Chapter 8

图灵机

本章, 我们将介绍图灵机 — 计算机的一种简单数学模型. 尽管图灵机简单, 但它具有模拟通用计算机的能力. 人们研究图灵机不仅是为了研究它所定义的语言类 (递归可枚举语言), 也是为了研究它所计算的整数函数类 (部分递归函数).

8.1 不可判定的问题

典型问题

给定语言 $L \subseteq \Sigma^*$ 和字符串 $w \in \Sigma^*$, 判断是否 $w \in L$ 的问题, 称为语言 L 上的一个判定性问题 (decision problem).

(非形式) 定义

如果一个问题, 不存在能解决它的程序, 则称为不可判定问题.

是否存在不可判定的问题?

1. $\{L \mid L \subseteq \Sigma^*\}$ 是不可数的;
2. $\{P \mid P$ 是一个程序$\}$ 是可数的;
3. 问题显然比程序多, 必然存在不可判定问题.

hello-world 问题

判断任意给定的程序 P 在任意给定的输入 x 时,

是否会以 `hello, world` 为其输出的前 12 个字符.

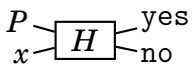
定理

hello-world 问题是不可判定的.

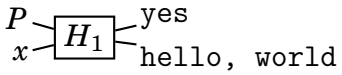
- “具有这个输入的这个程序是否显示 `hello, world`?”
- 解决这样问题的通用程序是不存在的.

(非形式) 证明: 反证法. 假设这样的程序 H 存在, 它可以在给定程序和输入时, 检查程序的输出是否以 `hello, world` 开始, 并正确的回答.

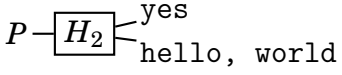
1. H 检查程序 P 在输入 x 时的输出, 并回答 `yes` 或 `no`:



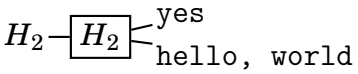
2. 修改 H, 在回答 `no` 时, 输出 `hello, world`:



3. 修改 H_1, 将程序 P 作为 P 自己的输入:



4. 那么, 当程序 H_2 以 H_2 为输入时:



5. H_2 的输出会出现矛盾 (悖论), 所以 H_2 不可能存在, 而从 H 到 H_2 的修改是合理且能行的, 所以 H 不可能存在. □

问题的归约

如何证明问题是不可判定的?

1. 归谬法 (反证法)
2. 问题的归约

calls-foo 问题

程序 Q 在输入 y 时, 是否会调用函数 `foo`?

例 1. 利用归约证明 calls-foo 问题是不可判定的.

证明: 将 hello-world 问题归约到 calls-foo 问题.

$$P \text{ 输入 } x \text{ 时会输出 } \texttt{hello, world.} \iff Q \text{ 输入 } y \text{ 时会调用 } \texttt{foo.}$$

1. P_1: 如果 P 中有 `foo` 函数, 将其重命名 (重构);
2. P_2: 给 P_1 增加函数 `foo`;

3. P_3: 保存 P_2 输出的前 12 个字符;

4. P_4: 当 P_3 输出为 `hello, world` 时, 调用 `foo`.

5. 令 $Q = P_4$, $y = x$. □

例 2. [Exercise 8.1.1a] 判断程序 R 在给定输入 z 时是否会运行结束 (halt)?

证明: 将 hello-world 问题归约到该问题.

$$P \text{ 输入 } x \text{ 时会输出 } \texttt{hello, world}. \iff R \text{ 输入 } z \text{ 时会运行结束}.$$

1. P_1: 在 P 主函数结束前增加死循环, 如 `while(1);`;

2. P_2: 保存 P_1 输出的前 12 个字符;

3. P_3: 当 P_2 输出为 `hello, world` 时, 结束程序;

4. 令 $R = P_3$, $z = x$. □

计算模型

研究计算或可计算性, 需要 "计算" 的模型和形式定义:

- *Simple*: 描述/理解/使用. 易于形式化推理, 与直觉相符.
- *Powerful*: 计算能力/表示能力. 可表示任意算法.

8.2 图灵机

在我们的课程中, 算法或 "机械而有效的计算过程" 的直觉概念已经出现过多次, 比如我们给出了一个有效过程判定有穷自动机接受的集合是否是空的, 有穷的或无穷的. 人们也会朴素的设想, 对于具有有穷描述的任何语言类, 总会存在一个有效过程来回答这类问题. 然而, 情况并非如此. 例如, 不存在算法判断一个 CFL 的补是否为空, 尽管我们可以判断这个 CFL 本身是否为空. 需要注意的是, 我们不是要求一个过程对某个具体的上下文无关语言回答这个问题, 而是要求单独一个过程, 对所有的 CFL 正确地回答这个问题.

20 世纪初, 数学家希尔伯特曾经试图寻找一个过程, 用来确定整数上一阶谓词演算中, 一个任意任何公式是否为真. 因为一阶谓词演算足以表达命题: 一个上下文无关语言产生的语言是 Σ^*. 如果这样的过程存在, 则判定一个 CFL 的补是否为空将被解决. 但是 1931 年, 哥德尔发表了著名的不完全性定理, 证明了这样的有效过程不可能存在. 在应用于整数的谓词演算

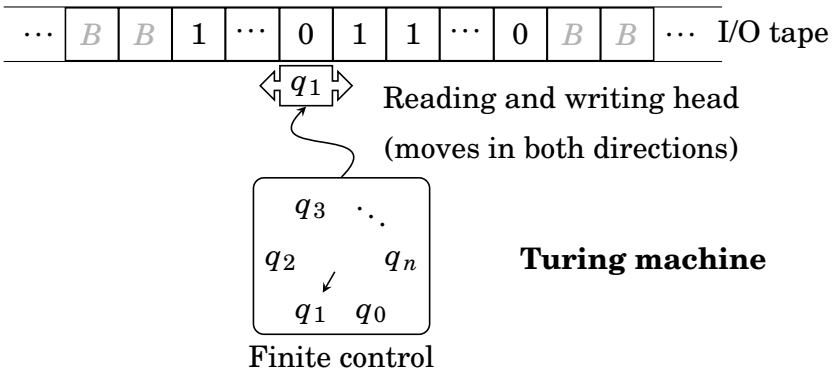
中, 他构造了一个公式, 该公式的定义表明, 其本身在这个逻辑系统中既不能被证明也不能被证否. 这个讨论的形式化, 以及后来关于有效果过程直觉概念的澄清和形式化, 是 20 世纪伟大的智力成就之一.

一旦有效过程概念被形式化, 就可以证明, 对于许多具体函数的计算没有有效过程. 今天, 图灵机已成为被人们接受的有效过程的形式定义. 我们虽然无法证明图灵机模型等价于我们关于计算的直觉概念, 但却有关于这个等价性的令人信服的证据, 也就是为人熟知的 Church 假设. 特别的, 就像我们已知的, 图灵机在计算能力上等价于数字计算机, 也等价于关于计算的所有最一般的数学概念.

有效过程的形式模型应该具有某些性质. 首先, 每个过程都应该是有穷可描述的; 其次, 过程应该由离散的步组成, 每一步能够机械的被之行. 图灵在 1936 年介绍了一个这样的模型, 被后人称为图灵机 (Turing Machine).

8.2.1 形式定义

图灵机的基本模型具有一个有穷控制器, 一条两端无穷的输入输出带和一个带头. 带划分为许多单元格, 带头每次扫视带上的一个单元格, 每个单元格可以放置有穷带符号集中的一个. 开始时, 带上的连续 n 个单元格放着输入, 它是一个字符串, 符号选自带符号的一个子集, 即输入符号集. 余下的无穷多个单元格都放着空白符, 它是一个不属于输入符号集的特殊带符号.



在一个动作中, 图灵机根据带头处单元格的符号和有限控制器的状态, (1) 改变状态, (2) 在单元格中写下一个符号, 以代替原来的符号, (3) 向左或向右移动一个单元格.

图灵机的形式定义

定义. 图灵机 *(TM, Turing Machine) M* 为七元组

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

1. Q: 有穷状态集;

2. Σ: 有穷输入符号集;

3. Γ: 有穷带符号集, 且总有 $\Sigma \subset \Gamma$;

4. δ: $Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$ 转移函数, 而 δ 可以对某些自变量没有定义;

5. $q_0 \in Q$: 初始状态;

6. $B \in \Gamma - \Sigma$: 空格符号或空白符;

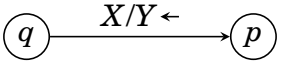
7. $F \subseteq Q$: 终态集或接受状态集.

图灵机的动作及状态转移图

有穷控制器处于状态 q, 带头所在单元格为符号 X, 如果动作的定义为

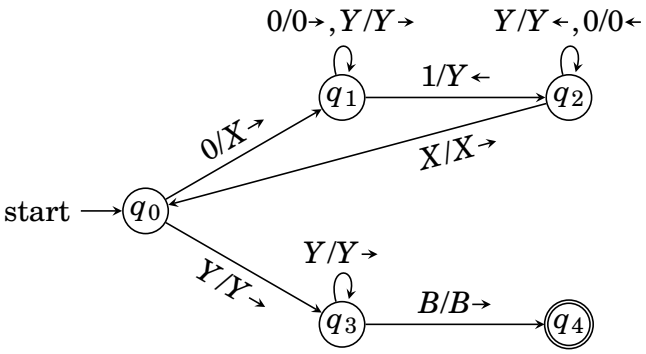
$$\delta(q,X) = (p,Y,L),$$

表示状态转移到 p, 单元格改为 Y, 然后带头向左移动一个单元格.



因为每个动作都是确定的, 因此是“确定的图灵机”.

例 3. 设计识别 $\{0^n 1^n \mid n \geq 1\}$ 的图灵机.



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

δ	0	1	X	Y	B
q_0	(q_1,X,R)	–	–	(q_3,Y,R)	–
q_1	$(q_1,0,R)$	(q_2,Y,L)	–	(q_1,Y,R)	–
q_2	$(q_2,0,L)$	–	(q_0,X,R)	(q_2,Y,L)	–
q_3	–	–	–	(q_3,Y,R)	(q_4,B,R)
q_4	–	–	–	–	–

8.2.2 瞬时描述及其转移

瞬时描述

定义. 图灵机虽有无穷长的带, 但是在有限步移动之后, 带上的非空内容总是有限的. 因此用带上最左边到最右边全部的非空符号、当前状态和带头位置, 同时定义瞬时描述 *(ID, Instantaneous Description)* 或格局 *(Configuration)* 为

$$X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n$$

其中的信息包括:

1. 图灵机的当前状态 q;
2. 带头在左起第 i 个非空格符上;
3. $X_1X_2\cdots X_n$ 是输入带上从最左到最右非空格内容, 虽然中间可能有空格符.
4. 一般假定 Q 和 Γ 不相交以避免混淆.

转移符号

定义. 图灵机 M 中, 如果 $\delta(q,X_i)=(p,Y,L)$, 定义 *ID* 转移为

$$X_1\cdots X_{i-1}qX_i\cdots X_n \vdash_M X_1\cdots X_{i-2}pX_{i-1}YX_{i+1}\cdots X_n$$

如果 $\delta(q,X_i)=(p,Y,R)$ 那么

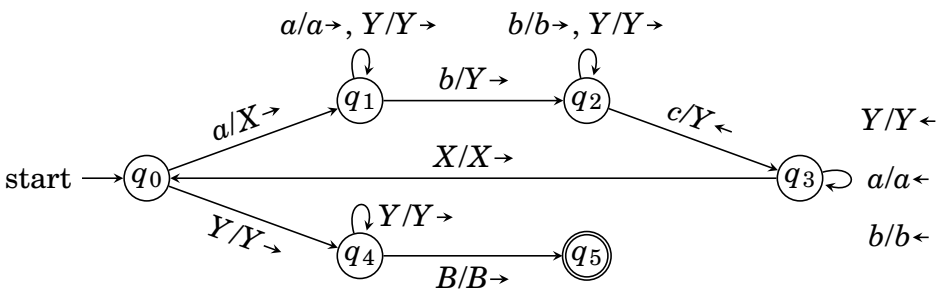
$$X_1\cdots X_{i-1}qX_i\cdots X_n \vdash_M X_1\cdots X_{i-1}YpX_{i+1}\cdots X_n$$

若某 *ID* 是从另一个经有限步 *(包括零步)* 转移而得到的, 记为 $\overset{*}{\vdash}_M$. 若 M 已知, 简记为 $\vdash$ 和 $\overset{*}{\vdash}$.

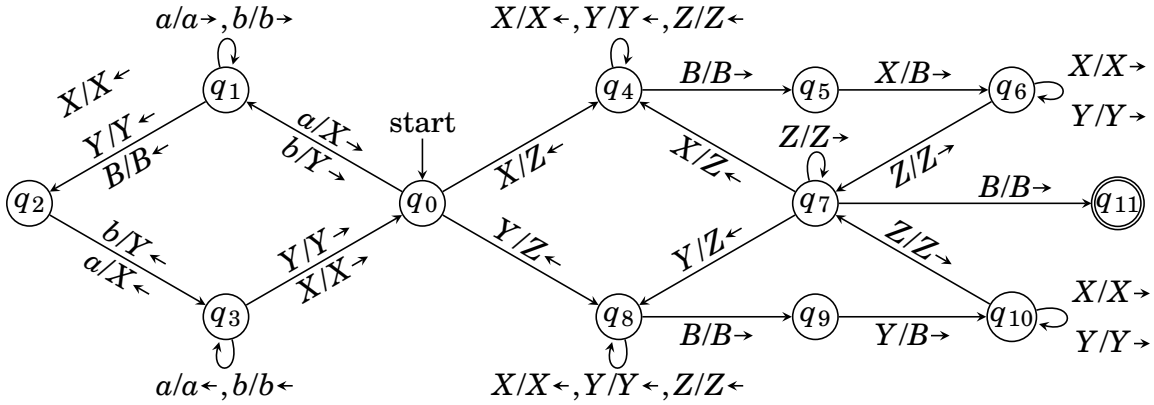
接受 0011 的 ID 序列 (M 的一个计算) 为

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

例 4. 设计接受 $L=\{a^nb^nc^n \mid n\geq 1\}$ 的图灵机.



例 5. 设计接受 $L = \{ww \mid w \in \{a,b\}^+\}$ 的图灵机.



思考

DFA 和 TM 的主要区别? — 能够“写”是多么重要

8.2.3 语言与停机

图灵机的语言

定义. 如果 $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ 是一个图灵机, 则 M 接受的语言为

$$\mathbf{L}(M) = \{w \mid w \in \Sigma^*,\ q_0 w \overset{*}{\vdash} \alpha p \beta,\ p \in F,\ \alpha, \beta \in \Gamma^*\}.$$

输入串 w 放在输入带上, M 处于 q_0, 带头位于输入串的第一个字符上, 输入串最终会导致 M 进入某个终结状态.

定义. 如果 L 是图灵机 M 的语言, 即 $L = \mathbf{L}(M)$, 则称 L 是递归可枚举语言.

一般假定当输入串 w 被接受时图灵机 M 总会停机 (*halt*), 即没有下一个动作 (的定义). 而对于不接受的输入, 图灵机可能永不停止. 能够被图灵机接受的语言类, 称为递归可枚举的 (*recursively enumerable*, RE). 术语“可枚举”是指, 这些语言中的串可以被某个图灵机枚举出来. “递归”是一个在计算机出现之前的数学名词, 其意义与计算机科学家所说的“递归式”相近. 递归可枚举语言类非常广, 它真包含上下文无关语言. 有些包含在这个语言类中的语言, 无法机械的确定其成员资格. 若 $\mathbf{L}(M)$ 是这样的语言, 识别 $\mathbf{L}(M)$ 的任何图灵机, 在不属于 $\mathbf{L}(M)$ 的某些输入上 M 停不下来. 若 $w \in \mathbf{L}(M)$, 那么在 w 上, M 最终是会停下来的. 然而, 只要 M 还在某个输入上运行, 我们就永远也不会知道, 到底是因为运行的时间不够长而没有接受呢, 还是根本就不会停机.

定义. 对接受和不接受的输入, 都保证停机的图灵机, 所接受的语言称为递归语言.

从递归可枚举语言类中, 分出一个由保证停机的图灵机所识别的子类是比较方便的. 递归语言类是这样的一个子类, 它们至少被一个在所有输入上都能停机的图灵机接受. 在下一章中

我们会看到, 递归语言类是递归可枚举语言类的真子集. 那么, 由 CYK 算法, 每个 CFL 都是一个递归语言.

算法的形式化

保证停机的图灵机, 正是算法的好模型, 即算法概念的形式化.

- λ-caculus — Alonzo Church, Stephen Kleene
- Partial recursive functions — Kurt Gödel
- Post machines — Emil Post
- Turing machines — Alan Turing

8.2.4 整数函数计算器

图灵机可以作为语言的识别器或枚举器, 也可以用作整数到整数的函数计算器.

- 传统的方法, 把整数 $i \geq 0$ 写为 1 进制, 用字符串 0^i 表示;
- 若计算 k 个自变量 $i_1, i_2, \ldots, i_k$ 的函数 f, 用

$$0^{i_1}10^{i_2}1\cdots10^{i_k}$$

作为 TM M 的输入;

- M 停机, 且输入带上为 0^m, 表示 $f(i_1, i_2, \ldots, i_k) = m$.
- M 计算的 f, 不必对所有不同的 $i_1, i_2, \ldots, i_k$ 都有结果.

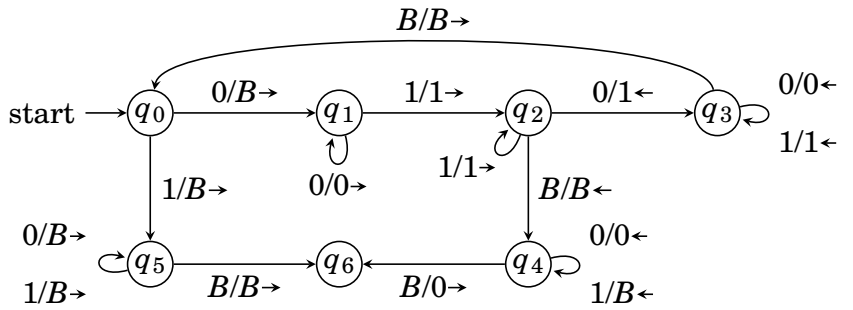
定义. *如果 $f(i_1, i_2, \ldots, i_k)$ 对所有不同的 $i_1, i_2, \ldots, i_k$ 都有定义, 称 f 为全递归函数.*

被图灵机计算的函数 $f(i_1, i_2, \ldots, i_k)$ 称作部分递归函数.

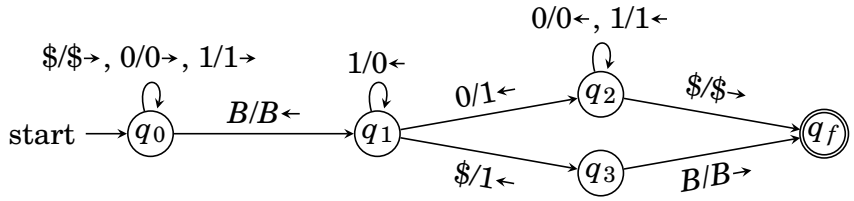
部分递归函数对应于递归可枚举语言, 它被一个在给定输入上可停可不停的图灵机计算. 全递归函数对应于递归语言, 它被总是能停机图灵机计算. 所有常用的整数函数都是全递归函数, 如乘积, $n!$, $\lceil \log_2 \rceil n$ 和 2^{2^n} 等.

例 6. 给出计算整数真减法 ($\dot{-}$) 的图灵机, 其定义为

$$m \dot{-} n = \begin{cases} m - n & m \geq n \\ 0 & m < n \end{cases}.$$



例 7. 二进制数的加 1 函数, 使用符号 \$ 作为数字前的占位标记. 例如 $q_0\$10011 \vdash \$q_f10100$, $q_0\$111 \vdash^* q_f1000$.



8.3 图灵机的变形

以给出完整的状态集和动作函数的方法详细的设计一个图灵机是一项相当费力的任务. 为了描述复杂的图灵机结构, 我们需要 “高级” 的概念性工具和技术.

状态中存储

有限控制器中可以存储有限个符号的图灵机:

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

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

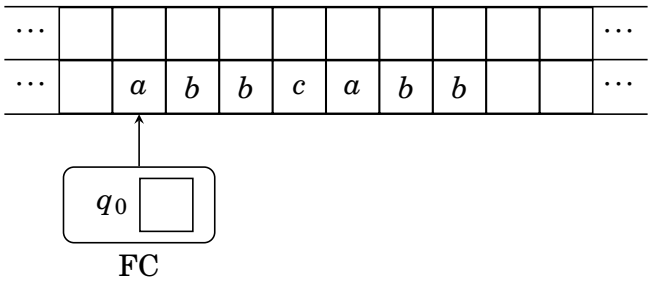
多道

多道图灵机:

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

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

例 8. 利用状态中存储与多道设计 TM 识别 $L = \{ wcw \mid w \in \{a,b\}^* \}$.



子程序

设计 TM 的一部分作为一个子程序:

- 具有一个指定的初始状态;
- 具有一个指定的返回状态, 但暂时没有定义动作;
- 可以具有参数和返回值.

通过进入子程序的初始状态, 实现调用; 通过返回状态的动作, 实现返回.

例 9. 设计 TM 实现全递归函数 “乘法”.

8.3.1 扩展的图灵机

多带图灵机

有穷控制器、k 个带头和 k 条带组成. 每个动作, 根据状态和每个带头符号:

1. 改变控制器中的状态;
2. 修改带头单元格中的符号;
3. 每个带头独立的向左或右移动一个单元格, 或保持不动.

开始时, 输入在第 1 条带上, 其他都是空的, 其形式定义非常繁琐.

定理 40. *如果语言 L 被一个多带图灵机接受, 那么 L 能够被某个单带图灵机接受.*

证明方法:

1. 用 $2k$ 道的单带图灵机 N 模拟 k 带图灵机 M;
2. N 用两道模拟 M 一带, 一道放置内容, 另一道标记带头;
3. 模拟 M 的一个动作, N 需要从左至右, 再从右至左扫描一次;
4. 第一次扫描搜集当前格局, 第二次扫描更新带头和位置.

图灵机的运行时间

定义. 图灵机 M 在输入 w 上的运行时间是 M 在停机之前移动的步数. M 的时间复杂度是在所有长度为 n 的输入上, 运行时间最大值的函数 $T(n)$.

- 如果 M 在某些 w 上不停机, 则 $T(n)$ 是无穷的;
- 所以, 保证停机的图灵机, 其 $T(n)$ 才有意义;
- 但是, 只有多项式时间的 $T(n)$, 才是问题实际可解的边界.
- 然而, 对很多问题, 还没有比精确到多项式时间更好的结果.

定理 41. 单带图灵机 N 模拟 k 带图灵机 M 的 n 步移动, 需要使用 $O(n^2)$ 的时间.

证明:

1. M 的 n 步移动, 带头相距不会超过 $2n$;
2. 而标记带头并调转方向至多需要 $2k$ 步;
3. 因此 N 模拟 M 的 1 步至多需要 $4n+2k$ 步, 即 $O(n)$ 时间;
4. 因此模拟 n 步需要 $O(n^2)$ 时间. □

实际可行性

使用单带 TM 或多带 TM, 不会改变问题是否实际可解.

非确定图灵机 (NTM)

图灵机在每组状态 q 和带符号 X 的转移 $\delta(q,X)$, 可以有有限个选择:

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

- NTM 接受语言的方式, 与 NFA 和 PDA 是类似的
- 存在从初始 ID 到某接受状态 ID 的转移, 其他选择可以忽略

定理 42. 如果 L 被非确定图灵机接受, 那么 L 被图灵机接受.

证明:

1. TM M 用控制器保存并用两条带模拟 NTM N 的动作: 第 1 条带存储 N 未处理的 ID, 第 2 条带模拟 N 的带.

2. M 将第 1 条带最前端的 ID 复制到第 2 带, 若接受则停止;
3. 把当前 ID 可能的 k 个转移 ID 复制到第 1 条带的最末端;
4. 将第 1 带上最前端的 ID 抹掉, 从第 2 步重复.

- 只有 N 进入接受的 ID 时, M 才会且一定会接受;
- 因为若 N 每步最多 m 个选择, 那么从初始 ID 经过 n 步最多可到

$$1+m+m^2+\cdots+m^n$$

个 ID, 而 M 会以 "先广 (*breadth first*)" 顺序检查这些最多为 nm^n 个的 ID. □

TM 模拟 *NTM* 的时间

- NTM N 的 n 步计算, TM M 需要指数时间 $O(m^n)$ 才能完成.
- 但这里 "指数时间的增长" 是否必然, 仍是未知的.
- NTM 以多项式时间解决的问题, TM 是否也能以多项式时间解决呢?

$$\mathrm{P} \stackrel{?}{=} \mathrm{NP}$$

思考题

为什么非确定性没有改变图灵机识别语言的能力?

多维图灵机

1. 具有通常的有穷控制器和一个带头;
2. 由 k 维阵列组成的带, 在 $2k$ 个方向上都是无限的;
3. 根据状态和读入符号改变状态, 并沿着 k 个轴的正和负向移动;
4. 开始时, 输入沿着某一个轴排列, 带头在输入的左端.

同样, 这样的扩展也没有增加额外的能力, 仍然等价于基本的图灵机.

8.3.2 受限的图灵机

半无穷带图灵机

图灵机的输入输出带只有一侧是无穷的.

定理 43. *半无穷带图灵机, 与图灵机等价.*

证明方法: 一侧无穷的带上使用多道技术, 模拟双侧无穷的带.

多栈机

基于下推自动机的扩展, k 栈机器是具有 k 个栈的确定型下推自动机.

定理 44. *如果图灵机接受 L, 那么双栈机接受 L.*

证明方法:

1. 一个堆栈保存带头左边内容, 一个堆栈保存带头右边内容;
2. 带头的移动用两个栈分别弹栈和压栈模拟;
3. 带头修改字符 A 为 B, 用一个栈弹出 A 而另一个压入 B 来模拟;
4. 开始时输入在双栈机的输入带, 但先将输入扫描并压入一个栈, 再依次弹出并压入另一个栈, 然后开始模拟图灵机.

例 10. 设计双栈机接受 $L = \left\{ a^n b^n c^n \,\middle|\, n \geq 0 \right\}$.

8.3.3 图灵机的抽象描述

- 形式化表示 – 底层、准确
  - 精确的数学语言
  - 相当于实现算法的程序代码
- 实现细节说明 – 明确、规范
  - 准确的读写头动作和带内容管理
  - 相当于描述算法的伪代码
- 抽象叙述 – 精简、高效
  - 说明性的日常语言

- 相当于概括算法的思想

chunyu@hit.edu.cn
http://nclab.net/~chunyu