计算理论导论 课程讲义

酥雨
zusuyu@stu.pku.edu.cn

March 20, 2022

Outline

1. DFA/NFA, Regular Language, Pumping Lemma
2. Context-free Language, Pumping Lemma
3. Turing Machine
4. Undecidable Language
5. Time Complexity **P** and **NP**
6. Space Complexity **PSPACE**, **L** and **NL**
7. Polynomial Hierarchy
8. Circuit Complexity
9. Random Computation
10. Interactive Proof
11. (optional) Crypt, Quant, Learning

1 正则语言

定义 1 (Deterministic Finite Automaton, DFA). (确定性) 有限自动机是一个五元组 $(Q, \Sigma, \delta, q_0, F)$, 其中

- Q 是称为**状态**的有限集.
- Σ 是称为**字符集**的有限集.
- $\delta : Q \times \Sigma \to Q$ 被称为**转移函数**.
- $q_0 \in Q$ 称为**起始态**.
- $F \subseteq Q$ 称为**接受态 (终止态) 集合**.

称字符串 $w = w_1 w_2 \cdots w_m (w_i \in \Sigma)$ 可以被 DFA $M = (Q, \Sigma, \delta, q_0, F)$ 接受, 如果存在状态序列 $r_0, r_1, \cdots, r_m \in Q$ 满足(i)$r_0 = q_0$, (ii)$r_{i+1} = \delta(r_i, w_{i+1})$ $(\forall i = 0, 1, \cdots, m-1)$, (iii)$r_m \in F$.

所有可被 M 识别的字符串 w 构成集合 A, 则称 A 是 DFA M 的语言 (或者说 DFA M 识别/接受 A), 记为 $L(M) = A$.

定义 2 (正则语言). 正则语言就是能够被有限自动机识别的语言.

定义 3 (正则操作). 定义如下三种正则操作

- **Union**: $A \cup B = \{x | x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \cdots x_k | k \geqslant 0 \text{ and each } x_i \in A\}$.

注 1. 补集 $\overline{A} = \Sigma^* - A$ 操作在正则语言下是封闭的: 只需要把终止态集合 F 改成 $Q - F$ 即可.

定理 1. 正则操作 union 在正则语言下是封闭的: 把两个自动机放在一起跑就行了.

由于只利用已有的有限自动机模型证明 concatenation 和 star 的封闭性是困难的, 我们引入 “非确定性”.

定义 4 (Nondeterministic Finite Automaton, NFA). 非确定性有限自动机是一个五元组 $(Q, \Sigma, \delta, q_0, F)$, 其中 δ 不再是 $Q \times \Sigma \to Q$ 的函数, 而是 $Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ 的, 其中 $\mathcal{P}$ 表示幂集, Σ_ε 表示 $\Sigma \cup \{\varepsilon\}$.

相应的, 称字符串 $w = w_1 w_2 \cdots w_m (w_i \in \Sigma)$ 可以被 NFA $N = (Q, \Sigma, \delta, q_0, F)$ 接受, 如果 w 可以写成 $w = y_1 y_2 \cdots y_{m'} (y_i \in \Sigma_\varepsilon)$, 且存在状态序列 $r_0, r_1, \cdots, r_{m'} \in Q$ 满足(i)$r_0 = q_0$, (ii)$r_{i+1} \in \delta(r_i, y_{i+1})$ $(\forall i = 0, 1, \cdots, m'-1)$, (iii)$r_{m'} \in F$.

注 2. DFA 的每个状态对每种字符都有恰好一条转移出边, 而相对的, NFA 可能有零条、一条或者多条, 有几条出边就表示会创建出多少个独立的 “后继进程”. 此外还存在 ε 的出边, 表示可以不输入任何字符创建进程.

定理 2 (NFA 与 DFA 的等价性). 任何 NFA 都存在等效的 DFA.

证明. 对 k 个状态的 NFA, 构造一个 2^k 个状态的 DFA, 每个状态表示 “可能处在的 NFA 状态” 的子集.

形式化的, 对于 NFA $M = (Q, \Sigma, \delta, q_0, F)$, 构造 DFA $M' = (Q', \Sigma, \delta', q_0', F')$, 其中

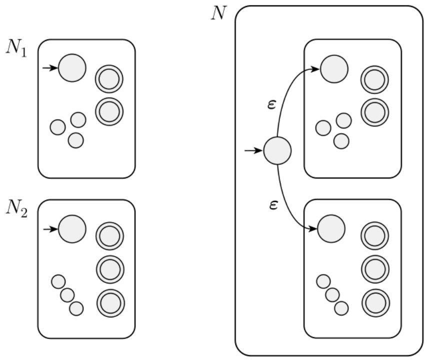
- $Q' = \mathcal{P}(Q)$.
- $\forall R \in Q', a \in \Sigma, \delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$.
- $q_0' = \{q_0\}$.
- $F' = \{R \in Q' | R \cap F \neq \varnothing\}$.

$\square$

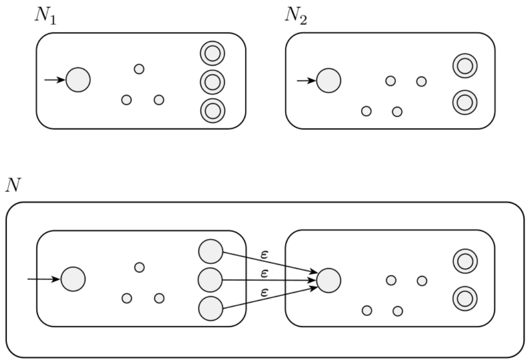
推论 1. 一个语言是正则的当且仅当可以被一台非确定性有限自动机识别.

定理 3. union, concatenation 和 star 在正则语言下都是封闭的.

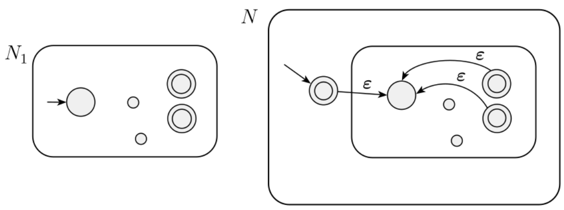
证明. 不多说了看图.



(a) proof for union



(b) proof for concatenation



(c) proof for star

□

定义 5 (正则表达式). 称 R 是正则表达式, 如果 R 为

- $\{a\}$, 其中 a 是字符集 Σ 中的某个元素
- $\{\varepsilon\}$, 其中 ε 表示空串
- $\varnothing$
- $(R_1 \cup R_2)$, 其中 R_1, R_2 是某两个正则表达式
- $(R_1 \circ R_2)$, 其中 R_1, R_2 是某两个正则表达式
- (R_1^*), 其中 R_1 是某个正则表达式

例 1. 对于任意正则表达式 R, $R \cup \varnothing = R \circ \varepsilon = R$, $R \circ \varnothing = \varnothing$, $\varnothing^* = \{\varepsilon\}$.

定理 4 (正则表达式与有限自动机的等价性). 一个语言是正则的当且仅当它可以被一个正则表达式描述.

证明. "$\Leftarrow$" 的证明是简单的, 只需要根据正则表达式 R 构造 NFA, 利用 "union, concatenation, star 的封闭性" 的构造性证明即可.

"$\Rightarrow$" 的证明中, 我们引入 GNFA 的定义 (每条转移边上的 label 是一个正则表达式), 然后分别展示如何把 DFA 转化成 GNFA 以及如何根据 GNFA 构造正则表达式.

DFA 转 GNFA 是简单的——只需要额外加入两个状态表示 q_{start} 和 q_{accept} 即可.

观察到一个 GNFA 有 $k \geqslant 2$ 个状态. 如果 $k = 2$, 那么 q_{start} 到 q_{accept} 的转移边上的正则表达式就是该有限自动机对应的正则表达式. 如果 $k > 2$, 那么考虑选出一个状态 q_{rip} 删除, 此时对于 $q_i, q_j \in Q \setminus \{q_{\text{rip}}\}$, 如果 $\delta(q_i, q_{\text{rip}}) = R_1, \delta(q_{\text{rip}}, q_{\text{rip}}) = R_2, \delta(q_{\text{rip}}, q_j) = R_3, \delta(q_i, q_j) = R_4$, 则修改 $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$. 归纳即可. □

定义 6 (**Generalized Nondeterministic Finite Automaton, GNFA**). 广义非确定性有限自动机是一个五元组 $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, 其中 δ 是 $(Q \setminus \{q_{\text{accept}}\}) \times (Q \setminus \{q_{\text{start}}\}) \to \mathcal{R}$ 的转移函数, $\mathcal{R}$ 表示字符集 Σ 上的所有正则表达式. 注意不失一般性地要求了只有唯一的接受态, 以及 $q_{\text{start}} \neq q_{\text{accept}}$.

定理 5 (**Pumping Lemma for Regular Language**). 如果 A 是正则语言, 那么存在一个数 p (称为 **pumping length**), 使得对于任意 A 中长度至少为 p 的字符串 s, s 都可分成三部分 $s = xyz$ 满足

- for each $i \geqslant 0$, $xy^iz \in A$,
- $|y| > 0$,
- $|xy| \leqslant p$.

证明. 取 pumping length p 为识别此正则语言的 DFA M 的状态集大小 $|Q|$. 对于任意长度至少为 p 的 $s \in A$, 其经过的状态序列至少长为 $p+1$. 根据**鸽巢原理**, 存在一个状态 q 经过了至少两次, 于是把从 q_{start} 走到 q 的部分视作 x, q 回到自身的环视作 y, 从 q 走到 q_{accept} 的部分视作 z, 便构造出了划分. □

注 3. 利用 pumping lemma 可以证明某个语言 B 不是正则语言, 通用的方式是: 先假设 B 是正则的, 导出 pumping length p 的存在性, 然后根据这个 p 构造 $s \in B$, 并验证其**不能**被划分为 $s = xyz$. 第三个条件 $|xy| \leqslant p$ 有时也是有用的.

例 2. $B = \{0^n1^n | n \geqslant 0\}$ 不是正则语言.

证明. 假设 B 是正则语言, 那么就存在 pumping length p. 考虑串 0^p1^p, 无论 y 取其何种子串, $xyyz$ 都不可能 $\in B$. 因此 B 不是正则语言. □

例 3. $C = \{w | w \text{ has an equal number of 0s and 1s}\}$ 不是正则语言.

证明. 假设 C 是正则语言, 那么就存在 pumping length p. 考虑串 0^p1^p, 注意到我们要求了 $|xy| \leqslant p$, 所以 y 只能包含 0, 此时 $xyyz \notin B$. 因此 C 不是正则语言.

另一种证法是: 考虑 $C \cap 0^*1^* = B$, 正则语言在 intersection 下是封闭的, 所以 C 正则会导出 B 正则. □

例 4. $F = \{ww | w \in \{0,1\}^*\}$ 不是正则语言.

证明. 考虑串 0^p10^p1, 注意到 y 只能包含 0, 从而 $xyyz \notin F$, 因此 F 不是正则语言. □

例 5. $D = \{1^{n^2} | n \geqslant 0\}$ 不是正则语言.

证明. 考虑串 1^{p^2}. 由于 $|y| \leqslant p$, 所以 $|xyyz| = p(p+1) < (p+1)^2$ 不可能是完全平方数, $xyyz \notin D$, 说明 D 不是正则语言. □

例 6. $E = \{0^i1^j | i > j\}$ 不是正则语言.

证明. 考虑串 $0^{p+1}1^p$, y 只能包含 0, 且 $|y| > 0$, 因此 xz 中 0 的个数不超过 1 的个数, $xz \notin E$, 说明 E 不是正则语言. □

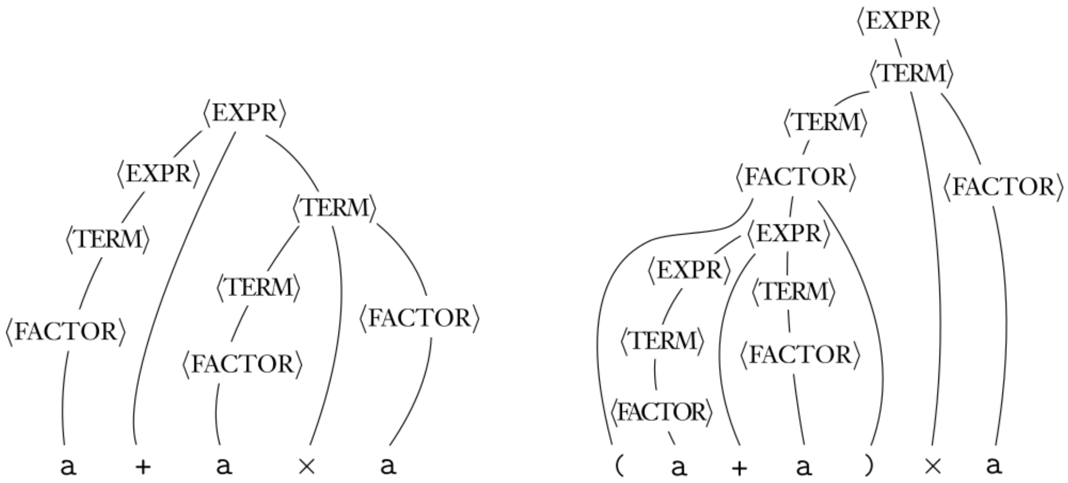
2 上下文无关文法

定义 7 (Context-Free Grammar/Language, CFG/CFL). 一个上下文无关文法是一个四元组 (V, Σ, R, S), 其中

- V 是称为**变量**的有限集,
- Σ 是称为**终止符**的有限集, 与 V 不交,
- R 是称为**规则**的有限集, 是从 V 到 $(V \cup \Sigma)^*$ 的映射,
- $S \in V$ 称为**起始变量**.

上下文无关语言就是上下文无关文法导出/生成的语言, 即 $\{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$.

定义 8 (parse tree). 形如这样子的东西.



命题 1. CFG 的描述能力严格强于有限自动机 (或者正则表达式).

证明. 对于任意的 DFA, 都可以构造与其等价的 CFG: 对每个状态 q_i 构造一个变量 R_i, 起始变量 R_0 对应起始态 q_0, 如果 $\delta(q_i, a) = q_j$, 就添加规则 $R_i \to aR_j$, 而如果 q_i 是接受态, 就添加规则 $R_i \to \varepsilon$.

而显然存在可被 CFG 描述的非正则语言, 比如 $\{0^n 1^n | n \in \mathbb{N}\}$. □

定义 9 (歧义性). 称一个串 w 由 CFG G 歧义生成, 如果存在 G 下 w 的两种 lestmost derivation (每次只替换最左边的变量) 方式, 或者说存在两棵不同的 parse tree 可以生成 w. 称一个 CFG G 是歧义的, 如果它可以歧义生成某些串.

定义 10 (固有歧义). 称一个 CFL L 是固有歧义的, 如果 L 只能由歧义的 CFG G 生成.

例 7. $\{a^i b^j c^k | i = j \text{ or } j = k\}$ 是固有歧义的.

定义 11 (Pushdown Automaton, PDA). 下推自动机是一个六元组 $(Q, \Sigma, \Gamma, \delta, q_0, F)$, 其中

- Q 是状态集,
- Σ 是输入字符集,
- Γ 是栈字符集,
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ 是转移函数,
- $q_0 \in Q$ 是起始态,

- $F \subseteq Q$ 是接受态集合.

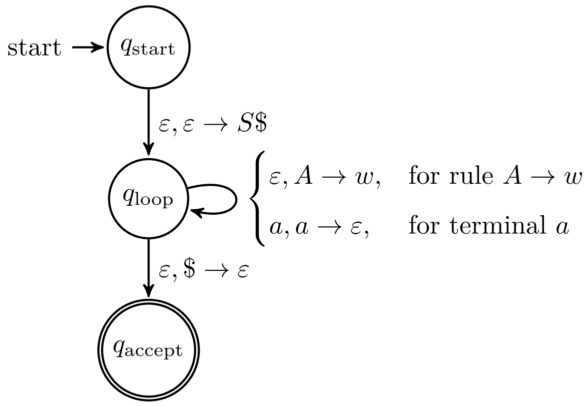
其中 $\Sigma_\varepsilon, \Gamma_\varepsilon$ 分别表示 $\Sigma \cup \{\varepsilon\}, \Gamma \cup \{\varepsilon\}$. $(q', b) \in \delta(q, c, a)$ 表示在状态 q 上被输入 c 字符时, 会先从栈顶 pop 出字符 a, 再向栈顶 push 进字符 b, 最后转移到状态 q'. 幂集 $\mathcal{P}$ 暗含了下推自动机是 nondeterministic 的.

称字符串 $w = w_1 w_2 \cdots w_m (w_i \in \Sigma_\varepsilon)$ 可以被 PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ 接受, 如果存在状态序列 $r_0, r_1, \cdots, r_m \in Q$ 和字符串 (栈) 序列 $s_0, s_1, \cdots, s_m \in \Gamma^*$, 满足

- $r_0 = q_0, s_0 = \varepsilon$,
- For $i = 0, 1, \cdots, m-1$, $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at, s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$,
- $r_m \in F$.

定理 6 (**下推自动机与上下文无关文法的等价性**). 一个语言是上下文无关的, 当且仅当存在某个下推自动机可以识别它.

证明. "⇒" : 需要根据 CFG 来构造 PDA. 一开始把 CFG 的起始变量写在栈上, 并保证在替换过程中栈顶始终是一个尚未替换的变量. 利用 nondeterminism 尝试每一种变量的替换方式. 每次只考虑替换栈顶的变量, 而如果栈顶是一个终止符, 就直接和输入匹配掉. 当输入匹配完且栈为空时, 代表输入串可接受.



"⇐" : 需要根据 PDA 来构造 CFG. 不妨假设[1]该 PDA 有如下特性: (i)只有一个接受态 q_{accept}, (ii)会在接受前清空栈, (iii)每次转移都会要么 push 要么 pop, 没有 both 和 neither 的情况. 构造变量 A_{pq} 表示所有能够使 PDA 从 "状态 p 且栈空" 转移到 "状态 q 且栈空" 的串组成的语言, 其中 $A_{q_0 q_{\text{accept}}}$ 是该 CFG 的起始变量. 按如下方式构造 CFG 的规则集合:

- 对于任意 $p, q, r, s \in Q, u \in \Gamma, a, b \in \Sigma_\varepsilon$, 如果 $(r, u) \in \delta(p, a, \varepsilon), (q, \varepsilon) \in \delta(s, b, u)$, 就添加规则 $A_{pq} \to a A_{rs} b$,
- 对于任意 $p, q, r \in Q$, 添加规则 $A_{pq} \to A_{pr} A_{rq}$,
- 对于任意 $p \in Q$, 添加规则 $A_{pp} \to \varepsilon$.

构造思路来源于考虑压栈弹栈的括号序列, 该序列要么被一个大括号包裹 (第一种), 要么由两个括号序列组成 (第二种). 可以归纳证明 A_{pq} 的构造方式与其含义的等价性. □

定理 7 (**Pumping Lemma for CFL**). 如果 A 是上下文无关语言, 那么存在一个数 p(称为 **pumping length**), 使得对于任意 A 中长度至少为 p 的字符串 s, s 都可以分成五部分 $s = uvxyz$ 满足

- for each $i \geqslant 0, uv^i xy^i z \in A$,
- $|vy| > 0$,

[1]需要简短地说明转化的可行性. 前两条只需要添加额外的结束状态和转移函数即可, 第三条需要在所有 both 和 neither 的转移中间插入中间状态.

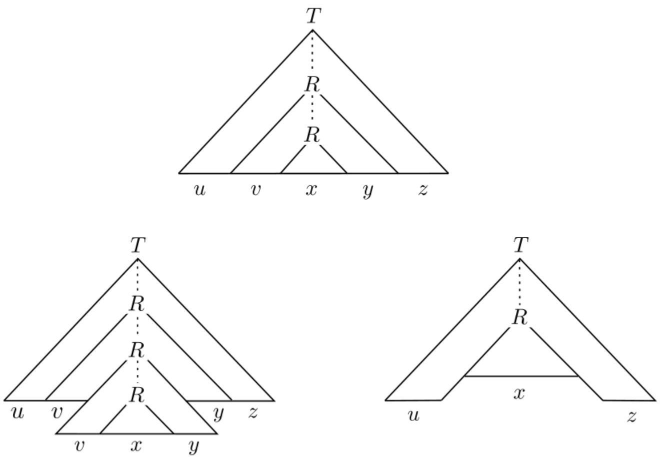
- $|vxy| \leqslant p$.

证明. 设 b 为规则中的最大“度数”, 即替换字符串的最大长度. 如果 parse tree 的树高是 h(根的深度是 0), 那么生成的字符串长度至多为 b^h.

取 pumping length p 为 $b^{|V|+1}$. 长度至少为 p 的串对应的 parse tree 树高至少为 $|V|+1$, 故存在一条“直链”上有至少 $|V|+1$ 个变量, 根据**鸽巢原理**, 存在一个变量出现至少两次, 记为 R, 那么对于 R 就可以无限复制或者把两次出现压缩成一次 (如图).

为了满足第二个条件, 我们要求 parse tree 必须是”最简”的, 因为只有冗余的替换方式才会导致两次 R 的出现之间没有任何字符实际生成.

为了满足第三个条件, 取 R 为满足条件的“深度最大”的, 即两个 R 都出现在最底下 $|V|+1$ 层. 此时 $|vxy|$ 对应上面的 R 的子树大小, 受深度限制不超过 $b^{|V|+1} = p$.



□

例 8. $B = \{a^n b^n c^n | n \geqslant 0\}$ 不是上下文无关语言.

证明. 假设 B 是正则语言, 那么就存在 pumping length p. 考虑串 $a^p b^p c^p$, 注意到 $|vxy| \leqslant p$ 故不可能含有超过两种字符, 那么在重复时就不可能保证三种字符出现次数仍然相同, 从而 B 不是上下文无关语言. □

例 9. $C = \{a^i b^j c^k | 0 \leqslant i \leqslant j \leqslant k\}$ 不是上下文无关语言.

证明. 考虑串 $a^p b^p b^p$, 注意 v, y 分别只能包含一种字符, 否则就会出现顺序错乱. 无论分别包含什么字符, 考虑 pumping up 或者 pumping down, 总可以使得生成的新字符串不属于 C, 从而 C 不是上下文无关语言. □

例 10. $D = \{ww | w \in \{0,1\}^*\}$ 不是上下文无关语言.

证明. 考虑 $s = 0^p 1^p 0^p 1^p$.

首先指出 vxy 必须跨域 s 的中点. 假设 vxy 只出现在 s 的左半边, 那么 uv^2xy^2z 中, 中点右侧的字符一定是 1(因为 $|vy| \leqslant p$, 只会把 $\frac{|vy|}{2} \leqslant \frac{p}{2}$ 个 1 推到右半边), 而起始字符是 0 说明该串不是 ww 形式的.

但如果 vxy 跨越 s 的中点, 那么就一定跟前 p 个 0 与后 p 个 1 无交, 因此 uxz 就会形如 $0^p 1^i 0^j 1^p$, 其中 $i, j < p$, 这显然不属于 D. □

3 图灵机

定义 12 ((Deterministic Turing Machine, TM)). (确定性) 图灵机是一个七元组 $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, 其中

- Q 是状态集.
- Σ 是不包含空格符 $\square$ 的输入字符集, Γ 是纸带字符集. $\Sigma \subseteq \Gamma$.
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, S, R\}^k$ 是转移函数.
- $q_0, q_{\text{accept}}, q_{\text{reject}}$ 分别是起始态, 接受态和拒接态.

输入字符串写在第一条纸带的开头, 之前有一个 $\triangleright$ 作为起始表示.

当图灵机运行到 q_{accept} 或者 q_{reject} 时, 会停机. 因此可以将其看成一个 $\Sigma^* \to \{0,1\}$ 的函数. 一般地, 对于图灵机 M 和字符串 $x \in \Sigma^*$, 我们有 $M(x) \in \{0,1\}$, 称 $M(x) = 1$ 如果对于输入 x M 会到达接受态, 称 $M(x) = 0$ 如果对于输入 x M 会到达拒绝态**或者不停机**.

一个 **configuration** 包含当前所在状态, 当前所有纸带上的信息, 和当前所有纸带头的位置. 称图灵机 M 接受字符串 w, 如果存在一个 configuration 序列 $C_0, C_1, \cdots, C_t$ 满足 (i) C_0 是起始 configuration, (ii) 每个 C_i 都在一步内跳转到 C_{i+1}, (iii) C_t 是接受 configuration.

定义 13 (图灵可识别与图灵可判定). 称一个语言是**图灵可识别 (Turing-recognizable)** 的, 如果存在一台图灵机可以识别它 (i.e. 接受其中的每一个字符串). 称一台图灵机是一个 **decider**, 如果它对于任何输入都不会无限循环. 称一个语言是**图灵可判定 (Turing-decidable)** 的, 如果存在一台 decider 可以识别它.

定义 14 (函数的计算, 运行时间). 考虑函数 $f : \{0,1\}^* \to \{0,1\}^*$ 以及 $T : \mathbb{N} \to \mathbb{N}$, 令 M 为一图灵机. 我们称 M 计算了函数 f, 如果对于任意 $x \in \{0,1\}^*$, 只要 M 的输入被初始化为 x, 它就能在输出 "输出纸带" 上写下 $f(x)$ 并停机. 称 M 在 $T(n)$ 的时间内计算了 f, 如果它计算每个 x 都可以在 $T(|x|)$ 步内停机.

定义 15 (Time-constructible functions). 称一个函数 $T : \mathbb{N} \to \mathbb{N}$ 是 time-constructible 的, 如果 $T(n) \geqslant n$ 且存在运行时间为 $T(n)$ 的计算函数 $x \to \llcorner T(|x|) \lrcorner$ 的图灵机 M, 其中 $\llcorner x \lrcorner$ 表示 x 的 binary representation.

例 11. $A = \{w\#w | w \in \{0,1\}^*\}$ 可以被 (单纸带) 图灵机识别.

证明. 给待匹配的两个位置打上标记, 每次前后移动找标记, 用状态记录已经看过的字符, 若比较失败则直接 reject, 否则向后移动标记继续比较直到全部比完. 以上的描述的图灵机的运行时间是 $T(n) = O(n^2)$ 的. □

例 12. $B = \{ww | w \in \{0,1\}^*\}$ 可以被双纸带图灵机识别.

证明. 没有 # 记号, 无法方便地找到中间位置. 可以在二号纸带上把输入复制一遍, 然后二号纸带头移动到中间 (一号纸带头走一步, 二号纸带头走两步), 顺序比较即可. 运行时间是 $O(n)$. □

图灵机由字符集大小, 纸带数量等的不同, 产生了许多不同的变种. 它们在计算能力上会有所不同吗?

命题 2 (大字符集规约到小字符集). 对于任意 $f : \{0,1\}^* \to \{0,1\}$ 以及 time-constructible function $T : \mathbb{N} \to \mathbb{N}$, 如果 f 可以被图灵机 M 以 $T(n)$ 时间计算, 那么它也可以被一台字符集为 $\{0, 1, \square, \triangleright\}$ 的图灵机 $\tilde{M}$ 以 $O(T(n) \log |\Gamma|)$ 的时间计算.

证明. 任意 Γ 中的字符都可以用 $\log |\Gamma|$ 个比特表示. 每步转移时先用 $\log |\Gamma|$ 步读出纸带上一个字符的 encoding 并存入状态, 再根据转移函数进行移动, 最后用 $\log |\Gamma|$ 步写下新字符的 encoding 表示. □

命题 3 (多纸带规约到单纸袋). 对于任意 $f : \{0,1\}^* \to \{0,1\}$ 以及 time-constructible function $T : \mathbb{N} \to \mathbb{N}$, 如果 f 可以被有 k 条纸带的图灵机 M 以 $T(n)$ 时间计算, 那么它也可以被一台单纸带图灵机 $\tilde{M}$ 以 $O(kT^2(n))$ 的时间计算. 单纸带指的是只有一条可读可写的纸带, 它同时扮演了输入、工作和输出纸带的角色.

证明. 把单条纸带上的位置按照模 k 余数分配给 k 条纸带, 对每种字符 a 新建字符 $\hat{a}$ 表示所在纸带的纸带头指向这个字符. 注意到运行时间 $T(n)$ 的图灵机, 对于长度为 n 的输入, 最多只会用到前 $T(n)$ 个位置, 所以每步转移时花费 $O(T(n))$ 的代价搜索每个纸带头的位置, 运行时间为 $O(kT^2(n))$. □

注 4 (健忘的图灵机, oblivious Turing Machine). 头部移动与输入长度有关, 而与输入的具体内容无关, 即对于任意 $x \in \{0,1\}^*$ 以及 $i \in \mathbb{N}$, $M(x)$ 执行到第 i 步时读写头的位置是只关于 $|x|$ 和 i 的函数. 可以证明图灵机可以平方规约到健忘的图灵机.

命题 4 (双向图灵机规约到单向图灵机). 对于任意 $f : \{0,1\}^* \to \{0,1\}$ 以及 time-constructible function $T : \mathbb{N} \to \mathbb{N}$, 如果 f 可以被双向图灵机 (纸带的两个方向都有无限长)M 以 $T(n)$ 时间计算, 那么它也可以被一台单向图灵机 $\tilde{M}$ 以 $O(T(n))$ 的时间计算.

论点 1 (Church-Turing Thesis). 任何物理上可实现的计算设备都可以被图灵机实现.

定理 8 (通用图灵机存在). 存在图灵机 $\mathcal{U}$ 使得对于任意 $x, \alpha \in \{0,1\}^*$, $\mathcal{U}(x,\alpha) = M_\alpha(x)$, 其中 M_α 为被 α 表示的图灵机. 进一步地, 如果 M_α 对于 x 在 T 步内停机, 则 $\mathcal{U}(x,\alpha)$ 可以在 $CT\log T$ 步内停机, 其中 C 是一个仅依赖于 M_α 的字符集大小、纸袋条数、状态数的常数.

证明. 构造 $\mathcal{U}$ 为一台五纸带图灵机, 五条纸带分别为

- Input, 被模拟图灵机 M 的输入.
- Description of M, 主要是转移函数 δ.
- Simulation of M, 记录纸带信息. 这里需要把 M 规约成单纸带, 因而会产生平方的 overhead.
- Current state of M, 这部分不能简单地存在 $\mathcal{U}$ 的状态里, 因为对于 $\mathcal{U}$ 来说这不是常数.
- Output, M 的输出.

注意每步模拟的过程中, 读取 M 所在状态, 读取转移函数都是关于 n 常数时间的.

可以设计一种类似势能分析的算法, 把多纸带规约单纸带的 overhead 降到 $O(n\log n)$, 从而使通用图灵机模拟的复杂度优化到 $O(T\log T)$. □

4 不可判定语言

定义 16 (可识别语言 (Recognizable Language), 可判定语言 (Decidable Language)). 可识别语言就是能够被一台图灵机识别的语言. 可判定语言就是能够被一台 decider 识别的语言.

定理 9. 存在不可识别语言.

证明. 只需要考虑 "语言" 与 "可识别语言" 的基数. 前者的基数是 $2^{\aleph_0} = \aleph_1$, 后者的基数不超过图灵机的基数 (因为存在可识别语言到图灵机的单射), 而图灵机可以被有限长的字符串描述, 因此是可数的. □

定义 17 (可计算函数 (Computable Function)). 可计算函数就是可以被一台图灵机计算的函数. 特别的, 考虑函数 $f : \{0,1\}^* \to \{0,1\}$, 则 f 是可计算函数当且仅当 $L = \{x \in \{0,1\}^* | f(x) = 1\}$ 是可判定语言.

定理 10. 存在不可计算函数/存在不可判定语言.

证明. 假设存在一个映射 $\alpha \to M_\alpha$ 可以把任意字符串映到一台图灵机 (以任意格式编码, 再把非法格式的串映到某台特定的图灵机即可). 考虑函数 $\mathrm{UC}(\alpha) = 1 - M_\alpha(\alpha)$, 我们指出 UC 是不可计算函数.

假设 UC 可计算, 考虑计算 UC 的图灵机 M. 考虑 $\mathrm{UC}(\llcorner M \lrcorner)$, 由于 M 计算了 UC, 我们知道 $\mathrm{UC}(\llcorner M \lrcorner) = M(\llcorner M \lrcorner)$, 但根据 UC 的定义, 又有 $\mathrm{UC}(\llcorner M \lrcorner) = 1 - M(\llcorner M \lrcorner)$, 产生了矛盾. □

例 13 (停机问题不可判定). $\mathsf{HALT} = \{\langle \llcorner M \lrcorner, \alpha \rangle | M \text{ halts on } \alpha\}$ 是不可判定语言.

证明. 假设存在 M_{HALT} 可以判定 HALT.

利用 M_{HALT} 可以构造判定语言 $L = \{\alpha | \mathrm{UC}(\alpha) = 1\}$ 的 decider: 计算 $M_{\mathsf{HALT}}(\langle \alpha, \alpha \rangle)$, 如果得到 0 (说明 M_α 对 α 不停机) 则直接输出 1, 否则输出 $M_\alpha(\alpha)$ 的结果. 这与 UC 不可计算相矛盾. 因此 HALT 不可判定. □

例 14 (接受问题不可判定). $\mathsf{AC} = \{\langle \llcorner M \lrcorner, \alpha \rangle | M \text{ accepts } \alpha\}$ 是不可判定语言.

证明. 利用 M_{AC} 可以直接构造 M_{UC}: 只要把 $M_{\mathsf{AC}}(\langle \alpha, \alpha \rangle)$ 的输出取反即可. □

定义 18 (映射规约, Mapping Reduction). 称语言 A 可映射规约到 (is mapping reducible to) 语言 B, 如果存在可计算函数 $f : \Sigma^* \to \Sigma^*$, 使得 $w \in A$ 当且仅当 $f(w) \in B$, 记作 $A \leqslant_m B$.

命题 5. 如果 $A \leqslant_m B$, 则

- 如果 B 可判定, 则 A 也可判定.
- 如果 A 不可判定, 则 B 也不可判定.

例 15. $\mathsf{NAC} = \{\llcorner M \lrcorner | M \text{ accept nothing}\}$ 是不可判定语言.

证明. 考虑构造映射 $f : \overline{\mathsf{AC}} \to \mathsf{NAC}$ 满足 $w \in \overline{\mathsf{AC}} \Leftrightarrow f(w) \in \mathsf{NAC}$. 令 $f(\langle \llcorner M \lrcorner, \alpha \rangle) = \llcorner M' \lrcorner$ 其中 M' 不管输入直接运行 $M(\alpha)$. f 显然是可计算的, 故 $\overline{\mathsf{AC}} \leqslant_m \mathsf{NAC}$, 而可判定语言关于补集的封闭性导致 $\overline{\mathsf{AC}}$ 是不可判定语言, 从而 NAC 是不可判定语言. □

例 16. $\mathsf{EQU} = \{\langle \llcorner M_1 \lrcorner, \llcorner M_2 \lrcorner \rangle | L(M_1) = L(M_2)\}$ 是不可判定语言.

证明. 考虑构造映射 $g : \mathsf{NAC} \to \mathsf{EQU}$. 取 $g(\llcorner M \lrcorner) = \langle \llcorner M \lrcorner, \llcorner M' \lrcorner \rangle$ 其中 M' 是拒绝一切输入的图灵机. 于是 $\llcorner M \lrcorner \in \mathsf{NAC} \Leftrightarrow \langle \llcorner M \lrcorner, \llcorner M' \lrcorner \rangle \in \mathsf{EQU}$, $\mathsf{NAC} \leqslant_m \mathsf{EQU}$, 从而 EQU 是不可判定语言. □

定理 11. 语言 A 可判定当且仅当 A 与 $\overline{A}$ 均可识别.

证明. $\Rightarrow$: 显然.

$\Leftarrow$: 记 A 可被 M_1 识别, $\overline{A}$ 可被 M_2 识别, 考虑**并行**运行 M_1 和 M_2, 总有一个会给出结果. □

推论 2. HALT 和 AC 都是不可判定的可识别语言. 这说明 $\overline{\mathsf{HALT}}$ 和 $\overline{\mathsf{AC}}$ 都是不可识别语言.

后面都是自己瞎写的阿拉丁.

定义 19 (DTIME 与 P). 对于函数 $T:\mathbb{N}\to\mathbb{N}$, 称语言 $L\in\mathbf{DTIME}(T(n))$, 如果存在常数 $c>0$ 和一台运行时间为 $c\cdot T(n)$ 的图灵机可以决定 L.

$\mathbf{P}=\bigcup_{c\geqslant 1}\mathbf{DTIME}(n^c)$.

5 NP 与 NP-complete

定义 20 (NP). 语言 $L \subseteq \{0,1\}^*$ 属于 **NP**, 如果存在一个多项式 $p : \mathbb{N} \to \mathbb{N}$ 和一个多项式时间图灵机 M(称其为 L 的 **verifier**) 使得对于任意的 $x \in \{0,1\}^*$, 都有

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)}, M(x,u) = 1$$

如果 $x \in L$ 与 $u \in \{0,1\}^{p(|x|)}$ 满足 $M(x,u) = 1$, 则称 u 是 x 的一个 **certificate**.

命题 6. 定义 $\mathbf{EXP} = \bigcup_{c>1} \mathbf{DTIME}(2^{n^c})$, 则 $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$.

定义 21 (非确定图灵机与 NTIME). 非确定图灵机 (Nondeterministic Turing Machine, NDTM) 是有两个转移函数 δ_0, δ_1 和一个特定状态 q_{accept} 的图灵机 M, 每步转移时, 可以任意选择遵从某一个转移函数. 对于输入 x, 称 $M(x) = 1$ 当且仅当存在一个选择序列可以使 M 到达 q_{accept} 状态, 否则——任意选择序列都无法在停机前到达 q_{accept}——就认为 $M(x) = 0$. 称 M 的运行时间为 $T(n)$, 如果对于任意输入 $x \in \{0,1\}^*$ 以及任意的选择序列, M 都会在 $T(|x|)$ 步内到达 q_{accept} 或者 q_{halt}.

对于 $T : \mathbb{N} \to \mathbb{N}$ 和语言 $L \subseteq \{0,1\}^*$, 称 $L \in \mathbf{NTIME}(T(n))$, 如果存在常数 $c > 0$ 和一个运行时间为 $c \cdot T(n)$ 的非确定图灵机 M, 满足对于任意的 $x \in \{0,1\}^*$, $x \in L \Leftrightarrow M(x) = 1$.

定理 12. $\mathbf{NP} = \bigcup_{c \geqslant 1} \mathbf{NTIME}(n^c)$.

证明. 证明的核心思路在于非确定图灵机的选择序列可以看作 x 的一个 certificate, 反之亦然. □

定义 22 (规约, NP-hard 与 NP-complete). 称语言 $L \subseteq \{0,1\}^*$ 可**多项式时间规约**到语言 $L' \subseteq \{0,1\}^*$(记作 $L \leqslant_p L'$), 如果存在一个多项式时间可计算函数 $f : \{0,1\}^* \to \{0,1\}^*$ 使得对于任意 $x \in \{0,1\}^*$, $x \in L \Leftrightarrow f(x) \in L'$.

称 L' 是 **NP**-hard, 如果对于任意 $L \in \mathbf{NP}$, $L \leqslant_p L'$. **NP**-complete = **NP** ∩ **NP**-hard.

定理 13 ($\leqslant_p$ 的传递性).

- 若 $L \leqslant_p L'$ 且 $L' \leqslant_p L''$, 则 $L \leqslant_p L''$.
- 如果 L 是 **NP**-hard 且 $L \in P$, 则 $\mathbf{P} = \mathbf{NP}$.
- 如果 L 是 **NP**-complete, 则 $L \in P$ 当且仅当 $\mathbf{P} = \mathbf{NP}$.

定理 14 (Cook-Levin Theorem). SAT, 3SAT 是 **NP**-complete.

6 对角线法则

定理 15 (Time Hierarchy Theorem). f, g 是满足 $f(n)\log f(n) = o(g(n))$ 的 time constructible 的函数, 则

$$\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n))$$

证明. 考虑这样的图灵机 D: 对于 x, 用通用图灵机 $\mathcal{U}$ 模拟 M_x(x 描述的图灵机) 运行至多 $g(|x|)$ 步 (是 U 的 $g(|x|)$ 步而不是 M_x 的 $g(|x|)$ 步), 如果 $\mathcal{U}$ 在 $g(|x|)$ 步数内输出了 $b \in \{0,1\}$, 则 D 输出 $1-b$; 否则 D 输出 0.

根据定义, D 对于任何输入 x 都会在 $g(|x|)$ 步内停机, 因此 D 决定的语言 L 属于 $\mathbf{DTIME}(g(n))$. 我们通过反证法证明 $L \notin \mathbf{DTIME}(f(n))$. 先叙述否命题: 存在图灵机 M 和常数 c, 使得对于任意输入 $x \in \{0,1\}^*$, M 都能在 $cf(|x|)$ 步内输出与 D 相同的结果.

对于输入 x, 用通用图灵机 $\mathcal{U}$ 模拟 M 只需要 $c'cf(|x|)\log f(|x|)$ 步, 其中 c' 是不依赖于 $|x|$ 的一个常数. 由于 $f(n)\log f(n) = o(g(n))$[2], 故存在充分大的 n_0 使得 $g(n) > c'cf(n)\log f(n)$ 对于任意 $n \geqslant n_0$ 均成立. 令 x' 表示 M 的某个长度大于 n_0 的表示, 那么

- D 会输出与 M 相同的结果, 因为这是 M 的定义;
- D 会输出与 M 不同的结果, 因为 $c'cf(n)\log f(n) < g(n)$ 使得 $\mathcal{U}$ 对 M 的模拟已经结束了, 根据 D 的定义, D 应该输出相反的结果.

产生了矛盾. 因此 $\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n))$. □

定理 16 (Nondeterministic Time Hierarchy Theorem). f, g 是满足 $f(n+1) = o(g(n))$ 的 time constructible 的函数, 则

$$\mathbf{NTIME}(f(n)) \subsetneq \mathbf{NTIME}(g(n))$$

定理 17 (Ladner's Theorem). 如果 $\mathbf{P} \neq \mathbf{NP}$, 则存在语言 $L \in \mathbf{NP} \setminus \mathbf{P}$, 即非 $\mathbf{NP}$-complete 的 $\mathbf{NP}$ 语言.

[2] little-o 不能替换成 big-O, 我只能说懂的都懂.

7 空间复杂性

定义 23 (**运行空间, SPACE 与 NSPACE**). 对于 $S : \mathbb{N} \to \mathbb{N}$ 和 $L \subseteq \{0,1\}^*$, 称 $L \in \mathbf{SPACE}(S(n))$, 如果存在常数 c 以及可以决定 L 的图灵机 M, 满足在对任意长度为 n 的输入的计算中, M 只会访问到至多 $c \cdot S(n)$ 个 work tapes 上 (不包含 input) 的位置, 称 M 的运行空间为 $O(S(n))$.

类似地可以定义 **NSPACE**, 这里要求在任何一种决策下用到的位置数量都不超过 $c \cdot S(n)$.

定义 24 (**Space constructible functions**). 称 $S : \mathbb{N} \to \mathbb{N}$ 是 space constructible 的, 如果存在图灵机可以对于输入 x, 在 $O(S(|x|))$ 的空间内计算 $S(|x|)$.

注 5. 相比于 time constructible functions, 我们不要求 space constructible functions 满足 $S(n) \geqslant n$, 但为了能够 "记住在输入纸带上的位置", 我们一般会要求 $S(n) \geqslant \log n$.

定理 18. 对于任何 space constructible 的函数 $S : \mathbb{N} \to \mathbb{N}$, 有

$$\mathbf{DTIME}(S(n)) \subseteq \mathbf{SPACE}(S(n)) \subseteq \mathbf{NSPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))})$$

证明. 前两个 $\subseteq$ 都是平凡的, 只考虑证明最后一个.

我们称一台 (确定或非确定) 图灵机 M 的一个 **configuration** 包含 (i) work tape 上的所有非空字符; (ii) 所有纸带的 head 位置; (iii) M 所处的状态, 则对于确定的输入 $x \in \{0,1\}^*$, 一个 configuration 的后继 configuration 是 (a) 对于图灵机来说, 唯一确定的; (b) 对于非确定图灵机来说, 至多唯二确定的. 把 configuration 之间的转移看成一张有向图, 记作 $G_{M,x}$. 不失一般性假设 M 只有一种 configuration C_{accept} 满足 "输出 1 后停机" (可以让图灵机在停机前擦除所有中间记录), 这样 $M(x) = 1$ 就等价于 $G_{M,x}$ 中存在一条 C_{start} 到 C_{accept} 的路径.

陈述两个事实:

- 给定 M, x, $G_{M,x}$ 中的每个节点用 $O(S(n))$ 个 bit 来表示, 也即, $G_{M,x}$ 只有 $2^{O(S(n))}$ 个节点.
- 对于任意两个 configuration C, C', 存在 $O(S(n))$ 大小的 CNF $\varphi_{M,x}$ 满足 $\varphi_{M,x}(C, C') = 1$ 当且仅当 $G_{M,x}$ 中 C 有边连向 C'.

因此用 $2^{O(S(n))}$ 的时间把整张 $G_{M,x}$ 建出来, 再 BFS 一下即可验证 C_{start} 到 C_{accept} 是否连通. □

定义 25 (**PSPACE, NPSPACE, L and NL**).

$$\mathbf{PSPACE} = \bigcup_{c \geqslant 1} \mathbf{SPACE}(n^c)$$
$$\mathbf{NPSPACE} = \bigcup_{c \geqslant 1} \mathbf{NSPACE}(n^c)$$
$$\mathbf{L} = \mathbf{SPACE}(\log n)$$
$$\mathbf{NL} = \mathbf{NSPACE}(\log n)$$

推论 3. $\mathbf{NP} \subseteq \mathbf{PSPACE}$, 因为都可以暴力枚举答案, 用多项式空间存下来然后验证.

推论 4. 在 定理 18 中分别代入 $S(n) = \log n, S(n) = n^c$, 可以得到

$$\mathbf{L} \subseteq \mathbf{P} \qquad \mathbf{PSPACE} \subseteq \mathbf{EXP}$$

例 17.

$$\mathsf{PATH} = \{\langle G, s, t\rangle : G \text{ is a direct graph in which there is a path from } s \text{ to } t\}$$

即判断图中两点之间是否存在一条路径. 显然 $\mathsf{PATH} \in \mathbf{NL}$, 但其是否属于 $\mathbf{L}$ 仍是一个 open problem.

定理 19 (Space Hierarchy Theorem). f, g 是满足 $f(n) = o(g(n))$ 的 space constructible 的函数, 则

$$\mathbf{SPACE}(f(n)) \subsetneq \mathbf{SPACE}(g(n))$$

证明. 技术细节在于通用图灵机 $\mathcal{U}$ 模拟图灵机 M 只需要常数倍的空间, 所以相比于 Time Hierarchy Theorem 没有了对数项. 其余部分跟 Time Hierarchy Theorem 的证明类似, 就不再赘述了. □

定义 26 (PSPACE-hard, PSPACE-complete). 称 L' 是 **PSPACE**-hard, 如果对于任意 $L \in \mathbf{PSPACE}$, $L \leqslant_p L'$. **PSPACE**-complete = **PSPACE** ∩ **PSPACE**-hard.

例 18.

$$\mathsf{SPACE\ TMSAT} = \{\langle M, w, 1^n\rangle : \text{DTM } M \text{ accepts } w \text{ in space } n\}$$

这是一个 **PSPACE**-complete 语言.

定义 27 (Quantified Boolean formula, QBF). 一个 QBF 是形如 $Q_1x_1Q_2x_2\cdots Q_nx_n\varphi(x_1, x_2, \cdots, x_n)$ 的公式, 其中 $Q_i \in \{\forall, \exists\}$, x_i 的取值是 $\{0, 1\}$, φ 是一个 plain(unquantified) boolean formula .

上述定义专注于讨论**前束范式**的 QBF , 因为非前束范式都可以转化成等价的前束范式. 一个 QBF 有真值 true 或 false .

用 TQBF 表示所有为真的 QBF 的集合.

定理 20. TQBF is **PSPACE**-complete.

证明. 先证明 TQBF ∈ **PSPACE**. 这个是简单的, 因为判定可以通过 dfs 实现, 而 dfs 只需要 $O(n+m)$ 的空间, 其中 n 是变量数, m 是 QBF 的长度.

再证明任意 $L \in \mathbf{PSPACE}$ 都满足 $L \leqslant_p \mathsf{TQBF}$. 假设 M 是在 $S(n)$ 空间内计算 L 的图灵机, 考虑输入 $x \in \{0, 1\}^*$. 考虑 configuration graph $G_{M,x}$, 我们陈述过图中每个点可以用 $m = O(S(n))$ 个 bit 来表示, 以及存在一个 CNF $\varphi_{M,x}$ 满足 $\varphi_{M,x}(C, C') = \text{true}$ 当且仅当 $G_{M,x}$ 中有 $C \to C'$ 的边.

考虑根据 $\varphi_{M,x}$ 来构造我们想要的 QBF ψ . 用 ψ_i 表示一个 QBF , $\psi_i(C, C') = \text{true}$ 当且仅当 $G_{M,x}$ 中存在一条长度不超过 2^i 从 C 到 C' 的路径, 那么显然 $\psi = \psi_m(C_{\text{start}}, C_{\text{accept}}), \psi_0(C, C') = \varphi_{M,x}(C, C') \vee (C = C')$. ψ_i 可以递归定义: 对于 $i \geqslant 1$, $\psi_i(C, C') = \exists C'' \psi_{i-1}(C, C'') \wedge \psi_{i-1}(C'', C')$.

一个技术细节是需要改进递归定义的具体方式以保证 ψ 的长度是多项式级别的. 可以用一种看上去有点奇怪, 但与前述定义等价的形式:

$$\psi_i(C, C') = \exists C'' \forall D_1 \forall D_2 ((D_1, D_2) = (C, C'') \wedge (D_1, D_2) = (C'', C')) \Rightarrow \psi_{i-1}(D_1, D_2)$$

这样构造出的 QBF ψ 的长度是 $O(m^2) = O(S^2(n))$ 的. □