CHAPTER 6

将语言划分为可判定与不可判定两类，固然有助于我们认清问题的难度，但是无疑太过粗糙。在可判定的语言之中，是否有些语言比另一些更「难」判定？判定一门语言所耗时间及存储代价是否高于另一门语言？这个问题的解答必须仰赖于对「复杂性」的精确刻画。自本章起，我们便来建立一套完整的体系。

欲度量「复杂性」，最容易想到的两个维度便是时间与空间——判定一门语言要花费多少时间、占用多少存储空间？当然，可以有许多别的维度，例如TM会有多少状态、改写了多少次存储单元，等等。但事实上，在一个维度上深究以后，往往可将研究方法移植到其它维度，因而我们仅将笔墨花在研究时间复杂性与空间复杂性上。本章研究前者。

凡是研究一个对象，必须得统一标准，研究时间复杂性亦然。什么是时间？是在什么模型上花费的时间？这些都是须达成共识的。

def 运行时间.

设M是一台TM（或多带TM），那么M在输入w时计算流程之长度称为M在w下的运行时间。

设N是一台非确定TM (NTM)，那么N在输入w时，所有计算流程中最长的长度称为N在w下的运行时间。

remark. 为何N的运行时间之定义如此奇异呢？若结合N的接纳/拒绝原则来看，则显得非常合理：N接纳w ⇔ ∃计算

流程使终态为q_{accept}，也就是说，不等到所有分支都结束，我们就不能100%肯定N是否接纳w。从而上面定义的「运行时间」意义为「能够~~确定N是否接纳~~让N的计算结果尘埃落定的时间」。

以上定义真的绝对合理吗？可以说是，也可以说不是。一方面，它很形象地刻画了TM运转的场景——如同钟表一样过一秒转移一步。另一方面，它又隐含了这种假设：无论每步转移做的工作是轻松（比如仅仅右移读写头）还是繁复（比如将存储单元改写以后再移读写头），其耗时都是一样的。

为此，我们要说：它相当合理，但不完美，可能与现实有一点差距。不过，为了让讨论不那么繁琐，有误差就有误差罢；更何况，后面我们还会看到，这点误差根本不影响我们对于「P类」和「NP类」的定义。

一台TM/NTM的运行时间是与输入w相关的。为简化之，我们有如下定义：

def 时间复杂度.

设M是一台TM/多带TM/NTM。定义函数$T: \mathbb{N} \to \mathbb{N}$

$$T(n) := \max_{w:|w|=n} (\text{M在w下的运行时间})$$

称$T(n)$为M的时间复杂度。

remark. 该定义充满了悲观色彩，故可称为「最坏情形时间复杂度」。我们可定义更为乐观的「平均时间复杂度」：$T(n) = \dfrac{\sum_{|w|=n} (\text{M在w下的}\dots)}{|\Sigma|^n}$

值得说明的是，对于同一门语言A，我们可以造出无数种TM/多带TM/NTM来判定A，其时间复杂度可以各不相同。

用聪明办法比用笨办法来得快，那是自然的；借助不确定性使计算加速，也是极可能的。所谓「时间复杂度」，是对于一台给定机器的效率度量，而根本无关乎语言的难度。欲度量后者，则要把具体的机器给「抽离掉」：

def TIME($f(n)$)语言类。

$TIME(f(n)) := \{$语言$A \mid \exists$ TM $M: \mathcal{L}(M)=A$ 且M的时间复杂度$T(n)=O(f(n))\}$

def k-TIME($f(n)$)语言类。

$k\text{-}TIME(f(n)) := \{$语言$A \mid \exists$ k带TM $M: \mathcal{L}(M)=A$ 且M的时间复杂度$T(n)=O(f(n))\}$

def NTIME($f(n)$)语言类.

$NTIME(f(n)) := \{$语言$A \mid \exists$ NTM $N: \mathcal{L}(N)=A$ 且N的时间复杂度$T(n)=O(f(n))\}$

为什么这样就能度量语言的时间复杂性呢？试想语言$A \in TIME(n^2)$，语言 ~~$B \in TIME(n^2)$~~ $B \notin TIME(n^2)$，那么在n较大时，即便用最聪明的办法判定B，时间开销也大于用最聪明办法判定A的开销，从而可以说：B确实比A有着更高的时间复杂性。

e.g. 设$B := \{0^n1^n \mid n \in \mathbb{N}\}$，$A := \{0^n \mid n \in \mathbb{N}\}$

我们可用笨办法来判定B:

1° 每找到一个0，就去右侧匹配一个1，再折返

2° 直至0与1匹配完毕。若恰好个数相等，则接纳；否则拒绝。

显然其时间复杂度$T(n)=O(n^2)$，从而$B \in TIME(n^2)$.

我们也可用聪明办法来判定B:

1° 重复以下步骤直至0或1耗尽

(1) 将0减半（$\cancel{0}\,0\,\cancel{0}\,0\ldots$）

(2) 将1减半（$\cancel{1}\,1\,\cancel{1}\,1\ldots$）

(3) 若0与1个数奇偶性相同则继续，否则拒绝.

经简单分析可知其时间复杂度 $T(n)=O(n\log n)$，从而 $B\in TIME(n\log n)$。可以证明这是最快的方法。

由此可见不同方法~~识别~~判定同一门语言的时间复杂度是不同的；时间复杂度只是具体某种方法的效率度量。

再来看 A。显然可以找到一种时间复杂度为 $O(n)$ 的机器来判定 A。当然，也可以设计出某台更慢的机器来判定 A，比方说先原地打转 $O(n^3)$ 步再干正事。

在比较语言 A 与 B 的时间复杂性时，显然不能拿 B 的聪明方法去和 A 的笨方法比，否则将得出「A难于B」的荒谬结果。我们必须以 B 最聪明的方法 ($O(n\log n)$) 比上 A 中某方法 ($O(n)$)，若前者仍逊于后者，则确实证实了「B难于A」。这便是我们定义 TIME、k-TIME、NTIME 语言类的缘由。

下面，我们来找三者间的关系。

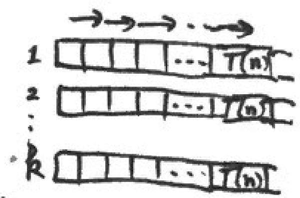
Theorem 1 若 $A\in k\text{-}TIME(f(n))$，则

$$A\in TIME(f^2(n)).$$

proof. $A\in k\text{-}TIME(f(n))$

$\Rightarrow$ $\exists$ k带TM M: $\mathcal{L}(M)=A$ 且 M 的时间复杂度 $T(n)=O(f(n))$

由于每一步转移只能移动一格，所以 M 运行时至多只能到达第 $T(n)$ 个存储单元。



下面，我们用 Chapter 2 Theorem2 的方法，构造 TM M' 以模拟 M 的行为。

由于 M 每转移一步，M' 均要折返于存储单元之间（至多 $kT(n)+c$ 个），故要花费 $O(f(n))$ 的步骤。又因为 M 共有 $T(n)=O(f(n))$ 步，故 M' 共有 $O(f(n))\cdot O(f(n))=O(f^2(n))$ 步

$\Rightarrow A\in TIME(f^2(n))$. ∎

Theorem 2 若 $A\in NTIME(f(n))$，则

$$A\in TIME(2^{O(f(n))})$$

proof. 思路与Theorem 1 类似，留作习题 ■

remark. 上述结论是单向的、松的。即，

$A\in TIME(f^2(n)) \not\Rightarrow A\in k\text{-}TIME(f(n))$;

$A\in k\text{-}TIME(f(n)) \not\Rightarrow A\notin TIME(o(f^2(n)))$.

remark. 不要妄想通过不断加大 k 而无限制地加速计算。这是因为，设判定A的最优TM之复杂度为 $T_1(n)$，又设 k 带TM M亦能判定A，复杂度为 $T_2(n)$，那么由Theorem 1 有 $T_1(n)=O(T_2^2(n))$。换言之，无论 k 多么大，$T_2(n)$ 的阶是有下界的。

接下来，我们给出一个重要的定义，它刻画了在现实中被认为「可判定」的一类问题——即它们的判定时间代价不至于太大。

def P语言类（P指多项式）

$$P := \bigcup_{k\in\mathbb{N}_0} TIME(n^k)$$

remark. 由于 $\bigcup_{k\in\mathbb{N}_0} O(n^k)$ = 所有多项式之集合，故P也可表述为

{语言A | ∃TM M: $\mathcal{L}(M)=A$ 且 M的时间复杂度 $T(n)$ 是一个多项式}

即P是那些可由TM在多项式时间内判定的语言。

用k-TIME换掉定义中的TIME，得到的语言类是相同的。这是因为Theorem 1告诉我们，凡是能用k带TM在多项式时间内解决的，用TM仍能在多项式时间内解决。可见P的定义在某种程度上与模型无关，从而具有较好的泛化能力。（再回头看先前的讨论，TM运行时间的定义虽则不完美，但至多差了常数倍，显然不影响包括TIME、k-TIME、NTIME及P的定义）

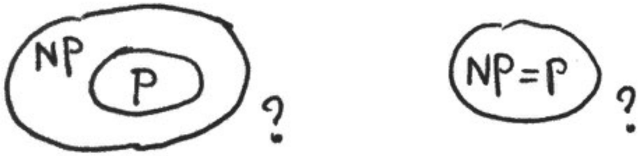
依着前面的思路，我们来定义NP语言类

def NP语言类（NP指非确定多项式）

$$NP := \bigcup_{k\in\mathbb{N}_0} NTIME(n^k)$$

前面我们说，P的定义在某种程度上与模型无关，那么把模型换作NTM（即把TIME设作NTIME），P还堪承受吗？

换言之，$P \overset{?}{=} NP$；可用NTM在多项式时间内解决的，是否也必可用TM在多项式时间内解决？此即当今困扰数学家的最大难题之一。



e.g. 前面讲过，$\{0^n1^n \mid n\in\mathbb{N}\} \in TIME(n^2) \subseteq P \subseteq NP$.

e.g. $\{\langle G\rangle \mid G$ 有Hamilton回路$\} =: HAMCYCLE$

我们可设计如下的NTM来判定之：

$N :=$ "输入$\langle G\rangle$时，

1° 解码出G含有个顶点

2° 不确定地产生分支，生成所有可能的元排列（每条分支生成一种）

3° 每条分支均检查生成的排列是否为G中的Hamilton回路。若是，则接纳，否则（该分支）拒绝。"

自然 $L(N) = HAMCYCLE$ 且 N的时间复杂度为 $T(n) = O(n^k)$（k是某常数），于是 $HAMCYCLE \in NP$。

至于是否能设计出TM来在多项式时间内判定HAMCYCLE，目前尚不得知。

一件事物从不同角度看，总是能带来新的感受。下面，我们用另一种方式来刻画NP语言类。

先作一个必要的定义。

def 验证器.

设A是一门语言，V是一台TM/多带TM/NTM. 若V满足

1° V的输入格式为$\langle w,e\rangle$

2° $\forall w\in\Sigma^*$，有

$w\in A \iff \exists e$使得V接纳$\langle w,e\rangle$

那么就称V是A的验证器。

直观而言，V的能力不及A的判定器，因为它要借助于外来的「通关文牒」e方可决策w与A的关系。只要有人能出示某个关键凭据e，那么V即断言$w\in A$；若关键凭据e找不到，则$w\notin A$。至于e怎么找、是否找得到，V一概不关心。

e.g. 我们可构造HAMCYCLE的验证器如下:

V:= "输入$\langle G,e\rangle$时，将e解码为一条路径p，检查p是否是G中历经所有顶点的合法回路"。

自然 $\langle G\rangle\in$ HAMCYCLE $\iff \exists$一个Hamilton回路编码e使得V接纳$\langle G,e\rangle$.

def 多项式时间验证器.

设A是一门语言，V是一台TM/多带TM/NTM. 若V满足

1° V的输入格式为$\langle w,e\rangle$

2° $\forall w\in\Sigma^*$，有

$w\in A \iff \exists e: |e|\le p(|w|)$ 使得V接纳$\langle w,e\rangle$，其中p是确定多项式

3° V的时间复杂度$T(n)=O($n的某个多项式$)$. 其中 $n=|\langle w,e\rangle|$

则称V是A的多项式时间验证器.

remark. 该定义把验证器的定义加强了，既限定了凭据e的长度上界，又限定了V的运行时间上界。显然前面例子中的V符合限定。

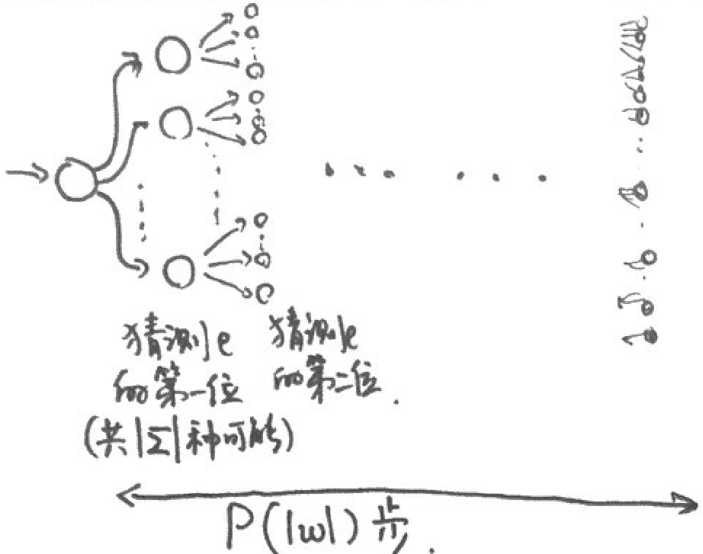
Theorem3 $A\in NP \iff$ 存在A的多项式时间验证器V.

proof. 1° "$\Leftarrow$":

已知有V，我们来构造一台NTM N，使N能够判定A。N:= "输入w时，

(1) 不确定地生成所有长度$\le p(|w|)$的凭据e.

(2) 每条分支均调用$V(\langle w,e\rangle)$，并依照V的结果来接纳/拒绝。"



易知 N 接纳 w $\Leftrightarrow$ w∈A。又因为「猜测」过程仅需 P(|w|)步，而调用 V(⟨w,e⟩)后仅需运行关于|⟨w,e⟩|的多项式步，亦即关于|w|的多项式步，从而 N 的运行时间上界为|w|的多项式。故 A∈NP。

2° "$\Rightarrow$":

因为 A∈NP，故存在NTM N：N能判定 A。从而 $\forall w\in\Sigma^*$，w∈A $\Leftrightarrow$ ∃N的一条计算流程，终态为接纳。依据此，很容易设计出 A 的多项式时间验证器 V:="输入⟨w,e⟩时，

(1) 将e解码成~~A的~~一条计算流程

(2) 检查该计算流程是否为N的合法计算流程，且终态是否为 q_{accept}。若两者皆满足，则接纳；否则拒绝。"

又由于一条计算流程的编码长度是|w|的多项式函数，所以 V 是 A 的多项式时间验证器。■

在上一章，我们曾引入序关系"≤"来表达两门语言 A 与 B 之间「相互转化」的可能性。当时我们说，如果能借用 B 的判定器来构造 A 的判定器，则 A≤B，它表明判定 A 不比判定 B 难。(仅就可判定层面而言)

把类似的思路搬到这儿来，我们便有如下定义：

def 多项式下的归约关系 $\le_p$.

$\le_p$:= { (A,B)∈全体语言类^2 | 能够通过 B 的、多项式时间的判定器 构造出 A 的、多项式时间的判定器 }

remark. 注意此处所谓「判定器」必须是TM或多带TM，而不能是NTM。

Proposition 4 设 $A \le_p B$。若 $B \in P$，则 $A \in P$。反过来，若 $A \notin P$，则 $B \notin P$。

Lemma 5 设A与B是两门语言。若存在变换 $f: \Sigma^* \to \Sigma^*$，f 可用TM在多项式时间内实现，且 $\forall w \in \Sigma^*$，

$$w \in A \iff f(w) \in B$$

那么 $A \le_p B$.

Lemma 6 设A与B是两门语言。若存在变换 $f: \Sigma^* \to \Sigma^*$，f 可用TM在多项式时间内实现，且 $\forall w \in \Sigma^*$，

$$w \in A \iff f(w) \notin B$$

那么 $A \le_p B$.

这些只是上一章相关命题的转写。我们仅证明Lemma 5。

proof. 设M是B的判定器，且M的时间复杂度是多项式。构造 M' := "输入w时，

1° 计算 $f(w)$

2° 调用 $M(f(w))$，并依M的结果接纳/拒绝。"

显然M'判定A。M'的第1步运行时间是 $|w|$ 的多项式，第2步运行时间是 $|f(w)|$ 的多项式。由于 $|f(w)|$ 也是 $|w|$ 的多项式（否则第1步根本就无法生成 $f(w)$），所以第2步运行时间也是 $|w|$ 的多项式。综上，M'是多项式时间内的判定器。■

与以前一样，纵使我们目前未知语言A与B是否在P中，研究 $A \overset{?}{\le}_p B$ 依旧是有意义的。我们将得以构建一条关系链，以备将来使用。

作为例子，我们在接下来的两个定理中演示Lemma 5, 6的应用。设

$3SAT := \{\langle \phi \rangle \mid \phi$ 是合取范式，且每个子句均为 $(\cdot \vee \cdot \vee \cdot)$ 的形式，且 ϕ 可满足$\}$

~~Theorem7~~

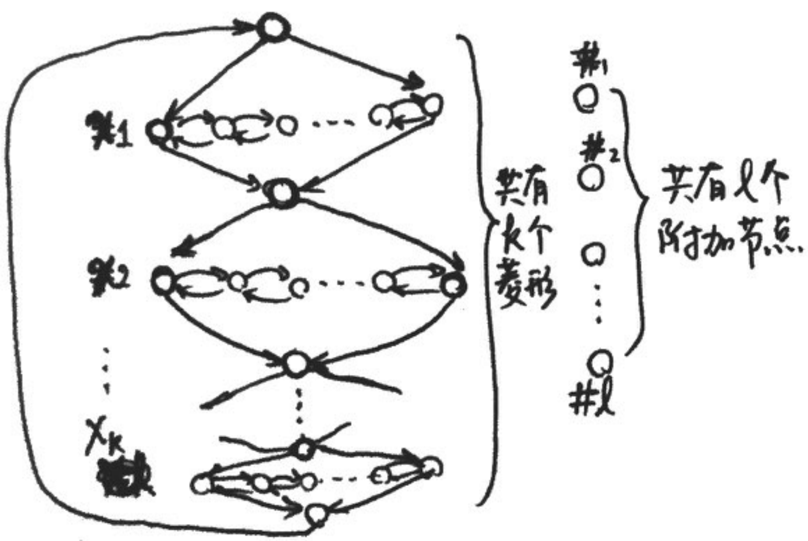
SUBSETSUM := $\{\langle S, s\rangle \mid S$ 是有穷整数列，$s \in \mathbb{Z}$，且存在 S 的子列 T：$\sum_{t\in T} t = s\}$

Theorem 7. $3SAT \le_p HAMCYCLE$.

proof. 我们将尝试构建Lemma5中的变换 f：

$\langle\phi\rangle \in 3SAT$	$\xLeftrightarrow{f(\langle\phi\rangle)=\langle G\rangle}$	$\langle G\rangle \in HAMCYCLE$
$\langle\phi\rangle \notin 3SAT$		$\langle G\rangle \notin HAMCYCLE$

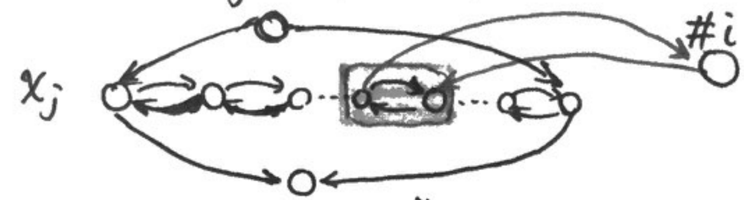
设 $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_l \vee b_l \vee c_l)$.
即 ϕ 中共有 l 个子句。又设 ϕ 中所有变量为 $\{x_1, x_2, \cdots, x_k\}$，$a_i, b_i, c_i \in \{x_1, x_2, \cdots, x_k\} \cup \{\overline{x_1}, \overline{x_2}, \cdots, \overline{x_k}\}$。我们构造图 G 如下：



其间仍有细节留待填充、完善。总的思路是：用一个菱形对应于 ϕ 中一个变量，用通行方向（从左往右/从右往左）对应于变量取1还是取0。

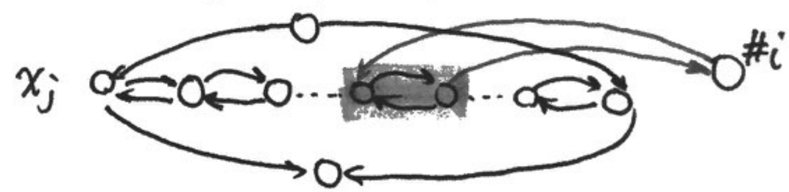
接下来，考虑每个子句 $(a_i \vee b_i \vee c_i)$。

(1) 若 $a_i = x_j$，那么如下连边：



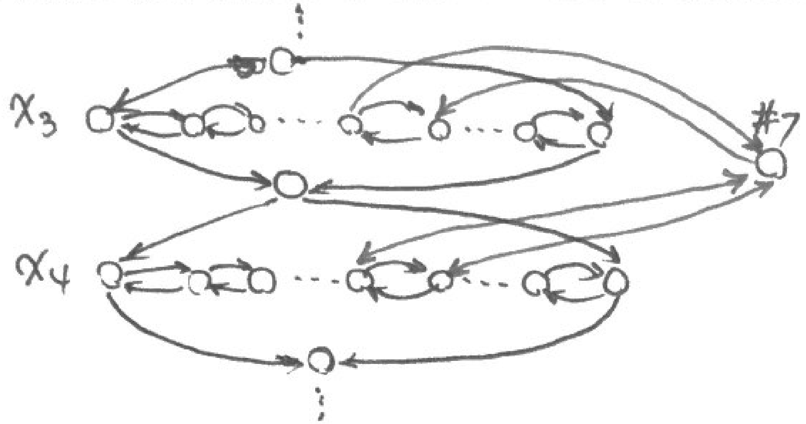
其中蓝色框部分是菱形 x_j 中专门给子句#i预留的「插槽」。注意插槽之间要求两两隔离（即○○○○○○○○）

(2) 若 $a_i = \overline{x_j}$，那么如下连边：



类似地，对 b_i、c_i 也相应地为#i节点添加边。最终，#i节点将有三条入边、三条出边，可能通向不同的菱形。

比如，子句(设第7号)为 $(x_3 \vee \overline{x_4} \vee x_4)$。那么其连边为



下面我们说明 ϕ 可满足 $\Longleftrightarrow$ G中有Hamilton回路。

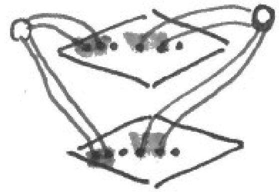
1° 若 ϕ 可满足，则存在一种对 $\{x_1,\dots,x_k\}$ 的真值指派，使 ϕ 中所有子句为真。也就是说，$\forall i$，$\{a_i, b_i, c_i\}$ 中至少有一个为1。不失一般性，可认为 $a_i=1$.

(1) 如果 $a_i = x_j$，则从左往右地行经第j个菱形，且在到达第i个插槽时绕路到#i节点。

(2) 如果 $a_i=\overline{x_j}$，则从右往左地行经第j个菱形，且在到达第i个插槽时绕路到#i节点。

不难看出，按上述走法必能遍历G中所有节点恰好一次且回到原点，故G中有Hamilton回路。

2° 若G中有Hamilton回路（不妨设从最上方出发），那么它必须由上至下地顺序遍历每个菱形，而不可能从某个菱形跳到附加节点，再跳到别的菱形。这是因为一旦跳到附加节点，



那么就无法通过该节点返回，而只能从别的附加节点返回。可是，即便返回，也毫无遍历所有节点的可能。（请注意插槽之间是有隔离的）

于是，该回路只得老老实实地走完一个菱形再走下一个菱形。既是如此，每个菱形必居「从左至右走」或「从右至左走」之一。前者发生时，则给对应变量赋1；否则赋0。不难看出此种指派必然满足 ϕ 中所有子句，因而 ϕ 可满足。

又由于以上变换f是可在多项式时间内实现的，故 $3SAT \le_p HAMCYCLE$. ■

Theorem 8 3SAT $\le_p$ SUBSETSUM.

proof. $\boxed{\begin{array}{c}\langle\phi\rangle\in 3SAT\\ \hline \langle\phi\rangle\notin 3SAT\end{array}} \overset{f(\langle\phi\rangle)=\langle S,s\rangle}{\Longleftrightarrow} \boxed{\begin{array}{c}\langle S,s\rangle\in SUBSETSUM\\ \hline \langle S,s\rangle\notin SUBSETSUM\end{array}}$

设 $\phi=(a_1\vee b_1\vee c_1)\wedge\cdots\wedge(a_l\vee b_l\vee c_l)$.
又设 ϕ 中所有变量为 $\{x_1,\dots,x_k\}$.
我们先来构造一张表以及 s.

$$\begin{array}{cc|cccc|cccc}
 & & x_1 & x_2 & \cdots & x_k & \#1 & \#2 & \cdots & \#l \\
\hline
\multirow{2}{*}{} & x_1 & 1 & & & & & & & \\
 & \bar{x}_1 & 1 & & & & & & & \\
 & x_2 & & 1 & & & & & & \\
 & \bar{x}_2 & & 1 & & & & * & & \\
 & \vdots & & & \ddots & & & & & \\
 & x_k & & & & 1 & & & & \\
 & \bar{x}_k & & & & 1 & & & & \\
\hline
\text{辅} & & & & & & 1 & & & \\
 & & & & & & 1 & & & \\
\text{助} & & & & & & & 1 & & \\
 & & & 0 & & & & 1 & & \\
\text{行} & & & & & & & & \ddots & \\
 & & & & & & & & & 1 \\
 & & & & & & & & & 1 \\
\end{array}$$

$s=1\,1\,1\cdots\cdots 1\;3\;3\cdots\cdots 3$

其中「*」区域依据每个子句中出现的文字来填写1或0。例如，

第2个子句为 $(x_1\vee\bar{x}_3\vee x_5)$，则标注了"#2"的一列对应 x_1、$\bar{x}_3$、x_5 的行填1，其余行填0。

表填完以后，把表中每一行读成十进制数，比如第一行为 $100\cdots01101$。那么，共计有 $2(k+l)$ 个数，由它们构成集合 S。不难看出 $\sum_{x\in S}x$ 将不会有进位。

显然，若想选出 S 的子集 T，使得 $\sum_{t\in T}t=s=\underbrace{11\cdots1}_{k}\underbrace{33\cdots3}_{l}$，则必然有

(1) 第 $2i$ 个数与第 $2i-1$ 个数不可同时选中

(2) $\forall j=1,\dots,l$，$\exists t\in T$：t 对应"$\#j$"的一列上含有1.

所以 ϕ 可满足 $\Longleftrightarrow$ 可选出 S 的子集 T 使 $\sum_{t\in T}t=s$ ■

$$\begin{tikzcd}
\boxed{3SAT} \arrow[r, "\le"] \arrow[dr, "\le"] \arrow[ddr, "\le"'] & \boxed{HAMCYCLE} \\
 & \boxed{SUBSETSUM} \\
 & \cdots
\end{tikzcd}$$

可见3SAT有些类似于上一章的 A_{TM}，是许多归约的原点。如果能证明3SAT $\notin P$，那么便有许多语言 $\notin P$。

那么，3SAT究竟是个难的语言，还是个容易的语言？事实上，它是NP中「最难的」一类语言。（从而能由它归约的问题如HAMCYCLE、SUBSETSUM也是NP中「最难的」一类语言）

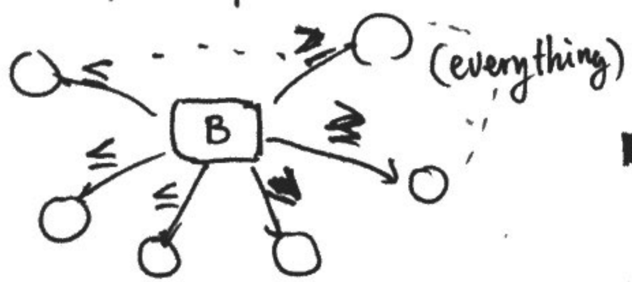
def NP完全（NPC）：

若语言 $B \in NP$，且 $\forall A \in NP$ 有 $A \le_p B$，则称B是NPC的。

（若B未必 $\in NP$，但 $\forall A \in NP$ 有 $A \le_p B$，则称B是NP难的）

Proposition 9　若B是NPC的，且 $B \in P$，则 $P = NP$.

proof.



■

Theorem 10 [Cook-Levin]

3SAT是NPC的。

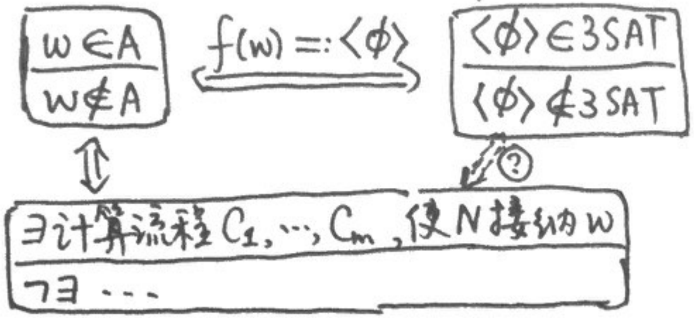
proof. 也就是要证明，$\forall A \in NP$，均有 $A \le_p 3SAT$。（$3SAT \in NP$ 是显然的，略去）。

注意这与Theorem 8, 9的方向相反。此前说的是3SAT不难于HAMCYCLE与SUBSETSUM，此处说的则是NP类中所有语言均不难于3SAT。二者并不矛盾，其自然推论是：NP类中所有语言均不难于3SAT、HAMCYCLE及SUBSETSUM。

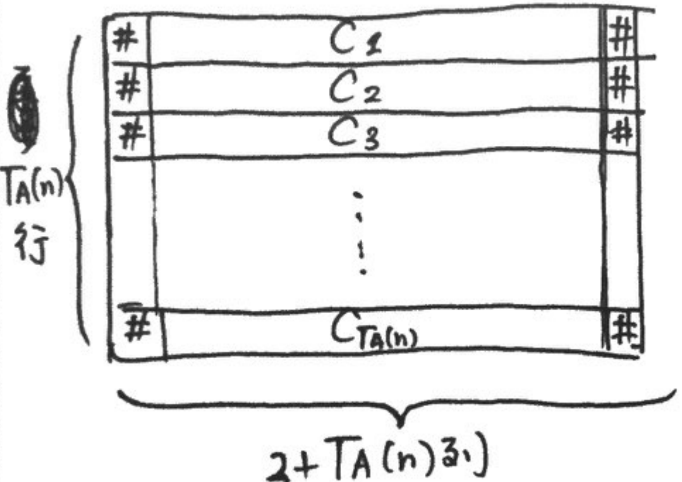
由于我们不确切清楚A的形式，所以证明时仅能依靠基础定义。因 $A \in NP$，故必存在一台NTM N_A，N_A 的时间复杂度为多项式，且 N_A 能判定A。给定了一个A，我们就有一台 N_A，就能用 N_A 的抽象性质去按下面所讲的方式证明 $A \le_p 3SAT$。

以下设N_A的时间复杂度为$T_A(n)=O(n^k)$.

我们希望找出Lemma 5中的变换f:



我们用以下形式呈现计算流程:



换句话说，每个合法的N的计算流程均可被一张$T_A(n)\times(2+T_A(n))$的表格表示。我们希望构造出一个ϕ，使得

ϕ可满足 $\Longleftrightarrow$ ∃一张表，它所表达的计算流程终态为接纳。

$$\phi := \phi_{cell}\wedge\phi_{start}\wedge\phi_{end}\wedge\phi_{move}$$

其中ϕ_{cell}用以约束表中每一格只能填一个字符，ϕ_{start}约束首行为起始格局，ϕ_{end}约束某行为接纳格局，ϕ_{move}约束行与行的推导关系。

定义变量 $\mathbb{1}_{ija} := \begin{cases}1, & \text{若第}i\text{行第}j\text{列字符为}a\\ 0, & \text{否则}\end{cases}$

$(i=1,2,\cdots,T_A(n)\ ;\ j=1,2,\cdots,T_A(n)+3\ ;$
$a\in \Gamma\cup Q\cup\{\#\}=:C\ )$

显然变量总数为$O(n^k)\cdot O(n^k)=O(n^{2k})$.

1° ϕ_{cell}.

$$\phi_{cell} := \bigwedge_{\substack{1\le i\le T_A(n)\\ 1\le j\le 3+T_A(n)}}\left[\left(\bigvee_{a\in C}\mathbb{1}_{ija}\right)\wedge\left(\bigwedge_{\substack{a,b\in C\\ a\ne b}}\left(\overline{\mathbb{1}_{ija}}\vee\overline{\mathbb{1}_{ijb}}\right)\right)\right]$$

意即：对每个格子(i,j)，必须填有C中的字符，而且不能填有2个或以上的字符。

2° ϕ_{start}

$$\phi_{start} := \mathbb{1}_{1,1,\#}\wedge\mathbb{1}_{1,2,q_0}\wedge\left(\mathbb{1}_{1,3,w_1}\wedge\cdots\wedge\mathbb{1}_{1,n+2,w_n}\right)\wedge\left(\mathbb{1}_{1,n+3,\sqcup}\wedge\cdots\wedge\mathbb{1}_{1,T_A(n)+2,\sqcup}\right)\wedge\mathbb{1}_{1,T_A(n)+3,\#}$$

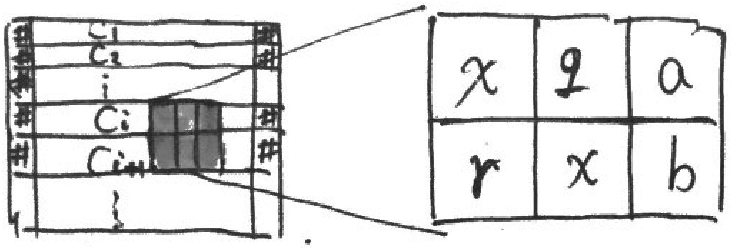
3° ϕ_{end}.

$$\phi_{end} := \bigvee_{\substack{1\le i\le T_A(n)\\ 1\le j\le 3+T_A(n)}} \mathbb{1}_{i,j,q_{accept}}$$

即在某个格局进入了 q_{accept}.

4° ϕ_{move}

这是最有难度的一部分，关键在于TM转移时改变的内容很有限，故可用「窗口」进行约束。观察表中所有的 2×3 窗口，假若每个窗口均「合法」，则格局之间的推导关系是正确的。



$\delta(q,a)$
$\Downarrow$
(r,b,L)

所谓的「合法」，即所开窗口的第一行有可能据 δ 转移至下一行。例如 $\delta(q,a)\ni(r,b,L)$，则右上图的窗口即为合法的。

严谨地表达出来，我们规定仅有以下几种情形的窗口是合法的：

s_1	s_2	s_3
t_1	t_2	t_3

(1) $s_1,s_2,s_3\notin Q$ 且 $s_1=t_1$, $s_2=t_2$, $s_3=t_3$

(2) $s_1\in Q$, $s_2,s_3\notin Q$，且满足以下两情形之一：

2.1 $t_1,t_2,t_3\notin Q$，$s_3=t_3$，且存在 $q\in Q$ 使得 $\delta(s_1,s_2)\ni(q,t_2,L)$

2.2 $t_1,t_3\notin Q$，$t_2\in Q$，$s_3=t_3$，且 $\delta(s_1,s_2)\ni(t_2,t_1,R)$.

(3) $s_2\in Q$，$s_1,s_3\notin Q$ （类似）

(4) $s_3\in Q$，$s_1,s_2\notin Q$ （类似）.

显然 (1)-(4) 均可用逻辑联结词来表达，而且长度是与 n 无关的常数。因而

$$\phi_{move} := \bigwedge_{\substack{1\le i\le T_A(n)-1\\ 1\le j\le T_A(n)+1}} (\text{左上角在}(i,j)\text{的窗口合法})$$

下面，我们来检查 ϕ_{cell}、ϕ_{start}、ϕ_{end}、ϕ_{move} 的长度与格式。

首先，ϕ_{cell} 是诸多子式的合取，而每一子式的长度是与 n 无关的常数。

故 ϕ_{cell} 写成合取范式 $((\cdots\vee\cdots)\wedge(\cdots\vee\cdots)\wedge\cdots\cdots)$ 以后子句数为 $O(n^{2k})$.

其次，ϕ_{start} 本就是子句数为 $O(n^{k})$ 的合取范式

再次，ϕ_{end} 本就是子句数为 1 的合取范式.

最后，ϕ_{move} 是子句数为 $O(n^{2k})$ 的合取范式。

总而言之，ϕ 可表为子句数为 $O(n^{2k})$ 的合取范式。当然，其中不少子句会有1/2/$\geq$4 个文字，不合3SAT的规定。为此，

1° 若子句形如 ~~$(a\vee b)$~~ (a)，则改为 $(a\vee a\vee a)$

2° 若子句形如 $(a\vee b)$，则改为 $(a\vee b\vee b)$

3° 若子句形如 $(a_1\vee a_2\vee\cdots\cdots\vee a_t)$，则改为 $(a_1\vee a_2\vee z_1)\wedge(\overline{z_1}\vee a_3\vee z_2)\wedge\cdots\wedge(\overline{z_{t-3}}\vee a_{t-1}\vee a_t)$.

(原因是 $a_1\vee a_2\vee\cdots\vee a_t$ 与 $(a_1\vee a_2\vee z)\wedge(\overline{z}\vee a_3\vee\cdots\vee a_t)$ 同可满足)

如此的改写只会引入少量新子句，故 ϕ 的长度仍然为 n 的多项式。■

remark. 梳理一下证明思路。

给定 $A\in NP$ $\xrightarrow[\text{定义}]{\text{根据}}$ 找到一台多项式时间判定A的NTM N_A

$\longrightarrow$ 确定 N_A 的时间复杂度 $T_A(n)$.

(以上是准备工作)

$\forall w\in\Sigma^*$，据 N_A 的行为和 w 来产生一个逻辑公式 ϕ，并保证 $w\in A\iff\phi\in 3SAT$.

于是 $A\leq_p 3SAT$.

这里似乎隐含一个深邃的哲学原理。根据定义，$\forall A\in NP$，必存在对应的 N_A，但是存在是否意味着可知、可得？这是值得思量的。只有我们确切地把 N_A 拿到手了，Theorem 10 的证明才成立。换言之，Theorem 10 的正确性建立在「凡存在的事物必可探明」的原理之上。