形式语言与自动机

课 程 讲 义

计算机科学与技术学院

软件基础教研室

王春宇

Contents

Chapter 1

简介

1.1 形式语言与自动机理论

计算机科学是关于计算知识的有系统的整体, 其始源可回溯到欧几里德关于一些算法的设计和巴比伦人关于渐进复杂性和归约性的使用. 然而, 现今的计算机学科的发展, 起源于两个重要的事件: 现代数字计算机的出现和算法概念的形式化.

计算机科学有两个主要的部分: 第一, 构成计算系统基础的一些基本概念和模型; 第二, 设计计算系统 (软件和硬件) 的工程技术. 形式语言与自动机理论, 就是作为第一部分, 即构成计算基础的基本概念的引论.

1.2 基础知识

1.2.1 基本概念

(1) 字母表 (*Alphabet*): 有穷非空符号集. 例如, $\Sigma = \{0, 1\}$, $\Sigma = \{a, b, \ldots, z\}$. (“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.)

(2) 字符串 (*Strings*): 某个字母表中符号的有穷序列, 也称字 (*words*). 例如, 若 $\Sigma = \{0, 1\}$, 则 000, 111, 0101, 10101 为 Σ 上的字符串.

(3) 空串 (*Empty string*): 长度为 0 的串. 一般表示为 ε.

(4) 串的长度: 串中符号的个数. 更准确的说, 是串中符号所占的位置数. 若串为 w, 那么长度记为 $|w|$. 例如, $|011| = 3$, $|\varepsilon| = 0$.

(5) 串的连接 (*Concatenation*): 如果 x 和 y 是串, 那么 xy 标识将 x 和 y 进行连接后得到的串. 例如 $x = 01101$ 和 $y = 110$, 那么 $xy = 01101110$.

对任意串 w, 都有 $\varepsilon w = w\varepsilon = w$, 所以 ε 也称为连接运算的单位元.

对任意串 w, 记 $w^0 = \varepsilon$ 和 $w^n = w^{n-1}w$.

例如 $a^3b^2 = aaabb$, $0^n1^n = 00\cdots011\cdots1$(0 和 1 都是 n 个).

(6) 串的逆序 (*Reverse*): 若 $w = a_1a_2\ldots a_n$, 则记 w 的逆序为 $w^R = a_na_{n-1}\ldots a_1$.

(7) 集合的连接 (或乘积): $AB = \{ab \mid a \in A, b \in B\}$.

例如 $\{0, 1\}\{0, 1\} = \{00, 01, 10, 11\}$; $\{0, 1\}\{a, b, c\} = \{0a, 0b, 0c, 1a, 1b, 1c\}$.

(8) 集合的幂: 递归定义 (1) $\Sigma^0 = \{\varepsilon\}$; (2) $\Sigma^n = \Sigma^{n-1}\Sigma$ $(n \geq 1)$.

那么, 若 Σ 是字母表, 那么定义 Σ^k 就是, 长度为 k 的串的集合. 而且 $\Sigma^0 = \{\varepsilon\}$ 对任何 Σ 都成立.

例如, 若 $\Sigma = \{0, 1\}$, 那么 $\Sigma^2 = \{00, 01, 10, 11\}$, $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$.

(9) 正闭包 (*Positive closure*): $\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4 \cup \cdots$

(10) 克林闭包 (*Kleene closure*): $\Sigma^* = \{\varepsilon\} \cup \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$, 显然 $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

Σ^* 就是字母表 Σ 上的所有的串的集合. Σ^+ 就是字母表 Σ 上全部非空串的集合.

例如 $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \cdots\}$

(11) 串的前缀 (*prefix*)、后缀 (*suffix*)、真前缀 (*proper prefix*)、真后缀 (*proper suffix*): 略

1.2.2 语言

语言 (*Languages*): 若 Σ 是字母表, 那么 $\forall L \subseteq \Sigma^*$, L 称为字母表 Σ 上的一个语言.

例如

(1) 自然语言, C 语言等

(2) The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$(对某个 $n \geq 0$, 形如 n 个 0 后面跟着 n 个 1 的所有串, 所构成的语言):

$$\{\varepsilon, 01, 0011, 000111, \ldots\}$$

(3) The set of strings of 0's and 1's with an equal number of each:

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

(4) Σ^* 是任意字母表 Σ 上的语言

(5) $\emptyset$ 是任意字母表 Σ 上的语言, 空语言

(6) $\{\varepsilon\}$ 是任意字母表 Σ 上的语言, 仅有一个空串的语言, 但 $\emptyset \neq \{\varepsilon\}$.

语言以集合的方式来描述: $\{w \mid \text{something about } w\}$, 例如

(1) $\{w \mid w \text{ consists of an equal number of 0's and 1's}\}$

(2) $\{w \mid w$ 是素数的二进制表示$\}$

(3) $\{0^n 1^n \mid n \geq 1\}$

(4) $\{0^i 1^j \mid 0 \leq i \leq j\}$

Chapter 2

有穷自动机

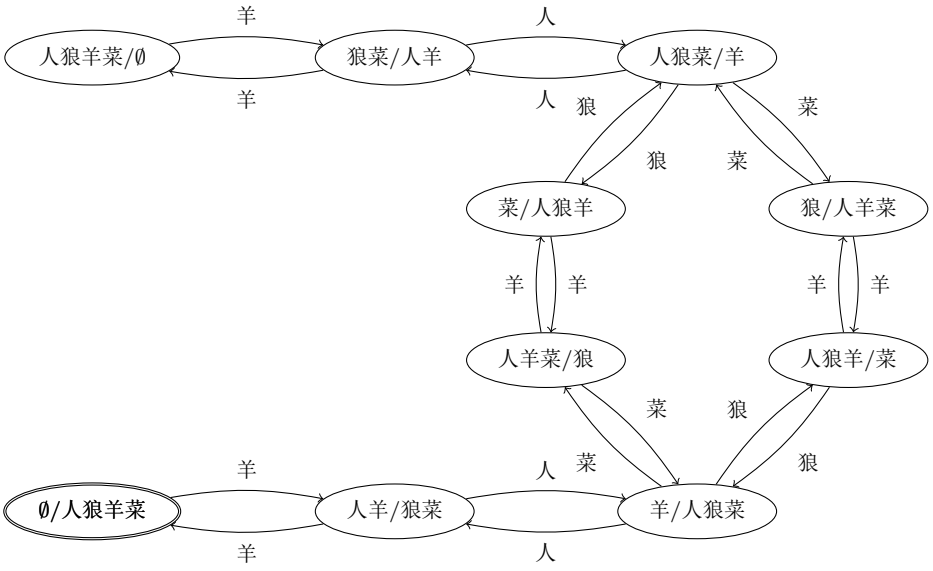
2.1 有穷状态系统

有穷自动机是具有离散输入和输出系统的一种数学模型. 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. 电梯的控制机构, 就是有穷状态系统的一个典型例子.

计算机科学中, 有穷系统的例子有很多, 常见的比如计算机的控制器、词法分析器、协议分析等.

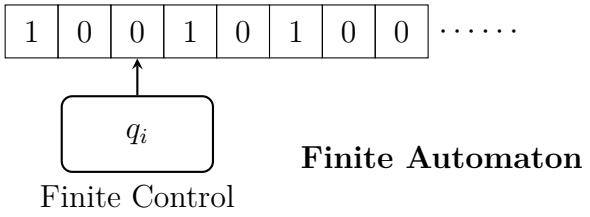
示例

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

确定型的有穷自动机具有一个有穷控制器, 一条输入带和一个读头. 有穷控制器可以记住有穷个状态, 其中一个是当前状态; 输入带上分为单元格, 每个单元格可以放置一个输入字符, 输入带放置一个输入串; 读头可以读取单元格上的字符, 并向后移动一个单元格.



一个确定的有穷自动机由一个有穷状态集和一个从状态到状态的转移集组成, 转移出现在输入时. 每个输入是字母表中的一个符号. 对每个输入符号, 从每一个状态恰有一个转移 (可以转移到这个状态本身). 有一个初始状态, 自动机从它开始. 某些状态被规定为终态或接受状态.

2.2.1 形式定义

确定型有穷自动机 (*Deterministic Finite Automaton*, DFA) A 的形式定义为五元组

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

其中

(1) Q: 有穷状态集;

(2) Σ: 有穷输入符号集或字母表;

(3) $\delta : Q \times \Sigma \mapsto Q$, 状态转移函数; (即 $\delta(q, a) = p$)

(4) q_0: 初始状态, $q_0 \in Q$;

(5) F: 终结状态集或接受状态集, $F \subseteq Q$.

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.

示例

接受全部含有 01 子串的 0 和 1 构成的串.

首先是字母表 $\Sigma = \{0, 1\}$, 然后分析串的特点: 01 是子串, 则在扫描输入串的过程中需要记住:

(1) 当前已经发现 01, 那么串的其余部分不用再关心;

(2) 还没发现 01, 但刚刚已经读入了一个 0, 那么只要再读入 1 就符合条件了;

(3) 还没发现 01, 甚至 0 都还没出现.

刚好这三种情况可以对应三个状态, 因此

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

其中 δ :

$$\begin{array}{ll} \delta(q_0, 1) = q_0 & \delta(q_0, 0) = q_1 \\ \delta(q_1, 0) = q_1 & \delta(q_1, 1) = q_2 \\ \delta(q_2, 1) = q_2 & \delta(q_2, 0) = q_2 \end{array}$$

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*)

DFA 的转移图

状态转移图的定义

(1) 每个状态对应一个节点, 用圆圈表示;

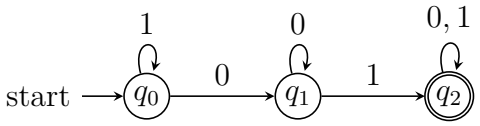
(2) 每个 $\delta(q,a)=p$ 对应一条从节点 q 到 p 的有向边, 边的标记为 a;

(3) 开始状态 q_0 有一个标有 *start* 的箭头;

(4) 接受状态的节点, 用双圆圈表示.

示例

前面的例子的转移图如下



DFA 的转移表

示例

前面例子的转移表如下

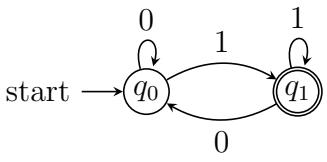
	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
$*q_2$	q_2	q_2

2.2.3 DFA 的设计

典型的问题: 给定语言 L, 设计 DFA 使其接受 (且仅接受) 语言 L.

示例

若 $\Sigma=\{0,1\}$, 给出接受全部以 1 结尾的串的 DFA.



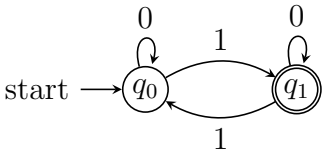
示例

$\Sigma=\{0,1\}$, 给出接受 Σ^* 的 DFA.

$\Sigma = \{0, 1\}$, 给出接受 $\{\varepsilon\}$ 的 DFA.

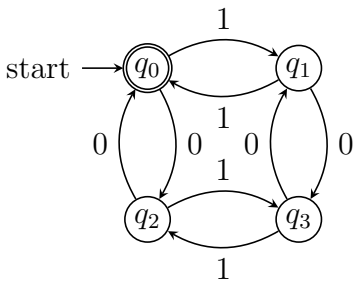
示例

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



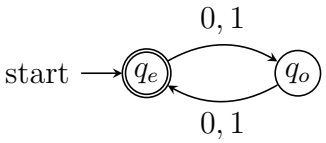
示例

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



示例

$\Sigma = \{0, 1\}$, 给出一个 DFA, 其接受长度为偶数的任意串.



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 $\hat{\delta} : Q \times \Sigma^* \mapsto Q$, 如下

(1) $\hat{\delta}(q, \varepsilon) = q$;

(2) 若 $w = xa$, 那么 $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$.

$\hat{\delta}$ 的含义可以理解为, 从一个状态开始, 读入某个串后, 所到达的状态.

那么, 如果 $w = a_0 a_1 \cdots a_n$, 那么 $\hat{\delta}(q, w) = \delta(\delta(\cdots \delta(\hat{\delta}(q, \varepsilon), a_0) \cdots, a_{n-1}), a_n)$.

2.2.5 DFA 的语言

DFA $A = (Q, \Sigma, \delta, q_0, F)$ 接受的语言记为为 $\mathbf{L}(A)$, 定义如下:

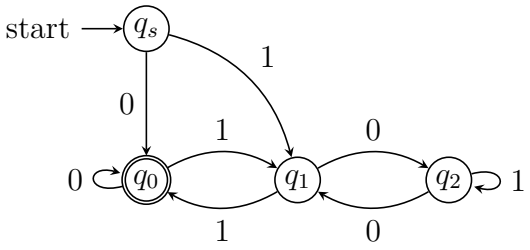
$$\mathbf{L}(A) = \{w \mid \hat{\delta}(q_0, w) \in F\}.$$

如果一个语言 L 是某个 DFA A 的语言, 即 $L = \mathbf{L}(A)$, 则称 L 是正则语言.

示例

Design a DFA that accepts all strings w over $\{0, 1\}$ such that w is the binary representation of a number that is a multiple of 3.

设 q_0, q_1, q_2 分别对应模 3 为 0, 1, 2 的状态; 此外, 因为不含空串, 设开始状态为 q_s; 对 q_0, q_1, q_2 每个当前状态, 输入 0 相当于乘 2, 输入 1 相当于乘 2 加 1, 那么可以找到相应的转移规律.



2.3 非确定的有穷自动机

下面给出非确定的有穷自动机的概念. 我们将最终证明, 被非确定的有穷自动机接受的任何集合, 都能够被确定的有穷自动机所接受. 然而, 非确定性概念无论在语言理论还是在计算理论中都起着重要的作用. 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. 后面, 我们将碰到确定形式和非确定形式不等价的自动机, 以及另外一些自动机, 这两种形式的等价性是一个深刻的、重要的悬而未决的问题.

修改 FA 模型, 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.

示例

由 0 和 1 构成的串中, 接受全部以 01 结尾的串.

2.3.1 形式定义

非确定型的有穷自动机 (*Nondeterministic Finite Automaton*, NFA) A 的形式定义为五元组

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

其中

(1) Q: 有穷状态集;

(2) Σ: 有穷输入符号集或字母表;

(3) $\delta : Q \times \Sigma \mapsto 2^Q$, 状态转移函数; (即 $\delta(q, a) = \{p_1, p_2, \ldots, p_n\}$)

(4) q_0: 初始状态, $q_0 \in Q$;

(5) F: 终结状态集或接受状态集, $F \subseteq Q$.

NFA 与 DFA 的区别是转移函数和接受方式: NFA 转移函数一般形式 $\delta(q,a) = \{p_1, p_2, \cdots, p_n\}$; 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.

示例

前面的例子中, NFA 的定义可以形式化的表述为

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

其中转移函数 δ 如下:

$$\begin{array}{lll} \delta(q_0,0) = \{q_0, q_1\} & \delta(q_1,0) = \emptyset & \delta(q_2,0) = \emptyset \\ \delta(q_0,1) = \{q_0\} & \delta(q_1,1) = \{q_2\} & \delta(q_2,1) = \emptyset \end{array}$$

若表示为状态转移表:

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

2.3.2 扩展转移函数

与 DFA 类似, 将 δ 扩展到输入串. 定义转移函数 $\hat{\delta} : Q \times \Sigma^* \mapsto 2^Q$ 的扩展为

(1) $\hat{\delta}(q, \varepsilon) = \{q\}$;

(2) 若 $w = xa$, 那么 $\hat{\delta}(q,w) = \bigcup_{p \in \hat{\delta}(q,x)} \delta(p,a)$.

示例

前面的例子中, 若输入是 00101, 每一步的状态转移分别是:

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

(2) $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$

(3) $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$

(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) $\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$

(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 NFA $A = (Q, \Sigma, \delta, q_0, F)$ 的语言 $\mathbf{L}(A)$ 为

$$\mathbf{L}(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}.$$

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 被某个 NFA 接受, 当且仅当 L 被某个 DFA 接受.*

证明. 设 NFA N=(Q_N, Σ, δ_N, q_0, F_N) 接受 L, 那么构造 DFA D=(Q_D, Σ, δ_D, $\{q_0\}$, F_D) 如下:

(1) $Q_D = 2^{Q_N}$, 即 DFA 的状态是 NFA 的状态集;

(2) 开始状态 $\{q_0\}$;

(3) 终态 $F_D = \{S \mid S \subseteq Q_N, S \cap F_N \neq \emptyset\}$, 即包含 NFA 终态的状态集, 作为 DFA 的终态;

(4) 状态转移函数 δ_D, 对每个 $S \subseteq Q_N$ 和每个输入字符:

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$

即 $\delta_D(S, a)$ 的计算, 就是在 NFA 中, 从 S 中每个状态 p 在输入 a 之后所能达到的全部状态的集合.

下面用归纳法, 证明 $\mathbf{L}(D) = \mathbf{L}(N)$. 对串 w 的长度进行归纳, 往证

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

成立.

[归纳基础] 当 $|w| = 0$ 时, 即 $w = \varepsilon$ 时, 显然 $\hat{\delta}_D(\{q_0\}, \varepsilon) = \hat{\delta}_N(q_0, \varepsilon) = \{q_0\}$.

[归纳假设] 假设 $|w| = n$ ($n \geq 0$) 时, 上式成立;

[归纳递推] 那么, 当 $|w| = n + 1$ 时, 将 w 拆分成 $w = xa$ 的形式, a 是 w 最后一个符号. 由 $\hat{\delta}_N$ 的定义, 有

$$\hat{\delta}_N(q_0, w) = \hat{\delta}_N(q_0, xa) = \bigcup_{p \in \hat{\delta}_N(q_0, x)} \delta_N(p, a)$$

由 $\hat{\delta}_D$ 的定义, 有

$$\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_D(\{q_0\}, xa) = \delta_D(\hat{\delta}_D(\{q_0\}, x), a)$$

由上面的 DFA D 的构造, 有

$$\delta_D(\hat{\delta}_D(\{q_0\}, x), a) = \bigcup_{p \in \hat{\delta}_D(\{q_0\}, x)} \delta_N(p, a)$$

又因为, 由归纳假设

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

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

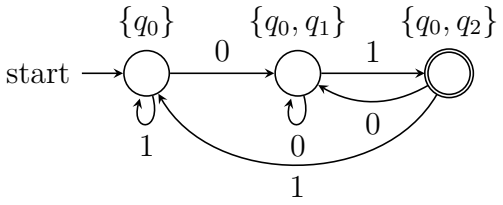
因此 $\forall w \in \mathbf{L}(N)$, 有 $\hat{\delta}_N(q_0, w) \cap F_N \neq \emptyset$, 那么 $\hat{\delta}_D(\{q_0\}, w) \in F_D$, 所以 $w \in \mathbf{L}(D)$, 即 $\mathbf{L}(N) \subseteq \mathbf{L}(D)$; 且 $\forall w \in \mathbf{L}(D)$, 有 $\hat{\delta}_D(\{q_0\}, w) \in F_D$, 那么 $\hat{\delta}_N(q_0, w) \cap F_N \neq \emptyset$, 所以 $w \in \mathbf{L}(N)$, 即 $\mathbf{L}(D) \subseteq \mathbf{L}(N)$; 因此 $\mathbf{L}(D) = \mathbf{L}(N)$.

□

示例 (子集构造法, 构造与 NFA 等价的 DFA)

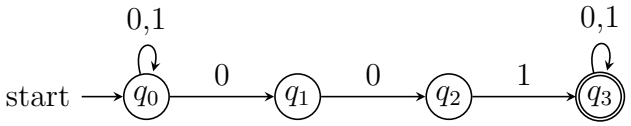
通过前面例子中 NFA 的状态转移表, 可以如下构造 DFA(横线划掉了无用状态)

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



示例

Design a NFA that accepts strings with 001 as substring.



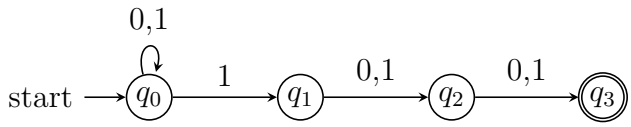
2.4 带有空转移的有穷自动机 (FA with ε-Transition)

2.4.1 ε-NFA 举例

为 FA 增加另一种扩展: 在空串 (ε) 发生状态转移.

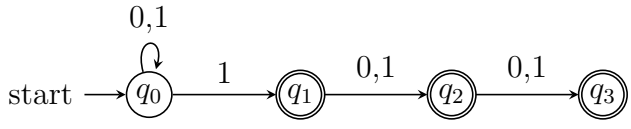
示例

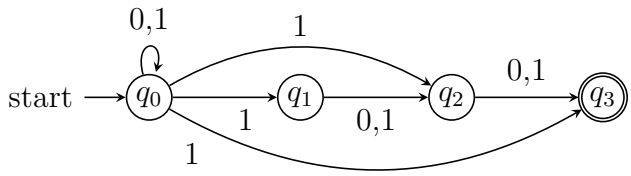
L=$\{w \mid w$ 倒数第 3 个字符是 1, $w \in \{0,1\}^*\}$



示例

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





2.4.2 ε-NFA 形式定义

带有空转移的非确定有穷自动机 (ε-NFA) A 的形式定义为五元组

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

其中:

(1) Q: 有穷状态集;

(2) Σ: 有穷输入符号集或字母表;

(3) δ: $Q \times (\Sigma \cup \{\varepsilon\}) \mapsto 2^Q$ 状态转移函数, 允许在空串上转移 (未读入字符就转移);

(4) q_0: 初始状态, $q_0 \in Q$;

(5) F: 终结状态集或接受状态集, $F \subseteq Q$.

与 DFA 相比, ε-NFA 的主要区别:

(1) 自动机处于某状态, 读入某个字符时, 可能有多余一个的转移;

(2) 自动机处于某状态, 读入某个字符时, 可能没有转移;

(3) 自动机处于某状态, 可能不读入字符, 就进行转移.

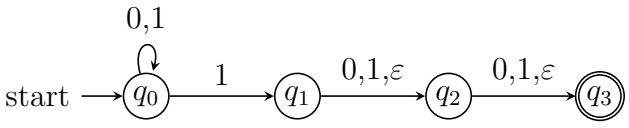
示例

前面例子的语言 L=$\{w \mid w$ 倒数 3 个字符至少有一个是 1, $w \in \{0,1\}^*\}$

可以构造 ε-NFA 为 $E = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_3\})$ 其中 δ 如转移表:

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

或状态转移图:



2.4.3 ε-闭包 (ε-Closure)

状态 q 的 ε-闭包: 从状态 q 经过全部标记 ε 的边所能到达的所有状态 (以及 q 自身) 的集合.

ε-闭包的递归定义:

(1) 状态 $q \in$ ECLOSE(q);

(2) $\forall p \in$ ECLOSE(q), 若 $r \in \delta(p, \varepsilon)$, 则 $r \in$ ECLOSE(q).

如果 S 是状态集, 则定义:

$$\text{ECLOSE}(S) = \bigcup_{q \in S} \text{ECLOSE}(q)$$

示例

前面例子

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

2.4.4 扩展转移函数

(1) $\hat{\delta}(q, \varepsilon) = \text{ECLOSE}(q)$

(2) 如果 $w = xa$, 则必然 $a \neq \varepsilon$, 因为 $\varepsilon \notin \Sigma$, 则

$$\hat{\delta}(q, w) = \text{ECLOSE}(\bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a))$$

即: 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 $\hat{\delta}(q, w) = \text{ECLOSE}(\{r_1, r_2, \ldots, r_m\})$

例: 前面的例子中, 求 $\hat{\delta}(q_0, 10) =?$

$\hat{\delta}(q_0, \varepsilon) = \text{ECLOSE}(q_0) = \{q_0\}$

$\hat{\delta}(q_0, 1) = \text{ECLOSE}(\hat{\delta}(q_0, 1)) = \text{ECLOSE}(\{q_0, q_1\}) = \{q_0\} \cup \{q_1, q_2, q_3\} = \{q_0, q_1, q_2, q_3\}$

$\hat{\delta}(q_0, 10) = \text{ECLOSE}(\hat{\delta}(q_0, 0) \cup \hat{\delta}(q_1, 0) \cup \hat{\delta}(q_2, 0) \cup \hat{\delta}(q_3, 0)) = \text{ECLOSE}(\{q_0, q_2, q_3\}) = \{q_0, q_2, q_3\}$

2.4.5 ε-NFA 的语言

若 ε-NFA $E = (Q, \Sigma, \delta, q_0, F)$, 则定义 E 的语言 $\mathbf{L}(E)$ 为

$$\mathbf{L}(E) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}.$$

2.4.6 消除空转移

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.

设 ε-NFA $E = (Q_E, \Sigma, \delta_E, q_E, F_E)$, 构造 DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$, 规则如下:

(1) $Q_D = 2^{Q_E} = \{S \mid S \subseteq Q_E\}$ – 但实际上只有满足 $S = \text{ECLOSE}(S)$ 是有效的;

(2) $q_D = \text{ECLOSE}(q_E)$ – 即使用 q_E 的闭包作为 D 的开始状态;

(3) $F_D = \{S \mid S \in Q_D, S \cap F_E \neq \emptyset\}$ – 只要包含 F_E, 就是 D 的接受状态;

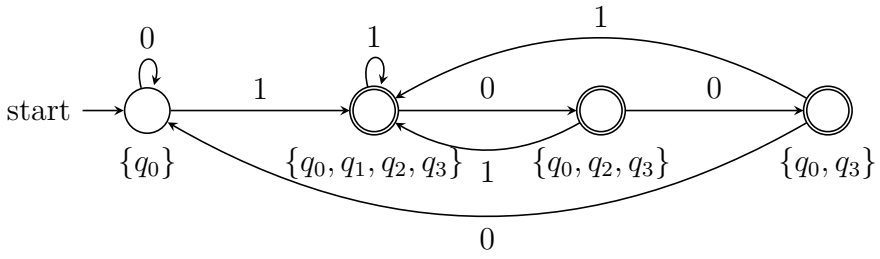
(4) δ_D: $\forall S \in Q_D$, $\delta_D(S, a) = \text{ECLOSE}(\bigcup_{p \in S} \delta_E(p, a))$.

示例

续前例, 得到消除 ε 转移后, 得到的 DFA 状态转移表

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

转移图为



2.4.7 ε-NFA 与 DFA 等价性

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

证明. ($\Leftarrow$) 已知 DFA $D = (Q, \Sigma, \delta_D, q_0, F)$ 只需构造 ε-NFA E 的构造函数 $\delta_E(q, \varepsilon) = \emptyset$, 且对每个 $\delta_D(q, a) = p$ 都有 $\delta_E(q, a) = \{p\}$; E 与 D 完全一样, 但没有任何空转移, 接受的语言也一样.

($\Rightarrow$) 先证明 $\hat{\delta}_E(q_E, w) = \hat{\delta}_D(q_D, w)$, 对 $|w|$ 归纳

(1) 当 $|w| = 0$ 时, 有 $\hat{\delta}_E(q_E, \varepsilon) = \text{ECLOSE}(q_E)$
由构造方法得 $q_D = \text{ECLOSE}(q_E)$, 所以 $\hat{\delta}(q_D, \varepsilon) = q_D = \text{ECLOSE}(q_E)$

(2) 假设 $|w| = n$ 时, 等式成立, 则当 $|w| = n + 1$ 时, $w = xa$, 则有 $\hat{\delta}_E(q_E, x) = \hat{\delta}_D(q_D, x)$.
设 $S = \hat{\delta}_E(q_E, x) = \hat{\delta}_D(q_D, x)$,
$\hat{\delta}_E(q_E, w) = \text{ECLOSE}(\bigcup_{p \in S} \delta_E(p, a))$,
又 $\hat{\delta}_D(q_D, w) = \delta_D(\hat{\delta}_D(q_D, x), a) = \delta_D(S, a) = \text{ECLOSE}(\bigcup_{p \in S} \delta_E(p, a))$
所以 $\hat{\delta}_E(q_E, w) = \hat{\delta}_D(q_D, w)$.

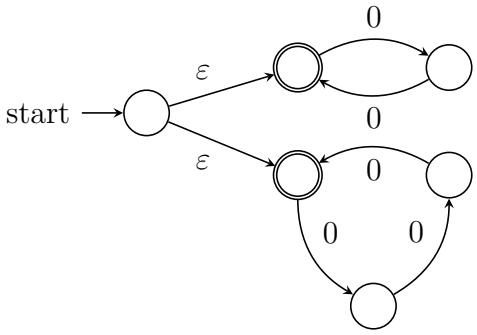
对 $\forall w \in \mathbf{L}(E)$ 有 $\hat{\delta}_E(q_E, w) \cap F_E \neq \emptyset$ 即 $\hat{\delta}_D(q_D, w) \cap F_E \neq \emptyset$ 即 $\hat{\delta}_D(q_D, w) \in F_D$ 即 $x \in \mathbf{L}(D)$ 所以 $\mathbf{L}(E) \subseteq \mathbf{L}(D)$;

又 $\forall w \in \mathbf{L}(D)$ 有 $\hat{\delta}_D(q_D, w) \in F_D$ 即 $\hat{\delta}_D(q_D, w) \cap F_E \neq \emptyset$ 即 $\hat{\delta}_E(q_E, w) \cap F_E \neq \emptyset$ 即 $x \in \mathbf{L}(E)$ 所以 $\mathbf{L}(D) \subseteq \mathbf{L}(E)$.

所以 $\mathbf{L}(E) = \mathbf{L}(D)$. □

示例

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



Chapter 3

正则表达式和正则语言

3.1 正则表达式

自动机通过识别来定义语言, 正则表达式通过规则产生语言 (或表示语言); 正则表达式所表示的语言与正则语言等价.

3.1.1 语言的运算

如果 L 和 M 是两个语言:

(1) $L \cup M$ 为两个语言的并

(2) LM 为两个语言的连接

(3) L^* 为语言的 (克林) 闭包

示例

若 $L = \{0, 11\}$, $M = \{\varepsilon, 001\}$, 那么

(1) $L \cup M = \{0, 11, \varepsilon, 001\}$

(2) $LM = \{0, 0001, 11, 11001\}$, $ML = \{0, 11, 0010, 00111\}$

(3) $L^0 = \{\varepsilon\}$, $L^1 = L = \{0, 11\}$, $L^2 = \{00, 011, 110, 1111\}$,
$L^3 = \{000, 0011, 0110, 01111, 1100, 11011, 11110, 111111\}$, $\cdots$
$L^* = \bigcup_{i=0}^{\infty} L^i$

示例

四则运算表达式的定义

(1) 任何的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 $a + b$, $a - b$, $a \times b$, $a \div b$ 和 (a) 都是四则运算表达式.

3.1.2 正则表达式的递归定义

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

(1) $\emptyset$ 是一个正则表达式, 表示空语言;

(2) ε 是一个正则表达式, 表示语言 $\{\varepsilon\}$;

(3) Σ 中的任意字符 a, 都是一个正则表达式, 分别表示语言 $\{a\}$;

(4) 如果正则表达式 r 和 s 分别表示语言 R 和 S, 则 $r+s$, rs, r^* 和 (r) 也是正则表达式, 分别表示语言 $R \cup S$, RS, R^* 和 R.

此外 $r^+ = rr^*$ 也称为正闭包, 显然 $r^* = r^+ + \varepsilon$. 而且 $r^* = r^+$ 当且仅当 $\varepsilon \in \mathbf{L}(r)$.

示例

给出正则表达式 $(a+b)^*(a+bb)$ 定义的语言.

因为

(1) $a \to \{a\}$, $b \to \{b\}$;

(2) $a+b \to \{a\} \cup \{b\} = \{a, b\}$, $bb \to \{b\}\{b\} = \{bb\}$;

(3) $a + bb \to \{a\} \cup \{bb\} = \{a, bb\}$

(4) $(a+b)^* = \{a, b\}^*$

(5) $(a+b)^*(a+bb) \to \{a, b\}^*\{a, bb\} = \{a, bb, aa, abb, ba, bbb, \ldots\}$

所以 $\mathbf{L}((a+b)^*(a+bb)) = \{w \mid w$ 由 a 和 b 组成, 仅以 a 或 bb 结尾.$\}$

示例

给出正则表达式 $(aa)^*(bb)^*b$ 定义的语言.

$\mathbf{L}((aa)^*(bb)^*b) = (\{a\}\{a\})^*(\{b\}\{b\})^*\{b\} = \{aa\}^*\{bb\}^*\{b\} = \{a^{2n}b^{2m+1} \mid n \geq 0, m \geq 0\}$

示例

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

$(0+1)^*01(0+1)^*$

示例

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.$\}$

$(0+1)^*1(0+1)(0+1)$

示例

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

$1^*(011^*)^*(0+\varepsilon)$ 或 $(1+01)^*(0+\varepsilon)$

示例

00 表示语言 $\{00\}$. $(0+1)^*$ 表示任意 0 和 1 构成的串. $(0+1)^*00(0+1)^*$ 表示至少有两个连续的 0 的串. $(0+\varepsilon)(1+10)^*$ 表示没有两个连续 0 的串.

3.1.3 运算符的优先级

正则表达式的三种运算: “加” $(+)$, “连接” $(\cdot$ 一般省略$)$ 和 “星” $(*)$.

正则表达式中运算符的优先级: 括号的优先级最高, 但括号本身并不是运算

(1) 首先, “星” 优先级最高: r^*

(2) 其次, “连接” : rs, $r \cdot s$

(3) 最后, “加” 优先级最低: $r + s$

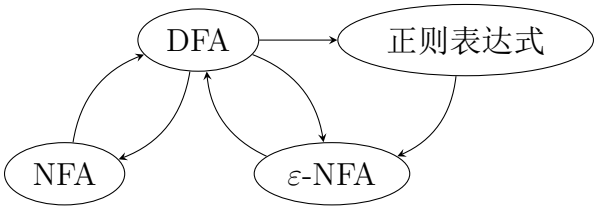
示例

$01^* + 1 = (0(1^*)) + 1$

3.2 有穷自动机和正则表达式

3.2.1 正则语言的表示

DFA, NFA, ε-NFA 和正则表达式在表示语言的能力上是等价的.

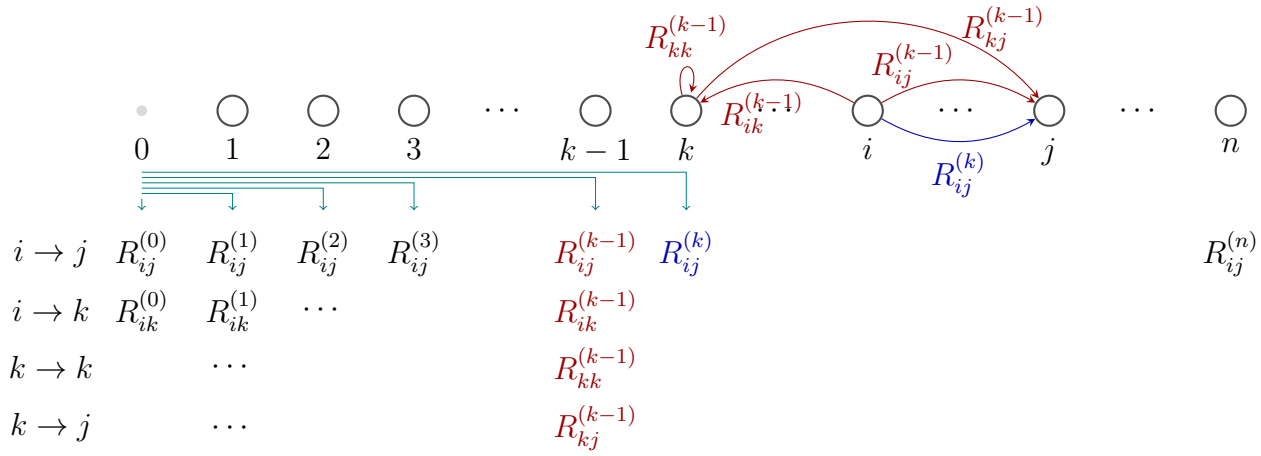


3.2.2 DFA⇒ 正则表达式, 递归构造 $R_{ij}^{(k)}$

定理 3. *如果 $L = \mathbf{L}(A)$ 是某个 DFA A 的语言, 则有一个正则表达式 R, 且 $L = \mathbf{L}(R)$.*

设 DFA A 的状态共有 n 个, 若为每个结点编号, DFA A 的状态可表示为 $\{1, 2, \ldots, n\}$. 设 $R_{ij}^{(k)} = \{x \mid \hat{\delta}(q_i, x) = q_j\}$. 也就是说, $R_{ij}^{(k)}$ 所有那样的字符串的集合 (也就是正则表达式), 即它能够使有穷自动机从状态 q_i 到达状态 q_j, 而不通过编号高于 k 的任何状态. 正则表达式 $R_{ij}^{(k)}$ 表示, 在结点 i 到 j 的全部路径中, 路径所经过的结点不超过 k 的全部路径. 但不包括起点 i 和终点 j, 即起点和终点可以和 k 无关.

如果 1 是开始结点, 则该 DFA 的正则表达式就是 $\cup_{j \in F} R_{1j}^{(n)}$.



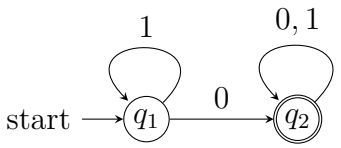
递推关系为:

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

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

示例

例 1. 把下图所示 DFA 转换为正则表达式. 这个 DFA 接受至少含有一个 0 的串.



首先 $R_{ij}^{(0)}$:

	应用公式 $R_{ij}^{(0)}$
$R_{11}^{(0)}$	$\varepsilon + 1$
$R_{12}^{(0)}$	0
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\varepsilon + 0 + 1$

简化规则:
$(\varepsilon + R)^* = R^*$
$R + RS^* = RS^*$
$\emptyset R = R\emptyset = \emptyset$ (零元)
$\emptyset + R = R + \emptyset = R$ (单位元)

计算 $R_{ij}^{(1)}$:

	应用公式 $R_{ij}^{(k)}$	化简
$R_{11}^{(1)}$	$\varepsilon + 1 + (\varepsilon + 1)(\varepsilon + 1)^*(\varepsilon + 1)$	1^*
$R_{12}^{(1)}$	$0 + (\varepsilon + 1)(\varepsilon + 1)^*0$	1^*0
$R_{21}^{(1)}$	$\emptyset + \emptyset(\varepsilon + 1)^*(\varepsilon + 1)$	$\emptyset$
$R_{22}^{(1)}$	$\varepsilon + 0 + 1 + \emptyset(\varepsilon + 1)^*0$	$\varepsilon + 0 + 1$

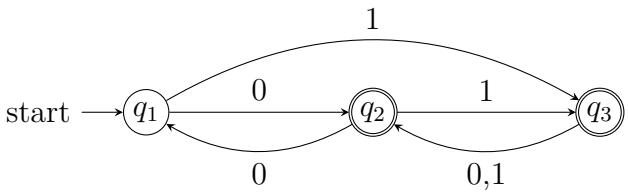
计算 $R_{ij}^{(2)}$:

	应用公式 $R_{ij}^{(k)}$	化简
$R_{11}^{(2)}$	$1^* + 1^*0(\varepsilon + 0 + 1)^*\emptyset$	1^*
$R_{12}^{(2)}$	$1^*0 + 1^*0(\varepsilon + 0 + 1)^*(\varepsilon + 0 + 1)$	$1^*0(0 + 1)^*$
$R_{21}^{(2)}$	$\emptyset + (\varepsilon + 0 + 1)(\varepsilon + 0 + 1)^*\emptyset$	$\emptyset$
$R_{22}^{(2)}$	$\varepsilon + 0 + 1 + (\varepsilon + 0 + 1)(\varepsilon + 0 + 1)^*(\varepsilon + 0 + 1)$	$(0 + 1)^*$

由于 1 是开始状态, 2 是唯一的接受状态, 结点个数是 2, 所以 DFA 的正则表达式只需要 $R_{12}^{(2)} = 1^*0(0+1)^*$.

示例

例 2. 把下图所示 DFA 转换为正则表达式.



得:

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

状态 2 和 3 是接受状态, 所以仅计算:

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

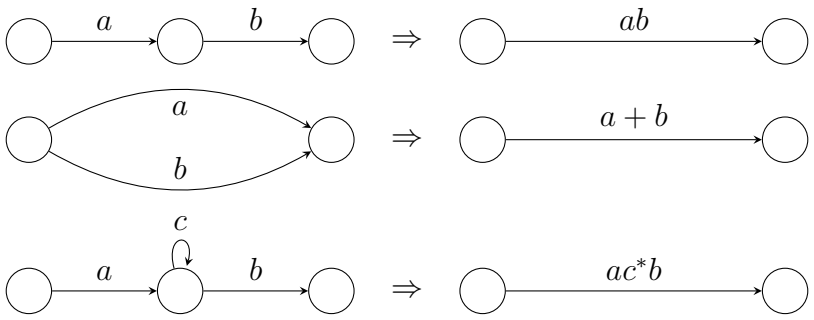
$$\begin{aligned} R_{13}^{(3)} &= R_{13}^{(2)} + R_{13}^{(2)}(R_{33}^{(2)})^* R_{33}^{(2)} \\ &= 0^*1(\varepsilon+(0+1)0^*1)^*(\varepsilon+(0+1)0^*1) + 0^*1 \\ &= 0^*1((0+1)0^*1)^* \end{aligned}$$

则 DFA 的正则表达式为:

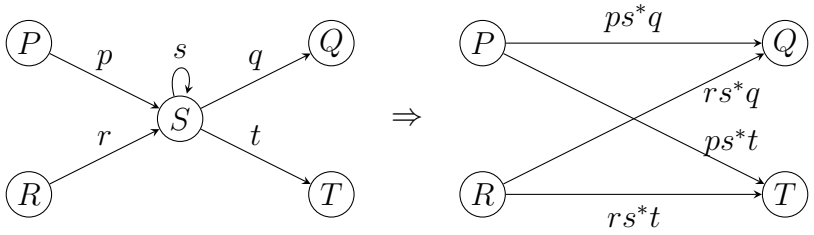
$$R_{12}^{(3)} + R_{13}^{(3)} = 0^*1((0+1)0^*1)^*(\varepsilon+(0+1)(00)^*) + 0(00)^*.$$

3.2.3 DFA⇒ 正则表达式, 状态消除

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, 在新路径上构造新的正则表达式, 产生一个等价的自动机. 那么, 删除一个状态, 可能有的最简单的情况如下三种:

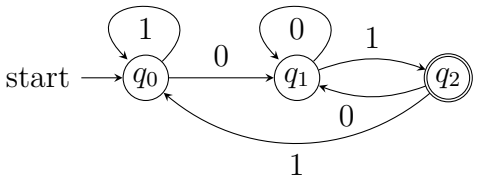


更一般的情况, 如下图, 要为被删除的状态 S 的每个“入”和“出”路径的组合, 补一条等价的新路径, 并用新的正则表达式表示:

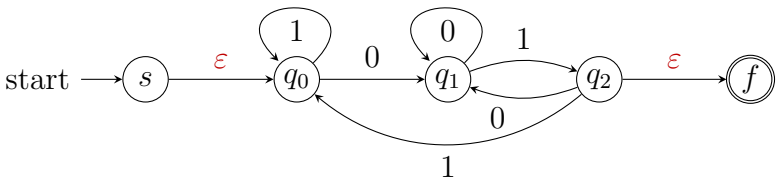


示例

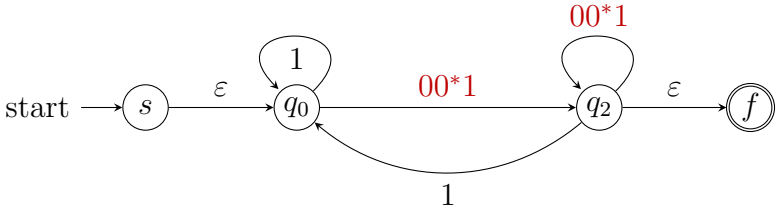
状态转移图如下, 通过状态消除构造正则表达式.



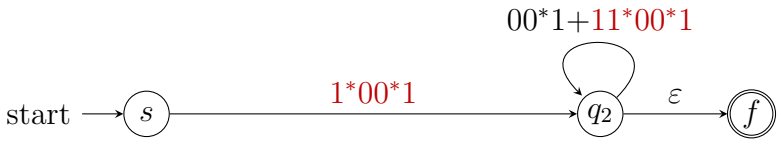
增加新的开始 (s) 和结束状态 (f) 及空转移:



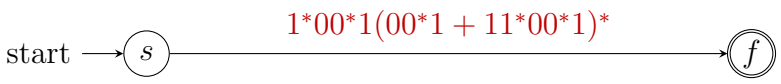
消除状态 q_1, 需要增加 $q_0 \to q_2$ 和 $q_2 \to q_2$ 两条路径:



消除状态 q_0, 需要增加 $s \to q_2$ 和 $q_2 \to q_2$ 两条路径:



消除状态 q_2, 需要增加 $s \to f$ 一条路径:



3.2.4 正则表达式 ⇒DFA, 构造 ε-NFA

定理 4. *每个由正则表达式定义的语言都可以由有穷自动机识别.*

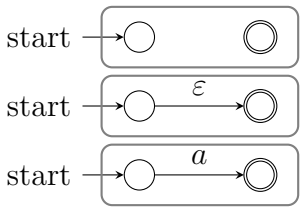
命题: 如果 R 是正则表达式, 则存在一个 ε-NFA E, 使 $\mathbf{L}(E) = \mathbf{L}(R)$, 且 E 满足 (1) 仅有一个接收状态; (2) 没有进入开始状态的边; (3) 没有离开接受状态的边.

则通过归纳法, 可以证明. 其中归纳基础为

(1) 对正则表达式 $\emptyset$, 有 ε-NFA:

(2) 对正则表达式 ε, 有 ε-NFA:

(3) $\forall a \in \Sigma$, 对正则表达式 a, 有 ε-NFA:



归纳递推为:

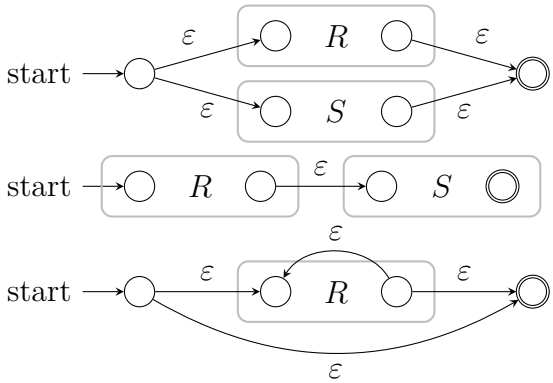
若 r 和 s 为正则表达式, 则它们对应的 ε-NFA 分别为 R 和 S

则正则表达式 $r + s$, rs 和 r^*, 则可以分别由 R 和 S 构造如下:

(1) $r + s$ 的 ε-NFA:

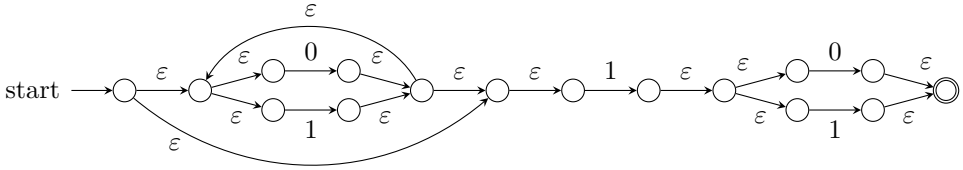
(2) rs 的 ε-NFA:

(3) r^* 的 ε-NFA:



示例

正则表达式 $(0+1)^*1(0+1)$ 构造为 ε-NFA.



3.3 正则表达式的代数定律

在保持表达的语言不变的前提下, 对表达式进行化简的代数规则.

3.3.1 结合律 (Associativity) 和交换律 (Commutativity)

- $L + M = M + L$ 并的交换律
- $(L + M) + N = L + (M + N)$ 并的结合律
- $(LM)N = L(MN)$ 连接的结合律
- 连接不满足交换律 $LM \neq ML$

3.3.2 单位元 (Identities) 与零元 (Annihilators)

- $\emptyset + L = L + \emptyset = L$ 并运算的单位元 $\emptyset$
- $\varepsilon L = L\varepsilon = L$ 连接运算的单位元 ε
- $\emptyset L = L\emptyset = \emptyset$ 连接运算的零元

3.3.3 分配率 (Distributive Laws)

- $L(M + N) = LM + LN$ 连接对并满足左分配律
- $(M + N)L = ML + NL$ 连接对并满足右分配律

示例

$$0 + 01^* = 0\varepsilon + 01^* = 0(\varepsilon + 1^*) = 01^*$$

3.3.4 幂等律 (The Idempotent Law)

- $L + L = L$ 并的幂等律

3.3.5 有关闭包的定律

- $(L^*)^* = L^*$ 对某语言的闭包再取闭包, 并不改变语言
- $\emptyset^* = \varepsilon$
- $\varepsilon^* = \varepsilon$ 空串的连接仍然是空串
- $L^* = L^+ + \varepsilon$
- $L? = \varepsilon + L$

3.3.6 发现正则表达式的定律

- $(L+M)^* = (L^*M^*)^*$
- $L^*L^* = L^*$
- $(\varepsilon + L)^* = L^*$

Chapter 4

正则语言的性质

4.1 证明语言的非正则性

4.1.1 泵引理 (Pumping Lemma)

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

示例

$L_{01} = \{0^n 1^n | n \geq 0\}$ 是否是正则语言?

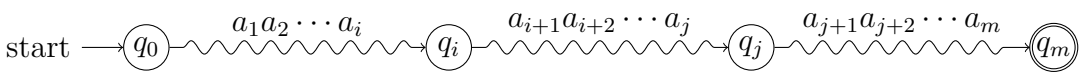
定理 5. *(正则语言的泵引理)* 若语言 L 是正则的, 则存在一个正整数 *(常数)* N, 对于任何 $w \in L$, 只要 $|w| \geq N$, 则可以把 w 分为三部分 $w = xyz$ 使得

(1) $y \neq \varepsilon$; *(或* $|y| > 0$*)*

(2) $|xy| \leq N$;

(3) 对任何 $k \geq 0$, 有 $xy^k z \in L$.

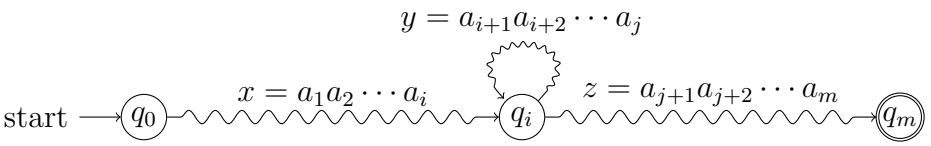
证明. 如果 L 是正则的, 则存在 DFA A, $L = \mathbf{L}(A)$. 设 A 有 n 个状态, 对于长度不小于 n 的串 $w = a_1 a_2 \cdots a_m$ $(m \geq n)$, 定义 $q_i = \hat{\delta}(q_0, a_1 a_2 \cdots a_i)$ $(i = 1, \cdots, m)$, q_0 是开始状态. 当 A 输入 w 的前 n 个字符时, 经过的状态分别是 $q_0, q_1, \cdots, q_n$ 共 $n+1$ 个, 根据鸽巢原理, 一定有两个状态相同, 即有满足 $0 \leq i < j \leq n$ 的两个状态 $q_i = q_j$.



则 w 可以被分为三个部分:

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

如果从 q_i 出发, 输入 y, 会到达 q_j, 而因为 $q_i = q_j$, 当输入 $y^k (k \geq 0)$, 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 $xyz \in \mathbf{L}(A)$, 有 $xy^k z \in \mathbf{L}(A)$ 对所有 $k \geq 0$ 成立.



又因为 $i < j$ 所以 $y \neq \varepsilon$(即 $|y| > 0$), 并且因为 i 和 j 取自 w 的前 n 个字符, 所以 $|xy| \leq n$. □

解释: 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, 可以近似的看做是状态的个数.

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

示例

证明 L_{eq} ={w | w 由数量相等的 0 和 1 构成} 不是正则的.

(思考: 能否使用 L_{eq} 的一个子集 $L_{01} = \{0^n1^n | n \geq 0\}$ 说明 L_{eq} 不是正则的?)

证明. 假设 L_{eq} 是正则的, 则一定存在正整数 N, 对任何 $w \in L_{eq}(|w| \geq N)$ 满足泵引理.

取 $w = 0^N1^N$, 则显然 $w \in L_{eq}$; 又因为 $|w| = 2N > N$,

那么有 $w = xyz$, 且 $|xy| \leq N$, $|y| > 0$; 那么 y 一定是 0^m $(m = |y| > 0)$;

根据泵引理 $xy^2z \in L_{eq}$, 但是 $xy^2z = 0^{N+m}1^N$, 则显然 $xy^2z \notin L_{eq}$;

因此与假设矛盾, 所以 L_{eq} 一定不是正则的. □

示例

证明 $L = \{0^i1^j \,|\, i > j\}$ 不是正则的.

证明. 假设 L 是正则的, 则一定存在正整数 N, 对任何 $w \in \mathbf{L}(|w| \geq N)$ 满足泵引理.

取 $w = 0^{N+1}1^N$, 所以 $w \in L$; 因为 $|w| = 2N + 1 > N$,

那么有 $w = xyz$, 且 $|xy| \leq N$, $|y| > 0$, 这里设 $|y| = m$, 那么有 $m > 0$;

根据泵引理 $xy^kz \in L$ $(k > 0)$; 但是当 $k = 0$ 时, $xy^kz = xz = 0^{N+1-m}1^N$, 而 $N + 1 - m \leq N$, 所以 $xz \notin L$;

因此与假设矛盾, 所以 L 一定不是正则的. □

示例

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

取 $w = a^{N!}$, 当 $|y| = m$ 时, 则 $|xy^2z| = N! + m$, 而 $0 \leq m \leq N < N!$, 所以 $N! < |xy^2z| = N! + m < N! + N! < N \cdot N! + N! = (N + 1)!$, 即 $|xy^2z|$ 不是阶乘数.

注意: 对于有限的语言, 比如 $\emptyset$, $\{00, 11\}$, $\{0^n1^n \,|\, 0 \leq n \leq 100\}$ 泵引理如何解释.

4.1.3 泵引理只是必要条件

“正则语言的泵引理”是正则语言的必要条件, 即“正则 $\Rightarrow$ 泵引理成立”, 所以“$\neg$ 泵引理成立 $\Rightarrow$ $\neg$ 正则”. 但是否有与正则语言等价的条件呢, 有, 即“Myhill-Nerode Theorem”.

示例

下面的语言, 每个串都可以应用泵引理, 却不是正则的

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

后一部分是正则的, 因此可以应用泵引理. 而前一部分不是正则的, 但每个串都可以利用 \$ 符号泵到后一部分中.

4.2 正则语言的封闭性

如果语言类在某些特定的运算下保持封闭, 称为这个语言类的封闭性 *(closure property)*. 正则语言的封闭性: 正则语言类中, 从某些语言经过某些运算, 得到某个语言 L, 并保持 L 还是正则的.

4.2.1 布尔运算下的封闭性

定理 6. *正则语言在并, 连接和克林闭包运算下保持封闭.*

证明. 由正则表达式的定义得证. □

定理 7. *正则语言在补运算下封闭. 即: 如果 L 是字母表 Σ 上的正则语言, 即 $L \subseteq \Sigma^*$, 则 $\overline{L} = \Sigma^* - L$ 也是正则的.*

证明. 设 DFA $A = (Q, \Sigma, \delta, q_0, F)$ 识别 L, 即 $L = \mathbf{L}(A)$. (注意 A 对不接受的输入要有 *dead state*.) 那么构造 DFA $B = (Q, \Sigma, \delta, q_0, Q - F)$, 则 $\overline{L} = \mathbf{L}(B)$. 因为任何 $w \notin L$, $\hat{\delta}(q_0, w) \notin F$. □

示例

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

由泵引理很难直接证明 L_{neq} 不是正则的, 无论如何取 w, 都无法将其打断为 $w = xyz$ 形式, 并利用 y 产生不属于 L_{neq} 的串.

而 $L_{neq} = \overline{L_{eq}}$, 而 L_{eq} 不是正则的很容易证明 (当然, 前面已经证明), 所以 L_{neq} 不是正则的.

定理 8. *正则语言在交运算下封闭.*

证明 1. 由 $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ 得证. □

证明 2. 设 DFA $A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ 和 DFA $A_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ 分别识别 L_1 和 L_2, 则构造 DFA B

$$B = (Q_1 \times Q_2, \Sigma, \delta, [q_1, q_2], F_1 \times F_2)$$

其中

$$\delta([p_1, p_2], a) = [\delta(p_1, a), \delta(p_2, a)]$$

还需证明构造是否正确, 即 $\mathbf{L}(B) = L_1 \cap L_2$, 此处略. □

示例

$L_{01} = \mathbf{L}\{0^*1^*\} \cap L_{eq}$ 若已知 L_{01} 是非正则的和 0^*1^* 是正则的, 这可以说明 L_{eq} 是非正则的. (为什么又可以用 L_{eq} 的子集说明 L_{eq} 非正则了?)

定理 9. *正则语言在差运算下封闭. 如果 L 和 M 是正则语言, 那么 $L - M$ 也是正则的.*

证明. 因为 $L - M = L \cap \overline{M}$, 得证. □

4.2.2 反转 (Reverse)

如果串 $w = a_1a_2\cdots a_n$, 称 $a_na_{n-1}\cdots a_1$ 为 w 的*反转* (*reverse*), 用 w^R 表示. 例如 $00110^R = 01100$, $\varepsilon^R = \varepsilon$. 如果 L 是一个语言, 则定义 $L^R = \{w^R \,|\, w \in L\}$.

定理 10. *如果 L 是正则的, 那么 L^R 也是正则的.*

证明. 我们证明命题“正则表达式 E, $L = \mathbf{L}(E)$, 则存在正则表达式 E^R 使得 $(\mathbf{L}(E))^R = \mathbf{L}(E^R)$”.

由正则表达式的定义, 对 E 的结构进行归纳:

首先, 如果 E 分别是 ε, $\emptyset$ 或 a, 则分别对它们去反转, $\varepsilon^R = \varepsilon$, $\emptyset^R = \emptyset$ 和 $a^R = a$, 所以分别都存在 E^R 且 $(\mathbf{L}(E))^R = \mathbf{L}(E^R)$.

其次, 对 E 所有可能三种递归的结构, 有:

(1) 如果 E 的结构是 $E = E_1 + E_2$, 则由归纳假设, 存在 E_1^R 和 E_2^R 分别有 $(\mathbf{L}(E_1))^R = \mathbf{L}(E_1^R)$ 和 $(\mathbf{L}(E_2))^R = \mathbf{L}(E_2^R)$, 那么构造 $E^R = E_1^R + E_2^R$ 则 $\mathbf{L}(E^R) = \mathbf{L}(E_1^R) \cup \mathbf{L}(E_2^R) = (\mathbf{L}(E_1))^R \cup (\mathbf{L}(E_1))^R = (\mathbf{L}(E_1) \cup \mathbf{L}(E_2))^R = (\mathbf{L}(E))^R$;

(2) 如果 E 的结构是 $E = E_1E_2$, 则构造 $E^R = E_2^RE_1^R$; 而任意两个串 w_1 和 w_2, 有 $(w_1w_2)^R = w_2^Rw_1^R$, 反之亦然, 因此有 $(\mathbf{L}(E_1E_2))^R = \mathbf{L}(E_2^RE_1^R)$;

(3) 如果 E 的结构是 $E = E_1^*$, 则构造 $E^R = (E_1^R)^*$; 因为任意 $w \in \mathbf{L}(E_1^*)$, 可以看做是 n 个串 $w_i \in \mathbf{L}(E_1)(i = 1, 2, \ldots, n)$ 的连接, 即 $w = w_1w_2\cdots w_n$, 则 $w^R = w_n^Rw_{n-1}^R\cdots w_1^R$, 而其中 $w_i^R \in \mathbf{L}(E_1^R)$, 所以 $w^R \in \mathbf{L}((E_1^R)^*)$, 反之亦然, 所以 $(\mathbf{L}(E_1^*))^R = \mathbf{L}((E_1^R)^*)$.

因此, 命题成立, 因此 L^R 也是正则语言. □

也可以通过 L 的 DFA 构造 L^R 的 ε-NFA 证明.

4.2.3 同态 (Homomorphism)

设 Σ 和 Γ 为两个字母表, 首先, 定义同态为函数 $h : \Sigma \mapsto \Gamma^*$, 即每个字符 $a \in \Sigma$, 在 h 的作用下, 替换为 Γ 上的一个串. 其次, 扩展同态函数为 $h : \Sigma^* \mapsto \Gamma^*$, 如果 Σ^* 中的串 $w = a_1a_2\cdots a_n$, 则 $h(w) = h(a_1)h(a_2)\cdots h(a_n)$. 再扩展 h 到语言, 若 L 是字母表 Σ 上一个语言, 则 $h(L) = \{h(w) \,|\, w \in L\}$.

示例

设 $\Sigma = \{0, 1\}$, $\Gamma = \{a, b\}$, 若同态函数为 $h(0) = ab$, $h(1) = \varepsilon$, 则串 0011 在 h 的作用下 $h(0011) = h(0)h(0)h(1)h(1) = abab\varepsilon\varepsilon = abab$.

对于正则表达式, 则定义 $h(E)$ 为, 将 E 中的每个符号替换后得到的表达式, 即:

$h(\emptyset) = \emptyset$ $\quad h(rs) = h(r)h(s)$
$h(\varepsilon) = \varepsilon$ $\quad h(r+s) = h(r) + h(s)$
$h(a) = h(a)$ $\quad h(r^*) = (h(r))^*$

示例

续上例, 语言 $L = 1^*0 + 0^*1$, 在上例的同态 h 的作用下, 那么
$h(1^*0 + 0^*1) = (h(1))^*h(0) + (h(0))^*h(1) = (\varepsilon)^*(ab) + (ab)^*(\varepsilon) = (ab)^*$.

定理 11. *如果 L 是字母表 Σ 上的正则语言, h 是 Σ 上的一个同态, 则 $h(L)$ 也是正则的.*

证明. 设 E 是正则表达式, 往证: $h(\mathbf{L}(E)) = \mathbf{L}(h(E))$.

对 E 的结构进行归纳, 首先, $\mathbf{L}(\varepsilon) = \varepsilon$, $\mathbf{L}(\emptyset) = \emptyset$, 以及若 $E = a$, $(a \in \Sigma)$, 则 $h(\mathbf{L}(E)) = h(\mathbf{L}(a)) = h(\{a\}) = \{h(a)\} = \mathbf{L}(h(E))$, 所以对 ε, $\emptyset$ 和 $\forall a \in \Sigma$ 命题成立.

对 E 可能的三种递归结构, 分别有:

(1) 如果 $E = F + G$:

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

(2) 如果 $E = FG$:

$$\begin{array}{rcll} h(\mathbf{L}(E)) & = & h(\mathbf{L}(F)\mathbf{L}(G)) & \text{正则表达式的定义} \\ & = & h(\mathbf{L}(F))h(\mathbf{L}(G)) & \heartsuit \\ & = & \mathbf{L}(h(F))\mathbf{L}(h(G)) & \text{归纳假设} \\ & = & \mathbf{L}(h(F)h(G)) & \text{正则表达式的定义} \\ & = & \mathbf{L}(h(FG)) & h(E) \text{ 的定义, 仅替换字符} \\ & = & \mathbf{L}(h(E)) & \end{array}$$

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

(3) 如果 $E = F^*$

略. (提示: 任意 $w \in F^*$ 可以看做 $w = w_1 w_2 \cdots w_n$, 其中 $w_i \in F$.)

□

4.2.4 逆同态 (Inverse homomorphism)

若 h 是字母表 Σ 到字母表 Γ 的同态, 并且 L 是 Γ 上的一个语言, 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

$$h^{-1}(L) = \{w \,|\, h(w) \in L\}$$

定理 12. *如果 h 是字母表 Σ 到字母表 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. 设接受 L 语言的 DFA $M = (Q, \Gamma, \delta, q_0, F)$, 构造 DFA $M' = (Q', \Sigma, \delta', q_0, F)$, 其中 $\delta'(q, a) = \hat{\delta}(q, h(a))$.

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

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

归纳递推: 若 $w=xa$, 则

$$\begin{aligned}\hat{\delta}'(q,xa) &= \delta'(\hat{\delta}'(q,x),a)\\ &= \delta'(\hat{\delta}(q,h(x)),a)\\ &= \hat{\delta}(\hat{\delta}(q,h(x)),h(a))\\ &= \hat{\delta}(q,h(x)h(a))\\ &= \hat{\delta}(q,h(xa))\end{aligned}$$

所以任意串 w, $\hat{\delta}'(q_0,w)=\hat{\delta}(q_0,h(w))$, 即 w 被 M' 接受当且仅当 $h(w)$ 被 M 接受, 即 M' 是识别 $h^{-1}(L)$ 的 DFA, 因此是正则的. □

4.3 正则语言的判定性质

正则 (或任何) 语言, 典型的 3 个判定问题:

(1) 所描述的语言是否为空?

(2) 某个特定的串 w 是否属于所描述的语言?

(3) 语言的两种描述, 是否实际上描述的是同一语言? (即语言的等价性)

要回答这样的问题, 我们想知道, 具体的算法, 是否存在.

4.3.1 空性, 有穷性和无穷性 (Emptiness, finiteness and infiniteness)

正则语言的空, 有穷和无穷, 可以通过如下定理来判定.

定理 13. *具有 n 个状态的有穷自动机 M 接受的串的集合 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, $n\le m<2n$.

证明. (1) ($\Rightarrow$) 如果 S 非空, 则 M 接受某个串, w 是 M 接受的串中长度最小的串之一, 那么一定有 $|w|<n$, 因为如果 $|w|\ge n$, 由泵引理 $w=xyz$, 那么 xz 是 M 接受的一个更短的串. ($\Leftarrow$) 显然.

(2) ($\Leftarrow$) 如果 $w\in\mathbf{L}(M)$ 且 $n\le|w|<2n$, 由泵引理, S 是无穷的. ($\Rightarrow$, 反证法) 如果 S 是无穷的, 假设没有任何一个串, 长度是 n 到 $2n-1$; 那么假设 w 是长度 $\ge 2n$ 中最短的串之一, 由泵引理 $w=xyz$, $1\le|y|<n$, 则 $xz\in\mathbf{L}(M)$; 于是, 或者 w 不是长度 $2n$ 及以上最短的, 或者 xz 长度在 n 到 $2n-1$ 之间. 两种情况之下, 都有矛盾. □

定理的第 (1) 部分, 说明存在一个算法, 判断 $\mathbf{L}(M)$ 是否为空: 只需要检查长度小 n 的串, 是否在 $\mathbf{L}(M)$ 中; 第 (2) 部分说明, 存在一个算法, 判断 $\mathbf{L}(M)$ 是否为无穷: 只需要检查长度在 n 到 $2n-1$ 的串, 是否在 $\mathbf{L}(M)$ 中.

4.3.2 等价性

定理 14. *存在一个算法, 判定两个有穷自动机是否等价 (是否接受同一语言).*

证明. 设 M_1 和 M_2 是分别接受 L_1 和 L_2 的有穷自动机, 则 $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$ 可以被某个有穷自动机接受 M_3 接受; 如果 M_3 接受某个串, 当且仅当 $L_1 \neq L_2$, 由于存在算法判断 M_3 是否空, 因此得证. □

4.4 自动机最小化

4.4.1 状态的等价性

DFA 中, 称两个状态 p 和 q 是等价的, 如果对任意串 $w \in \Sigma^*$ 满足:

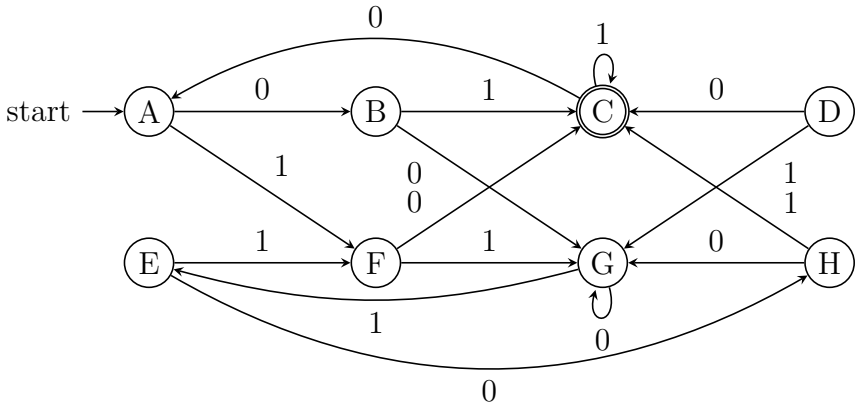
$$\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$$

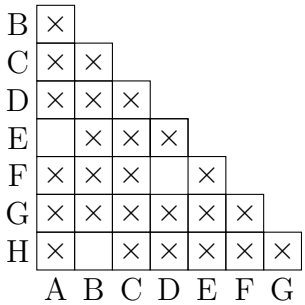
即, 只要对任何串 w, $\hat{\delta}(p, w)$ 和 $\hat{\delta}(q, w)$ 只需同时在 F 中或同时不在 F 中. 如果两个状态不等价, 则称为可区分的.

4.4.2 填表算法

填表算法递归的发现 DFA 中全部的可区分状态对:
基础: 如果 p 是接受状态而 q 是非接受状态, 则 $[p, q]$ 对是可区分的;
归纳: 如果某个 $a \in \Sigma$, 有 $[r = \delta(p, a), s = \delta(q, a)]$ 是可区分的, 则 $[p, q]$ 是可区分的.

示例





由于只有 C 是接受状态, 所以 {C}×{A,B,D,E,F,G,H}={[C,A], [C,A], [C,B], [C,D], [C,E], [C,F], [C,G], [C,H]} 都是可区分的; 状态 B,H 通过字符 1 到接受状态, 而 A,D,E,F,G 通过字符 1 都到不接受状态, 因此 {B,H}×{A,D,E,F,G} 都是可区分的; 类似的, 对字符 0, 有 {D,F}×{A,B,E,G,H} 都是可区分的; 其余的再逐个检查即可.

定理 15. *如果不能通过填表算法区分两个状态, 则这两个状态是等价的.*

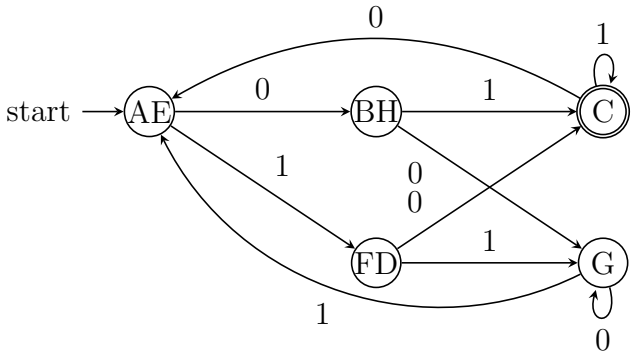
4.4.3 DFA 最小化

根据填表算法取得的 DFA A 状态间等价性, 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.

注意: 不能使用同样的方法最小化 NFA.

示例

续前例, 化简的 DFA 为



Chapter 5

上下文无关文法和上下文无关语言

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，可以表示大多数程序设计语言的语法; 实际上, 几乎所有程序设计语言都是通过上下文无关文法来定义的. 另一方面, 上下文无关文法又足够简单, 使得我们可以构造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)

5.1.1 文法的示例

自然语言

上下文无关文法最初是用来描述自然语言的, 比如如下文法规则:

$$
\begin{aligned}
&\langle \textit{sentence} \rangle \to \langle \textit{noun phrase} \rangle \ \langle \textit{verb phrase} \rangle \\
&\langle \textit{noun phrase} \rangle \to \langle \textit{adjective} \rangle \ \langle \textit{noun phrase} \rangle \\
&\langle \textit{noun phrase} \rangle \to \langle \textit{noun} \rangle \\
&\langle \textit{noun} \rangle \to \text{boy} \\
&\langle \textit{adjective} \rangle \to \text{little} \\
&\cdots
\end{aligned}
$$

算数表达式

用于产生具有 $+, *$ 的表达式的规则, 用 **id** 表示操作数

$$
\begin{aligned}
&\langle \text{expression} \rangle \to \langle \text{expression} \rangle + \langle \text{expression} \rangle \\
&\langle \text{expression} \rangle \to \langle \text{expression} \rangle * \langle \text{expression} \rangle \\
&\langle \text{expression} \rangle \to (\ \langle \text{expression} \rangle\ ) \\
&\langle \text{expression} \rangle \to \textbf{id}
\end{aligned}
$$

这种表示方法也称为 BNF(Backus-Naur Form).

示例

串 w 称为“回文 (*palindrome*)”, 当且仅当 $w = w^R$, 比如 *drawkward* (Dr. Awkward), “僧游云隐寺, 寺隐云游僧”. 那么 $\{0, 1\}$ 上的回文语言 L_{pal} 可定义为:

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

并且, 很容易证明是 L_{pal} 是非正则的. 显然 ε,0,1 都是回文, 而且如果 w 是回文, $0w0$ 和 $1w1$ 也是回文. 如果我们把这种递归的定义表示为文法, 可以如下定义:

$$P \to \varepsilon \qquad P \to 0P0$$
$$P \to 0 \qquad P \to 1P1$$
$$P \to 1$$

使用上面的文法, 可以通过 P 产生 $\{0,1\}$ 上全部的回文串, 比如串 0010100 可以通过先使用 $P \to 0P0$ 两次, 再使用 $P \to 1P1$ 一次, 再使用 $P \to 0$ 一次得到.

5.1.2 上下文无关文法的形式定义

上下文无关文法 (也称文法, *Context-Free Grammar*, CFG) G 是一个四元组 $G = (V, T, P, S)$, 其中:

(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, 用来构成语言的串, 这里 $V \cap T = \emptyset$;

(3) P: 产生式 (*Production*) 的有穷集, 表示语言的递归定义, 其中每个产生式都包括 3 部分:

a) 一个变元, 称为产生式的头 (*head*) 或左部, 是产生式中被定义的部分;

b) 一个产生式符号 $\to$;

c) 一个 $(V \cup T)^*$ 中的串, 称为产生式的体 (*body*) 或右部, 表示如何定义产生式的头;

(4) S: 初始符号 (*Start symbol*), $S \in V$, 表示文法开始的地方.

产生式 $A \to \alpha$, 读作 A 定义为 α. 如果有多个 A 的产生式 $A \to \alpha_1$, $A \to \alpha_2$, $\cdots A \to \alpha_n$, 可以简写为 $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$. 文法中定义 A 的所有产生式, 称为 A 产生式.

字符使用的一般约定: a, b 等表示终结符; A, B 等表示非终结符; x, y, z 等表示终结符串; X, Y 等表示终结符或非终结符; α, β 等表示终结符或非终结符的串.

示例

例 1. 用于产生 $\{0,1\}$ 上回文的 CFG G_{pal} 的产生式:

$$P \to \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$$

例 2. 用 **id** 表示数, 产生具有 $+$ 和 $*$ 的算数表达式的文法产生式:

$G = (\{E\}, \{\mathbf{id}, +, *, (,)\}, P, E)$, P 如下

$$E \to E + E \qquad E \to E * E \qquad E \to (E) \qquad E \to \mathbf{id}$$

例 3. $L = \{0^n 1^m \mid n \neq m\}$.

$$S \to AC \mid CB \qquad C \to 0C1 \mid \varepsilon \qquad A \to A0 \mid 0 \qquad B \to 1B \mid 1$$

例 4. $L_{eq} = \{w \in \{0,1\}^* \mid w$ 中 0 和 1 个数相等$\}$.

$$S \to 0S1 \mid 1S0 \mid SS \mid \varepsilon$$

例 5. $L_{j \geq 2i} = \{a^i b^j \mid j \geq 2i\}$.

$$S \to aSbb \mid B \qquad B \to \varepsilon \mid bB$$

5.1.3 文法的派生

分析文法和串的关系, 可以由产生式的头向产生式的体分析, 称为派生或推导 (*derivation*), 也可以由产生式的体向头分析, 称为递归推理 (*recursive inference*) 或归约 (*reduction*).

CFG $G=(V,T,P,S)$, 设 $\alpha,\beta,\gamma\in(V\cup T)^*$ 且 $A\in V$, 如果 $A\to\gamma$ 是 G 的产生式, 那么称在文法 G 中由 $\alpha A\beta$ 可派生出 $\alpha\gamma\beta$, 记为

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

如果 G 已知, 可记为 $\alpha A\beta\Rightarrow\alpha\gamma\beta$. 表示产生式 $A\to\gamma$ 应用到串 $\alpha A\beta$ 的变元 A, 得到 $\alpha\gamma\beta$. 相应的, 称在文法 G 中 $\alpha\gamma\beta$ 可归约为 $\alpha A\beta$.

设 $\alpha_1,\alpha_2,\cdots,\alpha_m\in(V\cup T)^*$, 这里 $m\geq 1$, 对 $i=1,2,\cdots,m-1$ 有 $\alpha_i\underset{G}{\Rightarrow}\alpha_{i+1}$ 成立, 即

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

则记为 $\alpha_1\overset{*}{\underset{G}{\Rightarrow}}\alpha_m$, 表示 α_1 经过零步或多步派生得到 α_m, 如果 G 已知, 可记为 $\alpha_1\overset{*}{\Rightarrow}\alpha_m$. (类似 FA 中的 δ 与 $\hat{\delta}$.) 若 α 经过 i 步派生出 β, 可记为 $\alpha\overset{i}{\Rightarrow}\beta$.

示例

算数表达式 $\mathbf{id}*(\mathbf{id}+\mathbf{id})$ 的派生过程如下:

$$\begin{aligned}E&\Rightarrow E*E\Rightarrow E*(E)\Rightarrow \mathbf{id}*(E)\Rightarrow \mathbf{id}*(E+E)\\&\Rightarrow \mathbf{id}*(E+\mathbf{id})\Rightarrow \mathbf{id}*(\mathbf{id}+\mathbf{id})\end{aligned}$$

5.1.4 最左派生和最右派生

为限制派生过程的随意性, 要求派生的每一步, 只能替换最左边变元的派生过程, 称为最左派生 (或最左推导), 记为 $\underset{\text{lm}}{\Rightarrow}$ 和 $\overset{*}{\underset{\text{lm}}{\Rightarrow}}$; 而只能替换最右变元的派生过程, 称为最右派生 (或最右推导), 记为 $\underset{\text{rm}}{\Rightarrow}$ 和 $\overset{*}{\underset{\text{rm}}{\Rightarrow}}$.

$\mathbf{id}*(\mathbf{id}+\mathbf{id})$ 的最左派生和最右派生分别为:

$$\begin{aligned}E&\underset{\text{lm}}{\Rightarrow}E*E\underset{\text{lm}}{\Rightarrow}\mathbf{id}*E\underset{\text{lm}}{\Rightarrow}\mathbf{id}*(E)\underset{\text{lm}}{\Rightarrow}\mathbf{id}*(E+E)\\&\underset{\text{lm}}{\Rightarrow}\mathbf{id}*(\mathbf{id}+E)\underset{\text{lm}}{\Rightarrow}\mathbf{id}*(\mathbf{id}+\mathbf{id})\end{aligned}$$

$$\begin{aligned}E&\underset{\text{rm}}{\Rightarrow}E*E\underset{\text{rm}}{\Rightarrow}E*(E)\underset{\text{rm}}{\Rightarrow}E*(E+E)\underset{\text{rm}}{\Rightarrow}E*(E+\mathbf{id})\\&\underset{\text{rm}}{\Rightarrow}E*(\mathbf{id}+\mathbf{id})\underset{\text{rm}}{\Rightarrow}\mathbf{id}*(\mathbf{id}+\mathbf{id})\end{aligned}$$

任何派生都有等价的最左派生和最右派生, 即 $A\overset{*}{\Rightarrow}w$ 当且仅当 $A\overset{*}{\underset{\text{lm}}{\Rightarrow}}w$ 当且仅当 $A\overset{*}{\underset{\text{rm}}{\Rightarrow}}w$.

5.1.5 文法产生的语言

由 CFG $G=(V,T,P,S)$ 产生的语言定义为

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

那么串 w 在 $\mathbf{L}(G)$ 中, 要满足:

(1) w 仅能由终结符组成;
(2) w 能由初始符号 S 派生出来.

语言 $L = \mathbf{L}(G)$ 称为**上下文无关语言** (*Context-Free Language*, CFL). 这里的上下文无关, 就是指文法派生的每一步

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

中, 串 γ 仅根据 A 的产生式派生, 而无需依赖上下文的 α 和 β. 如果有两个 CFG G_1 和 G_2, $\mathbf{L}(G_1) = \mathbf{L}(G_2)$, 则称 G_1 和 G_2 等价.

示例

描述 CFG $G = (\{S\}, \{a, b\}, \{S \to aSb, S \to ab\}, S)$ 产生的语言?

$\mathbf{L}(G) = \{a^n b^n \mid n \geq 1\}$, 因为

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow \cdots \Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^n b^n$$

示例

CFG $G = (V, T, P, S)$, 其中 $V = \{S, A, B\}$, $T = \{a, b\}$, P 如下

$$\begin{aligned} S &\to bA \mid aB \\ A &\to bAA \mid aS \mid a \\ B &\to aBB \mid bS \mid b \end{aligned}$$

能够产生 a, b 数目相等串的集合, 可以对 w 长度归纳证明.

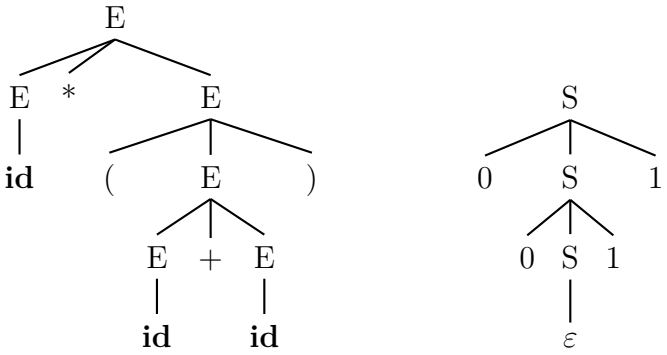
5.1.6 句型

在文法 G 中, 能由初始符号派生出来的串, 对文法本身有重要意义, 称为 G 的**句型** (*sentential form*). 即任何 $\alpha \in (V \cup T)^*$, 如果 $S \overset{*}{\Rightarrow} \alpha$, 则称为句型. 如果 $S \underset{\text{lm}}{\overset{*}{\Rightarrow}} \alpha$ 的 α 称为**左句型**, $S \underset{\text{rm}}{\overset{*}{\Rightarrow}} \alpha$ 的 α 称为**右句型**. 只含有终结符的句型, 也称为 G 的**句子**. 而 $\mathbf{L}(G)$ 就是文法 G 全部的句子.

5.2 语法分析树

语法分析树用来表示派生 (或递归推理), 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.

例如, 产生表达式 $\mathbf{id} * (\mathbf{id} + \mathbf{id})$ 和 L_{eq} 中 0011 的过程, 可以分别用语法分析树表示为



CFG $G = (V, T, P, S)$ 的**语法分析树** (*parse tree*), 也称为**派生树** (*derivation tree*), 形式定义为:

(1) 每个内节点的标记是 V 中的变元符号;

(2) 每个叶节点的标记是 $V \cup T \cup \{\varepsilon\}$ 中的符号;

(3) 如果内节点的标记是 A, 其子节点从左至右分别为

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

那么, $A \to X_1X_2\cdots X_n$ 是 P 的一个产生式; 若某个 X_i 是 ε, 则 X_i 一定是 A 唯一的子节点, 且 $A \to \varepsilon$ 是一个产生式.

语法分析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*, 并且总是能由根节点变元派生出来. 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 $\mathbf{L}(G)$.

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

5.2.1 语法树和派生的等价性

定理 16. *CFG $G=(V,T,P,S)$, 那么文法 G 中 $S \overset{*}{\Rightarrow} \alpha$ 当且仅当在文法 G 中存在以 S 为根节点产物为 α 的语法分析树.*

证明. 对任意 $A \in V$, 如果有 $A \overset{*}{\Rightarrow} \alpha$, 当且仅当存在以 A 为根且产物为 α 的语法分析树.

($\Rightarrow$) 对派生的步骤数做归纳: (1) 当仅需 1 次时, 那么一定有产生式 $A \to \alpha$, 根据语法树定义, 显然命题成立; (2) 假设派生小于等于 n 步时命题都成立, 当需要 $n+1$ 步时的第 1 步派生, 一定有 $A \to X_1X_2\cdots X_r$, 其中 X_i 的派生都不超过 n 步, 根据归纳假设, 都有一棵语法分析树, 再构造得 A 的语法分析树.

($\Leftarrow$) 对树的内部节点数做归纳: (1) 当只有 1 个内节点时, 一定是 A, 产物是 α, 所以 $A \to \alpha$ 是产生式, 所以有 $A \overset{*}{\Rightarrow} \alpha$; (2) 若假设内节点数量 $\le n$ 时命题成立, 当内节点数为 $n+1$ 时, 根节点 A 的子节点 X_i 或者是叶子, 或者是 X_i 子树, 而且每个的内节点数都 $\le n$, 所以有 $X_i \overset{*}{\Rightarrow} \alpha_i$, 而因为 $A \to X_1X_2\cdots X_n$, 所以 $A \Rightarrow X_1X_2\cdots X_n \overset{*}{\Rightarrow} \alpha_1X_2\cdots X_n \overset{*}{\Rightarrow} \alpha_1\alpha_2\cdots\alpha_n = \alpha$. □

给定 CFG $G=(V,T,P,S)$, $A \in V$, 下面 5 个命题等价:

(1) 通过递归推理, 可以确定终结符串 w 在变元 A 的语言中;

(2) $A \overset{*}{\Rightarrow} w$;

(3) $A \overset{*}{\underset{\text{lm}}{\Rightarrow}} w$;

(4) $A \overset{*}{\underset{\text{rm}}{\Rightarrow}} w$;

(5) 存在以 A 为根节点, 产物为 w 的语法分析树;

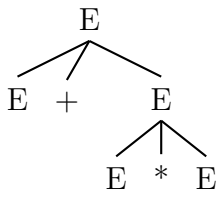
每棵语法分析树都有唯一的最左派生和唯一的最右派生. 因在 $A \Rightarrow \alpha$ 过程中, 第一步使用产生式 $A \to X_1X_2\cdots X_n$, 再 (递归的) 使用 $X_i \overset{*}{\Rightarrow} \alpha_i$ 的最左派生, 从左至右依次派生 X_i, 就得到唯一的最左派生; 同理, 若从右至左的依次使用 $X_i \overset{*}{\Rightarrow} \alpha_i$ 的最右派生, 则得到唯一的最右派生.

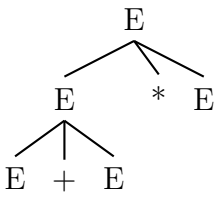
5.3 文法和语言的歧义性

如果 CFG G 使某些串有两棵不同的语法分析树, 则称 G 是*歧义* (*ambiguity*) 的, 也称*二义性的*.

示例

文法 $G=(\{E\},\{id,*,+,(,)\},\{E \to E+E \mid E*E \mid (E) \mid id\},E)$, 产生句型 $E+E*E$ 时, 会有下面两棵语法分析树:





(1) $E \Rightarrow E+E \Rightarrow E+E*E$

(2) $E \Rightarrow E*E \Rightarrow E+E*E$

5.3.1 文法歧义性的消除

有些文法的歧义性, 可以通过重新设计来文法消除. 比如考虑了运算的优先级表达式文法:

$$E \to E+T \mid T \qquad T \to T*F \mid F \qquad F \to (E) \mid \mathbf{id}$$

5.3.2 文法的固有歧义性

产生同样的语言可以有多种文法, 如果上下文无关的语言 L 的所有文法都是歧义的, 那么称 L 是**固有歧义** (*Inherent Ambiguity*) 的. 这样的语言是确实存在的.

比如语言 $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$ 是固有歧义的.

“判定任何给定 CFG G 是否是歧义的”是一个不可判定问题.

5.4 上下文无关文法的化简

典型的分析问题: 给定 CFG G 和串 w, 判断是否 $w \in \mathbf{L}(G)$. 这是编译器设计和自然语言处理的基本问题之一. 由于文法的形式非常自由, 为了便于分析和解决问题, 我们希望在不改变文法能力的前提下, 化简文法和限制文法的格式.

文法的化简主要包括:

(1) 消除**无用符号** (*useless symbols*): 不在 $S \overset{*}{\Rightarrow} w (w \in T^*)$ 的派生过程中出现的变元和终结符;

(2) 消除 ε **产生式** (ε-*productions*): 形式为 $A \to \varepsilon$, A 是变元 (得到语言 $L - \{\varepsilon\}$);

(3) 消除**单元产生式** (*unit productions*): 形式为 $A \to B$, A 和 B 都是变元.

5.4.1 消除无用符号

符号 $X \in (V \cup U)$, 如果由初始符号 $S \overset{*}{\Rightarrow} \alpha X \beta$, 称 X **是可达的** (*reachable*); 如果有 $\alpha X \beta \overset{*}{\Rightarrow} w$ $(w \in T^*)$, 称 X **是产生的** (*generating*). 如果 X 同时是产生的和可达的, 即 $S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$ $(w \in T^*)$, 称 X **是有用的**, 否则称为**无用符号**. 从文法中消除无用符号, 同时不改变文法产生的语言.

计算“产生的”符号集的算法

(1) 每个 T 中的符号都是产生的;
(2) 如果有产生式 $A \to \alpha$ 且 α 中符号都是产生的, 则 A 是产生的.

计算“可达的”符号集的算法

(1) 符号 S 是可达的;
(2) 如果有产生式 $A \to \alpha$ 且 A 是可达的, 则 α 中的符号都是可达的.

定理 17. *每个非空的 CFL 都能被一个不带无用符号的 CFG 产生.*

示例

例如文法	消除非产生的	消除非可达的
$S \to AB \mid a$	$S \to a$	$S \to a$
$A \to b$	$A \to b$	

注意: 要先消除非产生的, 再消除非可达的.

5.4.2 消除 ε-产生式

形如 $A \to \varepsilon$ 的产生式称为*空产生式*或 ε-*产生式*. 某个 CFL L, 消除其中全部的 ε-产生式后得到语言 $L - \{\varepsilon\}$ 也是 CFL. 如果变元 $A \overset{*}{\Rightarrow} \varepsilon$, 称 A *是可空的*. 则消除 ε 产生式的算法如下:

先确定全部可空的变元:

(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.

再替换全部带有可空符号的产生式, 如果 $A \to X_1X_2\cdots X_n$ 是产生式, 那么用所有的 $A \to Y_1Y_2\cdots Y_n$ 产生式代替, 其中:

(1) 若 X_i 不是可空的, 则 $Y_i = X_i$;

(2) 若 X_i 是可空的, 则 Y_i 是 X_i 或 ε;

(3) 但 Y_i 不能全部为 ε.

定理 18. *对于某个 CFG G, 有一个不带无用符号和 ε-产生式的 CFG G', 使 $\mathbf{L}(G') = \mathbf{L}(G) - \{\varepsilon\}$.*

示例

给定 CFG $G = (\{S, A, B\}, \{a, b\}, P, S)$	则 CFG $G' = (\{S, A, B\}, \{a, B\}, P', S)$
$S \to AB$	$S \to AB \mid A \mid B$
$A \to AaA \mid \varepsilon$	$A \to AaA \mid Aa \mid aA \mid a$
$B \to BbB \mid \varepsilon$	$B \to BbB \mid Bb \mid bB \mid b$

5.4.3 消除单元产生式

消除单元产生式, 先确定全部的 $A \overset{*}{\Rightarrow} B$ 的变元对 A 和 B, 如果有 $B \to \alpha$ 不是单元产生式, 就增加一条产生式 $A \to \alpha$, 再删除全部单元产生式.

定理 19. *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来定义.*

示例

消除文法的单元产生式

$S \to A \mid B \mid 0S1$ $\quad A \to 0A \mid 0$ $\quad B \to 1B \mid 1$

先确定非单元产生式, 在确定单位对 (S,A) 和 (S,B), 再带入非单元产生式

$S \to 0S1$ $\quad A \to 0A \mid 0$ $\quad B \to 1B \mid 1$

$S \to 0A \mid 0$ $\quad S \to 1B \mid 1$

5.4.4 文法简化的顺序

文法简化步骤的顺序是重要的, 一个可靠的顺序是:

(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.

5.5 乔姆斯基范式和格雷巴赫范式

文法格式的限制通过两个范式定理给出: 乔姆斯基范式 (*Chomsky Normal Form*, CNF) 定理和格雷巴赫范式 (*Greibach Normal Form*, GNF) 定理.

5.5.1 乔姆斯基范式

定理 20 (乔姆斯基范式定理)**.** *每个不带 ε 的 CFL 都可以由这样的 CFG G 产生, G 中所有产生式的形式或为 $A \to BC$, 或为 $A \to a$, 这里的 A, B 和 C 是变元, a 是终结符.*

设文法不带无用符号, ε 产生式和单元产生式. 则考虑文法每个形式为

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

的产生式, 若 X_i 是终结符 a, 则引进变元 C_a 替换 X_i 并新增产生式 $C_a \to a$. 此时, 对每个形式为

$$A \to B_1 B_2 \cdots B_m \quad (m \geq 3)$$

的产生式, 引进变元 $D_1, D_2, \cdots, D_{m-2}$ 并用一组产生式

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

来替换, 就完成了到 CNF 的转换.

利用 CNF 派生长度为 n 的串, 刚好需要 $2n-1$ 步, 因此可以设计算法判断串是否在 CFL 中. 此外利用 CNF 可以实现多项式时间的解析算法 (CYK 算法).

示例

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

$$\begin{aligned} S &\to bA \mid aB \\ A &\to bAA \mid aS \mid a \\ B &\to aBB \mid bS \mid b \end{aligned}$$

构造等价的 CNF 文法.

$$\begin{aligned} S &\to C_b A \mid C_a B \\ A &\to C_a S \mid C_b D_1 \mid a \\ B &\to C_b S \mid C_a D_2 \mid b \\ D_1 &\to AA \\ D_2 &\to BB \\ C_a &\to a \\ C_b &\to b \end{aligned}$$

5.5.2 格雷巴赫范式

定理 21 (格雷巴赫范式定理)**.** *每个不带 ε 的 CFL 都可以由这样的 CFG G 产生, G 中的每个产生式的形式为 $A \to a\alpha$, 这里 A 是变元, a 是终结符, α 是零个或多个变元的串.*

因为格雷巴赫范式的每个产生式都会引入一个终结符, 所以长度为 n 的串的派生恰好是 n 步.

示例

将文法 $S \to AB,\ A \to aA \mid bB \mid b,\ B \to b$ 转换为 GNF.

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

将文法 $S \to 01S1 \mid 00$ 转换为 GNF.

$$S \to 0ASA \mid 0B \qquad A \to 1 \qquad B \to 0$$

消除直接左递归

直接左递归 (*immediate left-recursion*) 形式为 $A \to A\alpha$, 若有 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; 消除方法是: 引入新的变元 B, 并用如下产生式替换

$$A \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \mid \beta_1 B \mid \beta_2 B \mid \cdots \mid \beta_m B$$

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

消除间接左递归

间接左递归 (*indirect left-recursion*) 形式为

$$A \to B\alpha \mid C \qquad B \to A\beta \mid D$$

派生过程可能产生 $A \Rightarrow B\alpha \Rightarrow A\beta\alpha$.

为变元重命名为 $A_1, A_2, \cdots, A_n$, 通过替换变元, 使每个 A_i 产生式, $A_i \to \alpha$ 的右半部, 或以终结符开始, 或以 A_j $(j \geq i)$ 开始; 然后消除产生的直接左递归.

Chapter 6

下推自动机

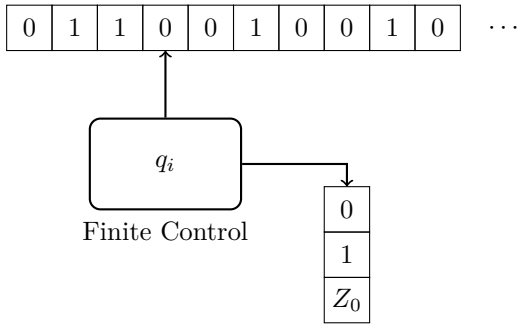
6.1 介绍

下推自动机是可以看做带有堆栈的 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



6.2 下推自动机的定义

6.2.1 形式定义

下推自动机 (*Pushdown Automata*, PDA) P 的形式定义, 为七元组 $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$:

(1) Q, 有穷状态集;

(2) Σ, 有穷输入字母表;

(3) Γ, 有穷栈字母表, 或栈符号集;

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

(5) $q_0 \in Q$, 初始状态;

(6) $Z_0 \in \Gamma - \Sigma$, 初始符号, PDA 开始时, 栈中包含这个符号的一个实例, 用来表示栈底, 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

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 $\beta_i \in \Gamma^*$, 那么映射

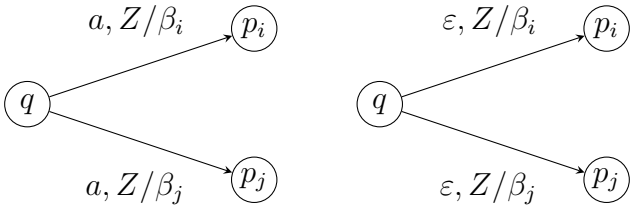
$$\delta(q, a, Z) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots, (p_m, \beta_m)\}$$

的解释是: 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 $i \neq j$, 不能同时选择 p_i 和 β_j. 而

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

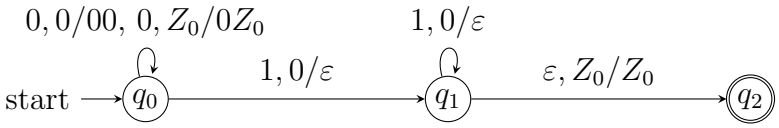
的解释是: 与扫描的输入符号无关, 只要 Z 是栈符号, 处于状态 q 的 PDA, 就可以进行上面的动作, 但输入头不向前移动.

PDA 的图形表示



示例

设计识别 $L_{0n1n} = \{0^n 1^n \mid n \geq 1\}$ 的 PDA P.



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

(1) 初始状态 (q_0, Z_0) 输入 0 或 1, 状态不变, 则直接压栈:

$$\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\},\ \delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\};$$

(2) 继续输入, 则对不同的栈顶, 仍然压栈:

$$\delta(q_0, 0, 0) = \{(q_0, 00)\},\ \delta(q_0, 1, 0) = \{(q_0, 10)\},$$
$$\delta(q_0, 0, 1) = \{(q_0, 01)\},\ \delta(q_0, 1, 1) = \{(q_0, 11)\};$$

(3) 不论栈顶是 Z_0, 0, 或 1, 开始匹配后半部分, 非确定的转移到弹栈状态:

$$\delta(q_0, \varepsilon, Z_0) = \{(q_1, Z_0)\},\ \delta(q_0, \varepsilon, 0) = \{(q_1, 0)\},\ \delta(q_0, \varepsilon, 1) = \{(q_1, 1)\};$$

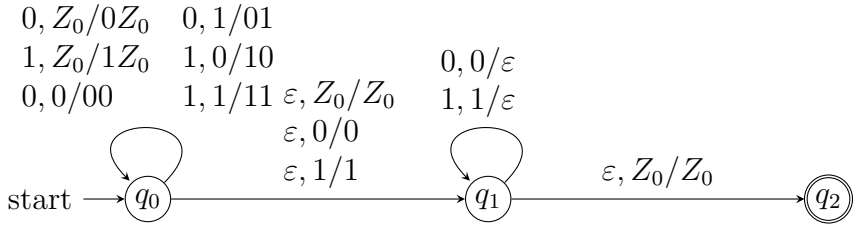
(4) 处于弹栈状态, 弹出的符号必须和输入符号一致:

$$\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\},\ \delta(q_1, 1, 1) = \{(q_1, \varepsilon)\};$$

(5) 只有看到栈底符号了, 才允许非确定的转移到接受状态:

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

状态转移图



6.2.2 瞬时描述和转移符号

瞬时描述 为了形式描述 PDA 在一个给定瞬间的格局, 定义瞬时描述 (*Instantaneous Description*, ID) 为三元组 (q, w, γ), 是 $Q \times \Sigma^* \times \Gamma^*$ 中的元素, q 表示状态, w 表示剩余的输入串, γ 表示栈中的符号串.

ID 转移符号 $\vdash_{\mathrm{P}}$ 和 $\vdash_{\mathrm{P}}^*$ 在 PDA P 中，如果 $(p, \beta) \in \delta(q, a, Z)$, 那么, 定义 ID 转移符号 $\vdash_{\mathrm{P}}$ 为

$$(q, aw, Z\alpha) \vdash_{\mathrm{P}} (p, w, \beta\alpha)$$

其中 $w \in \Sigma^*$, $\alpha \in \Gamma^*$. 并递归的定义 $\vdash_{\mathrm{P}}^*$ 为

(1) 对每个 ID I, 有 $I \vdash_{\mathrm{P}}^* I$;

(2) 对 ID I, J 和 K, 若 $I \vdash_{\mathrm{P}} J$, $J \vdash_{\mathrm{P}}^* K$, 则 $I \vdash_{\mathrm{P}}^* K$.

若 P 已知, 则可以省略, 记为 $\vdash$ 和 $\vdash^*$.

定理 22. *如果 $(q, x, \alpha) \vdash_P^* (p, y, \beta)$, 则任意 $w \in \Sigma^*$ 和任意 $\gamma \in \Gamma^*$, 有*

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

定理 23. *如果 $(q, xw, \alpha) \vdash_P^* (p, yw, \beta)$, 则 $(q, x, \alpha) \vdash_P^* (p, y, \beta)$.*

6.3 PDA 接受的语言

设 PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, 则分别定义两种接受方式下的语言如下.

以终态方式接受

P 以终态方式接受的语言 $\mathbf{L}(P)$ 是

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

以空栈方式接受

P 以空栈方式接受的语言 $\mathbf{N}(P)$ 是

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

示例

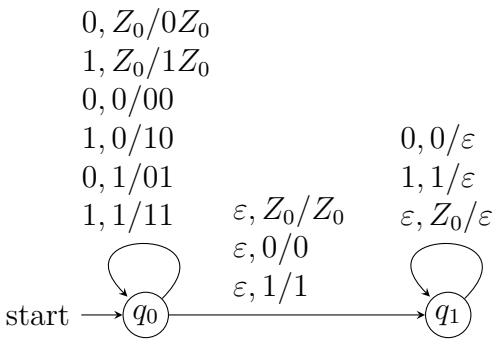
识别 L_{wwr} 的 PDA P, 从终态方式接受, 改为空栈方式接受, 只需用

$$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$$

代替

$$\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$$

即可.

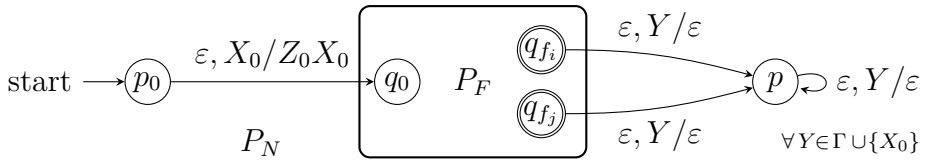


6.3.1 从终态方式到空栈方式

定理 24. *如果以终态方式接受的 PDA P_F 接受的语言 $L = \mathbf{L}(P_F)$, 那么一定存在以空栈方式接受的 PDA P_N 使 $L = \mathbf{N}(P_N)$.*

证明. 设 $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. 构造 P_N, 增加新的初始状态 p_0 和新的状态 p, 使用新的栈底符号 X_0, 并定义新的转移函数 δ_N, 即

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



其中 δ_N 定义如下:

(1) P_N 开始时, 将 P_F 栈底符号压入栈, 并准备开始模拟 P_F:

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

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

$$\delta_N(q, a, Y) \text{ 包含 } \delta_F(q, a, Y) \text{ 的诸元素}$$

(3) 从 $q_f \in F$ 开始弹出栈符号, 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, 弹出全部栈符号, 即 $\forall Y \in \Gamma \cup \{X_0\}$:

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

需要证明 $w \in \mathbf{L}(P_F) \Leftrightarrow w \in \mathbf{N}(P_N)$.

($\Rightarrow$) 如果 $w \in \mathbf{L}(P_F)$, 则有到 $q_f \in F$ 的 ID 序列

$$(q_0, w, Z_0) \vdash_{P_N}^* (q_f, \varepsilon, \gamma)$$

这些动作也是 P_F 的合法动作, 因此

$$(q_0, w, Z_0) \vdash_{P_F}^* (q_f, \varepsilon, \gamma)$$

又因为, 栈底之下增加符号不会影响这些动作, 因此

$$(q_0, w, Z_0X_0) \vdash_{P_F}^* (q_f, \varepsilon, \gamma X_0)$$

以及 P_N 在开始状态的空转移

$$(p_0, w, X_0) \vdash_{P_N} (q_0, w, Z_0X_0)$$

和 $q_f \in F$ 时, 会清空栈

$$(q_f, \varepsilon, \gamma X_0) \vdash_{P_N}^* (p, \varepsilon, \varepsilon)$$

所以

$$(p_0, w, X_0) \vdash_{P_N} (q_0, w, Z_0X_0) \vdash_{P_N}^* (q_f, \varepsilon, \gamma X_0) \vdash_{P_N}^* (p, \varepsilon, \varepsilon)$$

因此 $w \in \mathbf{N}(P_N)$. ($\Leftarrow$) 反之, 类似, 略.

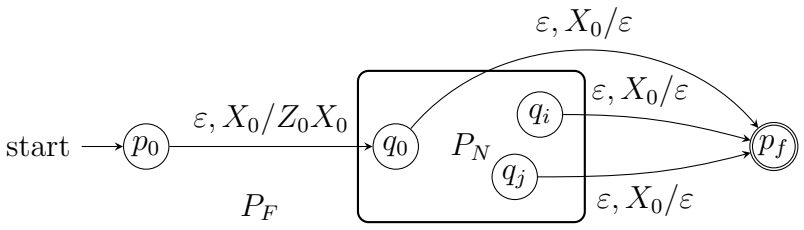
□

6.3.2 从空栈方式到终态方式

定理 25. *如果以空栈方式接受的 PDA P_N 接受的语言 $L = \mathbf{N}(P_N)$, 那么一定存在以终态方式接受的 PDA P_F 使 $L = \mathbf{L}(P_F)$.*

证明. 设 $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0, \emptyset)$. 构造 P_F, 增加新的初始状态 p_0 和新的终态 p_f, 使用新的栈底符号 X_0, 并定义新的转移函数 δ_F, 即

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



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) 在任何 $q \in Q$ 时, 看到 P_F 的栈底 X_0, 就可以转移到新终态 p_f:

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

需要证明 $w \in \mathbf{N}(P_N) \Leftrightarrow w \in \mathbf{L}(P_F)$.

($\Rightarrow$)

$$(p_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_N} (q, \varepsilon, X_0) \vdash_{P_F} (p_f, \varepsilon, \varepsilon)$$

($\Leftarrow$) 如果 $w \in \mathbf{L}(P_F)$, 由于 (3) 接受 w 的时栈符号只能是 ε, 即 ID 是 $(p_f, \varepsilon, \varepsilon)$; 那么倒数第二个 ID 只能是 (q, ε, X_0); 而因为 (1) 开始时的 ID 只能由 (p_0, w, X_0) 得到 (q_0, w, Z_0X_0); 所以有

$$(p_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_F} (q, \varepsilon, X_0) \vdash_{P_F} (p_f, \varepsilon, \varepsilon)$$

而其中 $(q_0, w, Z_0X_0) \overset{*}{\vdash}_{P_F} (q, \varepsilon, X_0)$, 因为 (2) 是 P_F 模拟 P_N 所以与 X_0 无关, 因此

$$(q_0, w, Z_0) \overset{*}{\vdash}_{P_N} (q, \varepsilon, \varepsilon)$$

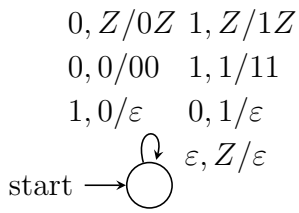
是 P_N 的合法 ID, 因此 $w \in \mathbf{N}(P_N)$.

□

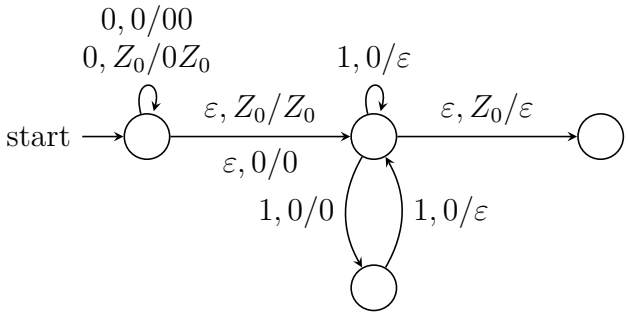
示例

所有 0 和 1 个数相同的 0 和 1 的串的集合.

以空栈方式接受:



接受 $\{0^n1^m | n \le m \le 2n\}$ 的 PDA.



6.4 PDA 与 CFG 的等价性

6.4.1 由 CFG 到 PDA

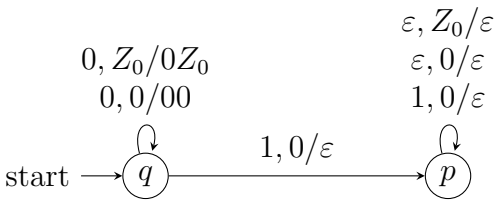
示例

识别 $L = \{0^n1^m \mid 1 \le m \le n\}$ 的 CFG 和 PDA 有

CFG $G, L = \mathbf{L}(G)$:

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

PDA $P, L = \mathbf{N}(P)$:



CFG G 有关串 00011 的文法派生过程如下:

$$S \Rightarrow AB \Rightarrow 0AB \Rightarrow 0B \Rightarrow 00B1 \Rightarrow 00011$$

PDA P 识别该串的 ID 转移序列如下:

$$\begin{aligned}(q, 00011, Z_0) &\vdash (q, 0011, 0Z_0) \vdash (q, 011, 00Z_0) \vdash (q, 11, 000Z_0) \\ &\vdash (p, 1, 00Z_0) \vdash (p, \varepsilon, 0Z_0) \vdash (p, \varepsilon, Z_0) \vdash (p, \varepsilon, \varepsilon)\end{aligned}$$

$(q_0,\ 00011,\ S)$	$\varepsilon, S/AB$	$S \to AB$	S
$\vdash (q_0,\ 00011,\ AB)$	$\varepsilon, A/0A$	$A \to 0A$	$\Rightarrow AB$
$\vdash (q_0,\ 00011,\ 0AB)$	$0, 0/\varepsilon$		$\Rightarrow 0AB$
$\vdash (q_0,\ 0011,\ AB)$	$\varepsilon, A/\varepsilon$	$A \to \varepsilon$	$\Rightarrow 0AB$
$\vdash (q_0,\ 0011,\ B)$	$\varepsilon, B/0B1$	$B \to 0B1$	$\Rightarrow 0B$
$\vdash (q_0,\ 0011,\ 0B1)$	$0, 0/\varepsilon$		$\Rightarrow 00B1$
$\vdash (q_0,\ 011,\ B1)$	$\varepsilon, B/01$	$B \to 01$	$\Rightarrow 00B1$
$\vdash (q_0,\ 011,\ 011)$	$0, 0/\varepsilon$		$\Rightarrow 00011$
$\vdash (q_0,\ 11,\ 11)$	$1, 1/\varepsilon$		
$\vdash (q_0,\ 1,\ 1)$	$1, 1/\varepsilon$		
$\vdash (q_0,\ \varepsilon,\ \varepsilon)$			

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派生过程即可.

定理 26. *如果 L 是上下文无关语言, 那么存在 PDA P, 使 $L = \mathbf{N}(P)$.*

证明. **构造 PDA**: 设 CFG $G = (V, T, P', S)$ 且 $L(G) = L$, 构造 PDA

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

其中 δ 定义:

(1) 对每个变元 A:

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

(2) 对每个终结符 a:

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

那么 P 可以模拟 G 的最左派生, 每个动作只根据栈顶的符号: 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.

充分性: 要证明 $S \overset{*}{\Rightarrow} w \implies (q, w, S) \overset{*}{\vdash} (q, \varepsilon, \varepsilon)$. 那么任意 $w \in \mathbf{L}(G)$, 则存在最左派生 $S \overset{*}{\underset{\text{lm}}{\Rightarrow}} w$, 并且除最后一步派生的 w 外, 每次派生的最左句型都有 $xA\alpha$ 的形式, 这里 $x \in T^*$, $A \in V$, $\alpha \in (V \cup T)^*$.

$$\begin{array}{ccccccccc}
S = x_1A_1\alpha_1 & \underset{\text{lm}}{\Rightarrow} & x_2A_2\alpha_2 & \underset{\text{lm}}{\Rightarrow} & \cdots & \underset{\text{lm}}{\Rightarrow} & x_{n-1}A_{n-1}\alpha_{n-1} & \underset{\text{lm}}{\Rightarrow} & x_n\alpha_n = w \\
w = x_1y_1 & = & x_2y_2 & = & \cdots & = & x_{n-1}y_{n-1} & = & x_ny_n = w \\
(q, w, S) = (q, y_1, A_1\alpha_1) & \overset{*}{\vdash} & (q, y_2, A_2\alpha_2) & \overset{*}{\vdash} & \cdots & \overset{*}{\vdash} & (q, y_{n-1}, A_{n-1}\alpha_{n-1}) & \overset{*}{\vdash} & (q, y_n, \alpha_n) \overset{*}{\vdash} (q, \varepsilon, \varepsilon)
\end{array}$$

当处于 $S \overset{*}{\underset{\text{lm}}{\Rightarrow}} w$ 的第 i 步时, 若 $x_iy_i = w$, 有:

$$x_iA_i\alpha_i \overset{*}{\underset{\text{lm}}{\Rightarrow}} x_{i+1}A_{i+1}\alpha_{i+1} \implies (q, y_i, A_i\alpha_i) \overset{*}{\vdash} (q, y_{i+1}, A_{i+1}\alpha_{i+1})$$

因为, 当最左派生处于左句型 $x_iA_i\alpha_i$ 时, ID 为 $(q, y_i, A_i\alpha_i)$, 如果有 $A_i \to \beta$ 的产生式, 则

$$x_iA_i\alpha_i \underset{\text{lm}}{\Rightarrow} x_i\beta\alpha_i$$

而 $x_i\beta\alpha_i$ 也是左句型, 所以最左的变量即为 A_{i+1}, 则 A_{i+1} 之前 x_i 之后的终结符记为 x', A_{i+1} 之后的记为 α_{i+1}, 那么就有

$$x_iA_i\alpha_i \underset{\text{lm}}{\Rightarrow} x_i\beta\alpha_i = x_ix'A_{i+1}\alpha_{i+1}$$

那么由 (1) P 模拟 $A_i \to \beta$ 的动作得到

$$(q, y_i, A_i\alpha_i) \vdash (q, y_i, \beta\alpha_i) = (q, y_i, x'A_{i+1}\alpha_{i+1})$$

而 $w = x_iy_i = x_ix'y_{i+1}$, 所以 $y_i = x'y_{i+1}$, 由 (2) P 会弹出 x', 那么

$$(q, y_i, x'A_{i+1}\alpha_{i+1}) = (q, x'y_{i+1}, x'A_{i+1}\alpha_{i+1}) \vdash^* (q, y_{i+1}, A_{i+1}\alpha_{i+1})$$

因此当 $S \underset{\text{lm}}{\overset{*}{\Rightarrow}} w$ 有 $(q, w, S) \vdash^* (q, \varepsilon, \varepsilon)$, 即 $\mathbf{L}(G) \subseteq \mathbf{N}(P)$.

必要性: 要证明 $(q, w, S) \vdash^* (q, \varepsilon, \varepsilon) \Longrightarrow S \overset{*}{\Rightarrow} w$. 我们证明更一般的结论

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

通过对 ID 转移的次数进行归纳证明. *归纳基础*: 当仅需要 1 次时, 只能是 $x = \varepsilon$ 且 $A \to \varepsilon$ 为产生式. (因为即使 $x = a$ 和产生式 $A \to a$, 也需要 2 步才能清空栈: 替换栈顶 A 为 a, 再弹出 a.) 所以 $A \overset{*}{\Rightarrow} \varepsilon$ 成立.

归纳递推: 假设转移次数不大于 $n\,(n \geq 0)$ 步时结论成立. 当需要 $n+1$ 步时, 因为 A 是变元, 其第 1 步转移一定是 $(q, x, A) \vdash (q, x, Y_1Y_2\cdots Y_m)$ 且 $A \to Y_1Y_2\cdots Y_m$ 是产生式, 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 不论是变元或终结符, 从栈中被完全弹出时, 都会消耗掉部分的 x, 记为 x_i, 那么显然有 $x = x_1x_2\cdots x_m$. 而且为了清空每个 Y_i, 需要的转移次数都不超过 n, 所以对 $i = 1, 2, \cdots, m$ 有

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

再由 A 的产生式有

$$A \underset{\text{lm}}{\Rightarrow} Y_1Y_2\cdots Y_m \underset{\text{lm}}{\overset{*}{\Rightarrow}} x_1Y_2\cdots Y_m \underset{\text{lm}}{\overset{*}{\Rightarrow}} x_1x_2\cdots Y_m \underset{\text{lm}}{\overset{*}{\Rightarrow}} x_1x_2\cdots x_m = x.$$

因此 $(q, w, S) \vdash^* (q, \varepsilon, \varepsilon) \Longrightarrow S \overset{*}{\Rightarrow} w$, 即 $\mathbf{N}(G) \subseteq \mathbf{L}(G)$. □

示例

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

构造 PDA $P = (\{q\}, \{a, b\}, \{a, b, A, S\}, \delta, q, S, \emptyset)$, 其中 δ 为

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

$\delta(q, \varepsilon, A) = \{(q, aS), (q, bS), (q, a)\}$ $\quad$ $\delta(q, b, b) = \{(q, \varepsilon)\}$.

利用 GNF 的构造方法

将文法转换为 GNF 格式的 CFG $G = (V, T, P', S)$, 那么 PDA P 的另一种构造方式为:

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

为每个产生式, 定义 δ:

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

示例

上例中的文法是 GNF, 构造 PDA $P' = (\{q\}, \{a, b\}, \{S, A\}, \delta, q, S, \emptyset)$, 其中 δ 为

$\delta(q, a, S) = \{(q, AA)\}$

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

$\delta(q, b, A) = \{(q, S)\}$.

6.4.2 由 PDA 到 CFG

定理 27. *如果 PDA P, 有 $L = \mathbf{N}(P)$, 那么 L 是上下文无关语言.*

证明. **文法构造**: 设 PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$. 那么构造 CFG $G = (V, T, P, S)$, 其中 V 是形如 $[qXp]$ 的对象和符号 S 的集合, 其中 $p, q \in Q$, $X \in \Gamma$, 产生式集合 P 包括:

(1) 为 Q 中的每个 p, 构造一个产生式:

$$S \to [q_0 Z_0 p]$$

(2) 如果 $\delta(q, a, X)$ 包括 $(p, Y_1 Y_2 \cdots Y_n)$, 构造一组产生式:

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

这里 $a \in \Sigma \cup \{\varepsilon\}$; $X, Y_i \in \Gamma$; $p, q \in Q$; 而 $r_1, r_2, \cdots r_n$ 是 Q 中各种可能的 n 个状态; 若 $i = 0$ 则构造产生式 $[qXp] \to a$.

那么

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

充分性: ID 序列 $(q, w, X) \vdash^* (p, \varepsilon, \varepsilon)$ 表示栈弹出 X 而消耗了串 w (状态从 q 到了 p); 而 $[qXp] \overset{*}{\Rightarrow} w$ 表示在 P 中经过栈符号 X 可以产生出 w (状态从 q 到了 p); 设 $w = ax$, $a \in \Sigma \cup \{\varepsilon\}$.

(1) 我们要证明

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

(2) 左边部分 ID 变化如果需多步完成, 那么第 1 步时, 一定有 $\delta(q, a, X)$ 包含 $(p, Y_1 Y_2 \cdots Y_n)$,

(i) 则第 1 步为

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

而其余步骤中, 为弹出 Y_i 会消耗 x 中的一部分 x_i, 显然 $w = ax = ax_1x_2\cdots x_n$;

(ii) 设弹出 Y_i 之前和之后 (弹出 Y_{i+1} 之前) 的状态分别是 r_{i-1} 和 r_i, 消耗的串是 x_i, 这里 $i = 1, 2, \cdots n$ 且 $r_0 = p$, 那么其他步骤就是

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

(iii) 而根据文法的构造规则, 有

$$[qXp] \Rightarrow a[pY_1r_1][r_1Y_2r_2]\cdots[r_{n-1}Y_nr_n]$$

(iv) 因此, 只要 $i = 1, 2, \cdots, n$ 有

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

成立, 就有

$$[qXp] \Rightarrow a[pY_1r_1][r_1Y_2r_2]\cdots[r_{n-1}Y_nr_n] \overset{*}{\Rightarrow} ax_1x_2\cdots x_n = w$$

成立.

(3) 而左边部分 ID 动作变化如果仅需 1 步完成 (或者说 i=0 时), 由于 $(q, a, X) \vdash (p, \varepsilon, \varepsilon)$, P 只能消耗不超过一个的字符, 即 $w = a$ ($a \in \Sigma \cup \{\varepsilon\}$), 且 (p, ε) 在 $\delta(q, a, X)$, 所以, 由文法构造规则 $[qXp] \to a$, 即

$$(q, a, X) \vdash (p, \varepsilon, \varepsilon) \Longrightarrow [qXp] \Rightarrow a$$

因此 $(q,w,X)\vdash^*(p,\varepsilon,\varepsilon) \Longrightarrow [qXp] \stackrel{*}{\Rightarrow} w$.

必要性: 略. □

示例

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) $\delta(p,0,Z)=\{(q,Z)\}$

(0)	$S\to[qZq]$	✓
	$S\to[qZp]$	消掉, 因含有 $[qZp]$
(1)	$[qZq]\to 1[qXq][qZq]$	
	$[qZq]\to 1[qXp][pZq]$	✓
	$[qZp]\to 1[qXq][qZp]$	消掉 $[qZp]$，因非产生 (与自己循环)
	$[qZp]\to 1[qXp][pZp]$	
(2)	$[qXq]\to 1[qXq][qXq]$	
	$[qXq]\to 1[qXp][pXq]$	消掉 $[pXq]$, 因无此产生式
	$[qXp]\to 1[qXq][qXp]$	
	$[qXp]\to 1[qXp][pXp]$	✓
(3)	$[qXq]\to 0[pXq]$	
	$[qXp]\to 0[pXp]$	✓
(4)	$[qZq]\to\varepsilon$	✓
(5)	$[pXp]\to 1$	✓
(6)	$[pZp]\to 0[qZp]$	
	$[pZq]\to 0[qZq]$	✓

6.5 确定型下推自动机 (DPDA)

PDA $P=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ 是确定型下推自动机 (*DPDA*), 当且仅当:

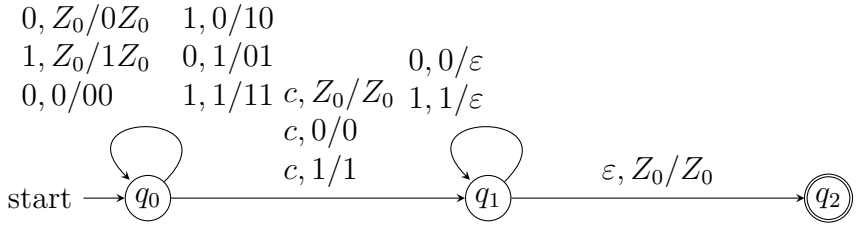
(1) $\delta(q,a,X)$ 至多有一个动作, 这里 $a\in\Sigma\cup\{\varepsilon\}$;

(2) $\forall a\in\Sigma$, 如果 $\delta(q,a,X)\neq\emptyset$, 那么 $\delta(q,\varepsilon,X)=\emptyset$.

即 $\forall(q,a,Z)\in Q\times\Sigma\times\Gamma$, $|\delta(q,a,Z)|+|\delta(q,\varepsilon,Z)|\le 1$. 下面给出关于这种装置的一些事实.

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.

任何 DPDA 都无法接受 L_{wwr}, 但是可以接受 $L_{wcwr}=\{wcw^R \mid w\in(0+1)^*\}$.



6.5.1 RL 与 DPDA

定理 28. *如果语言 L 是正则的, 那么有 DPDA P 以终态方式接受 L, 即 $L = \mathbf{L}(P)$.*

DPDA P 可以不使用栈, 而仅模拟 DFA 即可. 又因为 L_{wcwr} 显然是 CFL 不是正则语言, 所以 $\mathbf{L}(P)$ 语言类真包含正则语言.

定理 29. *DPDA P 且 $L = \mathbf{N}(P)$, 当且仅当 L 有前缀性质, 且存在 DPDA P' 使 $L = \mathbf{L}(P')$.*

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 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.

6.5.2 DPDA 与 CFL

DPDA P 无法识别 L_{wwr}. 所以 $\mathbf{L}(P)$ 语言类真包含于上下文无关语言.

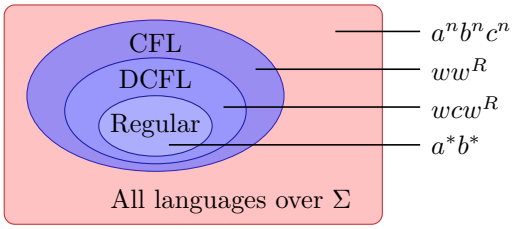
6.5.3 DPDA 与歧义文法

定理 30. *如果有 DPDA P, 语言 $L = \mathbf{L}(P)$, 那么 L 有无歧义的 CFG.*

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

证明略. DPDA 也因此在语法分析中占重要地位. 但是并非所有非固有歧义 CFL 都会被 DPDA 识别. 例如 L_{wwr} 有无歧义文法 $S \to 0S0 \mid 1S1 \mid \varepsilon$.

6.5.4 语言间的关系



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

定理 32 (上下文无关语言的泵引理)**.** *设 L 是任意 CFL, 那么存在常数 N, 它仅依赖于 L, 使得若 $z \in L$, $|z| \geq N$, 则总可以将 z 写成 $z = uvwxy$, 满足:*

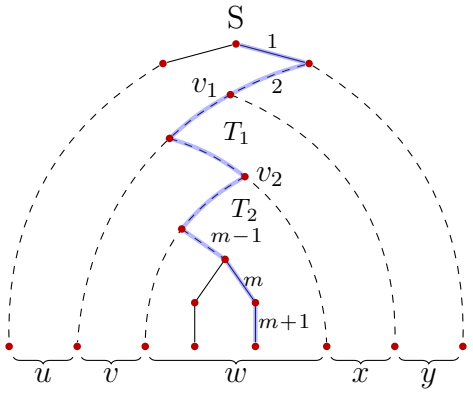
(1) $|vwx| \leq N$;

(2) $vx \neq \varepsilon$*(或* $|vx| \neq 0$*)*;

(3) 对任意 $i \geq 0$, $uv^i wx^i y \in L$.

证明. 设 $G = (V, T, P, S)$ 是接受 $L - \{\varepsilon\}$ 的 CNF 文法. 在 CNF 文法的派生树中, 若最长路径为 k, 则产物的长度最多为 2^{k-1}. 设 G 变元数为 m, $N = 2^m$, 那么若有 $z \in \mathbf{L}(G)$, $|z| \geq N$, 则 z 的派生树中最长路径长度至少也是 $m+1$, 这个路径上有至少 $m+2$ 个节点, 除最后一个节点外, 其余标记都是变元. 只考虑在接近树底部连续的 $m+1$ 个变元标记, 其中至少有两个是相同的.

如果这两个节点分别是 v_1 和 v_2, 标记均为 A, v_1 比 v_2 更接近树根. 设以 v_1 为根的子树为 T_1, 它的产物 z_1 长度不会超过 2^m, 因为 T_1 最长路径不超过 $m+1$. 设以 v_2 为根的子树为 T_2 产物为 w, 那么 $z_1 = vwx$. 而且 v 和 x 不能同时为空, 因为派生 z_1 的第一个产生式必须是 $A \to BC$, T_2 不是完全处于 B 中就是完全处于 C 中, 而 B 和 C 都至少产生一个终结符.



那么可以得到

$$A \overset{*}{\Rightarrow} vAx \quad \text{和} \quad A \overset{*}{\Rightarrow} w$$

而且 $|vwx| = |z_1| \leq 2^m = N$. 所以对任意 $i \geq 0$, $A \overset{*}{\Rightarrow} v^i wx^i$. 那么串 z 可以写成 $uvwxy$, u 和 y 为某个串, 即 $S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uv^i wx^i y$. □

7.1.1 CFL 泵引理的应用

示例

证明 $L = \{0^n 1^n 2^n \mid n \geq 1\}$ 不是上下文无关语言.

证明. 假设 L 是上下文无关的, 那么存在整数 N. 取 $z = 0^N 1^N 2^N$. 由泵引理, $z = uvwxy$, 其中 $|vwx| \leq N$, $vx \neq \varepsilon$. 如果 vwx 只包含 0, 1 或 2, 那么 uwy 不在 L 中; vwx 若只包含 0 和 1, 或只包含 1 和 2, uwy 也不在 L 中. 而由于泵引理 $uwy = uv^0wx^0y \in L$, 因此假设不成立, L 不是上下文无关的. □

证明 $L = \{a^i b^j c^i d^j \mid i \geq 1 \text{ and } j \geq 1\}$ 不是上下文无关的. (取 $z = a^n b^n c^n d^n$.)

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 $uv^i wx^i y \in L$, 满足泵引理. □

(正确的) 证明. 假设 L 是 CFL. 取 $z = 0^N 1^N 0^N 1^N$, 那么 $z = uvwxy$ 时

(1) 若 vwx 在 z 中点的任意一侧, uv^0wx^0y 显然不可能属于 L;

(2) 若 vwx 中包括 z 中点, 那么 uv^0wx^0y 只能形如 $0^N 1^i 0^j 1^N$, 也不可能属于 L.

所以假设不成立. □

7.2 上下文无关语言的封闭性

7.2.1 代换

代换 (*substitution*) 是映射 $s : \Sigma \mapsto 2^{\Gamma^*}$. Σ 中的一个字符 a 在 s 的作用下成为语言 L_a, 即

$$s(a) = L_a.$$

再推广 s 到 Σ 的字符串:

$$\begin{aligned} &(1)\ \ s(\varepsilon) = \varepsilon \\ &(2)\ \ s(xa) = s(x)s(a) \end{aligned}$$

再推广 s 到 Σ 的语言 L:

$$s(L) = \bigcup_{x \in L} s(x).$$

定理 33. 上下文无关语言在代换下封闭.

说明: 如果 L 是 Σ 上的上下文无关语言, s 是 Σ 上的代换, 且每个 $a \in \Sigma$, $s(a)$ 都是 CFL, 那么 $s(L)$ 是 CFL.

证明. **文法构造**: 若任意 $a \in \Sigma$, $s(a)$ 都是 CFL, 那么设 $s(a)$ 的文法为 $G_a = (V_a, T_a, P_a, S_a)$; 设 L 的文法 $G = (V, T, P, S)$. 那么 $s(L)$ 的文法可以构造为 $G' = (V', T', P', S)$:

(1) $V' = (\bigcup_{a \in T} V_a) \cup V$

(2) $T' = \bigcup_{a \in T} T_a$

(3) P' 包括:

(i) 每个 P_a 中的产生式;

(ii) P 的产生式, 但要替换产生式中的终结符 a 为 S_a.

那么, 需证明 $s(L) = \mathbf{L}(G')$.

充分性 ($s(L) \subseteq \mathbf{L}(G')$): 对 $\forall w \in s(L)$, 那么一定存在某个 $x \in L$ 使 $w \in s(x)$. 设 $x = a_1 a_2 \cdots a_n$ 即

$$w \in s(x) = s(a_1)s(a_2)\cdots s(a_n),$$

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 $w \in \mathbf{L}(G')$.

必要性 ($\mathbf{L}(G') \subseteq s(L)$): 对 $\forall w \in \mathbf{L}(G')$, 有 $S \overset{*}{\underset{G'}{\Rightarrow}} w$, 又因为 w 中每个终结符仅能由某个 S_a 派生出来, 所以存在仅由 S_a 构成的句型 α, 有

$$S \overset{*}{\underset{G'}{\Rightarrow}} \alpha \overset{*}{\underset{G'}{\Rightarrow}} w.$$

不妨设 $\alpha = S_{a_1} S_{a_2} \cdots S_{a_n}$, 那么因为 $S \overset{*}{\underset{G'}{\Rightarrow}} \alpha$, 所以

$$S \overset{*}{\underset{G'}{\Rightarrow}} a_1 a_2 \cdots a_n$$

那么 $x = a_1 a_2 \cdots a_n \in L$. 而又因为 $\alpha = S_{a_1} S_{a_2} \cdots S_{a_n} \overset{*}{\underset{G'}{\Rightarrow}} w$, 所以 w 可以分为 $w = w_1 w_2 \cdots w_n$, 且对 $i = 1, 2, \cdots, n$ 有

$$S_{a_i} \overset{*}{\underset{G'}{\Rightarrow}} w_i,$$

所以 $w_i \in s(a_i)$, 那么

$$w = w_1 w_2 \cdots w_n \in s(a_1)s(a_2)\cdots s(a_n) = s(a_1 a_2 \cdots a_n) = s(x),$$

所以 $w \in s(L)$. □

示例

设 $L = \{w \mid w$有相等个数的a和$b\}$, 代换 $s(a) = L_a = \{0^n 1^n \mid n \geq 1\}$, $s(b) = L_b = \{ww^R \mid w \in (0+1)^*\}$, 求 $s(L)$ 的文法.

设计 L 的文法为: $S \to aSbS \mid bSaS \mid \varepsilon$

设计 L_a 的文法为: $S_a \to 0S_a1 \mid 01$

设计 L_b 的文法为: $S_b \to 0S_b0 \mid 1S_b1 \mid \varepsilon$

那么 $s(L)$ 的文法为:

$S \to S_a S S_b S \mid S_b S S_a S \mid \varepsilon$

$S_a \to 0S_a1 \mid 01$

$S_b \to 0S_b0 \mid 1S_b1 \mid \varepsilon$

7.2.2 并, 连接, 闭包, 同态/逆同态, 反转

定理 34. *上下文无关语言在并, 连接, 闭包, 正闭包, 同态运算下封闭.*

证明. 若 $\Sigma = \{1, 2\}$, 语言 $\{1, 2\}$, $\{12\}$, $\{1\}^*$ 和 $\{1\}^+$ 显然都是 CFL. 设 L_1 和 L_2 是任意的 CFL, 并定义代换 $s(1) = L_1$, $s(2) = L_2$, 那么:

(1) 因为 $s(\{1, 2\}) = s(1) \cup s(2) = L_1 \cup L_2$, 所以并运算下封闭;

(2) 因为 $s(\{12\}) = s(12) = s(\varepsilon)s(1)s(2) = L_1L_2$, 所以连接运算下封闭;

(3) 因为

$$\begin{aligned} s(\{1\}^*) &= s(\{\varepsilon, 1, 11, 111, \cdots\}) \\ &= s(\varepsilon) \cup s(1) \cup s(11) \cup s(111) \cup \cdots \\ &= s(\varepsilon) \cup s(1) \cup s(1)s(1) \cup s(1)s(1)s(1) \cup \cdots \\ &= \{\varepsilon\} \cup L_1 \cup L_1L_1 \cup L_1L_1L_1 \cup \cdots \\ &= (s(1))^* = L_1^* \end{aligned}$$

所以闭包运算下封闭 (正比包, 同理).

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

$$h(L) = \{h(w) \mid w \in L\} = \bigcup_{w \in L} \{h(w)\} = \bigcup_{w \in L} s(w) = s(L),$$

所以在同态运算下封闭. □

也可以使用文法来证明 CFL 并, 连接, 闭包的封闭性.

证明. 若 L_1 和 L_2 是 CFL, 那么设文法分别为 $G_1 = (V_1, T_1, P_1, S_1)$ 和 $G_2 = (V_2, T_2, P_2, S_2)$. 那么

(1) $L_1 \cup L_2$ 的文法为

$$G_{union} = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}, S);$$

(2) L_1L_2 的文法为

$$G_{concat} = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1S_2\}, S);$$

(3) L_1^* 的文法为

$$G_{closure} = (V_1 \cup \{S\}, T_1, P_1 \cup \{S \to S_1S \mid \varepsilon\}, S).$$

再证明所构造文法的正确性, 略. □

定理 35 (反转)**.** *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. □

定理 36. *CFL 在逆同态下封闭.*

证明. (构造部分)

已知 L 是字母表 Δ 上的 CFL, h 是 Σ 到 Δ^* 的同态. 设 PDA $P = (Q, \Delta, \Gamma, \delta, q_0, Z_0, F)$ 有 $L = \mathbf{L}(P)$. 构造识别 $h^{-1}(L)$ 的 PDA P' 使用缓冲暂存 a ($a \in \Sigma$) 的同态串 $h(a)$, 然后利用 P 的状态和缓冲中未消耗的 $h(a)$, 即其后缀, 形成的二元组作为 P' 的当前状态. 构造如下

$$P' = (Q', \Sigma, \Gamma, \delta', [q_0, \overline{\varepsilon}], Z_0, F \times \{\overline{\varepsilon}\})$$

其中

(1) 有限状态集 $Q' \subset Q \times \Delta^*$ 中的状态为 $[q, \overline{x}]$, 用 q 模拟 P 的状态, $\overline{x}$ 模拟缓冲;

(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.

(证明部分) 略. □

7.2.3 交, 补

CFL 在交运算下不封闭

例如, 语言 L_1 和 L_2 分别为

$$L_1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$$
$$L_2 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$$

都是 CFL, 而

$$L = \{0^n 1^n 2^n \mid n \geq 1\} = L_1 \cap L_2$$

不是 CFL.

CFL 在补运算下不封闭

因为 $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, 以及 CFL 在并运算下封闭, 而在交运算下不封闭.

定理 37. *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 $P' = (Q, \Sigma, \Gamma, \delta, q_0, F)$ 如下:

(1) $Q = Q_1 \times Q_2$

(2) $q_0 = [q_1, q_2]$

(3) $F = F_1 \times F_2$

(4) δ 为

$$\delta([p, q], a, Z) = \begin{cases} \{([p, s], \beta) \mid (s, \beta) \in \delta_2(q, a, Z)\} & \text{when } a = \varepsilon \\ \{([r, s], \beta) \mid r = \delta_1(p, a) \text{ and } (s, \beta) \in \delta_2(q, a, Z)\} & \text{when } a \neq \varepsilon \end{cases}$$

那么 $\mathbf{L}(P') = L \cap R$. 证明略. □

7.2.4 封闭性的应用

语言 $L = \{ww \mid w \in (a+b)^*\}$ 不是 CFL, 可以利用封闭性和不是 CFL 的 L' 来证明. 因为

$$L \cap a^+b^+a^+b^+ = L' = \{a^ib^ja^ib^j \mid i \geq 1, j \geq 1\}$$

因为 L' 不是 CFL, 所以 L 不是 CFL.

示例

证明 $L = \{w \mid w \in \{a,b,c\}^*, n_a = n_b = n_c\}$ 不是上下文无关语言, 其中 n_a 表示 w 中 a 的个数.

证明. 假设 L 是 CFL, 而 $L \cap a^*b^*c^* = L' = \{a^nb^nc^n \mid n \geq 0\}$, 所以 L' 也应该是 CFL, 但显然不是，所以假设不成立，L 不是 CFL □

7.3 上下文无关语言的判定性质

7.3.1 可判定的 CFL 问题

测试 CFL 的空性: 只需判断文法的开始符号 S 是否是产生的.

测试 CFL 的成员性: 利用 CNF 范式, 有 CYK 算法检查串 w 是否属于 L.

7.3.2 不可判定的 CFL 问题

与 CFL 有关的几个不可判定问题:

1. 判断给定 CFG G 的歧义性.
2. 判断给定 CFL 的固有歧义性.
3. 判断两个 CFL 的交是否为空.
4. 判断两个 CFL 是否相同.
5. 判断给定 CFL 的补是否为空. (尽管有算法判断 CFL 是否为空.)
6. 判断给定 CFL 是否等于 Σ^*.

7.4 乔姆斯基文法体系

文法 $G = (V, T, P, S)$, P 中的产生式都形如

$$\alpha \to \beta$$

其中 $\alpha \in (V \cup T)^*V(V \cup T)^*$, 即 α 中至少有一个变元, $\beta \in (V \cup T)^*$:

(1) 则 G 称为 0 型文法, 或短语结构文法 (PSG); $L(G)$ 称为 0 型语言, 短语结构语言 (PSL), 或递归可枚举语言;

(2) 若要求 $|\beta| \geq |\alpha|$, 则称 G 为 1 型文法, 或上下文有关文法 (*Context-Sensitive Language*, CSL); $L(G)$ 称为 1 型语言或上下文有关语言 (CSL);

(3) 若要求 $\alpha \in V$, 则称 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) 若要求 $\alpha \to \beta$ 都是形如 $A \to aB$ 或 $A \to a$, 其中 $A \in V$, $a \in T$, 则称 G 是 3 型文法或正则文法; $L(G)$ 称为 3 型语言或正则语言.

乔姆斯基把文法分成这 4 种类型, 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

8.1 图灵机

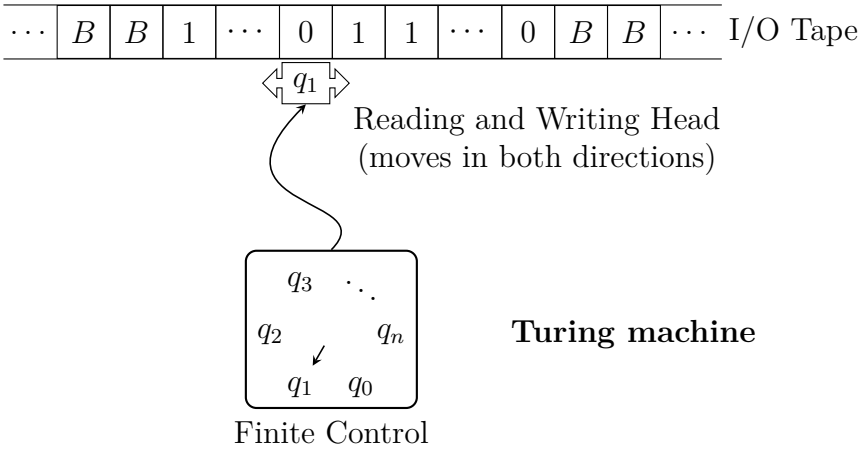
从大约公元前 3 世纪起, 人们就有了算法 (*algorithm*) 的直觉概念, 并寻找解决各种数学问题的算法, 最著名的就是欧几里德算法 (辗转相除法). 20 世纪初, 希尔伯特 (D. Hilbert) 打算寻求一个机械的有效过程, 用以确定任何数学命题的真和假. 特别的, 寻找这样的过程, 用以确定整数上的一阶谓词演算中任意公式是否为真. 希尔伯特所要寻找的有效过程就是算法, 但当时算法的概念还没有被形式化. 1931 年, 哥德尔 (K.Gödel) 发表了著名的不完全性定理, 构造了一个公式, 该公式在这个逻辑系统中既不能被证明也不能被证否, 所以不可能存在通用的过程, 证明任何命题的真假.

而什么是算法呢, 在研究可计算的整数函数的过程中, 数理逻辑学家给出了几种不同的定义. 20 世纪 30 年代, 美国的丘奇 (A. Church) 提出了 λ-演算, 哥德尔和克林 (S. Kleene) 给出了递归演算系统, 并由此定义了递归函数类. 同时, 在英国的图灵 (A. M. Turing) 也在研究可计算的本质. 图灵分析了人类进行算法演算的过程, 并定义了用来模拟这一过程的机器, 即图灵机. 当人们在用纸和笔进行计算时, 要在纸的一定部位写上或擦去所用的符号, 人的眼光在纸上移动以进行计算, 并需要根据一定的规则进行每一步的计算. 尽管这个机器很简单, 但是图灵断言它在功能上等价于一个进行数学运算的人.

图灵机既可以作为语言的识别器, 也可以作为整数函数的计算器. 它的运算过程真正体现了机械而有效的特点, 虽然与其等价的递归演算系统和 λ-演算也都反映了计算的本质, 但和图灵机相比, 似乎都需要更多的智慧. 也是因为图灵和丘奇的工作, 算法的概念才首次被形式化的定义.

8.1.1 形式定义

图灵机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. 带划分为单元格, 每个单元格可以放置一个符号, 带头每次根据当前状态和带头处单元格的符号内容, 根据转移规则选择下一个动作, 每个动作都包括下一个状态, 修改带头处单元格的符号以及带头向左或向右移动一个单元格.



图灵机 (TM) M 的形式定义为七元组:

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

其中

(1) Q: 有穷状态集;

(2) Σ: 有穷字母表;

(3) Γ: 有穷带符号集 (*tape symbols*), 总有 $\Sigma \subset \Gamma$;

(4) δ: $Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$, 状态转移函数, L 和 R 表示带头向左和向右的移动;

(5) $q_0 \in Q$: 开始状态;

(6) $B \in \Gamma - \Sigma$: 空格 (*blank*) 符号, 开始时, 带上除输入字符串, 其余都是空格;

(7) $F \subseteq Q$: 终态集或接受状态集.

8.1.2 瞬时描述

图灵机有无穷长的带, 但是在有限步移动之后, 带上的非空格内容是有限个的. 因此用带上最左边到最右边的非空格内容, 状态和带头的位置, 同时来定义瞬时描述 (ID) 或称格局 (*Configuration*), 即表示为

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n$$

其中的信息包括

(1) 图灵机所处的状态 q;

(2) 带头在左起第 i 个非空格符号上;

(3) $X_1 X_2 \cdots X_n$ 是最左到最右非空格内容. (也可能一端有空格符号, 比如 i 在 1 或 n 时.)

使用转移符号 $\vdash_{\mathrm{M}}$ 和 $\vdash_{\mathrm{M}}^*$ 表示图灵机的移动 (*move*), 若 M 已知, 记为 $\vdash$ 和 $\vdash^*$. 如果 $\delta(q, X_i) = (p, Y, L)$, 那么

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash_{\mathrm{M}} X_1 X_2 \cdots X_{i-2} p X_{i-1} Y X_{i+1} \cdots X_n$$

如果 $\delta(q, X_i) = (p, Y, R)$ 那么

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash_{\mathrm{M}} X_1 X_2 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$$

但在当 i 为 1 或 n 时的空格符号要视情况记入 (或不计入) 下一个 ID.

示例

设计识别 $\{0^n1^n \mid n \geq 1\}$ 的图灵机.

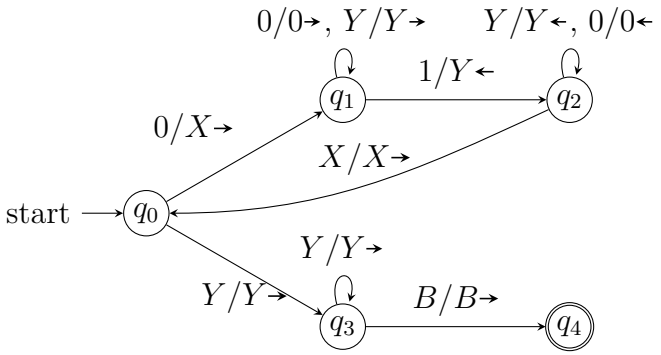
$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

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	–	–	–	–	–

状态 q_0, q_1, q_2 的一个循环, 将一对 0, 1 改为 X, Y, 然后指向下一个 0 准备下次循环; 若发现当前是 1 则停机 (无动作定义) 并拒绝该串; 若发现当前是 Y, 循环 q_3, 略过 Y 一直向右移动, 直到发现 B, 进入 q_4 接受该串并停机. 例如, 接受 0011 的 ID 序列为

$$\begin{aligned} q_00011 &\vdash Xq_1011 \quad \vdash X0q_111 \quad \vdash Xq_20Y1 \quad \vdash q_2X0Y1 \quad \vdash Xq_00Y1 \\ &\vdash XXq_1Y1 \vdash XXYq_11 \quad \vdash XXq_2YY \quad \vdash Xq_2XYY \vdash XXq_0YY \\ &\vdash XXYq_3Y \vdash XXYYq_3B \vdash XXYYBq_4B \end{aligned}$$

状态转移图为



示例

构造接受 $L = \{a^nb^nc^nd^n \mid n \geq 0\}$ 的图灵机.

8.1.3 语言与停机

图灵机 M 接受的语言 $\mathbf{L}(M)$

$$\mathbf{L}(M) = \{w \mid w \in \Sigma^*,\ q_0w \overset{*}{\vdash} \alpha p\beta,\ p \in F,\ \alpha, \beta \in \Gamma^*\}.$$

输入串放在输入带上, 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.

一般假定当输入串被接受时 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, TM 可能永远不停止. 我们永远也不会知道, 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.

能够被图灵机接受的语言类, 称为**递归可枚举的** (*recursively enumerable*, RE). “可枚举”的意思是这些语言中的串可以被某个图灵机枚举出来. 这个语言类中包含某些语言 $\mathbf{L}(M)$, 在不属于 $\mathbf{L}(M)$ 的某些输入上 M 停不下来.

在不接受的输入上也能保证停机的图灵机, 所接受的语言称为**递归的**, 因此递归的语言是递归可枚举语言的一个子类. 而能保证停机的图灵机, 正是算法的好模型, 这也是算法概念的首次形式化, 并由此建立了计算机科学的基础.

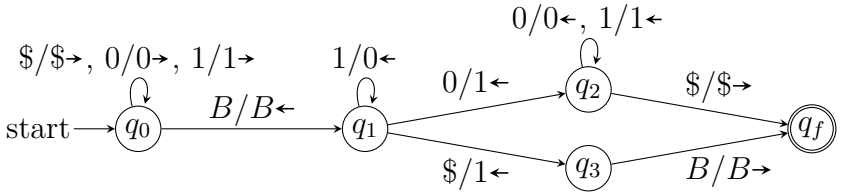
8.1.4 整数计算器

图灵机可以作为语言的识别器, 也可以用作整数函数计算器和语言的枚举器.

示例

二进制数的加 1 函数, 用符号 \$ 作为数前的占位标记. 例如 $q_0\$10011 \vdash^* \$q_f 10100$, $q_0\$111 \vdash^* q_f 1000$.

$$M = (\{q_0, q_1, q_2, q_3, q_f\}, \{0, 1\}, \{0, 1, \$, B\}, \delta, q_0, B, \{q_f\})$$



8.2 扩展的图灵机

TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$.

8.2.1 状态中存储

设计状态可以存储字符的图灵机:

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

其中 $Q' = Q \times \Gamma \times \cdots \times \Gamma$, $q_0' = [q_0, B, \cdots, B]$.

8.2.2 多道图灵机

设计多道图灵机:

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

其中 $\Gamma' = \Gamma \times \Gamma \times \cdots \times \Gamma$.

8.2.3 多带图灵机

多带图灵机由有限控制器, k 个带头和 k 条带组成. 在一个动作中, 根据有限控制器的状态和每个带头扫视的符号, 机器能够:

(1) 改变状态;

(2) 在带头所在单元, 打印一个符号;

(3) 独立的向左或向右移动每个单元, 或保持不动.

开始时, 输入在第 1 条带上, 其他都是空的. 形式定义非常繁琐, 因此省略.

定理 38. *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. 用 $2k$ 道的单带图灵机 N 模拟 k 带图灵机 M, N 用两道模拟 M 一带, 其中一道放置内容, 另一道空白, 但在被模拟带头的对应位置放标记. 为模拟 M 的一个动作, N 需要从左至右, 再从右至左扫描一次. 扫描时, 计算右侧的带头数, 每经过一个带头, 保存带头处的符号; 当收集完全部带头内容, 再从右至左更新有动作的带头符号和位置. □

8.2.4 非确定图灵机

非确定型图灵机 (NTM) 的对每个状态 q 和每个带符号 X 的转移可以有有限个选择, 即

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \cdots, (q_k, Y_k, D_k)\}.$$

NTM 每步能选择任何一个三元组. 图灵机增加非确定性并未使这个装置接受新的语言.

定理 39. *如果 L 被一个非确定图灵机接受, 那么 L 被某个确定的图灵机接受.*

证明方法. 同样利用多带技术, 用确定的 TM M 模拟非确定的 TM N. M 用第 1 条带存储 N 未处理的 ID, M 若处于某个 N 的当前 ID, 则将其复制到第 2 条带, 用来模拟. 如果不接受, 则把当前 ID 可能的 k 种下一个 ID 复制到第 1 条带的最末端, 然后继续模拟下一个 ID. □

8.2.5 多维图灵机

这种装置具有通常的有穷控制器, 但带由 k 维单元阵列组成, 在所有 $2k$ 个方向上都是无限的. 根据状态和所扫描的符号, 改变状态, 并沿着 k 轴中的一轴左正向和负向移动. 开始时输入沿着某一个轴排列, 带头在输入左端. 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.

8.3 受限的图灵机

8.3.1 半无穷带

定理 40. *图灵机的输入带, 若只有一侧是无穷的, 其能力与图灵机等价.*

证明方法. 让一侧无穷的图灵机带使用多道技术, 模拟双侧无穷的图灵机带. □

8.3.2 多栈机器

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 41. *如果图灵机接受 L, 那么双栈机接受 L.*

证明方法. 用两个堆栈模拟图灵机带, 一个堆栈保存带头左边内容, 一个堆栈保存带头右边内容, 但都不包括无穷的空格符号. 带头的移动用两个栈分别弹栈和压栈模拟, 带头修改字符 A 为 B, 用一个栈弹出的字符 A 而另一个栈压入 B 来模拟. 开始时输入在双栈机的输入带, 可以先将输入扫描并压入一个栈, 然后再依次把每个符号弹出再压入另一个栈, 然后进入模拟图灵机的状态. □

示例

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 问题

非形式的, 我们使用问题来表示诸如“一个给定的 CFG 是否是歧义? ”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.

可判定问题和不可判定问题

一个问题, 如果它的语言是递归的, 就称为**可判定** (*decidable*) 的问题, 否则, 该问题是**不可判定的** (*undecidable*). 也就是说, 对于不可判定的问题, 不存在能够保证停机的图灵机, 识别该问题的语言, 或者说不存在解决该问题的算法. 下面就给出两个不可判定的问题.

9.2 非递归可枚举的语言

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 w_i

将全部 $(0+1)^*$ 中的串按长度和字典序排序, 那么第 i 个串就是 w_i, 即

$$\text{binary}(i) = 1w_i$$

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

将字母表为 $\{0,1\}$ 的任意 TM 用二进制串进行编码. 设 TM M 为

$$M = (Q,\ \{0,1\},\ \Gamma,\ \delta,\ q_1,\ B,\ F)$$

再为状态, 栈符号和移动方向指派整数编码:

(1) $Q = \{q_1, q_2, \cdots, q_{|Q|}\}$, 指派开始状态为 q_1, 终态为 q_2, 且终态 (一定停机) 只需一个;

(2) $\Gamma = \{X_1, X_2, \cdots, X_{|\Gamma|}\}$, 这里总有 $X_1 = 0$, $X_2 = 1$, $X_3 = B$;

(3) 方向 L 为 D_1, 方向 R 为 D_2.

那么任意的转移函数

$$\delta(q_i, X_j) = (q_k, X_l, D_m)$$

可用一条编码 (C) 表示:

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

而 M 全部的 n 个转移动作的编码合在一起, 就可以作为整个 TM 的编码:

$$C_1\ 11\ C_2\ 11\ \cdots\ 11\ C_{n-1}\ 11\ C_n.$$

第 i 个图灵机

那么如果 TM M 编码为第 i 个串 w_i, 则称 M 是“第 i 个图灵机”, 记为 M_i. 而任意的串 w_i 也都可以看作 TM 编码, 如果不合法, 则可以认为是没有任何动作的 TM, 只有一个状态, 在任何输入上立即停机并拒绝输入, 其接受的语言是 $\emptyset$.

有序对 (M, w)

一个 TM 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.

9.2.3 对角化语言 L_d

定义对角化语言 L_d:

$$L_d = \{w_i \mid w_i \notin \mathbf{L}(M_i), i \geq 1\}$$

即, 使第 i 个串 w_i 不属于第 i 个图灵机的语言 $\mathbf{L}(M_i)$ 的所有 w_i 的集合.

L_d 可以由下图的矩阵给出. 矩阵的上边顺序排列每个 w_j, 矩阵的左边顺序排列每个 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 M_i 是否接受 w_i. 那么只需将对角线的值取补 (*complementary*), 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不同, 因此也不可能是任何图灵机 (的语言) 的特征向量.

M_i \ w_j	1	2	3	4	5	6	⋯
1	0	0	1	1	0	1	⋯
2	1	0	0	1	0	0	⋯
3	0	1	1	0	0	1	⋯
4	0	0	1	1	1	1	⋯
5	1	1	0	0	0	1	⋯
6	0	1	0	1	1	1	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

9.2.4 L_d 不是递归可枚举的

定理 42. *L_d 不是递归可枚举语言, 即不存在 TM 接受 L_d.*

证明. 假设存在 TM M 使 $\mathbf{L}(M) = L_d$, 由于 L_d 是 $\{0,1\}$ 上的语言, 因此 M 可以被编码成二进制串, 不妨设为 w_i, 则 $M = M_i$. 那么 w_i 是否在 L_d 中呢?

(1) 若 $w_i \in L_d$, 根据假设, 则 $w_i \in \mathbf{L}(M_i)$; 而如果 $w_i \in \mathbf{L}(M_i)$, 根据 L_d 定义，则 $w_i \notin L_d$;

(2) 若 $w_i \notin L_d$, 根据假设, 则 $w_i \notin \mathbf{L}(M_i)$; 而如果 $w_i \notin \mathbf{L}(M_i)$, 根据 L_d 定义，则 $w_i \in L_d$.

因此假设不成立, 不存在 TM 能够接受 L_d. □

因此图灵机所能接受的语言也不是任意的, 至少有一个语言 L_d 是无法被图灵机接受的.

9.3 递归可枚举但非递归的语言

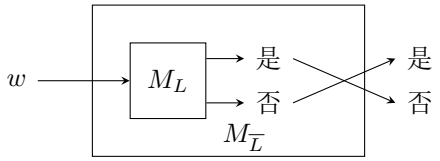
语言 L_d 是不能被图灵机接受的, 那么肯定不存在算法解决语言为 L_d 的问题, 显然这样的问题都是不可判定的. 但是即使存在图灵机, 却无法保证停机, 对于问题的解决也没有实质的贡献, 因此将“问题”区分为可判定的和不可判定的, 要比区分问题是否具有 TM 更有意义. 所以这里给出一个语言的实例 L_u, 属于递归可枚举语言但不属于递归语言.

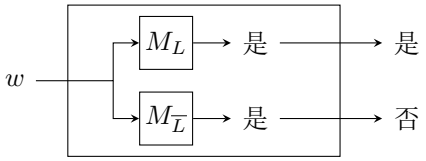
9.3.1 递归语言的封闭性

这里我们只给出递归语言封闭性的两个定理.

定理 43. *如果 L 是递归的, 那么 $\overline{L}$ 也是递归的.*

定理 44. *如果语言 L 和它的补 $\overline{L}$ 都是递归可枚举的, 那么 L 是递归的.*





9.3.2 通用图灵机

如果 TM M 接受串 w, 那么由有序对 (M, w) 构成的语言, 称为通用语言 (*universal language*), 记为 L_u.

定理 45. *L_u 是递归可枚举的, 但不是递归的.*

证明. 识别 L_u 的图灵机 U 可以这样构造, 利用多带技术让 U 模拟输入 (M, w) 中 M 识别 w 的动作, U 使用 3 条带, 第 1 带放置 (M, w), 即存储 M 动作的定义, 第 2 带模拟 M 的带, 第 3 带存储 M 的状态. 因此 L_u 是递归可枚举的.

利用反证法证明 L_u 不是递归的. 假设 L_u 是递归的, 则存在识别 L_u 的算法 A, 那么可以这样得到识别 L_d 的算法 B: 将输入 $w = w_i$ 转换为 (M_i, w_i), 并交给 A 判断; 当 A 接受 (M_i, w_i), 表示 $w_i \in \mathbf{L}(M_i)$, 则 B 拒绝; 当 A 拒绝 (M_i, w_i), 表示 $w_i \notin \mathbf{L}(M_i)$, 则 B 接受. 而由于 L_d 不是递归的, 所以这样的算法 B 不可能存在, 所以算法 A 并不存在, 所以 L_u 不可能是递归的. □

因为识别 L_u 的图灵机 U, 可以模拟任意图灵机, 因此也称为通用图灵机 (*univerasl Turing machine*). 正是因为通用图灵机的概念, 帮助冯 • 诺伊曼产生了通用电子计算机体系的设计思想, 这也可以看出抽象理论的先期发展可以对实际问题有很大帮助.

9.4 语言间的关系