دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

نظریه زبان‌ها و ماشین‌ها

امیررضا آذری - ۹۹۱۰۱۰۸۷

پاسخ تمرین پنجم

۱. پرسش نخست

گزاره‌های زیر را نشان دهید.

۱. $n^k \in o(n^r)$ برای هر $k < r$. (۳ نمره)

۲. $n^{O(1)} = 2^{O(\log n)}$. (۳ نمره)

۳. $2^n \in o(3^n)$. (۴ نمره)

.۱

پاسخ.

.۱

می‌دانیم:

Let f and g be functions $f, g : \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = o(g(n))}$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

In other words, $f(n) = o(g(n))$ means that for any real number $c > 0$, a number n_0 exists, where $f(n) < c\,g(n)$ for all $n \geq n_0$.

با فرض $r > k$ داریم:

$$\lim_{n\to\infty} \frac{n^k}{n^r}$$
=
$$\lim_{n\to\infty} \frac{۱}{n^{r-k}}$$
= ۰

در نتیجه اثبات شد.

۲.

داریم:

- $\forall f(n) \in O(\log(n)) : \exists c, n_{\circ}\ s.t.\ \forall\ n \geq n_{\circ} :\ f(n) \leq c.\log(n) \ \rightarrow\ \frac{f(n)}{\log(n)} \leq c \ \rightarrow\ \frac{f(n)}{\log(n)} \in O(۱) \implies f(n) \in O(۱)\log(n)$
- $\forall f(n) \in O(۱)\log(n) : \exists c, n_{\circ}\ s.t.\ \forall\ n \geq n_{\circ} :\ f(n) \leq \log(n) * ۱ * c \ \rightarrow\ f(n) \leq c.\log(n) \implies f(n) \in O(\log(n))$

در نتیجه:

$$O(\log(n)) = O(۱)\log(n)$$

سپس:

$$O(\log(n)) = O(۱)\log(n) \implies ۲^{O(\log(n))} = ۲^{O(۱)\log(n)} = n^{O(۱)}$$

دقت کنید می‌دانیم که $n = ۲^{\log(n)}$.

۳.

ابتدا به راه سورس درس برای این سوال نگاه می‌کنیم:

(c)

True.

The statement $2^n = o(3^n)$ is valid, because the function 2^n runs shower than

[redacted]

the function 3^n.

Then, $2^n < 3^n$.

That is $f(n) < c \cdot g(n)$

Hence from the definition of small – *o* notation, $2^n = o(3^n)$ is true.

حال راه خود را ارائه می‌کنیم:
مانند بخش اول داریم:

$$\lim_{n\to\infty} \frac{۲^n}{۳^n} = \lim_{n\to\infty} (\frac{۲}{۳})^n = ۰$$

به بیان دیگر:

$$۲^n < c * ۳^n$$
$$\frac{۱}{c} < \frac{۳^n}{۲^n} = (۱/۵)^n \Leftrightarrow \log_{۱/۵}(\frac{۱}{c}) < n$$
$$\rightarrow n_۰ \leq \log_{۱/۵}(\frac{۱}{c})$$
$$\text{for } n > n_0, \text{ we have } n > n_0 > \log_{1.5}(\frac{1}{c})$$

۲. پرسش دوم

بگمارید که L یک زبان منظم است. آنگاه نشان دهید که $L \in DTIME(n)$. همچنین اگر L زبان مستقل از متن باشد، آنگاه $L \in DTIME(n^3)$ و $L \in NTIME(n)$ است. (۱۰ نمره)

پاسخ.

۱. $L \in DTIME(n)$

زبان L منظم است پس یک DFA دارد. حال ماشین تورینگ M را برای این زبان میسازیم. به نوعی که با خواندن ورودی، هد به راست رفته و روی استیت های DFA جا به جا می‌شود. بعد از پایان ورودی، اگر در accepting state بودیم، آن را اکسپت و در غیر این صورت آن را رجکت می‌نماییم. به دلیل اینکه ماشین M تنها یک بار روی رشته ورودی حرکت کرده است، پیچیدگی زمانی برابر $O(n)$ است و $L \in DTIME(n)$.

Let $t\colon \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, **TIME($t(n)$)**, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

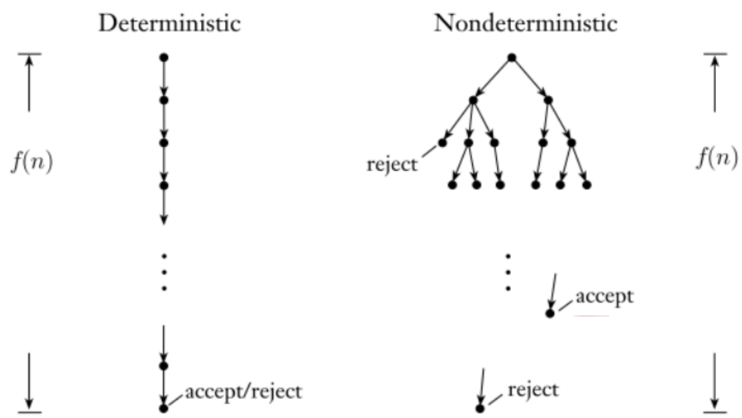
۲. $L \in DTIME(n^۳)$

حال در نظر می‌گیریم که L مستقل از متن می‌باشد. از دروس قبل الگوریتم cyk را به یاد داریم.
فرم نرمال چامسکی (CNF) زبان L را در نظر بگیرید. (که می‌دانیم در زمان چندجمله‌ای حاصل می‌شود.)
حال برای اجرای الگوریتم CYK، یک ماشین تورینگ دونواره را در نظر بگیرید. یک نوار را برای خواندن ورودی اختصاص می‌دهیم. از نوار دیگر برای نگه داشتن dynamic programming state استفاده می‌کنیم.
نوار دوم را در واقع با ترتیب lexicographic مجموعه ها قرار گرفته اند و با # جدا شده‌اند. برای هر متغیر نیز یک خانه در تمام مجموعه ها در نظر میگیریم و آن خانه را در صورت برابری، ۱ می‌کنیم. که در این حالت هر مجموعه تعداد $|V|$ خانه روی نوار دوم اشغال می‌کند. البته این موضوع را در همان اسلایدها هم بررسی کردیم و نشان دادیم که این الگوریتم، پیچیدگی زمانی برابر $O(n^۳)$ دارد. در نتیجه $L \in DTIME(n^۳)$.

۳. $L \in NTIME(n)$

ابتدا به تصویر زیر توجه کنید.

Let N be a nondeterministic Turing machine that is a decider. The ***running time*** of N is the function $f: \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n, as shown in the following figure.



حال همان فرم نرمال چامسکی یا CNF را در نظر بگیرید. می‌دانیم برای هر رشته w عضو این زبان، یک derivation به حداکثر طول $۲|w| - ۱$ وجود دارد. ($|w| - 1$ adding variable and $|w|$ converting to terminals.) حال ماشین تورینگ غیرقطعی M را برای زبان مورد نظر میسازیم که به طور غیر قطعی عمل میکند. طبق تصویر بالا و نکته ذکر شده داریم:

$$.O(۲|W|) = O(|w|) = O(n) \implies L \in NTIME(n)$$

۳. پرسش سوم

دو تعریف زیر از رده **NP** را در نگر بگیرید و برابری آن دو را نشان دهید. (۱۰ نمره)

□ زبان L در رده **NP** است اگر یک رایانه تورینگ تشخیص‌دهنده غیر قطعی زمان چند جمله‌ای برای آن موجود باشد.

□ زبان L در رده **NP** است اگر یک رابطه $R \subset \Sigma^* \times \Sigma^*$ به همراه یک چند جمله‌ای p موجود باشد که

$$L = \{x \in \Sigma^* | \exists y \in \Sigma^{p(n)} (x, y) \in R\}$$

پاسخ. برای حل این سوال از این لینک کمک گرفته شده است.
به ترتیب دو طرف را اثبات می‌نماییم:

طرف اول:

ماشین تورینگ غیرقطعی M برای زبان L را در نظر بگیرید. حال maximum branching factor را برابر b در نظر بگیرید. از آنجایی که تعداد استیت‌ها و الفبا محدود است، این مقدار نیز محدود است. همچنین فرض کنید پیچیدگی زمانی M برابر $f(m)$ باشد. برای رشته s_1 که توسط ماشین بیان شده پذیرفته می‌شود، رشته s_2 را جوری در نظر میگیریم و تعریف می‌کنیم که هر کاراکتر مشخص می‌کند که در راس فعلی درخت computation، به کدام راس در لایه بعدی برویم. چون رشته s_2 مسیری را مشخص می‌کند که M با طی کردن آن مسیر برای ورودی s_1، آن را تشخیص داده و می‌پذیرد و به accept میرود. در نتیجه برای هر رشته s_1 یک رشته s_2 وجود دارد که طول رشته s_2 حداکثر برابر $f(|s_1|)$ می‌باشد.

در نتیجه مطابق نوتیشن صورت سوال، برای هر شته رشته x رشته $y \in \Sigma^{p(|x|)}$ وجود دارد و آن را در R قرار می‌دهیم. بنابراین:

$$L = \{x \in \Sigma^* |\ \exists y \in \Sigma^{p(|x|)},\ (x, y) \in R\}$$

در نتیجه از تعریف اول به تعریف دوم رسیدیم (فراموش نکنید که زبان L در رده NP است.)

طرف دوم:

چندجمله‌ای p و رابطه R را داریم. برای اثبات برابری این تعریف با تعریف اول، یک ماشین تورینگ غیرقطعی را به شکلی میسازیم که رشته به طول $p(|x|)$ را حدس بزند و بعد از آن بررسی نماید که آیا (x, y) عضو R هستند یا خیر حال در این صورت اگر حدس، درست بود قبول می‌شود و اگر نبود، رد خواهد شد.

در نتیجه ماشین تورینگ ذکر شده، در زمان چندجمله‌ای، زبان گفته شده را تشخیص خواهد داد.

در نتیجه از تعریف دوم به تعریف اول رسیدیم.

در واقع توجه داشته باشید ماشین تورینگ چندنواره غیرقطعی ساختیم که میدانیم با تک نواره معادل است.

برای تکمیل بحث و این سوال، به توضیحات زیر توجه فرمایید:

4.6.2 Polynomial relations and their projections

Let Σ be an alphabet. For a relation $R \subseteq \Sigma^* \times \Sigma^*$, let $\exists R$ be its projection to the first component, i.e.,

$$\exists R \quad = \quad \{x : (\exists y)\, R(x, y)\}. \tag{54}$$

For simplicity, assume $|\Sigma| \geq 2$; we can then fix an encoding of a pair of words (x, y) by a single word $\alpha(x, y)$, such that $|\alpha(x, y)| \leq 2|x| + |y|$ (c.f. Exercise 2.5.1.1). We say that a relation R is in the class P if so is the corresponding language $\{\alpha(x, y) : R(x, y)\}$. We call a relation R *polynomial* if it satisfies the following two conditions:

$$R \quad \text{is in} \quad P \tag{55}$$

$$(\forall x, y)\, R(x, y) \quad \Rightarrow \quad |y| \leq p_R(|x|), \tag{56}$$

for some polynomial p_R.

The class NP consists of those languages L, which can be presented as $L = \exists R$, for some polynomial relation R.

In this context, we say that the relation R *verifies* language L, and we call y a *witness* for x, whenever $R(x, y)$. Not that, by definition, a witness, if any, must be "short" (of the length polynomially related to $|x|$), and the property of being a witness must be efficiently checkable.

Proposition 3. *A language L is in NP iff $L = L(M)$, for some non-deterministic Turing machine working in polynomial time.*

Proof. ($\Rightarrow$) Let $L = \exists R$, for a polynomial relation R. A non-deterministic machine recognizing L acts as follows. Given an input x, the machine uses existential states to generate a word y, $|y| \leq p_R(|x|)$ (c.f. (56)). Then it checks deterministically if $R(x, y)$ holds and accepts if it is the case.

($\Leftarrow$) Let $L = L(M)$, for a non-deterministic machine M working in time $p(n)$, for some polynomial p. Let R consist of the pairs (x, y), where y encodes a sequence of $\leq p(|x|)$ transitions of M, such that if M follows these transitions then it accepts x. By construction, R is polynomial and $L = \exists R$. □

۴. **پرسش چهارم**

رایانه‌های تورینگ استاندارد رده‌ای از رایانه‌های تورینگ هستند که تنها یک نوار دارند و الفبا نوار آنها $\{0,1,\Delta\}$ است. برای هر دسته از رایانه‌های تورینگ زیر یک تبدیل کارا [۱] به یک رایانه تورینگ استاندارد بیاورید. در واقع بایستی نشان دهید که برای هر رایانه با پیچیدگی محاسباتی $T(n)$ می‌توان یک رایانه تورینگ استاندارد با پیچیدگی محاسباتی $p(T(n))$ آورد.

۱. رایانه تورینگ تک نواره با الفبا نوار دلخواه Γ.

۲. رایانه تورینگ دو نواره با الفبا نوار $\{0,1,\Delta\}$.

۳. رایانه تورینگ تک نواره با الفبا $\{0,1,\Delta\}$ که دو سر [۲] دارد.

پاسخ. برای حل این سوال از لینک‌های یک و دو کمک می‌گیریم.

۱.

برای این بخش الفبای روی tape که همان Γ است را به ۰ و ۱ انکود می‌کنیم. با استفاده از Δ بین انها فاصله میگذاریم. حال برای انکود کردن هر یک از الفبای ورودی tape ماشین تورینگ به ماشین تورینگ استاندارد به اندازه هر سیمبل در تورینگ ماشین با یک رشته به طول $q=\log_2(m)$ انکود می‌شود.بنابراین برای نوشتن و خواندن روی tape در ماشین استاندارد، q برابر پیچیدگی در ماشین قبلی را داریم. پس پیچیدگی در ماشین استاندارد برابر است با: $qT(n)=O(qT(n))=p(T(n))$.
که تبدیل گفته شده به دست آمد.
در واقع به طور خلاصه و شفاف‌تر، اگر $|\Gamma|=n$، هر کاراکتر را با $\log_2(n)$ بیت کد میکنیم. بین هر دو کاراکتر نیز Δ می‌گذاریم. برای مشخص کردن پایان رشته نیز ۲ تا Δ می‌گذاریم. حال برای شبیه‌سازی، هر حرکت راست یا چپ در تورینگ اول، معادل $\log_2(n)$ بیت حرکت راست یا چپ خواهد بود. همچنین برای تغییر یک کاراکتر روی tape، معادل تغییر دادن همه $\log_2(n)$ بیت می‌باشد.
در نتیجه اگر پیچیدگی ماشین اولیه برابر $T(n)$ باشد، پیچیدگی ماشین استاندارد برابر $O(\log_2(n)T(n))$ است که کارا بودن اثبات می‌شود.

۲.

برای این بخش، طبق اسلایدها می‌دانیم هر ماشین تورینگ ۲ نواره را میتوان در زمان چندجمله‌ای به ماشین تورینگ تک نواره معادل تبدیل کرد. حال یک ماشین تورینگ تک نواره مانند بخش ۱ خواهیم داشت. بنابراین کاراست.
برای توضیح درست و کامل‌تر داریم:
میدانیم که برای هر ماشین تورینگ دو نواره با پیچیدگی زمانی $T(n)$، ماشین تورینگ تک‌نواره با پیچیدگی زمانی $O(T^2(n))$ وجود دارد که معادل هستند. بنابراین ابتدا ماشین دو نواره را به ماشین تک نواره تبدیل میکنیم . تنها ممکن است الفبا لزوما همان الفبای گفته شده نباشد. حال از این بخش به بعد مانند بخش اول این سوال عمل می‌کنیم و کارا بودن اثبات می‌شود.

۳.

طبق اسلایدهای درس می‌دانیم هر ماشین تورینگ با ۲ سر را می‌توان به ماشین تورینگ ۲ نواره معادل در زمان چندجمله‌ای تبدیل کرد. یک کپی از ورودی میگیریم و ان را در نوار دوم قرار میدهیم. باید محتویات این دو نوار را یکسان نگه داریم. اگر سر نوار اول در خانه k بود، در صورت تغییر خانه، سر نوار دوم به k میرود و بعد انجام تغییر به جایگاه قبلی باز خواهد گشت. (برای هر دو نوار برقرار است.)
هر عملیات تغییر خانه توسط یکی از سرهای ماشین اولیه، $O(T(n))$ زمان خواهد برد. در نتیجه پیچیدگی زمانی ماشین دو نواره برابر $O(T^{۲}(n))$ خواهد شدو تبدیل چندجمله‌ای است. حال از اینجا به بعد مطابق بخش قبل عمل خواهیم کرد. کارا بودن این بخش هم اثبات می‌شود.

۵. پرسش پنجم

برای مسئله‌های تصمیم‌گیری [۳] زیر یک زبان صوری بیاورید و سپس نشان دهید که در **NP-complete** هستند.

۱. آیا می‌توان راس‌های گراف G را با k رنگ رنگ کرد به‌گونه‌ای که هیچ دو راس همسایه هم‌رنگ نباشند. (۲ نمره)

۲. بگمارید که $S_1, \ldots, S_k$ مجموعه‌هایی متناهی هستند و $s = \cup S_i$. آیا یک زیرمجموعه $J \subset \{1, 2, \ldots, k\}$ موجود است که به ازای هر $i, j \in J$ متمایز، $S_i \cap S_j = \emptyset$ و $\cup_{i \in J} S_i = S$ باشد. (۴ نمره)

۳. بگمارید که $S = \{s_1, s_2, \ldots, s_n\}$ زیرمجموعه‌ای از شماره‌های صحیح و M یک شماره صحیح باشد. آیا زیر مجموعه $J \subset \{1, 2, \ldots, n\}$ موجود است که $\sum_{i \in J} a_i = M$ باشد. (۴ نمره)

پاسخ.

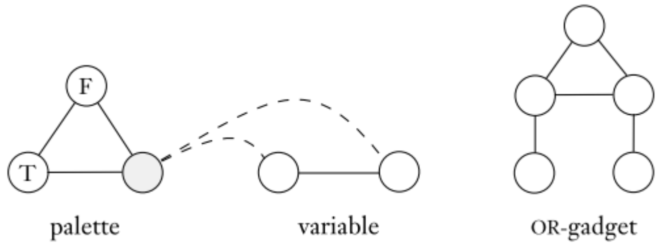
۱.

در ابتدا راه سورس برای سوال مشابه را نشان می‌دهیم:

7.29 A ***coloring*** of a graph is an assignment of colors to its nodes so that no two adjacent nodes are assigned the same color. Let

$$3COLOR = \{\langle G \rangle \mid G \text{ is colorable with 3 colors}\}.$$

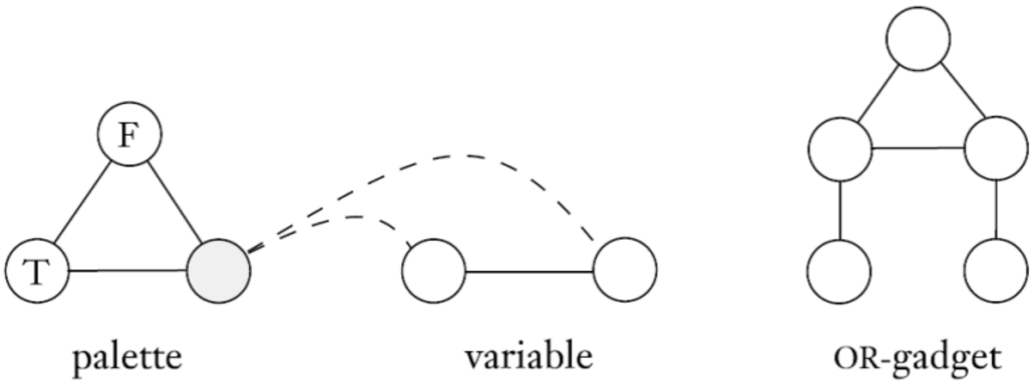
Show that *3COLOR* is NP-complete. (Hint: Use the following three subgraphs.)



A ***coloring*** of a graph is an assignment of colors to its nodes so that no two adjacent nodes are assigned the same color. Let

$$3COLOR = \{\langle G \rangle \mid G \text{ is colorable with 3 colors}\}.$$

Show that *3COLOR* is NP-complete. (Hint: Use the following three subgraphs.)



Step-by-step solution

Step 1 of 2

***NP* – complete:**

A language *B* is *NP* – complete if it satisfies following two conditions:

1. *B* is in *NP*

2. Every *A* in *NP* is polynomial time reducible to *B*.

Step 2 of 2

1.3 COLOR is in *NP* because a coloring can be verified in polynomial time.

2. $3SAT \leq_p 3COLOR$:

• " $3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable } 3cnf - \text{formula}\}$ " and "$3cnf$ -formula is the one in which all the clauses have three literals"

• Let $\phi = c_1 \wedge c_2 \wedge \ldots \wedge c_l$ be a $3cnf$ formula over the variable $x_1, \ldots, x_n$.

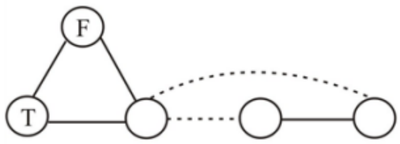
• To build a graph *G* with $2n + 6l + 3$ nodes, containing a variable gadget for each variable x_i, one clause gadget for each clause and one palette gadget as follows.

• Label the nodes of the palette gadget *T*, *F* and *R*.

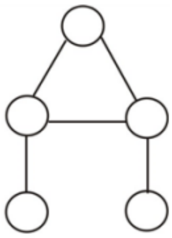
• Label the node since each variable gadget + and – and cannot reach to the *R* node in the palette gadget.

• For each clause, create a gadget.

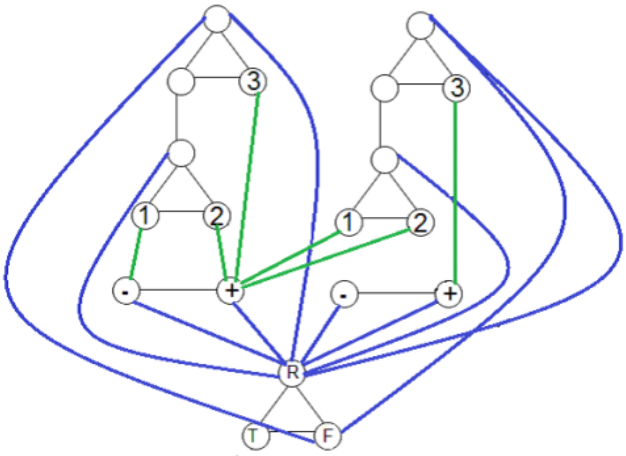
• Given three sub graphs.



Palette



- Connect the *F* and *R* nodes to the top of the clause gadget in the palette.
- Also, connect the top of its bottom triangle to the *R* node.
- For every clause c_j, connect the $i^{th}(1 \le i \le 3)$ bottom node of its clause gadget to the literal node that appears in its i^{th} location.
- An example is shown below.



- To show that the construction is correct, we first demonstrate that if ϕ is satisfiable, the graph is 3- colored.
- The three colors are called T, F and R.
- Color the palette with its labels.
- For each variable, color the + node T and – node F if the variable is true in a satisfying assignment: otherwise reverse the colors.
- Because each clause has one True literal in the assignment, we can color the nodes of that clause so that the node connected to the F node in the palette is not colored F.
- Hence we have proper 3-coloring.
- Similarly, if we are given a 3-coloring, we can obtain a satisfying assignment by taking the colors assigned to the + nodes of each variable.
- Observe that neither node of the variable gadget can be can be colored R, because all variable nodes are connected to the R node in the palette.
- Furthermore, if both bottom nodes of the clause gadget are colored F, the top node must be colored F, and hence, each clause must contain a true literal.

Hence, $3\text{SAT} \le_P 3$COLOR

Therefore, from (1) and (2) 3 COLOR is NP- complete.

همچنین از این لینک و این لینک نیز کمک گرفته‌ایم. حال خودمان به طور مختصر شرح می‌دهیم :)).

زبان صوری:

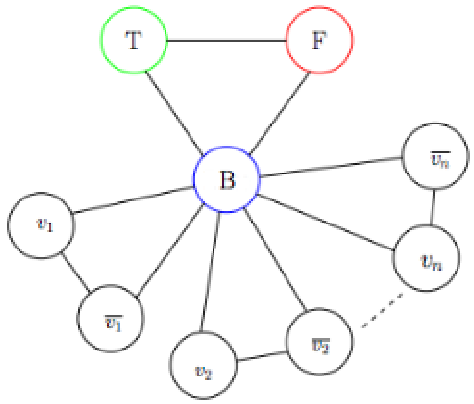
$$L = \{< G, k > \mid \text{G is a graph of size n and } \exists c_1, \ldots, c_n \in \{1, \ldots, k\} \text{ such that } \forall i, j \in 1, \ldots, n : (v_i, v_j) \in E(G) \longrightarrow c_i \neq c_j\}$$

شرط نوشته شده همان رنگK پذیری در گراف است. حال به اثبات می‌پردازیم:
یک verifire مانند v برای L میسازیم.

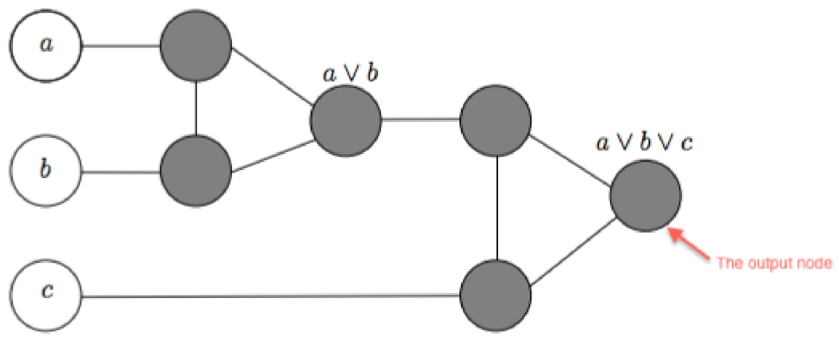
بدینگونه عمل میکند که به ازای هر یال در G چک میکند که رنگ ۲ سر یال متفاوت باشند. همچنین تعداد رنگ‌ها ماکسیمم برابر K باشد. اگر n راس داشته باشیم، $O(n^۲)$ یال داریم که نتیجه میشود verifire، چندجمله‌ای است.

حال برای اینکه نشان دهیم این مسئله NP-complete است، مسئله 3-sat را به آن کاهش می‌دهیم. این کار را به کاهش به زیرمجموعه‌ی سوالمان یعنی 3 coloring انجام میدهیم.

ابتدا یک مثلث با سه راس T,F,B(true, false, base) میسازیم. هر ست نشاندهنده لیبل ها برای vertices است. از انجا که مثلث به ۳ رنگ برای رنگ امیزی نیاز دارد، به ۳ رنگ برای رنگ امیزی G نیاز داریم. حال دو راس v_i و $\bar{v}_i$ اضافه میکنیم. (برای هر lateral x_i در مسئله $sat - ۳$). حال برای هر i، یک مثلث با راس های B، v_i، $\bar{v}_i$ میسازیم. این کار باعث میشود رنگ متفاوتی با B داشته باشند. شکل زیر را ببینید:

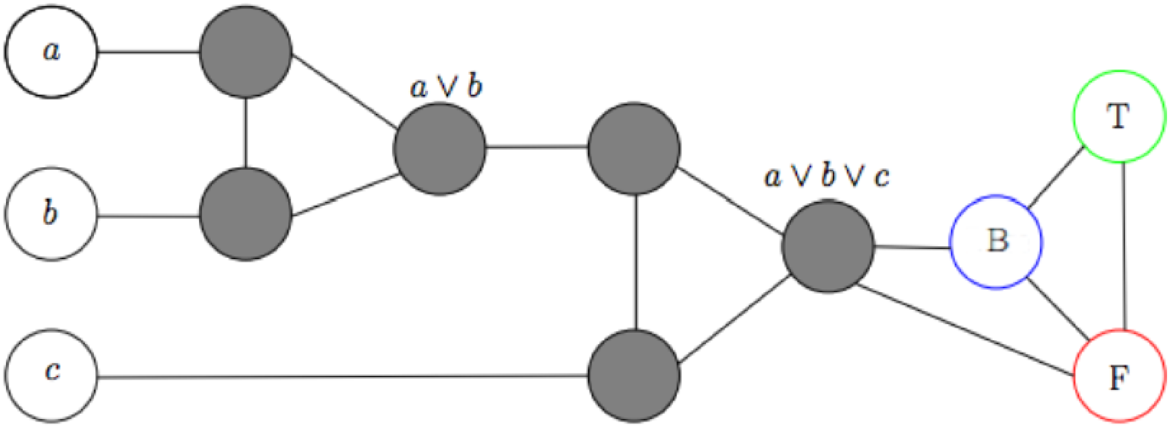


حال به سراغ کلاز میرویم. برای کلاز $C_i = a \vee b \vee c$ میخواهیم خروجی برابر T باشد اگر C_i satisfied شده باشد و در غیر این صورت برابر F باشد. شکل زیر را ببینید:



نود $a \vee b$ خروجی $a \vee b$ را کپچر می‌کند و procedure یکسان برای $(a \vee b) \vee c$ تکرار می‌کنیم.

اگر تمامی انها به F شدند آنگاه نود خورجی نیز باید F باشد. اگر یکی از انها T باشد یعنی یک رنگ امیزی درست وجود دارد پس خروجی نیز باید T باشد. برای اطمینان اینکه خروجی ما رنگ امیزی درستی میدهد، از خروجی به F یک یال میکشیم تا نتواند F بشود.



حال اگر مسئله ۳-sat، به نوعی satisfiable باشد، اگر x_i True باشد، ما v_i را با T و $\bar{v}_i$ را با F رنگ میکنیم.

پس هر کلاز satisfiable است، بنابراین حداقل یکی از a، b، c برای هر عبارت True هستند، به این معنی که با توجه به ساختار ما، ابزار ۳ رنگ می شود و گره خروجی به رنگ T است. برای تکمیل کامل این قسمت به راه زیر که به عنوان سورس ذکر شد، توجه بفرمایید:

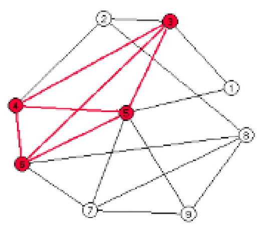
We next show that 3-colouring is NP-complete. What's the **colouring problem** on graphs?

Given a graph $G(V, E)$, the colouring problem asks for an assignment of k colours to the vertices $c : V \to \{1, 2, ..., k\}$. We say that a colouring is **proper** if adjacent vertices receives different colours: $\forall(u, v) \in E : c(u) \neq c(v)$.

The **minimum colouring problem** asks for the smallest k to properly colour G. The k**-colouring problem** asks whether G can be properly coloured using at most k colours.

We call the subset of vertices that receive the same colour a **colour class**. It is easy to see that every colour class is an independent set (yeah?). Notice that we've encountered the colouring problem before! Recall in Assignment 1, we wanted to send the minimum number of journalists to cover all the world cup games. Well the subset of games a journalist got is a colour class! We were able to solve this optimization problem in $\mathcal{O}(|V| \log |V|)$ time, so how can we claim that the k-colouring decision problem is NP-complete?

It is the same reasoning for the Independent Set problem. One way to deal with NP-completeness is to restrict the problem to subsets of the input (in this assignment, we restricted "arbitrary" graphs to interval graphs). In this lecture we will show that the colouring problem on arbitrary graphs becomes NP-complete even for $k = 3$! Crazy! No? Think about it: One easy to rule out a 3-colouring is to check if G has a clique of size 4, like in the example below[1]:



How hard is it to check for a clique of size at least $k = 4$? We just showed that CLIQUE is NP-complete! No luck! Even with small subgraphs (4 vertices :( ). Without further ado, let's prove our theorem:

Theorem 1. *3-COLOURING is NP-complete.*

Where:

3-COLOURING: Given a graph $G(V, E)$, return 1 if and only if there is a proper colouring of G using at most 3 colours.

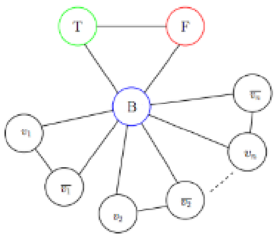
Proof. To show the problem is in NP, our verifier takes a graph $G(V, E)$ and a colouring c, and checks in $\mathcal{O}(n^2)$ time whether c is a proper coloring by checking if the end points of every edge $e \in E$ have different colours.

[1]All the pictures are stolen from Google Images and UIUC's algo course.

To show that 3-COLOURING is NP-hard, we give a polytime reduction from 3-SAT to 3-COLOURING. That is, given an instance ϕ of 3-SAT, we will construct an instance of 3-COLOURING (i.e. a graph $G(V, E)$) where G is 3-colourable iff ϕ is satisfiable.

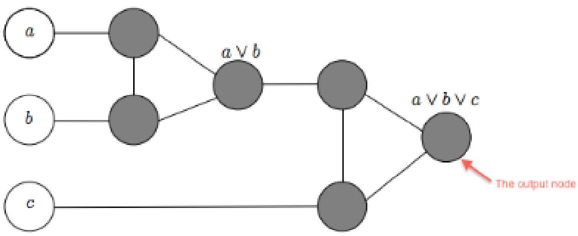
Let ϕ be a 3-SAT instance and $C_1, C_2, ..., C_m$ be the clauses of ϕ defined over the variables $\{x_1, x_2, ..., x_n\}$. The graph $G(V, E)$ that we will construct needs to capture two things: 1. somehow establish the truth assignment for $x_1, x_2, ..., x_n$ via the colors of the vertices of G; and 2. somehow capture the satisfiability of every clause C_i in ϕ.

To achieve these two goals, we will first create a triangle in G with three vertices $\{T, F, B\}$ where T stands for True, F for False and B for Base. Think of $\{T, F, B\}$ as the set of colours we will use to colour (label) the vertices of G. Since this triangle is part of G, we already need 3 colours to colour G. We next add two vertices $v_i, \overline{v}_i$ for every literal x_i and create a triangle $B, v_i, \overline{v}_i$ for every $(v_i, \overline{v}_i)$ pair, as shown below:



Notice that so far, this construction captures the truth assignment of the literals. Since if G is 3-colourable, then either v_i or $\overline{v}_i$ gets the colour T, and we just interpret this as the truth assignment to v_i. Now we just need to add constraints (edges? extra vertices?) to G to capture the satisfiability of the clauses of ϕ. To do so, we introduce the **Clause Satisfiability Gadget**, a.k.a the OR-gadget.

For a clause $C_i = (a \vee b \vee c)$, we need to express the OR of its literals using our colours $\{T, F, B\}$. We achieve this by creating a small *gadget* graph that we connect to the literals of the clause. The OR-gadget is constructed as follows:



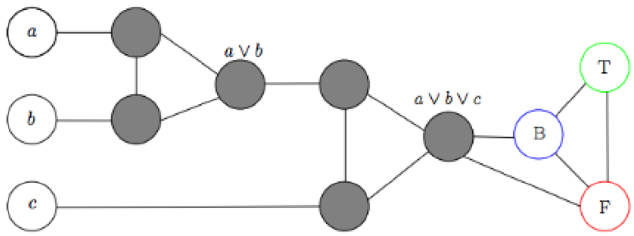
But what is it actually doing? You can think of this gadget graph as a circuit whose output is the node labeled $a \vee b \vee c$. We basically want this node to be coloured T if C_i is satisfied and F otherwise. Notice that this is a two step construction: The node labelled $a \vee b$ captures the output of $(a \vee b)$ and we repeat the same operation for $((a \vee b) \vee c)$.

If you play around with some assignments to a, b, c, you will notice that the gadget satisfies the following properties:

1. If a, b, c are all coloured F in a 3-colouring, then the output node of the OR-gadget **has** to be coloured F. Thus capturing the unsatisfiability of the clause $C_i = (a \vee b \vee c)$.

2. If one of a, b, c is coloured T, then **there exists a** valid 3-colouring of the OR-gadget where the output node is coloured T. Thus again capturing the satisfiability of the clause.

We're almost done our construction. Once we add the OR-gadget of every C_i in ϕ, we connect the output node of every gadget to the Base vertex **and** to the False vertex of the initial triangle, as follows:



Done :) Now we prove that our initial 3-SAT instance ϕ is satisfiable if and only the graph G as constructed above is 3-colourable.

Suppose ϕ is satisfiable and let $(x_1^*, x_2^*, ..., x_n^*)$ be the satisfying assignment. If x_i^* is assigned True, we colour v_i with T and $\overline{v}_i$ with F (recall they're connected to the Base vertex, coloured B, so this is a valid colouring). Since ϕ is satisfiable, every clause $C_i = (a \vee b \vee c)$ must be satisfiable, i.e. at least of a, b, c is set to True. By the second property of the OR-gadget, we know that the gadget corresponding to C_i can be 3-coloured so that the output node is coloured T. And because the output node is adjacent to the False and Base vertices of the initial triangle only, this is a proper 3-colouring.

Conversely, suppose G is 3-colourable. We construct an assignment of the literals of ϕ by setting x_i to True if v_i is coloured T and vice versa. Now suppose this assignment is not a satisfying assignment to ϕ, then this means there exists at least one clause $C_i = (a \vee b \vee c)$ that was not satisfiable. That is, all of a, b, c were set to False. But if this is the case, then the output node of corresponding OR-gadget of C_i must be coloured F (by property (1)). But this output node is adjacent to the False vertex coloured F; thus contradicting the 3-colourability of G!

To conclude, weve shown that 3-COLOURING is in NP and that it is NP-hard by giving a reduction from 3-SAT. Therefore 3-COLOURING is NP-complete. $\square$

۲.

این مسئله همان مسئله exact-cover است.

زبان صوری:

$$L = \{< S, \{S_1, \dots, S_k\} > \ | \bigcup_{s \in \{S_1,\dots,S_k\}} s = S \text{ and for some } J \subset \{1, \dots, k\} \text{ we have } \cup_{j \in J} S_j = S,\ \forall i, j\ \in\ J : S_i \cap S_j = \emptyset\}$$

ابتدا نشان می‌دهیم مسئله NP است. یک وریفایر چندجمله‌ای به عنوان certificate تعدادی اندیس ورودی میگیرد. ابتدا چک میکند که اندیس بین ۱ تا k باشند. سپس روی هر S_i حرکت میکند و هر عضوی را که از ان میبند، علامت میزند. اگر تمام اعضای S دقیقا یکبار علامت خورده باشند، ورودی را میپذیرد. این عملیات چندجمله ای است بنابراین $L \in NP$.

برای NP-complete، مسئله sat را به آن کاهش می‌دهیم. ورودی σ را به صورت conjuctive normal form برای sat در نظر میگیریم. فرض کنید σ شامل nتا متغیر x_i و l کلاز باشد. کلاز c_j را به سورت زیر نمایش میدهیم:

$$c_j = L_{j_1} \vee \dots \vee L_{j_{m_j}}$$

در این نمایش، m_j تعداد لیترالهای کلاز جی ام است. حال S را به شکل زیر در نظر بگیرید:

$$S = \{x_i \mid ۱ \leq i \leq n\} \cup \{c_j \mid ۱ \leq j \leq l\} \cup \{p_{jk} \mid ۱ \leq j \leq l,\ ۱ \leq k \leq m_j\}$$

حال مجموعه S_iها:

- $\{p_{jk}\}$ for every p_{jk}
- : $T_i = \{x_i\} \cup \{p_{jk} | L_{jk} = \bar{x}_i$ for every $x_i\}$
- : $F_i = \{x_i\} \cup \{p_{jk} | L_{jk} = x_i$ for every $x_i\}$
- $\{c_j, p_{jk}\}$: for every c_j and every p_{jk}

حال ادعا میکنیم که σ به نوعی satisfiable است اگر و فقط اگر مجموعه S با S_iهای ساخته شده، exact-cover داشته باشد. اگر satisfiable بود، اگر $x_i = ۱$ انگاه T_i و در غیر این صورت، F_i را انتخاب میکنیم. به ازای هر c_j نیز یکی از مجموعه های $\{c_j, p_{jk}\}$ را برمیداریم که لیترال جایگاه kام آن برابر یک است. اگر T_i انتخاب شده باشد، تمام تکررهای صفرکننده x_i نیز انتخاب شده است. همین قضیه برای F_i نیز برقرار است. بنابراین با برداشتن T_iها و F_iها به شکل گفته شده، p_{jk}های مربوط به لیترالهای صفر انتخاب شدهاند. چونکه حداقل یک لیترال برابر ۱ دارند هم تمام c_j ها برداشته میشوند. حال مجموعه های $\{p_{jk}\}$ را طوری انتخاب میکنیم که هر p_{jk} نیز دقیقا یکبار تکرار شوند.بنابراین یک exact-cover برای S ساخته میشود.

حال اگر یک exact cover برای S داشته باشیم؛ از بین هر T_iو F_i، دقیقا یکی انتخاب شده است. اگر T_i انتخاب شده بود مقدار x_i را یک و در غیر این صورت ۰ میگذاریم. برای هر کلاز c_j نیز یکی از $\{c_j, p_{jk}\}$ها انتخاب شده است. اگر $L_{jk} = x_i$ باشد، در این صورت T_i انتخاب شده و x_i برابر یک است. مشابه همین برای $\bar{x}_i$ برقرار است و اگر F انتخاب شده باشد، صفر میشود. بنابراین تمام کلازها satisfy شدهاند و σ به نوعی satisfiable است.

بنابراین این زبان NP-complete است.

برای حل این سوال از یکی از بچهها و همچنین به طور کامل از این لینک و این لینک کمک گرفته شده است.

10.2 Proofs of $\mathcal{NP}$-Completeness

(1) **Exact Cover**

To prove that **Exact Cover** is $\mathcal{NP}$-complete, we reduce the **Satisfiability Problem** to it:

Satisfiability Problem $\leq_P$ Exact Cover

Given a set $F = \{C_1, \ldots, C_\ell\}$ of ℓ clauses constructed from n propositional variables $x_1, \ldots, x_n$, we must construct in polynomial time an instance $\tau(F) = (U, \mathcal{F})$ of **Exact Cover** such that F is satisfiable iff $\tau(F)$ has a solution.

Example 10.2. If

$$F = \{C_1 = (x_1 \vee \overline{x_2}),\ C_2 = (\overline{x_1} \vee x_2 \vee x_3),\ C_3 = (x_2),\\ C_4 = (\overline{x_2} \vee \overline{x_3})\},$$

then the universe U is given by

$$U = \{x_1, x_2, x_3, C_1, C_2, C_3, C_4, p_{11}, p_{12}, p_{21}, p_{22}, p_{23}, p_{31},\\ p_{41}, p_{42}\},$$

and the family $\mathcal{F}$ consists of the subsets

$$\begin{aligned}
&\{p_{11}\}, \{p_{12}\}, \{p_{21}\}, \{p_{22}\}, \{p_{23}\}, \{p_{31}\}, \{p_{41}\}, \{p_{42}\}\\
&T_{1,\mathbf{F}} = \{x_1, p_{11}\}\\
&T_{1,\mathbf{T}} = \{x_1, p_{21}\}\\
&T_{2,\mathbf{F}} = \{x_2, p_{22}, p_{31}\}\\
&T_{2,\mathbf{T}} = \{x_2, p_{12}, p_{41}\}\\
&T_{3,\mathbf{F}} = \{x_3, p_{23}\}\\
&T_{3,\mathbf{T}} = \{x_3, p_{42}\}\\
&\{C_1, p_{11}\}, \{C_1, p_{12}\}, \{C_2, p_{21}\}, \{C_2, p_{22}\}, \{C_2, p_{23}\},\\
&\{C_3, p_{31}\}, \{C_4, p_{41}\}, \{C_4, p_{42}\}.
\end{aligned}$$

It is easy to check that the set $\mathcal{C}$ consisting of the following subsets is an exact cover:

$T_{1,\mathbf{T}} = \{x_1, p_{21}\}, T_{2,\mathbf{T}} = \{x_2, p_{12}, p_{41}\}, T_{3,\mathbf{F}} = \{x_3, p_{23}\}$, $\{C_1, p_{11}\}, \{C_2, p_{22}\}, \{C_3, p_{31}\}, \{C_4, p_{42}\}$.

The general method to construct $(U, \mathcal{F})$ from $F = \{C_1, \ldots, C_\ell\}$ proceeds as follows. Say

$$C_j = (L_{j1} \vee \cdots \vee L_{jm_j})$$

is the jth clause in F, where L_{jk} denotes the kth literal in C_j and $m_j \geq 1$. The universe of $\tau(F)$ is the set

$$\begin{aligned} U = \{x_i \mid 1 \leq i \leq n\} \cup \{C_j \mid 1 \leq j \leq \ell\} \\ \cup \{p_{jk} \mid 1 \leq j \leq \ell,\ 1 \leq k \leq m_j\} \end{aligned}$$

where in the third set p_{jk} corresponds to the kth literal in C_j.

The following subsets are included in $\mathcal{F}$:

(a) There is a set $\{p_{jk}\}$ for every p_{jk}.

(b) For every boolean variable x_i, the following two sets are in $\mathcal{F}$:

$$T_{i,\mathbf{T}} = \{x_i\} \cup \{p_{jk} \mid L_{jk} = \overline{x_i}\}$$

which contains x_i and all negative occurrences of x_i, and

$$T_{i,\mathbf{F}} = \{x_i\} \cup \{p_{jk} \mid L_{jk} = x_i\}$$

which contains x_i and all its positive occurrences. Note carefully that $T_{i,\mathbf{T}}$ involves negative occurrences of x_i whereas $T_{i,\mathbf{F}}$ involves positive occurrences of x_i.

(c) For every clause C_j, the m_j sets $\{C_j, p_{jk}\}$ are in $\mathcal{F}$.

It remains to prove that F is satisfiable iff $\tau(F)$ has a solution.

We claim that if v is a truth assignement that satisfies F, then we can make an exact cover $\mathcal{C}$ as follows:

For each x_i, we put the subset $T_{i,\mathbf{T}}$ in $\mathcal{C}$ iff $v(x_i) = \mathbf{T}$, else we we put the subset $T_{i,\mathbf{F}}$ in $\mathcal{C}$ iff $v(x_i) = \mathbf{F}$.

Also, for every clause C_j, we put some subset $\{C_j, p_{jk}\}$ in $\mathcal{C}$ for a literal L_{jk} which is made true by v.

By construction of $T_{i,\mathbf{T}}$ and $T_{i,\mathbf{F}}$, this p_{jk} is not in any set in $\mathcal{C}$ selected so far. Since by hypothesis F is satisfiable, such a literal exists for every clause.

Having covered all x_i and C_j, we put a set $\{p_{jk}\}$ in $\mathcal{C}$ for every remaining p_{jk} which has not yet been covered by the sets already in $\mathcal{C}$.

Going back to Example 10.2, the truth assigment $v(x_1) = \mathbf{T}, v(x_2) = \mathbf{T}, v(x_3) = \mathbf{F}$ satisfies

$$\begin{aligned} F = \{&C_1 = (x_1 \vee \overline{x_2}),\ C_2 = (\overline{x_1} \vee x_2 \vee x_3),\ C_3 = (x_2),\\ &C_4 = (\overline{x_2} \vee \overline{x_3})\}, \end{aligned}$$

so we put

$$T_{1,\mathbf{T}} = \{x_1, p_{21}\}, T_{2,\mathbf{T}} = \{x_2, p_{12}, p_{41}\}, T_{3,\mathbf{F}} = \{x_3, p_{23}\},$$
$$\{C_1, p_{11}\}, \{C_2, p_{22}\}, \{C_3, p_{31}\}, \{C_4, p_{42}\}$$

in $\mathcal{C}$.

Conversely, if $\mathcal{C}$ is an exact cover of $\tau(F)$, we define a truth assigment as follows:

For every x_i, if $T_{i,\mathbf{T}}$ is in $\mathcal{C}$, then we set $v(x_i) = \mathbf{T}$, else if $T_{i,\mathbf{F}}$ is in $\mathcal{C}$, then we set $v(x_i) = \mathbf{F}$.

۳.

برای حل این سوال از این لینک کمک گرفته شده است. همچنین ایده سوال از یکی از بچه های کلاس پرسیده شده است.

زبان صوری:

$$L = \{\langle S, M\rangle \mid n \in \mathbb{N}, (a_1, \ldots, a_n, M) \in \mathbb{Z}^{n+1}, S = \{a_1, a_2, \ldots, a_n\}, \exists J \subset \{1, 2, \ldots, n\} \text{ s.t } \sum_{i \in J} a_i = M\}$$

ابتدا راه یکی از منابع را ببینیم:

The SUBSET SUM problem is as follows: given n non-negative integers $w_1, \ldots, w_n$ and a target sum W, the question is to decide if there is a subset $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i = W$. This is a very special case of the KNAPSACK problem: In the KNAPSACK problem, items also have values v_i, and the problem was to maximize $\sum_{i \in I} v_i$ subject to $\sum_{i \in I} w_i \leq W$. If we set $v_i = w_i$ for all i, SUBSET SUM is a special case of the KNAPSACK problem that we discussed when considering dynamic programming. In that section, we gave an algorithm for the problem that runs in time $O(nW)$. This algorithm works well when W isn't too large, but we note that this algorithm is not a polynomial time algorithm. To write down an integer W, we only need $\log W$ digits. In is natural to assume that all $w_i \leq W$, and so the input length is $(n+1)\log W$, and the running time of $O(nW)$ is not polynomial in this input length.

In this handout we show that, in fact, SUBSET SUM is NP-complete. First we show that SUBSET SUM is in NP.

Claim 1. *SUBSET SUM is in NP.*

Proof. Given a proposed set I, all we have to test if indeed $\sum_{i \in I} w_i = W$. Adding up at most n numbers, each of size W takes $O(n \log W)$ time, linear in the input size. □

To establish that SUBSET SUM is NP-complete we will prove that it is at least as hard asSAT.

Theorem 1. SAT $\leq$ SUBSET SUM.

Proof. To prove the claim we need to consider a formula Φ, an input to SAT, and transform it into an equivalent input to SUBSET SUM. Assume Φ has n variables $x_1, \ldots, x_n$, and m clauses $c_1, \ldots, c_m$, where clause c_j has k_j literals.

We will define our SUBSET SUM problem using a very large base B, so will write numbers as $\sum_{j=0}^{n+m} a_j B^j$, and we set the base B as $B = 2\max_j k_j$, which will make sure that additions among our numbers will never cause a carry.

Written in base B the digits $i = 1, \ldots, n$ will correspond to the n variables $x_1, \ldots, x_n$, and the goal of these digits will be to make sure that we set each variable to either true of false (and not both). We'll have two numbers w_i and w_{i+n} corresponding to the variable x_i being set true or false, and digit i will make sure that we use one of w_i and w_{i+n} in any solution. To so this, we set the ith digits of W, w_i and w_{i+n} to be 1, and set this digit in all other numbers to be 0.

The next m digits will correspond to the m clauses, and the goal digit $n+j$ is to make sure that the jth clause is satisfied by our setting of the variables.

The target value will be $W = \sum_{i=1}^{n} B^i + \sum_{j=1}^{m} k_j B^{n+j}$.

We start by defining $2n$ numbers, for of each of the literals x_i and $\overline{x}_i$. The digits $1, \ldots, n$ will make sure that any subset that sums to W will use only exactly 1 of the two numbers x_i and $\overline{x}_i$, and the the next m digits will aim to guarantee that each clause is satisfied. We will need a few additional numbers that we'll define later.

The number corresponding to literal x_i is as follows $w_i = B^i + \sum_{j: x_i \in c_j} B^{n+j}$, while the number corresponding to literal $\bar{x}_i$ is $w_{i+n} = B^i + \sum_{j: \bar{x}_i \in c_j} B^{n+j}$. If we add a set of n numbers

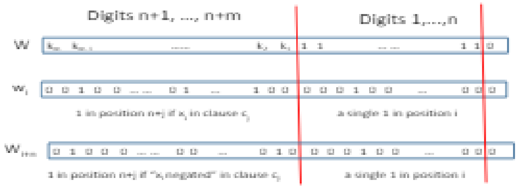


Figure 1: The total W and the numbers w_i and w_{i+n}.

corresponding to a satisfying truth assignment for Φ, we get a some of the form $\sum_{i=1}^{n} B^i + \sum_{j=1}^{m} b_j B^{n+j}$ where b_j is the number of literals true in clause c_j. Since this was a satisfying assignment, we must have $b_j \geq 1$.

As a final detail, we will add $k_j - 1$ copies of the number B^{n+j} for all clauses c_j. This now defined our subset sum problem, with target W and the $2n + \sum_j (k + j - 1)$ numbers defined, suing these additional numbers will allow us to exactly reach W.

To prove that this a valid reduction, we need to establish two claims below establishing the *if* and the *only if* direction of the proof respectively. □

Claim 2. *If the* SAT *problem defined by formula Φ is solvable, than the* Subset Sum *problem we just defined with $2n - m + \sum_j k_j$ numbers is also solvable.*

Proof. Suppose we have a satisfying assignment for the formula Φ, first consider adding the numbers that correspond to the true literals. We used exactly one of w_i and w_{n+i} so will have 1 in the ith digit, and get a sum that is of the form $\sum_{i=1}^{n} B_i + \sum_{j=1}^{m} a_j B^{n+j}$.

Further, will have $1 \leq a_i \leq k_i$, where a_i is at least 1, as the assignment satisfied the formula, so at least one of the numbers added has a 1 in the $(n+j)$th digit, and at most k_j as even adding all numbers at most k_j of them has a 1 in the $(n+j)$th digit. In particular, with $B > k_j$, there will be no carries.

To make this sum to exactly W, we add $k_j - a_j$ copies of the number B^{n+j} we added at the end of the construction. □

Next we need to prove the other direction:

Claim 3. *If the* Subset Sum *problem we just defined with $2n - m + \sum_j k_j$ numbers is solvable, than the* Subset Sum *defined by formula Φ is solvable.*

Proof. First notice that for any subset we may add, there will never be a carry in any digit. To see why, note that all numbers to be summed have all digits 0 or 1; for digit $i = 1, \ldots, n$ we have two numbers with a 1 in that digit w_i and w_{i+n}; the 0th digit is always 0; and for the $n + j$th digit we have exactly $2 \max_j k_j - 1$ numbers that have a 1 on that digits: k_j corresponding to the k_j literals in the clause, and $k_j - 1$ extra numbers B^{n+j} we added at the end. So even is we add all of the numbers, we cannot cause a carry in any of the digits!

Based on the above observation about not having any carries, to get the number W, we need to find a subset I that has exactly the right number of 1's in every digit. First focus on digits $1, \ldots, n$. This digit in W is a 1, and the two numbers that have a 1 in this digit are w_i and w_{i+n}, to to sum to W, we must use exactly one of these, let $I' \subset I$ corresponding to the literals. This

shows that the selected numbers among the first $2n$ of them correspond to a truth assignment of the variables $x_1, \ldots, x_n$.

Finally, we need to show that this truth assignment satisfied the formula Φ. Consider the sum $W' = \sum_{j \in I'} w_j$, just adding the subset I that corresponds to variables. Note that $W' = \sum_{i=1}^{n} B_i + \sum_{j=1}^{m} a'_j B^{n+j}$ with $a'_j \leq k_j$. We need to show that that $a'_j \geq 1$ which will prove that we have a satisfying assignment. Recall that the subset I sums to exactly W. To be able to extend I' with a subset of the additional numbers to sum to W, we must have $a'_j \geq 1$ as there are only $k_j - 1$ copies of B^{n+j}. □

یک وریفایر چندجمله ای به عنوان سرتیفیکیت تعدادی اندیس ورودی میگیرد. وریفایر ابتدا چک میکند که این اندیس ها بین ۱ تا n باشند. سپس اعضا را جمع زده و بررسی میکند که حاصل M شده است یا خیر.این کار در زمان چند جمله‌ای قابل انجام است و $L \in NP$.

حال مشئله $3-sat$ را به ان کاهش میدهیم. مانند بخش قبل فرض میکنیم σ یک ورودی از $3-sat$ با متغیرهای x_i و کلازهای c_j است. به ازای هر x_i ۲ عدد q و w را در نظر میگیریم. به ازای هر کلاز هم ۲ عدد g و h را در نظر میگیریم. فرض میکنیم l متغیر x_i و همچنین k متغیر c_j داریم. اعداد w و q در l رقم پرارزش خود دارای یک رقم یک و باقی رقم صفر هستند. در واقع اعداد w_i و q_i در جایگاه i این مقدار را دارند. رقم jام از k رقم کم ارزش اگر متغیر ما در c_j امده باشد برابر یک و درغیر این صورت صفر هستند. اعداد g و h نیز در k رقم کم ارزشتر تنها دارای یک در جایگاه i هستند. مثال زیر را ببینید:

$$(x_1 \vee \bar{x_2} \vee x_3) \wedge (x_2 \vee x_3 \vee \ldots) \wedge \ldots \wedge (\bar{x_3} \vee \ldots \vee \ldots)$$

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	$\cdots$	0
z_l						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

ادعا میکنیم که ورودی σ به نوعی satisfiable است اگرو تنها اگر مجموعه اعداد ساخته شده و مقدار t عضو زبان L باشد. دقت کنید y و z در تصویر همان q و w گفته شده هستند.

طرف اول

اگر satisfiable باشد، زیرمجموعه از اعداد ورودی را به این شکل میسازیم:

اگر متغیر x برابر یک بود، y_i و در غیر این صورت z_i را انتخاب می‌کنیم. اگر اعداد را جمع کنیم، عدد حاصل دارای یک رقم ۱ در هر یک از رقمlهای پرارزشتر خود است. همچنین هریک از رقمkهای کم ارزش ان بین ۱ و ۳ اسن زیرا هر کلاز satisfy شده است و بین یک تا سه لیترال ان مقدار یک دارد. در نتیجه از g و h ها تعداد مناسب را طوری انتخاب میکنیم که جمع به t برسد.

طرف دوم

حال اگر جمع برابر t باشد، اگر این زیرمجموعه متغیر y داشت، مقدار متغیر متناظر x ان را یک میکنیم. در غیر این صورت ۰ میگذاریم. همچنین با توجه به جدول بالا از بین همه متغیرهای y و z دقیقا یکی انتخاب شده است متغیر x متناظر مقداردهی شده است. چون جمع هر ستون برابر ۳ هست، نتیجه میشود حتما یکی از لیترالهای مربوط به هر کلاز انتخاب شده و ۱ شده است. در نتیجه همه کلازها satisfy شده و σ نیز satisfiable است.

بنابراین اثبات شد.

حل این سوال از سیپسر برداشته شده است.:

SUBSET-SUM is NP-complete.

PROOF IDEA We have already shown that *SUBSET-SUM* is in NP in Theorem 7.25. We prove that all languages in NP are polynomial time reducible to *SUBSET-SUM* by reducing the NP-complete language *3SAT* to it. Given a 3cnf-formula ϕ, we construct an instance of the *SUBSET-SUM* problem that contains a subcollection summing to the target t if and only if ϕ is satisfiable. Call this subcollection T.

To achieve this reduction, we find structures of the *SUBSET-SUM* problem that represent variables and clauses. The *SUBSET-SUM* problem instance that we construct contains numbers of large magnitude presented in decimal notation. We represent variables by pairs of numbers and clauses by certain positions in the decimal representations of the numbers.

We represent variable x_i by two numbers, y_i and z_i. We prove that either y_i or z_i must be in T for each i, which establishes the encoding for the truth value of x_i in the satisfying assignment.

Each clause position contains a certain value in the target t, which imposes a requirement on the subset T. We prove that this requirement is the same as the one in the corresponding clause—namely, that one of the literals in that clause is assigned TRUE.

PROOF We already know that *SUBSET-SUM* $\in$ NP, so we now show that *3SAT* $\leq_{\mathrm{P}}$ *SUBSET-SUM*.

Let ϕ be a Boolean formula with variables $x_1, \ldots, x_l$ and clauses $c_1, \ldots, c_k$. The reduction converts ϕ to an instance of the *SUBSET-SUM* problem $\langle S, t \rangle$, wherein the elements of S and the number t are the rows in the table in Figure 7.57, expressed in ordinary decimal notation. The rows above the double line are labeled

$$y_1, z_1, y_2, z_2, \ldots, y_l, z_l \quad \text{and} \quad g_1, h_1, g_2, h_2, \ldots, g_k, h_k$$

and constitute the elements of S. The row below the double line is t.

Thus, S contains one pair of numbers, y_i, z_i, for each variable x_i in ϕ. The decimal representation of these numbers is in two parts, as indicated in the table. The left-hand part comprises a 1 followed by $l - i$ 0s. The right-hand part contains one digit for each clause, where the digit of y_i in column c_j is 1 if clause c_j contains literal x_i, and the digit of z_i in column c_j is 1 if clause c_j contains literal $\overline{x_i}$. Digits not specified to be 1 are 0.

The table is partially filled in to illustrate sample clauses, c_1, c_2, and c_k:

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3 \vee \cdots) \wedge \cdots \wedge (\overline{x_3} \vee \cdots \vee \cdots).$$

Additionally, S contains one pair of numbers, g_j, h_j, for each clause c_j. These two numbers are equal and consist of a 1 followed by $k - j$ 0s.

Finally, the target number t, the bottom row of the table, consists of l 1s followed by k 3s.

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	$\cdots$	0
z_l						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

FIGURE 7.57
Reducing *3SAT* to *SUBSET-SUM*

Next, we show why this construction works. We demonstrate that ϕ is satisfiable iff some subset of S sums to t.

Suppose that ϕ is satisfiable. We construct a subset of S as follows. We select y_i if x_i is assigned TRUE in the satisfying assignment, and z_i if x_i is assigned FALSE. If we add up what we have selected so far, we obtain a 1 in each of the first l digits because we have selected either y_i or z_i for each i. Furthermore, each of the last k digits is a number between 1 and 3 because each clause is satisfied and so contains between 1 and 3 true literals. We additionally select enough of the g and h numbers to bring each of the last k digits up to 3, thus hitting the target.

Suppose that a subset of S sums to t. We construct a satisfying assignment to ϕ after making several observations. First, all the digits in members of S are either 0 or 1. Furthermore, each column in the table describing S contains at most five 1s. Hence a "carry" into the next column never occurs when a subset of S is added. To get a 1 in each of the first l columns, the subset must have either y_i or z_i for each i, but not both.

Now we make the satisfying assignment. If the subset contains y_i, we assign x_i TRUE; otherwise, we assign it FALSE. This assignment must satisfy ϕ because in each of the final k columns, the sum is always 3. In column c_j, at most 2 can come from g_j and h_j, so at least 1 in this column must come from some y_i or z_i in the subset. If it is y_i, then x_i appears in c_j and is assigned TRUE, so c_j is satisfied. If it is z_i, then $\overline{x_i}$ appears in c_j and x_i is assigned FALSE, so c_j is satisfied. Therefore, ϕ is satisfied.

Finally, we must be sure that the reduction can be carried out in polynomial time. The table has a size of roughly $(k+l)^2$ and each entry can be easily calculated for any ϕ. So the total time is $O(n^2)$ easy stages.

در هر ۳ بخش، راه حداقل یک سورس به همراه توضیحات بنده وجود دارد تا نقصانی وجود نداشته باشد.

۶. **پرسش امتیازی**

نشان دهید که رده‌های **P** و **NP** زیر اجتماع، اشتراک ، الحاق و عملگر ستاره بسته هستند.

پاسخ.

P

Intersection

فرض کنید L_1 و L_2 در P باشند. میخواهیم نشان دهیم اجتماعشان نیز در P است. ماشین تورینگ M_i برای زبان L_i در نظر بگیرید. هر دو دیسایدر هستند. حال ماشین تورینگ M را به شکل زیر میسازیم:

- به ازای ورودی w:
- ماشین M_1 را روی ورودی ران کن.
- اگر رجکت کرد، رجکت کن. در غیر این صورت M_2 را ران کن.
- اگر رجکت کرد، رجکت کن. در غیر این صورت اکسپت کن.

واضحا این ماشین زبان گفته شده را میپذیرد و هالت میکند. در نتیجه M یک دیسایدر است.
اگر پیچیدگی M_i را برابر k_i در نظر بگیریم، داریم:

$$O(n^{k_1} + n^{k_2}) = O(n^{max(k_1,k_2)})$$

در نتیجه $L \in P$.

Union

همه چیز را مانند بخش قبل فرض کنید. M را به این گونه تعریف میکنیم:

- به ازای ورودی w:
- ماشین اول را ران کن. اگر اکسپت کرد، اکسپت و در غیر این صورت ماشین دوم را ران کن.
- اگر ماشین دوم اکسپت کرد، اکسپت و در غیر این صورت رجکت کن.

استدلال ها عینا طبق بخش قبل است بنابراین $L \in P$

- **Concatenation.** We want to show that if $L_1, L_2 \in$ P then $L_1 \circ L_2 \in$ P. Assume so that $L_1 \in$ P and that $L_2 \in$ P. By definition, this means that there exist deciders M_1 and M_2 such that M_1 is a decider for L_1 with time complexity $O(n^{k_1})$ and M_2 is a decider for L_2 with time complexity $O(n^{k_2})$ for some constants k_1 and k_2.

  The concatenation $L_1 \circ L_2$ is defined as

  $$L_1 \circ L_2 = \{x_1 x_2 \mid x_1 \in L_1, x_2 \in L_2\}$$

  The decider for $L_1 \circ L_2$ must, given an input x, try to find a partition of x into $x_1 x_2$ such that $x_1 \in L_1$ and $x_2 \in L_2$. Here is the decider:

  "On input $x = a_1 \ldots a_n$

  1. For $i = 0$ to n do
     i. Let $x_1 = a_1 \ldots a_i$ and $x_2 = a_{i+1} \ldots a_n$. (By agreement $a_1 \ldots a_0 = \epsilon$ and $a_{n+1} \ldots a_n = \epsilon$).
     ii. Run M_1 on the input x_1.
     iii. Run M_2 on the input x_2.
     iv. If both M_1 and M_2 accepted, then accept
  2. If no choice of x_1 and x_2 led to acceptance, then reject"

  We must now show that the decider has polynomial time complexity. The main loop of the decider is traversed at most (n+1)-times. If we run M_1 on a substring of x, this will take at most $O(n^{k_1})$ steps. Similarly, running M_2 on a substring of x will take at most $O(n^{k_2})$ steps. Consequently, a single traversal of the loop body uses no more than $O(n^{k_1}) + O(n^{k_2}) = O(n^k)$ steps, where $k = \max(k_1, k_2)$. The whole decider thus uses $(n+1) \cdot O(n^k) = O(n^{k+1})$ steps. Hence $L(M) = L_1 \circ L_2 \in$ P.

Kleene star

Prove that the class P is closed under Kleene star. (**Hint:** use dynamic programming.)

Solution:
Let $A \in \mathrm{P}$. We want to show that $A^* \in \mathrm{P}$. Since $A \in \mathrm{P}$ there exists a deterministic Turing machine M_A with time complexity $O(n^k)$ for some $k \geq 0$.

We now build, using M_A, a deterministic decider for A^* and show that its time complexity is bounded by a polynomial. The central observation in our construction is that $w \in A^*$ if and only if one of the following conditions is true

- $w = \varepsilon$, or
- $w \in A$, or
- $\exists u, v : w = uv$ and $u \in A^*$ and $v \in A^*$.

In the decider described below we let $w_{i,j}$ denote the substring of $w = w_1 w_2 \dots w_n$ starting with w_i and ending with w_j. The decider builds a table where $table(i,j) = true$ if $w_{i,j} \in A^*$. We do this by considering all substrings of w starting with substrings of length 1 and ending with the substring of length n.

"On input $w = w_1 w_2 \dots w_n$:
1. If $w = \varepsilon$ then accept, else
2. For $\ell := 1$ to n
3.   For $i := 1$ to $n - (\ell - 1)$
4. $j := i + \ell - 1$
5.   Run M_A on $w_{i,j}$
6.   If M_A accepts $w_{i,j}$ then $table(i,j) := true$
7.   Else
8.   For $k := i$ to $j - 1$
9.   If $table(i,k) = true$ and $table(k+1,j) = true$
10.   then $table(i,j) := true$
11. If $table(1,n) = true$ then accept, else reject."

We now analyze the complexity of our decider. The algorithm uses three nested loops, each of which can be traversed at most $O(n)$ times. In the second loop we run M_A on an input of length at most n, so the total time is at most $O(n) \cdot O(n) \cdot (O(n^k) + O(n)) = O(n^{2+(\max(k,1))})$ steps, which is polynomial in n.

برای این بخش از این استفاده شده است.

Intersection

It is an open problem whether NP is closed under complement or not. The proofs for the remaining four language operations can go as follows. Assume that $L_1, L_2 \in$ NP. This means that there are nondeterministic deciders M_1 and M_2 such that M_1 decides L_1 in nondeterministic time $O(n^k)$ and M_2 deciders L_2 in nondeterministic time $O(n^\ell)$. We want to show that

1. there is a nondeterministic poly-time decider M such that $L(M) = L_1 \cap L_2$, and
2. there is a nondeterministic poly-time decider M such that $L(M) = L_1 \cup L_2$, and
3. there is a nondeterministic poly-time decider M such that $L(M) = L_1 \circ L_2$, and
4. there is a nondeterministic poly-time decider M such that $L(M) = L_1^*$.

Now we provide the four machines M for the different operations. The constructions are the standard ones, the additional part is the complexity analysis of the running time. Note that we can use the power of nondeterministic choices to make the constructions very simple.

1. **Intersection:**
   M = "On input w:
   1. Run M_1 on w. If M_1 rejected then reject.
   2. Else run M_2 on w. If M_2 rejected then reject.
   3. Else accept."

   Clearly, the longest branch in any computation tree on input w of length n is $O(n^{\max\{k,\ell\}})$. So M is a poly-time nondeterministic decider for $L_1 \cap L_2$.

Union

Union:
M = "On input w:
1. Run M_1 on w. If M_1 accepted then accept.
2. Else run M_2 on w. If M_2 accepted then accept.
3. Else reject."

Clearly, the longest branch in any computation tree on input w of length n is $O(n^{\max\{k,\ell\}})$. So M is a poly-time nondeterministic decider for $L_1 \cup L_2$. Note that in our case, we do not have the run M_1 and M_2 in parallel, as it was necessary e.g. in the proof that recognizable languages are closed under union. Another possible construction would be to nondeterministically choose either M_1 or M_2 and simulate only the selected machine.

Concatenation

Concatenation:
M = "On input w:
1. Nondeterministically split w into w_1, w_2 such that $w = w_1 w_2$.
2. Run M_1 on w_1. If M_1 rejected then reject.
3. Else run M_2 on w_2. If M_2 rejected then reject.
4. Else accept."

Clearly, the longest branch in any computation tree on input w of length n is still $O(n^{\max\{k,\ell\}})$ because step 1. takes only $O(n)$ steps on e.g. a two tape TM. So M is a poly-time nondeterministic decider for $L_1 \circ L_2$.

Kleene star:
M = "On input w:
1. If $w = \epsilon$ then accept.
2. Nondeterministically select a number m such that $1 \leq m \leq |w|$.
3. Nondeterministically split w into m pieces such that $w = w_1 w_2 \ldots w_m$.
4. For all i, $1 \leq i \leq m$: run M_1 on w_i. If M_1 rejected then reject.
5. Else (M_1 accepted all w_i, $1 \leq i \leq m$), accept."

Observe that steps 1. and 2. take time $O(n)$, because the size of the number m is bounded by n (the length of the input). Step 3. is also doable in polynomial time (e.g. by nondeterministically inserting m separation symbols $\#$ into the input string w). In step 4. the for loop is run at most n times and every run takes at most $O(n^k)$. So the total running time is $O(n^{k+1})$. This means that M is a poly-time nondeterministic decider for L_1^*.

برای این سوال از این استفاده شده است.

موفق باشید :)