۱ ماشین‌های تورینگ و زبان‌های تورینگ‌ـ تشخیص‌پذیر (شمارش‌پذیر بازگشتی)

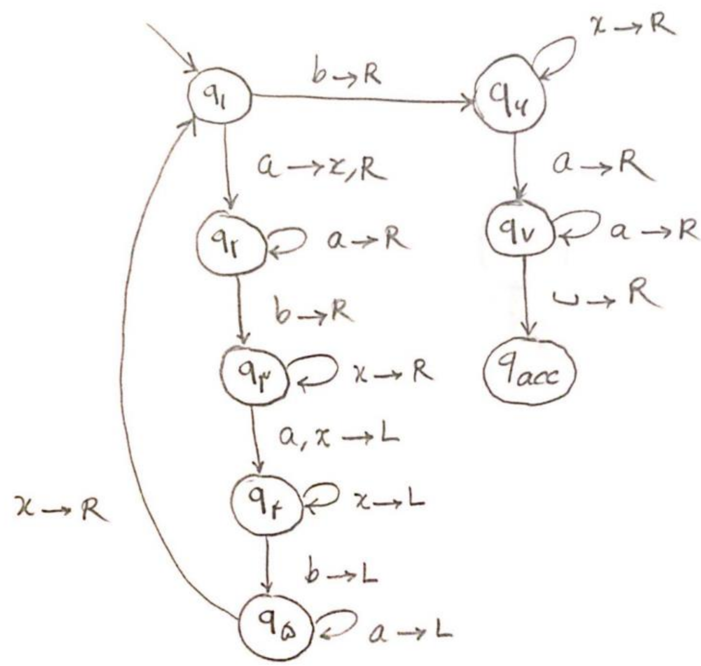
۱,۱

***a*)**

ماشین تورینگ پیش‌نهادی در تصویر زیر قابل مشاهده است. همواره رشته‌ی روی نوار چنین شکلی دارد:

$$xx \ldots xaa \ldots a\ b\ xx..xaa \ldots a$$

در واقع الگوریتم بدین شکل است که از تکه‌ی اول رشته، اولین a را تبدیل به x می‌کنیم و سپس وارد تکه دوم شده و اولین a را تبدیل به x می‌کنیم. مجدداً وارد تکه‌ی اول شده و این کار را تکرار می‌کنیم تا زمانی که همه‌ی a های تکه اول تبدیل به x شوند. سپس بررسی می‌کنیم که انتهای رشته فقط a باقی‌مانده باشد و حداقل یک a هم باشد. در این صورت رشته را می‌پذیریم.



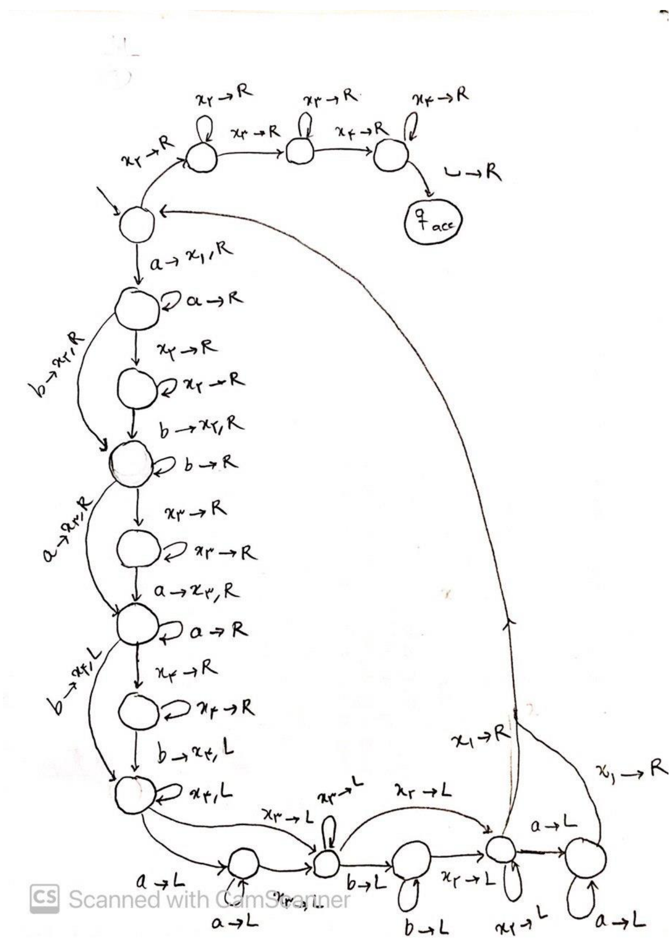
***b*)**

ماشین تورینگ پیشنهادی در تصویر زیر قابل مشاهده است. رشته‌ی روی نوار همواره چنین شکلی دارد:

$$x_1x_1 \dots x_1aa \dots a\ x_2x_2 \dots x_2bb \dots b\ x_3x_3 \dots x_3aa \dots a\ x_4x_4 \dots x_4bb \dots b$$

که تعداد x_1, x_2, x_3 و x_4 با هم برابر است. الگوریتم بدین شکل است که رشته ۴ بلوک دارد. در بلوک اول، اولین a را تبدیل به x_1 می‌کنیم، سپس در بلوک دوم اولین b را تبدیل به x_2 می‌کنیم، سپس در بلوک سوم اولین a را تبدیل به x_3 می‌کنیم و نهایتا در بلوک چهارم اولین b را تبدیل به x_4 می‌کنیم. سپس مجدداً به بلوک اول برمیگردیم و این کار را تکرار می‌کنیم.

زمانی که بلوک اول هیچ a ای نداشت، بررسی می‌کنیم که آیا همه‌ی رشته تبدیل به $x_1 \dots x_1\ x_2 \dots x_2\ x_3 \dots x_3\ x_4 \dots x_4$ شده است یا خیر. اگر همه رشته مارک شده باشد، این رشته پذیرفته می‌شود.



۲,۱

***a*)**

به دلیل پیچیدگی بیشتر در این سوال صرفا الگوریتم ماشین تورینگ را توصیف می‌کنیم. فرایند بدین شکل است که ابتدا اولین ۱ را تبدیل به s می‌کنیم.

سپس الگوریتم زیر را تا زمانی که ۱ در رشته داریم اجرا می‌کنیم:

- اولین ۱ در رشته را پیدا کن، آن را تبدیل به x کن. سپس به تعداد ۱ ها و x ها و s ها در رشته، به انتهای رشته y اضافه کن.

پس از تمام شدن ۱ ها در رشته، همه‌ی کاراکترهای رشته را تبدیل به ۱ کن.

یک نمونه از اجرا به ازای $x = 3$ به صورت زیر است (نوار ماشین تورینگ در هر گام مشخص شده است).

$$111 \to s11 \to sx1 \to sx1yyy \to sxxyyy \to sxxyyyyyy \to 111111111$$

***b*)**

به دلیل پیچیدگی بیشتر در این سوال صرفا الگوریتم ماشین تورینگ را توصیف می‌کنیم. مقدار $\lceil \log_2(x+1) \rceil$ بدین شکل است که اگر n بزرگترین عددی باشد که $2^n \leq x$، مقدار $\lceil \log_2(x+1) \rceil = n + 1$ است. پس ایده این است که بزگترین n را بیابیم.

اولین ۱ را تبدیل به s می‌کنیم و انتهای رشته یک y اضافه می‌کنیم. سپس الگوریتم زیر را اجرا می‌کنیم:

- در هر گام به تعداد s های موجود در ابتدای رشته، ۱ های ابتدا را تبدیل به x می‌کنیم
  - اگر در حین انجام این کار، ۱ ها تمام شد و نتوانستیم به تعداد لازم ۱ ها را تبدیل به x کنیم، از این فرایند خارج می‌شویم.
  - وگرنه اگر به تعداد لازم ۱ ها تبدیل به x شدند همه‌ی این xها را تبدیل به s می‌کنیم و یک y به انتهای رشته اضافه می‌کنیم

  (دقت کنید که تعداد s ها در رشته همواره توان ۲ ای است که x بزرگتر مساوی آن است و هر بار یکی به y ها که تعداد جواب را می‌شمارد اضافه می‌کنیم).

با خروج از مراحل بالا، تعداد y ها برابر پاسخ مسئله است. این y ها در انتهای نوار هستند. آن‌ها را ابتدا انتقال می‌دهیم و باقی رشته را پاک می‌کنیم. سپس همه‌ی y ها را تبدیل به ۱ می‌کنیم.

یک نمونه از اجرا به ازای $x = 6$ به صورت زیر است (نوار ماشین تورینگ در هرگام مشخص شده است).

$$111111 \to s11111y \to sx1111y \to ss1111yy \to ssxx11yy \to ssss11yyy \to ssssxxyyy \to yyy \to 111$$

دقت کنید که در مرحله‌ی $ssss11yyy$ به تعداد لازم ۱ نداریم و اجرا متوقف می‌شود.

$\lceil \log_2(6+1) \rceil = 3$

۳,۱

ماشین‌های تعریف سوال، قدرتی در حد NFA دارند و پذیرنده‌ی زبان‌های منظم هستند. دقت کنید که تابع δ بدین شکل است که براساس استیت موجود (s) و آنچه روی نوار نوشته شده است (a):

- یا محتوای نوار را تغییر می‌دهد و به **سمت راست می‌رود** و وارد استیت s' می‌شود که در این صورت محتوای تغییر داده شده هیچ اهمیتی ندارد چون هیچگاه به سمت چپ برنمی‌گردیم و این بخش از نوار را دیگر دسترسی نداریم.
- یا محتوای نوار را تغییر می‌دهد و در **جایش باقی می‌ماند** و وارد استیت s_1 می‌شود. در این صورت همچنان سر جایمان هستیم و آنقدر محتوای نوار را تغییر داده و تغییر استیت می‌دهیم تا نهایتا به وضعیت s_k برسیم (در واقع با تغییر محتوای نوار در آن خانه، بدون تغییر مکان استیت‌های $s \to s_1 \to s_2 \to \cdots \to s_k$ را طی کرده‌ایم) که در وضعیت s_k حالت اول رخ می‌دهد و محتوای نوار را تغییر می‌دهیم و به **سمت راست رفته** و وارد استیت s' می‌شویم. دقت کنید که در این حالت هم محتویاتی که بارها در خانه‌ی قبلی روی نوار تغییر دادیم همچنان اهمیتی ندارد چون هیچگاه دوباره به آن دسترسی نخواهیم داشت.

پس در عمل در هر دو حالت، ما از استیت s با محتوای روی نوار a به وضعیت s' رفته و در نوار هم یکی به سمت راست حرکت می‌کنیم. پس در واقع می‌توانیم در NFA مان، یال a را از استیت s به استیت s' بکشیم و فرایند پارس رشته‌ی ما دقیقا مشابه NFA است و هر عضو رشته دقیقا یکبار اثر گذار است.

پس این ماشین‌ها پذیرنده‌ی زبان‌های منظم هستند.

۴,۱

حل با کمک این منبع.

نشان می‌دهیم ماشین‌های صف‌دار با ماشین‌های تورینگ معادلند. برای این منظور، ابتدا نشان می‌دهیم هر ماشین تورینگ را می‌توان با یک ماشین صف‌دار مدل کرد؛ ضمنا هر ماشین صف‌دار هم با ماشین تورینگ قابل پیاده‌سازی است.

شبیه‌سازی ماشین تورینگ با ماشین صف‌دار

می‌دانیم ماشین صف‌دار بدین شکل عمل می‌کند که یک صف دارد و هر بار عنصر ابتدای صف را می‌خواند و می‌تواند در انتهای صف بنویسد. ضمنا یک بار هم رشته‌های ورودی را پارس می‌کند. ایده شبیه‌سازی ماشین تورینگ با ماشین صف‌دار این است که محتوای نوار را همواره توی صف داشته باشیم.

در واقع صف بدین شکل است که عنصر ابتدای آن، عنصری است که در ماشین تورینگ روی آن هستیم و ضمنا شروع نوار را با کاراکتر ویژه‌ی \$ در صف معلوم می‌کنیم. در حقیقت محتوای صف همواره به شکل $ax\$y$ است که a ابتدای صف بوده و محتوای خانه‌ای است که ماشین تورینگ روی آن است. رشته‌ی x محتوای سمت راست نوار ماشین تورینگ است و رشته‌ی y محتوای سمت چپ ماشین تورینگ. در حقیقت نوار ماشین تورینگ در این وضعیت صف، معادل yax است.

اکنون می‌خواهیم یک گام از عملیات ماشین تورینگ را شبیه‌سازی کنیم. فرض می‌کنیم کاراکتر a در محل فعلی ماشین است و صف به صورت $ax\$y$ و نوار به صورت yax است.

- اگر ماشین تورینگ با خواندن a جای آن b بنویسد و به سمت راست برود ما در صف:

a را که ابتدای صف است حذف می‌کنیم و b را به انتهای صف اضافه می‌کنیم. در این حالت صف به شکل $x\$yb$ می‌شود که معادل نوار ybx است که ماشین تورینگ را به درستی مشخص می‌کند و هد نوار هم در شروع x است (اول صف).

- اگر ماشین تورینگ با خواندن a به جای آن b بنویسد و به سمت چپ برود ما در صف:
a را که ابتدای صف است حذف می‌کنیم و b را به انتهای صف اضافه می‌کنیم. در این حالت صف به شکل $x\$yb$ می‌شود که معادل نوار ybx است که ماشین تورینگ را به درستی مشخص می‌کند اما هد نوار در شروع x است که در حالی که باید پشت b باشد. برای این منظور، **دو بار محتوای صف را به سمت راست شیفت دوری** می‌دهیم. در این حالت محتوای صف تبدیل به $y_{last}bx\$y_{[0:last-1]}$ می‌شود که معادل به نوار ybx است که درست بوده و هد نوار هم جای درستی قرار دارد (حرف قبل از b اول صف است).

پس اگر عملیات شیفت دوری را تعریف کنیم و بتوانیم آن را انجام دهیم، ماشین تورینگ را شبیه‌سازی کرده‌ایم. عملیات شیفت دوری کل محتوای تورینگ ماشین هم به سادگی قابل انجام است. فرض کنید در حال حاضر روی نوار t نوشته باشد. وارد استیت q_t می‌شویم و سپس به سمت راست می‌رویم. در حالت جدید روی نوار محتوای t' را می‌بینیم. چون در وضعیت q_t هستیم مقدار t روی نوار می‌نویسیم و چون t' خوانده‌ایم، وارد وضعیت $q_{t'}$ می‌شویم. بدین ترتیب کل محتوای نوار را شیفت داده‌ایم. صرفا لازم است محتوای آخر نوار را هم به ابتدا انتقال دهیم که مجدداً از ایده‌ی مشابه استفاده کرده و این کار را انجام می‌دهیم (می‌توان در ابتدای نوار یک مقدار # گذاشت که همواره بدانیم ابتدای نوار دقیقا کجاست).

ضمناً ابتدای کار هم همه‌ی رشته را می‌خوانیم و آن‌ها را به صف اضافه می‌کنیم تا دقیقا در ابتدا صف مثل نوار ماشین تورینگ بشود. پس ثابت شد قدرت هر ماشین تورینگ حداکثر به اندازه‌ی ماشین صف‌دار است.

پیاده‌سازی ماشین صف‌دار با ماشین تورینگ

ما ماشین صف‌دار را با ماشین تورینگ ۲ ـ نواره (که در کلاس ثابت شد معادل ماشین ۱ ـ نواره است) شبیه‌سازی می‌کنیم. در این صورت نشان داده‌ایم ماشین صف‌دار را می‌توان با ماشین تورینگ شبیه‌سازی کرد.

برای شبیه‌سازی، نوار اول همان رشته‌ی ورودی را شامل می‌شود و نوار دوم دربرگیرنده صف است. صف را به این شکل شبیه‌سازی می‌کنیم که در ابتدای نوار دوم تعدادی $ وجود دارد و صف از بعد آن‌ها شروع می‌شود (در ابتدا هم یک $ در نوار دوم قرار داده‌ایم).

- هرگاه عملیات $push$ کاراکتر a به صف انجام شد:
  اولین کاراکتر خالی (همان $blank$) ته نوار را تبدیل به a می‌کنیم.
- هر گاه عملیات pop انجام شد:
  اولین کاراکتر غیر $ از ابتدای نوار را پیدا می‌کنیم و آن را تبدیل به $ می‌کنیم.

مثلا فرض کنید عملیات‌های صف بدین شکل انجام شوند، نوار دوم بدین صورت تغییر می‌کند.

push a, push b, push a, pop, pop, push c, push d, pop

$\$a \rightarrow \$ab \rightarrow \$aba \rightarrow \$\$ba \rightarrow \$\$\$a \rightarrow \$\$\$ac \rightarrow \$\$\$acd \rightarrow \$\$\$\$cd$

دقت کنید که چون نوار ماشین تورینگ بی‌نهایت است، پس مشکلی پیش نمی‌آید و هر چقدر بخواهیم می‌تواند جلو برود.

پس بدین شکل ماشین صف‌دار با ماشین تورینگ ۲ـ نواره شبیه‌سازی شد که نشان می‌دهد قدرتش حداکثر ماشین تورینگ است.

پس با اثبات هر دو بخش، ثابت شد این دو ماشین معادلند. ■

۲ ماشین‌های خطی کران‌دار و یک کلاس جدید از زبان‌ها

۱,۲

***a*)**

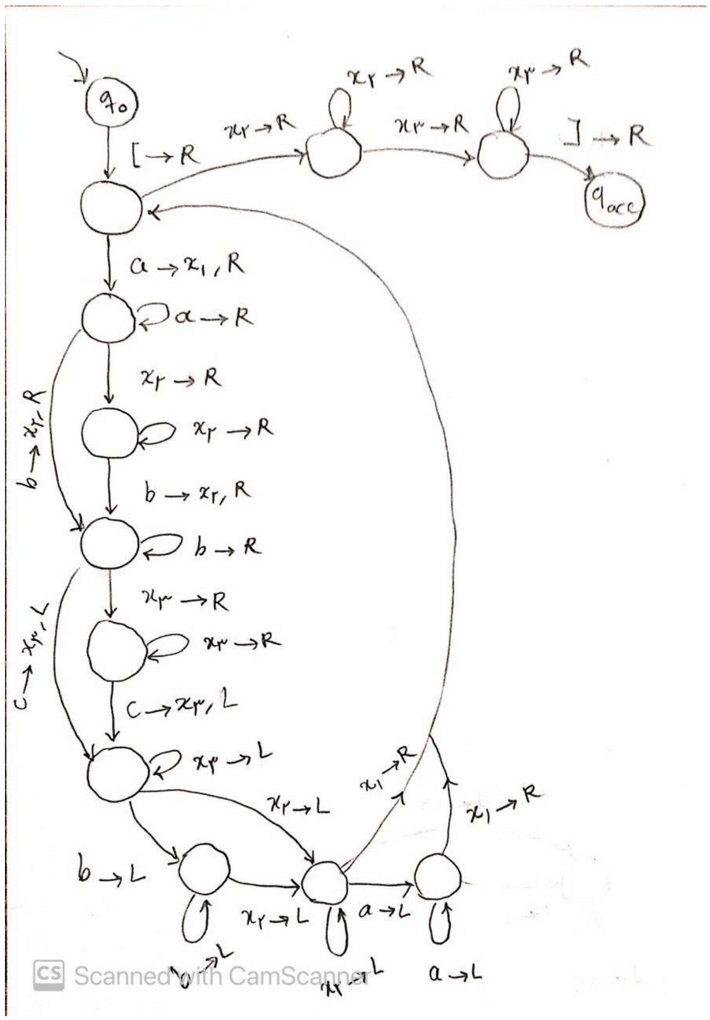
ایده‌ی طراحی مشابه بخش b سوال ۱,۱ است. در آنجا هم هیچگاه از محدوده‌ی رشته خارج نمی‌شدیم و ماشینی که تعریف کرده بودیم کران‌دار بود.

ماشین تورینگ پیشن‌نهادی در تصویر زیر قابل مشاهده است. رشته‌ی روی نوار همواره چنین شکلی دارد:

$$[x_1x_1 \dots x_1aa \dots a\ x_2x_2 \dots x_2bb \dots b\ x_3x_3 \dots x_3cc \dots c]$$

که تعداد x_1, x_2, x_3 با هم برابر است. الگوریتم بدین شکل است که رشته ۳ بلوک دارد. در بلوک اول، اولین a را تبدیل به x_1 می‌کنیم، سپس در بلوک دوم اولین b را تبدیل به x_2 می‌کنیم، سپس در بلوک سوم اولین c را تبدیل به x_3 می‌کنیم. سپس مجدداً به بلوک اول برمیگردیم و این کار را تکرار می‌کنیم.

زمانی که بلوک اول هیچ a ای نداشت، بررسی می‌کنیم که آیا همه‌ی رشته تبدیل به $[x_1 \dots x_1\ x_2 \dots x_2\ x_3 \dots x_3]$ شده است یا خیر. اگر همه رشته مارک شده باشد، این رشته پذیرفته می‌شود.



***b*)**

ایده‌ی راه حل این است که نخست تکه‌ی وسط رشته را پیدا کنیم. بدین شکل عمل می‌کنیم که اول کاراکتر ابتدای رشته را از a به $\tilde{a}$ و b را به $\tilde{b}$ تبدیل می‌کنیم. سپس کاراکتر انتهای رشته را این کار را برایش می‌کنیم. مجددا اولین کاراکتر a, b ابتدای رشته را تبدیل به $\check{a}, \tilde{b}$ می‌کنیم و سپس آخرین کاراکتر a, b انتهای رشته را تبدیل می‌کنیم و ...

پس از انجام موفقیت‌آمیز این بخش، همه a, b ها تبدیل به $\tilde{a}, \tilde{b}$ می‌شوند و هدر نوار در وسط رشته قرار می‌گیرد.

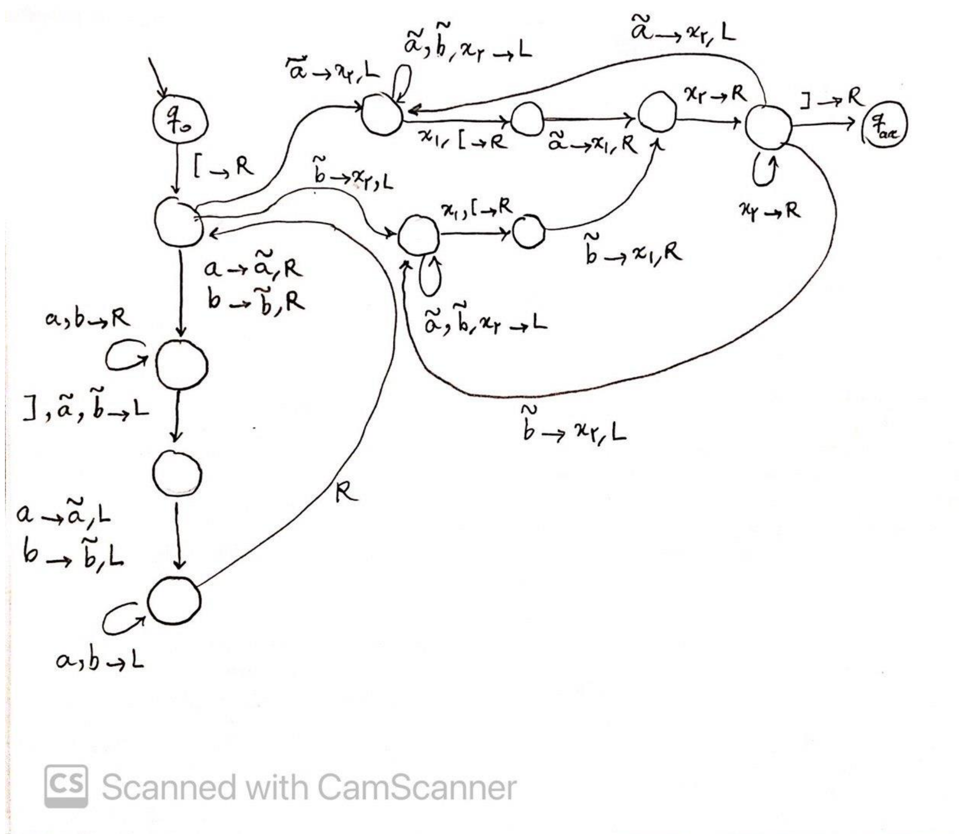
$$ababaa \to \tilde{a}babaa \to \tilde{a}baba\tilde{a} \to \tilde{a}\tilde{b}aba\tilde{a} \to \tilde{a}\tilde{b}ab\tilde{a}\tilde{a} \to \tilde{a}\tilde{b}\tilde{a}b\tilde{a}\tilde{a} \to \tilde{a}\tilde{b}\tilde{a}\tilde{b}\tilde{a}\tilde{a}$$

در این‌جا هدر نوار نهایتا روی حرف چهارم قرار می‌گیرد.

سپس در ادامه بدین شکل عمل می‌کنیم که در تکه دوم اولین حرف را تبدیل به x_2 می‌کنیم و سپس وارد تکه‌ی اول می‌شویم و اولین حرف غیر x_1 را چک می‌کنیم که همان حرفی باشد که دیدیم و آن را تبدیل به x_1 می‌کنیم. سپس مجدداً در تکه‌ی دوم، اولین حرف را تبدیل به x_2 می‌کنیم و در تکه‌ی اول چک می‌کنیم که همان باشد و تبدیل به x_1 می‌کنیم.. مثلا برای رشته $abaaba$ که پس از فاز اول تبدیل به $\tilde{a}\tilde{b}\tilde{a}\tilde{a}b\tilde{a}$ می‌شود فرایند به صورت زیر است:

$\tilde{a}\tilde{b}\tilde{a}\tilde{\tilde{a}}b\tilde{a} \rightarrow \tilde{a}\tilde{b}\tilde{a}x_2b\tilde{a} \rightarrow x_1\tilde{b}\tilde{a}x_2b\tilde{a} \rightarrow x_1\tilde{b}\tilde{a}x_2x_2\tilde{a} \rightarrow x_1x_1\tilde{a}x_2x_2\tilde{a} \rightarrow x_1x_1\tilde{a}x_2x_2x_2 \rightarrow x_1x_1x_1x_2x_2x_2$

نهایتا اگر رشته تبدیل به $x_2 \dots x_1x_2 \dots x_1$ شد، رشته پذیرفته می‌شود. ماشین تورینگ طراحی شده به صورت زیر است:



۲,۲

برای تعیین اینکه این گرامر چه زبانی را توصیف می‌کند، تا چند مرحله عملیات‌های پروداکشن را انجام می‌دهیم. این گرامر علیرغم شکل پیچیده‌اش زبان ساده‌ای را پارس می‌کند.

$S \rightarrow ACaB \rightarrow AaaCB \rightarrow AaaE$ or $AaaDB$

- $Aa\boldsymbol{aD}B \rightarrow A\boldsymbol{aD}aB \rightarrow \boldsymbol{AD}aaB \rightarrow A\boldsymbol{Ca}aB \rightarrow Aaa\boldsymbol{Ca}B \rightarrow Aaaaa\boldsymbol{CB} \rightarrow AaaaaE$ or $AaaaaDB$
- $Aa\boldsymbol{aE} \rightarrow A\boldsymbol{aE}a \rightarrow \boldsymbol{AE}aa \rightarrow aa$

پس همانطور که مشاهده می‌شود، از وضعیت اولیه یا

- به حالت $AaaE$ می‌رویم که در این حالت رشته‌ی aa پذیرفته می‌شود.
- یا به وضعیت $AaaDB$ می‌رویم که از آن وضعیت به یکی از دو وضعیت $AaaaaDB$ یا $AaaaaE$ می‌رسیم.
  - اگر به وضعیت دوم برویم، نهایتا رشته‌ی $aaaa$ پذیرفته می‌شود

- اگر به وضعیت اول برویم مجدداً تعداد a های وسط دو برابر می‌شود به یکی از دو حالت $AaaaaaaaaE$ یا $AaaaaaaaaDB$ می‌رسیم و به همین شکل الگو ادامه پیدا می‌کند.

در واقع زبان فوق رشته‌هایی را می‌پذیرد که تعداد a در آن‌ها 2^n است ($n \geq 1$).

$$L(G) = \{aa, aaaa, aaaaaaaa, \dots\} = \{a^{2^n} | n \geq 1\}$$

اکنون یک گرامر وابسته به متن ارائه می‌دهیم که زبان فوق را بپذیرد. برای این موضوع نخست یک گرامر *Noncontracting* ارائه می‌دهیم که زبان فوق را تولید کند (از این منبع استفاده شده است). سپس با استفاده از این منبع آن را به یک گرامر وابسته به متن که **قاعده‌های آن به فرمت $\alpha X\gamma \rightarrow \alpha\beta\gamma$ هستند تبدیل می‌کنیم.**

گرامر *Noncontracting* ارائه شده به صورت زیر است:

$$\begin{aligned}
&S \rightarrow EAE \mid aa \\
&EA \rightarrow E2A \mid TA \\
&2A \rightarrow AA2 \\
&2AE \rightarrow AAE \\
&TA \rightarrow aaaaT \\
&TAE \rightarrow aaaa
\end{aligned}$$

گرامر فوق رشته‌ی‌های به صورت a^{2^n} را می‌پذیرد. شهود گرامر فوق این است سر و ته رشته همواره E وجود دارد و در وسط تعدادی A است (این تعداد توانی از ۲ می‌باشد). با کمک قاعده‌ی $EA \rightarrow E2A$ یک 2 به ابتدای A ها اضافه می‌کنیم و سپس با کمک قاعده‌ی $2A \rightarrow AA2$ تعداد A های وسط را دو برابر می‌کنیم. نهایتا با قاعده‌ی $2AE \rightarrow AAE$ حرف 2 را از رشته حذف می‌کنیم. هرگاه خواستیم تمامی متغیرها را حذف کنیم از قاعده‌ی $EA \rightarrow TA$ استفاده می‌کنیم و با ظهور یک T در ابتدای رشته سپس از قاعده‌ی $TA \rightarrow aaaaT$ استفاده می‌کنیم و همه‌ی A ها را (جز آخری) تبدیل به ۴ تا a می‌کنیم. نهایتا از قاعده‌ی $TAE \rightarrow aaaa$ استفاده کرده و همه‌ی متغیرها را پاک می‌کنیم. رشته به دست آمده، ۴ برابر تعداد A های رشته a دارد که همچنان توانی از ۲ است. در واقع مثلا از وضعیت $EAE \rightarrow aaaa$ می‌رسیم، از وضعیت $EAAE \rightarrow aaaaaaaa$ می‌رسیم و …

حالت aa هم به شکل مستقیم در گرامر وجود دارد.

اکنون این گرامر را تبدیل به گرامر وابسته به متن می‌کنیم. از ایده‌ی لینک ارائه شده در ویکی‌پدیا استفاده شده است.

از میان قواعد بالا، چهار قاعده‌ی $2A \rightarrow AA2$، $2AE \rightarrow AAE$، $TA \rightarrow aaaaT$ و $TAE \rightarrow aaaa$ در فرمت قواعد گرامرهای وابسته به متن نیستند که آن‌ها را باید اصلاح کنیم.

متغیر $\tilde{A}$ را تعریف می‌کنیم و قاعده‌ی $\tilde{A} \rightarrow aaaa$ را به مجموعه‌ی قواعد می‌افزاییم.

همچنین متغیر $\tilde{a}$ را نیز تعریف کرده و قاعده‌ی $\tilde{a} \rightarrow a$ را نیز به مجموعه‌ی قواعد اضافه می‌کنیم.

متغیر D را نیز تعریف می‌کنیم و قاعده‌ی $D \to 2$ را نیز به مجموعه‌ی قواعد می‌افزاییم.

رفع مشکل قاعده‌ی $2A \to AA2$:

نخست این قاعده را تبدیل به $DA \to AAD$ می‌کنیم و مجموعه‌ی قواعد زیر را جایگزین می‌کنیم.

$DA \to AAD$
$- - - - - - - - - -$
$DA \to Z_1A$
$Z_1A \to Z_1Z_2D$
$Z_1Z_2D \to AZ_2D$
$AZ_2D \to AAD$

رفع مشکل قاعده‌ی $2AE \to AAE$:

این قاعده را تبدیل به $DAE \to AAE$ می‌کنیم.

رفع مشکل قاعده‌ی $TA \to aaaaT$:

نخست این قاعده را تبدیل به $TA \to \tilde{A}T$ می‌کنیم و مجموعه قواعد زیر را جایگزین می‌کنیم.

$TA \to \tilde{A}T$
$- - - - - - - - - -$
$TA \to Y_1A$
$Y_1A \to Y_1Y_2$
$Y_1Y_2 \to \tilde{A}Y_2$
$\tilde{A}Y_2 \to \tilde{A}T$

رفع مشکل قاعده‌ی $TAE \to aaaa$:

نخست ین قاعده را تبدیل به $TAE \to \tilde{a}\tilde{a}\tilde{a}\tilde{a}$ می‌کنیم و مجموعه قواعد زیر را جایگزین می‌نماییم.

$TAE \to \tilde{a}\tilde{a}\tilde{a}\tilde{a}$
$- - - - - - - - - -$
$TAE \to F_1AE$
$F_1AE \to F_1F_2E$
$F_1F_2E \to F_1F_2\tilde{a}\tilde{a}$
$F_1F_2\tilde{a}\tilde{a} \to \tilde{a}F_2\tilde{a}\tilde{a}$
$\tilde{a}F_2\tilde{a}\tilde{a} \to \tilde{a}\tilde{a}\tilde{a}\tilde{a}$

لذا گرامر وابسته به متن با رعایت الگوی مطلوب سوال بدین شکل است ($V = \{S, T, A, E, D, \tilde{a}, \tilde{A}, Z_1, Z_2, Y_1, Y_2, F_1, F_2\}$):

$S \to EAE \mid aa$
$EA \to E2A \mid TA$
$DA \to Z_1A$

$Z_1A \to Z_1Z_2D$
$Z_1Z_2D \to AZ_2D$
$AZ_2D \to AAD$
$DAE \to AAE$
$TA \to Y_1A$
$Y_1A \to Y_1Y_2$
$Y_1Y_2 \to \tilde{A}Y_2$
$\tilde{A}Y_2 \to \tilde{A}T$
$TAE \to F_1AE$
$F_1AE \to F_1F_2E$
$F_1F_2E \to F_1F_2\tilde{a}\tilde{a}$
$F_1F_2\tilde{a}\tilde{a} \to \tilde{a}F_2\tilde{a}\tilde{a}$
$\tilde{a}F_2\tilde{a}\tilde{a} \to \tilde{a}\tilde{a}\tilde{a}\tilde{a}$
$D \to 2$
$\tilde{a} \to a$
$\tilde{A} \to aaaa$

۳ ماشین‌های خطی کران‌دار و یک کلاس جدید از زبان‌ها

۱,۳

مجموعه‌ی $\sum *$ مجموعه‌ی همه‌ی کلماتی است که با حروف الفبای $\sum$ تولید می‌شوند. مجموعه‌ی $\sum *$ یک مجموعه‌ی شمارا است. اما یک زبان، زیرمجموعه‌ای از این مجموعه است لذا مجموعه‌ی همه‌ی زبان‌های ممکن مجموعه‌ی توانی و در واقع $P(\sum *)$ است که مجموعه‌ای ناشماراست.

اکنون نشان می‌دهیم مجموعه‌ی همه ماشین‌های تورینگ‌ـ تشخیص‌پذیر یک مجموعه‌ی شماراست. هر ماشین تورینگ با توصیف $M = (Q, T, b, \sum, \delta, q0, F)$ تعریف می‌شود که می‌توان آن را به فرمت یک رشته نمایش داد. برای نمایش به فرمت یک رشته، رأس‌های Q را شماره‌گذاری می‌کنیم و شماره‌ها را داخل رشته قرار می‌دهیم؛ اعضای استک را در آن رشته قرار می‌دهیم؛ عضو b (همان $blank$) را در رشته قرار می‌دهیم؛ تابع δ را با در نظر گرفتن همه‌ی حالاتش نوشته و در رشته قرار می‌دهیم؛ راس $q0$ و مجموعه‌ی F را هم در این رشته قرار می‌دهیم.

چنین رشته‌ای الفبای محدودی دارد. این الفبا را R می‌نامیم. پس همه‌ی رشته‌های ممکن (همه‌ی ماشین‌های ممکن)، نهایتا زیرمجموعه‌ای از R^* است که مجموعه‌ای شمارا است.

بنابراین مجموعه‌ی همه‌ی ماشین تورینگ‌های تشخیص‌پذیر مجموعه‌ای شماراست و هر ماشین هم یک زبان را می‌پذیرد، در حالی که مجموعه‌ی همه‌ی زبان‌های ممکن ناشماراست. پس اگر S_1 را مجموعه‌ی زبان‌هایی که ماشین‌های توصیف شده می‌پذیرند قرار دهیم، S_1 شماراست ولی S_2 که همه‌ی زبان‌های ممکن است ناشماراست. پس طبق لم صورت سوال، S_2 بی‌شمار عنصر دارد که عضو S_1 نیستند و این یعنی بی‌نهایت زبان داریم که توسط همه‌ی ماشین‌های ممکن تشخیص‌پذیر نیستند.

۲,۳

کافی است ماشینی ارائه دهیم که زبان L را تصمیم بگیرد. می‌دانیم هر دو $L, \overline{L}$ تشخیص‌پذیر هستند پس ماشین‌های M_1, M_2 وجود دارند که M_1 زبان L و M_2 زبان $\overline{L}$ را تشخیص دهد.

ماشین M را بدین شکل تعریف می‌کنیم:

M on input x:
run these two machines parallel:

* run M_1 on input x
* run M_2 on input x

if M_1 accepts x then **accept** else if M_2 accepts x then **reject**.

دقت کنید که ماشین فوق تصمیم‌پذیر است. علت امر این است که برای ورودی x از دو ماشین M_1, M_2 استفاده می‌کند که هر دو تشخیص‌پذیرند و کلمات زبانشان را بالاخره می‌پذیرند. پس چون زبان این دو ماشین مکمل یکدیگر است. برای هر رشته مثل x بالاخره یکی از M_1, M_2 آن را می‌پذیرد. سپس براساس اینکه کدام یک پذیرفته است می‌توانیم بفهمیم که x را باید بپذیریم یا نه. پس ماشینی تصمیم‌پذیر برای زبان L ارائه دادیم. پس L تصمیم‌پذیر است.

از طرفی اگر یک زبان تصمیم‌پذیر باشد، بدیهتاً مکمل آن هم تصمیم‌پذیر است. تنها کافی است ماشین تورینگ M' را طراحی کنیم که برای هر ورودی x، آن را به M که تصمیم‌پذیر است می‌دهد و اگر M ورودی x را بپذیرد، ورودی x را رد کند وگرنه آن را بپذیرد.

پس ثابت شد هر دو $L, \overline{L}$ تصمیم‌پذیرند.

۳,۳

ماشین تورینگی را توصیف می‌کنیم که یک رشته x بگیرد و بگوید آیا در مکمل زبان مستقل از متن (که نامش را S می‌گذاریم) قرار می‌گیرد یا خیر. در واقع ماشین تورینگ قرار است تخشیص دهد که آیا S ورودی را می‌پذیرد یا خیر. اگر رشته‌ی ورودی توسط زبان مستقل از متن S پذیرفته شود خروجی باید **reject** باشد وگرنه **accept**.

هر زبان مستقل از متن با یک گرامر توصیف می‌شود. در حقیقت گرامری وجود دارد که آن زبان را می‌پذیرد. آن گرامر را در فرم نرمال چامسکی قرار می‌دهیم. فرض کنید طول رشته‌ی ورودی (با نام x) برابر با k باشد. می‌دانیم دقیقا $2k - 1$ اشتقاق لازم است تا رشته تولید شود.

فرض کنید تعداد قواعد گرامران c باشد و در اشتقاق‌ها همواره اشتقاق را روی چپ‌ترین متغیر اعمال کنیم.

ماشین تورینگی توصیف می‌کنیم که این الگوریتم را پیاده‌سازی کند:

همه‌ی حالات ممکن برای اشتقاق و تولید یک رشته با طول k را در نظر بگیرید. تعداد کل حالات ممکن c^{2k-1} است. همه‌ی حالات را تولید می‌کنیم و نهایتاً می‌بینیم که آیا رشته‌ی تولید شده، همان رشته‌ی x هست یا خیر. اگر در بین کل حالات توانستیم x را تولید کنیم در این صورت الگوریتم **reject** برمیگرداند وگرنه **accept**.

پس در زمان متناهی و به شکل تصمیم‌پذیر ماشین ما خروجی می‌دهد و می‌فهمیم که رشته‌ی x در مکمل زبان مستقل از متن S قرار می‌گیرد یا خیر.

۴,۳

M_A را ماشین تصمیم‌پذیر زبان A و M_B را ماشین تصمیم‌پذیر زبان B می‌نامیم.

M_C را ماشین تشخیص‌پذیر زبان C می‌نامیم.

الف) برای اثبات اینکه $A \cap B$ تصمیم‌پذیر است، یک ماشین M ارائه می‌دهیم که اشتراک زبان‌های A, B را بپذیرد.

M on input x:
run M_A on input x
run M_B on input x
if both M_A, M_B accepts x then **accept** else **reject**.

واضح است که ماشین فوق تصمیم‌پذیر است چون از خروجی دو ماشین تصمیم‌پذیر استفاده می‌کند و زبان اشتراک را می‌یابد.

برای اثبات اینکه $A \cup B$ تصمیم‌پذیر است، یک ماشین M ارائه می‌دهیم که اجتماع زبان‌های A, B را بپذیرد. دقیقا از ایده‌ی قسمت قبل استفاده می‌کنیم.

M on input x:
run M_A on input x
run M_B on input x
if at least one of M_A, M_B accepts x then **accept** else **reject**.

ب) برای اثبات اینکه A^R تصمیم‌پذیر است، ماشین M را ارائه می‌دهیم که تصمیم‌پذیر بوده و زبان $Reverse$ را می‌پذیرد.

M on input x:
reverse input and change input to x^R
run M_A on new input (i.e. x^R)
if M_A accepts that input then **accept** else **reject**.

واضح است که ماشین فوق تصمیم‌پذیر است و یک رشته را می‌پذیرد اگر و تنها اگر زبان A وارونه آن را بپذیرد.

برای اثبات تشخیص‌پذیر بودن C^R هم دقیقا ایده قبلی را استفاده می‌کنیم منتها از ماشین M_C باید استفاده کنیم که سبب می‌شود ماشین ما هم تشخیص‌پذیر شود.

M on input x:
reverse input and change input to x^R

return "run M_C on new input $(\text{i.e.}\ x^R)$"

دقت کنید که وارونه کردن رشته‌ی ورودی کار بسیار ساده‌ای است که در زمان متناهی (وابسته به طول رشته) انجام می‌شود.

ج) برای اینکه نشان دهیم زبان $\bar{A}$ تورینگ‌ـ تصمیم‌پذیر است ماشین M' را می‌سازیم که زبان مکمل را بپذیرد. برای ساخت این ماشین کافی است ماشین M_A را در نظر بگیریم و نقیض خروجی این ماشین را به عنوان خروجی ماشین M' قرار دهیم.

M' on input x:
run M_A with input x
if M_A accepts x then **reject** else **accept**

واضح است که ماشین بالا تورینگ‌ـ تصمیم‌پذیر است چون تنها از خروجی یک ماشین تصمیم‌پذیر استفاده می‌کند.

زبان $\bar{C}$ لزوما تورینگ‌ـ تشخیص‌پذیر نیست. چون اگر همواره تورینگ‌ـ تشخیص‌پذیر باشد، در این صورت زبان C ای داریم که هم خودش و هم مکملش تشخیص‌پذیرند. لذا طبق قسمت ۲٫۳، هر دوی زبان‌های $C, \bar{C}$ تصمیم‌پذیر می‌شوند. اما می‌دانیم هر زبان تشخیص‌پذیری لزوماً تصمیم‌پذیر نیست. بنابراین $\bar{C}$ لزوما تشخیص‌پذیر نیست. مثالش هم مثلا $C = A_{TM}$ است که تشخیص‌پذیر هست اما تصمیم‌پذیر نیست پس مکملش یعنی $\overline{A_{TM}}$ تشخیص‌پذیر نیست.

۴ کاهش‌پذیری و دیگر روش‌های اثبات تصمیم‌ناپذیری

۱٫۴

می‌دانیم مسئله A_{TM} تصمیم ناپذیر است. مسئله A_{TM} را به مسائل بخش‌های الف، ب و ج کاهش می‌دهیم و با کمک مسائل بخش‌های الف، ب و ج مسئله A_{TM} را حل می‌کنیم که ثابت می‌کند مسائل این بخش‌ها همگی تصمیم‌ناپذیرند.

الف) برهان خلف می‌زنیم و فرض کنید این زبان تصمیم‌پذیر باشد و ماشین تورینگ تصمیم پذیر G آن را بپذیرد.

ماشین T را بدین شکل تعریف می‌کنیم.

T on input x:
if M accepts ω then **accept** else **reject**

اکنون مسئله A_{TM} را حل‌میکنیم و ماشینی به صورت زیر ارائه می‌دهیم.

on input (M, ω):
if G accepts T then **accept** else **reject**

واضح است که خروجی ماشین بالا که تصمیم‌پذیر هم هست (چون صرفا از خروجی یک ماشین تصمیم‌پذیر استفاده می‌کند)، در صورتی که T همه‌ی رشته‌ها را بپذیرد accept وگرنه reject است. T تنها در صورتی همه‌ی رشته‌ها را می‌پذیرد که M رشته‌ی ω را پذیرفته باشد.

پس با الگوریتم فوق به شکل تصمیم‌پذیر می‌فهمیم که (M, ω) در A_{TM} هست یا نه که تناقض است؛ لذا G تصمیم‌پذیر نیست.

ب) برهان خلف می‌زنیم و فرض کنید این زبان تصمیم‌پذیر باشد و ماشین تورینگ تصمیم‌پذیر G آن را بپذیرد.

ماشین T_2 را بدین شکل تعریف می‌کنیم.

T_2 on input x:
if M accepts ω then **accept** else **reject**

ماشین T_1 هم ماشینی تعریف می‌کنیم که همه‌ی رشته‌های زبان را می‌پذیرد.

T_1 on input x:
accept

اکنون با کمک ماشین G مسئله A_{TM} را حل می‌کنیم و ماشینی به صورت زیر ارائه می‌دهیم.

on input (M, ω):
if G accepts (T_1, T_2) then **accept** else **reject**

واضح است که خروجی ماشین بالا که تصمیم‌پذیر هم هست (چون صرفا از خروجی یک ماشین تصمیم‌پذیر استفاده می‌کند)، در صورتی accept است که $L(T_1)$ زیرمجموعه‌ی $L(T_2)$ باشد. زبان $L(T_1)$ شامل همه‌ی رشته‌ها است و زبان $L(T_2)$ در صورتی که M رشته‌ی ω را بپذیرد همه‌ی رشته‌ها وگرنه تهی است. پس اگر خروجی الگوریتم فوق accept باشد، یعنی (M, ω) در A_{TM} هست وگرنه نیست. لذا توانستیم به شکلی تصمیم‌پذیر مسئله A_{TM} را حل‌کنیم.

پس فرض خلف باطل است و G تصمیم‌پذیر وجود ندارد.

ج) مجدداً از همان ایده‌ی قبلی استفاده می‌کنیم. برهان خلف می‌زنیم و فرض کنید G تصمیم‌پذیر موجود باشد که این زبان را بپذیرد. ماشین T را بدین شکل تعریف می‌کنیم.

T on input x:
if M accepts ω then **accept** else **reject**

زبان ماشین فوق در صورتی که M رشته‌ی ω را بپذیرد شامل همه‌ی رشته و نامتناهی است وگرنه تهی است.

اکنون مسئله‌ی A_{TM} را با کمک G به شکل تصمیم‌پذیر حل می‌کنیم (این الگوریتم فقط از ماشین تصمیم‌پذیر G استفاده می‌کند).

on input (M, ω):
if G accepts T then **accept** else **reject**

واضح است که اگر G ماشین T را بپذیرد یعنی (M,ω) عضو A_{TM} است وگرنه نیست. پس به شکل تصمیم‌پذیری توانستیم مسئله‌ی A_{TM} را حل کنیم که تناقض است. پس فرض خلف باطل است و G وجود ندارد.

۲,۴

الف) از ایده‌ای مشابه مسئله‌ی $Halt$ استفاده می‌کنیم (برای نوشتن پاسخ این سوال از این منبع استفاده شده است).

برهان خلف می‌زنیم و فرض می‌کنیم مسئله P_R تصمیم‌پذیر باشد. پس ماشین تورینگ H موجود است که $H(T)$ برابر $“yes”$ است اگر T ماشین تورینگ دارای ویژگی R باشد و $“no”$ است اگر ماشین تورینگ T دارای ویژگی R نباشد.

$$H(T) = \begin{cases} \text{yes : if machine } T \text{ has property } R \\ \text{no if machine } T \text{ does not have propert } R \end{cases}$$

چون ویژگی R غیربدیهی است پس ماشین تورینگ‌های Y,N موجودند که Y ویژگی R را دارد و N آن را ندارد. لذا در واقع $H(Y) = \text{yes}$ و $H(N) = \text{no}$ است.

اکنون ماشین تورینگ A (در واقع برنامه‌ی A) روی ورودی x را بدین شکل تعریف می‌کنیم.

$$A(x) = \begin{cases} \text{if } H(\text{A}) = \text{yes then } N(x) \\ \text{else } Y(x) \end{cases}$$

این ماشین بدین شکل است که در صورتی که خودش ویژگی R را داشته باشد، در خروجی به ازای ورودی x مقدار $N(x)$ را قرار می‌دهد وگرنه مقدار $Y(x)$ را به عنوان خروجی قرار می‌دهد.

اکنون ماشین A را در نظر بگیرید:

- چنانچه ویژگی R را نداشته باشد $(H(A) = \text{no})$، به ازای هر ورودی x، خروجی $Y(x)$ است. پس در واقع ماشین A همان ماشین Y می‌شود که طبق فرض اولیه‌مان در مورد Y می‌دانیم $H(Y) = \text{yes}$ است که تناقض است.
- چنانچه ویژگی R را داشته باشد $(H(A) = \text{yes})$، به ازای هر ورودی x، خروجی $N(x)$ است. پس در واقع ماشین A همان ماشین N می‌شود که طبق فرض اولیه‌مان در مودر N می‌دانیم $H(N) = \text{no}$ است که تناقض حاصل می‌شود.

لذا ماشین H وجود ندارد و فرض خلف باطل است و مسئله P_R تصمیم‌پذیر نیست.

ب) ویژگی $Not\ Empty$ یا همان «آیا برای ماشین تورینگ T **حداقل یک رشته در $L(T)$ وجود دارد (ویژگی R) یا خیر**» یک ویژگی غیربدیهی است و یک زیرمجموعه‌ی غیربدیهی از ماشین‌های تورینگ آن را دارند. پس بنابر قضیه‌ی رایس، تعیین اینکه آیا ماشین T این ویژگی را دارد یا نه یک مسئله تصمیم‌ناپذیر است.

برای اینکه نشان دهیم R ویژگی غیربدیهی است کافی است یک ماشین تورینگ ارائه دهیم که ویژگی R را داشته باشد و یک ماشین تورینگ ارائه دهیم که ویژگی R را نداشته باشد. ماشین تورینگی که ویژگی R را داشته باشد، می‌تواند ماشین تورینگ معادل با

DFA پذیرنده‌ی زبان «کلمات متشکل از ۰ و ۱ که تعداد ۱ ها زوج است» باشد و ماشین تورینگی که ویژگی R را نداشته باشد می‌تواند ماشین تورینگ ردکننده‌ی همه‌ی ورودی‌ها باشد (ماشین تورینگ فقط دارای رأس $reject$). پس ویژگی R غیربدیهی است و قضیه‌ی رایس در مورد آن برقرار بوده و مسئله P_R تصمیم‌ناپذیر است.