表为 Σ, 可递归定义为:

$$|w| = \begin{cases} 0 & w = \varepsilon \\ |x| + 1 & w = xa \end{cases},$$

   其中 $a \in \Sigma$, w 和 x 是 Σ 中字符组成的字符串.

$\star$. 符号使用的一般约定:

- 字母表: $\Sigma, \Gamma, \Delta, \ldots$
- 字符: $a, b, c, \ldots$
- 字符串: $\ldots, w, x, y, z$
- 集合: $A, B, C, \ldots$

5. 字符串 x 和 y 的连接: 将首尾相接得到新字符串的运算, 记为 $x \cdot y$ 或 xy.

   同样, 可递归定义为

$$x \cdot y = \begin{cases} x & y = \varepsilon \\ (x \cdot z)a & y = za \end{cases},$$

   其中 $a \in \Sigma$, 且 x, y, z 都是字符串.

   对任何字符串 x, 有 $\varepsilon \cdot x = x \cdot \varepsilon = x$.

   连接运算的符号“$\cdot$”一般省略.

6. 字符串 x 的 n 次幂$(n \geq 0)$, 递归定义为

$$x^n = \begin{cases} \varepsilon & n = 0 \\ x^{n-1}x & n > 0 \end{cases}.$$

例如,

$$\begin{aligned} (ba)^2 &= (ba)^1 ba & ba^2 &= ba^1 a \\ &= (ba)^0 baba & &= ba^0 aa \\ &= \varepsilon baba & &= b\varepsilon aa \\ &= baba & &= baa \end{aligned}$$

7. 集合 A 和 B 的连接, 记为 $A{\cdot}B$ 或 AB, 定义为

$$A{\cdot}B = \{w \mid w = x \cdot y,\ x \in A \text{且} y \in B\}.$$

8. 集合 A 的 n 次幂$(n \geq 0)$, 递归定义为

$$A^n = \begin{cases} \{\varepsilon\} & n = 0 \\ A^{n-1}A & n \geq 1 \end{cases}.$$

那么, 若 Σ 为字母表, 则 Σ^n 为 Σ 上长度为 n 的字符串集合. 如果 $\Sigma = \{0, 1\}$, 有

$$\begin{aligned} \Sigma^0 &= \{\varepsilon\} \\ \Sigma^1 &= \{0, 1\} \\ \Sigma^2 &= \{00, 01, 10, 11\} \\ \Sigma^3 &= \{000, 001, 010, 011, 100, 101, 110, 111\} \\ &\vdots \end{aligned}$$

一般来说, 我们不需要明确区分长度为 1 的字符与字符串, 简单的认为 $\Sigma = \Sigma^1$.

9. 克林闭包 (*Kleene Closure*):

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i.$$

10. 正闭包 (*Positive Closure*):

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i.$$

显然,

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}.$$

其他的概念如有向图, 树, 字符串的前缀, 后缀等定义这里省略.

1.2.2 语言和问题

定义. *若 Σ 为字母表且 $\forall L \subseteq \Sigma^*$, 则 L 称为字母表 Σ 上的语言.*

- 自然语言, 程序设计语言等
- $\{0^n 1^n \mid n \geq 0\}$
- The set of strings of 0's and 1's with an equal number of each:

$$\{\varepsilon, 01, 10, 0011, 0101, 1100, \ldots\}$$

- $\varnothing$, $\{\varepsilon\}$ 和 Σ^* 分别都是任意字母表 Σ 上的语言, 但注意 $\varnothing \neq \{\varepsilon\}$

关于语言

唯一重要的约束就是所有字母表都是有穷的.

自动机理论中的典型问题

判断给定的字符串 w 是否属于某个具体的语言 L,

$$w \in L?$$

- 任何所谓问题, 都可以转为语言成员性的问题
- 语言和问题其实是相同的东西

"$w \in L$?" 也称为语言的成员性问题, 这样的问题是具有广泛性的, 各种实际问题都可以通过编码等方式转换成这种问题. 语言和问题其实是相同的东西. 我们想用这种特定的语言问题, 去探索类似的一般性问题的通用解法.

我们可以让 w 是参数, 比如是一个数字, 让 L 是一个具体的语言 (集合), 比如是所有的素数, 那这个问题就是判断 "某数字是否是素数?" 我们关心的是, 是否能用某种计算模型来回答它, 无论答案是肯定的还是否定的, 都能明确回答. 显然这个判断数字是否为素数的问题是可以被回答的, 因为我们可以逐个检查比它小的数是否是它的因子. 无论答案是肯定或否定, 都能明确回答的, 这样的问题, 我们称为是可判定的问题.

如果我们让 L 以一种特定的规则来描述, 比如, 让 w 是某个程序设计语言 (比如 C) 的程序源码, 而 L 以该语言的语法规则描述, 那么这个问题就是 "程序 w 能否被正确的编译?" 这就是程序设计语言和编译器设计首先要考虑的问题. 是否容易回答依赖于语法规则的设计是否合理. 我们将会看到, 只有当满足特定条件的语法规则时, 这个问题才容易实现, 如果语法

规则过于自由, 是无法设计编译器的. 正是因为计算理论在这些方面的研究成果, 才使得现在程序设计语言与编译器的设计变得容易.

如果我们再放宽限制, 将 w 和 L 都看做是可变的参数, 那么解决这样问题的方法, 如果存在, 将会非常有用. 同样, 我们会看到, 有些这样的问题, 是干脆无法回答的; 而有些, 对肯定的答案可以明确回答, 但对否定的答案则不然, 其中最著名的就是图灵停机问题, 这些都是不可判定的问题, 在课程的最后两章我们会详细的介绍.

1.2.3 形式化证明

形式化证明: 演绎法, 归纳法和反证法

例 1. 若 x 和 y 是 Σ 上的字符串, 请证明 $|xy| = |x| + |y|$.

证明: 通过对 $|y|$ 的归纳来证明

1. 基础: 当 $|y| = 0$, 即 $y = \varepsilon$

$$\begin{aligned} |x\varepsilon| &= |x| && \text{连接的定义} \\ &= |x| + |\varepsilon| && \text{长度的定义} \end{aligned}$$

2. 递推: 假设 $|y| = n$ $(n \geq 0)$ 时命题成立, 那么当 $|y| = n + 1$, 即 $y = wa$

$$\begin{aligned} |x(wa)| &= |(xw)a| && \text{连接的定义} \\ &= |xw| + 1 && \text{长度的定义} \\ &= |x| + |w| + 1 && \text{归纳假设} \\ &= |x| + |wa| && \text{长度的定义} \end{aligned}$$

□

证明: 通过对 y 的结构归纳来证明

1. 基础: $y = \varepsilon$ 时

$$\begin{aligned} |x\varepsilon| &= |x| && \text{连接的定义} \\ &= |x| + |\varepsilon| && \text{长度的定义} \end{aligned}$$

2. 递推: 假设 $y = w$ $(w \in \Sigma^*)$ 时命题成立, 那么当 $y = wa$ 时

$$\begin{aligned} |x(wa)| &= |(xw)a| && \text{连接的定义} \\ &= |xw| + 1 && \text{长度的定义} \\ &= |x| + |w| + 1 && \text{归纳假设} \\ &= |x| + |wa| && \text{长度的定义} \end{aligned}$$

□

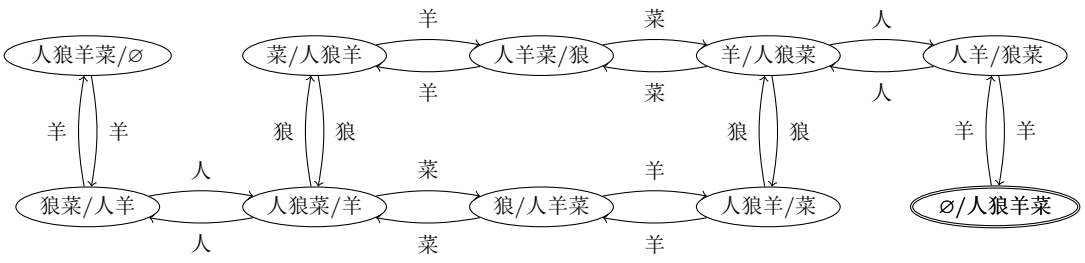
Chapter 2

有穷自动机

2.1 有穷状态系统

有穷状态系统是具有离散输入和输出系统的一种模型. 系统内可以处于任一有穷个内部的格局或称“状态”. 系统的状态概括了关于过去输入的某些信息, 并为确定系统以后的行为所必须. 有穷自动机, 也称为有限状态机, 是有穷状态系统的抽象模型. 有穷自动机是关于存储量极其有限的计算机的很好的模型, 一台计算机用如此小的存储能做什么呢? 回答是: 能做很多有用的事情! 在实际应用中使用最多的两种有穷自动机的变形是: Moore 机和 Mealy 机, 它们的应用, 在我们日常生活中可以说到处都是, 电灯开关, 电梯控制, 自动售货机, 自动取款机等等. 有限状态机的应用领域非常广泛, 比如数字电路的设计, 电脑游戏的 AI 设计, 几乎所有的通讯协议, 比如 TCP, HTTP, 蓝牙, Wifi, 甚至整个电信行业的通讯协议等等. 在计算机应用中也非常多, 比如文本搜索, 词法分析等等. 而我们学习的有穷自动机, 是作为语言的一种识别装置.

例. 狼, 羊, 菜, 人的过河问题. 一个人带着狼、山羊、白菜在一条河的左岸, 有一条船, 大小正好能装下这个人和其它三者之一. 每次只能带一件东西过河, 剩下的两件, 如果没人照顾, 狼会吃羊, 羊会吃菜. 问是否有可能安全过河, 使得羊和白菜都不会被吃掉?

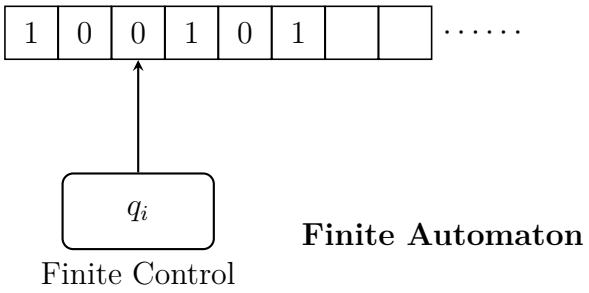


2.2 确定的有穷自动机

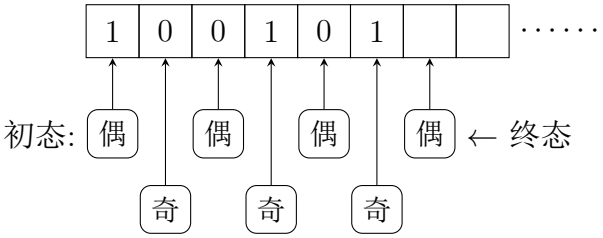
2.2.1 形式定义

确定的有穷自动机

- 一条输入带
- 一个读头
- 一个有穷控制器



例 1. 用有穷自动机识别 $\{w \in \{0,1\}^* \mid w$ 的长度 $|w|$ 是偶数.$\}$



有穷自动机可以看做是这样的一个抽象装置, 它具有一条输入带, 一个读头和一个有穷控制器 (*Finite Control*). 输入带上划分为单元格, 每个格子可以放置一个字符, 那么整条带子就可以用来放置一个被扫描的字符串; 因为字符串的长度总是有限的, 在字符串之外的格子上, 总是空白的; 读头可以读入带子上单元格中的字符, 并可以从左向右移动, 每次移动一个单元格; 有穷控制器可以存储有限个状态, 并且可以根据读头读入的字符和当前的状态进行状态改变.

有穷自动机, 在扫描输入的字符串之前, 读头在输入串的第一个字符下面, 然后从左向右一次一个单元格的读入字符, 移动读头, 并修改状态, 然后自动的循环这个过程, 直到扫描完整个字符串之后, 通过有穷控制器中的当前状态, 对这个字符串进行判断, 回答有两种: 接受或拒绝.

我们在需要表示这样几个方面信息: 有穷控制器中的状态, 输入带上的符号, 状态改变的规则, 第一个状态, 哪些状态可以被接受. 将这样的抽象装置再抽象为数学语言, 也就是它的形式定义.

确定的有穷自动机的形式定义

定义. *确定的有穷自动机 (DFA, Deterministic Finite Automaton) A 为五元组*

$$A = (Q, \Sigma, \delta, q_0, F)$$

1. Q : 有穷状态集;
2. Σ : 有穷输入符号集或字母表;
3. $\delta : Q \times \Sigma \to Q$, 状态转移函数;
4. $q_0 \in Q$: 初始状态;
5. $F \subseteq Q$: 终结状态集或接受状态集.

开始时, 输入串在输入带上, 读头在第一个字符, 有穷控制器初始处于 q_0. 自动机的读头每次读入一个字符, 根据转移函数修改当前状态, 并向后移动一个单元格. 若输入串全部读入后, 处于接受状态, 那么自动机接受这个输入串, 否则拒绝该串.

不论是自动机还是艺术品, 设计都是一个创作过程, 因此不可能把它归结为一个简单处方或公式. 但是在设计时, 将自己放在自动机的位置去思考, 总是有帮助的. 假定你自己就是这台有穷自动机, 去判断一个输入字符串是不是应该被接受. 你一个接一个的读这个字符串的符号, 并不知道什么时候会结束, 需要随时准备好答案. 为了能够作出判断, 必须估算出当读入一个字符时需要记住的那些东西, 而不是读入的所有的内容. 因为你是一台有穷自动机, 你的存储是有限的. 幸运的是, 对许多语言你不需要记住整个输入.

例 2. 请设计 DFA, 在任何由 0 和 1 构成的串中, 接受含有 01 子串的全部串.

1. 未发现 01, 即使 0 都还没出现过;
2. 未发现 01, 但刚刚读入字符是 0;
3. 已经发现了 01.

因此 DFA A 的可定义为:

$$A = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta, q_1, \{q_3\})$$

其中 δ 为:

$$\begin{array}{lll} \delta(q_1, 1) = q_1 & \delta(q_2, 1) = q_3 & \delta(q_3, 1) = q_3 \\ \delta(q_1, 0) = q_2 & \delta(q_2, 0) = q_2 & \delta(q_3, 0) = q_3 \end{array}$$

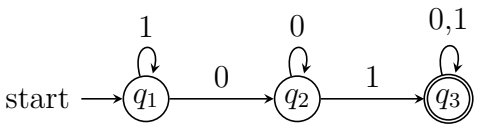
2.2.2 DFA 的表示

DFA 除了使用其形式定义的五元组表示, 也可以有两种简化的表示方法, 分别为状态转移图 (*transition diagram*) 和状态转移表 (*transition table*).

状态转移图

1. 每个状态 q 对应一个节点, 用圆圈表示;
2. 状态转移 $\delta(q,a)=p$ 为一条从 q 到 p 且标记为字符 a 的有向边;
3. 开始状态 q_0 用一个标有 start 的箭头表示;
4. 接受状态的节点, 用双圆圈表示.

续例 2. 含有 01 子串的全部串的状态转移图



状态转移表

1. 每个状态 q 对应一行, 每个字符 a 对应一列;
2. 若有 $\delta(q,a)=p$, 用第 q 行第 a 列中填入的 p 表示;
3. 开始状态 q_0 前, 标记箭头 $\rightarrow$ 表示;
4. 接受状态 $q\in F$ 前, 标记星号 $*$ 表示.

续例 2. 含有 01 子串的全部串的状态转移表

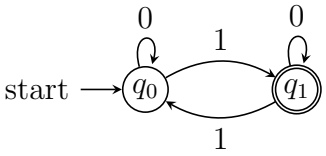
	0	1
$\rightarrow q_1$	q_2	q_1
q_2	q_2	q_3
$*q_3$	q_3	q_3

2.2.3 DFA 的设计举例

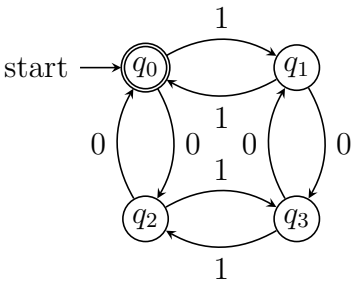
典型问题

设计 DFA 使其接受且仅接受给定的语言 L.

例 3. 若 $\Sigma=\{0,1\}$, 给出接受全部含有奇数个 1 的串 DFA.



例 4. 若 $\Sigma = \{0, 1\}$, 给出接受全部含有偶数个 0 和偶数个 1 的串 DFA.



思考题

若 $\Sigma = \{0, 1\}$

1. 如何设计接受 $\varnothing$ 的 DFA?
2. 如何设计接受 Σ^* 的 DFA?
3. 如何设计接受 $\{\varepsilon\}$ 的 DFA?

2.2.4 扩展转移函数

转移函数 δ 是 $Q \times \Sigma$ 上的函数, 所以只能处理 Σ 中的字符, 为了使用方便, 定义字符串上的转移函数 $\hat{\delta}$.

定义. *扩展 δ 到字符串, 定义扩展转移函数 $\hat{\delta} : Q \times \Sigma^* \to Q$ 为*

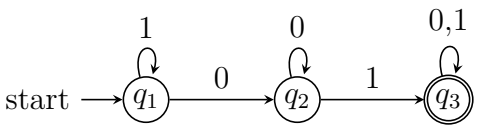
$$\hat{\delta}(q, w) = \begin{cases} q & w = \varepsilon \\ \delta(\hat{\delta}(q, x), a) & w = xa \end{cases}$$

其中 $a \in \Sigma$, $w, x \in \Sigma^*$.

那么, 当 $w = a_0 a_1 \cdots a_n$, 则有

$$\begin{aligned} \hat{\delta}(q, w) &= \delta(\hat{\delta}(q, a_0 a_1 \cdots a_{n-1}), a_n) \\ &= \delta(\delta(\hat{\delta}(q, a_0 a_1 \cdots a_{n-2}), a_{n-1}), a_n) = \cdots \\ &= \delta(\delta(\cdots \delta(\hat{\delta}(q, \varepsilon), a_0) \cdots, a_{n-1}), a_n) \end{aligned}$$

续例 2. 接受全部含有 01 子串的 DFA, $\hat{\delta}$ 处理串 0101 的过程.



$$\begin{aligned}
\hat{\delta}(q_0, 0101) &= \delta(\hat{\delta}(q_0, 010), 1) \\
&= \delta(\delta(\hat{\delta}(q_0, 01), 0), 1) \\
&= \delta(\delta(\delta(\hat{\delta}(q_0, 0), 1), 0), 1) \\
&= \delta(\delta(\delta(\delta(\hat{\delta}(q_0, \varepsilon), 0), 1), 0), 1) \\
&= \delta(\delta(\delta(\delta(q_0, 0), 1), 0), 1) \\
&= \delta(\delta(\delta(q_1, 1), 0), 1) \\
&= \delta(\delta(q_2, 0), 1) = \delta(q_2, 1) = q_2
\end{aligned}$$

思考题

1. 扩展转移函数 $\hat{\delta}$ 必须从开始状态 q_0 处理字符串吗?
2. 对任意的串 w, $\hat{\delta}$ 能保证一定会跳转到某个状态吗?

例 5. 对任何状态 q 及字符串 x 和 y, 证明 $\hat{\delta}(q, xy) = \hat{\delta}(\hat{\delta}(q, x), y)$.

证明: 对 y 使用归纳法.

1. 当 $y = \varepsilon$ 时

$$\begin{aligned}
\hat{\delta}(\hat{\delta}(q, x), \varepsilon) &= \hat{\delta}(q, x) && \hat{\delta}\text{的定义} \\
&= \hat{\delta}(q, x\varepsilon)
\end{aligned}$$

2. 假设 $y = w$ ($w \in \Sigma^*$) 时命题成立, 当 $y = wa$ ($a \in \Sigma$) 时

$$\begin{aligned}
\hat{\delta}(q, xwa) &= \delta(\hat{\delta}(q, xw), a) && \hat{\delta}\text{和连接的定义} \\
&= \delta(\hat{\delta}(\hat{\delta}(q, x), w), a) && \text{归纳假设} \\
&= \hat{\delta}(\hat{\delta}(q, x), wa) && \hat{\delta}\text{的定义}
\end{aligned}$$

□

2.2.5 DFA 的语言与正则语言

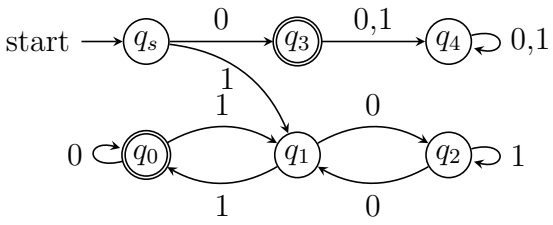
定义. *若 $D = (Q, \Sigma, \delta, q_0, F)$ 是一个 DFA, 则 D 接受的语言为*

$$\mathbf{L}(D) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}.$$

定义. *如果语言 L 是某个 DFA D 的语言, 即 $L = \mathbf{L}(D)$, 则称 L 是正则语言.*

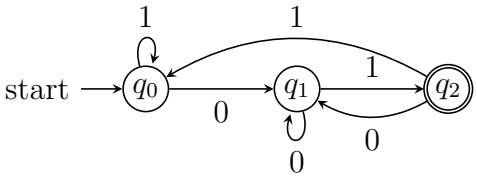
- $\varnothing$, $\{\varepsilon\}$ 都是正则语言
- 若 Σ 是字母表, Σ^*, Σ^n 都是 Σ 上的正则语言

例 6. 设计 DFA 接受 $\{0, 1\}$ 上的字符串 w, 且 w 是 3 的倍数的二进制表示.

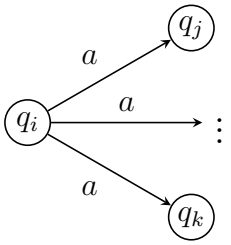


2.3 非确定有穷自动机

例 7. 由 0 和 1 构成的串中, 接受全部以 01 结尾的串, 如何设计 DFA?



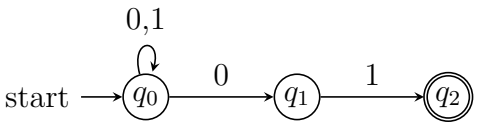
状态的非确定转移



- 同一个状态在相同的输入下, 可以有多个转移状态
- 自动机可以处在多个当前状态

- 使自动机的设计更容易

续例 7. 由 0 和 1 构成的串中, 接受全部以 01 结尾的串.



思考题

有穷自动机有了非确定性, 能否增加它识别语言的能力?

非确定性概念无论在计算理论中起着重要的作用. 在有穷自动机的简单情况下, 透彻的理解这个概念是非常有益的. 对 FA 的模型稍加修改, 使之对同一输入符号, 从一个状态可以有零个、一个或多个的转移. 这种新模型, 称为非确定有穷自动机. 非确定的有穷自动机具有同时处在几个状态的能力, 在处理输入串时, 几个当前状态能“并行的”跳转到下一个状态.

2.3.1 形式定义

非确定有穷自动机的形式定义

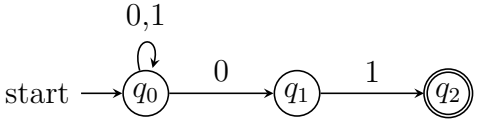
定义. *非确定有穷自动机 (NFA, Nondeterministic Finite Automaton) A 为五元组*

$$A = (Q, \Sigma, \delta, q_0, F)$$

1. Q *: 有穷状态集;*
2. Σ *: 有穷输入符号集或字母表;*
3. $\delta : Q \times \Sigma \to 2^Q$ *状态转移函数;*
4. $q_0 \in Q$ *: 为初始状态;*
5. $F \subseteq Q$ *: 为终结状态集或接受状态集.*

在形式定义上, NFA 与 DFA 的区别是转移函数和接受方式: NFA 转移函数一般为 $\delta(q, a) = \{p_1, p_2, \cdots, p_n\}$; 当输入串全部读入时, NFA 所处的状态中, 只要包括 F 中的状态, 就称为接受该串.

续例 7. 接受全部以 01 结尾的串的 NFA.



五元组为 $A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$, 转移函数 δ:

$$\delta(q_0, 0) = \{q_0, q_1\} \qquad \delta(q_1, 0) = \varnothing \qquad \delta(q_2, 0) = \varnothing$$
$$\delta(q_0, 1) = \{q_0\} \qquad \delta(q_1, 1) = \{q_2\} \qquad \delta(q_2, 1) = \varnothing$$

状态转移表:

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

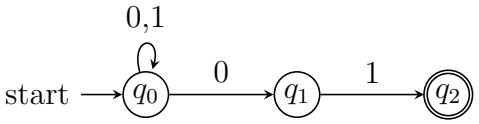
2.3.2 扩展转移函数

定义. 扩展 δ 到字符串, 定义扩展转移函数 $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$ 为

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

其中 $a \in \Sigma,\ w, x \in \Sigma^*$.

续例 7. 接受 01 结尾的串的 NFA, $\hat{\delta}$ 处理 00101 时每步的状态转移.



1. $\hat{\delta}(q_0, \varepsilon) = \{q_0\}$
2. $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$
3. $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \varnothing = \{q_0, q_1\}$
4. $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$
5. $\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \varnothing = \{q_0, q_1\}$
6. $\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

因为 q_2 是接受状态, 所以 NFA 接受 00101.

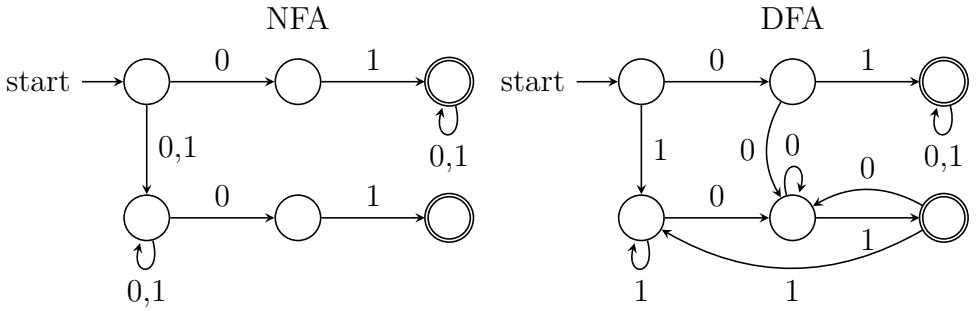
2.3.3 NFA 的语言

定义. *若 $N = (Q, \Sigma, \delta, q_0, F)$ 是一个 NFA, 则 N 接受的语言为*

$$\mathbf{L}(N) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \varnothing\}.$$

例 8. 设计 $L = \{w \in \{0,1\}^* \mid w$ 的首尾字符相同.$\}$ 的 NFA.

例 9. 设计 $L = \{w \in \{0,1\}^* \mid w$ either begin or ends with 01.$\}$ 的 NFA.



2.3.4 DFA 与 NFA 的等价性

每个 DFA 都是一个 NFA, 显然, NFA 接受的语言包含正则语言. 下面的定理给出, NFA 也仅接受正则语言. 证明的关键表明 DFA 能够模拟 NFA, 即, 对每个 NFA, 能够构造一个等价的 DFA. 使用一个 DFA 模拟一个 NFA 的方法是让 DFA 的状态对应于 NFA 的状态集合.

定理 1. *如果语言 L 被 NFA 接受, 当且仅当 L 被 DFA 接受.*

子集构造法

如果 NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ 构造 DFA

$$D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$$

1. $Q_D = 2^{Q_N}$;
2. $F_D = \{S \mid S \subseteq Q_N, S \cap F_N \neq \varnothing\}$;
3. $\forall S \subseteq Q_N$, $\forall a \in \Sigma$:
$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a).$$

那么有 $\mathbf{L}(D) = \mathbf{L}(N)$.

证明: 为证明 $\mathbf{L}(D) = \mathbf{L}(N)$, 对 $|w|$ 用归纳法, 往证

$$\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w).$$

1. 归纳基础: 当 $w = \varepsilon$ 时,

$$\hat{\delta}_D(\{q_0\}, \varepsilon) = \{q_0\} = \hat{\delta}_N(q_0, \varepsilon)$$

2. 归纳递推: 假设 $w = x$ ($x \in \Sigma^*$) 时成立, 当 $w = xa$ ($a \in \Sigma$) 时

$$\begin{aligned}
\hat{\delta}_N(q_0, xa) &= \cup_{p \in \hat{\delta}_N(q_0, x)} \delta_N(p, a) && \text{NFA 的 } \hat{\delta} \text{ 定义} \\
&= \cup_{p \in \hat{\delta}_D(\{q_0\}, x)} \delta_N(p, a) && \text{归纳假设} \\
&= \delta_D(\hat{\delta}_D(\{q_0\}, x), a) && D \text{ 的构造} \\
&= \hat{\delta}_D(\{q_0\}, xa) && \text{DFA 的 } \hat{\delta} \text{ 定义}
\end{aligned}$$

因此上式成立.

因为

$$\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$$

所以, 对 $\forall w \in \Sigma^*$ 有

$$\begin{aligned}
w \in \mathbf{L}(N) &\iff \hat{\delta}_N(q_0, w) \cap F_N \neq \varnothing && \text{NFA 的语言} \\
&\iff \hat{\delta}_D(\{q_0\}, w) \cap F_N \neq \varnothing && \text{刚证明的} \\
&\iff \hat{\delta}_D(\{q_0\}, w) \in F_D && D \text{ 的构造} \\
&\iff w \in \mathbf{L}(D) && \text{DFA 的语言}
\end{aligned}$$

所以

$$\mathbf{L}(D) = \mathbf{L}(N).$$

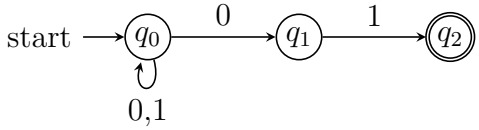
□

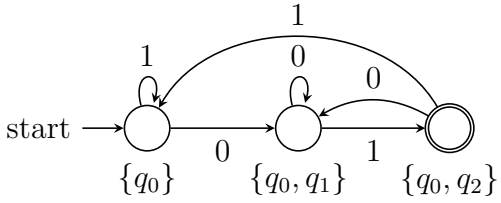
思考题

非确定性没能增加有穷自动机识别语言的能力, 原因是什么呢?

子集构造法: 构造与 NFA 等价的 DFA

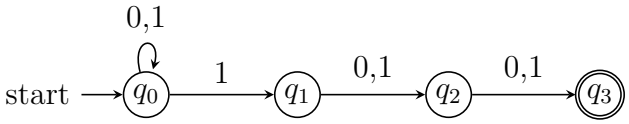
续例 7. 将接受全部以 01 结尾的串的 NFA 转换为 DFA.





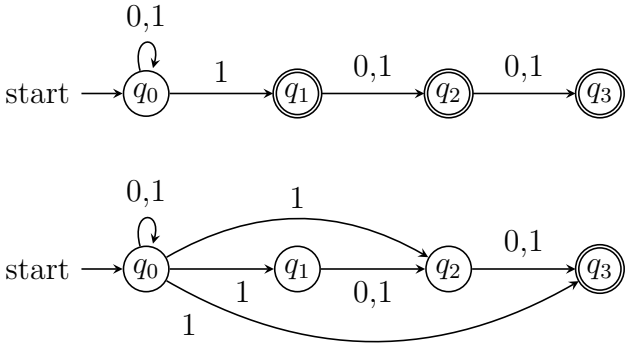
	0	1
$\to q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\varnothing$	$\{q_2\}$
$*q_2$	$\varnothing$	$\varnothing$
$\to \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
~~$\{q_1\}$~~	$\varnothing$	$\{q_2\}$
~~$*\{q_2\}$~~	$\varnothing$	$\varnothing$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
~~$\varnothing$~~	$\varnothing$	$\varnothing$
~~$*\{q_1, q_2\}$~~	$\varnothing$	$\{q_2\}$
~~$*\{q_0, q_1, q_2\}$~~	$\{q_0, q_1\}$	$\{q_0, q_2\}$

例 10. 设计 NFA 识别 $L = \{w \in \{0,1\}^* \mid w$ 倒数第 3 个字符是 1$\}$.

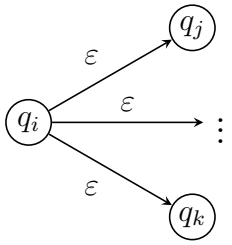


2.4 带有空转移的非确定有穷自动机

例 11. 设计 $L = \{w \in \{0,1\}^* \mid w$ 倒数 3 个字符至少有一个是 1$\}$ 的 NFA.

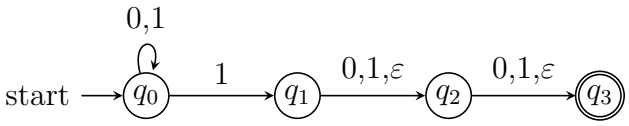


状态的 ε 转移



- 允许状态因空串 ε 而转移, 即不消耗输入字符就发生状态的改变
- 使自动机的设计更容易

续例 11.



2.4.1 形式定义

带空转移非确定有穷自动机的形式定义

定义. *带空转移非确定有穷自动机 (ε-NFA) A 为五元组*

$$A = (Q, \Sigma, \delta, q_0, F)$$

1. Q *: 有穷状态集;*
2. Σ *: 有穷输入符号集或字母表;*
3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$*, 转移函数;*
4. $q_0 \in Q$ *: 初始状态;*
5. $F \subseteq Q$ *: 终结状态集或接受状态集.*

ε-NFA, NFA, DFA 之间的主要区别

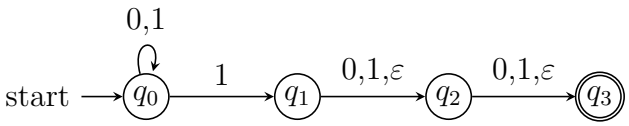
1. 自动机在某状态, 读入某个字符时, 可能有多个转移;
2. 自动机在某状态, 读入某个字符时, 可能没有转移;
3. 自动机在某状态, 可能不读入字符, 就进行转移.

注意

此后, 不再明确区分 ε-NFA 和 NFA, 而认为它们都是 NFA.

续例 11. $L = \{w \in \{0,1\}^* \mid w$ 倒数 3 个字符至少有一个是 1$\}$ 的 ε-NFA.

利用 ε 转移设计的有穷自动机:



状态转移表:

	0	1	ε
$\to q_0$	$\{q_0\}$	$\{q_0, q_1\}$	$\varnothing$
q_1	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$
$*q_3$	$\varnothing$	$\varnothing$	$\varnothing$

续例 11. $L = \{w \in \{0,1\}^* \mid w$ 倒数 3 个字符至少有一个是 1$\}$ 的 ε-NFA.

当输入字符串是 011 时, ε-NFA 的状态变化.

思考题

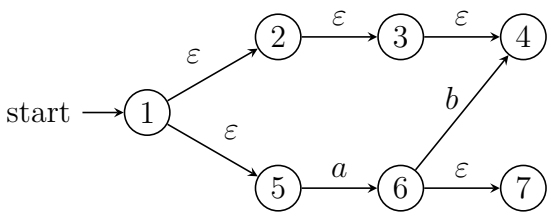
1. 如果初始状态有 ε 转移, 第 1 个字符该如何处理?
2. 如果最后的字符所到的状态有 ε 转移呢?

2.4.2 ε-闭包

状态的 ε-闭包

定义. 状态 q 的ε-闭包 *(ε-Closure)*, 记为 $\textsc{Eclose}(q)$, 表示从 q 经过 ε 序列可达的全部状态集合, 递归定义为:

1. $q \in \textsc{Eclose}(q)$;
2. $\forall p \in \textsc{Eclose}(q)$, 若 $r \in \delta(p, \varepsilon)$, 则 $r \in \textsc{Eclose}(q)$.

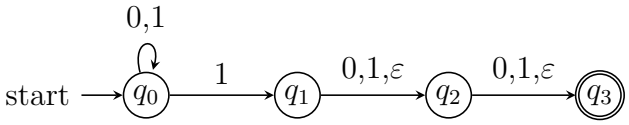


状态集合的 ε-闭包

定义. 状态集 S 的 ε-闭包为

$$\textsc{Eclose}(S) = \bigcup_{q \in S} \textsc{Eclose}(q).$$

续例 11. $L = \{w \in \{0,1\}^* \mid w$ 倒数 3 个字符至少有一个是 1$\}$ 的 NFA.



状态转移表及每个状态的闭包:

	0	1	ε	$\textsc{Eclose}(\sqcup)$
$\to q_0$	$\{q_0\}$	$\{q_0, q_1\}$	$\varnothing$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$	$\{q_1, q_2, q_3\}$
q_2	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$	$\{q_2, q_3\}$
$*q_3$	$\varnothing$	$\varnothing$	$\varnothing$	$\{q_3\}$

2.4.3 扩展转移函数

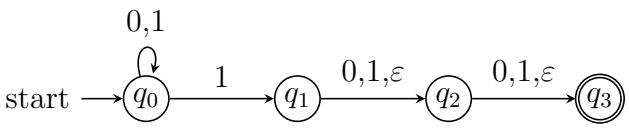
定义. 扩展 δ 到字符串, 定义扩展转移函数 $\hat{\delta} : Q \times \Sigma^* \to 2^Q$ 为

$$\hat{\delta}(q, w) = \begin{cases} \textsc{Eclose}(q) & w = \varepsilon \\ \textsc{Eclose}\big(\bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a)\big) & w = xa \end{cases}$$

其中 $a \in \Sigma,\ w, x \in \Sigma^*$.

若设 $\hat{\delta}(q, x) = \{p_1, p_2, \ldots, p_k\}$, 则从每个 p_i 经过 a 边到达的所有状态为 $\bigcup_{i=1}^{k} \delta(p_i, a)$; 再设 $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \ldots, r_m\}$, 则每个 r_j 再求 ε-闭包, 所得到的状态集, 定义为 $\hat{\delta}(q, w)$; 即 $\hat{\delta}(q, w) = \textsc{Eclose}(\{r_1, r_2, \ldots, r_m\})$

续例 11. 若 $L = \{w \in \{0,1\}^* \mid w$ 倒数 3 个字符至少有一个是 1$\}$ 的 ε-NFA 如下, 求 $\hat{\delta}(q_0, 10)$.



$$\begin{aligned} \hat{\delta}(q_0, \varepsilon) &= \textsc{Eclose}(q_0) = \{q_0\} \\ \hat{\delta}(q_0, 1) &= \textsc{Eclose}\big(\cup_{p \in \hat{\delta}(q_0, \varepsilon)} \delta(p, 1)\big) \\ &= \textsc{Eclose}(\hat{\delta}(q_0, 1)) = \textsc{Eclose}(\{q_0, q_1\}) = \{q_0, q_1, q_2, q_3\} \end{aligned}$$

$$\begin{aligned}\hat{\delta}(q_0, 10) &= \textsc{Eclose}\big(\cup_{p\in\hat{\delta}(q_0,1)}\,\delta(p,0)\big)\\ &= \textsc{Eclose}(\delta(q_0,0)\cup\delta(q_1,0)\cup\delta(q_2,0)\cup\delta(q_3,0))\\ &= \textsc{Eclose}(\{q_0,q_2,q_3\}) = \{q_0,q_2,q_3\}\end{aligned}$$

2.4.4 ε-NFA 的语言

定义. *若 $E=(Q,\Sigma,\delta,q_0,F)$ 是一个ε-NFA, 则 E 接受的语言为*

$$\mathbf{L}(E)=\{w\in\Sigma^*\mid\hat{\delta}(q_0,w)\cap F\neq\varnothing\}.$$

2.4.5 ε-NFA 与 DFA 等价性

若有 ε-NFA E, 构造 DFA D, 使 $\mathbf{L}(D)=\mathbf{L}(E)$, 方法与子集构造法类似, 但使用 ε 闭包代替状态转移后的集合, 也称为消除空转移的子集构造法.

消除空转移的子集构造法

构造方法

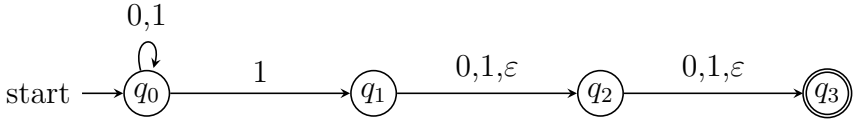
如果 ε-NFA $E=(Q_E,\Sigma,\delta_E,q_E,F_E)$, 构造 DFA

$$D=(Q_D,\Sigma,\delta_D,q_D,F_D)$$

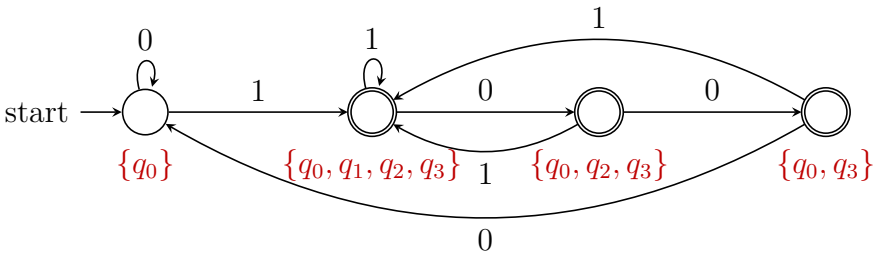
1. $Q_D=2^{Q_E}$, 或 $Q_D=\{S\subseteq Q_E\mid S=\textsc{Eclose}(S)\}$;
2. $q_D=\textsc{Eclose}(q_E)$;
3. $F_D=\{S\mid S\in Q_D, S\cap F_E\neq\varnothing\}$;
4. $\forall S\in Q_D$, $\forall a\in\Sigma$,
$$\delta_D(S,a)=\textsc{Eclose}\Big(\bigcup_{p\in S}\delta_E(p,a)\Big).$$

那么有 $\mathbf{L}(D)=\mathbf{L}(E)$.

续例 11. 将下图 L 的 ε-NFA, 转为等价的 DFA.



	0	1	ε	ECLOSE()
$\to q_0$	$\{q_0\}$	$\{q_0, q_1\}$	$\varnothing$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$	$\{q_1, q_2, q_3\}$
q_2	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$	$\{q_2, q_3\}$
$*q_3$	$\varnothing$	$\varnothing$	$\varnothing$	$\{q_3\}$



	0	1
$\to \{q_0\}$	$\{q_0\}$	$\{q_0, q_1, q_2, q_3\}$
$*\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$
$*\{q_0, q_2, q_3\}$	$\{q_0, q_3\}$	$\{q_0, q_1, q_2, q_3\}$
$*\{q_0, q_3\}$	$\{q_0\}$	$\{q_0, q_1, q_2, q_3\}$

定理 2. *如果语言 L 被 ε-NFA 接受, 当且仅当 L 被 DFA 接受.*

证明: 必要性显然成立, 因为任何 DFA 都是 ε-NFA. 为证明充分性, 对 w 归纳, 往证 $\hat{\delta}_E(q_E, w) = \hat{\delta}_D(q_D, w)$.

1. 当 $w = \varepsilon$ 时

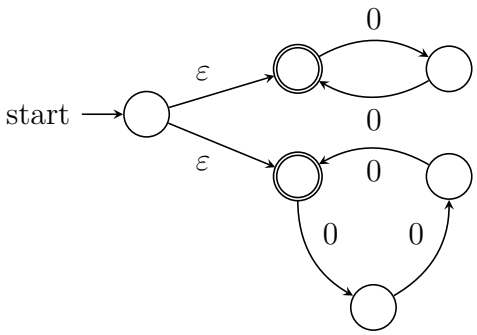
$$\hat{\delta}_E(q_E, \varepsilon) = \text{ECLOSE}(q_E) = q_D = \hat{\delta}_D(q_D, \varepsilon).$$

2. 当 $w = xa$ 时

$$\hat{\delta}_E(q_E, xa) = \text{ECLOSE}\Big(\bigcup_{p \in \hat{\delta}_E(q_E, x)} \delta_E(p, a)\Big) = \text{ECLOSE}\Big(\bigcup_{p \in \hat{\delta}_D(q_D, x)} \delta_E(p, a)\Big)$$
$$= \delta_D(\hat{\delta}_D(q_D, x), a) = \hat{\delta}_D(q_D, xa)$$

□

例 12. Design ε-NFA for language: $\{0^k \mid k \text{ is a multiple of 2 or 3}\}$.


start
ε
ε
0
0
0
0
0

Chapter 3

正则表达式

3.1 正则表达式

- 有穷自动机
  - 通过机器装置描述正则语言
  - 用计算机编写相应算法, 易于实现
- 正则表达式
  - 通过表达式描述正则语言, 代数表示方法, 使用方便
  - 应用广泛
    - `grep` 工具 (Global Regular Expression and Print)
    - Emacs / Vim 文本编辑器
    - `lex` / `flex` 词法分析器
    - 各种程序设计语言 Python / Perl / Haskull / $\cdots$

3.1.1 语言的运算

设 L 和 M 是两个语言, 那么

并 (Union) $$L \cup M = \{w \mid w \in L \text{ 或 } w \in M\}$$

连接 (Concatenation) $$L \cdot M = \{w \mid w = xy,\ x \in L \text{ 且 } y \in M\}$$

幂 (Power) $$L^0 = \{\varepsilon\}$$
$$L^1 = L$$
$$L^n = L^{n-1} \cdot L$$

克林闭包 (Kleene Closure) $$L^* = \bigcup_{i=0}^{\infty} L^i$$

例 1. 若有语言 $L = \{0, 11\}$ 和 $M = \{\varepsilon, 001\}$, 那么

$L \cup M =$ $L^0 =$

$LM =$ $L^1 =$

$ML =$ $L^2 =$

例 2. 对于空语言 $\varnothing$

$$\begin{aligned} & \varnothing^0 = \{\varepsilon\} \\ \forall n \geq 1, \quad & \varnothing^n = \varnothing \\ & \varnothing^* = \{\varepsilon\} \end{aligned}$$

四则运算表达式的递归定义:

1. 任何数都是四则运算表达式;
2. 如果 a 和 b 是四则运算表达式, 那么

$$a + b,\ a - b,\ a \times b,\ a \div b \text{和} (a)$$

都是四则运算表达式.

3.1.2 正则表达式的递归定义

定义. 如果 Σ 为字母表, 则 Σ 上的正则表达式 *(Regular Expression)* 递归定义为:

1. $\varnothing$ 是一个正则表达式, 表示空语言;
   $\boldsymbol{\varepsilon}$ 是一个正则表达式, 表示语言 $\{\varepsilon\}$;
   $\forall a \in \Sigma$, $\mathbf{a}$ 是一个正则表达式, 表示语言 $\{a\}$;
2. 如果正则表达式 $\mathbf{r}$ 和 $\mathbf{s}$ 分别表示语言 R 和 S, 那么

$$\mathbf{r} + \mathbf{s},\ \mathbf{rs},\ \mathbf{r}^* \text{ 和 } (\mathbf{r})$$

都是正则表达式, 分别表示语言

$$R \cup S,\ R \cdot S,\ R^* \text{ 和 } R.$$

此外正闭包定义为 $\mathbf{r}^+ = \mathbf{r}\mathbf{r}^*$, 显然 $\mathbf{r}^* = \mathbf{r}^+ + \boldsymbol{\varepsilon}$.

3.1.3 运算符的优先级

正则表达式中三种运算以及括号的优先级:

1. 首先, “括号”优先级最高;
2. 其次, “星”运算: $\mathbf{r}^*$;
3. 然后, “连接”运算: $\mathbf{rs}$, $\mathbf{r}\cdot\mathbf{s}$;
4. 最后, “加”最低: $\mathbf{r}+\mathbf{s}$, $\mathbf{r}\cup\mathbf{s}$;

例 3.

$$\begin{aligned}\mathbf{1}+\mathbf{01}^* &= \mathbf{1}+(\mathbf{0}(\mathbf{1}^*))\\ &\neq \mathbf{1}+(\mathbf{01})^*\\ &\neq (\mathbf{1}+\mathbf{01})^*\\ &\neq (\mathbf{1}+\mathbf{0})\mathbf{1}^*\end{aligned}$$

3.1.4 正则表达式示例

例 4.

E	$\mathbf{L}(E)$
$\mathbf{a}+\mathbf{b}$	$\mathbf{L}(\mathbf{a})\cup\mathbf{L}(\mathbf{b})=\{a\}\cup\{b\}=\{a,b\}$
$\mathbf{bb}$	$\mathbf{L}(\mathbf{b})\cdot\mathbf{L}(\mathbf{b})$=$\{b\}\cdot\{b\}=\{bb\}$
$(\mathbf{a}+\mathbf{b})(\mathbf{a}+\mathbf{b})$	$\{a,b\}\{a,b\}=\{aa,ab,ba,bb\}$
$(\mathbf{a}+\mathbf{b})^*(\mathbf{a}+\mathbf{bb})$	$\{a,b\}^*\{a,bb\}=\{a,b\}^*\{a\}\cup\{a,b\}^*\{bb\}=$ $\{w\in\{a,b\}^* \mid w$ 仅以 a 或 bb 结尾.$\}$
$\mathbf{1}+(\mathbf{01})^*$	$\{1,\varepsilon,01,0101,010101,\ldots\}$
$(\mathbf{0}+\mathbf{1})^*\mathbf{01}(\mathbf{0}+\mathbf{1})^*$	$\{x01y \mid x,y\in\{0,1\}^*\}$

例 5. 给出正则表达式 $(\mathbf{aa})^*(\mathbf{bb})^*\mathbf{b}$ 定义的语言.

$$\mathbf{L}((\mathbf{aa})^*(\mathbf{bb})^*\mathbf{b})=\mathbf{L}((\mathbf{aa})^*)\cdot\mathbf{L}((\mathbf{bb})^*)\cdot\mathbf{L}(\mathbf{b})$$

$$
\begin{aligned}
&= (\{a\}\{a\})^*(\{b\}\{b\})^*\{b\} \\
&= \{a^2\}^*\{b^2\}^*\{b\} \\
&= \{a^{2n}b^{2m+1} \mid n \geq 0, m \geq 0\}
\end{aligned}
$$

例 6. Design regular expression for $L = \{w \mid w$ consists of 0's and 1's, and the third symbol from the right end is 1.$\}$

$$(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})$$

例 7. Design regular expression for $L = \{w \mid w \in \{0,1\}^*$ and w has no pair of consecutive 0's.$\}$

$$\mathbf{1}^*(\mathbf{011}^*)^*(\mathbf{0}+\varepsilon) \text{ 或 } (\mathbf{1}+\mathbf{01})^*(\mathbf{0}+\varepsilon)$$

例. Design regular expression for $L = \{w \mid w \in \{0,1\}^*$ and w contains 01$\}$.

例. Write a regular expression for $L = \{w \in \{0,1\}^* \mid 0$ and 1 alternate in $w\}$.

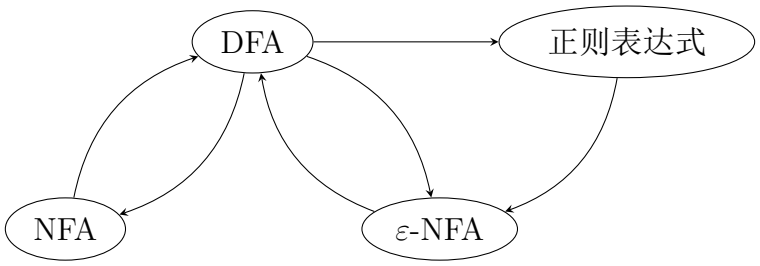
例. Find a regular expression for the set $\{a^n b^m \mid (n+m)$ is odd $\}$.

例. Give regular expression for the complement of $L = \{a^n b^m \mid n \geq 3, m \leq 4\}$.

例. Write a regular expression for the set of all C real numbers.

3.2 有穷自动机和正则表达式

DFA, NFA, ε-NFA 和正则表达式的等价性



3.2.1 由 DFA 到正则表达式, 递归表达式法

定理 3. *若 $L = \mathbf{L}(A)$ 是某 DFA A 的语言, 那么存在正则表达式 R 满足 $L = \mathbf{L}(R)$.*

证明: 设 DFA A 的状态共有 n 个, 对 DFA A 的状态编号, 令 1 为开始状态, 即

$$A = (\{1, 2, \ldots, n\}, \Sigma, \delta, 1, F),$$

设正则表达式 $R_{ij}^{(k)}$ 表示从 i 到 j 但中间节点不超过 k 全部路径的字符串集:

$$R_{ij}^{(k)} = \{x \mid \hat{\delta}(i, x) = j, x \text{经过的状态除两端外都不超过} k\}.$$

也就是说, 正则表达式 $R_{ij}^{(k)}$ 是所有那样的字符串的集合, 它能够使有穷自动机从状态 i 到达状态 j, 但中间的路径上, 不经过编号高于 k 的任何状态.

如果 1 是开始结点, 对每个属于终态 F 的结点 j, 正则表达式 $R_{1j}^{(n)}$ 都是这个自动机所识别语言的一部分. 那么与 $A = (\{1, 2, \ldots, n\}, \Sigma, \delta, 1, F)$ 等价的正则表达式为

$$\bigcup_{j \in F} R_{1j}^{(n)}$$

且递归式为

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$$

$$R_{ij}^{(0)} = \begin{cases} \{a \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\} & i = j \end{cases}$$

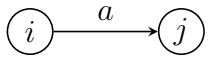
下面对 k 归纳, 证明可用以上递归式求得 $R_{ij}^{(k)}$.

归纳基础: 当 $i \neq j$, $k = 0$ 时, 即 i 到 j 没经过任何中间节点

- 没有 i 到 j 的状态转移

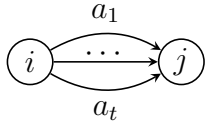
$$R_{ij}^{(0)} = \varnothing$$

- 有一个 i 到 j 的状态转移



$$R_{ij}^{(0)} = \mathbf{a}$$

- 有多个 i 到 j 的状态转移



$$R_{ij}^{(0)} = \mathbf{a}_1 + \mathbf{a}_2 + \cdots + \mathbf{a}_t$$

归纳基础 (续): 当 $i = j$, $k = 0$ 时, 即从 i 到自身没经过任何中间节点

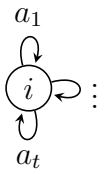
- 状态 i 没有到自己的转移

$$R_{ii}^{(0)} = \boldsymbol{\varepsilon}$$

- 状态 i 有一个到自身的转移

$$R_{ii}^{(0)} = \mathbf{a} + \boldsymbol{\varepsilon}$$

- 状态 i 有多个到自身的转移

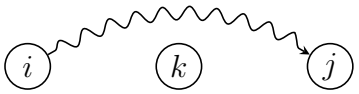


$$R_{ii}^{(0)} = \mathbf{a}_1 + \mathbf{a}_2 + \cdots + \mathbf{a}_t + \boldsymbol{\varepsilon}$$

归纳假设: 假设已知 $R_{ij}^{(k-1)}$, $R_{ik}^{(k-1)}$, $R_{kk}^{(k-1)}$ 和 $R_{kj}^{(k-1)}$.

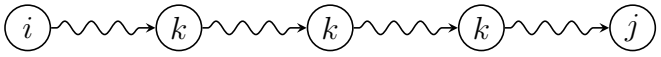
归纳递推: 那么 $R_{ij}^{(k)}$ 中全部路径, 可用节点 k 分为两部分

- 从 i 到 j 不经过 k 的



$$R_{ij}^{(k)} = R_{ij}^{(k-1)}$$

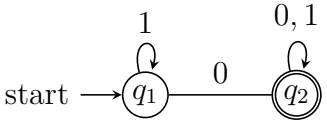
- 从 i 到 j 经过 k 的



$$R_{ij}^{(k)} = R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$$

因此 $R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$. □

例 8. 将如图 DFA 转换为正则表达式.



- 计算 $R_{ij}^{(0)}$

$R_{ij}^{(k)}$	$k=0$
$R_{11}^{(0)}$	$\boldsymbol{\varepsilon+1}$
$R_{12}^{(0)}$	$\mathbf{0}$
$R_{21}^{(0)}$	$\varnothing$
$R_{22}^{(0)}$	$\boldsymbol{\varepsilon+0+1}$

- 计算 $R_{ij}^{(1)}=R_{ij}^{(0)}+R_{i1}^{(0)}(R_{11}^{(0)})^*R_{1j}^{(0)}$

$R_{ij}^{(k)}$	$k=0$
$R_{11}^{(0)}$	$\boldsymbol{\varepsilon+1}$
$R_{12}^{(0)}$	$\mathbf{0}$
$R_{21}^{(0)}$	$\varnothing$
$R_{22}^{(0)}$	$\boldsymbol{\varepsilon+0+1}$

$R_{ij}^{(k)}$	$k=1$
$R_{11}^{(1)}$	$(\boldsymbol{\varepsilon+1})+(\boldsymbol{\varepsilon+1})(\boldsymbol{\varepsilon+1})^*(\boldsymbol{\varepsilon+1})$
$R_{12}^{(1)}$	$\mathbf{0}+(\boldsymbol{\varepsilon+1})(\boldsymbol{\varepsilon+1})^*\mathbf{0}$
$R_{21}^{(1)}$	$\varnothing+\varnothing(\boldsymbol{\varepsilon+1})^*(\boldsymbol{\varepsilon+1})$
$R_{22}^{(1)}$	$\boldsymbol{\varepsilon+0+1}+\varnothing(\boldsymbol{\varepsilon+1})^*\mathbf{0}$

- 几个基本的化简规则

  如果 **r** 和 **s** 是两个正则表达式

$$
\begin{aligned}
(\boldsymbol{\varepsilon}+\mathbf{r})^* &= \mathbf{r}^* \\
(\boldsymbol{\varepsilon}+\mathbf{r})\mathbf{r}^* &= \mathbf{r}^* \\
\mathbf{r}+\mathbf{r}\mathbf{s}^* &= \mathbf{r}\mathbf{s}^* \\
\varnothing\mathbf{r}=\mathbf{r}\varnothing &= \varnothing \qquad \text{(零元)} \\
\varnothing+\mathbf{r}=\mathbf{r}+\varnothing &= \mathbf{r} \qquad \text{(单位元)}
\end{aligned}
$$

- 化简 $R_{ij}^{(1)}$

$R_{ij}^{(k)}$	$k=1$	化简
$R_{11}^{(1)}$	$(\boldsymbol{\varepsilon+1})+(\boldsymbol{\varepsilon+1})(\boldsymbol{\varepsilon+1})^*(\boldsymbol{\varepsilon+1})$	$\mathbf{1}^*$
$R_{12}^{(1)}$	$\mathbf{0}+(\boldsymbol{\varepsilon+1})(\boldsymbol{\varepsilon+1})^*\mathbf{0}$	$\mathbf{1}^*\mathbf{0}$
$R_{21}^{(1)}$	$\varnothing+\varnothing(\boldsymbol{\varepsilon+1})^*(\boldsymbol{\varepsilon+1})$	$\varnothing$
$R_{22}^{(1)}$	$\boldsymbol{\varepsilon+0+1}+\varnothing(\boldsymbol{\varepsilon+1})^*\mathbf{0}$	$\boldsymbol{\varepsilon+0+1}$

- 计算 $R_{ij}^{(2)}=R_{ij}^{(1)}+R_{i2}^{(1)}(R_{22}^{(1)})^*R_{2j}^{(1)}$

$R_{ij}^{(k)}$	$k=1$
$R_{11}^{(1)}$	$\mathbf{1}^*$
$R_{12}^{(1)}$	$\mathbf{1}^*\mathbf{0}$
$R_{21}^{(1)}$	$\varnothing$
$R_{22}^{(1)}$	$\boldsymbol{\varepsilon+0+1}$

$R_{ij}^{(k)}$	$k=2$
$R_{11}^{(2)}$	$\mathbf{1}^*+\mathbf{1}^*\mathbf{0}(\boldsymbol{\varepsilon+0+1})^*\varnothing$
$R_{12}^{(2)}$	$\mathbf{1}^*\mathbf{0}+\mathbf{1}^*\mathbf{0}(\boldsymbol{\varepsilon+0+1})^*(\boldsymbol{\varepsilon+0+1})$
$R_{21}^{(2)}$	$\varnothing+(\boldsymbol{\varepsilon+0+1})(\boldsymbol{\varepsilon+0+1})^*\varnothing$
$R_{22}^{(2)}$	$\boldsymbol{\varepsilon+0+1}+(\boldsymbol{\varepsilon+0+1})(\boldsymbol{\varepsilon+0+1})^*(\boldsymbol{\varepsilon+0+1})$

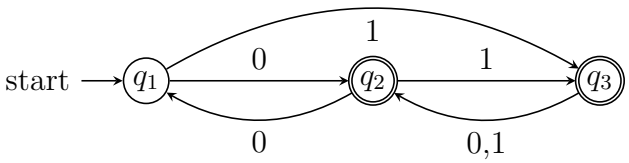
- 化简 $R_{ij}^{(2)}$

$R_{ij}^{(k)}$	$k=2$	化简
$R_{11}^{(2)}$	$\mathbf{1}^* + \mathbf{1}^*\mathbf{0}(\varepsilon + \mathbf{0} + \mathbf{1})^*\varnothing$	$\mathbf{1}^*$
$R_{12}^{(2)}$	$\mathbf{1}^*\mathbf{0} + \mathbf{1}^*\mathbf{0}(\varepsilon + \mathbf{0} + \mathbf{1})^*(\varepsilon + \mathbf{0} + \mathbf{1})$	$\mathbf{1}^*\mathbf{0}(\mathbf{0} + \mathbf{1})^*$
$R_{21}^{(2)}$	$\varnothing + (\varepsilon + \mathbf{0} + \mathbf{1})(\varepsilon + \mathbf{0} + \mathbf{1})^*\varnothing$	$\varnothing$
$R_{22}^{(2)}$	$\varepsilon + \mathbf{0} + \mathbf{1} + (\varepsilon + \mathbf{0} + \mathbf{1})(\varepsilon + \mathbf{0} + \mathbf{1})^*(\varepsilon + \mathbf{0} + \mathbf{1})$	$(\mathbf{0} + \mathbf{1})^*$

- 因只有 q_2 是接受状态, 所以该 DFA 正则表达式为

$$R_{12}^{(2)} = \mathbf{1}^*\mathbf{0}(\mathbf{0} + \mathbf{1})^*.$$

例 9. 将如图 DFA 转换为正则表达式.



	$k=0$	$k=1$	$k=2$
$R_{11}^{(k)}$	ε	ε	$(\mathbf{00})^*$
$R_{12}^{(k)}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}(\mathbf{00})^*$
$R_{13}^{(k)}$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{0}^*\mathbf{1}$
$R_{21}^{(k)}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}(\mathbf{00})^*$
$R_{22}^{(k)}$	ε	$\varepsilon + \mathbf{00}$	$(\mathbf{00})^*$
$R_{23}^{(k)}$	$\mathbf{1}$	$\mathbf{1} + \mathbf{01}$	$\mathbf{0}^*\mathbf{1}$
$R_{31}^{(k)}$	$\varnothing$	$\varnothing$	$(\mathbf{0} + \mathbf{1})(\mathbf{00})^*\mathbf{0}$
$R_{32}^{(k)}$	$\mathbf{0} + \mathbf{1}$	$\mathbf{0} + \mathbf{1}$	$(\mathbf{0} + \mathbf{1})(\mathbf{00})^*$
$R_{33}^{(k)}$	ε	ε	$\varepsilon + (\mathbf{0} + \mathbf{1})\mathbf{0}^*\mathbf{1}$

仅状态 2 和 3 是接受状态:

$$\begin{aligned}
R_{12}^{(3)} &= R_{12}^{(2)} + R_{13}^{(2)}(R_{33}^{(2)})^* R_{32}^{(2)} \\
&= \mathbf{0}(\mathbf{00})^* + \mathbf{0}^*\mathbf{1}(\varepsilon + (\mathbf{0} + \mathbf{1})\mathbf{0}^*\mathbf{1})^*(\mathbf{0} + \mathbf{1})(\mathbf{00})^* \\
&= \mathbf{0}(\mathbf{00})^* + \mathbf{0}^*\mathbf{1}((\mathbf{0} + \mathbf{1})\mathbf{0}^*\mathbf{1})^*(\mathbf{0} + \mathbf{1})(\mathbf{00})^* \\
R_{13}^{(3)} &= R_{13}^{(2)} + R_{13}^{(2)}(R_{33}^{(2)})^* R_{33}^{(2)} \\
&= \mathbf{0}^*\mathbf{1} + \mathbf{0}^*\mathbf{1}(\varepsilon + (\mathbf{0} + \mathbf{1})\mathbf{0}^*\mathbf{1})^*(\varepsilon + (\mathbf{0} + \mathbf{1})\mathbf{0}^*\mathbf{1}) \\
&= \mathbf{0}^*\mathbf{1}((\mathbf{0} + \mathbf{1})\mathbf{0}^*\mathbf{1})^*
\end{aligned}$$

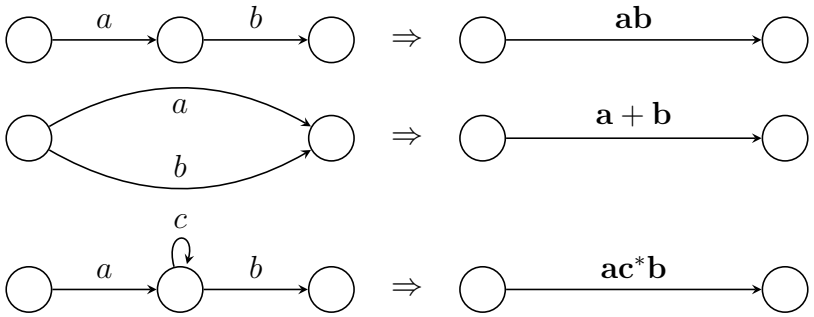
则 DFA 的正则表达式为:

$$R_{12}^{(3)} + R_{13}^{(3)} = \mathbf{0}^*\mathbf{1}((\mathbf{0} + \mathbf{1})\mathbf{0}^*\mathbf{1})^*(\varepsilon + (\mathbf{0} + \mathbf{1})(\mathbf{00})^*) + \mathbf{0}(\mathbf{00})^*.$$

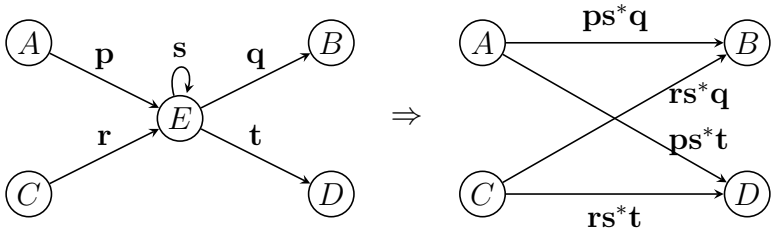
3.2.2 由 DFA 到正则表达式, 状态消除法

从有穷自动机中删除状态, 并使用新的路径替换被删除的路径, 在新路径上设计新的正则表达式, 产生一个等价的“自动机”.

- 从 DFA 中逐个删除状态
- 用标记了正则表达式的新路径替换被删掉的路径
- 保持“自动机”等价.

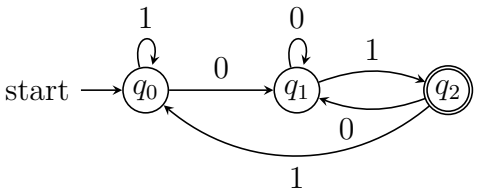


- 更一般的情况如图
- 若要删除状态 E, 需添加相应路径

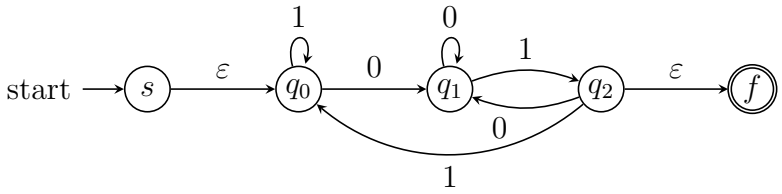


要为被删除的状态 S 的每个“入”和“出”路径的组合, 补一条等价的新路径, 结点上的循环用闭包表示. 保持新路径与被删掉的路径集合等价.

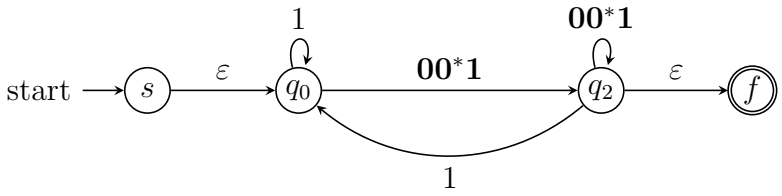
例 10. 利用状态消除法, 设计下图自动机的正则表达式.



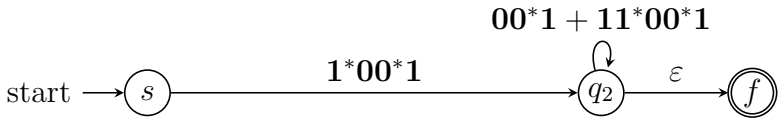
1. 利用空转移, 添加新的开始 s 和结束状态 f:



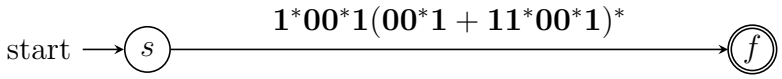
2. 消除状态 q_1, 添加路径 $q_0 \to q_2$ 和 $q_2 \to q_2$:



3. 消除状态 q_0, 添加路径 $s \to q_2$ 和 $q_2 \to q_2$:



4. 消除状态 q_2, 添加路径 $s \to f$:



5. 因此该自动机的正则表达式为

$$\mathbf{1^*00^*1(00^*1 + 11^*00^*1)^*}.$$

3.2.3 由正则表达式到 ε-NFA

定理 4. *正则表达式定义的语言, 都可被有穷自动机识别.*

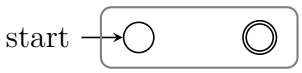
由正则表达式构造 ε-NFA(比定理 4 更严格的命题)

任何正则表达式 $\mathbf{r}$, 都存在等价的 ε-NFA A, 即 $\mathbf{L}(A) = \mathbf{L}(\mathbf{r})$, 并且 A 满足:

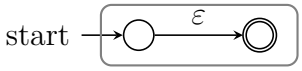
1. 仅有一个接收状态;
2. 没有进入开始状态的边;
3. 没有离开接受状态的边.

证明: 归纳基础:

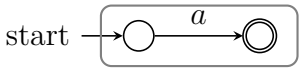
1. 对于 $\varnothing$, 有 ε-NFA:



2. 对于 $\boldsymbol{\varepsilon}$, 有 ε-NFA:



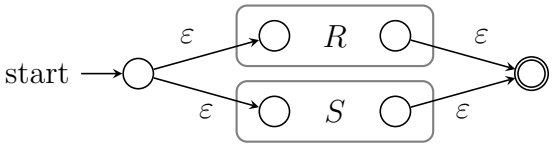
3. $\forall a \in \Sigma$, 对于 $\mathbf{a}$, 有 ε-NFA:



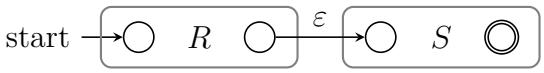
归纳递推: 假设正则表达式 $\mathbf{r}$ 和 $\mathbf{s}$ 的 ε-NFA 分别为 R 和 S

那么正则表达式 $\mathbf{r}+\mathbf{s}$, $\mathbf{rs}$ 和 $\mathbf{r}^*$, 可由 R 和 S 分别构造如下:

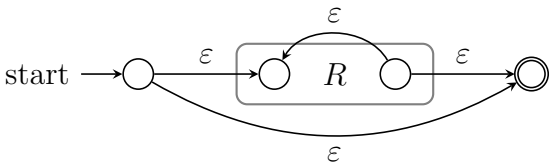
1. 对于 $\mathbf{r}+\mathbf{s}$, 有 ε-NFA:



2. 对于 $\mathbf{rs}$, 有 ε-NFA:

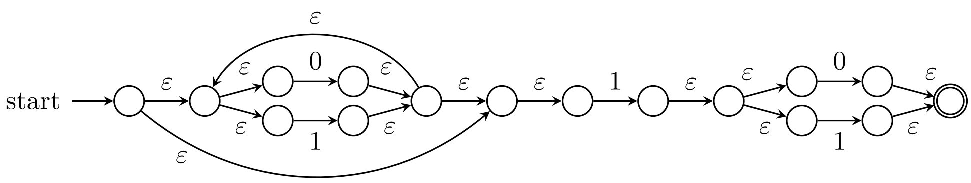


3. 对于 $\mathbf{r}^*$, 有 ε-NFA:



因此任何结构的正则表达式, 都可递归构造出等价的 ε-NFA. □

例 11. 正则表达式 $(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$ 构造为 ε-NFA.



思考题

正则表达式到 ε-NFA 构造方法中的 3 个限制条件, 都有必要吗?

3.3 正则表达式的代数定律

3.3.1 基本的代数定律

定义. 含有变量的两个正则表达式, 如果以任意语言替换其变量, 二者所表示的语言仍然相同, 则称这两个正则表达式等价. 在这样的意义下, 正则表达式满足一些代数定律.

- 并运算

$$(L+M)+N = L+(M+N) \quad \text{(结合律)}$$
$$L+M = M+L \quad \text{(交换律)}$$
$$L+L = L \quad \text{(幂等律)}$$
$$\varnothing + L = L + \varnothing = L \quad \text{(单位元 } \varnothing\text{)}$$

- 连接运算

$$(LM)N = L(MN) \quad \text{(结合律)}$$
$$\boldsymbol{\varepsilon} L = L\boldsymbol{\varepsilon} = L \quad \text{(单位元 } \boldsymbol{\varepsilon}\text{)}$$
$$\varnothing L = L\varnothing = \varnothing \quad \text{(零元 } \varnothing\text{)}$$
$$LM \neq ML$$

- 分配率

$$L(M+N) = LM+LN \quad \text{(左分配律)}$$
$$(M+N)L = ML+NL \quad \text{(右分配律)}$$

- 闭包运算

$$(L^*)^* = L^*$$

$$\varnothing^* = \varepsilon$$
$$\varepsilon^* = \varepsilon$$
$$L^* = L^+ + \varepsilon$$
$$(\varepsilon + L)^* = L^*$$

3.3.2 发现与验证代数定律

检验方法

要判断表达式 E 和 F 是否等价, 其中变量为 $L_1, \ldots, L_n$:

1. 将变量替换为具体表达式, 得正则表达式 $\mathbf{r}$ 和 $\mathbf{s}$, 例如, 替换 L_i 为 $\mathbf{a}_i$;
2. 判断 $\mathbf{L}(\mathbf{r}) \stackrel{?}{=} \mathbf{L}(\mathbf{s})$, 如果相等则 $E = F$, 否则 $E \neq F$.

例 12. 判断 $(L + M)^* = (L^*M^*)^*$.

1. 将 L 和 M 替换为 $\mathbf{a}$ 和 $\mathbf{b}$;
2. $(\mathbf{a} + \mathbf{b})^* \stackrel{?}{=} (\mathbf{a}^*\mathbf{b}^*)^*$;
3. 因为 $\mathbf{L}((\mathbf{a} + \mathbf{b})^*) = \mathbf{L}((\mathbf{a}^*\mathbf{b}^*)^*)$;
4. 所以 $(L + M)^* = (L^*M^*)^*$.

例 13. 判断 $L + ML = (L + M)L$.

1. 将 L 和 M 替换为 $\mathbf{a}$ 和 $\mathbf{b}$;
2. 判断 $\mathbf{a} + \mathbf{ba} \stackrel{?}{=} (\mathbf{a} + \mathbf{b})\mathbf{a}$;
3. 因为 $aa \notin \mathbf{a} + \mathbf{ba}$ 而 $aa \in (\mathbf{a} + \mathbf{b})\mathbf{a}$;
4. 所以 $\mathbf{a} + \mathbf{ba} \neq (\mathbf{a} + \mathbf{b})\mathbf{a}$;
5. 即 $L + ML \neq (L + M)L$.

注意

这种方法仅限于判断正则表达式, 否则可能会发生错误.

例 14. 若用此方法判断 $L \cap M \cap N \stackrel{?}{=} L \cap M$, 以 $\mathbf{a}, \mathbf{b}, \mathbf{c}$ 替换 L, M, N, 有

$$\{a\} \cap \{b\} \cap \{c\} = \varnothing = \{a\} \cap \{b\},$$

而显然

$$L \cap M \cap N \neq L \cap M.$$

例.

- $(L+M)^* = (L^*M^*)^*$
- $(\varepsilon + L)^* = L^*$
- $L^* \stackrel{?}{=} L^*L^*$

  由 $\mathbf{a}^* = \mathbf{a}^*\mathbf{a}^*$ 得 $L^* = L^*L^*$ 成立.
- $(L+M)^*M \stackrel{?}{=} (L^*M)^*$

  替换得 $(\mathbf{a}+\mathbf{b})^*\mathbf{b} \stackrel{?}{=} (\mathbf{a}^*\mathbf{b})^*$, 因为 $\boldsymbol{\varepsilon} \notin (\mathbf{a}+\mathbf{b})^*\mathbf{b}$ 且 $\boldsymbol{\varepsilon} \in (\mathbf{a}^*\mathbf{b})^*$, 所以不相等.
- $(R+S)^* \stackrel{?}{=} R^* + S^*$
- $(RS+R)^*R \stackrel{?}{=} R(SR+R)^*$
- $(RS+R)^*RS \stackrel{?}{=} (RR^*S)^*$
- $(R+S)^*S \stackrel{?}{=} (R^*S)^*$

Chapter 4

正则语言的性质

4.1 证明语言的非正则性

“泵引理”是正则语言的一个必要条件: 如果一个语言是正则的, 则一定满足泵引理.

例 1. $L = \{0^m 1^n \mid m, n \geq 0\}$ 是否是正则语言?

例 2. $L = \{0^m 1^n \mid m \geq 2, n \geq 4\}$ 是否是正则语言?

例 3. $L_{01} = \{0^n 1^n \mid n \geq 0\}$ 是否是正则语言?

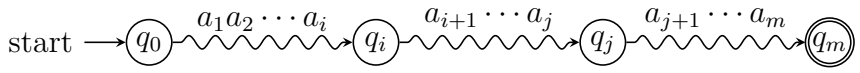
4.1.1 正则语言的泵引理

定理 5 (正则语言的泵引理 (Pumping Lemma))**.** *如果语言 L 是正则的, 那么存在正整数 N, 它只依赖于 L, 对 $\forall w \in L$, 只要 $|w| \geq N$, 就可以将 w 分为三部分 $w = xyz$ 满足:*

1. *$y \neq \varepsilon$ ($|y| > 0$);*
2. *$|xy| \leq N$;*
3. *$\forall k \geq 0$, $xy^k z \in L$.*

证明:

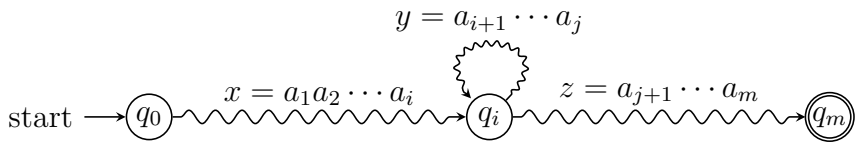
1. 如果 L 正则, 那么存在有 n 个状态 DFA A 使 $\mathbf{L}(A) = L$;
2. 取 $w = a_1 \dots a_m \in L$ ($m \geq n$), 定义 $q_i = \hat{\delta}(q_0, a_1 \dots a_i)$; q_0 是开始状态, 当 A 输入 w 的前 n 个字符时, 经过的状态分别是 $q_0, q_1, \cdots, q_n$ 共 $n+1$ 个;



3. 由鸽巢原理, 必有两状态相同 $q_i = q_j$ $(0 \le i < j \le n)$; 由 q_i 和 q_j 将 w 分为

$$x = a_1 a_2 \cdots a_i$$
$$y = a_{i+1} a_{i+2} \cdots a_j$$
$$z = a_{j+1} a_{j+2} \cdots a_m$$

4. 那么 $w = xyz$ 如图, 且有 $\forall k > 0,\ xy^k z \in L$;



因为如果从 q_i 出发, 输入 y, 会到达 q_j, 而 $q_i = q_j$, 所以当输入 $y^k (k \ge 0)$ 时, 始终会回到 q_i. 所以当 DFA A 输入 $xy^k z$ 时, 由 q_0 始终会达到 q_m. 那么, 如果 $xyz \in \mathbf{L}(A)$, 一定有 $xy^k z \in \mathbf{L}(A)$ 对所有 $k \ge 0$ 成立.

5. 而因为 $i < j$ 所以 $y \ne \varepsilon$ (即 $|y| > 0$), 因为 $j \le n$ 所以 $|xy| \le n$. □

任何从开始状态到接受状态的路径, 如果长度超过 n, 一定会经过 $n+1$ 个状态, 必定有一个重复状态, 因此会形成一个循环 (loop); 那么, 这个循环可以被重复多次后, 沿原路径还会到达接收状态. 泵引理中的 N, 是正则语言固有存在的.

泵引理可以用来确定特定语言不在给定语言类 (正则语言) 中. 但是它们不能被用来确定一个语言在给定类中, 因为满足引理是类成员关系的必要条件, 但不是充分条件.

4.1.2 泵引理的应用

续例 3. 证明 $L_{01} = \{0^n 1^n \mid n \ge 0\}$ 不是正则语言.

证明:

1. 假设 L_{01} 是正则的.
2. 那么, 存在 $N \in \mathbb{Z}^+$, 对 $\forall w \in L_{01}\ (|w| \ge N)$ 满足泵引理.
3. 从 L_{01} 中取 $w = 0^N 1^N$, 显然 $w \in L_{01}$ 且 $|w| = 2N \ge N$.
4. 那么, w 可被分为 $w = xyz$, 且 $|xy| \le N$ 和 $y \ne \varepsilon$.
5. 因此 y 只能是 0^m 且 $m > 0$.
6. 那么 $xy^2 z = 0^{N+m} 1^N \notin L_{01}$, 而由泵引理 $xy^2 z \in L_{01}$, 矛盾.
7. 所以假设不成立, L_{01} 不是正则的. □

例 4. 证明 $L_{\text{eq}} = \{w \in \{0,1\}^* \mid w \text{ 由数量相等的 0 和 1 构成}\}$ 不是正则的.

思考题

刚刚已经证明了

$$L_{01} = \{0^n 1^n \mid n \geq 0\}$$

不是正则语言, 那么能否使用

$$L_{01} \subseteq L_{\text{eq}}$$

来说明 L_{eq} 也不是正则的呢?

证明:

1. 假设 L_{eq} 是正则的.
2. 那么, 存在 $N \in \mathbb{Z}^+$, 对 $\forall w \in L_{\text{eq}} (|w| \geq N)$ 满足泵引理.
3. 从 L_{eq} 中取 $w = 0^N 1^N$, 显然 $w \in L_{\text{eq}}$ 且 $|w| = 2N \geq N$.
4. 那么, w 可被分为 $w = xyz$, 且 $|xy| \leq N$ 和 $y \neq \varepsilon$.
5. 因此 y 只能是 0^m 且 $m > 0$.
6. 那么 $xy^2z = 0^{N+m}1^N \notin L_{\text{eq}}$, 而由泵引理 $xy^2z \in L_{\text{eq}}$, 矛盾.
7. 所以假设不成立, L_{eq} 不是正则的. □

例 5. 证明 $L = \{0^i 1^j \mid i > j\}$ 不是正则的.

证明:

1. 假设 L 是正则的.
2. 那么, 存在 $N \in \mathbb{Z}^+$, 对 $\forall w \in L (|w| \geq N)$ 满足泵引理.
3. 从 L 中取 $w = 0^{N+1}1^N$, 则 $w \in L$ 且 $|w| = 2N + 1 \geq N$.
4. 由泵引理, w 可被分为 $w = xyz$, 且 $|xy| \leq N$ 和 $y \neq \varepsilon$.
5. 那么, y 只能是 0^m 且 $m \geq 1$.
6. 那么, $xz = xy^0z = 0^{N+1-m}1^N \notin L$, 因为 $N + 1 - m \leq N$, 而由泵引理 $xy^0z \in L$, 矛盾.
7. 所以假设不成立, L 不是正则的. □

例 6. Prove $L = \{a^3b^nc^{n-3} \mid n \geq 3\}$ is not regular.

证明:

1. 假设 L 是正则的.
2. 那么, 存在 $N \in \mathbb{Z}^+$, 对 $\forall w \in L\,(|w| \geq N)$ 满足泵引理.
3. 从 L 中取 $w = a^3b^Nc^{N-3}$, 则 $w \in L$ 且 $|w| = 2N \geq N$.
4. 由泵引理, w 可被分为 $w = xyz$, 且 $|xy| \leq N$ 和 $y \neq \varepsilon$.
5. 那么, 则 y 只可能有 3 种情况 $(m > 0, r > 0, s > 0)$:
   (a) $y = a^m$, 则 $xy^2z = a^{3+m}b^Nc^{N-3} \notin L$;
   (b) $y = b^m$, 则 $xy^2z = a^3b^{N+m}c^{N-3} \notin L$;
   (c) $y = a^rb^s$, 则 $xy^2z = a^3b^sa^rb^Nc^{N-3} \notin L$.
6. 无论 y 为何种情况, xy^2z 都不可能在 L 中, 与泵引理矛盾.
7. 所以假设不成立, L 不是正则的. □

例. 证明 $L = \{a^{n!} \mid n > 0\}$ 不是正则的.

... 取 $w = a^{N!}$, 那么 $|y| = m > 0$, $|xy^2z| = N! + m$, 而 $0 < m \leq N < N! < N \cdot N!$, 所以 $N! < |xy^2z| = N! + m < N! + N \cdot N! = (N+1)!$, 即 $|xy^2z|$ 在两个阶乘数之间, 不可能是阶乘数, ...

思考题

- $L = \{0^n1^n \mid 0 \leq n \leq 100\}$ 是否是正则语言?
- 有限的语言, 是否符合泵引理呢? 如 $\varnothing$, $\{\varepsilon\}$, $\{0, 00\}$ 等.

4.1.3 泵引理只是必要条件

即“正则 $\Rightarrow$ 泵引理成立”, 所以“$\neg$ 泵引理成立 $\Rightarrow \neg$ 正则”.

- 泵引理只是正则语言的*必要条件*
- 只能用来证明某个语言不是正则的
- 与正则语言等价的定理 — Myhill-Nerode Theorem

例 7. 语言 L 不是正则的, 但每个串都可以应用泵引理

$$L = \{ca^n b^n \mid n \geq 1\} \cup \{c^k w \mid k \neq 1, w \in \{a, b\}^*\}$$

- 其中 $A = \{ca^n b^n \mid n \geq 1\}$ 部分不是正则的
- 而 $B = \{c^k w \mid k \neq 1, w \in \{a, b\}^*\}$ 部分是正则的
- 而 A 的任何串 $w = ca^i b^i$, 都可应用泵引理, 因为

  $$w = (\varepsilon)(c)(a^i b^i)$$

  重复字符 c 生成的新串都会落入 B 中

思考题

对任何正则语言 L, 在泵引理中, 与 L 相关联的正整数 N

- 与识别 L 的 DFA 的状态数 n 之间有何关系?
- 与识别 L 的 NFA 的状态数之间呢?

思考题

语言

$$L = \{0^n x 1^n \mid n \geq 1, x \in \{0, 1\}^*\}$$

是否是正则语言?

4.2 正则语言的封闭性

定义. 正则语言经某些运算后得到的新语言仍保持正则, 称为在这些运算下封闭.

4.2.1 并/连接/闭包

定理 6 (并/连接/闭包的封闭性)**.** 正则语言在并, 连接和闭包运算下保持封闭.

证明: 由正则表达式的定义得证. □

4.2.2 补

定理 7 (补运算封闭性)**.** *如果 L 是 Σ 上的正则语言, 那么 $\overline{L} = \Sigma^* - L$ 也是正则的.*

证明: 设接受语言 L 的 DFA

$$A = (Q, \Sigma, \delta, q_0, F)$$

即 $\mathbf{L}(A) = L$. 构造 DFA

$$B = (Q, \Sigma, \delta, q_0, Q - F)$$

则有 $\overline{L} = \mathbf{L}(B)$, 因为 $\forall w \in \Sigma^*$

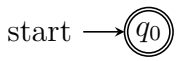
$$w \in \overline{L} \iff \hat{\delta}(q_0, w) \notin F \iff \hat{\delta}(q_0, w) \in Q - F \iff w \in \mathbf{L}(B).$$

□

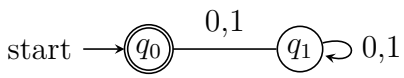
注意

使用这种方法求正则语言的补时, DFA 不能有缺失状态.

例 8. 若 $\Sigma = \{0, 1\}$, $L = \{\varepsilon\}$ 的 DFA 如图, 请给出 $\overline{L}$ 的 DFA.



应使用完整的 DFA 去求补:



思考题

如何求正则表达式的补?

例 9. 证明 $L_{\text{neq}} = \{w \mid w$ 由数量不相等的 0 和 1 构成 $\}$ 不是正则的.

证明:

- 由泵引理不易直接证明 L_{neq} 不是正则的;
- 因为无论如何取 w, 将其分为 $w = xyz$ 时, 都不易产生 L_{neq} 之外的串;
- 而证明 L_{eq} 非正则很容易;
- 由补运算的封闭性, 所以 $L_{\text{neq}} = \overline{L_{\text{eq}}}$ 也不是正则的. □

4.2.3 交

定理 8. *[习题 4.2.15]* 若 DFA A_L, A_M 和 A 的定义如下

$$\begin{aligned} A_L &= (Q_L, \Sigma, \delta_L, q_L, F_L) \\ A_M &= (Q_M, \Sigma, \delta_M, q_M, F_M) \\ A &= (Q_L \times Q_M, \Sigma, \delta, (q_L, q_M), F_L \times F_M) \end{aligned}$$

其中

$$\begin{aligned} \delta : (Q_L \times Q_M) \times \Sigma &\to Q_L \times Q_M \\ \delta((p,q),a) &= (\delta_L(p,a), \delta_M(q,a)). \end{aligned}$$

则对任意 $w \in \Sigma^*$,

$$\hat{\delta}((q_L, q_M), w) = (\hat{\delta}(q_L, w), \hat{\delta}(q_M, w)).$$

证明: 对 w 的结构归纳.

归纳基础：当 $w = \varepsilon$ 时

$$\begin{aligned} \hat{\delta}((q_L,q_M),\varepsilon) &= (q_L, q_M) && \hat{\delta}\text{的定义} \\ &= (\hat{\delta}_L(q_L,\varepsilon), \hat{\delta}_M(q_M,\varepsilon)) && \text{同理} \end{aligned}$$

归纳递推：当 $w = xa$ 时

$$\begin{aligned} \hat{\delta}((q_L,q_M),xa) &= \delta(\hat{\delta}((q_L,q_M),x),a) && \hat{\delta}\text{的定义} \\ &= \delta((\hat{\delta}(q_L,x), \hat{\delta}(q_M,x)),a) && \text{归纳假设} \\ &= (\delta_L(\hat{\delta}_L(q_L,x),a), \delta_M(\hat{\delta}_M(q_M,x),a)) && \delta\text{的构造} \\ &= (\hat{\delta}_L(q_L,xa), \hat{\delta}_M(q_M,xa)) && \hat{\delta}\text{的定义} \end{aligned}$$

□

定理 9 (交运算封闭性)**.** 如果 L 和 M 是正则语言, 那么 $L \cap M$ 也是正则语言.

证明 1: 由 $L \cap M = \overline{\overline{L} \cup \overline{M}}$ 得证. □

证明 2: 由定理 8 构造识别 $L \cap M$ 的 DFA A, 则 $\forall w \in \Sigma^*$,

$$\begin{aligned} w \in L \cap M &\iff \hat{\delta}_L(q_L,w) \in F_L \wedge \hat{\delta}_M(q_M,w) \in F_M \\ &\iff (\hat{\delta}_L(q_L,w), \hat{\delta}_M(q_M,w)) \in F_L \times F_M \\ &\iff \hat{\delta}((q_L,q_M),w) \in F_L \times F_M \\ &\iff w \in \mathbf{L}(A). \end{aligned}$$

因此 $\mathbf{L}(A) = L \cap M$, 所以 $L \cap M$ 也是正则的. □

例 10. 如果已知语言

$$L_{01} = \{0^n 1^n \mid n \geq 0\}$$

不是正则的, 请用封闭性证明语言

$$L_{\text{eq}} = \{w \in \{0,1\}^* \mid w \text{ 由数量相等的 0 和 1 构成}\}$$

也不是正则的.

证明:

1. 首先, 因为 $\mathbf{0}^*\mathbf{1}^*$ 是正则语言;
2. 而 $L_{01} = \mathbf{L}(\mathbf{0}^*\mathbf{1}^*) \cap L_{\text{eq}}$;
3. 如果 L_{eq} 是正则的, L_{01} 必然也是正则的;
4. 因为已知 L_{01} 不是正则的, 所以 L_{eq} 一定不是正则的. □

思考题

为什么又能用 L_{eq} 的子集 L_{01} 是非正则的, 来证明 L_{eq} 是非正则的呢?

例 11. 如果 L_1 和 L_2 都不是正则的, 那么 $L_1 \cap L_2$ 一定不是正则的吗?

4.2.4 差

定理 10 (差运算封闭性)**.** *如果 L 和 M 都是正则语言, 那么 $L - M$ 也是正则的.*

证明: $L - M = L \cap \overline{M}$. □

例 12. [习题 4.2.6 a)] 证明正则语言在以下运算下封闭

$$\min(L) = \{w \mid w \text{ is in } L, \text{but no proper prefix of } w \text{ is in } L\}$$

证明 1: 设 L 的 DFA 为 $A = (Q, \Sigma, \delta, q_0, F)$, 构造 $\min(L)$ 的 DFA $B = (Q, \Sigma, \delta', q_0, F)$ 其中 δ' 如下, 往证 $L(B) = \min(L)$:

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{if } q \notin F \\ \varnothing & \text{if } q \in F \end{cases}$$

1. $\forall w \in L(B)$, 存在转移序列 $q_0q_1\cdots q_n \in F$ 使 B 接受 w, 其中 $q_i \notin F\,(0 \le i \le n-1)$. $\therefore w \in \min(L)$.

2. $\forall w \in \min(L)$, 有 $w \in L$, A 接受 w 的状态序列为如果 $q_0q_1\cdots q_n \in F$, 则显然 $q_i \notin F\,(0 \le i \le n-1)$, 否则 w 会有 L 可接受的前缀. $\therefore w \in L(B)$ □

证明 2:

由封闭性

$$\min(L) = L - L\Sigma^+,$$

得证. □

4.2.5 反转

定义. 字符串 $w = a_1a_2\ldots a_n$ 的反转 *(Reverse)*, 记为 w^R, 定义为

$$w^R = a_na_{n-1}\ldots a_1.$$

定义. 语言 L 的反转, 记为 L^R, 定义为

$$L^R = \{w^R \in \Sigma^* \mid w \in L\}.$$

定理 11 (反转的封闭性)**.** 如果 L 是正则语言, 那么 L^R 也是正则的.

两种证明方法:

- 对正则表达式 E 的结构归纳, 往证

$$\mathbf{L}(E^R) = (\mathbf{L}(E))^R.$$

- 构造识别 L 的 NFA $A = (Q, \Sigma, \delta_A, q_0, F)$, 将其转换为识别 L^R 的 NFA

$$B = (Q, \Sigma, \delta_B, q_s, \{q_0\})$$

  1. 将 A 的边调转方向;
  2. 将 A 的初始状态 q_0, 改为唯一的接受状态;
  3. 新增初始状态 q_s, 且令 $\delta_B(q_s, \varepsilon) = F$.

例 13. 语言 L 及其反转 L^R 分别为

$$L = \{w \in \{0,1\}^* \mid w \text{ ends in } 01.\}$$

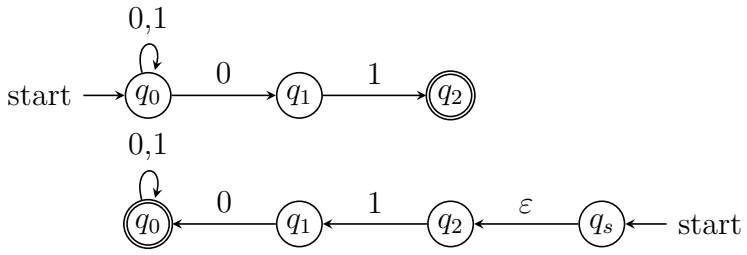
$$L^R = \{w \in \{0,1\}^* \mid w \text{ starts with } 10.\}$$

正则表达式分别为

$$L = (\mathbf{0} + \mathbf{1})^*\mathbf{01}$$

$$L^R = \mathbf{10}(\mathbf{0} + \mathbf{1})^*.$$

自动机分别为



证明: 往证如果有正则表达式 E, 则存在正则表达式 E^R 使

$$\mathbf{L}(E^R) = (\mathbf{L}(E))^R.$$

归纳基础:

1. 当 $E = \varnothing$ 时, 有 $\varnothing^R = \varnothing$;
2. 当 $E = \boldsymbol{\varepsilon}$ 时, 有 $\boldsymbol{\varepsilon}^R = \boldsymbol{\varepsilon}$;
3. $\forall a \in \Sigma$, 当 $E = \mathbf{a}$ 时, 有 $\mathbf{a}^R = \mathbf{a}$;

都满足 $\mathbf{L}(E^R) = (\mathbf{L}(E))^R$, 因此命题成立.

归纳递推:

1. 当 $E = E_1 + E_2$ 时, 有 $(E_1 + E_2)^R = E_1^R + E_2^R$

$$\begin{aligned}
&(\mathbf{L}(E_1 + E_2))^R \\
&= (\mathbf{L}(E_1) \cup \mathbf{L}(E_2))^R && \text{正则表达式的加} \\
&= \{w^R \mid w \in \mathbf{L}(E_1) \cup w \in \mathbf{L}(E_2)\} && \text{语言的反转} \\
&= (\mathbf{L}(E_1))^R \cup (\mathbf{L}(E_2))^R && \text{同上} \\
&= \mathbf{L}(E_1^R) \cup \mathbf{L}(E_2^R) && \text{归纳假设} \\
&= \mathbf{L}(E_1^R + E_2^R) && \text{正则表达式的加}
\end{aligned}$$

2. 当 $E = E_1E_2$ 时, 有 $(E_1E_2)^R = E_2^RE_1^R$

$$(\mathbf{L}(E_1E_2))^R = (\mathbf{L}(E_1)\mathbf{L}(E_2))^R \qquad \text{正则表达式的连接}$$

$$
\begin{aligned}
&= \{w_1 w_2 \mid w_1 \in \mathbf{L}(E_1), w_2 \in \mathbf{L}(E_2)\}^R && \text{语言的连接}\\
&= \{(w_1 w_2)^R \mid w_1 \in \mathbf{L}(E_1), w_2 \in \mathbf{L}(E_2)\} && \text{语言的反转}\\
&= \{w_2^R w_1^R \mid w_1 \in \mathbf{L}(E_1), w_2 \in \mathbf{L}(E_2)\} && \text{字符串的反转}\\
&= \{w_2^R \mid w_2 \in \mathbf{L}(E_2)\}\{w_1^R \mid w_1 \in \mathbf{L}(E_1)\} && \text{语言的连接}\\
&= (\mathbf{L}(E_2))^R(\mathbf{L}(E_1))^R && \text{语言的反转}\\
&= \mathbf{L}(E_2^R)\mathbf{L}(E_1^R) = \mathbf{L}(E_2^R E_1^R) && \text{正则表达式的连接}
\end{aligned}
$$

3. 当 $E = E_1^*$ 时, 有 $(E_1^*)^R = (E_1^R)^*$

$$
\begin{aligned}
&(\mathbf{L}(E_1^*))^R\\
&= \{w_1 w_2 \dots w_n \mid n \geq 0, w_i \in \mathbf{L}(E_1)\}^R && \text{正则表达式的闭包}\\
&= \{(w_1 w_2 \dots w_n)^R \mid n \geq 0, w_i \in \mathbf{L}(E_1)\} && \text{语言的反转}\\
&= \{w_n^R w_{n-1}^R \dots w_1^R \mid n \geq 0, w_i \in \mathbf{L}(E_1)\} && \text{字符串的反转}\\
&= \{w_n^R w_{n-1}^R \dots w_1^R \mid n \geq 0, w_i^R \in \mathbf{L}(E_1^R)\} && \text{归纳假设}\\
&= \{w_1 w_2 \dots w_n \mid n \geq 0, w_i \in \mathbf{L}(E_1^R)\} && \text{变量重命名}\\
&= \mathbf{L}((E_1^R)^*) && \text{正则表达式的闭包}
\end{aligned}
$$

都满足 $(\mathbf{L}(E))^R = \mathbf{L}(E^R)$, 因此命题成立, 所以 L^R 也是正则语言. □

4.2.6 同态与逆同态

同态 (Homomorphism)

定义. 若 Σ 和 Γ 是两个字母表, 同态定义为函数 $h : \Sigma \to \Gamma^*$

$$\forall a \in \Sigma,\ h(a) \in \Gamma^*.$$

扩展 h 的定义到字符串,

$$(1)\quad h(\varepsilon) = \varepsilon$$
$$(2)\quad h(xa) = h(x)h(a)$$

再扩展 h 到语言, 对 $\forall L \subseteq \Sigma^*$,

$$h(L) = \{h(w) \mid w \in L\}.$$

例 14. 若由 $\Sigma = \{0, 1\}$ 到 $\Gamma = \{a, b\}$ 的同态函数 h 为

$$h(0) = ab,\quad h(1) = \varepsilon.$$

则 Σ 上的字符串 0011, 在 h 的作用下

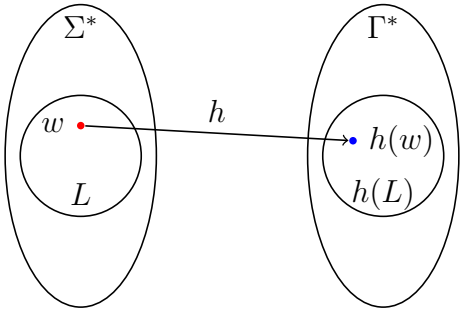
$$h(0011) = h(\varepsilon)h(0)h(0)h(1)h(1)$$

$$= \varepsilon \cdot ab \cdot ab \cdot \varepsilon \cdot \varepsilon$$
$$= abab.$$

语言 $L = \mathbf{1}^*\mathbf{0} + \mathbf{0}^*\mathbf{1}$, 在 h 的作用下, $h(L)$ 为:

$$\begin{aligned} h(\mathbf{1}^*\mathbf{0} + \mathbf{0}^*\mathbf{1}) &= (h(\mathbf{1}))^* h(\mathbf{0}) + (h(\mathbf{0}))^* h(\mathbf{1}) \\ &= (\boldsymbol{\varepsilon})^* (\mathbf{ab}) + (\mathbf{ab})^* (\boldsymbol{\varepsilon}) \\ &= (\mathbf{ab})^* \end{aligned}$$

定理 12 (同态的封闭性)**.** *若 L 是字母表 Σ 上的正则语言, h 是 Σ 上的同态, 则 $h(L)$ 也是正则的.*



- 若 L 的正则表达式为 E, 即 $L = \mathbf{L}(E)$, 按如下规则构造表达式 $h(E)$

$$\begin{aligned} h(\varnothing) &= \varnothing & h(\mathbf{r} + \mathbf{s}) &= h(\mathbf{r}) + h(\mathbf{s}) \\ h(\boldsymbol{\varepsilon}) &= \boldsymbol{\varepsilon} & h(\mathbf{rs}) &= h(\mathbf{r})h(\mathbf{s}) \\ \forall a \in \Sigma,\ h(\mathbf{a}) &= h(a) & h(\mathbf{r}^*) &= (h(\mathbf{r}))^* \end{aligned}$$

- 往证 $\mathbf{L}(h(E)) = h(\mathbf{L}(E))$, 而 $h(E)$ 显然也是正则表达式, 因此 $h(L)$ 正则

证明: 对 E 的结构归纳, 往证 $\mathbf{L}(h(E)) = h(\mathbf{L}(E))$.

归纳基础:

- 当 $E = \boldsymbol{\varepsilon}$ 时
$$h(\mathbf{L}(\boldsymbol{\varepsilon})) = h(\{\varepsilon\}) = \{\varepsilon\} = \mathbf{L}(\boldsymbol{\varepsilon}) = \mathbf{L}(h(\boldsymbol{\varepsilon}))$$
- 当 $E = \varnothing$ 时
$$h(\mathbf{L}(\varnothing)) = h(\varnothing) = \varnothing = \mathbf{L}(\varnothing) = \mathbf{L}(h(\varnothing))$$
- $\forall a \in \Sigma$, 当 $E = \mathbf{a}$ 时
$$h(\mathbf{L}(\mathbf{a})) = h(\{a\}) = \{h(a)\} = \mathbf{L}(h(a)) = \mathbf{L}(h(\mathbf{a}))$$

所以命题成立.

归纳递推: 假设对正则表达式 F, G 分别有

$$\mathbf{L}(h(F)) = h(\mathbf{L}(F)), \quad \mathbf{L}(h(G)) = h(\mathbf{L}(G))$$

- 当 $E = F + G$ 时:

$$\begin{aligned}
h(\mathbf{L}(F+G)) &= h(\mathbf{L}(F) \cup \mathbf{L}(G)) && \text{正则表达式的加} \\
&= h(\mathbf{L}(F)) \cup h(\mathbf{L}(G)) && h \text{ 作用在每个集合的串上} \\
&= \mathbf{L}(h(F)) \cup \mathbf{L}(h(G)) && \text{归纳假设} \\
&= \mathbf{L}(h(F) + h(G)) && \text{正则表达式的加} \\
&= \mathbf{L}(h(F+G)) && h(F+G) \text{ 的定义}
\end{aligned}$$

- 当 $E = FG$ 时:

$$\begin{aligned}
h(\mathbf{L}(E)) &= h(\mathbf{L}(F)\mathbf{L}(G)) && \text{正则表达式的连接} \\
&= h(\mathbf{L}(F))h(\mathbf{L}(G)) && \heartsuit \\
&= \mathbf{L}(h(F))\mathbf{L}(h(G)) && \text{归纳假设} \\
&= \mathbf{L}(h(F)h(G)) && \text{正则表达式的连接} \\
&= \mathbf{L}(h(FG)) && h(FG) \text{ 的定义} \\
&= \mathbf{L}(h(E))
\end{aligned}$$

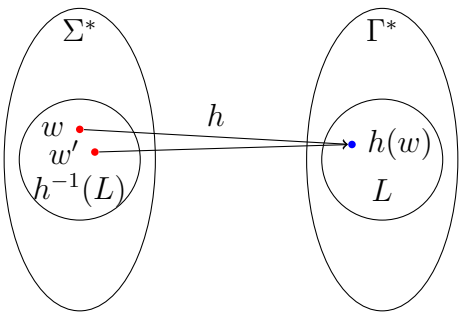
$\heartsuit$: $h(a_1 \cdots a_n b_1 \cdots b_m) = h(a_1) \cdots h(b_m) = h(a_1 \cdots a_n)h(b_1 \cdots b_m)$

- 当 $E = F^*$ 时: 略 (提示: $\forall w \in \mathbf{L}(F^*)$ 可看作 $w = w_1 w_2 \cdots w_n$, 其中 $w_i \in \mathbf{L}(F)$.) □

逆同态 (Inverse homomorphism)

定义. *若 h 是字母表 Σ 到 Γ 的同态, 且 L 是 Γ 上的语言, 那么使 $h(w) \in L$ 的 w $(w \in \Sigma^*)$ 的集合, 称为语言 L 的 h 逆, 记为 $h^{-1}(L)$, 即*

$$h^{-1}(L) = \{w \in \Sigma^* \mid h(w) \in L\}.$$



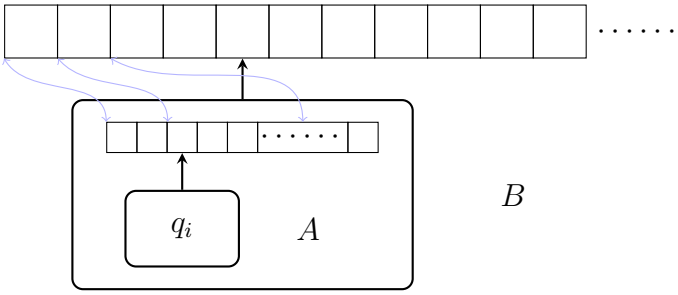
定理 13 (逆同态的封闭性)**.** 如果 h 是字母表 Σ 到 Γ 的同态, L 是 Γ 上的正则语言, 那么 $h^{-1}(L)$ 也是正则语言.

证明: 由 L 的 DFA $A = (Q, \Gamma, \delta, q_0, F)$, 构造识别 $h^{-1}(L)$ 的 DFA

$$B = (Q, \Sigma, \delta', q_0, F),$$

其中

$$\delta'(q, a) = \hat{\delta}(q, h(a)).$$



为证明 $\mathbf{L}(B) = h^{-1}(L)$, 先证明 $\widehat{\delta'}(q, w) = \hat{\delta}(q, h(w))$.

对 $|w|$ 归纳, 往证 $\widehat{\delta'}(q, w) = \hat{\delta}(q, h(w))$.

1. 归纳基础: 若 $w = \varepsilon$

$$\hat{\delta}(q, h(\varepsilon)) = \hat{\delta}(q, \varepsilon) = q = \widehat{\delta'}(q, \varepsilon),$$

2. 归纳递推: 若 $w = xa$

$$\begin{aligned}
\widehat{\delta'}(q, xa) &= \delta'(\widehat{\delta'}(q, x), a) && \widehat{\delta'} \text{ 定义} \\
&= \delta'(\hat{\delta}(q, h(x)), a) && \text{归纳假设} \\
&= \hat{\delta}(\hat{\delta}(q, h(x)), h(a)) && \delta' \text{ 构造} \\
&= \hat{\delta}(q, h(x)h(a)) && \text{DFA 节例 5} \\
&= \hat{\delta}(q, h(xa)).
\end{aligned}$$

所以 $\forall w \in \Sigma^*$, $\widehat{\delta'}(q_0, w) = \hat{\delta}(q_0, h(w)) \in F$, 即 w 被 B 接受当且仅当 $h(w)$ 被 A 接受, B 是识别 $h^{-1}(L)$ 的 DFA, 因此 $h^{-1}(L)$ 是正则的. □

例 15. Prove that $L = \{0^n 1^{2n} \mid n \geq 0\}$ is a language not regular.

证明: 设同态 $h : \{0, 1\} \to \{0, 1\}^*$ 为

$$h(0) = 0,$$

$$h(1) = 11,$$

那么

$$h^{-1}(L) = \{0^n 1^n \mid n \geq 0\} = L_{01},$$

我们已知 L_{01} 非正则, 由封闭性, L 不是正则的. □

例 16. 若语言 $L = (\mathbf{00} + \mathbf{1})^*$, 同态 $h : \{a, b\} \to \{0, 1\}^*$ 为

$$h(a) = 01, \quad h(b) = 10,$$

请证明 $h^{-1}(L) = (\mathbf{ba})^*$.

证明: 往证 $h(w) \in L \iff w = (ba)^n$.

($\Leftarrow$) 若 $w = (ba)^n$, 而 $h(ba) = 1001$, 因此 $h(w) = (1001)^n \in L$.

($\Rightarrow$) 若 $h(w) \in L$, 假设 $w \notin (\mathbf{ba})^*$, 则只能有四种情况:

1. w 以 a 开头, 则 $h(w)$ 以 01 开头, 显然 $h(w) \notin (\mathbf{00} + \mathbf{1})^*$;
2. w 以 b 结尾, 则 $h(w)$ 以 10 结尾, 显然 $h(w) \notin (\mathbf{00} + \mathbf{1})^*$;
3. w 有连续的 a, 即 $w = xaay$, 则 $h(w) = z1010v$, 显然 $h(w) \notin (\mathbf{00} + \mathbf{1})^*$;
4. w 有连续的 b, 即 $w = xbby$, 则 $h(w) = z0101v$, 显然 $h(w) \notin (\mathbf{00} + \mathbf{1})^*$;

因此 w 只能是 $(ba)^n, n \geq 0$ 的形式. □

例 17. For a language L, define head(L) to be the set of all prefixes of strings in L. Prove that if L is regular, so is head(L).

证明. 设 L 是 Σ 上的正则语言且 $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, a, b\}$. 定义同态 $h : \Gamma \to \Sigma^*$ 和 $g : \Gamma \to \Sigma^*$ 分别为:

$$\begin{array}{llll} h(0) = 0 & h(a) = 0 & g(0) = 0 & g(a) = \varepsilon \\ h(1) = 1 & h(b) = 1 & g(1) = 1 & g(b) = \varepsilon \end{array}$$

则因为 $(\mathbf{0} + \mathbf{1})^*(\mathbf{a} + \mathbf{b})^*$ 是 Γ 上的正则语言, 所以

$$(\mathbf{0} + \mathbf{1})^*(\mathbf{a} + \mathbf{b})^* \cap h^{-1}(L)$$

是 Γ 上的正则语言, 所以

$$\text{head}(L) = g((\mathbf{0} + \mathbf{1})^*(\mathbf{a} + \mathbf{b})^* \cap h^{-1}(L))$$

是 Σ 上的正则语言, 因此 head(L) 是正则的. □

4.3 正则语言的判定性质

正则语言, 或任何语言, 典型的 3 个判定问题:

1. 以某种形式化模型描述的语言是否为空? 是否无穷?
2. 某个特定的串 w 是否属于所描述的语言?
3. 以两种方式描述的语言, 是否是相同的? — 语言的等价性

我们想知道, 要回答这类问题的具体*算法*, *是否存在*.

4.3.1 空性, 有穷性和无穷性

正则语言的空, 有穷和无穷 (Emptiness, finiteness and infiniteness), 可以通过定理14来判定.

定理 14. *具有 n 个状态的有穷自动机 M 接受的集合 S:*

1. *S 是非空的, 当且仅当 M 接受某个长度小于 n 的串;*
2. *S 是无穷的, 当且仅当 M 接受某个长度为 m 的串, $n \leq m < 2n$.*

所以，对于正则语言:

- 存在算法, 判断其是否为空, 只需检查全部长度小于 n 的串;
- 存在算法, 判断其是否无穷, 只需检查全部长度由 n 到 $2n-1$ 的串.

证明: 设接受正则语言 S 的 DFA 为 A.

1. 必要性: 显然成立. 充分性:
   - i 如果 S 非空, 设 w 是 A 接受的串中长度最小者之一;
   - ii 必然 $|w| < n$, 否则由泵引理 $w = xyz$, 接受 xz 更短.
2. 必要性: 由泵引理, 显然成立. 充分性:
   - i 如果 S 无穷, 假设没有长度 n 到 $2n-1$ 之间的串;
   - ii 那么取 $w \in \mathbf{L}(A)$ 是长度 $\geq 2n$ 中最小者之一;
   - iii 由泵引理 $w = xyz$, 且 A 会接受更短的串 xz;
   - iv 于是, 或者 w 不是长度最小的, 或者长度 n 到 $2n-1$ 之间有被接受的串, 因此假设不成立. □

4.3.2 等价性

定理 15. *存在算法, 判定两个有穷自动机是否等价 (接受语言相同).*

证明:

1. 设 M_1 和 M_2 是分别接受 L_1 和 L_2 的有穷自动机;
2. 则 $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$ 是正则的, 所以可被某个有穷自动机 M_3 接受;
3. 而 M_3 接受某个串, 当且仅当 $L_1 \neq L_2$;
4. 由于存在算法判断 $\mathbf{L}(M_3)$ 是否为空, 因此得证. □

4.4 自动机的最小化

4.4.1 DFA 状态的等价性

定义. *DFA $A = (Q, \Sigma, \delta, q_0, F)$ 中两个状态 p 和 q, 对 $\forall w \in \Sigma^*$:*

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

则称这两个状态是等价的, 否则称为可区分的.

- 等价性只要求 $\hat{\delta}(p, w)$ 和 $\hat{\delta}(q, w)$ 同时在或不在 F 中, 而不必是相同状态.

4.4.2 填表算法与 DFA 最小化

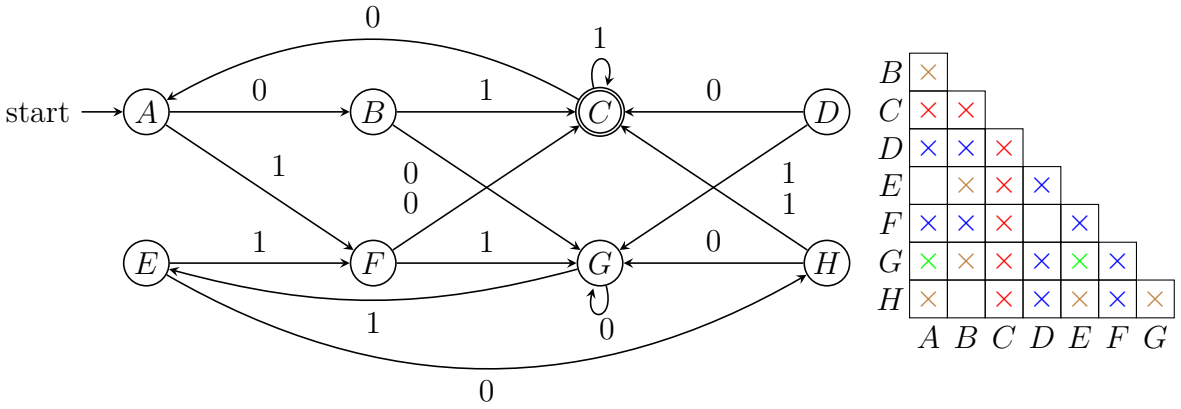
填表算法

递归寻找 DFA 中全部的可区分状态对:

1. 如果 $p \in F$ 而 $q \notin F$, 则 $[p, q]$ 是可区分的;
2. $\exists a \in \Sigma$, 如果
$$[r = \delta(p, a), s = \delta(q, a)]$$
是可区分的, 则 $[p, q]$ 是可区分的.

定理 16. *如果填表算法不能区分两个状态, 则这两个状态是等价的.*

例 18. 用填表算法找到如图 DFA 中全部可区分状态对.



1. 直接标记终态和非终态之间的状态对:

$$\{C\}\times\{A,B,D,E,F,G,H\}.$$

2. 标记所有经过字符 0 到达终态和非终态的状态对:

$$\{D,F\}\times\{A,B,C,E,G,H\}.$$

3. 标记所有经过字符 1 到达终态和非终态的状态对:

$$\{B,H\}\times\{A,C,D,E,F,G\}.$$

4. 此时还有 $[A,E]$, $[A,G]$, $[B,H]$, $[D,F]$, $[E,G]$ 未标记, 只需逐个检查.

   × $[A,G]$ 是可区分的, 因为经串 01 到可区分的 $[C,E]$;

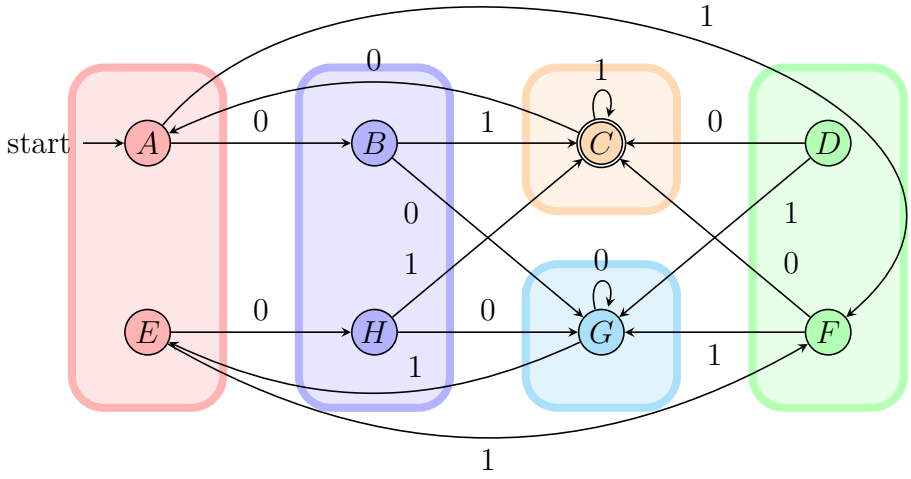
   × $[E,G]$ 是可区分的, 因为经串 10 到可区分的 $[C,H]$.

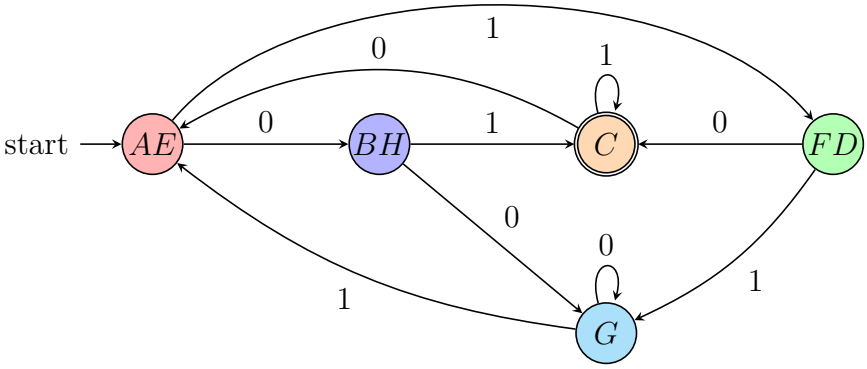
5. 而 $[A,E]$, $[B,H]$ 和 $[D,F]$ 在经过很短的字符串后, 都会到达相同状态, 因此都是等价的.

DFA 最小化

根据等价状态, 将状态集划分成块, 构造等价的最小化 DFA. 根据填表算法取得的 DFA A 状态间的等价性, 将状态集进行划分, 得到不同的块; 利用块构造新的 DFA B, B 的开始状态的为包含 A 初始状态的块, B 的接受状态为包含 A 的接收状态的块, 转移函数为块之间的转移; 则 B 是 A 的最小化 DFA.

续例 18. 构造其最小化的 DFA.





思考题

NFA 能否最小化?

Chapter 5

上下文无关文法

5.1 上下文无关文法

上下文无关文法在程序设计语言的设计, 编译器的实现等方面有重要应用, 也应用在可扩展标记语言 (XML) 的格式类型定义 (DTD) 中等. 上下文无关文法重要的原因, 在于它们拥有足够强的表达能力而又足够简单, 使得我们可以设计有效的分析算法来检验一个给定字串是否是由某个上下文无关文法产生的. (如 LR 分析器和 LL 分析器)

自然语言的文法

$$
\begin{aligned}
\langle \textit{sentence} \rangle &\to \langle \textit{noun-phrase} \rangle \langle \textit{verb-phrase} \rangle \\
\langle \textit{noun-phrase} \rangle &\to \langle \textit{article} \rangle \langle \textit{noun} \rangle \mid \langle \textit{article} \rangle \langle \textit{adjective} \rangle \langle \textit{noun} \rangle \\
\langle \textit{verb-phrase} \rangle &\to \langle \textit{verb} \rangle \mid \langle \textit{verb} \rangle \langle \textit{noun-phrase} \rangle \\
\langle \textit{article} \rangle &\to \texttt{a} \mid \texttt{the} \\
\langle \textit{noun} \rangle &\to \texttt{boy} \mid \texttt{girl} \mid \texttt{cat} \\
\langle \textit{adjective} \rangle &\to \texttt{big} \mid \texttt{small} \mid \texttt{blue} \\
\langle \textit{verb} \rangle &\to \texttt{sees} \mid \texttt{likes} \\
&\cdots
\end{aligned}
$$

使用文法规则产生句子:

$$
\begin{aligned}
\langle \textit{sentence} \rangle &\Rightarrow \langle \textit{noun-phrase} \rangle \langle \textit{verb-phrase} \rangle \\
&\Rightarrow \langle \textit{article} \rangle \langle \textit{noun} \rangle \langle \textit{verb-phrase} \rangle \\
&\Rightarrow \langle \textit{article} \rangle \langle \textit{noun} \rangle \langle \textit{verb} \rangle \langle \textit{noun-phrase} \rangle \\
&\Rightarrow \langle \textit{article} \rangle \langle \textit{noun} \rangle \langle \textit{verb} \rangle \langle \textit{article} \rangle \langle \textit{adjective} \rangle \langle \textit{noun} \rangle \\
&\Rightarrow \texttt{the } \langle \textit{noun} \rangle \langle \textit{verb} \rangle \langle \textit{article} \rangle \langle \textit{adjective} \rangle \langle \textit{noun} \rangle \\
&\Rightarrow \texttt{the girl } \langle \textit{verb} \rangle \langle \textit{article} \rangle \langle \textit{adjective} \rangle \langle \textit{noun} \rangle \\
&\Rightarrow \cdots \\
&\Rightarrow \texttt{the girl sees a blue cat}
\end{aligned}
$$

如果把式子中表示定义的符号 $\to$ 写为 ::= 时, 也称为 BNF(Backus-Naur Form, 巴科斯范式), 程序设计语言的语法规范常使用 BNF 描述.

定义. 如果字符串 $w \in \Sigma^*$ 满足

$$w = w^R,$$

则称字符串 w 为回文 *(palindrome).*

定义. 如果语言 L 中的字符串都是回文, 则称 L 为回文语言

$$L = \{w \in \Sigma^* \mid w = w^R\}.$$

- ε, 010, 0000, radar, racecar, drawkward
- A man, a plan, a canal — Panama
- 僧游云隐寺, 寺隐云游僧

例 1. 字母表 $\Sigma = \{0, 1\}$ 上的回文语言

$$L_{\text{pal}} = \{w \in \{0, 1\}^* \mid w = w^R\}.$$

- 很容易证明是 L_{pal} 是非正则的. 但如何表示呢?
- 可使用递归的方式来定义:

  1. 首先 ε, 0, 1 都是回文
  2. 如果 w 是回文, $0w0$ 和 $1w1$ 也是回文

- 使用嵌套定义表示这种递归结构:

$$\begin{array}{ll} A \to \varepsilon & A \to 0A0 \\ A \to 0 & A \to 1A1 \\ A \to 1 & \end{array}$$

使用上面的文法, 可以通过 A 产生 $\{0,1\}$ 上全部的回文串, 比如串 0010100 可以通过先使用 $A \to 0A0$ 两次, 再使用 $A \to 1A1$ 一次, 再使用 $A \to 0$ 一次得到.

5.1.1 形式定义

定义. 上下文无关文法 *(CFG, Context-Free Grammar,* 简称文法*)* G 是一个四元组

$$G = (V, T, P, S),$$

1. V: 变元 *(Variable)* 的有穷集, 变元也称为非终结符或语法范畴;
2. T: 终结符 *(Terminal)* 的有穷集, 且 $V \cap T = \varnothing$;
3. P: 产生式 *(Production)* 的有穷集, 每个产生式包括:
   - *i* 一个变元, 称为产生式的头 *(head)* 或左部 *(left-hand side, LHS)*;
   - *ii* 一个产生式符号 $\to$, 读作定义为;
   - *iii* 一个 $(V \cup T)^*$ 中的符号串, 称为体 *(body)* 或右部 *(right-hand side, RHS)*;
4. $S \in V$: 初始符号 *(Start symbol)*, 文法开始的地方.

- 产生式 $A \to \alpha$, 读作 A 定义为 α
- 如果有多个 A 的产生式

$$A \to \alpha_1,\ A \to \alpha_2,\ \cdots,\ A \to \alpha_n$$

  可简写为

$$A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$$

- 文法中变元 A 的全体产生式, 称为 A 产生式

续例 1. 回文语言 $L_{\text{pal}} = \{w \in \{0,1\}^* \mid w = w^R\}$ 的文法可设计为

$$G = (\{A\}, \{0,1\}, \{A \to \varepsilon \mid 0 \mid 1 \mid 0A0 \mid 1A1\}, A).$$

在 $(V \cup T)^*$ 的串上使用 (*apply*) 产生式的过程, 就是将串中产生式左部的变元替换为产生式右部的串. 即如果有串 $\alpha, \beta, \gamma \in (V \cup T)^*$ 和变元 $A \in V$, 在串 $\alpha A \beta$ 上使用产生式 $A \to \gamma$ 后, 可得串 $\alpha\gamma\beta$.

- 上下文无关文法是形式化描述语言的又一种工具;
- 它表示语言的能力强于有穷自动机和正则表达式, 但也有无法它表示的语言;
- 基本思想是用变量表示串的集合;
- 递归的使用变量去定义变量, 因此善于表示嵌套的结构, 比如匹配的括号等;
- 变量之间仅使用了“连接”, 同一变量的不同定义使用了“并”.

字符使用的一般约定

- 终结符: $0, 1, \ldots,\ a, b, \ldots$
- 终结符串: $\ldots, w, x, y, z$
- 非终结符: $S, A, B, \ldots$
- 终结符或非终结符: $\ldots, X, Y, Z$
- 终结符或非终结符组成的串: $\alpha, \beta, \gamma, \ldots$

例 2. 简化版的算数表达式:

- 运算只有“加”和“乘”$(+, *)$, 参数仅为标识符;
- 标识符: 以 $\{a, b\}$ 开头由 $\{a, b, 0, 1\}$ 组成的字符串.

这样的表达式集合可用文法 G_{exp} 表示

$$G_{\text{exp}} = (\{E, I\}, \{a, b, 0, 1, +, *, (,)\}, P, E),$$

其中产生式集 P 中有 10 条产生式

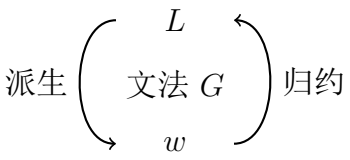
1. $E \to I$	5. $I \to a$	9. $I \to I0$
2. $E \to E + E$	6. $I \to b$	10. $I \to I1$
3. $E \to E * E$	7. $I \to Ia$	
4. $E \to (E)$	8. $I \to Ib$	

注意, 变元 I 所定义的标识符集合, 刚好是 $(\mathbf{a} + \mathbf{b})(\mathbf{a} + \mathbf{b} + \mathbf{0} + \mathbf{1})^*$.

5.1.2 归约和派生

非形式定义

从字符串到文法变元的分析过程, 称为递归推理 (*recursive inference*) 或归约 (*reduction*) ;
从文法变元到字符串的分析过程, 称为推导或派生 (*derivation*).



- 归约: 自底向上 (*bottom up*), 由产生式的体向头的分析
- 派生: 自顶向下 (*top down*), 由产生式的头向体分析

续例 2. 用算数表达式文法 G_{exp}, 将 $a*(a+b00)$ 归约的过程.

1. $E \to I$
2. $E \to E+E$
3. $E \to E*E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

	串归约到变元		应用产生式	重用结果
(1)	a	I	5. $I \to a$	–
(2)	b	I	5. $I \to b$	–
(3)	$b0$	I	9. $I \to I0$	(2)
(4)	$b00$	I	9. $I \to I0$	(3)
(5)	a	E	1. $E \to I$	(1)
(6)	$b00$	E	1. $E \to I$	(4)
(7)	$a+b00$	E	2. $E \to E+E$	(5), (6)
(8)	$(a+b00)$	E	4. $E \to (E)$	(7)
(9)	$a*(a+b00)$	E	3. $E \to E*E$	(5), (8)

派生和归约的形式定义

定义. *若 CFG $G=(V,T,P,S)$, 设 $\alpha,\beta,\gamma \in (V\cup T)^*$, $A \in V$, $A \to \gamma \in P$, 那么称在 G 中由 $\alpha A\beta$ 可派生出 $\alpha\gamma\beta$, 记为*

$$\alpha A\beta \underset{G}{\Rightarrow} \alpha\gamma\beta.$$

相应的, 称 $\alpha\gamma\beta$ 可归约为 $\alpha A\beta$.

- $\alpha A\beta \underset{G}{\Rightarrow} \alpha\gamma\beta$, 即用 $A \to \gamma$ 的右部 γ 替换串 $\alpha A\beta$ 中变元 A 得到串 $\alpha\gamma\beta$
- 如果语境中 G 是已知的, 可省略, 记为 $\alpha A\beta \Rightarrow \alpha\gamma\beta$

- 设 $\alpha_1, \alpha_2, \ldots, \alpha_m \in (V \cup T)^*$, $m \geq 1$, 对 $i = 1, 2, \ldots, m-1$ 如果有

$$\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$$

成立, 即 α_1 经过零步或多步派生可得到 α_m

$$\alpha_1 \underset{G}{\Rightarrow} \alpha_2 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \alpha_{m-1} \underset{G}{\Rightarrow} \alpha_m,$$

那么, 记为

$$\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m.$$

- 若 α 派生出 β 刚好经过了 i 步, 可记为

$$\alpha \overset{i}{\underset{G}{\Rightarrow}} \beta.$$

续例 2. 算数表达式 $a*(a+b00)$ 在文法 G_{exp} 中的派生过程.

$$\begin{aligned} E &\Rightarrow E*E \Rightarrow E*(E) \Rightarrow I*(E) \\ &\Rightarrow I*(E+E) \Rightarrow I*(E+I) \Rightarrow I*(I+I) \\ &\Rightarrow I*(a+I) \Rightarrow a*(a+I) \Rightarrow a*(a+I0) \\ &\Rightarrow a*(a+I00) \Rightarrow a*(a+b00) \end{aligned}$$

5.1.3 最左派生和最右派生

定义. 为限制派生的随意性, 要求只替换符号串中最左边变元的派生过程, 称为最左派生 *(left-most derivation)*, 记为

$$\underset{\text{lm}}{\Rightarrow}, \quad \overset{*}{\underset{\text{lm}}{\Rightarrow}},$$

只替换最右的, 称为最右派生 *(right-most derivation)*, 记为

$$\underset{\text{rm}}{\Rightarrow}, \quad \overset{*}{\underset{\text{rm}}{\Rightarrow}}.$$

- 任何派生都有等价的最左派生和最右派生

$$A \overset{*}{\Rightarrow} w \ \text{ 当且仅当 } \ A \overset{*}{\underset{\text{lm}}{\Rightarrow}} w \ \text{ 当且仅当 } \ A \overset{*}{\underset{\text{rm}}{\Rightarrow}} w.$$

续例 2. 表达式 $a*(a+a)$ 在 G_{exp} 中的最左派生和最右派生分别为:

$$E \underset{\text{lm}}{\Rightarrow} E*E \qquad\qquad E \underset{\text{rm}}{\Rightarrow} E*E$$

$$
\begin{aligned}
&\underset{\text{lm}}{\Rightarrow} I * E && \underset{\text{rm}}{\Rightarrow} E * (E)\\
&\underset{\text{lm}}{\Rightarrow} a * E && \underset{\text{rm}}{\Rightarrow} E * (E + E)\\
&\underset{\text{lm}}{\Rightarrow} a * (E) && \underset{\text{rm}}{\Rightarrow} E * (E + I)\\
&\underset{\text{lm}}{\Rightarrow} a * (E + E) && \underset{\text{rm}}{\Rightarrow} E * (E + a)\\
&\underset{\text{lm}}{\Rightarrow} a * (I + E) && \underset{\text{rm}}{\Rightarrow} E * (I + a)\\
&\underset{\text{lm}}{\Rightarrow} a * (a + E) && \underset{\text{rm}}{\Rightarrow} E * (a + a)\\
&\underset{\text{lm}}{\Rightarrow} a * (a + I) && \underset{\text{rm}}{\Rightarrow} I * (a + a)\\
&\underset{\text{lm}}{\Rightarrow} a * (a + a) && \underset{\text{rm}}{\Rightarrow} a * (a + a)
\end{aligned}
$$

5.1.4 文法的语言

定义. *CFG* $G = (V, T, P, S)$ 的语言定义为

$$\mathbf{L}(G) = \{w \mid w \in T^*,\ S \overset{*}{\underset{G}{\Rightarrow}} w\}.$$

那么符号串 w 在 $\mathbf{L}(G)$ 中, 要满足:

1. w 仅由终结符组成;
2. 初始符号 S 能派生出 w.

上下文无关语言

定义. 语言 L 是某个 *CFG* G 定义的语言, 即 $L = \mathbf{L}(G)$, 则称 L 为上下文无关语言 *(CFL, Context-Free Language).*

- 上下文无关是指在文法派生的每一步

$$\alpha A \beta \Rightarrow \alpha \gamma \beta,$$

符号串 γ 仅根据 A 的产生式派生, 而无需依赖 A 的上下文 α 和 β.

文法的等价性

定义. 如果有两个文法 *CFG* G_1 和 *CFG* G_2, 满足

$$\mathbf{L}(G_1) = \mathbf{L}(G_2),$$

则称 G_1 和 G_2 是等价的.

句型

定义. *若 CFG $G = (V, T, P, S)$, 初始符号 S 派生出来的符号串, 称为 G 的句型 (sentential form), 即*

$$\alpha \in (V \cup T)^* \text{ 且 } S \overset{*}{\Rightarrow} \alpha.$$

如果 $S \overset{*}{\underset{\text{lm}}{\Rightarrow}} \alpha$, 称 α 为左句型. 如果 $S \overset{*}{\underset{\text{rm}}{\Rightarrow}} \alpha$, 称 α 为右句型.

- 只含有终结符的句型, 也称为 G 的句子 (*sentence*)
- 而 $\mathbf{L}(G)$ 就是文法 G 全部的句子

5.1.5 文法设计举例

例 3. 给出语言 $L = \{w \in \{0,1\}^* \mid w \text{ contains at least three 1s}\}$ 的文法.

解: $S \to A1A1A1A$, $A \to 0A \mid 1A \mid \varepsilon$

例 4. 描述 CFG $G = (\{S\}, \{a, b\}, \{S \to aSb, S \to ab\}, S)$ 定义的语言?

解: $\mathbf{L}(G) = \{a^n b^n \mid n \geq 1\}$, 因为 $S \Rightarrow aSb \Rightarrow \cdots \Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^n b^n$.

例 5. 请为语言 $L = \{0^n 1^m \mid n \neq m\}$ 设计文法.

解:
$$\begin{array}{ll} S \to AC \mid CB & A \to A0 \mid 0 \\ C \to 0C1 \mid \varepsilon & B \to 1B \mid 1 \end{array}$$

例 6. 设计 $L_{\text{eq}} = \{w \in \{0,1\}^* \mid w \text{ 中 0 和 1 个数相等}\}$ 的文法.

解 1: $S \to 0S1 \mid 1S0 \mid SS \mid \varepsilon$, 寻找递归结构, 用变量构造递归结构;

解 2: $S \to S0S1S \mid S1S0S \mid \varepsilon$, “目标串”这样构成, 由变量定义变量.

例 7. 设计 $L_{j \geq 2i} = \{a^i b^j \mid j \geq 2i\}$ 的文法.

解:
$$\begin{array}{l} S \to AB \\ A \to aAbb \mid \varepsilon \\ B \to bB \mid \varepsilon \end{array} \quad \text{或} \quad \begin{array}{l} S \to aSbb \mid B \\ B \to \varepsilon \mid bB \end{array}$$

程序设计语言的文法定义

- C — ISO C 1999 definition

...
selection-statement:
`if (` *expression* `)` *statement*
`if (` *expression* `)` *statement* `else` *statement*
`switch (` *expression* `)` *statement*
...

- Python — Full Grammar specification — https://docs.python.org/3/reference/grammar.html

...

```
try_stmt: ('try' ':' suite
           ((except_clause ':' suite)+
            ['else' ':' suite]
            ['finally' ':' suite] |
           'finally' ':' suite))
```

...

例 8. [Exe. 5.1.3] Show that every regular laugnage is a context-free laugnage.

证明：对正则表达式 R 中运算符的个数 n 进行归纳.

归纳基础: 当 $n = 0$ 时, R 只能是 $\boldsymbol{\varepsilon}$, $\varnothing$ 或 $\mathbf{a}$ ($a \in \Sigma$), 可以构造仅有一条产生式的文法 $S \to \varepsilon$, $S \to \varnothing$ 或 $S \to a$ 得到.

归纳递推: 假设当 $n \leq m$ 时成立. 当 $n = m + 1$ 时, R 的形式只能由表达式 R_1 和 R_2 由连接、并或闭包形成：

- 若 $R = R_1 + R_2$, 则 R_1 和 R_2 中运算符都不超过 m, 所以都存在文法 G_1 和 G_2, 分别开始于 S_1 和 S_2, 只需构造新产生式和开始符号 $S \to S_1|S_2$, 连同 G_1 和 G_2 的产生式, 构成 R 的文法；
- 若 $R = R_1R_2$, 则同理构造 $S \to S_1S_2$ 即可；
- 若 $R = R_1^*$, 则构造 $S \to SS_1 \mid \varepsilon$ 即可.

且每种构造, 文法的语言与该表达式的语言等价. □

例 9. [Exe. 5.1.5] Let $T = \{0, 1, (,), +, *, \varnothing, e\}$. We may think of T as the set of symbols used by regular expressions over the alphabet $\{0, 1\}$; the only difference is that we use e for symbol ε, to avoid potential confusion in what follows. Your task is to design a CFG with set of terminals T that generates exactly the regular expressions with alphabet $\{0, 1\}$.

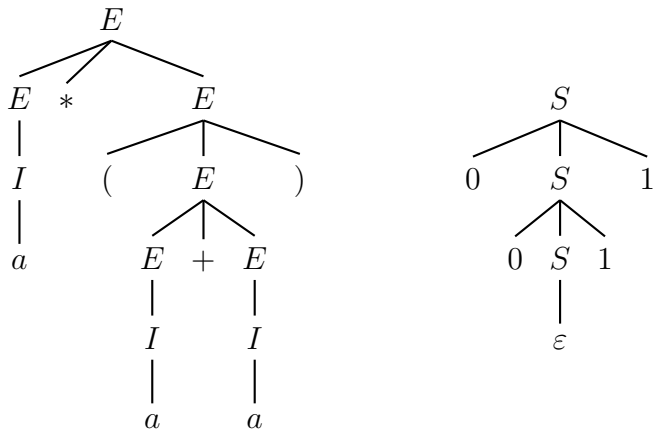
解: $S \to S + S \mid SS \mid S^* \mid (S) \mid 0 \mid 1 \mid \varnothing \mid e$.

5.2 语法分析树

语法分析树与派生 (或归约) 紧密关联, 可以从树中看出整个派生过程和最终产生的符号串. 在编译器设计中, 语法分析树是表示源程序的重要数据结构, 用来指导由程序到可执行代码的翻译. 语法分析树的一个重要应用, 语法的歧义性研究, 研究有关不适合用来作为程序设计语言的语法.

派生或归约的过程可以表示成树形结构.

- 例 2 文法 G_{exp} 中推导算数表达式 $a*(a+a)$ 的过程
- 例 6 中语言 L_{eq} 的文法中推导 0011 的过程



5.2.1 形式定义

定义. *CFG $G=(V,T,P,S)$ 的语法分析树 (parse tree, 简称语法树或派生树) 为:*

1. *每个内节点标记为 V 中的变元符号;*
2. *每个叶节点标记为 $V\cup T\cup\{\varepsilon\}$ 中的符号;*
3. *如果某内节点标记是 A, 其子节点从左至右分别为*

$$X_1,\ X_2,\ \cdots,\ X_n$$

那么

$$A\to X_1X_2\cdots X_n\in P,$$

若有 $X_i=\varepsilon$, 则 ε 是 A 唯一子节点, 且 $A\to\varepsilon\in P$.

定义. *语法树的全部叶节点从左到右连接起来, 称为该树的产物 (yield) 或结果.*

- 如果树根节点是初始符号 S, 叶节点是终结符或 ε, 那么该树的产物属于 $\mathbf{L}(G)$.

定义. *语法树中标记为 A 的内节点及其全部子孙节点构成的子树, 称为 A 子树.*

5.2.2 语法树和派生的等价性

定理 17. *CFG $G = (V, T, P, S)$ 且 $A \in V$, 那么文法 G 中*

$$A \overset{*}{\Rightarrow} \alpha$$

当且仅当 G 中存在以 A 为根节点产物为 α 的语法树.

证明:
[充分性 $\Rightarrow$] 对派生 $A \overset{j}{\Rightarrow} \alpha$ 的步骤 j 数做归纳证明.

1. 归纳基础: 当 $j = 1$ 时, 即 $A \Rightarrow \alpha$, 由派生的定义, 一定有产生式 $A \to \alpha \in P$, 再由语法树定义, 可构造以 A 为根、产物为 α 的语法树 $\overset{A}{\underset{\alpha}{\uparrow}}$, 因此命题成立;

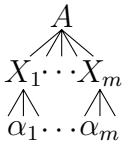
2. 归纳递推: 假设 $j \le n$ 时命题成立. 当 $i = n + 1$ 时, 则 $A \overset{n+1}{\Longrightarrow} \alpha$ 的派生过程为

   $$A \Rightarrow X_1 X_2 \cdots X_m \overset{n}{\Rightarrow} \alpha_1 \alpha_2 \cdots \alpha_m = \alpha,$$

   即派生的第 1 步, 一定首先由某产生式 $A \to X_1 X_2 \cdots X_m \in P$ 推导出, 其中每个 X_i 或为终结符或为变元, 而且每个变元的派生

   $$X_i \overset{*}{\Rightarrow} \alpha_i$$

   都不超过 n 步. 根据归纳假设, 每个 $X_i \overset{*}{\Rightarrow} \alpha_i$ 都有一棵以 X_i 为根、以 α_i 为产物的语法分析树 $\overset{X_i}{\underset{\alpha_i}{\uparrow}}$. 那么, 可以构造以 A 为根, 以 X_i 为子树 (或叶子) 的语法树, 其产物刚好为 $\alpha_1 \alpha_2 \cdots \alpha_m = \alpha$.

   因此命题成立.

[必要性 $\Leftarrow$] 对语法分析树的内部节点数 j 做归纳证明.

1. 归纳基础: 当 $j = 1$ 时, 该树一定是以 A 为根、α 为产物 $\overset{A}{\underset{\alpha}{\uparrow}}$, 由语法树定义, 产生式 $A \to \alpha \in P$, 那么 $A \overset{*}{\Rightarrow} \alpha$;

2. 归纳递推: 假设 $j \le n$ 时命题成立. 当 $j = n + 1$ 时, 设根节点 A 的全部子节点从左至右分别为 $X_1, X_2, \ldots, X_m$, 那么显然 $A \to X_1 X_2 \cdots X_m \in P$, 且

   $$A \Rightarrow X_1 X_2 \cdots X_m.$$

而每个 X_i 或为叶子或为 X_i 子树, 且 X_i 子树的内节点数都不超过 n, 根据归纳假设

$$X_i \overset{*}{\Rightarrow} \alpha_i.$$

又因为每个 X_i 子树的产物 α_i(或为叶子的 X_i 自身), 从左至右连接起来刚好为树的产物 α, 所以有

$$X_1X_2\cdots X_m \overset{*}{\Rightarrow} \alpha_1X_2\cdots X_m \overset{*}{\Rightarrow} \cdots \overset{*}{\Rightarrow} \alpha_1\alpha_2\cdots\alpha_m = \alpha.$$

因此 $A \Rightarrow X_1X_2\cdots X_m \overset{*}{\Rightarrow} \alpha$ 因此命题成立. □

语法树唯一确定最左 (右) 派生

- 每棵语法分析树都有唯一的最左 (右) 派生
- 给定 CFG $G = (V, T, P, S)$, $A \in V$, 以下命题等价:
  1. 通过递归推理, 确定串 w 在变元 A 的语言中.
  2. 存在以 A 为根节点, 产物为 w 的语法分析树.
  3. $A \overset{*}{\Rightarrow} w$.
  4. $A \overset{*}{\underset{\text{lm}}{\Rightarrow}} w$.
  5. $A \overset{*}{\underset{\text{rm}}{\Rightarrow}} w$.

例 10. [Exe. 5.2.2] Suppose that G is a CFG without any productions that have ε as the right side. If w is in $L(G)$, the length of w is n, and w has a derivation of m steps, show that w has a parse tree with $n + m$ nodes.

证明:

1. 派生 w 的每一步推导都对应语法树的一个内节点, 所以 w 语法树中共有 m 个内节点;
2. 每个 w 的终结符都构成一个叶节点, 所以至少有 n 个叶节点, 而由于 G 中没有空产生式, 因此不会有标记为 ε 的叶节点, 所以只能有 n 个叶节点. 所以 w 的语法树有 $n + m$ 个节点. □

例 11. [Exe. 5.2.3] Suppose all is as in Exercise 5.2.2, but G may have some productions with ε as the right side. Show that a parse tree for a string w other than ε may have as many as $n + 2m - 1$ nodes, but no more.

证明:

1. 派生 w 的每一步推导都对应语法树的一个内节点, 所以 w 语法树中共有 m 个内节点.

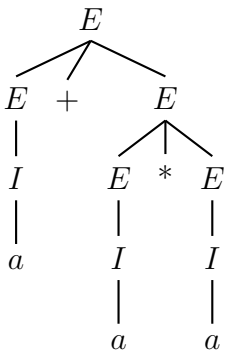
2. 每个 w 的终结符都构成一个叶节点, 所以至少有 n 个叶节点.

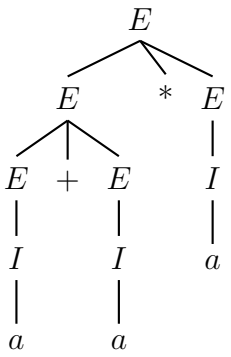
3. 推导过程中, 每次空产生式的应用, 都会增加一个标记 ε 的叶节点, 但显然不能全部的 m 步都使用空产生式, 所以最多增加 $m-1$ 个 ε 叶节点. 因此 w 的语法树有最多 $m+n+m-1=n+2m-1$ 个节点. □

5.3 文法和语言的歧义性

定义. 如果 *CFG* G 使某些符号串有两棵不同的语法分析树, 称文法 G 是歧义 *(ambiguity)* 的.

续例 2. 算数表达式的文法 G_{exp} 中, 对句型 $a+a*a$ 有下面两棵语法分析树:





(1) $E \Rightarrow E+E$
$\Rightarrow E+E*E$
$\overset{*}{\Rightarrow} a+a*a$

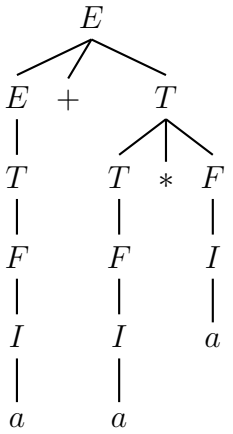
(2) $E \Rightarrow E*E$
$\Rightarrow E+E*E$
$\overset{*}{\Rightarrow} a+a*a$

5.3.1 文法歧义性的消除

有些文法的歧义性, 可以通过重新设计文法来消除.

续例 2. 文法 G_{exp} 重新设计为文法 $G_{\text{exp}'}$ 可消除歧义.

$$
\begin{aligned}
&E \to E + T \mid T \\
&T \to T * F \mid F \\
&F \to (E) \mid I \\
&I \to a \\
&I \to b \\
&I \to Ia \\
&I \to Ib \\
&I \to I0 \\
&I \to I1
\end{aligned}
$$



5.3.2 语言的固有歧义性

定义. 定义同样的语言可以有多个文法, 如果 *CFL* L 的所有文法都是歧义的, 那么称语言 L 是固有歧义 *(Inherent Ambiguity)* 的.

- 固有歧义的语言确实存在, 如语言

  $$L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$$

  中任何形为 $a^n b^n c^n$ 的串, 总会有两棵语法树.

- "判定任何给定 CFG G 是否歧义"是一个不可判定问题.

5.4 文法的化简与范式

为什么要化简

- 典型问题: 给定 CFG G 和串 w, 判断 $w \in \mathbf{L}(G)$?
- 编译器设计和自然语言处理的基本问题之一
- 但文法的形式非常自由, 过于复杂不易于自动处理
- 以不改变语言为前提, 化简文法和限制文法的格式

例 12. 如下文法中, 有无意义的变元和产生式

$$S \to 0DS1D \mid B \mid \varepsilon$$
$$B \to BC1 \mid 0CBC$$
$$A \to A0 \mid A1 \mid \varepsilon$$
$$C \to D$$
$$D \to \varepsilon$$

在这个文法中: B 无法生成全部为终结符的串; A 无法从 S 派生出来; $C \to D$ 在推导过程中仅增加了推导的长度; C 和 D 能派生出空串, 为其他串 w 的归约增加了难度. 所有这些对文法定义的语言没有贡献, 需要删除掉来简化文法.

文法的化简

1. 消除无用符号 (*useless symbols*): 对文法定义语言没有贡献的符号
2. 消除 ε 产生式 (ε-*productions*): $A \to \varepsilon$ (得到语言 $L - \{\varepsilon\}$)
3. 消除单元产生式 (*unit productions*): $A \to B$

无用符号, 从文法开始符号派生到终结符串的过程中用不到; ε-产生式, 除了空串 ε 自身, 没有贡献语言中其他的串; 单元产生式, 仅仅增加了推导 (或归约) 的步骤. 这三者对语言的定义都没有实质的作用.

5.4.1 消除无用符号

无用符号

定义. *CFG* $G = (V, T, P, S)$, *符号* $X \in (V \cup T)$*:*

1. *如果* $S \overset{*}{\Rightarrow} \alpha X \beta$, *称* X *是可达的 (reachable);*
2. *如果* $\alpha X \beta \overset{*}{\Rightarrow} w$ $(w \in T^*)$, *称* X *是产生的 (generating);*
3. *如果* X *同时是产生的和可达的, 即*

$$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w \quad (w \in T^*),$$

则称 X *是有用的, 否则称* X *为无用符号.*

消除无用符号

消除无用符号: 删除全部含有 “非产生的” 和 “非可达的” 符号的产生式

计算 “产生的” 符号集

1. 每个 T 中的符号都是产生的;
2. $A \to \alpha \in P$ 且 α 中符号都是产生的, 则 A 是产生的.

计算 “可达的” 符号集

1. 符号 S 是可达的;
2. $A \to \alpha \in P$ 且 A 是可达的, 则 α 中符号都是可达的.

定理 18. *每个非空的 CFL 都能被一个不带无用符号的 CFG 定义.*

注意

- 先寻找并消除全部非 “产生的” 符号
- 再寻找并消除全部非 “可达的” 符号
- 否则可能消除不完整

例 13. 消除如下文法无用符号

$S \to AB \mid a$
$A \to b$

解: 先消除非产生的

$S \to a$
$A \to b$

再消除非可达的

$S \to a$

5.4.2 消除 ε-产生式

定义. *文法中形如 $A \to \varepsilon$ 的产生式称为ε-产生式.*

如果变元 $A \overset{*}{\Rightarrow} \varepsilon$, 称 A 是可空的.

- ε-产生式在文法定义语言时, 除产生空串外没有其他帮助
- 对于 CFL L, 消除其文法中全部的 ε-产生式后, 得到语言 $L - \{\varepsilon\}$

确定“可空变元”

1. 如果 $A \to \varepsilon$, 则 A 是可空的;
2. 如果 $B \to \alpha$ 且 α 中的每个符号都是可空的, 则 B 是可空的.

替换带有可空变元的产生式

将含有可空变元的一条产生式 $A \to X_1X_2\cdots X_n$, 用一组产生式 $A \to Y_1Y_2\cdots Y_n$ 代替, 其中:

1. 若 X_i 不是可空的, Y_i 为 X_i;
2. 若 X_i 是可空的, Y_i 为 X_i 或 ε;
3. 但 Y_i 不能全为 ε.

定理 19. *任何 CFG G, 都存在一个不带无用符号和 ε-产生式的 CFG G', 使 $\mathbf{L}(G') = \mathbf{L}(G) - \{\varepsilon\}$.*

例 14. 消除 CFG $G = (\{S, A, B\}, \{a, b\}, P, S)$ 的 ε-产生式.

$$
\begin{aligned}
&S \to AB\\
&A \to AaA \mid \varepsilon\\
&B \to BbB \mid \varepsilon
\end{aligned}
$$

解: CFG G' 为

$$
\begin{aligned}
&S \to AB \mid A \mid B\\
&A \to AaA \mid Aa \mid aA \mid a\\
&B \to BbB \mid Bb \mid bB \mid b
\end{aligned}
$$

5.4.3 消除单元产生式

确定“单元对”

如果有 $A \overset{*}{\Rightarrow} B$, 则称 $[A, B]$ 为单元对.

1. $A \to B \in P$, 则 $[A, B]$ 是单元对;
2. 若 $[A, B]$ 和 $[B, C]$ 都是单元对, 则 $[A, C]$ 是单元对.

消除单元产生式

1. 删除全部形为 $A \to B$ 的单元产生式;
2. 对每个单元对 $[A, B]$, 将 B 的产生式复制给 A.

定理 20. *每个不带 ε 的 CFL 都可由一个不带无用符号, ε-产生式和单元产生式的文法定义.*

例 15. 消除文法的单元产生式

$$
\begin{aligned}
&S \to A \mid B \mid 0S1\\
&A \to 0A \mid 0\\
&B \to 1B \mid 1
\end{aligned}
$$

解: 单位对为 $[S, A]$ 和 $[S, B]$, 带入得:

$$
\begin{aligned}
&S \to 0S1 \qquad\qquad A \to 0A \mid 0\\
&S \to 0A \mid 0 \qquad\qquad B \to 1B \mid 1\\
&S \to 1B \mid 1
\end{aligned}
$$

1. 消除 ε-产生式;
2. 消除单元产生式;
3. 消除非产生的无用符号;
4. 消除非可达的无用符号.

例 16. [Exe. 7.1.2] Begin with the grammar:

$$
\begin{aligned}
&S \to ASB \mid \varepsilon\\
&A \to aAS \mid a\\
&B \to SbS \mid A \mid bb
\end{aligned}
$$

1. Eliminate $\varepsilon-$productions.
2. Eliminate any unit productions in the resulting grammar.
3. Eliminate any useless symbols in the resulting grammar.

限制文法格式

将任意形式的文法转换为:

1. 乔姆斯基范式 (CNF, Chomsky Normal Form)
2. 格雷巴赫范式 (GNF, Greibach Normal Form)

5.4.4 乔姆斯基范式

定理 21 (乔姆斯基范式 CNF)**.** *每个不带 ε 的 CFL 都可由这样的 CFG G 定义, G 中每个产生式都形为*

$$A \to BC \quad 或 \quad A \to a$$

其中 A, B 和 C 都是变元, a 是终结符.

- 利用 CNF 派生长度为 n 的串, 刚好需要 $2n-1$ 步
- 因此存在算法判断任意字符串 w 是否在给定的 CFL 中
- 利用 CNF 的 CYK 算法 — $O(n^3)$ 时间复杂度的解析算法

CFG 转为 CNF 的方法

1. 将产生式

   $$A \to X_1 X_2 \cdots X_m \quad (m \geq 2)$$

   中每个终结符 a 替换为新变元 C_a, 并增加新产生式

   $$C_a \to a$$

2. 引入新变元 $D_1, D_2, \cdots, D_{m-2}$, 将产生式

   $$A \to B_1 B_2 \cdots B_m \quad (m > 2)$$

   替换为一组级联的产生式

   $$A \to B_1 D_1$$
   $$D_1 \to B_2 D_2$$
   $$\cdots$$
   $$D_{m-2} \to B_{m-1} B_m$$

例 17. CFG $G = (\{S, A, B\}, \{a, b\}, P, S)$, 产生式集合 P 为:

$$S \to bA \mid aB$$
$$A \to bAA \mid aS \mid a$$
$$B \to aBB \mid bS \mid b$$

请设计等价的 CNF 文法.

解: CNF 为

$$S \to C_bA \mid C_aB$$
$$A \to C_aS \mid C_bD_1 \mid a \qquad D_1 \to AA \qquad C_a \to a$$
$$B \to C_bS \mid C_aD_2 \mid b \qquad D_2 \to BB \qquad C_b \to b$$

证明: 设 CFL L 不含 ε, 由定理 20, 存在不含 ε-产生式和单元产生式的等价文法 $G_1 = (V, T, P, S)$. 考虑 P 中一条产生式 $A \to X_1X_2 \dots X_m$ $(m \geq 2)$.

1. 若某个 X_i 是终结符 a, 则引入新变元 C_a 和新产生式 $C_a \to a$, 并用 C_a 替换 X_i, 设新的变元集为 V', 新的产生式集合为 P', 得文法 $G_2 = (V', T, P', S)$, 但要注意 G_2 还不是 CNF 形式.

2. 显然 $\mathbf{L}(G_1) \subseteq \mathbf{L}(G_2)$, 因为如果 $\alpha \underset{G_1}{\Rightarrow} \beta$, 那么 $\alpha \overset{*}{\underset{G_2}{\Rightarrow}} \beta$. 这是因为, $\alpha \underset{G_1}{\Rightarrow} \beta$ 使用的产生式, 如果不是 $A \to X_1X_2 \dots X_m$, 在 G_2 中也存在; 而如果是, 在 G_2 中可以使用替换后的产生式和所有的 $C_a \to a$ 来得到.

3. 为了证明 $\mathbf{L}(G_2) \subseteq \mathbf{L}(G_1)$, 用归纳法证明 $A \overset{i}{\underset{G_2}{\Rightarrow}} w \Longrightarrow A \overset{*}{\underset{G_1}{\Rightarrow}} w$, 这里的 $A \in V$, $w \in T^*$.

   i 当 $i = 1$ 时是显然的, 或者用了 P 中未修改的产生式, 或者用了被修改的产生式, 而二者都有 $A \overset{*}{\underset{G_1}{\Rightarrow}} w$.

   ii 假设当 $i \leq n$ 时命题成立. 当 $i = n+1$ 时, $A \overset{i}{\underset{G_2}{\Rightarrow}} w$ 的第 1 步, 必然使用了某个产生式 $A \to B_1B_2 \cdots B_m$, 即 $A \underset{G_2}{\Rightarrow} B_1B_2 \cdots B_m \overset{n}{\underset{G_2}{\Rightarrow}} w = w_1w_2 \cdots w_m$ 那么, 如果 $B_i \in V' - V$, B_i 一定是对应某个终结符 a_i 的 C_{a_i}, w_i 必然是 a_i, 令 $X_i = a_i$; 如果 $B_i \in V$, $B_i \overset{*}{\underset{G_2}{\Rightarrow}} w_i$ 一定不超过 n 步, 由归纳假设, $B_i \overset{*}{\underset{G_1}{\Rightarrow}} w_i$, 那么令 $X_i = B_i$. 由 P' 的结构, $A \to X_1X_2 \cdots X_m$ 是 P 的一条产生式, 所以 $A \underset{G_1}{\Rightarrow} X_1X_2 \cdots X_m \overset{*}{\underset{G_1}{\Rightarrow}} w_1w_2 \cdots w_m = w$.

   所以 $\mathbf{L}(G_2) \subseteq \mathbf{L}(G_1)$. □

5.4.5 格雷巴赫范式

定理 22 (格雷巴赫范式 GNF)**.** *每个不带 ε 的 CFL 都可由这样的 CFG G 定义, G 中每个产生式都形为*

$$A \to a\alpha$$

其中 A 是变元, a 是终结符, α 是零或多个变元的串.

- GNF 每个产生式都会引入一个终结符
- 长度为 n 的串的派生恰好是 n 步

例 18. 将以下文法转换为 GNF.

$$S \to AB$$
$$A \to aA \mid bB \mid b$$
$$B \to b$$

解: GNF 为

$$S \to aAB \mid bBB \mid bB$$
$$A \to aA \mid bB \mid b$$
$$B \to b$$

直接左递归

定义. *文法中形式为 $A \to A\alpha$ 的产生式, 称为直接左递归 (immediate left-recursion).*

消除直接左递归

1. 若 A 产生式
$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m$$
其中 $\alpha_i \neq \varepsilon$, β_j 不以 A 开始;
2. 引入新变元 B, 并用如下产生式替换
$$A \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \mid \beta_1 B \mid \beta_2 B \mid \cdots \mid \beta_m B$$
$$B \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n \mid \alpha_1 B \mid \alpha_2 B \mid \cdots \mid \alpha_n B$$

间接左递归

定义. *文法中如果有形式为*

$$A \to B\alpha \mid \ldots$$
$$B \to A\beta \mid \ldots$$

的产生式, 称为间接左递归 (indirect left-recursion).

- 会有 $A \Rightarrow B\alpha \Rightarrow A\beta\alpha$, 无法通过代换消除递归

消除间接左递归

1. 将文法中变元重命名为 $A_1, A_2, \cdots, A_n$;
2. 通过代入, 使产生式都形如

$$A_i \to A_j\alpha$$
$$A_i \to a\alpha$$

   但要求 $i \le j$;
3. 消除直接左递归 $A_i \to A_i\beta$, 再代入其他产生式.

例 19. Convert the following grammar to GNF.

$$S \to AB$$
$$A \to BS \mid b$$
$$B \to SA \mid a$$

解:

1. 重命名变元, 代换 $i > j$ 的 A_j

$$A_1 \to A_2A_3$$
$$A_2 \to A_3A_1 \mid b$$
$$A_3 \to a \mid \cancel{A_1A_2} \mid \cancel{A_2A_3A_2} \mid A_3A_1A_3A_2 \mid bA_3A_2$$

2. 消除直接左递归

$$A_1 \to A_2A_3$$
$$A_2 \to A_3A_1 \mid b$$
$$A_3 \to bA_3A_2 \mid a \mid bA_3A_2B_1 \mid aB_1$$
$$B_1 \to A_1A_3A_2 \mid A_1A_3A_2B_1$$

3. A_3 产生式代入到 A_2, A_2 产生式代入到 A_1, A_1 产生式代入 B_1

$$A_3 \to bA_3A_2 \mid a \mid bA_3A_2B_1 \mid aB_1$$
$$A_2 \to bA_3A_2A_1 \mid aA_1 \mid bA_3A_2B_1A_1 \mid aB_1A_1 \mid b$$
$$A_1 \to bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2B_1A_1A_3 \mid aB_1A_1A_3 \mid bA_3$$

$$B_1 \to bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2B_1A_1A_3A_3A_2 \mid aB_1A_1A_3A_3A_2 \mid bA_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2B_1 \mid aA_1A_3A_3A_2B_1 \mid bA_3A_2B_1A_1A_3A_3A_2B_1 \mid aB_1A_1A_3A_3A_2B_1 \mid bA_3A_3A_2B_1$$

GNF 引理 1

如果有文法 $G = (V, T, P, S)$, 设 $A \to \alpha_1 B \alpha_2$ 是 P 中的一个产生式, 且 $B \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$ 是 P 中的全部 B 产生式. 将产生式 $A \to \alpha_1 B \alpha_2$ 从 P 中删除, 并增加

$$A \to \alpha_1\beta_1\alpha_2 \mid \alpha_1\beta_2\alpha_2 \mid \cdots \mid \alpha_1\beta_n\alpha_2$$

一组产生式, 得到文法 $G_1 = (V, T, P', S)$, 那么 $\mathbf{L}(G) = \mathbf{L}(G_1)$.

证明:

1. 显然 $\mathbf{L}(G_1) \subseteq \mathbf{L}(G)$, 因为 G_1 的派生中, 如果用到了 $A \to \alpha_1\beta_i\alpha_2$, 在 G 中可以使用 $A \underset{G}{\Rightarrow} \alpha_1 B \alpha_2 \underset{G}{\Rightarrow} \alpha_1\beta_i\alpha_2$.
2. 而因为 $A \to \alpha_1 B \alpha_2$ 是唯一在 G 中而不再 G_1 中的产生式, 每当 G 的派生中用到了 $\alpha_1 B \alpha_2$ 时, 一定会在后面某一步中用到形如 $B \to \beta_i$ 的产生式来派生 B, 这两步在 G_1 中可以使用一步 $A \underset{G_1}{\Rightarrow} \alpha_1\beta_i\alpha_2$ 来代替, 所以 $\mathbf{L}(G) \subseteq \mathbf{L}(G_1)$. □

GNF 引理 2

如果有文法 $G = (V, T, P, S)$, 设带有直接左递归的 A 产生式为

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m$$

其中 β_i 不以 A 开头. 在 V 中引入新的变元 B 并用以下产生式

$$\begin{aligned} A \to & \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \mid \beta_1 B \mid \beta_2 B \mid \cdots \mid \beta_m B \\ B \to & \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n \mid \alpha_1 B \mid \alpha_2 B \mid \cdots \mid \alpha_n B \end{aligned}$$

替换全部 A 产生式, 得到文法 $G_1 = (V \cup \{B\}, T, P', S)$, 那么 $\mathbf{L}(G) = \mathbf{L}(G_1)$.

证明: 在文法 G 中一系列使用 $A \to A\alpha_i$ 的最左派生, 最后必以 $A \to \beta_j$ 结束, 而这样的最左派生

$$\begin{aligned} A &\underset{\text{lm}}{\Rightarrow} A\alpha_{i_1} \underset{\text{lm}}{\Rightarrow} A\alpha_{i_2}\alpha_{i_1} \underset{\text{lm}}{\Rightarrow} \cdots \\ &\underset{\text{lm}}{\Rightarrow} A\alpha_{i_p}\alpha_{i_{p-1}}\ldots\alpha_{i_1} \\ &\underset{\text{lm}}{\Rightarrow} \beta_j\alpha_{i_p}\alpha_{i_{p-1}}\ldots\alpha_{i_1}, \end{aligned}$$

在 G_1 中可以使用一系列最右派生来代替

$$\begin{aligned} A &\underset{\text{rm}}{\Rightarrow} \beta_j B \underset{\text{rm}}{\Rightarrow} \beta_j \alpha_{i_p} B \underset{\text{rm}}{\Rightarrow} \beta_j \alpha_{i_p} \alpha_{i_{p-1}} B \underset{\text{rm}}{\Rightarrow} \cdots \\ &\underset{\text{rm}}{\Rightarrow} \beta_j \alpha_{i_p} \alpha_{i_{p-1}} \dots \alpha_{i_2} B \\ &\underset{\text{rm}}{\Rightarrow} \beta_j \alpha_{i_p} \alpha_{i_{p-1}} \dots \alpha_{i_2} \alpha_{i_1}. \end{aligned}$$

而且, 相反的转换也成立, 因此 $\mathbf{L}(G) = \mathbf{L}(G_1)$. □

例. Convert the following grammar to GNF.

$$\begin{aligned} A_1 &\to A_2 b A_3 \mid a A_1 \\ A_2 &\to A_3 c A_3 \mid b \\ A_3 &\to A_1 c A_3 \mid A_2 b b \mid a \end{aligned}$$

例. [习题 7.1.1] 消除无用符号

$$\begin{aligned} S &\to AB \mid CA \\ A &\to a \\ B &\to AB \\ C &\to ab \mid b \end{aligned}$$

Chapter 6

下推自动机

6.1 下推自动机

正如正则表达式有一个等价的自动机—有穷自动机一样, 上下文无关文法也有其相应的机器—下推自动机. 这里的等价性不太令人满意, 因为下推自动机是一个非确定的装置, 对应的确定装置只能接受全部 CFL 的一个子集. 幸运的是这个子集包含了绝大多数的程序设计语言的文法.

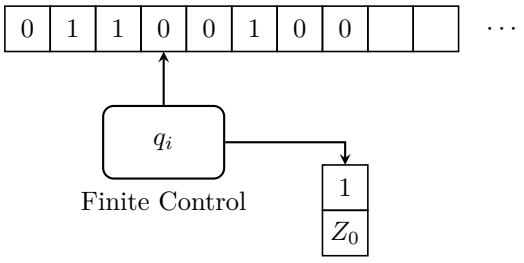
下推自动机实质上是一种能控制一条输入带和一个栈的有穷自动机, 可以看作带有栈的 ε-NFA. 工作方式类似 ε-NFA, 有一个有穷控制器, 并能够以非确定的方式进行状态转移, 并读入输入字符; 增加的堆栈, 用来存储无限的信息, 但只能以后进先出的方式使用.

$$\varepsilon\text{-NFA} + \text{栈} = \text{PDA}$$

ε-NFA: 有限状态, 非确定, ε 转移

栈: 后进先出, 只用栈顶, 长度无限

- *pop*: 仅弹出栈顶的一个符号
- *push*: 可压入一串符号



6.1.1 形式定义

定义. 下推自动机 *(PDA, Pushdown Automata)* P 为七元组

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

1. Q, 有穷状态集;

2. Σ, 有穷输入符号集 (即字母表);

3. Γ, 有穷栈符号集 (或栈字母表);

4. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$, 状态转移函数;

5. $q_0 \in Q$, 初始状态;

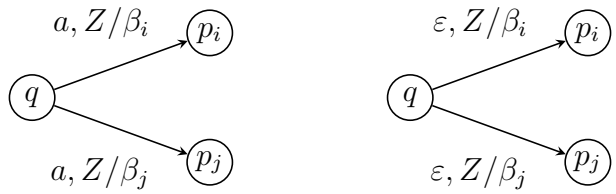
6. $Z_0 \in \Gamma - \Sigma$, 栈底符号, PDA 开始时, 栈中包含这个符号的一个实例, 用来表示栈底, 最初的栈底符号之下无任何内容;

7. $F \subseteq Q$, 接收状态集或终态集.

PDA 的动作和状态转移图

如果 $q, p_i \in Q$ $(1 \le i \le m)$, $a \in \Sigma$, $Z \in \Gamma$, $\beta_i \in \Gamma^*$, 可以有动作:

$$\delta(q, a, Z) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots, (p_m, \beta_m)\}, \text{ 或}$$
$$\delta(q, \varepsilon, Z) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots, (p_m, \beta_m)\}.$$



如果 q 和 p_i 是状态 $(1 \le i \le m)$, 输入符号 $a \in \Sigma$, 栈符号 $Z \in \Gamma$, 栈符号串 $\beta_i \in \Gamma^*$, 那么

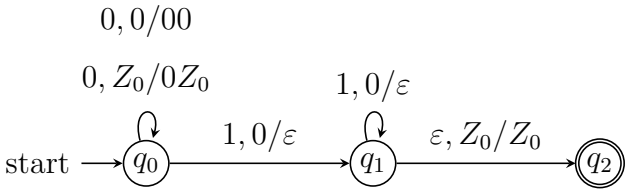
$$\delta(q, a, Z) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots, (p_m, \beta_m)\}$$

的意思是: 输入符号是 a, 栈顶符号 Z 的情况下, 处于状态 q 的 PDA 能够进入状态 p_i, 且用符号串 β_i 替换栈顶的符号 Z, 这里的 i 是任意的, 然后输入头前进一个符号. (约定 β_i 的最左符号在栈最上.) 但是若 $i \ne j$, 不能同时选择 p_i 和 β_j. 而

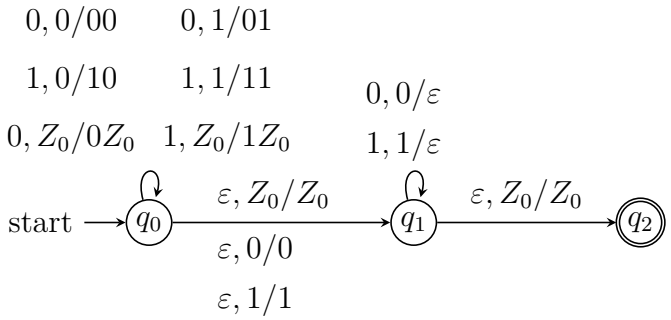
$$\delta(q, \varepsilon, Z) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots, (p_m, \beta_m)\}$$

的意思是: 与扫描的输入符号无关, 只要 Z 是栈符号, 处于状态 q 的 PDA, 就可以进行上面的动作, 输入头不移动.

例 1. 设计识别 $L_{01} = \{0^n 1^n \mid n \ge 1\}$ 的 PDA.



例 2. 设计识别 $L_{wwr} = \{ww^R \mid w \in (\mathbf{0}+\mathbf{1})^*\}$ 的 PDA.



1. 初始状态 q_0, 栈顶 Z_0, 无论输入 0 或 1 都直接压栈;
2. 继续压栈状态 q_0, 则对不同输入 (0/1) 和不同栈顶 (0/1), 都直接压栈;
3. 非确定的转到弹栈状态 q_1, 不论栈顶是 Z_0, 0, 或 1, 开始匹配后半部分;
4. 保持弹栈状态 q_1, 弹出的栈顶符号必须和输入一致;
5. 扫描到串结尾且只有看到栈底符号了, 才允许转移到接受状态 q_2.

6.1.2 瞬时描述和转移符号

定义. *为形式描述 PDA 在一个给定瞬间的格局 (Configuration), 定义 $Q \times \Sigma^* \times \Gamma^*$ 中三元组*

$$(q, w, \gamma)$$

为瞬时描述 (ID, Instantaneous Description), 表示此时 PDA 处于状态 q, 输入带上剩余输入串 w, 栈中的符号串为 γ.

定义. *在 PDA P 中如果 $(p, \beta) \in \delta(q, a, Z)$, 由 $(q, aw, Z\alpha)$ 到 $(p, w, \beta\alpha)$ 的变化, 称为 ID 的转移 $\vdash_P$, 记为*

$$(q, aw, Z\alpha) \vdash_P (p, w, \beta\alpha)$$

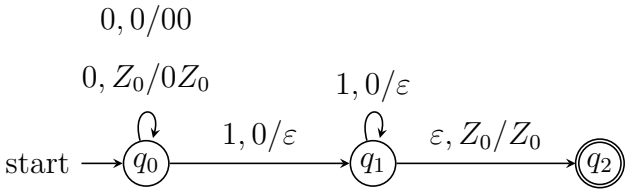
其中 $w \in \Sigma^*$, $\alpha \in \Gamma^*$.

若有 ID I, J 和 K, 递归定义 $\vdash_P^*$ 为:

1. $I \vdash_P^* I$;
2. *若 $I \vdash_P J$, $J \vdash_P^* K$, 则 $I \vdash_P^* K$.*

若 P 已知, 可省略, 记为 $\vdash$ 和 $\vdash^*$.

续例 1. 语言 $L_{01} = \{0^n 1^n \mid n \geq 1\}$ 的 PDA, 识别 0011 时的 ID 序列.



有关 ID 的序列

对 PDA P 的一个合法 ID 序列 (计算):

1. 把相同的字符串加到所有 ID 的输入串末尾, 所得到的计算合法;
2. 把相同的栈符号串加到所有 ID 的栈底之下, 所得到的计算合法;
3. 把所有 ID 中都未消耗的部分输入串去掉, 所得到的计算合法.

定理 23. 对 $\forall w \in \Sigma^*, \forall \gamma \in \Gamma^*$, 如果

$$(q, x, \alpha) \vdash_P^* (p, y, \beta),$$

那么

$$(q, xw, \alpha\gamma) \vdash_P^* (p, yw, \beta\gamma).$$

定理 24. 对 $\forall w \in \Sigma^*$, 如果

$$(q, xw, \alpha) \vdash_P^* (p, yw, \beta),$$

那么

$$(q, x, \alpha) \vdash_P^* (p, y, \beta).$$

6.2 下推自动机接受的语言

定义. *PDA* $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, 以两种方式接受语言:

- P 以终态方式接受的语言, 记为$\mathbf{L}(P)$, 定义为

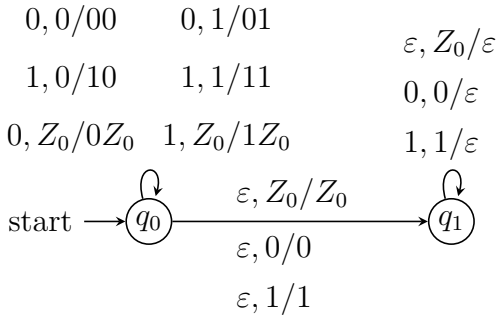
$$\mathbf{L}(P) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \gamma),\ p \in F\}.$$

- P 以空栈方式接受的语言, 记为$\mathbf{N}(P)$, 定义为

$$\mathbf{N}(P) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon)\}.$$

续例 2. 识别 L_{wwr} 的 PDA P, 从终态方式接受, 改为空栈方式接受.

用 $\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$ 代替 $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$ 即可.

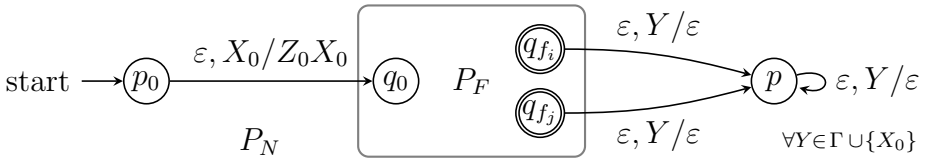


6.2.1 从终态方式到空栈方式

定理 25. *如果 PDA P_F 以终态方式接受语言 L, 则存在 PDA P_N 以空栈方式接受 L.*

证明: 设 $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$, 构造 PDA P_N,

$$P_N = (Q \cup \{p_0, p\},\ \Sigma,\ \Gamma \cup \{X_0\},\ \delta_N,\ p_0,\ X_0,\ \varnothing).$$



其中 δ_N 定义如下:

1. P_N 首先将 P_F 的栈底符号压栈, 开始模拟 P_F:

$$\delta_N(p_0, \varepsilon, X_0) = \{(q_0, Z_0X_0)\};$$

2. P_N 模拟 P_F 的动作: $\forall q \in Q,\ \forall a \in \Sigma \cup \{\varepsilon\},\ \forall Y \in \Gamma$:

$$\delta_N(q, a, Y) \text{ 包含 } \delta_F(q, a, Y) \text{ 的全部元素;}$$

3. 从 $q_f \in F$ 开始弹出栈中符号, 即 $\forall q_f \in F, \forall Y \in \Gamma \cup \{X_0\}$:

$$\delta_N(q_f, \varepsilon, Y) \text{ 包含 } (p, \varepsilon);$$

4. 在状态 p 时, 弹出全部栈中符号, 即 $\forall Y \in \Gamma \cup \{X_0\}$:

$$\delta_N(p, \varepsilon, Y) = \{(p, \varepsilon)\}.$$

对 $\forall w \in \Sigma^*$ 有

$$w \in \mathbf{L}(P_F) \Rightarrow (q_0, w, Z_0) \overset{*}{\vdash}_{P_F} (q_f, \varepsilon, \gamma)$$

$$
\begin{aligned}
&\Rightarrow (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_F} (q_f, \varepsilon, \gamma X_0) && \text{定理23}\\
&\Rightarrow (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_N} (q_f, \varepsilon, \gamma X_0) && P_N\text{模拟}P_F\\
&\Rightarrow (p_0, w, X_0) \vdash_{P_N} (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_N} (q_f, \varepsilon, \gamma X_0) && \delta_N\text{构造}\, p_0\, \text{部分}\\
&\Rightarrow (p_0, w, X_0) \overset{*}{\vdash}_{P_N} (q_f, \varepsilon, \gamma X_0) \overset{*}{\vdash}_{P_N} (p, \varepsilon, \varepsilon) && \delta_N\text{构造}\, q_f\text{和}p\, \text{部分}\\
&\Rightarrow w \in \mathbf{N}(P_N)
\end{aligned}
$$

即 $\mathbf{L}(P_F) \subseteq \mathbf{N}(P_N)$.

对 $\forall w \in \Sigma^*$ 有

$$
\begin{aligned}
w \in \mathbf{N}(P_N) &\Rightarrow (p_0, w, X_0) \overset{*}{\vdash}_{P_N} (p, \varepsilon, \varepsilon) && \text{其他状态不可能空栈}\\
&\Rightarrow (p_0, w, X_0) \vdash_{P_N} (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_N} (p, \varepsilon, \varepsilon) && \text{第一个动作必然到}\, q_0\\
&\Rightarrow (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_N} (q_f, \varepsilon, \gamma X_0) \overset{*}{\vdash}_{P_N} (p, \varepsilon, \varepsilon) && \text{必经}\, q_f \in F\, \text{消耗完}\, w\\
&\Rightarrow (q_0, w, Z_0) \overset{*}{\vdash}_{P_N} (q_f, \varepsilon, \gamma) && P_N\, \text{中未用过栈底的}\, X_0\\
&\Rightarrow (q_0, w, Z_0) \overset{*}{\vdash}_{P_F} (q_f, \varepsilon, \gamma) && \text{均为模拟}\, P_F\\
&\Rightarrow w \in \mathbf{L}(P_F)
\end{aligned}
$$

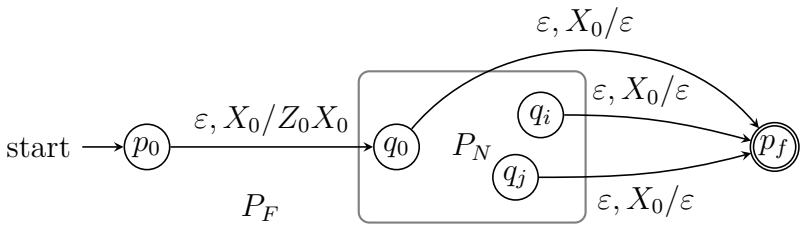
即 $\mathbf{N}(P_N) \subseteq \mathbf{L}(P_F)$. 所以 $\mathbf{N}(P_N) = \mathbf{L}(P_F)$. □

6.2.2 从空栈方式到终态方式

定理 26. *如果 PDA P_N 以空栈方式接受语言 L, 则存在 PDA P_F 以终态方式接受 L.*

证明: 设 $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0, \varnothing)$. 构造 PDA P_F,

$$P_F = (Q \cup \{p_0, p_f\},\ \Sigma,\ \Gamma \cup \{X_0\},\ \delta_F,\ p_0,\ X_0,\ \{p_f\})$$



其中 δ_F 定义如下:

1. P_F 开始时, 将 P_N 栈底符号压入栈, 并开始模拟 P_N,

$$\delta_F(p_0, \varepsilon, X_0) = \{(q_0, Z_0X_0)\};$$

2. P_F 模拟 P_N, $\forall q \in Q,\ \forall a \in \Sigma \cup \{\varepsilon\},\ \forall Y \in \Gamma$:

$$\delta_F(q, a, Y) = \delta_N(q, a, Y);$$

3. 在 $\forall q \in Q$ 时, 看到 P_F 的栈底 X_0, 则转移到新终态 p_f:

$$\delta_F(q, \varepsilon, X_0) = \{(p_f, \varepsilon)\}.$$

对 $\forall w \in \Sigma^*$ 有

$$\begin{aligned}
w \in \mathbf{N}(P_N) &\Rightarrow (q_0, w, Z_0) \overset{*}{\vdash}_{P_N} (q, \varepsilon, \varepsilon) \\
&\Rightarrow (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_N} (q, \varepsilon, X_0) && \text{定理23} \\
&\Rightarrow (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_F} (q, \varepsilon, X_0) && P_F \text{ 模拟 } P_N \\
&\Rightarrow (p_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_F} (q, \varepsilon, X_0) && \delta_F\text{构造, } p_0 \text{ 部分} \\
&\Rightarrow (p_0, w, X_0) \overset{*}{\vdash}_{P_F} (q, \varepsilon, X_0) \vdash_{P_F} (p_f, \varepsilon, \varepsilon) && \delta_F\text{构造, } p_f \text{ 部分} \\
&\Rightarrow (p_0, w, X_0) \overset{*}{\vdash}_{P_F} (p_f, \varepsilon, \varepsilon) \\
&\Rightarrow w \in \mathbf{L}(P_F)
\end{aligned}$$

即 $\mathbf{N}(P_N) \subseteq \mathbf{L}(P_F)$.

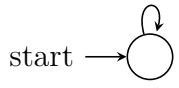
对 $\forall w \in \Sigma^*$ 有

$$\begin{aligned}
w \in \mathbf{L}(P_F) &\Rightarrow (p_0, w, X_0) \overset{*}{\vdash}_{P_F} (p_f, \varepsilon, \varepsilon) \\
&\Rightarrow (p_0, w, X_0) \overset{*}{\vdash}_{P_F} (q, \varepsilon, X_0) \vdash_{P_F} (p_f, \varepsilon, \varepsilon) && \text{经 } q \text{ 才可达 } p_f \\
&\Rightarrow (p_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_F} (q, \varepsilon, X_0) && P_F \text{ 第一个动作} \\
&\Rightarrow (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_F} (q, \varepsilon, X_0) && \text{即上式} \\
&\Rightarrow (q_0, w, Z_0) \overset{*}{\vdash}_{P_N} (q, \varepsilon, \varepsilon) && P_N \text{ 与 } X_0 \text{ 无关} \\
&\Rightarrow w \in \mathbf{N}(P_N)
\end{aligned}$$

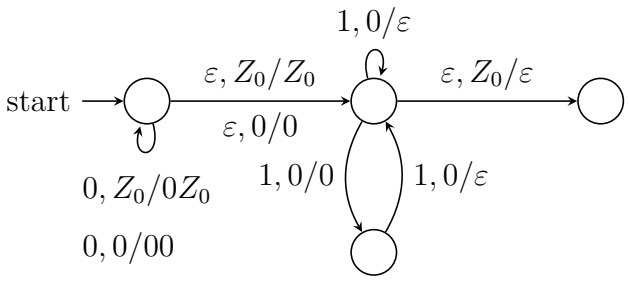
即 $\mathbf{N}(P_F) \subseteq \mathbf{L}(P_N)$. 所以 $\mathbf{L}(P_F) = \mathbf{N}(P_N)$. □

例 3. 接受 $L_{\text{eq}} = \{w \in \{0,1\}^* \mid w \text{ 中字符 } 0 \text{ 和 } 1 \text{ 的数量相同}\}$ 的 PDA.

$0, Z_0/0Z_0$	$1, 0/10$	$0, 0/00$
$1, Z_0/1Z_0$	$1, 1/11$	$0, 1/01$
$\varepsilon, Z_0/\varepsilon$	$1, 0/\varepsilon$	$0, 1/\varepsilon$



例 4. 接受 $L = \{0^n 1^m \mid 0 \le n \le m \le 2n\}$ 的 PDA.



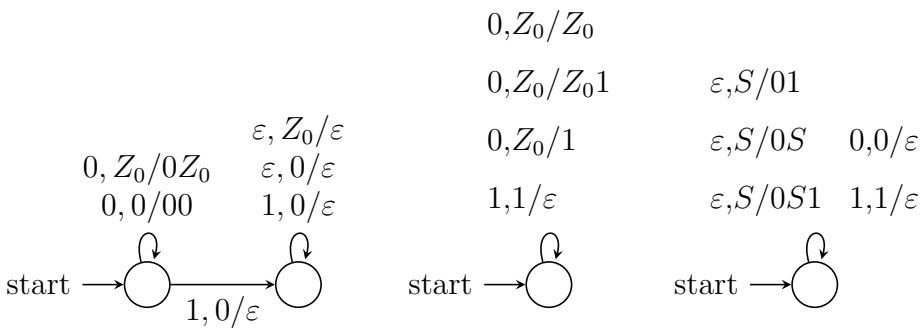
例. Design PDA for $L = \{a^i b^j c^k \mid i, j, k \geq 0,\ i = j \text{ or } j = k\}$.

例. Design PDA for the set of strings of 0's and 1's such that no prefix has more 1's than 0's.

6.3 下推自动机与文法的等价性

6.3.1 由 CFG 到 PDA

例 5. 设计语言 $L = \{0^n 1^m \mid 1 \leq m \leq n\}$ 的 PDA.



CFG G:

$$S \to AB$$
$$A \to 0A \mid \varepsilon$$
$$B \to 0B1 \mid 01$$

字符串 00011 的最左派生:

$$S \underset{\text{lm}}{\Rightarrow} AB \underset{\text{lm}}{\Rightarrow} 0AB \underset{\text{lm}}{\Rightarrow} 0B \underset{\text{lm}}{\Rightarrow} 00B1 \underset{\text{lm}}{\Rightarrow} 00011$$

用 PDA 栈顶符号的替换, 模拟文法的最左派生:

PDA				CFG	
PDA 的 ID 转移			PDA 的动作	产生式	最左派生
$(q_0,$	$00011,$	$S)$			S
$\vdash (q_0,$	$00011,$	$AB)$	$\varepsilon, S/AB$	$S \to AB$	$\underset{\text{lm}}{\Rightarrow} AB$
$\vdash (q_0,$	$00011,$	$0AB)$	$\varepsilon, A/0A$	$A \to 0A$	$\underset{\text{lm}}{\Rightarrow} 0AB$
$\vdash (q_0,$	$0011,$	$AB)$	$0, 0/\varepsilon$		
$\vdash (q_0,$	$0011,$	$B)$	$\varepsilon, A/\varepsilon$	$A \to \varepsilon$	$\underset{\text{lm}}{\Rightarrow} 0B$
$\vdash (q_0,$	$0011,$	$0B1)$	$\varepsilon, B/0B1$	$B \to 0B1$	$\underset{\text{lm}}{\Rightarrow} 00B1$
$\vdash (q_0,$	$011,$	$B1)$	$0, 0/\varepsilon$		
$\vdash (q_0,$	$011,$	$011)$	$\varepsilon, B/01$	$B \to 01$	$\underset{\text{lm}}{\Rightarrow} 00011$
$\vdash (q_0,$	$11,$	$11)$	$0, 0/\varepsilon$		
$\vdash (q_0,$	$1,$	$1)$	$1, 1/\varepsilon$		
$\vdash (q_0,$	$\varepsilon,$	$\varepsilon)$	$1, 1/\varepsilon$		

$S \to AB$

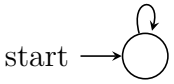
$A \to 0A \mid \varepsilon$

$B \to 0B1 \mid 01$

$\varepsilon, S/AB$

$\varepsilon, A/0A \quad \varepsilon, A/\varepsilon \quad 0, 0/\varepsilon$

$\varepsilon, B/0B1 \quad \varepsilon, B/01 \quad 1, 1/\varepsilon$



想要证明 CFG 和 PDA 的等价性, 需要思考如何使用 PDA 模拟文法的推导. 对任意属于某 CFL 的串 w, 其文法的推导过程, 就是使用产生式去匹配 (产生) w, 如果 w 放在某 PDA 的输入带上, 我们的目的就是通过文法构造动作, 让 PDA 能从左到右的扫描输入串, 利用栈来模拟文法的最左派生过程即可.

定理 27. *任何 CFL L, 一定存在 PDA P, 使 $L = \mathbf{N}(P)$.*

构造与文法等价的 PDA

如果 CFG $G = (V, T, P', S)$, 构造 PDA

$$P = (\{q\}, T, V \cup T, \delta, q, S, \varnothing),$$

其中 δ 为:

1. $\forall A \in V$:

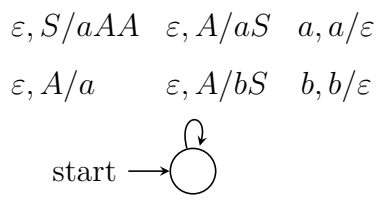
$$\delta(q, \varepsilon, A) = \{(q, \beta) \mid A \to \beta \in P'\}$$

2. $\forall a \in T$:

$$\delta(q, a, a) = \{(q, \varepsilon)\}$$

那么 $\mathbf{L}(G) = \mathbf{N}(P)$.

例 6. 为文法 $S \to aAA$, $A \to aS \mid bS \mid a$ 构造 PDA.



证明:

PDF P 可以模拟 CFG G 的最左派生, 每个动作只根据栈顶的符号确定: 如果是终结符则与输入串匹配, 如果是非终结符用产生式来替换. 为了完成定理, 只需往证

$$S \overset{*}{\Rightarrow} w \iff (q, w, S) \overset{*}{\vdash_{\mathrm{P}}} (q, \varepsilon, \varepsilon).$$

最左派生 $S \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} w$ 的每个左句型都可写作 $xA\alpha$ 的形式, 其中 A 是最左变元, x 是它之前的所有终结符串, 而 α 是它右边的符号串. 而在 P 中, 每个左句型的 $A\alpha$ 部分都会出现在栈中, 并且当 A 处于栈顶时, x 刚好是输入带上被扫描过的 (消耗完的) 部分. 那么当 PDA 处于 ID $(q, y, A\alpha)$ 时, 刚好有 $xy = w$ 成立.

[充分性] 往证

$$S \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} w \implies (q, w, S) \overset{*}{\vdash} (q, \varepsilon, \varepsilon).$$

设 $S \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} w$ 中第 i 个左句型为 $x_i A_i \alpha_i$, 其中 $x_i \in \Sigma^*$, $A_i \in V$, $\alpha_i \in (V \cup T)^*$. 并将 S 看作第 0 个左句型 $x_0 A_0 \alpha_0 = S$, 那么

$$x_0 = \varepsilon, A_0 = S, \alpha_0 = \varepsilon.$$

将 w 看作为第 n 个左句型 $x_n A_n \alpha_n = w$, 那么

$$x_n = w, A_n = \varepsilon, \alpha_n = \varepsilon.$$

再对派生步骤 i 归纳, 往证

$$S \overset{i}{\underset{\mathrm{lm}}{\Rightarrow}} x_i A_i \alpha_i \wedge w = x_i y_i \Longrightarrow (q, w, S) \overset{*}{\vdash} (q, y_i, A_i \alpha_i).$$

归纳基础: 最左派生在第 0 步时, 显然成立

$$(q, w, S) \overset{*}{\vdash} (q, y_0, A_0 \alpha_0) = (q, w, S).$$

归纳递推: 假设第 i 步时成立, 当第 $i+1$ 步时, 一定是 $A_i \to \beta$ 应用到 $x_i A_i \alpha_i$

$$S \overset{i}{\underset{\mathrm{lm}}{\Rightarrow}} x_i A_i \alpha_i \underset{\mathrm{lm}}{\Rightarrow} x_i \beta \alpha_i = x_{i+1} A_{i+1} \alpha_{i+1}.$$

即第 $i+1$ 个左句型的最左变元 A_{i+1} 一定在 $\beta\alpha_i$ 中, 设 A_{i+1} 之前的终结符为 x', 那么由

$$\begin{aligned} x_i\beta\alpha_i &= x_i x' A_{i+1}\alpha_{i+1} &&= x_{i+1}A_{i+1}\alpha_{i+1} \\ x_i y_i &= x_i x' y_{i+1} &&= x_{i+1}y_{i+1} = w \end{aligned}$$

则有

$$\begin{aligned} \beta\alpha_i &= x' A_{i+1}\alpha_{i+1}, \\ y_i &= x' y_{i+1}. \end{aligned}$$

那么, 在 PDA 中从 ID $(q, y_i, A_i\alpha_i)$ 模拟最左派生, 用产生式 $A_i \to \beta$ 替换栈顶 A_i 后, 有

$$\begin{aligned} (q, w, S) &\vdash^* (q, y_i, A_i\alpha_i) && \text{归纳假设} \\ &\vdash (q, y_i, \beta\alpha_i) && A_i \to \beta \\ &= (q, x'y_{i+1}, x'A_{i+1}\alpha_{i+1}) \\ &\vdash^* (q, y_{i+1}, A_{i+1}\alpha_{i+1}) && \text{弹出栈顶终结符} \end{aligned}$$

因此 $S \overset{n}{\underset{\text{lm}}{\Rightarrow}} w \Longrightarrow (q, w, S) \vdash^* (q, y_n, A_n\alpha_n) = (q, \varepsilon, \varepsilon)$, 即充分性得证.

[必要性] 往证更一般的, 对任何变元 A, 都有:

$$(q, x, A) \vdash^* (q, \varepsilon, \varepsilon) \Longrightarrow A \overset{*}{\Rightarrow} x.$$

这个过程, 可以看作 “从输入带中消耗掉 x” 与 “从栈中弹出 A” 两种作用相互抵消. 对 ID 转移 $(q, x, A) \vdash^i (q, \varepsilon, \varepsilon)$ 的次数 i 归纳证明.

归纳基础: 当 $i = 1$ 步时, 只能是 $x = \varepsilon$ 且 $A \to \varepsilon$ 为产生式, 所以 $A \overset{*}{\Rightarrow} \varepsilon$. 因为即使 $x = a$ 和产生式 $A \to a$, 也需要先替换栈顶 A 为 a 再弹出 a 两步才能清空栈.

归纳递推: 假设 $i \le n$ $(n \ge 1)$ 步时上式成立. 当 $i = n + 1$ 时, 因为 A 是变元, 其第 1 步转移一定是应用某产生式 $A \to Y_1Y_2\cdots Y_m$

$$(q, x, A) \vdash (q, x, Y_1Y_2\cdots Y_m)$$

其中 Y_i 是变元或终结符. 而其余的 n 步转移

$$(q, x, Y_1Y_2\cdots Y_m) \vdash^* (q, \varepsilon, \varepsilon)$$

中每个 Y_i 从栈中被完全弹出时, 将消耗掉的那部分 x 记为 x_i, 那么显然有

$$x = x_1x_2\cdots x_m.$$

而每个 Y_i 从栈中被完全弹出时, 都不超过 n 步, 所以由归纳假设,

$$(q, x_i, Y_i) \vdash^* (q, \varepsilon, \varepsilon) \Longrightarrow Y_i \overset{*}{\Rightarrow} x_i.$$

再由产生式 $A \to Y_1Y_2\cdots Y_m$, 有

$$\begin{aligned} A &\Rightarrow Y_1Y_2\cdots Y_m \\ &\overset{*}{\Rightarrow} x_1Y_2\cdots Y_m \\ &\overset{*}{\Rightarrow} x_1x_2\cdots Y_m \\ &\overset{*}{\Rightarrow} x_1x_2\cdots x_m = x. \end{aligned}$$

因此当 $A = S,\ x = w$ 时,

$$(q, w, S) \overset{*}{\vdash} (q, \varepsilon, \varepsilon) \Longrightarrow S \overset{*}{\Rightarrow} w$$

成立, 即必要性得证.

所以, 任何 CFL 都可由 PDA 识别. □

构造与 GNF 格式文法等价的 PDA

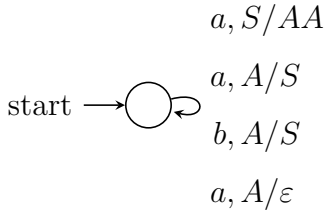
如果 GNF 格式的 CFG $G = (V, T, P', S)$, 那么构造 PDA

$$P = (\{q\},\ T,\ V,\ \delta,\ q,\ S,\ \varnothing),$$

为每个产生式, 定义 δ 为:

$$\delta(q, a, A) = \{(q, \beta) \mid A \to a\beta \in P'\}.$$

续例 6. 文法 $S \to aAA,\ A \to aS \mid bS \mid a$ 为 GNF 格式, 构造等价的 PDA.



6.3.2 由 PDA 到 CFG

定理 28. *如果 PDA P, 有 $L = \mathbf{N}(P)$, 那么 L 是上下文无关语言.*

构造与 PDA 等价的 CFG

如果 PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$, 那么构造 CFG $G = (V, \Sigma, P', S)$, 其中 V 和 P' 为

1. $V = \{[qXp] \mid p,q \in Q,\ X \in \Gamma\} \cup \{S\}$;
2. 对 $\forall p \in Q$, 构造产生式 $S \to [q_0 Z_0 p]$;
3. 对 $\forall (p, Y_1 Y_2 \cdots Y_n) \in \delta(q, a, X)$, 构造 $|Q|^n$ 个产生式

$$[qXr_n] \to a[pY_1r_1][r_1Y_2r_2]\cdots[r_{n-1}Y_nr_n]$$

其中 $a \in \Sigma \cup \{\varepsilon\}$, X,$Y_i \in \Gamma$, 而 $r_i \in Q$ 是 n 次 $|Q|$ 种状态的组合; 若 $i = 0$, 为 $[qXp] \to a$.

证明: 通过以上方法所构造的文法中, $[qXp] \overset{*}{\Rightarrow} w$ 的含义, 可以理解为 PDA 从状态 q 将栈符号 X 完全弹出后, 转移到了状态 p, 同时从输入带上消耗掉了 w. 那么, 只需证明

$$(q, w, X) \overset{*}{\vdash} (p, \varepsilon, \varepsilon) \iff [qXp] \overset{*}{\Rightarrow} w.$$

并令 $X = Z_0$, $q = q_0$, 与开始符号 S 的产生式一起, 即可完成定理的证明.

[充分性] 对 PDA 中 $(q, w, X) \overset{*}{\vdash} (p, \varepsilon, \varepsilon)$ 的转移次数 i 归纳证明.

归纳基础: 当 $i = 1$ 时, P 的输入带上只能消耗不超过一个的字符, 即 $w = a$

$$(q, w, X) = (q, a, X) \vdash (p, \varepsilon, \varepsilon),$$

其中 $a \in \Sigma \cup \{\varepsilon\}$ 且 $(p, \varepsilon) \in \delta(q, a, X)$, 则由文法的构造会有

$$[qXp] \to a,$$

因此 $[qXp] \overset{*}{\Rightarrow} a = w$.

归纳递推: 假设当 $i \le m$ $(m \ge 1)$ 时命题成立. 那么, 当 $i = m + 1$ 时, 转移的第 1 步, 一定由某个 $(r_0, Y_1 Y_2 \cdots Y_n) \in \delta(q, a, X)$ 开始

$$(q, ax, X) \vdash (r_0, x, Y_1 Y_2 \cdots Y_n),$$

其中 $a \in \Sigma \cup \{\varepsilon\}, w = ax$. 而其余的 m 步为

$$(r_0, x, Y_1 Y_2 \cdots Y_n) \overset{*}{\vdash} (p, \varepsilon, \varepsilon).$$

而这些转移, 会从栈中依次弹出 Y_i, 并相应的消耗掉输入带上的部分 x. 若分别记为 x_i, 则有

$$w = ax = ax_1x_2\cdots x_n.$$

若设从栈中完全弹出 Y_i(并消耗掉 x_i) 之前和之后的状态分别是 r_{i-1} 和 r_i, 这里 $i = 1, 2, \cdots n$, 那么有

$$(r_{i-1}, x_i, Y_i) \overset{*}{\vdash} (r_i, \varepsilon, \varepsilon),$$

且转移步数都不会超过 m. 那么, 由归纳假设有

$$(r_{i-1}, x_i, Y_i) \overset{*}{\vdash} (r_i, \varepsilon, \varepsilon) \implies [r_{i-1}Y_ir_i] \overset{*}{\Rightarrow} x_i.$$

而由动作 $(r_0, Y_1Y_2\cdots Y_n) \in \delta(q, a, X)$ 所构造的产生式会包含

$$[qXr_n] \to a[r_0Y_1r_1][r_1Y_2r_2]\cdots[r_{n-1}Y_nr_n].$$

而显然弹出 X 后的状态 p 与弹出 Y_n 后的状态 r_n 是同一个, 所以

$$[qXp] = [qXr_n] \Rightarrow a[r_0Y_1r_1][r_1Y_2r_2]\cdots[r_{n-1}Y_nr_n] \overset{*}{\Rightarrow} ax_1x_2\cdots x_n = w$$

因此充分性得证. 那么当 $X = Z_0, q = q_0$ 时有

$$(q_0, w, Z_0) \overset{*}{\vdash} (p, \varepsilon, \varepsilon) \Longrightarrow [q_0Z_0p] \overset{*}{\Rightarrow} w,$$

以及产生式 $S \to [q_0Z_0p]$ 有 $S \overset{*}{\Rightarrow} w$, 即 PDA 接受的串可由文法派生得到.

[必要性]: 略. □

例 7. 将 PDA $P = (\{p, q\}, (0, 1), \{X, Z\}, \delta, q, Z)$ 转为 CFG, 其中 δ 如下:

(1) $\delta(q, 1, Z) = \{(q, XZ)\}$
(2) $\delta(q, 1, X) = \{(q, XX)\}$
(3) $\delta(q, 0, X) = \{(p, X)\}$
(4) $\delta(q, \varepsilon, Z) = \{(q, \varepsilon)\}$
(5) $\delta(p, 1, X) = \{(p, \varepsilon)\}$
(6) $\delta(p, 0, Z) = \{(q, Z)\}$

解:

δ	产生式
(0)	$S \to [qZq]$
	$S \to [qZp]$
(1)	$[qZq] \to 1[qXq][qZq]$
	$[qZq] \to 1[qXp][pZq]$
	$[qZp] \to 1[qXq][qZp]$
	$[qZp] \to 1[qXp][pZp]$
(2)	$[qXq] \to 1[qXq][qXq]$
	$[qXq] \to 1[qXp][pXq]$
	$[qXp] \to 1[qXq][qXp]$
	$[qXp] \to 1[qXp][pXp]$
(3)	$[qXq] \to 0[pXq]$
	$[qXp] \to 0[pXp]$
(4)	$[qZq] \to \varepsilon$
(5)	$[pXp] \to 1$
(6)	$[pZp] \to 0[qZp]$
	$[pZq] \to 0[qZq]$

消除无用符号	重命名 (可选)
$S \to [qZq]$	$S \to A$
$[qZq] \to 1[qXp][pZq]$	$A \to 1BC$
$[qXp] \to 1[qXp][pXp]$	$B \to 1BD$
$[qXp] \to 0[pXp]$	$B \to 0D$
$[qZq] \to \varepsilon$	$A \to \varepsilon$
$[pXp] \to 1$	$D \to 1$
$[pZq] \to 0[qZq]$	$C \to 0A$

6.4 确定型下推自动机

定义. 如果 *PDA* $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ 满足

1. $\forall a \in \Sigma \cup \{\varepsilon\}$, $\delta(q, a, X)$ 至多有一个动作;
2. $\exists a \in \Sigma$, 如果 $\delta(q, a, X) \neq \varnothing$, 那么 $\delta(q, \varepsilon, X) = \varnothing$.

则称 P 为确定型下推自动机 *(DPDA).*

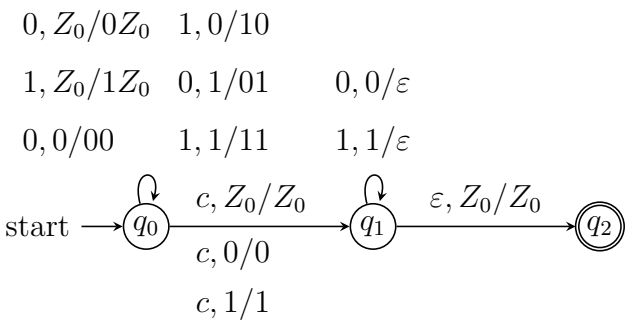
DPDA P 以终态方式接受的语言 $\mathbf{L}(P)$ 称为 *DCFL.*

- DPDA 中 $\forall (q, a, Z) \in Q \times \Sigma \times \Gamma$ 满足 $|\delta(q, a, Z)| + |\delta(q, \varepsilon, Z)| \leq 1$

DPDA 与 PDA 不等价

例 8. 任何 DPDA 都无法接受 L_{wwr}, 但是可以接受

$$L_{wcwr} = \{wcw^R \mid w \in (\mathbf{0} + \mathbf{1})^*\}.$$



DPDA 无法识别 L_{wwr}. 因为, 如果它想识别输入串 0^n110^n, 需要利用栈保存 n 个 0, 再根据 11 改变状态, 将栈弹出. 而这时如果输入带上还有 0^n110^n, 那么它还应该接受. 但如果接受 $0^n110^n0^n110^n$, 它同样会接受 $0^n110^n0^m110^m$. DPDA 使用栈无法同时记住 0^n 和 0^n110^n.

6.4.1 正则语言与 DPDA

定理 29. *如果 L 是正则语言, 那么存在 DPDA P 以终态方式接受 L, 即 $L = \mathbf{L}(P)$.*

证明: 显然, 因为 DPDA P 可以不用栈而模拟任何 DFA. □

- L_{wcwr} 显然是 CFL, 所以 DCFL 语言类真包含正则语言
- DPDA 无法识别 L_{wwr}, 所以 DCFL 语言类真包含于 CFL

定义. *如果语言 L 中不存在两个不同的字符串 x 和 y, 使 x 是 y 的前缀, 称语言 L 满足前缀性质.*

定理 30. *如果有 DPDA P 且 $L = \mathbf{N}(P)$, 当且仅当 L 有前缀性质且存在 DPDA P' 使 $L = \mathbf{L}(P')$.*

证明: $[\Rightarrow]$ $\forall x \in \mathbf{N}(P)$ 会弹空 P 的栈, 所以不会接受以 x 为前缀的其他串; 而转换为终态方式不改变确定性. $[\Leftarrow]$ 到达终态则弹空栈, 即可. □

- DPDA P 的 $\mathbf{N}(P)$ 更有限, 即使正则语言 $\mathbf{0}^*$ 也无法接受

DPDA P 若以空栈方式接受, 能够接受的语言更有限, 仅能接受具有前缀性质的语言. 前缀性质是指, 这个语言中不存在不同的串 x 和 y 使 x 是 y 的前缀. 即使正则语言 $\mathbf{0}^*$ 也无法接受, 因为其任何两个串中都有一个是前缀. 但以空栈方式接受的语言, 却可以被另一个 DPDA 以终态方式接受.

6.4.2 DPDA 与无歧义文法

定理 31. *DPDA P, 语言 $L = \mathbf{N}(P)$, 那么 L 有无歧义的 CFG.*

证明: 利用定理 28 由 P 构造的文法 G 一定无歧义, 因为:

1. P 是确定的, 那么它接受 w 的 ID 序列也是确定的;
2. 而由 $\delta(q, a, X) = \{(p, Y_1 \cdots Y_n)\}$ 继续弹出 Y_i 后的状态 r_i 也是确定的;
3. 那么由每个动作构造的一组产生式

   $$[qXr_n] \to a[pY_1r_1][r_1Y_2r_2] \cdots [r_{n-1}Y_nr_n]$$

   中, 仅会有一个是有效的;
4. 那么, G 中最左派生 $S \underset{\text{lm}}{\overset{*}{\Rightarrow}} w$ 就是唯一的, 所以是无歧义的. □

定理 32. *DPDA P, 语言 $L = \mathbf{L}(P)$, 那么 L 有无歧义的 CFG.*

证明:

1. 设符号 \$ 不在 L 中出现, 令 $L' = \{w\$ \mid w \in L\}$, 则 L' 具有前缀性质;
2. 可修改 P 接受 L', 则由定理 30, 存在 DPDA P' 使 $\mathbf{N}(P') = L'$;
3. 由定理 31, 存在无歧义文法 G' 使 $\mathbf{L}(G') = L'$;
4. 将 \$ 看作变元, 增加产生式 $\$ \to \varepsilon$, 修改 G' 为文法 G;
5. 则文法 G 和 G' 一样无歧义, 且 $\mathbf{L}(G) = L$. □

DCFL/DPDA 的重要应用

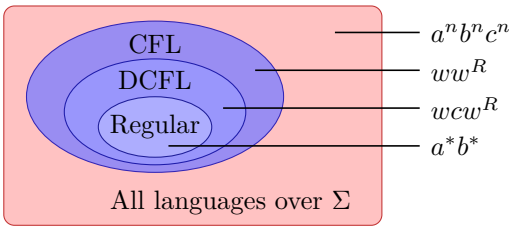
- 程序设计语言的语法分析器

  如 LR(k) 文法, `Yacc` 的基础, 解析的时间复杂度为 $O(n)$ 的算法
- 非固有歧义语言的真子集

  如 L_{wwr} 有无歧义文法 $S \to 0S0 \mid 1S1 \mid \varepsilon$

在任何情况下都不需要去选择可能的移动就是 DPDA, 以终态方式接受的语言也称为 DCFL. 虽然与 PDA 不等价, 但也有意义, 例如语法分析器通常都是 DPDA, DPDA 接受的语言是非固有歧义语言的真子集, Knuth 提出 LR(k) 文法的语言也恰好是 DPDA 接受语言的一个子集, 解析的时间复杂度为 $O(n)$, LR(k) 文法也是 YACC 的基础.

语言类之间的关系



Chapter 7

上下文无关语言的性质

7.1 上下文无关语言的泵引理

直观来讲, 正则语言是从一些字符串以*并 (union)*, *连接 (concatenation)* 和*重复 (repetition)* 构建而来的; 而上下文无关语言则是以*并*, *连接*和*递归 (recursion)* 构建而来的.

7.1.1 上下文无关语言的泵引理

任何 Σ 上的所有语言是不可数的

不妨设 $\Sigma = \{a\}$, 对任何 $0 \le x < 1$ 的实数 x, 定义语言

$$L_x = \{a^n \mid x \cdot 2^n \bmod 1 \ge 1/2\},$$

即 $a^n \in L_x$ 当且仅当 x 二进制表示的第 $n+1$ 位为 1.

1. 如果 $x \neq y$, 则 x 和 y 一定有某些位不同, 所以 $L_x \neq L_y$;
2. 所以 Σ 上的所有语言, 至少与 0 和 1 之间的实数一样多;
3. 因此, Σ 上的所有语言是不可数的.

任何 Σ 上的所有 CFL 是可数的

任何 CFG $G = (V, \Sigma, P, S)$ 可由符号集 $V \cup \Sigma \cup \{\varepsilon, \to, |, \lozenge\}$ 编码.

- 如文法 $S \to A \mid B,\ A \to aA \mid aC,\ B \to Bb \mid Cb,\ C \to \varepsilon \mid aCb$ 可编码为

$$S \to A|B \lozenge A \to aA|aC \lozenge B \to Bb|Cb \lozenge C \to \varepsilon|aCb;$$

- 用 0/1 编码这些符号

$$\begin{array}{lll} \varepsilon \mapsto 10 & a \mapsto 110 & S \mapsto 1110 \\ \to \mapsto 100 & b \mapsto 1100 & A \mapsto 11100 \\ \mid \mapsto 1000 & & B \mapsto 111000 \\ \Diamond \mapsto 10000 & & C \mapsto 1110000. \end{array}$$

- 文法编码再转换为 0/1 字符串

$$\begin{aligned} &11101001110010001110001000011100100110111001000110 \\ &11100001000011100010011100011001000111000011001000 \\ &01110000100101000110111000011 00; \end{aligned}$$

- 当作二进制表示则为整数

$$2486025347845581444133243339142670726924.$$

- 而 Σ 上两个文法如果不同, 这样编码会得到不同的整数;
- 因此 Σ 上所有 CFL 至多与正整数一样多, 是可数的.
- 因此, 并非所有的语言都是 CFL.

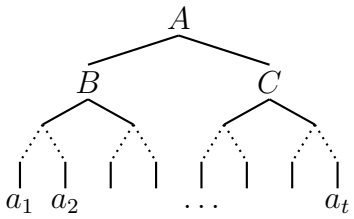
语法分析树的大小

定理 33. *对于乔姆斯基范式文法 $G = (V, T, P, S)$ 的语法树, 如果产物为终结符串 w, 且树中最长路径的长度是 n, 那么 $|w| \leq 2^{n-1}$.*

证明: 对最长路径的长度归纳.

归纳基础: 为 1 时, 只能是 $\begin{array}{c} A \\ \mid \\ a \end{array}$, 显然成立.

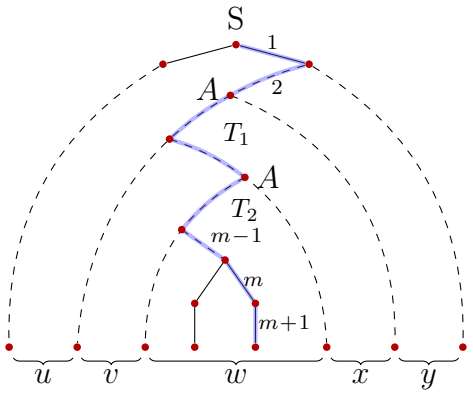
归纳递推: 为 n 时根节点一定是 $A \to BC$, 而 B 和 C 子树最长路径最多为 $n-1$, 由归纳假设, 产物最长都为 2^{n-2}. 因此整棵树产物最长 $2^{n-2} + 2^{n-2} = 2^{n-1}$. □



上下文无关语言的泵引理

定理 34. *如果语言 L 是 CFL, 那么存在正整数 N, 它只依赖于 L, 对 $\forall z \in L$, 只要 $|z| \geq N$, 就可以将 z 分为五部分 $z = uvwxy$ 满足:*

1. $vx \neq \varepsilon$ (或$|vx| > 0$);
2. $|vwx| \leq N$;
3. $\forall i \geq 0$, $uv^i wx^i y \in L$.



证明:

1. 设 CNF 格式的 CFG G 接受语言 $L - \{\varepsilon\}$. 在 CNF 文法的派生树中, 如果从树根到叶子的最长路径长度为 k, 则该树产物的长度最多为 2^{k-1}. 因为该树内节点构成的是二叉树, 当最长路径为 k 时, 内节点二叉树部分最长路径长度为 $k-1$, 叶子最多为满二叉树时的 2^{k-1} 个. 如果设文法 G 中变元数 $|V| = m$, 则令 $N = 2^m$, 那么若有字符串 $z \in L(G)$, 且 $|z| \geq N$.
2. 则 z 的派生树内节点是二叉树, 树中最长路径长度至少也是 $m+1$, 该路径上节点至少有 $m+2$ 个, 除最后一个节点外, 其余标记都是变元.
3. 在该路径上接近树底部由下至上 $m+1$ 个内节点的标记中, 必有两个节点 T_2 和 T_1 标记了相同的变元 A. 不妨设这两棵 A 子树分别为 T_1 和 T_2, 且 T_1 比 T_2 更接近树根.
4. 那么, 若记 A 子树 T_2 的产物为 w, 且又因为它是 T_1 的子树, 那么 T_1 的产物可记为 vwx, 则有 $A \overset{*}{\Rightarrow} vAx$ 和 $A \overset{*}{\Rightarrow} w$.
5. 那么, 对 $\forall i \geq 0$, 有 $A \overset{*}{\Rightarrow} v^i wx^i$. 又因为 vwx 是 z 的一部分, 所以不妨设 $z = uvwxy$, 则 $S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uv^i wx^i y$, 即 $\forall i \geq 0, uv^i wx^i y \in L$.
6. 又因为, 我们只考虑了接近树底部的 $m+1$ 个节点, 所以 T_1 子树中最长路径长度不超过 $m+1$, 那么 T_1 产物的长度不会超过 2^m, 所以 $|vwx| \leq 2^m = N$.

7. 而 T_1 子树派生 vwx 的第一个产生式必是 $A \to BC$, 那么 T_2 子树不是完全处于 B 中就是完全处于 C 中, 即 T_2 必定完全在 T_1 的左/右儿子中. 而不包括 T_2 的变元 B 或 C 都至少产生一个终结符, 所以 v 和 x 不可能同时为空, 即 $vx \neq \varepsilon$.

□

7.1.2 泵引理的应用

例 1. 证明 $L = \{0^n 1^n 2^n \mid n \geq 1\}$ 不是上下文无关语言.

证明:

1. 假设 L 是 CFL, 那么存在整数 N, 对 $\forall z \in L\,(|z| \geq N)$ 满足泵引理.
2. 从 L 中取 $z = 0^N 1^N 2^N$, 则显然 $z \in L$ 且 $|z| = 3N \geq N$.
3. 由泵引理, z 可被分为 $z = uvwxy$, 且有 $|vwx| \leq N$ 和 $vx \neq \varepsilon$.
4. 那么 vwx 只能包含一种或两种字符:
   - i 一种字符, 或为 0, 或为 1, 或为 2, 那么取 $i = 0$, 则 uv^0wx^0y 会只删除该字符的一部分, 所以 $uwy \notin L$;
   - ii 两种字符, 或为 0 和 1, 或为 1 和 2, 那么uv^0wx^0y 会只删除这两种字符的一部分, 所以也有 $uwy \notin L$;
5. 与泵引理 $uwy = uv^0wx^0y \in L$ 矛盾, 所以假设不成立.
6. L 不是上下文无关的. □

例 2. 证明 $L = \{ww \mid w \in \{0,1\}^*\}$ 不是上下文无关的.

(错误的) 证明: 假设 L 是 CFL. 取 $z = 0^N 10^N 1$, 那么 $z = uvwxy$ 为

$$z = \overbrace{\underbrace{00\cdots00}_{u}\underbrace{0}_{v}\underbrace{1}_{w}}^{0^N1}\overbrace{\underbrace{0}_{x}\underbrace{00\cdots01}_{y}}^{0^N1}$$

则对任意 $i \geq 0$, 有 $uv^iwx^iy \in L$, 满足泵引理. □

(正确的) 证明: 假设 L 是 CFL. 取 $z = 0^N 1^N 0^N 1^N$, 将 z 分为 $z = uvwxy$ 时

1. 若 vwx 在 z 中点的一侧, uv^0wx^0y 显然不可能属于 L;
2. 若 vwx 包括 z 中点, 那么 uv^0wx^0y 为 $0^N 1^i 0^j 1^N$, 也不可能属于 L.

所以假设不成立, L 不是 CFL. □

CFL 的泵引理同样只是必要条件

有些非 CFL, 泵引理对它们没有什么作用. 例如

$$L = \{a^i b^j c^k d^l \mid i = 0 \text{ 或 } j = k = l\}$$

不是上下文无关的.

- 如果选 $z = b^j c^k d^l$, 则可以让 $z = uvwxy$ 的 vwx 只含有 b, 那么对任何 m, 都有 $uv^m wx^m y \in L$;
- 如果选 $z = a^i b^j c^j d^j$, 则可以让 v 和 x 只包含 a, 那么对任何 m, 都有 $uv^m wx^m y \in L$.

所以无法使用泵引理证明 L 非 CFL.

Ogden 引理 (的较弱形式)

如果语言 L 是 CFL, 那么存在正整数 N, 它只依赖于 L, 对 $\forall z \in L$, 在 z 中至少 N 个任意位置作标记后, 就可以将 z 分为五部分 $z = uvwxy$ 满足:

1. v 和 x 一起至少含有一个标记位置;
2. vwx 中至多有 N 个标记位置;
3. $\forall i \geq 0$, $uv^i wx^i y \in L$.

7.2 上下文无关语言的封闭性

7.2.1 代换的封闭性

定义. 两个字母表 Σ 到 Γ 的函数 $s : \Sigma \to 2^{\Gamma^*}$ 称为代换 *(substitution)*. Σ 中的一个字符 a 在 s 的作用下为 Γ 上的一个语言 L_a, 即

$$s(a) = L_a.$$

扩展 s 的定义到字符串,

$$s(\varepsilon) = \{\varepsilon\}$$
$$s(xa) = s(x)s(a)$$

再扩展 s 到语言, 对 $\forall L \subseteq \Sigma^*$,

$$s(L) = \bigcup_{x \in L} s(x).$$

定理 35. *如果有 Σ 上的 CFL L 和代换 s, 且每个 $a \in \Sigma$ 的 $s(a)$ 都是 CFL, 那么 $s(L)$ 也是 CFL.*

构造方法

设 CFL L 的文法 $G = (V, T, P, S)$, 每个 $s(a)$ 的文法 $G_a = (V_a, T_a, P_a, S_a)$. 那么 $s(L)$ 的文法可以构造为

$$G' = (V', T', P', S) :$$

1. $V' = V \cup \left(\bigcup_{a \in T} V_a\right)$
2. $T' = \bigcup_{a \in T} T_a$
3. P' 包括每个 P_a 和 P 中产生式, 但是要将 P 的产生式中每个终结符 a 均替换为文法 G_a 的开始符号 S_a.

证明: 对 $\forall w \in s(L)$, 那么一定存在某个 $x = a_1 a_2 \cdots a_n \in L$ 使

$$w \in s(x) = s(a_1)s(a_2)\cdots s(a_n).$$

那么 w 可以分为 $w = w_1 w_2 \cdots w_n$ 且 $w_i \in s(a_i)$, 即

$$S_{a_i} \overset{*}{\underset{G_{a_i}}{\Rightarrow}} w_i.$$

由于 $S \overset{*}{\underset{G'}{\Rightarrow}} x = a_1 a_2 \cdots a_n$, 所以

$$S \overset{*}{\underset{G'}{\Rightarrow}} S_{a_1} S_{a_2} \cdots S_{a_n} \overset{*}{\underset{G'}{\Rightarrow}} w_1 w_2 \cdots w_n = w,$$

所以 $w \in \mathbf{L}(G')$, 即 $s(L) \subseteq \mathbf{L}(G')$.

因为 G' 的终结符仅能由每个 S_a 派生, 因此对 $\forall w \in \mathbf{L}(G')$ 有

$$S \overset{*}{\underset{G'}{\Rightarrow}} \alpha = S_{a_1} S_{a_2} \cdots S_{a_n} \overset{*}{\underset{G'}{\Rightarrow}} w.$$

因为 G' 中的每个 S_a 在 G 中是终结符 a, 所以

$$S \overset{*}{\underset{G'}{\Rightarrow}} a_1 a_2 \cdots a_n = x \in L$$

又因为 $\alpha = S_{a_1} \cdots S_{a_n} \overset{*}{\underset{G'}{\Rightarrow}} w = w_1 \cdots w_n$, 所以 $S_{a_i} \overset{*}{\underset{G'}{\Rightarrow}} w_i$, 即 $w_i \in s(a_i)$. 那么

$$w = w_1 w_2 \cdots w_n \in s(a_1)s(a_2)\cdots s(a_n) = s(a_1 a_2 \cdots a_n) = s(x) \subseteq s(L),$$

所以 $w \in s(L)$, 即 $\mathbf{L}(G') \subseteq s(L)$. 因此 $\mathbf{L}(G') = s(L)$. □

例 3. 设 $L = \{w \in \{a, b\}^* \mid w$ 有相等个数的 a 和 $b\}$, 代换

$$s(a) = L_a = \{0^n 1^n \mid n \geq 1\}$$

$$s(b) = L_b = \{ww^R \mid w \in (\mathbf{0}+\mathbf{1})^*\}$$

求 $s(L)$ 的文法.

解: 设计 L 的文法为: $S \to aSbS \mid bSaS \mid \varepsilon$

L_a 的文法为: $S_a \to 0S_a1 \mid 01$

L_b 的文法为: $S_b \to 0S_b0 \mid 1S_b1 \mid \varepsilon$

那么 $s(L)$ 的文法为:
$$S \to S_aSS_bS \mid S_bSS_aS \mid \varepsilon$$
$$S_a \to 0S_a1 \mid 01$$
$$S_b \to 0S_b0 \mid 1S_b1 \mid \varepsilon$$

7.2.2 并/连接/闭包/同态/逆同态/反转的封闭性

CFL 对并/连接/闭包/同态封闭

定理 36. *上下文无关语言在并, 连接, 闭包, 正闭包, 同态下封闭.*

证明 1: 设 $\Sigma = \{1, 2\}$, L_1, L_2 是任意 CFL. 定义代换

$$s(1) = L_1, \quad s(2) = L_2.$$

语言 $\{1, 2\}$, $\{12\}$, $\{1\}^*$ 和 $\{1\}^+$ 显然都是 CFL, 那么

1. 由 $s(\{1, 2\}) = s(1) \cup s(2) = L_1 \cup L_2$, 所以并运算封闭;
2. 由 $s(\{12\}) = s(12) = s(\varepsilon)s(1)s(2) = L_1L_2$, 所以连接运算封闭;
3. 闭包和正比包运算封闭, 因为

$$\begin{aligned} s(\{1\}^*) &= s(\{\varepsilon, 1, 11, \cdots\}) \\ &= s(\varepsilon) \cup s(1) \cup s(11) \cup \cdots \\ &= \{\varepsilon\} \cup s(1) \cup s(1)s(1) \cup \cdots \\ &= L_1^*. \end{aligned}$$

若 h 是 Σ 上的同态, L 是 Σ 上的 CFL, 对 $\forall a \in \Sigma$ 令代换 $s'(a) = \{h(a)\}$, 则

$$h(L) = \{h(w) \mid w \in L\} = \bigcup_{w \in L} \{h(w)\} = \bigcup_{w \in L} s'(w) = s'(L),$$

所以同态运算封闭. □

证明 2: 用文法证明并/连接/闭包的封闭性. 设 CFL L_1 和 L_2 的文法分别为

$$G_1 = (V_1, T_1, P_1, S_1),\ G_2 = (V_2, T_2, P_2, S_2)$$

那么, 分别构造

1. $L_1 \cup L_2$ 的文法为

$$G_{\text{union}} = (V_1 \cup V_2 \cup \{S\},\ T_1 \cup T_2,\ P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\},\ S);$$

2. $L_1 L_2$ 的文法为

$$G_{\text{concat}} = (V_1 \cup V_2 \cup \{S\},\ T_1 \cup T_2,\ P_1 \cup P_2 \cup \{S \to S_1 S_2\},\ S);$$

3. L_1^* 的文法为

$$G_{\text{closure}} = (V_1 \cup \{S\},\ T_1,\ P_1 \cup \{S \to S_1 S \mid \varepsilon\},\ S).$$

再证明所构造文法的正确性, 略. □

CFL 对反转封闭

定理 37. *如果 L 是 CFL, 那么 L^R 也是 CFL.*

证明:

设 L 的文法 $G = (V, T, P, S)$, 构造文法

$$G' = (V,\ T,\ \{A \to \alpha^R \mid A \to \alpha \in P\},\ S),$$

则 $L(G') = L^R$. 证明略. □

CFL 对逆同态封闭

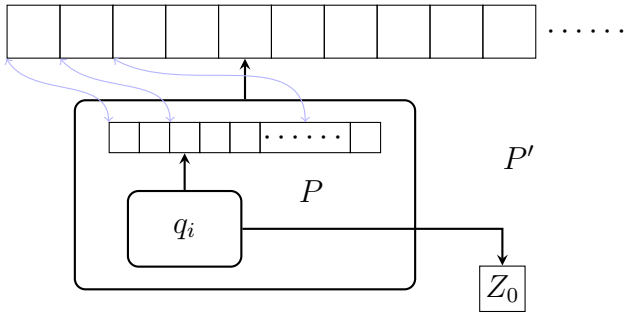
定理 38. *如果 L 是字母表 Δ 上的 CFL, h 是字母表 Σ 到 Δ^* 的同态, 那么 $h^{-1}(L)$ 也是 CFL.*

证明:

设 PDA $P = (Q, \Delta, \Gamma, \delta, q_0, Z_0, F)$, $\mathbf{L}(P) = L$. 构造 $\mathbf{L}(P') = h^{-1}(L)$ 的 PDA

$$P' = (Q',\ \Sigma,\ \Gamma,\ \delta',\ [q_0, \overline{\varepsilon}],\ Z_0,\ F \times \{\overline{\varepsilon}\}).$$

在 P' 的状态中, 使用缓冲, 暂存字符 $a \in \Sigma$ 的同态串 $h(a)$ 的后缀.



1. $Q' \subset Q \times \Delta^*$: 状态 $[q, \overline{x}]$ 中的 $\overline{x}$ 为缓冲;
2. 设 $q \in Q$, 那么 δ' 定义如下:

   i $\forall [q, \overline{\varepsilon}] \in Q \times \{\overline{\varepsilon}\}$, $\forall a \in \Sigma$, $\forall X \in \Gamma$

   $$\delta'([q, \overline{\varepsilon}], a, X) = \{([q, h(a)], X)\}$$

   ii 若 $\delta(q, \overline{a}, X) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots, (p_k, \beta_k)\}$, 则

   $$\delta'([q, \overline{ax}], \varepsilon, X) = \{([p_1, \overline{x}], \beta_1), ([p_2, \overline{x}], \beta_2), \cdots, ([p_k, \overline{x}], \beta_k)\}$$

   这里 $\overline{a} \in \Delta \cup \{\overline{\varepsilon}\}$, $\overline{x}$ 是某个 $h(a)$ 的后缀. □

7.2.3 交和补运算不封闭

CFL 对交运算不封闭

因为语言

$$L_1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$$
$$L_2 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$$

都是 CFL, 而

$$L_1 \cap L_2 = \{0^n 1^n 2^n \mid n \geq 1\}$$

不是 CFL.

CFL 对补运算不封闭

因为

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}.$$

定理 39. *若 L 是 CFL 且 R 是正则语言, 则 $L \cap R$ 是 CFL.*

证明: 设 DFA $D = (Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1)$ 且 $\mathbf{L}(D) = R$, PDA $P = (Q_2,\ \Sigma,\ \Gamma,\ \delta_2,\ q_2,\ Z_0,\ F_2)$ 且 $\mathbf{L}(P) = L$, 构造 PDA

$$P' = (Q_1 \times Q_2,\ \Sigma,\ \Gamma,\ \delta,\ [q_1, q_2],\ F_1 \times F_2)$$

其中 δ 为:

$$\delta([p,q],a,Z) = \begin{cases} \{([p,s],\beta) \mid (s,\beta) \in \delta_2(q,a,Z)\} & a = \varepsilon \\ \{([r,s],\beta) \mid r = \delta_1(p,a) \wedge (s,\beta) \in \delta_2(q,a,Z)\} & a \neq \varepsilon \end{cases}$$

再往证 $\mathbf{L}(P') = L \cap R$, 略. □

7.2.4 封闭性的应用

例 4. 请证明语言 L 不是 CFL

$$L = \{w \in \{a,b,c\}^* \mid \mathrm{n}_a(w) = \mathrm{n}_b(w) = \mathrm{n}_c(w)\},$$

其中 $\mathrm{n}_a(w)$ 表示 w 中 a 的个数.

证明:

1. 因为 $\mathbf{a}^*\mathbf{b}^*\mathbf{c}^*$ 是正则语言,
2. 而 $L \cap \mathbf{a}^*\mathbf{b}^*\mathbf{c}^* = \{a^n b^n c^n \mid n \geq 0\}$ 不是 CFL,
3. 由 CFL 与正则语言的交还是 CFL, 所以 L 不可能是 CFL. □

例. 请证明语言 $L = \{a^i b^j a^i b^j \mid i \geq 1, j \geq 1\}$ 不是 CFL.

证明:

1. 因为 $\mathbf{a}^+\mathbf{b}^+\mathbf{a}^+\mathbf{b}^+$ 是正则语言,
2. 而 $L_{ww} = \{ww \mid w \in \{a,b\}^*\}$ 不是 CFL,
3. 所以 $L_{ww} \cap \mathbf{a}^+\mathbf{b}^+\mathbf{a}^+\mathbf{b}^+ = L$ 也不可能是 CFL. □

7.3 上下文无关语言的判定性质

可判定的 *CFL* 问题

- 空性: 只需判断文法的开始符号 S 是否为非产生的.

- 有穷性和无穷性:
  1. 用不带无用符号的 CNF 的产生式画有向图;
  2. 变元为顶点, 若有 $A \to BC$, 则 A 到 B 和 C 各画一条有向边;
  3. 检查图中是否有循环.
- 成员性: 利用 CNF 范式, 有 CYK 算法检查串 w 是否属于 L.

CYK[1]算法

- CNF $G = (V, T, P, S)$, 以 $O(n^3)$ 时间检查 "$w = a_1 a_2 \cdots a_n \in \mathbf{L}(G)$?"
- 以动态规划方式, 在表中由下至上逐行计算 X_{ij}, 再检查 "$S \in X_{1n}$?"

$$X_{ij} = \{A \in V \mid A \overset{*}{\Rightarrow} a_i a_{i+1} \cdots a_j,\ 1 \le i \le j \le n\},$$

- 计算首行

$$X_{ii} = \{A \mid A \to a_i \in P\}$$

  共有 $O(n)$ 个 X_{ii} 需要计算.

- 计算其他

$$X_{ij} = \left\{ A \;\middle|\; \begin{array}{l} i \le k < j, \\ BC \in X_{ik} X_{k+1,j}, \\ A \to BC \in P \end{array} \right\}$$

  共计算 $O(n^2)$ 个 X_{ij}, 而每个要计算 $O(n)$ 组 $X_{ik} X_{k+1,j}$, 因此总时间复杂度为 $O(n^3)$.

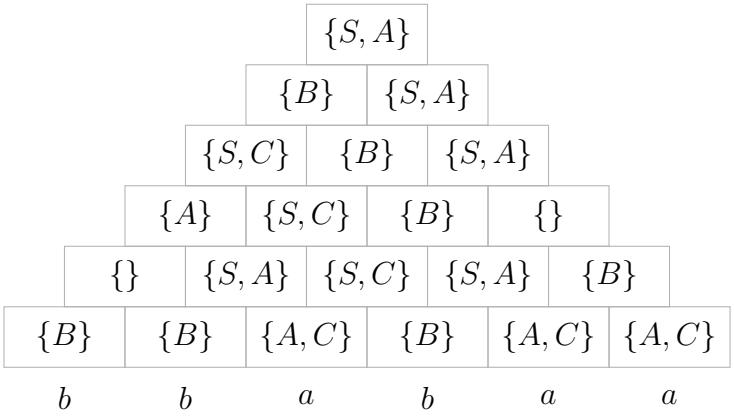
$$\begin{array}{ccccc} & & X_{15} & & \\ & X_{14} & & X_{25} & \\ X_{13} & & X_{24} & & X_{35} \\ X_{12} & X_{23} & X_{34} & X_{45} & \\ X_{11} & X_{22} & X_{33} & X_{44} & X_{55} \\ \hline a_1 & a_2 & a_3 & a_4 & a_5 \end{array}$$

例 5. CNF G 如下, 用 CYK 算法判断 $bbabaa \in \mathbf{L}(G)$?

$$S \to AB \mid BC$$

[1]J. Cocke, D. Younger, T. Kasami 分别独立发现了算法的基本思想

$$A \to BA \mid a$$
$$B \to CC \mid b$$
$$C \to AB \mid a$$



因为 $S \in X_{16} = \{S, A\}$, 所以 $bbabaa \in \mathbf{L}(G)$.

不可判定的 *CFL* 问题

1. 判断 CFG G 是否歧义的?
2. 判断 CFL 是否固有歧义的?
3. 两个 CFL 的交是否为空?
4. 两个 CFL 是否相同?
5. 判断 CFL 的补是否为空? 尽管有算法判断 CFL 是否为空
6. 判断 CFL 是否等于 Σ^*?

7.4 乔姆斯基文法体系

如果文法 $G = (V, T, P, S)$, 符号串 $\alpha \in (V \cup T)^* V (V \cup T)^*$, $\beta \in (V \cup T)^*$, 产生式都形如

$$\alpha \to \beta$$

即每个产生式的左部 α 中至少要有一个变元, 那么:

1. 称 G 为 0 型文法或短语结构文法 (PSG, *Phrase Structure Grammar*), $L(G)$ 称为 0 型语言, 短语结构语言 (PSL, *Phrase Structure Language*), 或递归可枚举语言 (*Recursively Enumerable Language*);

2. 若 $|\beta| \geq |\alpha|$, 称 G 为 1 型文法或上下文有关文法 (CSL, *Context-Sensitive Language*), $L(G)$ 称为 1 型语言或上下文有关语言 (CSL, *Context-Sensitive Language*);

3. 若 $\alpha \in V$, 称 G 为 2 型文法或上下文无关文法, $L(G)$ 称为 2 型语言或上下文无关语言;

4. 若 $\alpha \to \beta$ 都形如 $A \to aB$ 或 $A \to a$, 其中 $A \in V$, $a \in T$, 称 G 为 3 型文法, 右线性文法或正则文法, $L(G)$ 称为 3 型语言或正则语言.

乔姆斯基文法体系的四种类型中, 0 型文法的能力等价于图灵机, 1 型文法的能力等价于线性界限自动机. 2 型文法能力等价于非确定的下推自动机. 3 型文法也称右线性文法, 能力等价于有穷自动机. 文法描述语言的能力, 0 型文法最强, 3 型文法最弱.

Chapter 8

图灵机

本章, 我们将介绍图灵机 — 计算机的一种简单数学模型. 尽管图灵机简单, 但它具有模拟通用计算机的能力. 人们研究图灵机不仅是为了研究它所定义的语言类 (递归可枚举语言), 也是为了研究它所计算的整数函数类 (部分递归函数).

8.1 图灵机

在我们的课程中, 算法或 "机械而有效的计算过程" 的直觉概念已经出现过多次, 比如我们给出了一个有效过程判定有穷自动机接受的集合是否是空的, 有穷的或无穷的. 人们也会朴素的设想, 对于具有有穷描述的任何语言类, 总会存在一个有效过程来回答这类问题. 然而, 情况并非如此. 例如, 不存在算法判断一个 CFL 的补是否为空, 尽管我们可以判断这个 CFL 本身是否为空. 需要注意的是, 我们不是要求一个过程对某个具体的上下文无关语言回答这个问题, 而是要求单独一个过程, 对所有的 CFL 正确地回答这个问题.

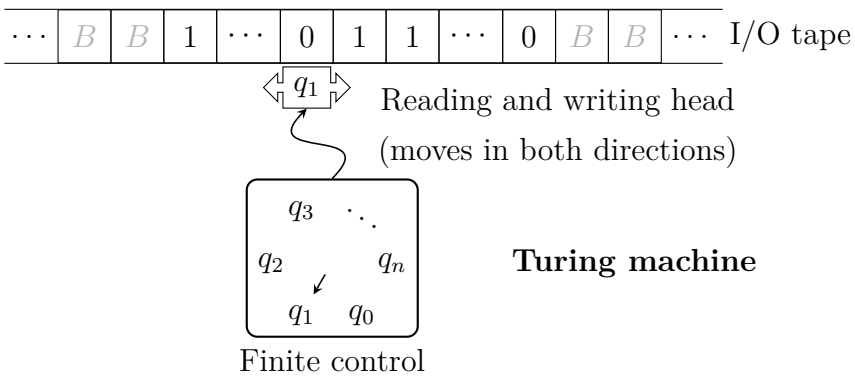
20 世纪初, 数学家希尔伯特曾经试图寻找一个过程, 用来确定整数上一阶谓词演算中, 一个任意任何公式是否为真. 因为一阶谓词演算足以表达命题: 一个上下文无关语言产生的语言是 Σ^*. 如果这样的过程存在, 则判定一个 CFL 的补是否为空将被解决. 但是 1931 年, 哥德尔发表了著名的不完全性定理, 证明了这样的有效过程不可能存在. 在应用于整数的谓词演算中, 他构造了一个公式, 该公式的定义表明, 其本身在这个逻辑系统中既不能被证明也不能被证否. 这个讨论的形式化, 以及后来关于有效果过程直觉概念的澄清和形式化, 是 20 世纪伟大的智力成就之一.

一旦有效过程概念被形式化, 就可以证明, 对于许多具体函数的计算没有有效过程. 今天, 图灵机已成为被人们接受的有效过程的形式定义. 我们虽然无法证明图灵机模型等价于我们关于计算的直觉概念, 但却有关于这个等价性的令人信服的证据, 也就是为人熟知的 Church 假设. 特别的, 就像我们已知的, 图灵机在计算能力上等价于数字计算机, 也等价于关于计算的所有最一般的数学概念.

有效过程的形式模型应该具有某些性质. 首先, 每个过程都应该是有穷可描述的; 其次, 过程应该由离散的步组成, 每一步能够机械的被之行. 图灵在 1936 年介绍了一个这样的模型, 被后人称为图灵机 (Turing Machine).

8.1.1 形式定义

图灵机的基本模型具有一个有穷控制器, 一条两端无穷的输入输出带和一个带头. 带划分为许多单元格, 带头每次扫视带上的一个单元格, 每个单元格可以放置有穷带符号集中的一个. 开始时, 带上的连续 n 个单元格放着输入, 它是一个字符串, 符号选自带符号的一个子集, 即输入符号集. 余下的无穷多个单元格都放着空白符, 它是一个不属于输入符号集的特殊带符号.



在一个动作中, 图灵机根据带头处单元格的符号和有限控制器的状态, (1) 改变状态, (2) 在单元格中写下一个符号, 以代替原来的符号, (3) 向左或向右移动一个单元格.

图灵机的形式定义

定义. *图灵机 (TM, Turing Machine) M 为七元组*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

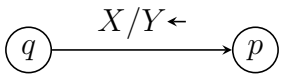
1. Q: *有穷状态集;*
2. Σ: *有穷输入符号集;*
3. Γ: *有穷带符号集, 且总有 $\Sigma \subset \Gamma$;*
4. δ: $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ *转移函数, 而 δ 可以对某些自变量没有定义;*
5. $q_0 \in Q$: *初始状态;*
6. $B \in \Gamma - \Sigma$: *空格符号或空白符;*
7. $F \subseteq Q$: *终态集或接受状态集.*

图灵机的动作及状态转移图

有穷控制器处于状态 q, 带头所在单元格为符号 X, 如果动作的定义为

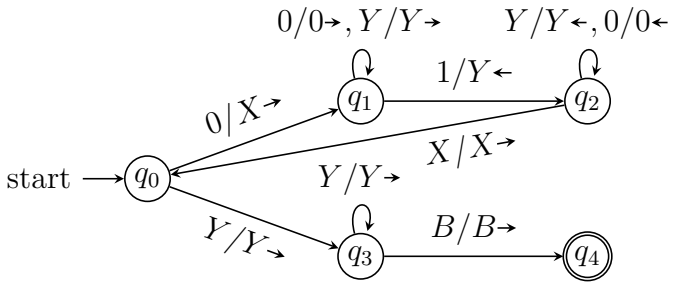
$$\delta(q, X) = (p, Y, L),$$

表示状态转移到 p, 单元格改为 Y, 然后带头向左移动一个单元格.



因为每个动作都是确定的, 因此是“确定的图灵机”.

例 1. 设计识别 $\{0^n1^n \mid n \geq 1\}$ 的图灵机.



$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

δ	0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

8.1.2 瞬时描述及其转移

瞬时描述

定义. 图灵机虽有无穷长的带, 但是在有限步移动之后, 带上的非空内容总是有限的. 因此用带上最左边到最右边全部的非空符号、当前状态和带头位置, 同时定义瞬时描述 *(ID, Instantaneous Description)* 或格局 *(Configuration)* 为

$$X_1X_2 \cdots X_{i-1}qX_iX_{i+1} \cdots X_n$$

其中的信息包括:

1. 图灵机的当前状态 q;

2. 带头在左起第 i 个非空格符上;

3. $X_1X_2\cdots X_n$ 是输入带上从最左到最右非空格内容, 虽然中间可能有空格符.

4. 为避免混淆, 一般假定 Q 和 Γ 不相交.

转移符号

定义. 图灵机 M 中, 如果 $\delta(q, X_i) = (p, Y, L)$, 定义 *ID* 转移为

$$X_1 \dots X_{i-1} q X_i \dots X_n \vdash_M X_1 \dots X_{i-2} p X_{i-1} Y X_{i+1} \dots X_n$$

如果 $\delta(q, X_i) = (p, Y, R)$ 那么

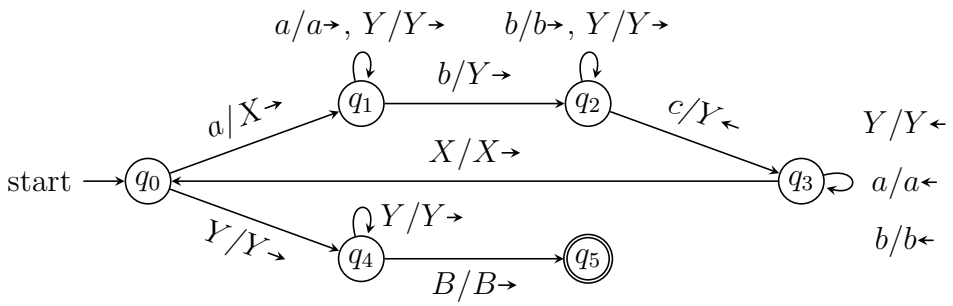
$$X_1 \dots X_{i-1} q X_i \dots X_n \vdash_M X_1 \dots X_{i-1} Y p X_{i+1} \dots X_n$$

若某 *ID* 是从另一个经有限步 *(*包括零步*)* 转移而得到的, 记为 $\vdash_M^*$. 若 M 已知, 简记为 $\vdash$ 和 $\vdash^*$.

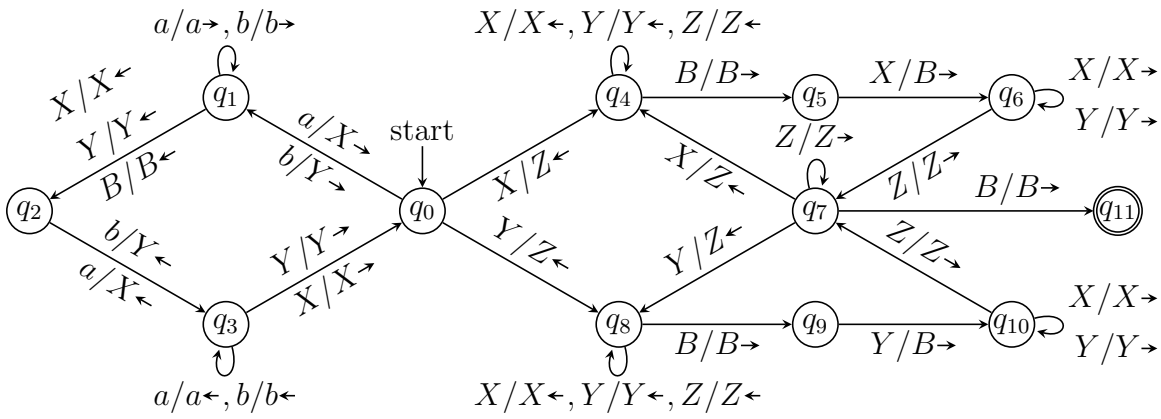
接受 0011 的 ID 序列 (M 的一个计算) 为

$$\begin{aligned} q_0 0011 &\vdash X q_1 011 \quad \vdash X 0 q_1 11 \quad \vdash X q_2 0 Y 1 \\ &\vdash q_2 X 0 Y 1 \quad \vdash X q_0 0 Y 1 \quad \vdash X X q_1 Y 1 \\ &\vdash X X Y q_1 1 \quad \vdash X X q_2 Y Y \vdash X q_2 X Y Y \\ &\vdash X X q_0 Y Y \quad \vdash X X Y q_3 Y \vdash X X Y Y q_3 B \\ &\vdash X X Y Y B q_4 B \end{aligned}$$

例 2. 设计接受 $L = \{a^n b^n c^n \mid n \geq 1\}$ 的图灵机.



例 3. 设计接受 $L = \{ww \mid w \in \{a, b\}^+\}$ 的图灵机.



思考

1. DFA 和 TM 的主要区别?
2. 计算机, 究竟是 TM 还是 DFA?

8.1.3 语言与停机

图灵机的语言

定义. 如果 $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ 是一个图灵机, 则 M 接受的语言为

$$\mathbf{L}(M) = \{w \mid w \in \Sigma^*,\ q_0 w \vdash^* \alpha p \beta,\ p \in F,\ \alpha, \beta \in \Gamma^*\}.$$

输入串 w 放在输入带上, M 处于 q_0, 带头位于输入串的第一个字符上, 输入串最终会导致 M 进入某个终结状态.

定义. 如果 L 是图灵机 M 的语言, 即 $L = \mathbf{L}(M)$, 则称 L 是递归可枚举语言.

一般假定当输入串 w 被接受时图灵机 M 总会停机 (*halt*), 即没有下一个动作 (的定义). 而对于不接受的输入, 图灵机可能永不停止. 能够被图灵机接受的语言类, 称为递归可枚举的 (*recursively enumerable*, RE). 术语“可枚举”是指, 这些语言中的串可以被某个图灵机枚举出来. “递归”是一个在计算机出现之前的数学名词, 其意义与计算机科学家所说的“递归式”相近. 递归可枚举语言类非常广, 它真包含上下文无关语言. 有些包含在这个语言类中的语言, 无法机械的确定其成员资格. 若 $\mathbf{L}(M)$ 是这样的语言, 识别 $\mathbf{L}(M)$ 的任何图灵机, 在不属于 $\mathbf{L}(M)$ 的某些输入上 M 停不下来. 若 $w \in \mathbf{L}(M)$, 那么在 w 上, M 最终是会停下来的. 然而, 只要 M 还在某个输入上运行, 我们就永远也不会知道, 到底是因为运行的时间不够长而没有接受呢, 还是根本就不会停机.

定义. 对接受和不接受的输入, 都保证停机的图灵机, 所接受的语言称为递归语言.

从递归可枚举语言类中, 分出一个由保证停机的图灵机所识别的子类是比较方便的. 递归语言类是这样的一个子类, 它们至少被一个在所有输入上都能停机的图灵机接受. 在下一章中我们会看到, 递归语言类是递归可枚举语言类的真子集. 那么, 由 CYK 算法, 每个 CFL 都是一个递归语言.

算法的形式化

保证停机的图灵机, 正是算法的好模型, 即算法概念的形式化.

- λ-caculus — Alonzo Church, Stephen Kleene
- Partial recursive functions — Kurt Gödel
- Post machines — Emil Post
- Turing machines — Alan Turing

8.1.4 整数函数计算器

图灵机可以作为语言的识别器或枚举器, 也可以用作整数到整数的函数计算器.

- 传统的方法, 把整数 $i \geq 0$ 写为 1 进制, 用字符串 0^i 表示;
- 若计算 k 个自变量 $i_1, i_2, \ldots, i_k$ 的函数 f, 用

  $$0^{i_1}10^{i_2}1\cdots10^{i_k}$$

  作为 TM M 的输入;
- M 停机, 且输入带上为 0^m, 表示 $f(i_1, i_2, \ldots, i_k) = m$.
- M 计算的 f, 不必对所有不同的 $i_1, i_2, \ldots, i_k$ 都有值.

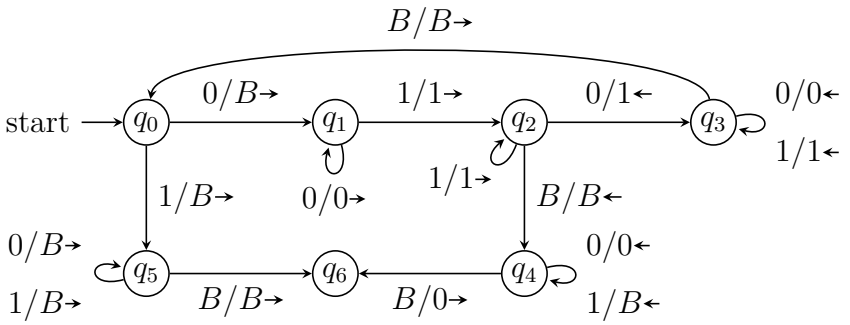
定义. *如果 $f(i_1, i_2, \ldots, i_k)$ 对所有不同的 $i_1, i_2, \ldots, i_k$ 都有定义, 称 f 为全递归函数.*

被图灵机计算的函数 $f(i_1, i_2, \ldots, i_k)$ 称作部分递归函数.

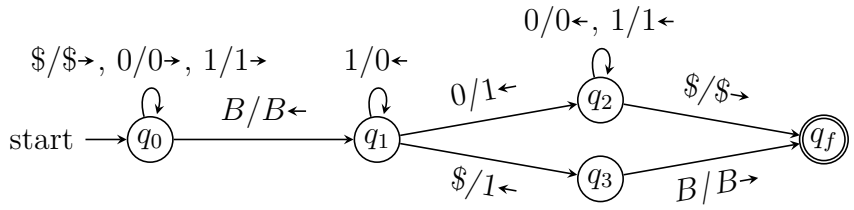
部分递归函数对应于递归可枚举语言, 它被一个在给定输入上可停可不停的图灵机计算. 全递归函数对应于递归语言, 它被总是能停机图灵机计算. 所有常用的整数函数都是全递归函数, 如乘积, $n!$, $\lceil \log_2 \rceil n$ 和 2^{2^n} 等.

例 4. 给出计算整数真减法 ($\dot{-}$) 的图灵机, 其定义为

$$m \dot{-} n = \begin{cases} m - n & m \geq n \\ 0 & m < n \end{cases}.$$



例 5. 二进制数的加 1 函数, 使用符号 $ 作为数字前的占位标记. 例如 $q_0\$10011 \vdash^* \$q_f10100$, $q_0\$111 \vdash^* q_f1000$.



8.2 图灵机的变形

以给出完整的状态集和动作函数的方法详细的设计一个图灵机是一项相当费力的任务. 为了描述复杂的图灵机结构, 我们需要 “高级” 的概念性工具和技术.

状态中存储

有限控制器中可以存储有限个符号的图灵机:

$$M' = (Q', \Sigma, \Gamma, \delta, q_0', B, F')$$

其中 $Q' = Q \times \Gamma \times \cdots \times \Gamma$, $q_0' = [q_0, B, \cdots, B]$.

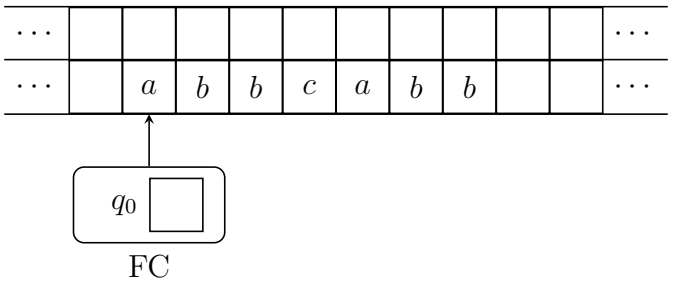
多道

多道图灵机:

$$M' = (Q, \Sigma, \Gamma', \delta, q_0, B', F)$$

其中 $\Gamma' = \Gamma \times \Gamma \times \cdots \times \Gamma$.

例 6. 利用状态中存储与多道设计 TM 识别 $L = \{wcw \mid w \in \{a, b\}^*\}$.



子程序

设计 TM 的一部分作为一个子程序:

- 具有一个指定的初始状态;
- 具有一个指定的返回状态, 但暂时没有定义动作;
- 可以具有参数和返回值.

通过进入子程序的初始状态, 实现调用; 通过返回状态的动作, 实现返回.

例 7. 设计 TM 实现全递归函数 “乘法”.

8.2.1 扩展的图灵机

多带图灵机

有穷控制器、k 个带头和 k 条带组成. 每个动作, 根据状态和每个带头符号:

1. 改变控制器中的状态;
2. 修改带头单元格中的符号;
3. 每个带头独立的向左或右移动一个单元格, 或保持不动.

开始时, 输入在第 1 条带上, 其他都是空的, 其形式定义非常繁琐.

定理 40. *如果语言 L 被一个多带图灵机接受, 那么 L 能够被某个单带图灵机接受.*

证明方法:

1. 用 $2k$ 道的单带图灵机 N 模拟 k 带图灵机 M;
2. N 用两道模拟 M 一带, 一道放置内容, 另一道标记带头;

3. 模拟 M 的一个动作, N 需要从左至右, 再从右至左扫描一次;

4. 第一次扫描搜集当前格局, 第二次扫描更新带头和位置.

非确定图灵机 (NTM)

在每个状态 q 和每个带符号 X 的转移, 可以有有限个选择的图灵机, 即

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \cdots, (q_k, Y_k, D_k)\}.$$

图灵机增加了非确定性, 并未改变图灵机接受语言的能力.

定理 41. *如果 L 被非确定图灵机接受, 那么 L 被图灵机接受.*

证明方法:

1. 同样用多带技术, 用确定的 TM M 模拟 NTM N;

2. M 用第 1 条带存储 N 未处理的 ID, 用第 2 条带模拟 N;

3. M 从第 1 条带取 N 的当前 ID 放到第 2 带;

4. 若不接受, 把当前 ID 可能的 k 个转移 ID 复制到第 1 条带的最末端;

5. 然后循环, 继续从第 1 带取下一个 ID 去模拟.

思考题

为什么非确定性没有改变图灵机识别语言的能力?

多维图灵机

1. 这种装置具有通常的有穷控制器;

2. k 维阵列组成的带, 在 $2k$ 个方向上都是无限的;

3. 根据状态和读入符号改变状态, 并沿着 k 个轴的正和负向移动;

4. 开始时，输入沿着某一个轴排列, 带头在输入的左端.

同样, 这样的扩展也没有增加额外的能力, 仍然等价于基本的图灵机.

8.2.2 受限的图灵机

半无穷带图灵机

图灵机的输入输出带只有一侧是无穷的.

定理 42. *半无穷带图灵机, 与图灵机等价.*

证明方法:

一侧无穷的图灵机带, 可使用多道技术, 模拟双侧无穷的图灵机带.

多栈机

基于下推自动机的扩展, k 栈机器是具有 k 个栈的确定型下推自动机.

定理 43. *如果图灵机接受 L, 那么双栈机接受 L.*

证明方法:

1. 一个堆栈保存带头左边内容, 一个堆栈保存带头右边内容;
2. 带头的移动用两个栈分别弹栈和压栈模拟;
3. 带头修改字符 A 为 B, 用一个栈弹出 A 而另一个压入 B 来模拟;
4. 开始时输入在双栈机的输入带, 但先将输入扫描并压入一个栈, 再依次弹出并压入另一个栈, 然后开始模拟图灵机.

例 8. 利用双栈机器接受 $L = \{a^n b^n c^n \mid n \geq 0\}$ 和 $L = \{a^n b^n c^n d^n e^n \mid n \geq 0\}$.

Chapter 9

不可判定性

9.1 不可判定性

非形式的, 我们使用问题来表示诸如“一个给定的 CFG 是否歧义? ”这样的询问. 那么一个具体的 CFG 就是一个问题的实例, 一般来说, 问题的一个实例就是一个自变量表, 每个自变量都表示问题的一个参数. 用某个字母表, 可以将问题的实例进行编码, 我们就能将是否存在解决某一问题的算法这一问题, 转化为一个特定的语言是否是递归的问题.

典型问题

给定语言 $L \subseteq \Sigma^*$ 和字符串 $w \in \Sigma^*$, 判断是否 $w \in L$ 的问题, 称为语言 L 上的一个判定性问题 (decision problem).

(非形式) 定义

如果一个问题, 不存在能解决它的程序, 则称为不可判定的 (undecidable).

是否存在不可判定的问题?

1. $\{L \mid L \in \Sigma^*\}$ 是不可数的;
2. $\{P \mid P$ 是一个程序$\}$ 是可数的;
3. 问题显然比程序多, 必然存在不可判定的问题.

hello-world 问题

判断带有给定输入的任意给定的程序, 是否以 `hello, world` 为其输出的前 12 个字符.

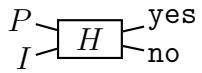
定理

hello-world 问题是不可判定的.

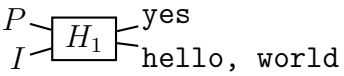
- “具有这个输入的这个程序是否显示 `hello, world`?”
- 解决这样问题的通用程序是不存在的.

(非形式) 证明: 反证法. 假设这样的程序 H 存在, 它可以在给定程序和输入时, 检查程序的输出是否以 `hello, world` 开始, 并正确的回答.

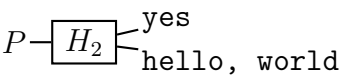
1. H 检查程序 P 在输入 I 时的输出, 并回答 `yes` 或 `no`:



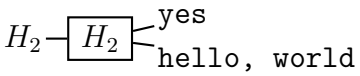
2. 修改 H, 在回答 `no` 时, 输出 `hello, world`:



3. 再修改 H_1, 将程序 P 作为 P 的输入:



4. 那么, 当程序 H_2 以 H_2 为输入时:



5. H_2 的输出会出现矛盾 (悖论), 所以 H_2 不可能存在, 而从 H 到 H_2 的修改是合理且能行的, 所以 H 不可能存在. □

问题的归约

如何证明问题是不可判定的?

1. 归谬法 (反证法)
2. 问题的归约

不可判定问题

- 递归可枚举语言 — 图灵机所识别
- 递归语言 — 保证停机的图灵机所识别

定义. 一个问题, 如果它的语言是递归的, 就称为可判定 *(decidable)* 问题, 否则称为不可判定 *(undecidable)* 问题.

不可判定的问题

- 不存在保证停机的图灵机识别该问题的语言
- 不存在解决该问题的算法

9.2 非递归可枚举的语言

可判定吗?

"图灵机 M 接受输入 w 吗? "

我们将使用对角线法证明一个特定的问题是不可判定的, 这个问题是"图灵机 M 接受输入 w 吗? ". 这里的 M 和 w 都是该问题参数, 并且限制 w 是 $\{0,1\}$ 上的串而 M 是仅接受 $\{0,1\}$ 上的串的图灵机. 这个受限的问题是不可判定的, 那么较一般的问题也肯定是不可判定的. 首先我们需要将问题实例编码为字符串, 将问题转化为语言.

9.2.1 第 i 个串

定义. 将全部 $(\mathbf{0}+\mathbf{1})^*$ 中的字符串按长度和字典序排序, 那么第 i 个串就是w_i. 且刚好有

$$\text{binary}(i) = 1w_i.$$

比如:

i	1	2	3	4	5	6	7	8	9	$\cdots$
binary(i)	1ε	10	11	100	101	110	111	1000	1001	$\cdots$
w_i	ε	0	1	00	01	10	11	000	001	$\cdots$

9.2.2 图灵机编码与第 i 个图灵机

图灵机编码

将 $\Sigma = \{0,1\}$ 上的全部图灵机, 用二进制字符串编码

$$M = (Q, \Sigma, \Gamma, \delta, q_1, B, F)$$

1. $Q = \{q_1, q_2, \cdots, q_{|Q|}\}$, 开始状态为 q_1, 终态为 q_2 且停机;
2. $\Gamma = \{X_1, X_2, \cdots, X_{|\Gamma|}\}$, 总有 $X_1 = 0$, $X_2 = 1$, $X_3 = B$;
3. 设带头移动方向 $D_1 = L,\ D_2 = R$;
4. 任意的转移 $\delta(q_i, X_j) = (q_k, X_l, D_m)$ 可用一条编码 $(Code)$ 表示为

$$C = 0^i 10^j 10^k 10^l 10^m;$$

5. 则全部 n 个转移的编码合并在一起, 作为图灵机 M 的编码:

$$C_1\,11\,C_2\,11\,\cdots\,C_{n-1}\,11\,C_n.$$

第 i 个图灵机 M_i

定义. 如果图灵机 M 的编码为第 i 个串 w_i, 则称 M 是第 i 个图灵机 M_i.

- 任意图灵机 M 都对应一个字符串 w
- 任意的字符串 w 都可以看作图灵机的编码
- 如果编码不合法, 将其看作接受 $\varnothing$ 且立即停机的图灵机

9.2.3 对角化语言 L_d

非递归可枚举的语言

定义. 使第 i 个串 w_i 不属于第 i 个图灵机 M_i 的语言 $\mathbf{L}(M_i)$ 的所有 w_i 的集合, 称为对角化语言 L_d, 即

$$L_d = \{w_i \mid w_i \notin \mathbf{L}(M_i), i \geq 1\}.$$

		$w_j \longrightarrow$						
		1	2	3	4	5	6	$\cdots$
$M_i \downarrow$	1	0	0	1	1	0	1	$\cdots$
	2	1	0	0	1	0	0	$\cdots$
	3	0	1	1	0	0	1	$\cdots$
	4	0	0	1	1	1	1	$\cdots$
	5	1	1	0	0	0	1	$\cdots$
	6	0	1	0	1	1	1	$\cdots$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

对角化语言 L_d 可以由上图的矩阵给出. 矩阵的每列上顺序排列每个字符串 w_j, 矩阵的每行前顺序排列每个图灵机 M_i. 如果 M_i 接受 w_j, 则矩阵中对应的位置为 1, 否则为 0. 矩阵的每行可以看做语言 $\mathbf{L}(M_i)$ 的特征向量 (*characteristic vector*). 处于对角线位置的值, 刚好表示图灵机 M_i 是否接受第 i 个串 w_i. 那么只需将对角线的值取补, 就是 L_d 的特征向量, 即给出了语言 L_d. 这里的对角化技术使 L_d 的特征向量与表中每行都在某列处不同, 因此也不可能是任何图灵机 (的语言) 的特征向量.

定理 44. L_d 不是递归可枚举语言, 即不存在图灵机接受 L_d.

证明: 反证法.

假设存在识别 L_d 的图灵机 M, 那么 M 也可被编码, 不妨设第 i 个图灵机 $M_i = M$, 即 $\mathbf{L}(M_i) = L_d$.

那么, 考虑第 i 个串 w_i 是否会被 M_i 识别:

1. 如果 $w_i \in \mathbf{L}(M_i) = L_d$, 那么由 L_d 的定义, 又有 $w_i \notin \mathbf{L}(M_i)$;
2. 如果 $w_i \notin \mathbf{L}(M_i)$, 那么由 L_d 的定义, 又有 $w_i \in L_d = \mathbf{L}(M_i)$.

无论如何都会矛盾, 因此假设不成立, 不存在接受 L_d 的图灵机. □

9.3 递归可枚举但非递归的语言

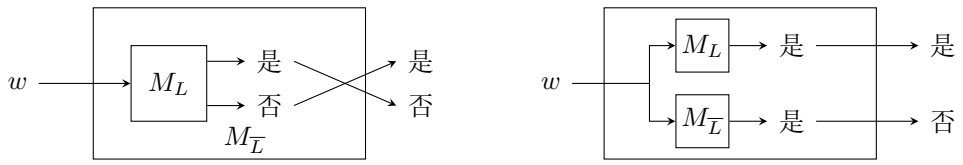
对角化语言 L_d 不存在图灵机, 那么肯定不存在算法解决语言为 L_d 的问题, 显然这样的问题都是不可判定的. 但即使存在图灵机, 如果无法保证停机, 对于问题的解决也没有实质的贡献, 因此将“问题”区分为可判定的和不可判定的, 要比区分问题是否具有图灵机更有意义. 这里将给出一个语言的实例—通用语言 L_u, 属于递归可枚举语言但不属于递归语言.

9.3.1 递归语言的封闭性

首先, 给出递归语言的两个封闭性定理. 递归语言中“递归”的含义是, 可以通过递归函数来解决, 而递归函数总会结束.

定理 45. *如果 L 是递归的, 那么 $\overline{L}$ 也是递归的.*

定理 46. *如果语言 L 和 $\overline{L}$ 都是递归可枚举的, 那么 L 是递归的.*



9.3.2 通用语言与通用图灵机

定义 (有序对 (M, w))**.** *一个图灵机 M 和一个输入串 w, 组成的有序对 (M, w), 可以表示为一个串即*

$$M111w.$$

这里的 M 不含任何连续 3 个的 1, 所以可以将 M 和 w 区分开.

定义. 如果图灵机 M 接受串 w, 那么由 $M111w$ 表示的有序对 (M, w) 构成的语言L_u, 称为通用语言 *(universal language)*

$$L_u = \{M111w \mid w \in \mathbf{L}(M)\}.$$

定义. 构造图灵机 U, 当输入 $M111w$ 时, 利用多带技术模拟 M 处理串 w 的过程. 因为 M 接受 w 时会停机, 因此 U 可以识别 L_u, 图灵机 U 称为通用图灵机 *(univerasl Turing machine)*.

递归可枚举但非递归的语言

定理 47. 通用语言 L_u 是递归可枚举的, 但不是递归的.

证明: L_u 是递归可枚举的. 用反证法证明 L_u 不是递归的.

通用图灵机 U 使用 3 条带分别: (1) 装载 M 的编码; (2) 放置 w, 模拟 M 的带; (3) 存储 M 的状态.

假设存在算法 A 识别 L_u, 那么可如下得到识别对角化语言 L_d 的算法 B.

将 B 的输入 $w = w_i$ 转换为 M_i111w_i 交给 A 判断:

- 当 A 接受, 表示 $w_i \in \mathbf{L}(M_i)$, 则 B 拒绝;
- 当 A 拒绝, 表示 $w_i \notin \mathbf{L}(M_i)$, 则 B 接受.

而由于 L_d 不是递归的, 所以 B 不可能存在, 所以 L_u 不可能是递归的. □

通用图灵机的重要意义

- 识别 L_u 的通用图灵机 U, 可以模拟任意图灵机
- 冯 • 诺伊曼通用数字电子计算机体系结构设计思想的灵感来源
- 抽象理论的先期发展可以对实际问题有很大帮助

9.4 语言类的关系