نظریه علوم کامپیوتر

نیم‌سال دوم ۱۳۹۹-۱۴۰۰

مدرس: دکتر جواد ابراهیمی، دستیار آموزشی: ساجد کریمی

دانشکده علوم ریاضی

تمرین سری دوم

نام و نام‌خانوادگی: امید یعقوبی

شماره دانشجویی: -

پرسش ۱

دو چیز را باید نشان دهیم:

$$1.\ A \leq_m 0^*1^* \Rightarrow \text{ A is Decidable}$$

$$2.\ \text{A is Decidable} \Rightarrow\ A \leq_m 0^*1^*$$

برای ۱ از آن جا که 0^*1^* یک *Regular Expression* است، پس منظم است. و از آن جا که هر زبان منظمی تصمیم پذیر است، 0^*1^* تصمیم پذیر است. اگر $A \leq_m 0^*1^*$ آن گاه بر اساس قضیه [1, 5.22, p. 236] A نیز تصمیم پذیر است. البته می شد به تفضیل توضیح داد که چون تابع محاسبه پذیر $f(x)$ وجود دارد که هر *Instance* از مساله A را به نمونه ای از مساله 0^*1^* در زمان متناهی می برد، به شکلی که اگر $x \in A$ آن گاه $f(x) \in 0^*1^*$ و برعکس، و همچنین برای 0^*1^* هم *DFA* داریم. پس A is Decidable.

■

برای ۲ ابتدا f را این گونه تعریف می کنیم:

$$f(w) := \begin{cases} 01, & \text{if } w \in A \\ 10, & \text{if } w \notin A \end{cases}$$

حال تابع f در صورتی که عضویت w در A تصمیم پذیر باشد، محاسبه پذیر خواهد بود و پس از زمان متناهی آن را محاسبه کرده و متوقف می شود. چراکه برای طراحی ماشینی که f را محاسبه کند، از آن جا که A تصمیم پذیر است، پس برای آن یک *Decider* وجود دارد، اگر فرض کنیم *Decider* آن R نام دارد، کافیست ابتدا R را روی w اجرا کرده سپس دو حالت داریم. اول اینکه R ورودی را *Accept* کند. یعنی $w \in A$ پس نوار را خالی کرده و 01 را روی آن می نویسم. در غیر این صورت یعنی R ورودی را *Reject* کرده، که مانند پیش این بار پس از پاک کردن *Tape* روی آن 10 می نویسیم. پس f یک تابع *Computable* است. حال داریم:

$$w \in A \Leftrightarrow f(w) \in 0^*1^*$$

در نتیجه:

$$A \leq_m 0^*1^*$$

و قضیه اثبات می شود.

■

پرسش ۲

دو چیز را باید نشان دهیم:

1. $A \leq_m A_{TM} \Rightarrow$ A is Recognizable
2. A is Recognizable $\Rightarrow$ $A \leq_m A_{TM}$

برای ۱ ابتدا باید نشان دهیم A_{TM} تشخیص پذیر است.

هر TM قابل شبیه سازی توسط *Universal Turing Machine* است. چراکه TMها مجموعه ای از توابع انتقالند. بنابراین هر TM با مجموعه ای از پنج‌ـتایی ها که اندازه ای محدود دارند قابل نمایش است. هر تابع δ به شکل زیر دارای یک پنج تایی متناظر است:

$$\delta(q_x, a) = (q_y, b, D) \leftrightarrow (q_x, a, q_y, b, D)\ s.t\ D \in \{L, R\}$$

پس هر TM مجموعه ای از پنج‌ـتایی هاست. از آنجا که UTM کد یک ماشین تورینگ را که به شکل مجموعه ای محدود از پنج‌ـتایی ها هستند، به همراه w که رشته ای محدود است، گرفته و اجرا می کند. با توجه به این حقیقت که خود UTM یک TM است. پس کد هر ماشین از یک نمونه ورودی A_{TM} نیز به همراه هر w قابل شبیه سازی در UTM است. پس از اجرا روی هر ورودی از آنجا که UTM یک TM است در سه حالت زیر قرار می گیرد:

$$A_{TM}\ on\ UTM = \begin{cases} Accept \\ Reject \\ Loop \end{cases}$$

پس A_{TM} تشخیص دهنده دارد و Recognizable است.

حال اگر $A \leq_m A_{TM}$ پس $(\forall w)\ (w \in A) \Leftrightarrow (f(w) \in A_{TM})$ و f یک تابع محاسبه پذیر است. یعنی در زمان محدود پایان میپذیرد. به عبارتی f در زمان محدود هر instance از A را به A_{TM} می برد:

$$f : x \mapsto <M, w>\ s.t\ x \in A \Leftrightarrow <M, w> \in A_{TM}$$

از آن جا که هر نمونه از ورودی A قابل اجرا توسط *Recognizer* مساله A_{TM} است. اگر نام $A_{TM}Recognizer$ را R_1 بگذاریم. می توانیم *Recognizer* برای A بسازیم. می دانیم که f محاسبه پذیر است. حال *Recognizer* برای A با نام R_2 می سازیم. به این شکل که R_2 برای هر w ابتدا $f(w)$ را تولید کرده (*تابع محاسبه پذیر است، پس طبق تعریف 5.17* [۱] $f(w)$ *را تولید کرده و متوقف می شود.*) سپس آن را به R_1 که تشخیص گر A_{TM} است می دهیم. از آن جا که R_1 تشخیص گر است. پس در نهایت در سه وضعیت $\{Accept, Reject, Loop\}$ خواهیم بود. پس R_2 تشخیص گری برای A است.
همچنین به شکل خلاصه، در یک جمله، می توانستیم بگوییم از آن جا که بر اساس [1, 4.11, p. 202] A_{TM} تشخیص پذیر است. پس طبق قضیه 5.28 [۱] A نیز تشخیص پذیر است.
درنهایت اینکه A is Recognizable.

■

برای ۲ اگر A تورینگ تشخیص پذیر باشد آن گاه برایش TM با نام M وجود دارد که A را تشخیص می دهد. حال تابع f را این گونه تعریف می کنیم که $<M, w>$ را روی نوار نوشته و هالت کند. یعنی:

$$f(w) := <M, w>$$

پس اگر w در $L(M)$ باشد A_{TM} هالت می کند و برعکس (*بنا به تعریف* A_{TM}) . پس داریم:

$$w \in A \Leftrightarrow <M, w> \in A_{TM}$$

در نتیجه:

$$A \leq_m A_{TM}$$

و قضیه اثبات می شود.

■

پرسش ۳

در این جا یک مجموعه تحت الفبای $Unary$ معرفی می کنیم که ناشماراست و پس از آن یک زبان تحت الفبای $\{1\}$ معرفی می‌کنیم که تصمیم ناپذیر است.

مجموعه‌ی ناشمارا ابتدا نشان می دهیم که بین $\mathbb{N}$ و $\{1\}^*$ یک $Bijection$ وجود دارد:

$$for\ every\ i \in \mathbb{N}\ ,\ f(i) := 1^i$$

مشخص است که f یک $Bijection$ است. پس $\{1\}^* \leftrightarrow \mathbb{N}$. *(اگر خیلی حساس باشیم، اثبات این تناظر ساده و با استقراست.)*

بر اساس قضیه $Cantor$ می دانیم که $|\mathcal{P}(\mathbb{N})| = card(\mathbb{R})$ یا به عبارتی مجموعه همه ی زیرمجموعه های $\mathbb{N}$ ناشمارست. و می دانیم که $\{1\}^* \leftrightarrow \mathbb{N}$ پس $|\mathcal{P}(\{1\}^*)| = card(\mathbb{R})$ و گویی اثبات کردیم که $\mathcal{P}(\{1\}^*)$ ناشماراست. حال اگر مجموعه S اینگونه تعریف کنیم:

$$S = \mathcal{P}(\{1\}^*)$$

آن گاه $|S|$ ناشماراست. پس S یک مجموعه‌ی $Uncountable$ تحت الفبای $Unary$ می باشد. *(ولی این مجموعه در تعریف یک زبان نمی‌گنجد.)*

زبان تصمیم‌ناپذیر تحت الفبای $\{1\}$: ابتدا نشان می دهیم بین $\{0,1\}^*$ و $\{1\}^*$ یک $Bijection$ وجود دارد:

$$for\ every\ x \in \{0,1\}^*\ ,\ f(i) := 1^{bin(1x)}$$

که $bin(x)$ عدد متناظر رشته ی باینری x است. توجه شود که اگر به جای تابع بالا از $1^{bin(x)}$ استفاده شود، دیگر تابع $Bijection$ نبود، چراکه همانطور که می بینید x های متمایز زیر همه به یک مقدار می روند که نقض تناظر یک به یک است:

$$1^{bin(1)} = 1^{bin(01)} = 1^{bin(001)} = ... = 1^{bin(0^n1)}$$

ولی واضح است که $1^{bin(1x)}$ یک تناظر یک به یک $(Bijection)$ است:

$$\begin{gathered} f(1) = 1^{bin(11)} = 1^3 \neq \\ f(01) = 1^{bin(101)} = 1^5 \neq \\ f(001) = 1^{bin(1001)} = 1^9 ... \end{gathered}$$

هر $<M, w>$ می تواند یک توصیف باینری باشد. به این ترتیب $A_{TM} = \{<M, w>|\ M\ accpets\ w\}$ نیز دارای یک $Binary\ Encode$ است. در نتیجه، اگر اعضای A_{TM} را به صورت رشته های باینری تصور کنیم، آن گاه $A_{TM} \subset \{0,1\}^*$ از آنجا که $\{1\}^* \leftrightarrow \{0,1\}^*$ پس اگر $UnaryA_{TM}$ را این گونه تعریف کنیم:

$$UnaryA_{TM} := \{f(x) | x \in A_{TM}\}$$

آن گاه هر عضو از A_{TM} دقیقا به یک عضو از $UnaryA_{TM}$ می رود. حال داریم:

$$x \in A_{TM} \leftrightarrow f(x) \in UnaryA_{TM}$$

اثبات تصمیم ناپذیری $UnaryA_{TM}$: اگر $UnaryA_{TM}$ تصمیم پذیر باشد، پس $Decider$ برای آن وجود دارد. فرض کنید این $Decider$ ماشین R باشد. حال برای A_{TM} یک $Decider$ با نام S می سازیم:

$$S(<M, w>) := \begin{cases} \text{Compute } f(<M, w>) \\ \text{Run TM } R \text{ on input } f(<M, w>) \\ \text{If } R \text{ rejects, } \textit{reject.} \\ \text{If } R \text{ accepts, } \textit{accept.} \end{cases}$$

می دانیم که محاسبه ی $f(x)$ برای هر $x \in \{0,1\}^*$ متوقف می شود. چراکه به ازای هر بیت با مقدار 1 روی نوار کافیست به تعداد ارزش آن که برابر 2^i است، 1 بنویسیم. برای مثال اگر اندیس 0 در x باینری برابر 1 بود، یک 1 کپی می کنیم. در صورتی که $x[i] = 1$ کافیست 2^i عدد 1 کپی کنیم. پس از اتمام رشته هم، اگر رشته در اندیس k تمام شده بود 2^{k+1} تا 1 دیگر برای الحاق 1 به x کپی می کنیم. ارزش 2^i را نیز این گونه حساب می کنیم: هر بار پس از اتمام پردازش کپی در اندیس i، این ارزش دو برابر شود، یعنی 2.2^i حساب شود، که برای ارزش بیت بعدی همان 2^{i+1} است که باید باشد. عمل کپی کردن و عمل 2 برابر کردن، برای رشته ی متناهی x، هر دو تمام شدنی است. در نتیجه محاسبه ی $f(x)$ تمام شدنی است. در نتیجه مرحله اول پس از اتمام زمان متناهی پایان می پذیرد.
اجرای R روی ورودی x که x همان $<M,w>$ است هم به دلیل فرض $Decider$ بودن R در زمان متناهی تمام می شود. پس S حتما $Halt$ می کند. به این ترتیب S یک $Decider$ برای A_{TM} است، که این تناقض است $\perp$. در نتیجه، $UnaryA_{TM}$ تحت الفبای $\Sigma = \{1\}$ ، تصمیم ناپذیر می باشد.

پرسش ۴

آ)

اگر L_k را تعریف زیر بگیریم:

$$L_k = \{\langle M, w, k\rangle | M \text{ uses at most k tape squares on input } w\}$$

آن گاه L_k تصمیم پذیر است. برای آن یک $Decider$ طراحی می کنیم. برای ماشین M اگر $|Q|$ تعداد حالات این ماشین به احتساب حالت توقف باشد، و $|\Gamma|$ تعداد الفبای نوار با احتساب کارکتر $Blank$ باشد، آن‌گاه تعداد $Configuration$ های ممکن قابل محاسبه و برابر $\alpha = |Q|.|\Gamma|^k.k$ می‌باشد. برای توضیح بیشتر، هد می‌تواند در k موقعیت متمایز قرار گیرد، همچنین هر خانه‌ی استفاده شده می‌تواند $|\Gamma|$ حالت متفاوت داشته باشد، و در نهایت حالت فعلی FSM داخلی TM می‌تواند در $|Q|$ حالت متفاوت قرار گیرد، که تعداد هر یک از این سه نیز از دیدگاه ترکیبیاتی مستقل بوده و در نتیجه در هم ضرب می‌شود. در نهایت اینکه تعداد کل $Configuration$ های ممکن، قابل محاسبه، و همچنین مستقل از طول ورودی، و برابر α است.

حال ماشین M را روی ورودی‌اش حداکثر α قدم [1] اجرا می‌کنیم. همچنین تعداد خانه‌های استفاده شده تا به حال را نیز در نواری مجزا نگه می‌داریم (*فرض کنید روی یک نوار شبیه سازی M روی w را انجام می‌دهیم و روی نوار دیگر اطلاعات کنترلی نظیر تعداد $Step$ های تا به حال طی شده و تعداد خانه‌های تا به حال استفاده شده را نگه می‌داریم. می‌دانیم که شبیه‌سازی ماشین M فرایندی الگوریتمیک بوده و پس از هر قدم که در شبیه‌سازی برمی‌داریم، مجازیم که اطلاعات کنترلی مورد نظرمان را محاسبه و ذخیره کنیم، به شکلی که $Configuration$ فعلی شبیه‌سازی M روی w دست نخورد. برای تاکید بیشتر، یادمان باشد که مجموعه توابع دلتای ماشین شبیه‌ساز، برای مثال UTM مستقل از توابع دلتایی است که به عنوان ورودی در قالب $\langle M\rangle$ دریافت و شبیه‌سازی می‌شود. در نتیجه در بین شبیه‌سازی هر تابع دلتای $\langle M\rangle$ ما مجازیم که با توابع اصلی ماشین شبیه ساز، در اینجا UTM ، کارهای مستقلی از شبیه‌سازی انجام دهیم.*) اگر تا α قدم طول خانه‌های استفاده شده، که آن‌ها را در نوار کنترلی نگه می‌داریم، بیش از k شد، به راحتی $Reject$ می‌کنیم. اگر به k نرسیده بود و α قدم را پشت‌سر گذاشتیم، بر اساس اصل لانه کبوتری، چون قدم‌هایی که تا به حال داشتیم، از کل حالات ممکنه برای $Configuration$ ها، یعنی α بیشتر بوده، حتما در یک قدم، برای مثال قدم i ام که آن را $Step_i$ می‌نامیم. $Configuration$ با نام C_x وجود داشته که این $Configuration$ در قدمی متفاوت مانند $Step_j$ تکرار شده، و در نتیجه حتما در $Loop$ افتاده‌ایم. در نهایت از آنجا که می‌دانیم در $Loop$ هستیم و تا به حال نیز بیش از k خانه حافظه استفاده نکرده‌ایم، پس از این پس نیز بیش از k خانه استفاده نخواهیم کرد. چراکه اگر فرض کنیم در آینده‌ای، این استفاده از حافظه بیشتر می‌شود، یعنی $Configuration$ در آینده خواهیم داشت که تا به حال آن را نداشتیم، از آن‌جا که کل تعداد $Configuration$ ها برابر α بوده، این فرض منجر به تناقض می‌شود. برای تاکید توجه شود که حداکثر استفاده از حافظه‌ی نوار، از دنباله‌ی $Configuration$ ها قابل محاسبه است. به این نحو که برای هر $C_i = (q_i, u\underline{a}v)$ که $q_i \in Q, u, v \in \Gamma^*, a \in \Gamma$ اگر پوزیشن‌های محتوای روی نوار را برای هر C_i با

$$Pos(C_i) = \{\ Positions\ of\ used\ tape\ squares\ in\ u\underline{a}v\}$$

نشان دهیم، یعنی پوزیشنهای اشغال شده در نوار را برای C_i درون مجموعه‌ای با نام $Pos(C_i)$ قرار دهیم. کافیست اجتماع این موقعیتهای تا به حال استفاده شده را حساب کنیم:

$$UsedCells = \bigcup_{C_i \in \text{Configurations}} Pos(C_i)$$

البته در اینجا ورودی روی نوار را بدون از دست دادن عمومیت $(w.l.o.g)$ در خانه‌های استفاده شده حساب کردیم. حال مشخص است که اگر در لحظه‌ای، پوزیشن جدیدی از خانه‌های نوار برای اولین بار استفاده شود، اگر موقعیت این خانه‌ی به تازگی مورد استفاده قرار گرفته شده را $posX$ نام‌گذاری کنیم، روشن است که $posX \in Pos(C_i)$ پس در محاسبه‌ی طول خانه‌های تا به حال استفاده شده لحاظ شده است. حال مشخص است که حد بالای استفاده از حافظه برابر $MaxMemory = Length(UsedCells)$ یا همان اندازه‌ی $UsedCells$ می‌باشد. توجه کردیم که این محاسبه روی دنباله‌ی $Configuration$ ها بود. در نتیجه اگر وارد $Loop$ شویم، در واقع یک زیردنباله‌ی تکراری را شروع کرده‌ایم، و می‌دانیم که اضافه کردن هر زیردنباله‌ی تکراری به دنباله‌ی $Configuration$ ها، مقدار محاسبه شده برای حد بالای استفاده

[1] Step

از خانه‌های نوار را افزایش نمی‌دهد. پس امکان ندارد به $UsedCells$ با اضافه کردن زیردنباله‌ی تکراری از *Configuration* ها، عضو جدیدی اضافه شود، و در نتیجه اندازه‌ی آن هم بی‌تغییر می‌ماند. حال به راحتی *Accept* می‌کنیم. چراکه می‌دانیم تا به حال بیش از k خانه از نوار استفاده نشده و نشان دادیم که پس از این نیز، بخاطر آن که در *Configuration* های تکراری و *Loop* افتاده‌ایم، بیش از این استفاده نخواهد کرد.

Decider که برای L_k تعریف می‌کنیم، یک تورینگ دو نواره است (*می‌دانیم که هر تورینگ چندنواره از نظر قدرت محاسبه‌ای برابر با تورینگ استاندارد است*).

در نوار اول شبیه‌سازی را انجام می‌دهیم و در نوار دوم اطلاعات کنترلی را نگه می‌داریم. اطلاعات کنترلی که نگهداری می‌شود برابر است با: $UsedCells$ که مجموعه‌ای است از پوزیشن‌های تا به حال استفاده شده از نوار به این شکل که پس از هر قدم از شبیه‌سازی در صورت استفاده شدن از یک خانه‌ی جدید حافظه آپدیت می‌شود، $Length(UsedCells)$ که طول این مجموعه است، که با آپدیت $UsedCells$ این متغیر نیز آپدیت می‌شود، و در نهایت $CurrentStep$ که شماره‌ی قدم فعلی در شبیه‌سازی را نگه می‌دارد، و با هر قدم شبیه‌ساز یک واحد به آن اضافه می‌شود.

حال *Decider* برای L_k را با نام M_k به صورت فرمال تعریف می‌کنیم:

$$M_k(\langle M, w, k\rangle) := \begin{cases} \text{Callcuate } \alpha \\ \\ \text{Simulate } M \text{ on } w \text{ for at most } \alpha \text{ steps} \\ \\ \text{For each } Step_i = \begin{cases} \text{1- Callcuate } Pos(C_i) \text{ for the Current configuration} \\ \text{2- Update } UsedCells \text{ with the } UsedCells \cup Pos(C_i) \\ \text{3- Update } Length(UsedCells) \\ \text{4- Update } CurrentStep \text{ with the } CurrentStep + 1 \\ \text{5- If } Length(UsedCells) > k \ \textit{reject} \\ \text{6- If } Length(UsedCells) \leq k \text{ and } CurrentStep > \alpha \ \textit{accept} \\ \text{7- If } Length(UsedCells) \leq k \text{ and } M \text{ accepts, } \textit{accept} \\ \text{8- If } Length(UsedCells) \leq k \text{ and } M \text{ rejects, } \textit{accept} \end{cases} \\ \\ \end{cases}$$

همانطور که مشاهده می‌شود. در ماشین M_k ابتدا $\alpha = |Q|.|\Gamma|^k.k$ براساس ورودی محاسبه می‌شود. حال می‌خواهیم، ماشین M را روی w تا حداکثر α قدم شبیه سازی کنیم. اگر در این بین بیش از k خانه‌ی نوار استفاده کرد، به سادگی Reject می‌کنیم. اگر تا قدم α از این مقدار خانه کمترمساوی استفاده کرد، به راحتی Accept می‌کنیم.

توصیف فرمال این ماشین، در مرحله اول به ازای هر قدم در شبیه‌سازی با نام $Step_i$ ، ابتدا اگر روی خانه‌ی فعلی *Blank* بوده و نوشته شده، این پوزیشن را با مجموعه‌ی $UsedCells$ اجتماع می‌کند. به این شکل که اگر این موقعیت پیش از این در $UsedCells$ نبود، آن را اضافه می‌کند. اگر پوزیشن جدید به $UsedCells$ اضافه شد، $Length(UsedCells)$ با مقدار $Length(UsedCells) + 1$ آپدیت می‌شود. همچنین به ازای این قدم برداشته شده در شبیه‌سازی $CurrentStep =$ $CurrentStep + 1$ می‌شود. در حقیقت $CurrentStep$ در $Step_i$ برابر i است. حال اگر مقدار حافظه مصرفی، یعنی $Length(UsedCells)$ بیش از k بود، یعنی ماشین بیش از k خانه استفاده کرده و Reject می‌کنیم. در غیر این صورت اگر این مقدار کمترمساوی k بود، تعداد قدم برداشته با α چک می‌شود. اگر بیش از α قدم برداشتیم accept می‌کنیم. چرا که پس از این *Configuration* جدیدی نخواهیم داشت. اگر کمتر از α قدم برداشته باشیم، ممکن است کار شبیه‌ساز پایان یابد و ماشین شبیه‌سازی شده در موقعیت Accept یا Reject قرار گرفته باشد. در هر دو صورت ماشین کمترمساوی k خانه استفاده کرده و در نتیجه Accept می‌کنیم.

توجه شود که حداکثر α قدم شبیه‌سازی انجام می‌شود، که α یک مقدار محدود است. همچنین به ازای هر قدم به تعداد ثابت کار انجام می‌شود. در نتیجه ما یک *Decider* برای L_k ساختیم. پس L_k تصمیم‌پذیر است.

ب) اگر $L_{FiniteMem}$ را تعریف زیر بگیریم:

$$L_{FiniteMem} = \{\langle M, w\rangle | Does\ there\ exist\ a\ k\ s.t.\ M\ uses\ at\ most\ k\ tape\ squares\ on\ input\ w\}$$

نشان می‌دهیم که اگر $L_{FiniteMem}$ تصمیم پذیر باشد، $Halting\ Problem$ که آن را با HP نشان می‌دهیم نیز تصمیم پذیر خواهد بود. از این تناقض نتیجه می‌گیریم که $L_{FiniteMem}$ تصمیم‌پذیر نیست.

فرض می‌کنیم که برای $L_{FiniteMem}$ یک $Decider$ با نام $M_{FiniteMem}$ وجود دارد. همچنین برای L_k نیز با تعریف انجام شده در بخش آ همین پرسش در این قسمت یک $Decider$ با نام M_k تعریف کردیم. در نتیجه از این پس می‌توانیم به ازای هر k بگوییم که آیا این TM از k حافظه بیشتر مصرف می‌کند یا نه. همچنین توسط $M_{FiniteMem}$ می‌توانیم بگوییم که آیا این k وجود دارد یا خیر.

ابتدا ادعا می‌کنیم، که اگر $M_{FiniteMem}$ رشته را $Reject$ کند. یعنی در صورتی که این k وجود نداشته باشد، حتما در $Loop$ افتاده‌ایم. اثبات این ادعا با برهان خلف هست. فرض کنیم که k وجود ندارد و ماشین نیز در $Loop$ نیفتاده است. پس ماشین به ازای رشته‌ی w حتما در یکی از حالات Accept یا Reject، $Halt$ کرده است. در نتیجه دنباله‌ای از $Configuration$ ها به شکل زیر قابل تصور است:

$$Configurations = C_0, C_1, ..., C_j$$

که در اینجا C_0 برابر $Initial\ Configuration$ و برای هر $i \neq 0$ حتما C_i به درستی از C_{i-1} استنتاج شده است. همچنین آخرین $Configuration$ یعنی C_j حتما یا $Accepting\ Configuration$ یا $Rejecting\ Configuration$ می‌باشد. حال مجموعه‌ی $UsedCells$ مانند بخش پیش تعریف می‌شود:

$$UsedCells = \bigcup_{C_i \in \text{Configurations}} Pos(C_i)$$

که

$$Pos(C_i) = \{\ Positions\ of\ used\ tape\ squares\ in\ u\underline{a}v\}$$

می‌باشد. آن‌گاه حداکثر خانه‌ی حافظه‌ی مصرف شده، درست مانند بخش آ همین سوال، برابر

$$MaxMemory = Length(UsedCells)$$

می‌باشد. پس $k = MaxMemory$ که این تناقض است. پس اگر k وجود نداشته باشد، حتما در $Loop$ افتاده‌ایم.

اگر k وجود داشته باشد، می‌توانیم آن را به وسیله‌ی M_k که در این قسمت تعریف شده، پیدا کنیم. به این شکل که $M_k(\langle M, w, i\rangle)$ را به ازای $i \in \underbrace{\{1, 2, \ldots, k\}}_{N_k}$ اجرا می کنیم. از آن‌جا که k وجود دارد. پس $|N_k|$ is $finite$ همچنین از آنجا که M_k یک $Decider$ است. به ازای هر i حتما در زمان متناهی متوقف خواهد شد. در نتیجه در زمان متناهی به k خواهیم رسید، که $M_k(\langle M, w, k\rangle)$ جواب Accept داشته باشد. یعنی اجرای ماشین M روی w به حداکثر k خانه‌ی حافظه احتیاج دارد. حال که حدبالای استفاده از حافظه‌ی نوار را در M به ازای ورودیِ w می‌دانیم، کافیست M را روی w تا α قدم شبیه‌سازی کنیم. که $\alpha = |Q|.|\Gamma|^k.k$ می‌باشد. اگر در کمترمساوی α قدم رشته‌ی w، Reject یا Accept شد. یعنی ماشین M روی w، $Halt$ می‌کند. اگر شبیه‌سازی بیش از α قدم طول کشید، بر اساس اصل لانه کبوتری، در دو قدم متمایز $Step_i$ و $Step_j$ یک $Configuration$ تکراری با نام C_x داشته‌ایم، پس بر اساس استدلال مشابه این قسمت حتما در Loop افتاده‌ایم. در نتیجه ماشین $Halt$ نخواهد کرد. اکنون برای همه‌ی حالات برای $Halt$ در قدمهای متناهی تصمیم گرفتیم. پس برای HP یک $Decider$ ساختیم. که این تناقض است. در صفحه‌ی بعد تعریف فرمال این $Decider$ را داریم.

تعریف فرمال $Decider$ تعریف شده برای HP:

$$M_{HP}(\langle M,w\rangle) := \begin{cases} \text{Run TM } M_{FiniteMem} \text{ on input } \langle M,w\rangle \\ \\ \text{If } M_{FiniteMem} \text{ rejects, } \textit{reject} \\ \\ \text{If } M_{FiniteMem} \text{ accepts, do:} \\ \\ \qquad \text{For each } i \in \{1,\dots,k\} = \begin{cases} \text{Run TM } M_k \text{ on input } \langle M,w,i\rangle \\ \text{If } M_k \text{ accepts let } k=i \text{ and break} \\ \text{If } M_k \text{ rejects let } i=i+1 \text{ and loop} \end{cases} \\ \\ \qquad \text{Callcuate } \alpha \\ \\ \qquad \text{Simulate } M \text{ on } w \text{ for at most } \alpha \text{ steps} \\ \\ \qquad \text{For each } Step_i = \begin{cases} \text{1- Update } CurrentStep \text{ with the } CurrentStep+1 \\ \text{2- If } CurrentStep > \alpha \ \textit{reject} \\ \text{3- If } CurrentStep \leq \alpha \text{ and } M \text{ accepts, } \textit{accept} \\ \text{4- If } CurrentStep \leq \alpha \text{ and } M \text{ rejects, } \textit{accept} \end{cases} \\ \end{cases}$$

همانطور که دیده می‌شود، با فرض $Decider$ داشتن برای $L_{FiniteMem}$ توانستیم برای HP یک تصمیم‌گیر با نام M_{HP} بسازیم. همانطور که گفته شد، در مرحله‌ی اول $M_{FiniteMem}$ را روی ورودی $\langle M,w\rangle$ اجرا می‌کنیم. اگر این ماشین روی این ورودی Reject کرد، بر اساس استدلال صفحه‌ی پیش، حتما در Loop افتاده‌ایم، در نتیجه ماشین هالت نخواهد کرد، پس M_{HP} نیز Reject می‌کند. اگر $M_{FiniteMem}$ روی ورودی $\langle M,w\rangle$ در حالت Accept توقف کرد، آنگاه در مرحله‌ی بعد، k را پیدا می‌کنیم. از آنجا که k وجود دارد، با افزایش یک مرحله‌ای i در ورودیِ M_k (*ماشین تورینگ بخش آ همین سوال*) در متناهی قدم k پیدا خواهد شد. از آنجا که M_k نیز یک $Decider$ است، پس برای هر i هم در متناهی قدم تصمیم گرفته می‌شود و به علت اطمینان از وجود k پس از متناهی قدم به k خواهیم رسید. پس از پیدا کردن k ابتدا $\alpha = |Q|.|\Gamma|^k.k$ را محاسبه می‌کنیم. بر اساس استدلال آمده در همین پرسش، بخش آ، این قسمت می‌دانیم که حداکثر تعداد $Configuration$ های ممکن برابر α است. در نتیجه اگر ماشین M را روی w تا α قدم اجرا کنیم. اگر این ماشین تا به حال هالت نکرده باشد، از این پس نیز هالت نمی‌کند. چراکه $Configuration$ تکراری داشتیم و این نشانه‌ی Loop است. پس Reject می‌کنیم. اگر تعداد قدمهای شبیه‌سازی کمترمساوی α باشد و ماشین شبیه سازی شده در یکی از حالتهای Reject یا Accept بیفتد، یعنی ماشین هالت کرده، پس Accept می‌کنیم.

نشان دادیم که M_{HP} یک $Decider$ برای HP است، که این یک تناقض است. پس $L_{FiniteMem}$ تصمیم‌ناپذیر یا $Undecidable$ است.

پ) -

ت) اگر L_{f_c} را تعریف زیر بگیریم:

$$L_{f_c} := \left\{ \text{ on input } w : \begin{cases} \text{If } |Loop\big(L(M_{1000})\big)| \text{ is } finite \text{ Output: } |Loop\big(L(M_{1000})\big)| \\ \text{If } |Loop\big(L(M_{1000})\big)| = \infty \text{ Output: } -1 \end{cases} \right.$$

همچنین زبان $Loop(L(M))$ اینگونه تعریف می شود:

$$Loop(L(M)) := \{w|M \text{ Does not Halt on input } w\}$$

می‌دانیم که برای هر مجموعه S یا $|S|$ متناهی است، یا نامتناهی. همینطور اندازه‌ی زبان $Loop\big(L(M_{1000})\big)$ یا متناهی است، یا نامتناهی. در نتیجه L_{f_c} یا برابر $\{k\}$ است که $k \in \mathbb{N}$ ،یا آن که روی نامتناهی رشته متوقف نمی‌شود که در این صورت برابر است با $\{-1\}$ ، در هر دو صورت برای این زبان یک تورینگ محاسبه کننده وجود دارد. چراکه در هر دو صورت ما تابع ثابت $f(w) = c$ را حساب می‌کنیم، که $c \in \{-1\} \cup \mathbb{N}$ می‌باشد. همچنین می‌دانیم که تابع ثابت محاسبه پذیر است. چراکه بی‌توجه به آرگومان ورودی‌اش مقدار c را چاپ می‌کند.

دو حالت وجود دارد:

1. $|Loop\big(L(M_{1000})\big)| \text{ is finite} \Rightarrow f(w) = k, k \in \mathbb{N}$
2. $|Loop\big(L(M_{1000})\big)| = \infty \Rightarrow f(w) = -1$

در هر دو حالت برای c $f : \Sigma^* \to$ یک تورینگ محاسبه کننده داریم. در اولی ابتدا ورودی را $Ignore$ کرده و روی نوار مقدار ثابت k را که عددی طبیعیست می‌نویسد. در حالت دوم ابتدا ورودی را $Ignore$ کرده و روی نوار مقدار ثابت -1 را می‌نویسد. از آنجا که در دنیایی قطعی زندگی می‌کنیم که در آن اندازه‌ی رشته‌هایی که این ماشین M_{1000} روی آن توقف نمی‌کند بی تغییر است، پس غیر این دو حالت حالتی ممکن نیست. پس این زبان تورینگ محاسبه پذیر است.

ث) اگر L_{tat} را تعریف زیر بگیریم:

$$L_{tat} := \left\{ \text{ on input } w : \begin{cases} \text{If } \langle M_{1000}\rangle \in Halt\big(L(M_{1000})\big) \text{ Accept} \\ \text{If } \langle M_{1000}\rangle \notin Halt\big(L(M_{1000})\big) \text{ Reject} \end{cases} \right.$$

همچنین زبان $Halt(L(M))$ اینگونه تعریف می شود:

$$Halt(L(M)) := \{w|M \text{ Halts on input } w\}$$

دو حالت وجود دارد:

1. $\langle M_{1000}\rangle \in Halt\big(L(M_{1000})\big) \Rightarrow L_{tat} = \Sigma^*$
2. $\langle M_{1000}\rangle \notin Halt\big(L(M_{1000})\big) \Rightarrow L_{tat} = \emptyset$

از آن جا که در دنیایی قطعی زندگی می‌کنیم، یا $M_{1000}(\langle M_{1000}\rangle)$ متوقف می‌شود، یا متوقف نمی‌شود. در هر دو حالت زبان L_{tat} منظم و تصمیم‌پذیر است. در حالت اول $L_{tat} = \Sigma^*$ که منظم و تصمیم‌پذیر است. در حالت دوم نیز $L_{tat} = \emptyset$ که منظم و تصمیم‌پذیر است.

ج) اگر $L_{EpsilonLeft}$ را تعریف زیر بگیریم:

$$L_{EpsilonLeft} = \{\langle M\rangle | M \text{ ever moves left while computing } \epsilon\}$$

آن‌گاه $L_{EpsilonLeft}$ تصمیم‌پذیر است. برای آن یک *Decider* با نام $M_{EpsilonLeft}$ طراحی می‌کنیم.

ابتدا ادعا می‌کنیم که هر TM که حرکت به چپ دارد *به طور کلی به ازای هر* w این حرکت تا حداکثر $|w| + |Q| + 1$ امین قدم شبیه‌سازی انجام می‌شود [۲] . در حالت خاص برای $w = \epsilon \Rightarrow |w| = 0$ تا $|Q| + 1$ امین قدم شبیه سازی اگر ماشین حرکتی به چپ داشته باشد، این حرکت را انجام داده است. برای درک شهودی، بزرگترین مسیر محاسبه‌ای را که در آن برای رشته‌ی ϵ حرکت به چپ اتفاق افتاده در نظر بگیرید:

$$ConfPath = \{\underbrace{(q_0, \square\square\dots)}_{C_0} \underbrace{\rightarrow}_{\delta(q_0,\square)} \underbrace{(q_{i_1}, \square\square\dots)}_{C_1} \underbrace{\rightarrow}_{\delta(q_{i_1},\square)} \underbrace{(q_{i_2}, \square\square\dots)}_{C_2} \dots \underbrace{\rightarrow}_{\delta(q_{i_{n-1}},\square)} \underbrace{(q_{i_n}, \square\square\dots)}_{C_n}\}$$

ابتدا ادعا می‌کنیم، تا رسیدن به اولین حرکت به چپ برای ورودی ϵ حداکثر حالات ممکنه برای تابع δ برابر $|Q|$ است. چراکه آرگومان دوم این تابع همیشه برابر $\square$ خواهد بود. پس تنها آرگومانی که تغییر می‌کند آرگومان اول یعنی تعداد حالات است. که این می‌تواند $|Q|$ حالت مختلف داشته باشد. ابتدا به صورت ضمنی اثبات می‌کنیم که امکان ندارد که بدون حرکت به چپ به ازای ورودی ϵ آرگومان دوم چیزی غیر از $\square$ باشد. فرض کنیم که حرکت به چپ نداشتیم و آرگومان دوم چیزی غیر از $\square$ بوده است. از آنجا که ورودی *blank* است. یعنی یکی از کارکترهایی را می‌خوانیم که پیشتر آن را نوشته بودیم. اگر روی نوار چیزی را نوشته باشیم و به راست رفته باشیم آن را رد کرده‌ایم. در نتیجه حرکت به چپ داشته‌ایم که این تناقض است.

از آنجا که در زنجیر محاسبه توسط δ از یک *Configuration* به *Configuration* بعدی می‌رویم. پس مسیری که روی گراف باید طی کنیم تا اطمینان داشته باشیم پس از آن دیگر حرکت به چپ نداریم، تعداد یالهایش $|Q|$ است، در نتیجه تعداد گره‌هایش، یعنی *Configuration* ها حداکثر برابر $|Q| + 1$ است. یعنی حداکثر *Configuration* مورد بررسی در شبیه‌سازی برابر $|Q| + 1$ است. پس برای اطمینان از این که این ماشین تورینگ به ازای ورودی ϵ هیچگاه به چپ حرکت نمی‌کند، بیشتر از $|Q|$، *Transition* اتفاق نخواهد افتاد. چرا که فرض کنیم که در $|Q|$، *Transition* به سمت راست حرکت کردیم ولی پس از آن به چپ رفتیم. در این صورت یعنی وجود دارد $\delta(q_i, \square) = (q_j, a, L)$ که از آن تا به حال استفاده نکردیم. در این صورت وجود دارد $q_i \in Q$ که تا $|Q|$، *Transition* اول به آن نرفته بودیم. در نتیجه $|Q| + k = |Q|$ که $k \neq 0$ و می‌دانیم که این تناقض است.

حال *Decider* زبان $L_{EpsilonLeft}$ به شکل فرمال تعریف می‌کنیم:

$$M_{EpsilonLeft}(\langle M\rangle) := \begin{cases} \text{Simmulate } M \text{ on } \epsilon \text{ for at most } |Q|+1 \text{ steps} \\ \\ \text{For each } Step_i = \begin{cases} \text{Update } CurrentStep \text{ with the } CurrentStep + 1 \\ \text{If } M \text{ ever moves its head left, reject} \\ \text{If } M \text{ accepts without moving its head left, accept} \\ \text{If } M \text{ rejects without moving its head left, accept} \\ \text{If } CurrentStep \geq |Q| + 1 \text{ accept} \end{cases} \end{cases}$$

نحوه‌ی کارکرد این *Decider* روشن است. ماشین M را روی ورودی ϵ تا $|Q| + 1$ امین *Configuration* از ماشین M شبیه‌سازی می‌کنیم. در این فاصله اگر ماشین هدش را به چپ حرکت داد Reject می‌کنیم. اگر بدون این که هدش را به چپ حرکت داده باشد، Accept یا Reject کرد، به راحتی Accept می‌کنیم. اگر تا $|Q| + 1$ امین *Configuration* هدش را به چپ حرکت نداد و هالت هم نکرد، با استدلال گفته شده، در *Configuration* تکراری می‌افتیم، در نتیجه در Loop خواهیم افتاد و در نتیجه دیگر هد این ماشین به چپ نخواهد رفت. پس Accept می‌کنیم.

پرسش ۵

آ) اگر L_2 را زبان زیر بگیریم:

$$L_2 = \{\langle M\rangle : |L(M)| \geq 2\}$$

آن گاه L_2 تشخیص پذیر (*recognizable*) است.

اثبات: برای آن یک *recognizer* می سازیم. برای ساخت *TM* آن از تکنیکی با نام *Dovetailing* استفاده می کنیم. این تکنیک معادل آنی است که در *Bijection* از $\mathbb{Q}$ به $\mathbb{N}$ داشتیم. به شکل تاریخی این *Bijection* مرتبط است با چیزی که در حال حاضر به آن *"Cantor pairing function"* گفته می شود. این تابع به شکل زیبایی هر دوتایی مثل (x, y) که $x, y \in \mathbb{N}$ را به یک $z \in \mathbb{N}$ می برد. به هر عضو از $\mathbb{Q}$ نیز می توان به شکل یک دوتایی نگاه کرد (که مولفه دوم هیچ گاه صفر نیست). برخی نیز به آن تناظر *"مارپیچی"* (*i.e. snaking*) می گویند، چراکه این تناظر در برخی جاها شکلی مارپیچی روی قطرهای ماتریس $\mathbb{Q}$ دارد. در اینجا ولی به جای شکل *"مارپیچ"* شکل قطرهای برهم موازی روی ماتریس دارد. اگر می خواستیم در شکلی که در ادامه می آید، حرکتمان، نمای مارپیچی به خود بگیرد، می بایست جهت فلشها را یکی در میان تغییر می دادیم. اگر بخواهیم در این جا بگوییم که این دوتایی چیست، به این فکر کنیم که دوتایی (x, k) یعنی ماشین M را k، *Step* روی ورودی $x \in \Sigma^*$ اجرا کنیم. از آن جا که Σ^* شماراست و اگر $x \in L$ پس در k، *Step* که $k \in \{0, 1, 2, \ldots\}$ پذیرفته خواهد شد، پس برای ورودی هایی که عضو زبانند در زمان محدود، با حرکت قطری قابل رسیدن خواهد بود. (در ادامه این ایده کمی بیشتر باز خواهد شد)

برای L_2 یک *Recognizer* طراحی می کنیم. برای آنکه نشان دهیم R یک *Recognizer* برای L_2 است، کافیست دو چیز را نشان دهیم:

1. $w \in L_2 \Rightarrow (R \text{ Halts and Accepts on input } w)$
2. $w \notin L_2 \Rightarrow (R \text{ Rejects or Loop on input } w)$

می دانیم که Σ^* شماراست و قابل ترتیب دهی است. اگر ترتیب آن را *Lexicographic* بگیریم که برچسب سطرهای ماتریس ماست. و همچنین برچسب ستون را $Step_k$ بگیریم. آن گاه درایه ی $[x, Step_k]$ یعنی اجرای k، Step از ماشین M روی ورودی x. برای مثال زمانی که به عضو $(\epsilon, 8)$ یعنی درایه ی $[\epsilon, Step_8]$ رسیدیم، ماشین M را 8، Step روی ورودی ϵ اجرا می کنیم. به این تکنیک *Dovetailing* می گویند. ماتریس زیر نمونه ای برای الفبای $\Sigma = \{0, 1\}$ است که به هر الفبای Σ نیز قابل تعمیم است:

	$Step_1$	$Step_2$	$Step_3$	$Step_4$	$\cdots$
ϵ	$(\epsilon, 1)$	$(\epsilon, 2)$	$(\epsilon, 3)$	$(\epsilon, 4)$	$\cdots$
0	$(0, 1)$	$(0, 2)$	$(0, 3)$	$(0, 4)$	$\cdots$
1	$(1, 1)$	$(1, 2)$	$(1, 3)$	$(1, 4)$	$\cdots$
00	$(00, 1)$	$(00, 2)$	$(00, 3)$	$(00, 4)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\pi(x, y)$ همان *Cantor pairing function* است. خروجی این تابع تعداد قدمهایی است که برای رسیدن به (x, y) لازم است. توجه شود که در اینجا منظور از قدم، حرکت روی ماتریس است و فعلا کاری با مفهوم Step در ماشین تورینگ نداریم. فرمول این تابع برابر:

$$\pi(k_1, k_2) = \tfrac{1}{2}(k_1 + k_2)(k_1 + k_2 + 1) + k_2$$

می باشد. برای مثال در اینجا برای رسیدن به عنصر $(1, 2)$ باید 8 قدم برداریم:

$$\pi(index(1), index(2)) = (\tfrac{3.4}{2}) + 2 = 8$$

توجه شود که $index(x)$ همان اندیس *lexicographic* برای x است، برای مثال $index(\epsilon) = 0$ و $index(00) = 3$ می باشد.

اگر وجود داشته باشد $w, x \in \Sigma^*$ که $w, x \in L_2$ آن گاه داریم:

$$\Big(\exists k\Big)\ \Big((w,k') = accept\Big)\ \&\Big((x,k'') = accept\Big)\ \&\ \Big(max(k',k'') = k\Big)$$

یعنی وجود دارد $k \in \mathbb{N}$ به نحوی که ماشین M رشته های w و x را در قدمهای به ترتیب k' و k'' می پذیرد، به شکلی که هر دویِ k' و k'' از k کوچکترمساوی اند. چون اگر رشته ای توسط ماشین تورینگی پذیرفته شود، حتما در متناهی Step پذیرفته شده است. همچنین هر دو درایه ی $[w, step_{k'}]$ و $[x, step_{k''}]$ پس از $max\left(\pi\Big(index(w),k'\Big), \pi\Big(index(x),k''\Big)\right)$ قدم با روش *Dovetailing* قابل رسیدن در ماتریس متناظرشان اند. توجه شود که اجرای M در هر درایه به اندازه ی متناهی Step است. اگر فرض کنیم که $\langle M\rangle \in L_2$ و اگر بدون از دست دادن عمومیت $(w.l.o.g)$ فرض کنیم که $max\left(\pi\Big(index(w),k'\Big), \pi\Big(index(x),k''\Big)\right) = k'$ آن گاه آخرین عنصر دیده شده در ماتریس برابر (w,k') است.
اگر همه ی عنصرهای ملاقات شده ی پیش از آن را در مجموعه S بریزیم آن گاه S برابر $S = \{(\epsilon,1), \ldots, (w,k')\}$ و همچنین به سادگی قابل فهم است که $|S| = \pi\Big(index(w),k'\Big)$. R برای همه ی عضوهای S محاسبه انجام می دهد. از آن جا که محاسبه برای هر عضو S متناهی زمان می برد و اندازه S نیز برابر $\pi\Big(index(w),k'\Big)$ و متناهی است. پس اگر $\langle M\rangle \in L_2$ آن گاه $\langle M\rangle$ پس از متناهی زمان توسط R پذیرفته خواهد شد. با جمله آخر نشان دادیم که 1 برقرار است.

اگر $\langle M\rangle \notin L_2$ آن گاه یا $|L| = 0$ یا $|L| = 1$ بدون از دست دادن عمومیت $(w.l.o.g)$ اگر فرض کنیم $|L| = 1$ آن گاه از آن جا که رشته دوم x به شکلی که $x \in L_2$ وجود ندارد، قدم زدن روی آرایه هیچ گاه تمام نمی شود و حتما *Loop* خواهیم داشت. به این ترتیب نشان دادیم 2 نیز برقرار است. در نتیجه R یک *Recognizer* برای L_2 است. پس L_2، *Recognizable* است.

ب)

اگر L_{FIN} را زبان زیر بگیریم:

$$L_{FIN} = \{\langle M\rangle : L(M) \text{ is finite}\}$$

آن گاه L_{FIN} تشخیص پذیر نیست ($i.e.$, $unrecognizable$).

نشان می دهیم که اگر برای L_{FIN} یک $Recognizer$ داشته باشیم، به کمک آن می توانیم $\overline{HP}$ را تشخیص دهیم. $\overline{HP}$ نقیض مساله $Halting\ Problem$ است. اگر که $\langle M, x\rangle \in \overline{HP}$ یعنی ماشین M روی ورودی x هالت نمی کند:

$$\overline{HP} = \{\langle M, x\rangle : M \text{ does not halt on } x)\}$$

می دانیم که $\overline{HP} \notin RE$ چراکه اگر این طور بود آن گاه HP که همان زبان $Halting\ Problem$ است، تصمیم پذیر بود. کافی بود ورودی را به هر دو $Recognizer$، HP_{TM} و $\overline{HP}_{TM}$ به طور همزمان بدهیم. برای هر ورودی دو حالت مجزا وجود دارد. یا ماشین M روی x هالت می کند، یا هالت نمی کند. به هر حال از آن جا که برای هر دو $Recognizer$ داریم، اگر هالت کند HP_{TM} روی $Accept$ هالت می کند، در غیر این صورت هم $\overline{HP}_{TM}$ روی $Accept$ هالت می کند، که یعنی برای HP یک $Decider$ داریم که بر اساس [۱، Thm. 5.1] تناقض است. پس می دانیم که $\overline{HP} \notin RE$.

فرض کنم برای L_{FIN} یک $Recognizer$ با نام F داریم. حال به کمک F برای $\overline{HP}$ یک $Recognizer$ با نام $\overline{HP}_{TM}$ می سازیم. ابتدا ماشین N را این گونه می سازیم که برای هر ورودی y ورودی y را از روی $Tape$ پاک کند. سپس M را روی x اجرا کند. اگر M هالت کرد، آن گاه $Accept$ کند. در این صورت اگر $\langle M, x\rangle \in \overline{HP}$ آن گاه ماشین N برای هیچ ورودی $Accept$ نمی کند. در نتیجه $L(N) = \emptyset$ پس داریم:

$$\langle M, x\rangle \in \overline{HP} \Leftrightarrow |L(N)| = 0 \Leftrightarrow L(N) \text{ is finite}$$

در غیر این صورت. یعنی زمانی که $\langle M, x\rangle \notin \overline{HP}$ آن گاه به ازای هر ورودی y ماشین N ورودی را پاک کرده و M را روی x اجرا می کند و M بر اساس فرض حتما $Halt$ می کند، در نتیجه همه ی ورودیها پذیرفته خواهند شد، در نتیجه $L(M) = \Sigma^*$ پس داریم:

$$\langle M, x\rangle \notin \overline{HP} \Leftrightarrow L(N) = \Sigma^* \Leftrightarrow L(N) \text{ is } NOT \text{ finite}$$

ماشین N را به شکل فرمال تعریف می کنیم:

$$N(y) := \begin{cases} \text{Erase y from the tape} \\ \text{Run TM } M \text{ on input } x \\ \text{Accept if } M \text{ halts} \end{cases}$$

حال $\overline{HP}_{TM}$ را این گونه می سازیم:

$$\overline{HP}_{TM}(\langle M, x\rangle) := \begin{cases} \text{If } \langle M, x\rangle \text{ is not a pair s.t. } M \text{ is a TM and } x \text{ is a String then reject} \\ \\ \text{Build a new machine } N(y) = \begin{cases} \text{Erase y from the tape} \\ \text{Run TM } M \text{ on input } x \\ \text{Accept if } M \text{ halts} \end{cases} \\ \\ \text{Run } F \text{ on input } \langle N\rangle \text{ and do the same} \end{cases}$$

توجه شود که ساخت ماشین N از روی ورودی $\langle M, x\rangle$ به صورت الگوریتمیک قابل انجام است. به عبارتی یک توصیف از ماشین N می سازیم و هرگز آن را اجرا نمی کنیم. این کار الگوریتمیک و در زمان متناهی قابل انجام است. از آن جا که فرض کردیم F یک $Recognizer$ برای L_{FIN} است، پس در صورتی که $L(N)$ متناهی باشد، حتما $Accept$ می کند. در نتیجه ماشین $\overline{HP}_{TM}$ ورودی را به درستی $Accept$ می کند. در صورتی که $L(N)$ متناهی نباشد هم می دانیم که F یا در $Loop$ افتاده یا در $Reject$، به هر حال هرگز $Accept$ نخواهد کرد و در نتیجه، اگر هر کاری را که F کرد به عنوان خروجی برگردانیم، یعنی برای $\overline{HP}_{TM}$ یک $Recognizer$ ساختیم. پس $\overline{HP}_{TM} \in RE$ که این تناقض است $\perp$.

در نتیجه $L_{FIN} \notin RE$.

پ)

اگر L_{CFL} را زبان زیر بگیریم:

$$L_{CFL} = \{\langle M\rangle : L(M) \text{ is context-free}\}$$

آن گاه L_{CFL} تشخیص پذیر نیست ($i.e.\ ,\ unrecognizable$).

مانند "ب" نشان می دهیم که اگر برای L_{CFL} یک *Recognizer* داشته باشیم، به کمک آن می توانیم $\overline{HP}$ را تشخیص دهیم. در صورتی که با توجه آنچه که در "ب" داشتیم، می دانیم $\overline{HP} \notin RE$.

تکنیک تا اندازه ی زیادی با روشی که در "ب" داشتیم نزدیکی دارد. فرض می کنیم L_{CFL}، *Recognizable* است، در نتیجه یک *Recognizer* با نام F دارد. حال به کمک F برای $\overline{HP}$ یک *Recognizer* با نام $\overline{HP}_{TM}$ می سازیم. ابتدا ماشین N را این گونه طراحی می کنیم که برای هر ورودی y ابتدا M را روی x اجرا کند، سپس اگر y فرم $a^nb^nc^n$ داشت، آن را *Accept* کند. به این ترتیب اگر M روی x هالت نکند، این ماشین هیچ ورودی را نمی پذیرد و در نتیجه $L(N) = \emptyset$ خواهد بود. پس داریم:

$$\langle M,x\rangle \in \overline{HP} \Leftrightarrow L(N) = \emptyset \Leftrightarrow L(N) \text{ is context-free}$$

در غیر این صورت، یعنی M روی x هالت کرده است. آن گاه پس از اجرای M روی x ابتدا چک می شود که ورودی فرم $a^nb^nc^n$ داشته باشد، سپس *Accept* می شود. پس در این حالت زبان N برابر $a^nb^nc^n$ خواهد بود. حال داریم:

$$\langle M,x\rangle \notin \overline{HP} \Leftrightarrow L(N) = a^nb^nc^n \Leftrightarrow L(N) \text{ is } NOT \text{ context-free}$$

ماشین N را به شکل فرمال تعریف می کنیم:

$$N(y) := \begin{cases} \text{Run TM } M \text{ on input } x \\ \text{Accept if } y \text{ has the form } a^nb^nc^n \end{cases}$$

حال $\overline{HP}_{TM}$ را این گونه می سازیم:

$$\overline{HP}_{TM}(\langle M,x\rangle) := \begin{cases} \text{If } \langle M,x\rangle \text{ is not a pair s.t. } M \text{ is a TM and } x \text{ is a String then reject} \\ \\ \text{Build a new machine } N(y) = \begin{cases} \text{Run TM } M \text{ on input } x \\ \text{Accept if } y \text{ has the form } a^nb^nc^n \end{cases} \\ \\ \text{Run } F \text{ on input } \langle N\rangle \text{ and do the same} \end{cases}$$

می دانیم که ساخت توصیف N به صورت الگوریتمیک از روی M و x قابل انجام است و دوباره تاکید می شود که تولید الگوریتمیک توصیفی از ماشین تورینگی که M را روی x اجرا کند سپس چک کند که y فرم $a^nb^nc^n$ داشته باشد، و اگر داشت آن را بپذیرد، راحت است. حال از آن جا که برای L_{CFL} یک *Recognizer* با نام F داریم، می توانیم CFL بودن آن را بررسی کنیم و اگر ورودی آن *context-free* نباشد، دو حالت $\{Loop, Reject\}$ قابل تصور است. اگر $\langle M,x\rangle \in \overline{HP}$ آن گاه $L(N) = \emptyset$ و در نتیجه F حتما روی حالت *Accept* هالت خواهد کرد. در غیر این صورت با توجه به آن چه در بالا گفتیم، اگر $\langle M,x\rangle \notin \overline{HP}$ پس $L(N) = a^nb^nc^n$ که *context-free* نیست. در نتیجه هیچ گاه F روی حالت *Accept* هالت نخواهد کرد و در یکی از دو حالت $\{Loop, Reject\}$ قرار خواهد گرفت. در این صورت $\overline{HP}_{TM}$ یک *Recognizer* برای زبان $\overline{HP}$ است، که این تناقض است $\perp$. پس $L_{CFL} \notin RE$.

مراجع

[۱] Sipser, M. (2012), *Introduction to the Theory of Computation* ,Cengage learning

[۲] Sriram Pemmaraju (2012), *Limits of Computation*, University of Iowa 22C:131 "http://homepage.divms.uiowa.edu/~sriram/131/spring07/homework2Solution.pdf"