Turing Machine

Turing Machine

Definition

TM 的构成包括一个状态寄存器（存储一个有限集中的状态），一个向左右无限延伸的 tape，tape 被分成很多个 cell，每个 cell 中可以填入一个 symbol，以及一个读写头

一个有限长的 string 被写在 tape 上，作为 input，而其余部分被 *blank* 填满，注意 blank 属于 tape alphabet，而不是 input alphabet

读写头总是指向某个 tape cell，默认开始时指向 input 的最左端。TM 的 move 由当前的状态和读写头指向的 cell 中的 symbol 决定，包含三个动作

- 修改状态
- 在当前 cell 写入新的符号
- 将读写头向左/右移动

更为形式化的叙述中，TM 由一个 7-tuple 定义

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

- Q : the finite set of state
- Σ : input alphabet
- Γ : tape alphabet, $\Sigma\subset\Gamma$
- δ : transition function
- q_0 : start state
- B : blank symbol, $B\in\Gamma-\Sigma$
- F : the finite set of accepting state, $F\in Q$

其中 δ 的定义为，δ 接受两个参数，当前状态 q 与当前 tape symbol X，而其输入为一个 3-tuple (p,Y,D)，其中

- $p\in Q$ 是下一个状态
- $Y\in\Gamma$ 是被写入当前 cell 的 symbol
- D 是读写头移动的方向，可以是 L,R （表示左右）或是 N （表示不移动）

Instantaneous Descriptions for TM

正如 PDA 的 ID 一样，我们同样需要一个 ID 来描述运行中某一时刻时 TM 的全部信息

虽然 TM 有一个无限长的 tape，但是在有限步的 move 中，TM 所能访问到的 cell 是有限的。而在无限的未被访问到的 prefix/suffix 中，只可能是 blank 或是有限个输入符号，因此只需要在 ID 中显示在最左的非 blank 符号和最右的非 blank 符号之间的部分

TM 的 ID 是一个 string 形如

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

其中

- q 是当前的状态
- 读写头指向第 i 个 symbol
- $X_1X_2\dots X_n$ 是最左的非 blank 符号和最右的非 blank 符号之间的部分

使用同样的记号 $\vdash$ 来表示 ID 间的转换，而使用 $\vdash^*$ 表示零步或多步转换。如果 $\delta(q, X_i) = (p, Y, R)$，则

$$X_1X_2\dots X_{i-1}qX_iX_{i+1}\dots X_n \vdash X_1X_2\dots X_{i-1}YpX_{i+1}\dots X_n$$

TM 同样可以用转换图来表示，若 $\delta(q, X) = (p, Y, L)$，则从 q 到 p 有一条标号为 $X/Y, L$ 的边

Language of TM

TM 同样有两种定义语言的方式

- Acceptance by final state
- Acceptance by halting

TM 的 halt 定义为当前状态为 q，tape symbol 为 X，而 $\delta(q, X)$ 未定义

对于 TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$，若 acceptance by final state，其语言定义为

$$L(M) = \{w : q_0w \vdash^* \alpha p\beta, p \in F\}$$

而 acceptance by halting 的定义为

$$H(M) = \{w : q_0w \vdash^* \alpha pX\beta, \delta(p, X) \text{ is undefined}\}$$

同样可以证明这两种定义语言的方式是等价的，即给定一个 TM M 满足 $L = L(M)$，则存在 TM M' 满足 $L = H(M')$，反之亦然

From Final State to Halting

考虑 TM M，则可以构造 M'，只需对于任意 $q \in F, X \in \Gamma$，删去 $\delta(q, X)$，这样只要 M 运行到 accept state，M' 就 halt

为了防止运行中的意外 halt，引入新的状态 s，对任意 $X \in \Gamma$ 有 $\delta(s, X) = (s, X, R)$，即进入状态 s 后 TM 无限向右移动，对于任意 $q \in Q - F, X \in \Gamma$，若 $\delta(q, X)$ 未定义，则令 $\delta(q, X) = (s, X, R)$，即可避免非接受的情况下 halt

From Halting to Final State

考虑 TM M，则可以构造 M'，引入新的状态 f 作为唯一的 final state，且对于所有 $q \in Q, X \in \Gamma$，若 $\delta(q, X)$ 未定义，则令 $\delta(q, X) = (f, X, R)$

RE and Recursive

使用 final state 与使用 halting 定义的语言集合已经被证明是相同的，这个语言集合被称为**递归可枚举语言 (recursively enumerable language)**

算法 (Algorithm) 是一种特殊的 TM，其满足 accepting by final state，且**不论是否接受都会 halt**，若某个 TM M 是一个 algorithm，则称其定义的语言 $L = L(M)$ 是**递归语言 (recursive language)**

Programming Techniques for TM

有很多有用的技巧可以使 TM 的编程更加方便，且这些技巧并没有改变基本的 TM 模型，而只是改变了表示的方式

具体的例子见课本 8.3 节

Storage in the State

可以扩展 TM 的 state，使其成为一个 tuple（或者另一种说法，vector），从而携带更多的信息

这样的技巧并不会改变原本的 TM 模型，因为 state tuple 的每个分量都来自某个有限的 alphabet，则 state tuple 的集合，即这些 alphabet 的笛卡尔积，也是有限的，故仅仅是基本的 TM，将其状态用另一种更适合理解的方式表示

Multiple Tracks

同理，可以扩展 tape cell，使其不止存储一个 symbol，而是存储一个 symbol 的 vector，transition function 对其整体进行修改

这样的技巧不改变原本的 TM 模型，因为其 tape alphabet 也是有限个有限 alphabet 的笛卡尔积，仍是有限集合，只是改变了 tape symbol 的表示方式

有了 multiple tracks 即可实现 marking 的功能，即在原来的基础上增加一个 track，用于存储记号，标记某些有特殊意义的位置

Subroutines

TM 的 subroutine 是一个 state 的集合，用于实现某种过程，其中有一个开始状态，也有一个状态在其上没有 move，用于实现返回的功能，对其的调用从状态迁移至开始状态开始。

Extension to the Basic TM

Multitape TM

multitape 的 TM 不同于多 track 的TM，后者虽然有多个 track 但是仍然是同时读写每一个 vector 的，而 multitape TM 有多个 **tape** ，即有多个读写头可以任意移动

在状态转换时，TM 改变状态，每个 tape 上的读写头修改当前 cell 的 symbol，然后独立地向左或右移动

可以证明 multitape TM 没有增加 TM 原有的能力，即

所有被 multitape TM 接受的语言都是 RE

> Proof. 考虑一个有 k 个 tape 的 TM M ，则可以使用一个 one-tape TM N 模拟其运行，N 有 $2k$ 个 track，其中一半是 M 的 tape 内容，另一半 tracks 每一个都有一个 mark，用于指示对应的 tape 的读写头位于何处，则将 multitape TM 的动作顺序处理即可，即找到第一个 tape 的 mark，然后处理第一条 tape，再找到第二个 tape 的 mark……

可以证明模拟 n 步 k-tape TM 的时间复杂度为 $O(n^2)$

Nondeterministic TM

与 TM 不同，NTM 的 transition function 的结果是一个集合，可以从中选择一个结果用于接下来的 move，即

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2) \dots (q_n, Y_n, D_n)\}$$

只要对一个输入 w 有一个序列接受了 w 即称该输入被 NTM 接受

NTM 并没有扩展 TM 的能力，如果 M_N 是 NTM，则可以构造出一个 DTM M_D 使得

$$L(M_N) = L(M_D)$$

> Proof. M_D 是一个 multitape 的 TM，第一个 tape 用于记录 M_N 的 ID，同时使用一个 mark 来标记当前正处理到的 ID
>
> 对于正处理到的 ID，M_D 读取其状态和指向的 tape symbol，如果当前状态是 accept，则 M_D 停止并接受，否则根据 M_N 的 transition function，确认接下来的 move。若有 k 个选择，则使用第二个 tape 将当前 ID 在 tape 结尾复制 k 份，然后根据不同选择修改这些 ID。修改完成后回到原本的 ID，将 mark 移至下一个 ID
>
> 如果把从初始 ID 开始的 ID 转换看作一颗树，则 M_D 所做的就是模拟树上的 BFS
>
> 如果 M_N 在 n 步后进入接受状态，且 M_N 选择数的上界为 k，则 M_D 最多只需要构造出 $(k^{n+1}-k)/(k-1)$ 个 ID 即可达到接受状态，即只要 M_N 能接受，在有限步中 M_D 也能接受。

Restricted TM

TM with semi-infinite tape

可以将 TM 的 tape 限制为仅向右边无限延伸，即在初始位置以左没有 cell，同样的可以限制 TM 永远不会写 blank，这样 tape 满足在一串非 blank symbol 后是无限的 blank，且这个非 blank 的序列从起始位置开始

semi-tape 的实现可以是两个 track，上半部分代表开始位置以右的部分，而下半部分代表开始位置以左的部分，第 i 个位置代表 X_i 和 X_{-i}，对于开始位置 X_0，其对应的下半部分为一个特殊符号用于标记边界

X_0	X_1	X_2	$\dots$
$*$	X_{-1}	X_{-2}	$\dots$

可以证明任何被 TM M_2 接受的语言都被一个 TM M_1 接受，且满足

- M_1 的 head 从不向开始位置以左移动
- M_1 从不写 blank

> Proof. 对于第二个限制，只需增加新的 tape symbol B' 用于模仿 blank 的功能，若 M_2 有
>
> $$\delta_2(q,X)=(p,B,D)$$
>
> 则将其改为
>
> $$\delta_2(q,X)=(p,B',D)$$

> 对所有 $q \in Q$ ，令 $\delta_2(q, B') = \delta_2(q, B)$
>
> 对于第一个限制，设 $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, B, F_2)$ ，且已修改过以满足限制 2，则构造
>
> $$M_1 = (Q_1, \Sigma \times \{B\}, \Gamma_1, \delta_1, q_0, [B, B], F_1)$$
>
> 其中
>
> Q_1 ：为 $\{q_0, q_1\} \cup (Q_2 \times \{U, L\})$ ，即两个状态 q_0, q_1 ，以及一个 2-tuple 的集合，第一个分量为 M_2 的状态，第二个分量指示哪一条 track (upper/lower) 。
>
> Γ_1 ：其中元素为一个 2-tuple，两个分量均来自 Γ_2 。对于输入符号，其第一个分量为一个来自 input alphabet 的 symbol，第二个分量为 blank，即 $[a, B], a \in \Sigma$ ，对于 blank，其两个分量均为 blank。此外还有特殊符号 $[X, *], X \in \Gamma_2, * \notin \Gamma_2$ ，用于标识 tape 的最左端
>
> δ_1 ：其定义为
>
> - $\delta_1(q_0, [a, B]) = (q_1, [a, *], R), a \in \Sigma$ ，即将 lower track 的最左端修改为 $*$ ，同时右移，因为在最左端不能左移
> - $\delta_1(q_1, [X, B]) = ([q_2, U], X, L), X \in \Gamma_2$ ，即转移至 M_2 的起始状态，同时左移，恢复到输入开始的位置，且指向 upper track
> - 如果 $\delta_2(q, X) = (p, Y, D)$ ，则对于任意 $Z \in \Gamma_2$ 有
>
> $$\delta_1([q, U], [X, Z]) = ([p, U], [Y, Z], D)$$
> $$\delta_1([q, L], [Z, X]) = ([p, L], [Z, Y], \overline{D})$$
>
>   即模拟 M_2 的运行，且注意由于折叠的原因，在 lower track 时方向相反
> - 如果 $\delta_2(q, X) = (p, Y, R)$ 则
>
> $$\delta_1([q, L], [X, *]) = \delta_1([q, U], [X, *]) = ([p, U], [Y, *], R)$$
>
>   这个转换用于处理从初始位置向右走
> - 如果 $\delta_2(q, X) = (p, Y, L)$ 则
>
> $$\delta_1([q, L], [X, *]) = \delta_1([q, U], [X, *]) = ([p, L], [Y, *], R)$$
>
>   用于处理从初始位置向左走
>
> F_1 ：其定义为 $F_2 \times \{U, L\}$ ，即第一分量为 M_2 接受状态，不论其在 upper 或 lower track
>
> 显然对 TM 的 move 归纳，可以得出 $L(M_1) = L(M_2)$

Multistack Machines

如果给 PDA 增加一个 stack，则其接受任何 TM 能够接受的语言

一个 k-stack machine 是一个 DPDA，有 k 个栈。其处理输入的方式同 PDA（顺序读入）而非 TM（存储在 tape/stack）。其同样有一个有限的状态集，以及一个 stack alphabet，用于所有的 stack，其每一步动作取决于

- 当前状态
- 当前读入的输入符号，multistack machine 同样可以在 ϵ 上转换，但是因为其是 deterministic，故不能同时定义一个非 ϵ 转换和一个 ϵ 转换
- 每个 stack 的栈顶

而其动作包括

- 修改状态
- 对每个 stack，将其栈顶的 symbol 替换为一个有零个或多个 stack symbol 的 string

其 transition function 形如

$$\delta(q, a, X_1, X_2, \ldots, X_n) = (p, \gamma_1, \gamma_2, \ldots, \gamma_n)$$

为了方便，可以引入一个不在 input symbol 的符号 \$ 用于代表输入的结束，则有

如果有 L 被一个 TM 接受，则 L 被一个 2-stack machine 接受

> Proof. 两个 stack 可以用于模拟 TM 的一条 tape，即一个 stack 用于代表 head 以左的部分，一个 stack 用于代表 head 以右的部分。设 TM 为 M，2-stack machine 为 S ，则
>
> - S 起始时每个栈有一个代表栈底的符号，这个 symbol 可以是栈的开始符号，仅能在栈底出现，当一个栈中只含这个符号时称这个栈为空
> - 若输入为 $w\$$ ，则 S 将 w 复制到第一个栈中，当读到 \$ 时停止复制
> - 将第一个栈中所有符号弹出并压入第二个栈，这样第一个栈空，第二个栈顶为 w 最左端
> - S 进入模拟 M 的状态，第一个栈空表示 head 左端都是 blank，第二个栈为 w 表示 head 及其右端都是 w
> - 则 S 可以模拟 M 的动作，根据
>   - M 当前状态，即 S 的状态（S 模拟 M 的 finite control）
>   - M 的 head 指向的 symbol，即第二个栈的栈顶，若栈顶为栈底符号，则认为 M 指向 blank
>   - 根据以上信息得知 M 新的状态
>   - 如果 M 写 Y 并向右移动，则将 Y 压入第一个栈，且将第二个栈的栈顶弹出。若第二个栈空，则第二个栈不变；同样的，若 Y 为 blank 且第一个栈空，则第一个栈不变
>   - 如果 M 写 Y 并向左移动，则将第一个栈的栈顶，设为 Z，弹出，并且将第二个栈的栈顶弹出，压入 ZY 。如果第一个栈为空，则第一个

> 栈不变，弹出第二个栈的栈顶，压入 BY 。
> - 若 M 接受则 S 接受

Power of Counter Machines

counter machine 能存储有穷个整数（counter），并且根据某些 counter 是否为零来实现不同的动作。counter machine 只能对某个 counter 增 1 或减 1，且不能区分两个非零的 counter。从另一种视角来看，counter 是一个只有两种符号的 stack，其中一个 symbol 代表栈底，另一个可以被压入或弹出（计数）。对于 counter machine 有以下两个结论

- 任何能被 counter machine 接受的语言都是 RE，因为 counter machine 是一种特殊的 multistack machine，而 multistack machine 是一种特殊的 multitape TM
- 任何能被 one-counter machine 接受的语言都是 CFL，因为 counter 可以看作 stack，而 one-stack machine 就是 PDA

而只有两个 counter 的 counter machine 即可模拟 TM，这个结论分为两步

任意 RE language 都能被一个 3-counter machine 接受

> Proof. 根据上文结论，任意 RE language 都能被一个 2-stack machine 接受，则可以用 counter 模拟 stack。
>
> 考虑 stack machine 使用了 $r-1$ 个 tape symbol，则可以用数字 1 到 $r-1$ 来代表这些 symbol，这样栈中的内容就可以表示为一个 r 进制的整数，即栈中 $X_1X_2\dots X_n$ 表示为整数 $X_nr^{n-1}+X_{n-1}r^{n-2}+\cdots+X_2r+X_1$
>
> 使用两个 counter 用于保存两个 stack 中的内容，第三个 counter 用于调整两个 counter，即用于将某个 counter 乘以或除以 r
>
> 对于栈的操作，可以拆分为三种基本操作：弹出栈顶，替换栈顶，压入新符号。对于一个用整数 i 代表的栈，这三种操作为
>
> - 弹出栈顶：将 i 变为 i/r ，并且不保留余数（即结果向下取整，余数为 X_1 ），其步骤为
>   - 第三个 counter 为 0
>   - 重复将目标 counter 减去 r ，同时将第三个 counter 加 1
>   - 当目标 counter 为 0 时，停止
>   - 重复将目标 counter 加 1，同时将第三个 counter 减 1
>   - 当第三个 counter 为 0 时目标 counter 为 i/r
> - 将栈顶的 X 变为 Y ：如果 $X>Y$ 则将目标 counter 减少 $X-Y$ ，否则将目标 counter 增加 $Y-X$
> - 压入 X ：需要将目标 counter 从 i 变为 $ir+X$ ，则
>   - 第三个 counter 为 0

> - 重复将目标 counter 减 1，同时将第三个 counter 加 r
> - 当目标 counter 为 0 时，第三个 counter 为 ir
> - 将第三个 counter 复制到目标 counter ，则第三个 counter 为 0
> - 将目标 counter 增加 X
>
> 这样只需要将 counter 设置为 stack 初始的情况，即将其设为代表 stack 开始符号的整数，然后即可模拟 2-stack machine

任意 RE language 都能被一个 2-counter machine 接受

> Proof. 根据上文结论，只需要证明一个 3-counter 能被一个 2-counter 模拟即可
>
> 思路为将 3 个 counter，i, j, k 用一个整数模拟，如 $m = 2^i 3^j 5^k$ 。其中一个 counter 用于保存 m，同时另一个 counter 用于辅助乘除操作
>
> - 增加 i, j, k： 将 m 乘以 2 或 3 或 5 即可
> - 测试 i, j, k 是否为 0：判断 m 能否被 2/3/5 除尽即可，这一步可以通过有限的状态转换做到
> - 减少 i, j, k： 只需将 m 除以 2/3/5 ，因为 counter 不能减到负数，因此当不能除尽时代表原来的 3-counter 的操作使某个 counter 减到了负数，这样只需要 halt 即可

Closure Properties of Recursive and RE languages

对于 recursive language 和 RE language，以下操作均封闭

- union
- concatenation
- star
- reversal
- intersection
- inverse homomorphism

对于 recursive language，以下操作封闭

- difference
- complementation

对于 RE language，以下操作封闭

- homomorphism

Union and Intersection

Union

令 $L_1 = L(M_1), L_2 = L(M_2)$ ，且 M_1, M_2 都是 single-semi-infinite-tape TM

构造一个 2-tape 的 TM M 并且将输入复制到两个 tape，平行地模拟 M_1, M_2 即可

- 对于 recursive language，若 M_1, M_2 均是 algorithm，则 M 必定能 halt，只需当两个 TM 至少有一个接受，M 接受，若两个 TM 都拒绝，则 M 拒绝
- 对于 RE language，只需两个 TM 至少有一个接受， M 接受

Intersection

intersection 的思路与 union 完全相同，只需改变接受条件即可

Difference and Complement

思路同 union，对于 recursive language，只需 halt 后判断是否 M_1 接受且 M_2 拒绝即可，但是对于 RE，由于其 TM 不保证一定能在不接受时给出结果，当 M_2 始终不 halt 时并不能判断输入是否属于 $L_1 - L_2$

complement 是全集的 difference

Concatenation

令 $L_1 = L(M_1), L_2 = L(M_2)$ ，且 M_1, M_2 都是 single-semi-infinite-tape TM

构造一个 2-tape 的 NTM M，根据猜测将输入 w 分割为 xy ，将 y 复制到第二个 tape 并开始平行地模拟 M_1, M_2 ，当两个模拟均接受时 M 接受

由于 M 是 NTM，根据定义其只要有一条路径能接受即可接受，而若使用 DTM 并顺序模拟拆分过程可能导致在某个拆分结果上永远运行（对 RE 而言，若是 recursive 则总能停止）

Star

思路同 concatenation，对于 RE 而言，构造 NTM 以猜测任意可能的 break，只要每一个 piece 都被接受则 M 接受。对于 recursive 而言，只需顺序尝试所有可能的 break 即可

Reversal

只需要在开始时将输入反转，然后正常模拟原本的 TM 即可，不论对 RE 还是 recursive 都是如此

Inverse Homomorphism

对于一个 homomorphism h ，只需要在开始时将 h 应用于输入 w ，然后模拟 TM 判断其是否接受 $h(w)$ 即可，不论对 RE 还是 recursive 都是如此

Homomorphism

对于 homomorphism h ，构造一个 NTM 猜测一个 x 使得 $h(x) = w$ ，若 x 能被原 TM 接受，则 w 被 NTM 接受，显然这个构造对 RE 是适用的而对 recursive 不适用，因为若是 $w \in h(L)$ 则总能找到符合的 x ，但是 x 的选择可以是 infinite 的，在没有符合要求的 x 时 TM 或许不会 halt