نظریه علوم کامپیوتر

نیم‌سال دوم ۱۳۹۹-۱۴۰۰

مدرس: دکتر جواد ابراهیمی، دستیار آموزشی: ساجد کریمی

دانشکده علوم ریاضی

تمرین سری سوم

نام و نام‌خانوادگی: امید یعقوبی — شماره دانشجویی: -

پرسش ۱

آ) می‌خواهیم نشان دهیم اگر ماشین تورینگی نتواند روی بخشی از نوار خود که محتوای ورودی دارد چیزی بنویسد، قدرت معنایی زبان آن با قدرت معنایی زبان‌های منظم برابر است.

در ابتدا با کمی تغییر در نام انتخابی برای این ماشین، ما نام آن را RO_n می‌گذاریم، چراکه تنها در n خانه شامل ورودی نمی‌تواند چیزی بنویسد و باقی پوزیشنهای نوار برای آن قابلیت خواندن و نوشتن دارد.

برای تاکید اضافه می‌شود که اثبات این قضیه با اثبات برابری قدرتی زبان *2FSA* که شباهت کوچکی با این مساله دارد متفاوت است. تفاوت این دو مساله اینجاست که در *2FSA* نوار به طور کلی فقط خواندنی و از دو سمت ورودی محدود است. از این رو در آنجا تابع انتقال δ از یک پنج‌تایی به یک چهارتایی کاهش پیدا می‌کند، چراکه با خواندن $a \in \Sigma$ نمی‌تواند چیزی را روی آن بنویسد. در نتیجه برابری قدرت زبانهای *2FSA* با زبانهای *FSA* در اینجا فرض گرفته می‌شود. برای خواننده‌ی علاقه‌مند به دانستن آن خواندن این اثبات از روی [2][pp. 124, 125, 126] توصیه می‌شود. اثبات به وسیله قضیه‌ی *Myhill-Nerode* انجام شده است.

در اینجا ولی تمرکز روی سوالی عمومی‌تر است. اینکه اگر ماشین ما نتواند در هنگام خواندن ورودی‌اش چیزی را به خاطر بسپارد، قدرت تشخیصیِ آن نسبت به رشته‌ها به اندازه‌ی قدرت تشخیصیِ یک ماشین حالت‌ـ متناهی *(FSA)* است.

پیش از نوشتن اثبات به شکل فرمال، بهتر است شهودی به چراییِ این اتفاق داشته باشیم. تفاوت ماشین تورینگ و ماشین متناهی‌ـ حالت در قدرت به یاد آوردن بخشی از ورودی است که تا به حال هد از روی آن‌ها رد شده است. این به‌یادآوری می‌تواند حتا با یک یا دو شمارنده انجام پذیرد. اولین چیزی که پس از محدود شدن به خواندن در بخش ورودی ممکن است به ذهن برسد، کپی کردن ورودی روی بخش آزاد نوار است. اما این کپی کردن با چه مکانیزمی انجام می‌پذیرد؟ معمولا این کار را با علامت زدن بخش ورودی نوار انجام می‌دهیم. حتا اگر یک کانتر در اختیار ما قرار می‌دادند که بتوانیم در بخش ورودی آن را تنها $+1$ کنیم و همچنین بتوانیم در بخش ورودی آن را بخوانیم، آن‌گاه می‌شد به ابتدای ورودی رفت. اشاره‌گر به حالت فعلی را روی حالتی برد که مرتبط با حرف خوانده شده است. کانتر را $+1$ کرد. به بخش آزاد نوار رفت و حرف را نوشت. به ابتدای نوار بازگشت. به اندازه‌ی کانتر جلو رفت و این پروسه را تکرار کرد. که تمام این‌ها فرقی با آن ندارد که حرف را پس از کپی علامت زده باشیم. حال اگر بخواهیم تصویر بزرگی از چرایی ناتوانیِ تورینگ RO_n داشته باشیم. این ماشین زمانی که روی ورودی با طول متغیر خود قرار می‌گیرد، تنها می‌تواند بخش محدودی از آن چه را پیشتر روی ورودی دیده بود به یاد آورد. چراکه حافظه‌ی تورینگ RO_n در بخش ورودی تنها حافظه‌ی حالتهای آن است. یعنی زمانی که هد به بخش ورودی وارد می‌شود، اینکه چه چیزهایی پیش از این از ورودی داشته است تنها تابعی است از تعداد بخش حالت‌ـ متناهی آن، از این رو تنها می تواند روی بخش متناهی از ورودی قدرت به یاد آوری داشته باشد، و در نتیجه به صورت غیرفرمال می‌توان حس کرد که چرا این ماشین قدرت تشخیصی در حد ماشین متناهی‌ـ حالت دارد. حال در ادامه اثباتی فرمال و دقیق خواهیم آورد.

ماشین تورینگ RO_n در طول اجرایش، زمان‌هایی درون بخش ورودی و زمان‌هایی خارج از آن است. حال دو رویداد *in-event* و *out-event* را تعریف می‌کنیم. *in-event* رویداد ورود از بخش غیرورودی (*بخش خواندنی و نوشتنی*) به بخش ورودی (*بخش فقط‌ـ خواندنی*) می‌باشد و *out-event* برعکس آن یعنی خروج از بخش ورودی (*بخش فقط‌ـ*

خواندنی) به بخش غیرورودی *(بخش خواندنی و نوشتنی)* است.

در ابتدا می‌خواهیم نشان دهیم نتیجه‌ی پردازش بخش ورودی همواره در مجموعه‌ای متناهی قابل تعریف است. به بیان دیگر در هر $out\text{-}event$ آنچه از ورودی خوانده شده که یادمان مانده است، در قالب آخرین $state$ دیده شده، یا همان $state$ فعلی در هنگام وقوع رویداد $out\text{-}event$ می‌باشد. از این رو هر تابعی از هر ورودی w به بیرون بخش ورودی، $co\text{-}domain$ محدود دارد. توجه شود که در اینجا تابع ما رفتار تورینگ ماشین پیش از $out\text{-}event$ است. به این ترتیب، ماشین تورینگ ما از آنجا که تنها روی ورودی توانایی خواندن دارد، هنگام خروج از آن روی یک $state$ قرار می‌گیرد. این $state$ خروجی تابع است. دقیق‌تر این‌که، تورینگ ما پس از محاسبه‌ی این تابع، ممکن است چیزهایی را روی بخش آزاد نوار هم بنویسد و دوباره به بخش ورودی‌اش بازگردد، هنگام ورود به بخش ورودی یا هنگام رویداد $in\text{-}event$ تورینگ آن‌چه همراه خود به داخل ورودی می‌برد چیزی جز یک $state$ نیست. پس رفتار تابع نه فقط بر اساس $input$ بلکه براساس حالت فعلی در رویداد $in\text{-}event$ و همچنین $input$ خواهد بود. اگر بخواهیم این توضیحات را در یک جمله خلاصه بگوییم، با وجود این که تورینگ ما برای بخش غیر از ورودی‌اش قدرت کامل دارد، اطلاعاتی که از ورودیِ اندازه‌ـ متغیرش دارد، محدود *(finite)* خواهد بود. اگر این محدودیت وجود داشته باشد، با استفاده از قضیه‌ی *Myhill-Nerode* ، تعداد کلاس‌های هم ارزی‌اش محدود و در نتیجه منظم است. این‌ها همه با دقت بیشتر در ادامه بررسی خواهد شد تا نکته‌ای در ابهام باقی نماند.

برای یک نمونه ماشین RO_n با نام M و $w \in \Sigma^*$ دو حالت وجود دارد:

$$|out\text{-}event| \geq 1$$

$$|out\text{-}event| = 0$$

به این توجه شود که این دو حالت دوبه‌دو مجزا *(Mutually exclusive)* و روی هم کامل *(collectively exhaustive)* هستند. یعنی امکان ندارد حالتی جز این دو متصور بود و امکان ندارد هر دوی این حالات باهم به وجود بیایند. می‌دانیم، که اگر اثبات کنیم که در هر دوی این حالات زبان توصیفی منظم *(regular)* است. ثابت کرده‌ایم که در تمامی حالات زبان توصیفی *(regular)* خواهد بود. اگر بخواهیم خیلی ریزبینانه نگاه کنیم، اول اینکه تشخیص این شرط توسط یک *2FSA* با دو حالت امکان پذیر است، حالت q_0 حالت آغازین و غیر پایانی و q_1 را پایانی می‌گیریم. می‌دانیم که الحاق زبان منظم با یک کاراکتر منظم است. الفبای زبان را به $\Sigma' = \Sigma \cup \{\$\}$ به نحوی که $\$ \notin \Sigma$ گسترش می‌دهیم. حال به ازای هر $\delta_M(q_i, a) = (q_j, b, D)$ در ماشین اصلی در *2FSA* که شرط $|out\text{-}event| = 0$ را تشخیص می‌دهد داریم $\delta_{M_{out}}(q_0, a) = (q_0, D)$ و به ازای $\$$ داریم $\delta(q_0, \$) = (q_1, R)$ به این ترتیب ماشین تنها در صورتی پس از انتهای خواندن روی q_1 خواهد بود که یک بار سمت راست‌ترین کارکتر خود را رد کند و از بخش ورودی‌اش خارج شود. چراکه $\$$ به انتهای w الحاق شده بود. حال اگر نام این زبان منظم را L_{out} بگیریم. دو زبان L_1 و L_2 را این‌گونه تعریف می‌کنیم:

$$L_1 := \{w | w \in L_{out} \land w \in L(M)\}$$
$$L_2 := \{w | w \notin L_{out} \land w \in L(M)\}$$

توجه شود که M یک نمونه دلخواه از ماشین RO_n است و L_{out} وابسته به M است. به این نحو که اگر اجرای M روی ورودیِ w از بخش ورودی بیرون رفت آن‌گاه $w \in L_{out}$ ،پس اگر $w \in L(M)$ باشد، یا $w \in L_1$ است یا $w \in L_2$ در نتیجه زبان اصلی برابر است با $L_1 \cup L_2$. حال اگر نشان دهیم که هم L_2 و هم L_1 منظم هستند، با توجه خاصیت‌های بستاری زبان‌های منظم این زبان نیز منظم است.

در حالت اول دست کم یک بار از بخش ورودی خارج شده‌ایم. حالت دوم یعنی از بخش ورودی یکبار هم خارج نشده‌ایم. ابتدا ادعا می‌کنیم به ازای حالت دوم یعنی زمانی که $w \notin L_{out}$ حتما یک *2FSA* وجود دارد. کافیست ورودی آن را درون نوار فقط خواندنی محصور در دو علامت ابتدایی و انتهایی قرار دهیم. همچنین برای هر

$$\delta(q_i, a) = (q_j, b, D)$$

که

$$q_i, q_j \in Q \ , \ a, b \in \Sigma \ , \ D \in \{L, R\}$$

در $2FSA$ متناظر که نام آن را M_2 می‌گذاریم تابع انتقال

$$\delta(q_i, a) = (q_j, D)$$

را تعریف می‌کنیم. حال ادعا می‌کنیم:

$$L_2 = L(M_2) \cap \overline{L_{out}}$$

فرض کنیم که این طور نباشد. یعنی وجود داشته است $w \in \overline{L_{out}}$ به نحوی که $\delta^*_M(w) = \text{acc}$ ولی $\delta^*_{M_2}(w) \neq \text{acc}$ این بدان معناست که در یکی از مراحل بازگشتی محاسبه‌ی δ^* یک تابع انتقال ساخته شده از روی M وجود داشته به نحوی که $\delta_M(q_i, a) \neq \delta_{M_2}(q_i, a)$ ولی تنها تفاوت این دو تابع در نوشتن روی نوار پس از خواندن $a \in \Sigma$ است. این بدان معناست که ماشین M تابع انتقالی دارد که به هنگام خواندن $a \in \Sigma$ جای آن $b \in \Sigma$ را می‌نویسد. طبق فرض $|out\text{-}event| = 0$ نتیجه می‌گیریم که این تابع انتقال لزوما روی بخش ورودی‌اش اقدام به نوشتن می‌کرده است. چراکه در غیر این صورت نباید در δ^* وجود می‌داشت. توجه کنید که $w \in \overline{L_{out}}$ پس فرض $|out\text{-}event| = 0$ آن را محدود به w هایی می‌کند که ماشین ما به ازای آن هیچ‌گاه از بخش ورودی‌اش خارج نمی‌شود. در این صورت اگر دلتا چیزی را بنویسد با استدلال آورده شده لزوما در بخش ورودی نوشته است که این تناقض است. پس به ازای حالت دوم یک $2FSA$ داریم. می‌دانیم که زبان تشخیصی هر $2FSA$ منظم است. توجه شود که برای $w \notin \overline{L_{out}}$ ممکن است رفتار ماشین M و M_2 متفاوت باشد. یعنی برای w که ماشین M به ازای آن از بخش ورودی اش خارج می‌شود، ممکن است روی بخشهای آزاد نوار چیزی بنویسد و ماشین M_2 از آن رو که نوشتن ندارد این کار را نکند. ولی برای ما مهم نیست که به ازای رشته‌های خارج از $\overline{L_{out}}$ این رفتار متفاوت است. چراکه ما تا به حال تنها گزاره‌ی زیر را اثبات کردیم:

$$L_2 = L(M_2) \cap \overline{L_{out}} = L(M) \cap \overline{L_{out}}$$

یعنی می‌دانیم که اگر شرط $|out\text{-}event| = 0$ برقرار باشد، آن‌گاه رفتار M_2 با M یکسان است. حال چون $L(M_2)$ منظم بوده و $\overline{L_{out}}$ هم منظم است، پس L_2 منظم است. می‌دانیم که زبان اصلی $L = L(M)$ برابر است با $L = L_1 \cup L_2$ پس اگر اثبات کنیم L_1 هم منظم است. از آن جا که منظم نسبت به اجتماع بسته است، ثابت کردیم که L منظم است و چون $L = (M)$ که M یک ماشین دلخواه RO_n است، پس ثابت کردیم که هر نمونه از RO_n منظم است.

ابتدا حالت $first_w$ را تعریف می‌کنیم. اگر $|out\text{-}event| \geq 1$ آن‌گاه حالتی را که هنگام اولین رویداد $out\text{-}event$ روی آن خواهیم بود را $first_w$ می‌نامیم. به طور استثنا در حالتی که دیگر به بخش ورودی باز نگردیم و ماشین M روی acc هالت کند، $first_w = q_{acc}$ قرار می‌دهیم و همچنین اگر که دیگر به بخش ورودی باز نگردیم و ماشین M روی rej هالت کند، $first_w = q_{rej}$ قرار می‌دهیم. حال $characteristic\ function$ به شکل $f_w(q_i) = q_j$ را برای هر $q_i \in Q$ و $q_j \in Q \cup \{acc, rej\}$ به این شکل تعریف می‌کنیم: آخرین حالت پیش از ورود به بخش ورودی را q_i می‌گیریم، پس از رویداد $in\text{-}event$ یا در همان بخش ورودی هالت می‌کنیم یا در رویداد $out\text{-}event$ به بخش خواندنی و نوشتنی غیر ورودی بازمی‌گردیم. در حالت اول، یعنی زمانی که در بخش ورودی هالت اتفاق افتاده است. اگر روی acc هالت اتفاق افتاد، آن‌گاه $q_j = q_{acc}$ و اگر روی rej هالت شده بود، آن‌گاه $q_j = q_{rej}$ در غیر این صورت q_j حالتی است که در رویداد بعدی $out\text{-}event$ روی آن خواهیم بود. به عبارتی اگر آخرین حالتی که پیش از ورود به بخش ورودی w روی آن بودیم را q_1 بگیریم و اولین حالتی را که پس از خروج از بخش ورودی w روی آن هستیم را q_2 بگیریم. آن‌گاه $f_w(q_1) = q_2$ اگر هم که بازنگردیم و روی acc هالت کنیم آن‌گاه $f_w(q_1) = q_{acc}$ و به همین ترتیب برای rej هم $f_w(q_1) = q_{rej}$ می‌باشد.

ادعا می‌کنیم که برای هر دو رشته‌ی متفاوت $w_1, w_2 \in \Sigma^*$ اگر $first_{w_1} = first_{w_2}$ و برای هر $q_i \in Q$ نیز $f_{w_1}(q_i) = f_{w_2}(q_i)$ آن‌گاه رشته‌های w_1 و w_2 جدا ناشدنی از M هستند. یعنی برای هر $z \in \Sigma^*$، M رشته‌ی w_1z را می‌پذیرد اگر و تنها اگر که w_2z را بپذیرد.

برای هر $Configuration\text{-}History$ به شکل $C_1, C_2, C_3..., C_m$ رویدادهای $in\text{-}event$ و $out\text{-}event$ یک پارتیشن به شکل $B_1, B_2, B_3..., B_{m'}$ وجود دارد به شکلی که بین هر B_{2i-1}, B_{2i} یک $out\text{-}event$ و بین هر B_{2i}, B_{2i+1} یک $in\text{-}event$ وجود دارد. به طور دقیق‌تر بخش‌هایی از پردازش که ما درون بخش ورودی هستیم در B_{2i-1} ها یا بلاک‌های

فرد و بخش‌هایی از پردازش که درون بخش غیر ورودی هستیم در B_{2i} ها یا بلاک‌های زوج قرار می‌گیرد.

شاید این سوال پیش بیاید که چرا f_w خوش تعریف است. در حقیقت f_w دارای *domain* و *co-domain* محدود بوده و خوش‌تعریف است. چراکه هر $f_w(q_i)$ معادل یک *2FSA* است. f_w خوش تعریف است چراکه δ_M روی بخش ورودی‌اش نمی‌نویسد و چراکه δ_M خوش‌تعریف است. در ادامه نشان می‌دهیم که برای هر $f_w(q_i)$ یک *2FSA* وجود دارد.

به ازای هر $f_w(q_i) = q_j$ که $q_i \in Q$ و $q_j \in Q \cup \{acc, rej\}$ و $w = a_1, a_2, \ldots, a_n$ یک *Configuration-History* معادل به شکل زیر وجود دارد:

$$\underbrace{(a_1, a_2, \ldots, a_n, q_i, \ldots)}_{C_1} \underbrace{\rightarrow}_{\textit{in-event}} \underbrace{(a_1, a_2, \ldots, q_{x_1}, a_n, \ldots)}_{C_2} \rightarrow \cdots \rightarrow$$
$$\underbrace{(a_1, a_2, \ldots, q_{x_2}, a_n, \ldots)}_{C_{m-1}} \underbrace{\rightarrow}_{\textit{out-event}} \underbrace{(a_1, a_2, \ldots, a_n, q_j, \ldots)}_{C_m}$$

ما *IRC* یا *Input Restricted Computations* را این‌گونه تعریف می‌کنیم: رشته محاسبه‌ای که در آن هد ماشین تورینگ در ابتدا روی راست‌ترین حرف ورودی است و در بخش ورودیِ نوار می‌ماند غیر از آخرین حرکت هد که به یک خانه سمت راست ورودی حرکت می‌کند. همچنین روی ورودی‌اش هیچ نمی‌نویسد *(read-only)* [3].

ادعا می‌کنیم برای هر *Input Restricted Computations* یک *2FSA* معادل وجود دارد. شهود وجود یک *2FSA* برای هر *IRC* به سادگی از این مشاهده به دست می‌آید که در *IRC* ما روی ورودی هیچ نمی‌نویسیم و همچنین محدوده‌یِ قابل دسترسِ ما محدود به طول ورودی است غیر از آخرین حرکت که در آن هنوز هیچ روی ورودی نمی‌نویسیم و همچنین تنها یک خانَه از ورودی به بیرون حرکت می‌کنیم.

برای ساخت *2FSA* متناظر ورودی را با یک $\$ \notin \Sigma$ الحاق می‌کنیم و آن را روی نوار فقط خواندی محصور در دو *End-Marker* چپ و راست با نامهای *LeftEnd* و *RightEnd* قرار می‌دهیم. یک *dummy-state* با نام q_{dummy} به Q اضافه می‌کنیم و آن را حالت آغازین قرار می‌دهیم. برای هر $a \in \Sigma$ تابع انتقال $\delta(q_{dummy}, a) = (q_{dummy}, R)$ را تعریف می‌کنیم. همچنین تابع انتقال $\delta(q_{dummy}, \$) = (q_x, L)$ را نیز تعریف می‌کنیم. تمامی تابع‌های انتقال M را با حذف پارمتر سوم (*آن‌چه جای حرف خوانده شده می‌نویسد*) به این ماشین منتقل می‌کنیم. همچنین همه‌ی حالتهای ماشین را نیز به این ماشین منتقل می‌کنیم. مشاهده شود که در ابتدای اجرای ماشین ساخته شده، به راست ترین بخش ورودی می‌رویم. در بقیه موارد دقیقا همانطور رفتار می‌کنیم که *IRC* رفتار می‌کند. در اینجا نیز هر سه حالت را داریم. حالت اول *accept* شدن درون *2FSA* ساخته شده که برابر $f_w(q_i) = q_{acc}$ است، حالت دوم *reject* شدن درون *2FSA* ساخته شده که برابر $f_w(q_i) = q_{rej}$ است، حالت سوم روی راست ترین کارکتر ورودی‌اش که $\$$ است قرار گیرد، در این صورت روی یکی از حالت Q است که بر اساس تعریف f_w آن را q_j نامیدیم، این حالت نیز برابر $f_w(q_i) = q_j$ می‌باشد. مشاهده شود که *2FSA* متناظر ورودی w دارد و وابسته به q_i است. چراکه q_i است که مشخص می‌کند q_x چه باشد. پس به ازای هر q_i در ماشین M یک *2FSA* متناظر با نام M_{q_i} داریم و نتیجه‌ی $f_w(q_i)$ در حقیقت اجرای M_{q_i} روی ورودیِ w است. از آنجا که M_{q_i} یک *2FSA* است، پس معادل یک زبان منظم با نام $L(M_{q_i})$ است. بر اساس قضیه‌ی *Myhill-Nerode* هر زبان منظم دارای متناهی کلاس هم‌ارزی است. درنتیجه اگر برای دو رشته‌ی w_1 و w_2 و هر حالت $q_i \in Q$ داشته باشیم $f_{w_1}(q_i) = f_{w_2}(q_i)$ از آن جا که برای $f_{w_1}(q_i)$ و $f_{w_2}(q_i)$ یک زبان منظم $L(M_{q_i})$ داریم، برابری مقادیر این تابع‌ها به این معناست که به تعداد $|Q|$ زبان منظم داریم و در هر کدام کلاس هم ارزی w_1 برابر w_2 است. تاکید می‌شود که به ازای هر q_i کلاس هم ارزی w_1 برابر q_j است. چراکه گویی M_{q_i} روی ورودیِ w_1 اجرا شود و به حالت q_j رود و همین اتفاق نیز برای w_2 می‌افتد. در نتیجه $w_1 \in [q_j]$ و $w_2 \in [q_j]$ (از *[4][p. 29]* *می‌دانیم که اگر دو رشته متفاوت به یک حالت ختم شوند در کلاس هم ارزی یکسان و معادل آن State قرار می‌گیرند*). پس به ازای هر $z \in \Sigma^*$ داریم $f_{w_1z}(q_i) = f_{w_2z}(q_i)$ (به سادگی از خاصیت *right invariant* در پارگراف اول [4][p. 29] آن را می‌دانیم).

ابتدا نشان می‌دهیم که $first_w$ مشاهده شود که $first_{w_1z} = first_{w_2z}$ نتیجه‌ی یک *IRC* است با این تفاوت که محاسبه‌اش از سمت چپ ورودی آغاز می‌شود. برای $first_w$ نیز یک *2FSA* داریم که ورودی‌اش w است. نحوه‌ی ساخت آن با ساخت برای *IRC* تعریف شده‌ی استاندارد یکی است به جز اینکه دیگر q_{dummy} و *transition* های آن را نداریم و

حالت آغازین نیز همان حالت آغازین ماشین اصلی یعنی q_0 است. نام $2FSA$ معادل آن را M'_{q_0} می‌گذاریم. بر اساس فرض $first_{w_1} = first_{w_2}$ یعنی اجرای M'_{q_0} روی w_1 و روی w_2 به یک حالت مثلا q_x رفته است. براساس *right-invariant* از این پس نیز به یک جا خواهند رفت. در نتیجه $first_{w_1z} = first_{w_2z}$ اثبات می‌شود.

حال که نشان دادیم، اگر $first_{w_1} = first_{w_2}$ آن‌گاه برای هر z گزاره $first_{w_1z} = first_{w_2z}$ بر قرار است. می خواهیم نشان دهیم اگر علاوه بر یکی بودن $first_{w_1}$ و $first_{w_2}$ برای هر $q_i \in Q$ داشته باشیم $f_{w_1}(q_i) = f_{w_2}(q_i)$ آن‌گاه این دو رشته از هم جدایی ناپذیرند. برای اثبات روی تعداد event ها استقرا می‌زنیم. مشاهده می‌شود که در بلاک‌های زوج در *non-input-portion* هستیم و تا آخر بلاک روی آن خواهیم بود. بخش *input-portion* یا بلاک بعدی توسط f_w شبیه‌سازی می‌شود و تنها تفاوتی که در بخش *non-input-portion* به وجود می‌آورد تغییر q_i به q_j توسط تابع f_w است. همچنین مشاهده شود که در یک بلاک از کانفیگوریشن‌ها اگر اطمینان داریم از نقطه‌ای سمت چپ‌تر نخواهیم رفت، پس از آن نقطه به چپ روی محاسبات ما تاثیر نخواهد گذاشت. چراکه هد روی آن قرار نخواهد گرفت و در نتیجه هیچ یک از بخش‌های تشکیل دهنده‌ی δ^* یعنی δ ها روی آن بخش از نوار قرار نخواهد گرفت. در نتیجه اگر یک بلاک از *non-input-portion* به شکل زیر در نظر بگیریم:

$$\underbrace{wq_x\alpha\overset{*}{\vdash}wq_i\beta}_{B_{2i}}$$

از آن‌جا که w در این رشته محاسبه دخالتی ندارد (چون در بلاک *non-input-portion* هستیم) پس می‌توانیم w را از آن حذف کنیم:

$$\underbrace{q_x\alpha\overset{*}{\vdash}q_i\beta}_{B_{2k}}$$

از این پس به بلاکهایی که در آن در بخش *non-input-portion* هستیم، *non-input-block* می‌گوییم. حال اگر انتقال از یک *non-input-block* به *non-input-block* بعدی را در نظر بگیریم از آن‌جا که می‌دانیم پس از هر *in-event* مقدار q_i به $f_w(q_i)$ تغییر پیدا می‌کند، می‌توانیم یک انتقال از یک *non-input-block* به *non-input-block* بعدی را این‌گونه بنویسیم:

$$\underbrace{q_x\alpha\overset{*}{\vdash}q_i\beta}_{B_{2k}}\overset{1}{\vdash}\underbrace{f_w(q_i)\beta\overset{*}{\vdash}q_l\beta'}_{B_{2(k+1)}}$$

فرض داریم که $f_{w_1}(q_i) = f_{w_2}(q_i)$ برای هر q_i و اثبات کردیم که با این فرض $f_{w_1z}(q_i) = f_{w_2z}(q_i)$ برای هر z و q_i برقرار است. حال نشان می‌دهیم هر انتقال از *non-input-block* به *non-input-block* بعدی برای هر دو رشته‌ی w_1z و w_2z یکی است. اگر به صورت استقرایی برای اولین انتقال آن را اثبات کنیم و اثبات کنیم اگر برای هر دو رشته w_1z و w_2z بلاک متناظر B_{2k} آن‌ها یکی باشد آن‌گاه حتما بلاک متناظر $B_{2(k+1)}$ آن‌ها نیز یکی است، آن گاه ثابت کردیم که برای این دو رشته همواره *non-input-block* ها یکسان هستند. حال دو حالت وجود دارد. اول این که محاسبه در *non-input-block* تمام شود که با این اثبات هر دو رشته عضو یک کلاس هم‌ارزی اند. دوم این که در *input-portion* هالت اتفاق بیفتد در این صورت نیز از آنجا که $f_{w_1z}(q_i) = f_{w_2z}(q_i)$ برای هر q_i باز هم این دو رشته در یک کلاس خواهند افتاد.

برای پایه استقرا *(Basis)* از بلاک شماره‌ی 2 به بلاک شماره‌ی 4 می‌رویم:

$$\underbrace{first_{w_1z}\Box\Box\ldots\overset{*}{\vdash}q_i\alpha}_{B_2}\overset{1}{\vdash}\underbrace{f_{w_1z}(q_i)\alpha\overset{*}{\vdash}q_l\beta}_{B_4}$$

در چند مرحله پیش نشان دادیم که $first_{w_1z} = first_{w_2z}$ پس به راحتی با جایگذاری داریم:

$$\underbrace{first_{w_2z}\Box\Box\ldots\overset{*}{\vdash}q_i\alpha}_{B_2}\overset{1}{\vdash}\underbrace{f_{w_2z}(q_i)\alpha\overset{*}{\vdash}q_l\beta}_{B_4}$$

در آخرین کانفیگوریشن B_2 معادل α از این استنتاج آورده شده است که در هردوی رشته‌ها کانفیگوریشن اول برابر بوده و همچنین δ نمی‌تواند غیرقطعی عمل کند. $f_{w_1z}(q_i)$ با $f_{w_2z}(q_i)$ به این علت جایگزین شده که در فرض $f_{w_1}(q_i) = f_{w_2}(q_i)$

بوده و پیش از این نیز اثبات کردیم که این فرض نتیجه می‌دهد که به ازای هر z نیز گزاره‌ی $f_{w_1z}(q_i) = f_{w_2z}(q_i)$ بر قرار است.

ادامه‌ی استقرا نیز به همین روشنی از تعاریف و اثباتهای پیشین نتیجه می‌شود:

$$\underbrace{q_x\alpha\overset{*}{\vdash}q_i\alpha'}_{B_{2k}}\overset{1}{\vdash}\underbrace{f_{w_1z}(q_i)\alpha'\overset{*}{\vdash}q_l\beta}_{B_{2(k+1)}}$$

بر اساس فرض استقرا در ابتدای بلاک متناظر در رشته‌ی w_2z هم کانفیگوریش زیر را داریم:

$$\underbrace{q_x\alpha\vdash\ldots}_{B_{2k}}$$

از آن‌جا این دو کانفیگوریشن برابر است در ادامه‌ی بلاک هم داریم:

$$\underbrace{q_x\alpha\overset{*}{\vdash}q_i\alpha'}_{B_{2k}}$$

حال برای *non-input-block* بعدی نیز داریم:

$$\underbrace{q_x\alpha\overset{*}{\vdash}q_i\alpha'}_{B_{2k}}\overset{1}{\vdash}\underbrace{f_{w_2z}(q_i)\alpha'\vdash\ldots}_{B_{2(k+1)}}$$

براساس اثبات‌های قبل می‌دانیم که $f_{w_1z}(q_i) = f_{w_2z}(q_i)$ پس از آنجا که رفتار ماشین تورین به ازای کانفیگوریشن‌های یکسان، قطعی و برابر است در ادامه نیز داریم:

$$\underbrace{q_x\alpha\overset{*}{\vdash}q_i\alpha'}_{B_{2k}}\overset{1}{\vdash}\underbrace{f_{w_2z}(q_i)\alpha'\overset{*}{\vdash}q_l\beta}_{B_{2(k+1)}}$$

در نتیجه w_1 و w_2 جدا ناشدنی از M هستند.

■

تا به حال نشان دادیم که اگر برای دو رشته‌ی w_1 و w_2 داشته باشیم $first_{w_1} = first_{w_2}$ و برای هر $q_i \in Q$ داشته باشیم $f_{w_1}(q_i) = f_{w_2}(q_i)$ پس w_1 و w_2، *indistinguishable* از M هستند. از آنجا که تعداد حالات ممکن برای $first_w$ برابر $|Q|$ بوده و همچنین تعداد توابع ممکن برای تعریف f_w برابر $|Q|^{|Q|}$ است. پس تعداد کلاس‌های هم ارزی محدود و برابر $|Q|^{|Q|+1}$ است. از آن‌جا که تعداد کلاس‌های هم‌ارزی محدود است طبق قضیه *Myhill-Nerode* که قضیه‌ی 1.56 منبع [1] است، زبان توصیفی M منظم است. از آنجا که M یک نمونه دلخواه از RO_n بود پس هر نمونه از RO_n زبانی منظم را توصیف می‌کند.

■

توجه شود که این اثبات مبتنی بر تشخیص درست out-event و in-event ها است. در نتیجه رشته‌ی ϵ یک حالت خاص محسوب می‌شود. پس شاید نشود f_w را برای $w = \epsilon$ به شکل f_ϵ تعریف کرد. در نهایت احتمالا این اثبات برای رشته‌ی ϵ کار نکند. با این صحبت‌ها، ما در این اثبات زبان را ϵ-free می‌گیریم. یا به عبارتی اثبات گفته شده برای $L - \{\epsilon\}$ درست است.

ب) نشان می‌دهیم که اگر الگوریتمی برای تبدیل یک نمونه RO_n به DFA داشته باشیم، می‌توانیم برای زبان L_ϵ که زبانی تصمیم‌ناپذیر است، یک $Decider$ طراحی کنیم.

$$L_\epsilon = \{\langle M\rangle | \epsilon \in L(M)\}$$

اگر برای تبدیل هر RO_n به یک DFA الگوریتم وجود داشته باشد. پس برای آن ماشین تورینگی وجود دارد که یک RO_n را به عنوان ورودی گرفته و DFA معادل را برمی‌گرداند نام این ماشین تورینگ را A می گذاریم. حال می‌خواهیم $Decider$ برای L_ϵ بسازیم. نام این $Decider$ را T_ϵ می‌گذاریم.

برای هر ورودی $\langle M\rangle$ ابتدا به M یک حالت q_{dummy} اضافه می‌کنیم. قرار است روی این حالت تا انتهای راست ورودی پیش برویم. پس برای هر $a \in \Sigma$ تابع $\delta(q_{dummy}, a) = (q_{dummy}, a, R)$ را تعریف می‌کنیم. برای کارکتر $Blank$ تابع $\delta(q_{dummy}, \Box) = (q_0, \$, R)$ را تعریف می‌کنیم که $\$ \notin \Gamma$ و q_0 همان حالت آغازین ماشین M است. حال می‌خواهیم علامت $\$$ در سمت راست ورودی نقش نگهبان بخش ورودی را بازی کند و همچنین جوری باشد که انتهای نوار از سمت چپ را شبیه سازی کند، به این ترتیب برای هر $q_i \in Q$ تابع $\delta(q_i, \$) = (q_i, \$, R)$ را تعریف می‌کنیم. به این ترتیب اگر به $\$$ بر خورد کند دوباره به اولین کارکتر پس از $\$$ بر می‌گردد. مشاهده شود که این ماشین یک RO_n است. چرا که نمی‌تواند هدش را روی ورودی‌اش بازگرداند. همچنین مشاهده شود که ماشین تغییر یافته که نام آن را M' می‌گذاریم، تنها در صورتی $Accept$ می‌کند که ماشین M رشته‌ی ϵ را بپذیرد. چرا که ماشین M را روی ورودی ϵ شبیه سازی می‌کند:

$$\epsilon \in L(M) \leftrightarrow L(M') = \{\epsilon\} \leftrightarrow \epsilon \in L(M')$$

$$\epsilon \notin L(M) \leftrightarrow L(M') = \{\ \} \leftrightarrow \epsilon \notin L(M')$$

مشاهده شود که تعداد حالات و $Transition$ های اضافه شده محدود و قابل محاسبه است. پس ساخت M' الگوریتمیک و انجام پذیر در زمان متناهی است.

حال M' را به A می‌دهیم. نتیجه‌ی آن یک DFA مثلا با نام D است. عضویت ϵ در DFA تصمیم‌پذیر است. اگر q_0 حالت آغازین D باشد. کافیست چک کنیم که q_0 عضو حالات پایانی هست یا خیر. به این ترتیب T_ϵ یک $Decider$ برای L_ϵ است. ولی می‌دانیم که $L_\epsilon \notin R$ پس این تناقض است. به این ترتیب چنین الگوریتمی وجود ندارد.

پرسش ۲

برای هر دو زبان A و B زبان C وجود دارد به شکلی که $A \leq_M C$ و $B \leq_M C$ برقرار باشد.

برای هر دو زبان دلخواه A و B دو زبان A' و B' را به شکل زیر تعریف می‌کنیم:

$$A' = \{0w | w \in A\}$$
$$B' = \{1w | w \in B\}$$

حال مجموعه‌ی C را به شکل $C = A' \cup B'$ تعریف می‌کنیم.

سپس تابع f_1 را این گونه تعریف می‌کنیم:

$$f_1(w) = 0w$$

و تابع f_2 را به شکل زیر تعریف می‌کنیم:

$$f_2(w) = 1w$$

حال داریم:

$$w \in A \Leftrightarrow f_1(w) \in C \Leftrightarrow A \leq_M C$$

$$w \in B \Leftrightarrow f_2(w) \in C \Leftrightarrow B \leq_M C$$

روشن است که الحاق یک کارکتر به رشته محاسبه پذیر است. در نتیجه f_1 و f_2 محاسبه پذیرند.

پرسش ۳

ابتدا تابع s را به شکل زیر تعریف می‌کنیم:

$$s(x,y) = \sum_{i=0}^{y} g(x,i)$$

ادعا می‌کنیم که s بازگشتی اولیه است:

$$s(x,y+1) = s(x,y) + g(x,y+1)$$
$$s(x,0) = g(x,0)$$

برای پایه $y = 0$ داریم:

$$s(x,0) = \sum_{i=0}^{0} g(x,i) = g(x,0)$$

می‌دانیم که g بازگشتی اولیه است، پس $g(x,0)$ حالت $Base\ Case$ آن نیز اولیه است. حال نام $Base\ Case\ Function$ تابع s را h' می‌گذاریم. در این صورت داریم:

$$h' = g(x,0) = g \circ (P_1^1, f_0)(x)$$

که P همان تابع Projection و f_0 همان تابع zero است. در نتیجه h' بازگشتی اولیه است.

حا داریم :

$$s(x,y) = \sum_{i=0}^{y} g(x,i)$$

پس برای $y+1$ داریم:

$$s(x,y+1) = \sum_{i=0}^{y+1} g(x,i) = \sum_{i=0}^{y} g(x,i) + g(x,y+1) = s(x,y) + g(x,y+1)$$

یا به عبارتی:

$$s(x,y+1) = add(s(x,y), g(x,S(y)))$$

که S همان Successor است. می‌دانیم که add بازگشتی اولیه است. همچنین می‌دانیم که g بازگشتی اولیه است. همچنین S نیز بازگشتی اولیه است. حال اگر نام $Recursive\ Case\ Function$ تابع s را $g'(x,y,s(x,y))$ بگذاریم، داریم:

$$g' = add \circ (P_3^3, g \circ (P_1^3, S \circ P_2^3))$$

توجه شود که s با S متفاوت است. s اول که Lower-Case است، همان تابعی است که تعریف کردیم، در صورتی که S دومی که Capital است، همان Successor است. دیدیم که همه‌ی توابع داخلی ترکیب بازگشتی اولیه‌اند، ترکیب نیز بازگشتی اولیه است. در نتیجه g' نیز بازگشتی اولیه است. در نهایت تابع اصلی یا $s = \rho^1(h', g')$ نیز بازگشتی اولیه است.
در صفحه‌ی بعد بازگشتی اولیه بودن f را نشان می‌دهیم.

حال می‌خواهیم نشان دهیم که تابع زیر نیز بازگشتی اولیه است:

$$f(x) = \sum_{i=0}^{x} g(x,i)$$

اگر حالت پایه‌ی تابع f را h'' بگیریم. برای نشان دادن بازگشتی اولیه بودن آن داریم:

$$h'' = f(0) = g(0,0) = h'(0)$$

مشاهده شود که $h'' = g(0,0)$ اولیه و تابعی به شکل $\mathbb{N}^0 \to \mathbb{N}$ است و مقدار ثابتی دارد. همچنین به صورت استقرایی ثابت می‌شود که هر f_c که تابعی به شکل $\mathbb{N}^0 \to \mathbb{N}$ باشد بازگشتی اولیه است.

اثبات اینکه هر f_c یا توابع 0-ary بازگشتی اولیه هستند بسیار ساده و با استقراست. برای $c = 0$ می‌دانیم که f_0 بازگشتی اولیه است. اگر f_c بازگشتی اولیه باشد. آن‌گاه $S(f_c) = f_{c+1}$ می‌باشد. از آن‌جا که 0-ary است و ورودی‌ای ندارد که روی آن بازگشت بزند. پس پایه نیازی ندارد و همچنین $S(f_c) = f_{c+1} = S \circ f_c$ که همه‌ی اجزا بازگشتی اولیه‌اند. تاکید می‌شود که منظورمان از f_c اینجا تابع ثابت **بدون آرگومان** است. یعنی هیچ ورودی دریافت نمی‌کند. در واقع به نظر می‌رسد که این‌ها همان اعداد ثابت در حساب $Peano$ باشند.

دیدیم که h'' بازگشتی اولیه است.

اگر حالت بازگشتی تابع f را با g'' نشان دهیم، برای نشان دادن بازگشتی اولیه بودن آن داریم:

$$g'' = f(x+1) = \sum_{x+1}^{i} g(x,i) = s(x+1, x+1)$$

یا به عبارتی:

$$g'' = s(S(x), S(x)) = s \circ (S(P_1^2), S(P_1^2))$$

(s کوچک با S بزرگ تفاوت دارد)

ابتدا s را نشان دادیم که بازگشتی اولیه است. می‌دانیم که S که همان Successor است هم بازگشتی اولیه است. پس ترکیب این‌ها نیز بازگشتی اولیه‌اند. در نهایت این‌که g'' نیز بازگشتی اولیه است. توانستیم تابع f را به شکل $f = \rho^1(h'', g'')$ تعریف کنیم. پس f بازگشتی اولیه است.
■

پرسش ۴

آ) می‌خواهیم فرم بازگشتی اولیه تابع زیر را بنویسیم:

$$f(x, y) = 2x + 3y$$

این کار را با دانش بر اینکه *mult(x,y)* که تابع ضرب است و *add(x,y)* که تابع جمع است، بازگشتی اولیه هستند انجام می‌دهیم.
اگر *Base Case function* تابع را h نام‌گذاری کنیم داریم:

$$h = f(x, 0) = 2x = mult(2, x) = mult(S(S(f_0)), P_1^1)$$

چراکه همه‌ی اجزای آن بازگشتی اولیه‌اند و *Composition* توابع نیز بازگشتی اولیه است. همچنین فرقی ندارد برای *Composition* از نماد $\circ$ استفاده کنیم یا آن‌که مثلا $(f \circ g)(x)$ را به شکل $f(g(x))$ بنویسیم. برای راحت خوانده شدن به این فرم نوشتیم.

اگر *Recursive Case function* را g نام‌گذاری کنیم داریم:

$$g = f(x, y + 1) = f(x, y) + 3 = add(P_3^3, S(S(S(f_0))))$$

روشن است که g بازگشتی اولیه است. چراکه همه‌ی اجزای آن بازگشتی اولیه‌اند و ترکیب متناهی اینها نیز بازگشتی اولیه است. در نتیجه می‌توانیم تابع f را به شکل $\rho^1(h, g)$ بنویسیم. پس f بازگشتی اولیه است.

ب) می‌خواهیم نشان دهیم تابع زیر بازگشتی اولیه است:

$$f(x, y) = |x - y|$$

با تعریف monus و بازگشتی اولیه بودن آن پیش از این آگاه شدیم. برای تاکید $x \dot{-} y$ برابر $x - y$ است اگر که $x \geq y$ و در غیر این صورت برابر 0 است. مشاهده شود که $2 = 3 \dot{-} 5$ ولی $0 = 5 \dot{-} 3$ با این تعریف می‌توانیم تابع absolute differebce را با ترکیب monus و add به شکل زیر تعریف کنیم:

$$f(x, y) = |x - y| = add(x \dot{-} y, y \dot{-} x)$$

تابع monus را با *sub* نشان می‌دهیم. حال برای تعریف f داریم:

$$f = add(sub(x, y), sub(y, x)) = add(sub(P_1^2, P_2^2), sub(P_2^2, P_1^2))$$

توجه شود که تابع f را به صورت مستقیم بدون بازگشت توسط ترکیب چند تابع بازگشتی اولیه دیگر تعریف کردیم.

پ) می‌خواهیم نشان دهیم تابع زیر بازگشتی اولیه است:

$$f(x, y) = max(x, y)$$

ابتدا مشاهده شود که

$$f(x, y) = max(x, y) = x + (y \dot{-} x)$$

چراکه اگر $y \geq x$ آن‌گاه $0 = y \dot{-} x$ و در نتیجه $x = x + 0$ خواهد بود. اگر هم $x \geq y$ آن‌گاه $y \dot{-} x = y - x$ در نتیجه

$$x + (y - x) = x + y - x = y$$

پس این گزاره درست است.

از آنجا که دو تابع monus و add بازگشتی اولیه اند. می‌توانیم به صورت مستقیم بدون تعریف بازگشتی تابع، f را به فرم بازگشتی اولیه تعریف کنیم:

$$f = add \circ (P_1^2, sub(P_2^2, P_1^2))$$

پرسش ۵

آ) می‌خواهیم قضیه‌ی رایس را به کمک قضیه‌ی بازگشت نشان دهیم. برای این کار ابتدا می‌بایست عددگذاری گودلی را تعریف کنیم.

عددگذاری گودلی *(Gödel numbering)* تابعی است که به هر سمبول از یک WFF یا *Well-Formed Formula* از یک زبان **فرمال** (*formal*) یک عدد نسبت می‌دهد.

او از روشی به نام *Prime Factorization* برای کدگذاری استفاده می‌کرد. برای مثال در حساب Peano او به تابع S یا Successor عدد 3 را نسبت می‌دهد.

تعریف WFF یا جملات خوش ساخت را از منطق می‌دانیم. برای یک مثال کوتاه: گودل به نماد تابعی تک موضعی 1-ary آشنای Successor عدد اول چهارم یا 3 را نسبت می دهد. به 0 یا همان f_0 هم عدد اول یکم یا 1 را نسبت می‌دهد. به این ترتیب معادل جمله‌ی خوش ساخت $S(S(0))$ برابر $3 \times 3 \times 1 = 9$ است.

یک نکته این‌که عددگذاری گودلی می‌تواند به نحوهای مختلفی انجام شود و Prime Factorization که گودل از آن استفاده کرد تنها راه ممکن نیست.

خواننده می‌تواند تصور کند که چگونه هر تابع بازگشتی اولیه‌ای که تعریف کردیم دارای عدد گودلی متناظر یکتای خودشان هستند. ولی آیا توابع محاسبه پذیر یا توابع کامل همه‌ی آنچه هستند که یک زبان فرمال توصیف می‌کند؟ خیر، توابع Partial Recursive نیز وجود دارند که به ازای برخی ورودی‌ها جواب ندارند. همانطور که ماشین‌های تورینگی هستند که به ازای برخی ورودی‌ها توقف نمی‌کنند.

هر ماشین تورینگ یک زبان فرمال را توصیف می‌کند و یک راه کدگذاری آن را از [۴] می‌دانیم. کافیست هر δ که به شکل یک پنج‌تایی (q_i, a_x, q_j, a_y, D) هست را به شکل مناسبی کد باینری کنیم و این‌ها را با لحاظ کردن ترتیب هر دلتا به شکل δ_i که i امین تابع انتقال است و گذاشتن جداکننده *(Seperator)* بین هر تابع انتقال، به هم بچسبانیم و در نهایت به ابتدای آن یک 1 الحاق کنیم به این ترتیب یک Bijection به شکل $TMs \leftrightarrow \mathbb{N}$ داریم. مشاهده شود که روش کدگذاری هر ماشین تورینگ که در تمرین‌های پیشین و هم منبع [۴] معرفی شده یک *(Gödel numbering)* است. چراکه هر ماشین تورینگ یک WFF است و همانطور که برای $(p_1 \wedge p_2) \vee p_3$ که یک WFF است مجموعه‌ای از Assignment ها به اتمها وجود دارد که این گزاره را درست می‌کند، در ماشین تورینگ نیز مجموعه‌ای از w ها در یک WFF وجود دارند که ماشین تورینگ ما آن‌ها را می‌پذیرد. هر $Assignment$ به اتمها با یک تابع Valuation با نماد v_i نمایش داده می‌شود. این تابع به همه‌ی اتمها که مجموعه‌ی نامتناهی به شکل زیر هستند:

$$P = \{p_0, p_2, p_3, p_4, p_5, \dots\}$$

مقداری در $\{0, 1\}$ انتساب می‌کند. به عبارتی:

$$v_i : P \to \{0, 1\}$$

به این ترتیب هر v_i یک رشته‌ی باینری نامتناهی را توصیف می‌کند. ولی در یک WFF ثابت می‌شود که تنها مقادیر مهم در تعیین صدق آن‌ها مقادیر اتمهای استفاده شده در آن هستند. پس برای WFF منطقی بالا تنها $\{p_1, p_2, p_3\}$ تعیین کننده اند و همچنین خود این زبان محدود شده به متغیرهای تعیین‌کننده‌اش و مکملش مجموعه‌ای محدودند ولی اگر توابع Valuation را در نظر بگیریم که این گزاره را درست می‌کنند و آنهایی را که گزاره را غلط می‌کنند، هر دو نامحدودند. هدف از این توضیحات این است که هر $w \in \Sigma^*$ یک v_i است. به عبارتی w مقداردهی به متغیرهای مساله‌ای است که ما به آن مساله، زبان می‌گوییم. همانطور که v_i مقدار دهی به اتم‌های جملات خوش‌ساخت است. پس هر w یک نمونه از یک مساله است که این نمونه از مساله یا درست است یا غلط و یا اینکه نمی‌توان گفت که درست است یا غلط برای مثال زبان پارادوکس راسل را در نظر بگیریم:

$$L_{russel} = \{x | x \notin x\}$$

که این معادل همان زبان قطری است:

$$L_d = \{\langle M \rangle | \langle M \rangle \notin L(M)\}$$

نمونه‌ی x در زبان L_{russel} یا $\langle M_d \rangle$ در زبان L_d دو نمونه هستند که در مساله مربوطه‌اشان جواب ندارند.

در ترجمه‌ی بازگشتی تصمیم‌ناپذیری، می‌گوییم ورودیهایی هستند که به ازای آن‌ها تابع جواب ندارد. آن چه می‌خواهیم اثبات کنیم این است که هر خاصیت معنایی غیربدیهی F برای توابع بازگشتی، تصمیم‌ناپذیر بوده یا اینکه تابع بازگشتیِ آن Partial است.

برای اثبات نیاز است نشان دهیم برای هر TM یک تابع بازگشتی داریم. تابع بازگشتی TM دلخواه M را از روی δ می‌سازیم به نحوی که به درستی هر کانفیگوریشن C_i را به کانفیگوریشن بعدی C_j ببرد. هر کانفیگوریشن توسط توصیف آنی یا (instantaneous description) آن به شکل $d_i = (X_1X_2X_3X_4 \dots q_iX_j \dots)$ نمایش داده می‌شود. برای راحتی در طراحی پیاده‌سازی، هر d_i را به شکل $d_i = (pos(q_i), q_i, X_1X_2X_3X_4 \dots)$ نمایش می‌دهیم که $pos(q_i)$ پوزیشن فعلی هد و q_i حالت فعلی می‌باشد. مشاهده شود که برای هر d_i یک کدینگ منحصر به فرد به شکل $d_i = 2^{pos(q_i)} \times 3^{q_i} \times 5^{X_1} \times 7^{X_2}, \dots, PrNo(n+2)^{X_n}$ وجود دارد. که $PrNo(n)$ عدد اول n ام را برمی‌گرداند. می‌خواهیم تابعی بازگشتی بسازیم که $f(d_i) = d_j$ اگر و تنها اگر که $d_i \vdash_M d_j$ برقرار باشد. توجه شود که این کار را برای یک δ_i دلخواه انجام می‌دهیم. سپس نشان می‌دهیم چگونه می‌توان تابعی بازگشتی تعریف کرد که به ما نشان دهد کدام δ_i باید اجرا شود.

برای یک δ_i دلخواه داریم: $\delta_i = (q_i, a_x, q_j, a_y, D)$ حال f_{δ_i} ابتدا باید q_i را به q_j تغییر دهد. سپس اگر $D = R$ بود مقدار $pos(q_i)$ را با $pos(q_i) + 1$ به روزرسانی کند یا برای $D = L$ آن را با مقدار $pos(q_i) - 1$ به روز رسانی کند. همچنین $X_{pos(q_i)}$ باید از مقدار a_x به مقدار a_y به روز رسانی شود. پس یک substitution برای تغییر حالت فعلی و یک substitution برای تغییر $X_{pos(q_i)}$ داریم که بازگشتی‌اند. همچنین $+1$ یا -1 نیز به وضوح بازگشتی‌اند. به صورت خلاصه نشان دادیم که برای هر δ_i یک تابع f_{δ_i} داریم که آن را شبیه سازی می‌کند. هنوز نشان ندادیم که f وجود دارد که $d_i \vdash_M d_j$ را به درستی شبیه‌سازی کند.

اینکه از چه δ_i استفاده شود وابسته این است که هد روی چه پوزیشنی است و حالت فعلی کدام است. مقدار خانه‌ی پوزشین فعلی هد از پارامتر اول تابع به دست می‌آید، یعنی $X_{P_1^n}$ و حالت فعلی پارامتر دوم تابع یعنی P_2^n است. تابعی که δ_i بعدی را مشخص می‌کند تابعی است از $Q \times \Gamma$ به $\{1, 2, ..., k\}$ که k تعداد توابع δ است. می‌دانیم که تعداد توابع انتقال محدودند پس تابع *co-domain* محدود دارد. می‌دانیم که Γ و Q محدود است. پس تابع *domain* محدود دارد. در نتیجه تابع با Lookup-Table قابل پیاده سازی است.

حال ابتدا با تابع دومی که تعریف کردیم f_{δ_i} که باید استفاده شود را از روی ورودی که یک توصیف آنی d_x است به دست می‌آوریم. سپس f_{δ_i} را روی d_x اجرا می‌کنیم و به درستی d_y را از آن استنتاج می‌کنیم. حالتهای پایه‌ی این بازگشت شروط پایان ماشین تورینگ هستند. یعنی به ازای *Accepting-Configuration* تابع مقدار 1 و برای *Rejecting-Configuration* تابع مقدار 0 برمی‌گرداند. به صورت شهودی حس می‌شود که تابع لزوما کامل نیست. به صورت خلاصه اثبات کردیم که برای هر ماشین تورینگ M یک تابع بازگشتی f وجود دارد که آن را شبیه‌سازی می‌کند.

حال می‌خواهیم نشان دهیم که هر تابع بازگشتی توسط یک TM قابل پیاده سازی است. این‌کار مطابق منبع [5][pp. 70 71] انجام می‌شود. یعنی به سادگی می‌بینیم که همه‌ی پایه های‌یک تابع بازگشتی مانند zero و successor و projection و composition و search قابل پیاده سازی‌اند. برای مثال به ازای composition ماشین تورینگ تو در تو داریم. از آن‌جا که توابع بازگشتی نیز به صورت استقرایی تعریف می‌شوند، مشاهده این‌که برای هر تابع f و g برای مثال اگر TM با نامهای M_f و M_g داشته باشیم چگونه $f \circ g$ به شکل $M_{f \circ g}$ قابل ساخت است آسان و استقرایی است.

حال می‌دانیم که بین *Partial Recursive Functions* و *Turing Machines* یک Bijection داریم. پس به ازای هر تابع بازگشتی Partial یک TM داریم و به ازای هر TM یک تابع بازگشتی Partial داریم.

حال می‌دانیم که برای TM ها و اعضای $P^{(1)}$ که توابع بازگشتی Partial هستند، یک عددگذاری گودلی داریم. این

عدد گذاری را به شکل تابع $\phi : \mathbb{N} \to P^{(1)}$ تعریف می‌کنیم. حال ϕ_e را این‌گونه تعریف می‌کنیم:

$$\phi_e := \phi(e)$$

یعنی ϕ_e همان e امین تابع بازگشتی partial است. هر خصوصیت معنایی با نام F به شکل $F \subseteq P^{(1)}$ است. یعنی بخشی یا همه‌ی توابع بازگشتی آن را دارند. همچنین از این پس منظورمان از تابع بازگشتی همان تابع بازگشتی partial است. در نتیجه تابع بازگشتی ϕ_e خصوصیت معنایی F را دارد اگر و تنها اگر که $\phi_e \in F$ برقرار باشد. روشن است که برای هر F یک Decision Problem قابل تعریف است به این شکل که کدام یک از گزاره‌های $\phi_e \in F$ یا $\phi_e \notin F$ درست است. همانند آنچه در ماشین‌های تورینگ داشتیم. اینجا ϕ_e هم‌ارز TM_e است و این که آیا خصوصیت معنایی متناظر F را دارد یا خیر. نام مساله‌ی تصمیم گیری خاصیت F را نیز D_F می‌گذاریم. چند خصوصیت معنایی غیر بدیهی در زبان محاسبات بازگشتی را در ادامه می‌آوریم:

۱. آیا تابع داده شده کامل است؟

۲. آیا تابع داده شده به ازای هر ورودی 0 برمی‌گرداند؟

۳. آیا تابع داده شده به ازای حداقل یک ورودی 0 برمی‌گرداند؟

۴. آیا تابع داده شده یک تابع ثابت (به ازای هر ورودی یک مقدار ثابت برگرداند) است؟

ترجمه‌ی قضیه‌ی رایس در زبان محاسبات بازگشتی ادعا دارد که مساله تصمیم‌گیری D_F تصمیم‌پذیر یا تابع بازگشتی کامل است اگر و تنها اگر $F = \emptyset$ یا $F = P^{(1)}$ درست باشد.

توجه شود که یک خاصیت F غیربدیهی است اگر و تنها اگر که $F \neq \emptyset \wedge F \neq P^{(1)}$ ، یعنی اگر حداقل یک تابع بازگشتی در F باشد و همچنین حداقل یک تابع بازگشتی در F نباشد. در این صورت مساله تصمیمی این خصوصیت F یا همان D_F تصمیم‌ناپذیر است.

حال قضیه‌ی بازگشت را با تعاریفی که نسبت به تناظر ماشین‌های تورینگ و توابع بازگشتی داشتیم این‌گونه ترجمه می‌کنیم: *"برای هر تابع بازگشتی $Q(x, y)$ یک تابع بازگشتی با اندیس گودلیِ e به شکل $\phi_e(y)$ وجود دارد که مقدار $Q(e, y)$ را برمی‌گرداند."* به صورت شهودی می‌توان حس کرد که متناظر تورینگیِ ϕ_e ماشینی است که سورس کد خود را به عنوان خروجی برمی‌گرداند. ترجمه‌ی سورس کد در توابع بازگشتی همان عدد گودلی تابع ϕ_e می‌باشند. یعنی ϕ_e یک *self-replicating function* است با این تفاوت که به جای اینکه به صورت مستقیم عدد گودلی‌اش را برگرداند، ابتدا آن را به Q پاس داده و سپس نتیجه‌ی Q روی عدد گودلی‌اش را برمی‌گرداند.

اگر F را مطابق تعریفهایی که کردیم یک خصوصیت معنایی غیربدیهی برای توابع بازگشتی در نظر بگیریم آن‌گاه طبق فرض $F \neq \emptyset \neq P^{(1)}$ است.

حال می‌دانیم که تابعی در $P^{(1)}$ وجود دارد با نام f که خاصیت F را دارد یا به تعریفی $f \in F$. همچنین تابعی وجود دارد که این خصوصیت را ندارد یا به عبارتی $g \notin F$. فرض کنیم که مجموعه‌ی اندیسهای گودلی x به شکلی که $\phi_x \in F$ دارای یک تابع بازگشتی کامل‌اند یا به عبارتی تصمیم‌پذیرند. این مجموعه‌ی تصمیم پذیر به شکل زیر است:

$$L_F = \{x | \phi_x \in F\}$$

پس به عبارتی تابع کاملی مثل $Q(x, y)$ برای آنها وجود دارد به شکلی که اگر $\phi_x \in F$ بود $g(y)$ را برمی‌گرداند و اگر $\phi_x \notin F$ بود $f(y)$ را برمی‌گرداند. به عبارتی $Q(x, y)$ خصوصیت F را $Decide$ می‌کند و سپس آرگومان دومش y را اگر جواب بله بود به f و اگر جواب نه بود به g پاس می‌دهد. به این ترتیب اگر $x \in L_F$ آن‌گاه $Q(x, y) = f(y)$ و در غیر این‌صورت $Q(x, y) = g(y)$:

$$\phi_x \in F \leftrightarrow Q(x, y) = f(y)$$

$$\phi_x \notin F \leftrightarrow Q(x,y) = g(y)$$

حال براساس قضیه‌ی بازگشت می‌دانیم که یک اندیس e وجود دارد به شکلی که

$$\phi_e(y) = Q(e,y)$$

ولی اگر $\phi_e \in F$ آن‌گاه $\phi_e(y) = g(y)$ و براساس تعریف می‌دانیم که g تابعی بود که خصوصیت F را ندارد یا به عبارتی $g \notin F$ پس با جایگذاری داریم $\phi_e \notin F$ که این تناقض است. اگر هم ϕ_e این خاصیت را نداشت یعنی $\phi_e \notin F$ آن‌گاه $\phi_e = f$ که می‌دانیم f تابعی بود که خصوصت F را داشت یا به عبارتی $f \in F$ پس $\phi_e \in F$ که این هم تناقض است. در هر دو حالت به تناقض خوردیم. پس $Q(x,y)$ کامل نیست چراکه به ازای ϕ_e جواب ندارد. در نتیجه L_F تصمیم‌پذیر نیست. همچنین F یک خصوصیت دلخواه بود. پس هر خصوصیت نابدیهیِ معنایی برای توابع بازگشتی تصمیم ناپذیر است. به این ترتیب قضیه‌ی رایس در زبان توابع بازگشتی و به وسیله‌ی قضیه‌ی بازگشت اثبات می‌شود. [۶]

■

مراجع

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

[۲] Kozen, Dexter C. (1997), *Automata and Computability* ,Undergraduate Texts in Computer Science

[۳] https://cs.stackexchange.com/users/8321/babou https://cs.stackexchange.com/questions/22082/single-tape-turing-machines-with-write-pro

[۴] Hopcroft, John E., and Jeffrey D. Ullman. (1969), *Formal languages and their relation to automata.* , Addison-Wesley Longman Publishing Co., Inc.

[۵] Enderton, Herbert B. (2010), *Computability theory: An introduction to recursion theory* , Academic Press

[۶] https://en.wikipedia.org/Rice's_theorem https://en.wikipedia.org/wiki/Rice%27s_theorem#Proof_by_Kleene's_recursion_theorem