Chapter 8

图灵机

8.1 图灵机

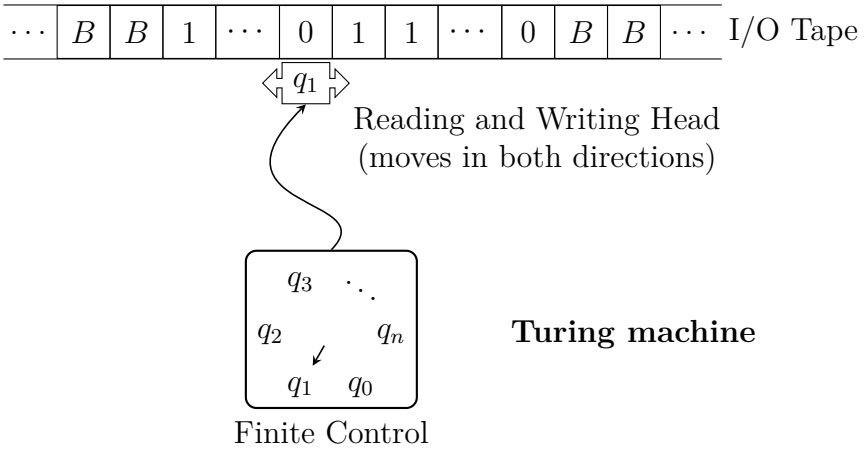
从大约公元前 3 世纪起, 人们就有了算法 (*algorithm*) 的直觉概念, 并寻找解决各种数学问题的算法, 最著名的就是欧几里德算法 (辗转相除法). 20 世纪初, 希尔伯特 (D. Hilbert) 打算寻求一个机械的有效过程, 用以确定任何数学命题的真和假. 特别的, 寻找这样的过程, 用以确定整数上的一阶谓词演算中任意公式是否为真. 希尔伯特所要寻找的有效过程就是算法, 但当时算法的概念还没有被形式化. 1931 年, 哥德尔 (K.Gödel) 发表了著名的不完全性定理, 构造了一个公式, 该公式在这个逻辑系统中既不能被证明也不能被证否, 所以不可能存在通用的过程, 证明任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

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.

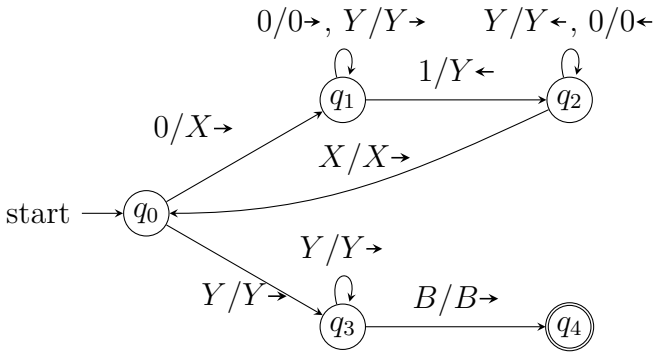
$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

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.

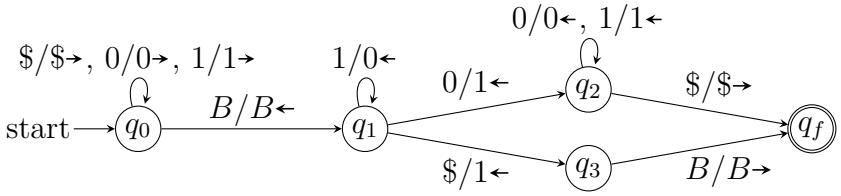
8.1.4 整数计算器

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, 符号 \$ 作为二进制数前的特殊标记. 例如 $q_0\$10011 \vdash^* \$q_f 10100$, $q_0\$111 \vdash^* q_f 1000$.

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



m+n

8.2 扩展的图灵机

Maybe 可以使用 递归

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 1. *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 2. *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 3. *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 4. *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 $L = \{a^n b^n c^n d^n e^n \mid n \geq 0\}$.