

مقدمه ای بر نظریه محاسبه

۱ مقدمه

فرض کنید که می خواهیم یک مسئله معین مثل پیدا کردن ویژه مقدارهای یک ماتریس را حل کنیم. برای این کاریک آلوگوریتم معین بکار می بریم و آلوگوریتم خود را بایک زبان برنامه نویسی روی یک رایانه پیاده می کنیم. آیا این مسئله یک مسئله ساده است؟ آیا این آلوگوریتم یک آلوگوریتم موثر برای حل این مسئله است؟ چگونه می توان تعریفی از محاسبه بدست داد؟ چگونه می توان تعریفی از مسائل حل پذیر و حل ناپذیرارایه داد؟ دشواری یک مسئله را با چه معیاری می توان سنجید؟ دشواری یک آلوگوریتم را چگونه می توان سنجید؟ آیا مسائلی وجود دارند که هرگز نتوان دربارۀ درستی یا نادرستی آنها تصمیم گرفت؟ آیا هیچ وجه مشترکی بین یک دستگاه اتوماتیک که برای کنترل درب فروشگاه ها بکار می رود و یک ابررایانه وجود دارد؟ آیا هیچ تفاوت کیفی یا اصولی بین حل کردن یک معادله یک مجهولی خطی مثل $x + 1 = 0$ و حل کردن یک دستگاه معادله دیفرانسیل غیرخطی وجود دارد؟ شاید به عنوان فیزیک پیشه هنوز نتوانیم اهمیت این سوالات را درک کنیم و یا حتی شاید نتوانسته باشیم این سوالات را درست طرح کرده باشیم. شاید بعد از خواندن این فصل و فصل های دیگر بتوانیم بفهمیم که سوال درست چیست و بتوانیم پاسخ بعضی از سوال های بالا را پیدا کنیم. می دانیم در فیزیک بعضی سوالات اهمیت اساسی دارند و یافتن پاسخ آنها باعث دگرگونی اساسی در بنیادهای دانش فیزیک می شود. این نوع سوالات در هر شاخه ای از علم مطرح است. می توان پرسید که برای دانشمندان علوم رایانه چه نوع سوال اساسی ای مطرح است؟

یک رهیافت خوب برای پاسخ دادن به این سوالات آن است که نخست یک تعریف نظری از محاسبه بدست بدهیم، تعریفی که به کلی مستقل از انواع ماشینی باشد که برای محاسبه بکار می بریم. ممکن است با ذکر یک مثال بتوانیم بفهمیم که چرا سوالات بالا لاف بی ربط نیستند. در ریاضیات به مصادیق متفاوتی از فضای برداری برمی خوریم. اگر بخواهیم تک تک این مصادیق را بررسی کنیم شاید هرگز نتوانیم به پیشرفتی در درک روابط پیچیده تری که در آنها وجود دارد نائل شویم. ولی بجای این می توانیم نخست درک درستی از مفهوم فضای برداری که فصل مشترک همه این مصادیق است پیدا کنیم. تنها در این صورت است که خواهیم توانست روابط پیچیده تری را که در یک فضای برداری و به تبع آن در همه مصادیق آن وجود دارد کشف کنیم. به همین ترتیب با مطالعه انواع و اقسام رایانه ها ما هرگز نمی توانیم به سوالات بیشمار از نوع فوق پاسخ گویم. برای این کار می بایست مدلی از محاسبه داشته باشیم که بکلی مستقل از جزئیات ساختاری ماشین محاسبه باشد.

۲ تعاریف مقدماتی

۱.۲ الفبا و زبان

تعریف الفبا و رشته ها : هر مجموعه ای از علائم^۱ یک الفبا^۲ خوانده می شود. یکی از ساده ترین الفبا ها از مجموعه دوحرفی $\Sigma = \{0, 1\}$ تشکیل می شود. به هر کدام از اعضای این مجموعه یک حرف و به هردنباله ای از حروف یک رشته^۳ یا یک کلمه می گوئیم. 000111 و 010101 هردو رشته هایی از الفبای Σ هستند. معمولاً برای نشان دادن رشته ها از حروف s, w و نظایر آن استفاده می کنیم و می نویسیم $s = s_1 s_2 s_3 \dots s_n$ که در آن s_i ها حرف های الفبا هستند. عدد n طول رشته است و می نویسیم $|s| = n$. مجموعه تمام رشته های ممکن از الفبای Σ را با Σ^* نشان می دهیم. واضح است که Σ^* یک مجموعه ی بی نهایت عضوی است. بنابراین رشته های $u = 000111$ و $v = 010101$ هردو عضو Σ^* هستند. رشته تهی رشته ای است که شامل هیچ حرفی نباشد. این رشته بانماد ϵ نشان داده می شود. طول رشته ی ϵ برابر با صفر است. این رشته نیز عضو Σ^* است. هرگاه w یک رشته باشد w^R معکوس آن رشته است. به عنوان مثال $u^R = 111000$ و $v^R = 101010$.

تعریف زبان:

یک زبان روی الفبای Σ ^۴ عبارت است از یک زیرمجموعه از Σ^* .
در زیر چند مثال از زبان های تشکیل شده از الفبای Σ آورده شده است:

$$\begin{aligned} L_1 &= \{w \in \Sigma^* | w \text{ ends with } 0\} \\ L_2 &= \{w \in \Sigma^* | w \text{ contains at least three zeros}\} \\ L_3 &= \{w \in \Sigma^* | w \text{ ends with } 1010\} \end{aligned} \quad (1)$$

یک زبان می تواند یک مجموعه ی متناهی یا یک مجموعه ی نامتناهی باشد. به عنوان مثال مجموعه ی $L_4 = \{00, 11\}$ نیز یک زبان روی الفبای Σ است.

۲.۲ اعمال منظم روی زبان ها

تعریف اعمال مقدماتی روی زبان ها: فرض کنید که A و B دو زبان باشند. در این صورت اجتماع^۵ این دوزبان $A \cup B$ و

Symbols¹
Alphabet²
String³
A language over Σ ⁴
Union⁵

ترکیب⁶ این دو زبان $A \circ B$ به صورت زیر تعریف می شوند:

$$\begin{aligned} A \cup B &= \{x \in \sigma^*, x \in A \text{ or } x \in B\} \\ A \circ B &= \{xy \in \sigma^*, x \in A \text{ and } y \in B\} \end{aligned} \quad (2)$$

به عبارت بهتر اجتماع دو زبان مجموعه تمامی رشته هایی است که در هر دو زبان وجود دارند و ترکیب دو زبان عبارت عبارت است از همه رشته هایی که از پشت سرهم نهادن کلمات اولی و دومی بدست می آید. دقت کنید که لزومی ندارد که هر دو زبان روی یک الفبا تعریف شده باشند.

مثال: هرگاه داشته باشیم

$$\begin{aligned} A &= \{aa, bb, cc\} \\ B &= \{00, 11\}, \end{aligned} \quad (3)$$

آنگاه

$$\begin{aligned} A \cup B &= \{aa, bb, cc, 00, 11\} \\ A \circ B &= \{aa00, aa11, bb00, bb11, cc00, cc11\} \\ B \circ A &= \{00aa, 11aa, 00bb, 11bb, 00cc, 11cc\}. \end{aligned} \quad (4)$$

تعریف: ستاره یک زبان L با نماد L^* نشان داده می شود و عبارت است از:

$$L^* = \{x_1 x_2 \cdots x_n \mid n \geq 0 \text{ and } x_i \in A\}. \quad (5)$$

منظور از $n = 0$ آن است که رشته ی ϵ نیز متعلق به A^* است. بنابراین در مثال قبل داریم

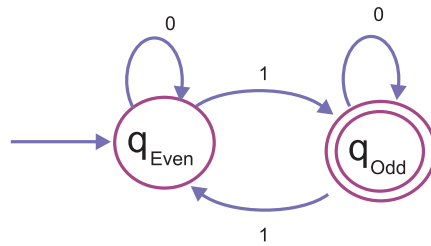
$$A^* = \{\epsilon, aa, bb, cc, aaaa, bbbb, cccc, aabb, aacc, bbcc, bbaa, ccaa, aaaaaa, \dots\}. \quad (6)$$

توجه به این نکته مهم است که بنابر تعریف $\epsilon \in A^*$.

۳ اتومات های متعین و محدود

مطالعه خود را از اساس یک کامپیوتر با ساده ترین مدل آغاز می کنیم که یک اتومات محدود نام دارد. چنین ماشینی تعداد محدودی حالت دارد که آنها را با $q_N, \dots, q_2, q_1$ نشان می دهیم. وقتی که این ماشین علامت s را دریافت می کند تغییر حالت

Concatenation⁶



شکل ۱: یک اتومات تعینی که رشته های شامل تعدادی فرد از 1 را می پذیرد.

داده و به یک حالت جدید می رود. در زندگی روزمره با انواع این ماشین ها سرو کار داریم. به عنوان مثال یک درب اتوماتیک که در ورودی یک فروشگاه بزرگ قرار دارد و یا یک آسانسور یا کنترل کننده ی از راه دور یک وسیله خانگی همگی مثالهایی از اتومات های محدود هستند. از آنجا که ما به مبانی نظریه محاسبه علاقمندیم به این کاربردهای خاص نمی پردازیم و در عوض سعی می کنیم که اتومات های محدود را در یک چارچوب نظری مطالعه کنیم. نخست یک مثال ساده را مطالعه می کنیم. شکل ۱ یک اتومات محدود را نشان می دهد که کارش آن است که پاریته رشته های متشکل از علائم 0 و 1 را تعیین کند. هرگاه رشته ای مثل 0011 به این اتومات داده شود، در حالت q_{Even} و هرگاه رشته ای مثل 01011011 به آن داده شود در حالت q_{Odd} قرار می گیرد. شکل ۱ نشان می دهد که این اتومات وقتی که در حالت های مختلف است چگونه به علائم دریافتی عکس العمل نشان می دهد. در واقع طرز کار این اتومات را می توان در توابع زیر که تابع انتقال خوانده می شوند خلاصه کرد:

$$\begin{aligned} \delta(q_{Even}, 0) &= q_{Even}, & \delta(q_{Even}, 1) &= q_{Odd}, \\ \delta(q_{Odd}, 0) &= q_{Odd}, & \delta(q_{Odd}, 1) &= q_{Even}. \end{aligned} \quad (7)$$

این روابط نشان دهنده یک برنامه هستند که اتومات برطبق آنها عمل می کند. این برنامه به سادگی در شکل ۱ که اصطلاحاً دیاگرام حالت⁷ نامیده می شود نشان داده شده است. می توان گفت که این اتومات رشته های با تعدادی فرد از 1 را پذیرفته و بقیه رشته ها را ردّ می کند. می گوئیم که زبان این ماشین مجموعه تمام رشته هایی است که تعداد فردی عدد ۱ دارند. قبل از آنکه مثال های بیشتری از اتومات های محدود ارائه دهیم، بهتر است که تعریف دقیق و رسمی آن را مشخص کنیم.

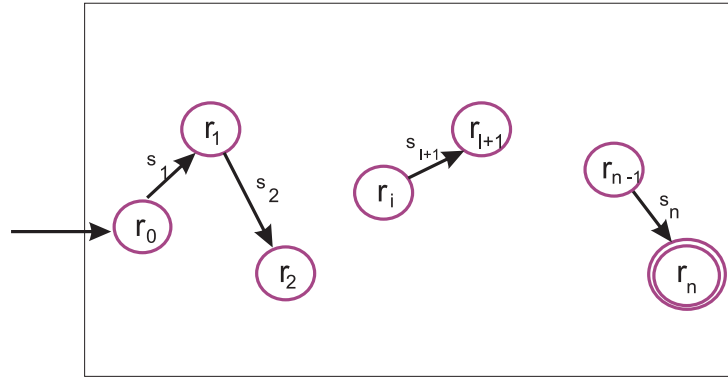
۱.۳ تعریف دقیق از یک اتومات متعین و محدود

تعریف: یک اتومات تعینی محدود⁸ دستگاهی است متشکل از پنج عنصر $DFA = \{Q, \Sigma, \delta, q_0, F\}$ که در آن :

الف : Q یک مجموعه محدود است. $Q = \{q_0, q_1, \dots, q_n\}$. هر عضو Q یک حالت⁹ خوانده می شود.

ب : Σ یک الفباست.

State Diagram⁷
 Deterministic Finite Automata (DFA)⁸
 State⁹



شکل ۲: پذیرش یک رشته توسط ماشین DFA .

ج: $\delta : Q \times \Sigma \longrightarrow Q$ یک نگاشت موسوم به نگاشت انتقال است که به هر حالت و به هر حرف الفبایک حالت دیگر نسبت می دهد.

د: q_0 یک حالت ویژه از Q موسوم به حالت آغازین¹⁰ است و

ه: $F \subset Q$ یک زیرمجموعه از حالات است که مجموعه حالات پذیرش¹¹ نامیده می شود.

هرگاه $w = w_1 w_2 \dots w_n \in \Sigma^*$ یک رشته باشد می گوییم ماشین M ، یا اتومات تعیینی M این رشته را می پذیرد هرگاه ماشین که ابتدا در حالت q_0 است بتواند با خواندن یکی یکی حرف های این رشته از چپ به راست، تغییر حالت داده و سرانجام پس از خواندن آخرین حرف یعنی w_n در یکی از حالت های پذیرش قرار گیرد. ۲
به بیان دیگر این رشته پذیرفته می شود هرگاه زنجیره ای از حالت های ماشین مثل $r_0, r_1, r_2, \dots, r_n \in Q$ وجود داشته باشند به قسمی که:

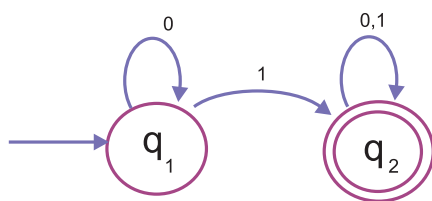
$$r_0 = q_0 \quad ۱$$

$$\delta(r_i, w_i) = r_{i+1} \quad ۲$$

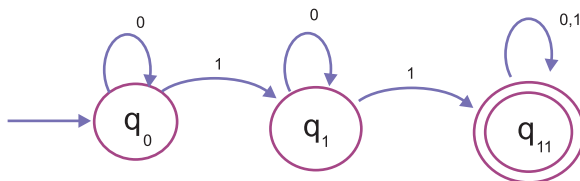
$$r_n \in F \quad ۳$$

دقت کنید که در مجموعه حالت های r_0 تا r_n ممکن است حالت های تکراری وجود داشته باشد، زیرا تعداد حالت های ماشین محدود است ولی طول رشته های پذیرفته شده ممکن است خیلی بیشتر از تعداد حالت های ماشین باشد. مجموعه تمام رشته هایی که توسط ماشین پذیرفته می شوند زبان آن ماشین خوانده می شود و با $L(M)$ نمایش داده می شود. در این حالت می گوییم که ماشین M آن زبان را تشخیص می دهد.

Start State¹⁰
Accept States¹¹



شکل ۳: این اتومات تعیینی رشته‌هایی را می‌پذیرد که حداقل شامل یک نماد 1 باشند.



شکل ۴: این اتومات محدود رشته‌هایی را می‌پذیرد که حداقل شامل دو نماد 1 باشند.

۲.۳ مثالهایی از اتومات‌های متعین و محدود

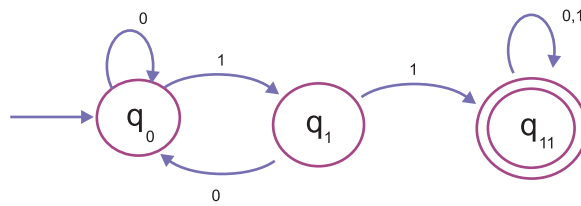
در این زیربخش با چند نمونه از اتومات‌های محدود آشنا می‌شویم. خواننده برای آنکه با طرز کار این اتومات‌ها آشنا شود و هم چنین برای اینکه بتواند اتومات‌هایی را برای زبان‌های دلخواه طراحی کند می‌بایست خود را به جای ماشین بگذارد و به خود بگوید که اگر من بخواهم رشته‌های معینی را تمیز بدهم (یعنی یک زبان معین را شناسایی کنم) چه فرایند فکری را طی خواهم کرد. منظم کردن این فرایند فکری و تدوین آن به صورت یک الگوریتم وی را به خودی خود به سوی طراحی یک اتومات سوق خواهد داد. این روش یعنی هم ذات‌پنداری با ماشین در بخش‌های آینده نیز به خواننده کمک خواهد کرد. ممکن است برای شناسایی یک زبان اتومات‌های متفاوتی به کار برده شود. به عبارت بهتر می‌توان برای شناسایی یک زبان یا حل یک مسئله الگوریتم‌های متفاوتی نوشت. بنابراین بجاست که در این جاتعریفی از ماشین‌های معادل بدست دهیم. می‌گوییم که دو ماشین M_1 و M_2 معادل اند اگر هر دو یک زبان را تشخیص دهند.

تعریف: دو ماشین M_1 و M_2 معادل اند اگر $L(M_1) = L(M_2)$.

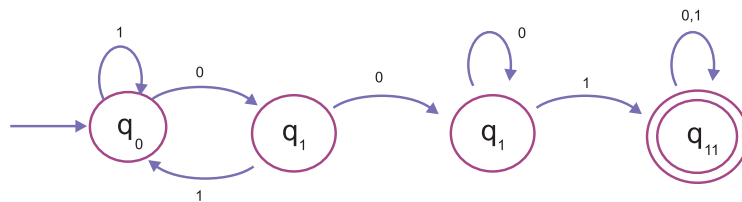
۴ زبان‌های منظم

زبان منظم^{۱۲} به زبانی گفته می‌شود که توسط یک اتومات محدود قابل شناسایی باشد. به عبارت بهتر تناظر یک به یک بین زبان‌های منظم و اتومات‌های محدود وجود دارد. تاکنون با زبان‌های منظم و هم چنین اتومات‌های بسیار ساده آشنا شدیم. ولی بسته به تعداد حالت‌های یک اتومات و هم چنین نوع توابع آن یک اتومات و در نتیجه زبان شناسایی شده توسط آن می

^{۱۲}Regular Language



شکل ۵: این اتومات تعینی رشته هایی را می پذیرد که حداقل شامل دو نماد 1 به صورت متوالی باشند.



شکل ۶: این اتومات تعینی رشته هایی را می پذیرد که شامل زیر رشته ی 001 باشند.

تواند بسیار پیچیده باشد. نکته جالب در مورد زبان ها آن است که می توان روی آنها اعمال مقدماتی ای انجام داد و همچنان منظم باقی بمانند.

قضیه: مجموعه زبان های منظم تحت اعمال اجتماع، چسباندن و ستاره یک مجموعه بسته است.

با دانش فعلی مان می توانیم قسمت اول این قضیه یعنی بسته بودن زبان های منظم تحت عمل اجتماع را اثبات کنیم. ولی برای اثبات بسته بودن این زبان ها تحت دو عمل دیگر بهتراست صبر کنیم تا اتومات های نامتعیین را بفهمیم. در آن موقع اثبات این دو قسمت بسیار آسان خواهد بود. روشی که ما برای اثبات قسمت اول این قضیه به کار می بریم خود به خود ما را به ابداع مفهوم جدیدی از اتومات خواهد شد.

اثبات بسته بودن زبان ها تحت عمل اجتماع: فرض کنید که ماشین $M_1 = (\Sigma, Q_1, q_1, \delta_1, F_1)$ زبان A و ماشین $M_2 = (\Sigma, Q_2, q_2, \delta_2, F_2)$ زبان B را شناسایی می کند. می خواهیم ماشینی بسازیم که زبان $A \cup B$ را شناسایی کند. می دانیم که هر رشته ی متعلق به $A \cup B$ یا متعلق به A است یا متعلق به B . اگر مطابق با فلسفه ای که برای فهم کارکرد ماشین ها و طراحی آنها به کار برده ایم عمل کنیم، راه بدیهی برای آنکه تشخیص دهیم رشته ای متعلق به $A \cup B$ هست یا نه آن است که هر رشته ای که به دستمان می رسد، یک نسخه اش را به ماشین M_1 بدهیم و یک نسخه اش را به ماشین M_2 . اگر این رشته متعلق به A باشد، ماشین M_1 آن را شناسایی خواهد کرد و اگر متعلق به B باشد، ماشین M_2 آن را شناسایی خواهد کرد. بنابراین مثل این است که برای شناسایی این زبان هر دو ماشین را توامان به کار می اندازیم و هر کدام که به یک حالت پذیرش رسید از روی آن می فهمیم که رشته ی مورد نظر متعلق به زبان $A \cup B$ بوده است و اگر هیچ کدام به حالت پذیرش نرسیدند نتیجه می گیریم که رشته مورد نظر متعلق به زبان $A \cup B$ نبوده است. با این توصیف ماشینی که زبان $A \cup B$ را شناسایی می کند عبارت است از:

$$M = (\Sigma, Q_1 \times Q_2, (q_1, q_2), \delta, F) \quad (8)$$

که در آن

$$\delta : \Sigma \times Q_1 \times Q_2 \longrightarrow Q_1 \times Q_2, \quad \delta(s, (q_1, q_2)) = (\delta_1(s, q_1), \delta_2(s, q_2)). \quad (9)$$

این رابطه بیان دقیق این امرشهودی است که هردو ماشین همزمان حروف یک رشته را پردازش می کنند. برای آنکه حالت های نهایی F را بفهمیم بازهم به ترجمه دقیق این بیان شهودی می پردازیم که ماشین جدید وقتی یک رشته را می پذیرد که یکی از ماشین ها آن را پذیرفته باشد. به عبارت دیگر اگر یکی از ماشین ها مثلاً ماشین M_1 به یک حالت پذیرش رسید مستقل از اینکه ماشین دیگری یعنی M_2 در چه حالتی است، آن رشته پذیرفته می شود و این به این معناست که آن رشته متعلق به A و در نتیجه متعلق به $A \cup B$ بوده است. بنابراین حالت های پذیرش نهایی عبارتند از:

$$F = (F_1 \times Q_2) \cup (Q_1 \times F_2). \quad (10)$$

به این ترتیب چون یک اتومات محدود و تعینی ساخته ایم که زبان $A \cup B$ را شناسایی می کند نتیجه می گیریم که مجموعه زبان های منظم تحت اجتماع بسته است.

۵ عدم تعین و اتومات های نامتعین و محدود

روشی که برای اثبات این قضیه به کار بردیم ما را به ایده جدیدی برای طراحی اتومات ها و ماشین ها رهنمون می سازد. این ایده متکی بر محاسبه همزمان است به این معنا که یک ماشین نامتعین می سازیم که نه تنها در ابتدای کار خود بلکه در هر مرحله ای از کار خود می تواند نمونه هایی از خود را به صورت مجازی تکثیر کند و یک رشته توسط همه این نمونه ها به صورت همزمان پردازش شود و هرگاه که یکی از نمونه ها این رشته را پذیرفت کل ماشین نامتعین آن رشته را بپذیرد. به طور شماتیک یک ماشین نامتعین مطابق شکل ۷ عمل می کند. به طور دقیق تر یک ماشین نامتعین به صورت زیر تعریف می شود.

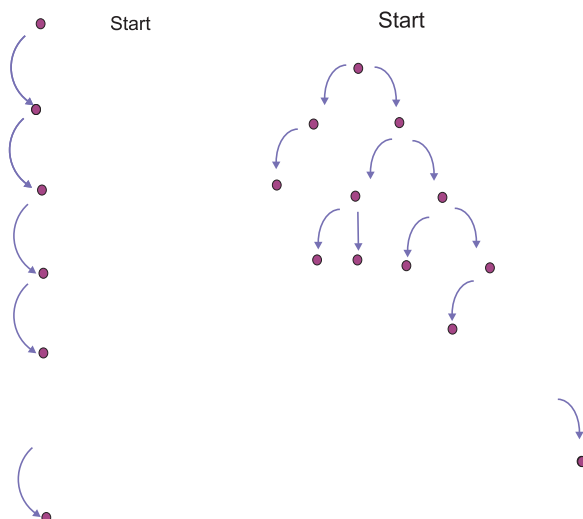
تعریف: یک اتومات محدود و نامتعین ¹³ دستگاهی است متشکل از پنج عنصر $NFA = \{Q, \Sigma_\epsilon, \delta, q_0, F\}$ که در آن :

الف : Q یک مجموعه محدود موسوم به مجموعه حالت هاست : $Q = \{q_0, q_1, \dots, q_n\}$.

ب : Σ یک الفباست و $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

ج : $\delta : Q \times \Sigma_\epsilon \longrightarrow \mathcal{P}(Q)$ یک نگاشت موسوم به نگاشت انتقال است که به هر حالت و به هر حرف از Σ_ϵ یک زیر مجموعه از حالت های Q را نسبت می دهد.

¹³ Non-deterministic Finite Automata (NFA)



شکل ۷: در یک اتومات نامعین، یک رشته مسیرهای متعددی را طی می کند و هر مسیری که به حالت قبولی برسد آن رشته پذیرفته می شود. اگر هیچ مسیری به یک حالت قبولی نرسد، آنگاه رشته مورد نظر پذیرفته نخواهد شد.

د: q_0 یک حالت ویژه از Q موسوم به حالت آغازین است و

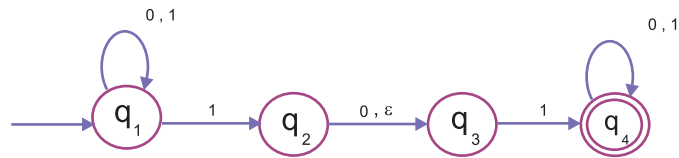
ه: $F \subset Q$ یک زیرمجموعه از حالات است که مجموعه پذیرش نامیده می شود.

در این تعریف منظور از $P(Q)$ مجموعه تمام زیرمجموعه های Q است.

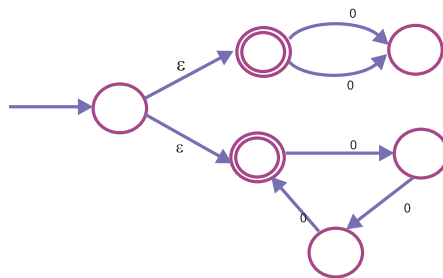
دقت کنید که چند تفاوت عمده بین تابع انتقال یک ماشین DFA و یک ماشین NFA وجود دارد. اولاً تابع δ حالت ϵ را نیز به عنوان ورودی می پذیرد به این معنا که با دریافت رشته تهی این ماشین می تواند تغییر حالت دهد، به عبارت بهتر خود به خود می تواند تغییر حالت دهد. ثانیاً خروجی تابع δ می تواند یک مجموعه تک عضوی، چند عضوی یا حتی مجموعه تهی باشد. هرگاه که این خروجی یک مجموعه تک عضوی باشد، عمل ماشین در این مرحله کاملاً تعیینی است و هرگاه خروجی ماشین یک مجموعه چند عضوی باشد، عمل ماشین در این مرحله نامتعیین است به این معنی که ماشین در این مرحله همه مسیرهای ممکن را همزمان طی می کند و بالاخره هرگاه که خروجی تابع مجموعه تهی باشد به این معناست که عمل ماشین در این شاخه متوقف می شود و به اصطلاح ماشین در این شاخه می میرد. بنابراین در دیگرام حالت یک ماشین نامتعیین از هر دایره که نشان دهنده ای یک حالت است می تواند هیچ خط، یک خط، یا چندین خط خارج شده باشند. این کیفیت در شکل های مربوط به مثال ها بخوبی دیده می شود. نحوه محاسبه ای که در بالا شرح دادیم هرگاه به صورت دقیق درآید به شکل زیر خواهد بود:

محاسبه توسط ماشین NFA : یک رشته $s = s_1 s_2 \dots s_n$ توسط یک ماشین NFA پذیرفته می شود هرگاه زنجیره ای از حالت های ماشین مثل $r_0, r_1, r_2, \dots, r_n \in Q$ وجود داشته باشند به قسمی که :

$$r_0 = q_0 \quad \text{و} \quad r_i = \delta(r_{i-1}, s_i) \quad \text{برای} \quad i = 1, 2, \dots, n$$



شکل ۸: این اتومات نامعین رشته‌هایی را می‌پذیرد که یا شامل زیر رشته‌های ۱۱ باشند یا شامل رشته‌های ۱۰۱.



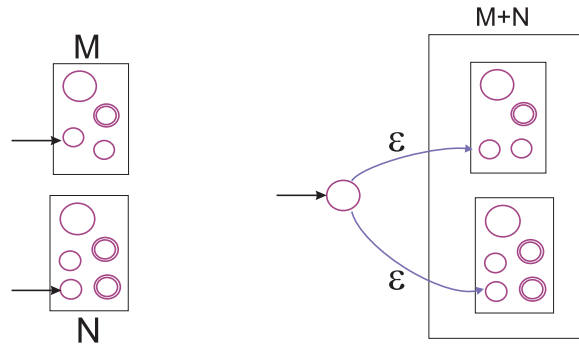
شکل ۹: این اتومات نامعین رشته‌هایی را می‌پذیرد که فقط شامل مضربهای ۲ یا ۳ از نماد ۰ باشند.

$$r_{i+1} \in \delta(r_i, w_i) - ۲$$

$$r_n \in F - ۳$$

شکل‌های ؟؟ و ۹ نمونه‌هایی از ماشین‌های نامتعین و محدود هستند. اگرچه ظاهر امر آن است که ماشین‌های نامتعین کارایی بیشتری نسبت به ماشین‌های متعین دارند، ولی می‌توان ثابت کرد که این دو نوع ماشین در واقع یک نوع زبان را تشخیص می‌دهند به این معنا که هر ماشین نامتعین را می‌توان با یک ماشین متعین شبیه‌سازی کرد. البته ممکن است که برای شبیه‌سازی یک ماشین نامتعین که تعداد خیلی کمی حالت دارد مجبور شویم که ماشین متعینی با تعداد بسیار زیادی حالت طراحی کنیم. این امر ضمن آنکه نشان دهنده سادگی و کارایی ماشین‌های NFA است نشان می‌دهد که ماشین‌های NFA و DFA با هم معادل‌اند.

قضیه: ماشین‌های NFA با ماشین‌های DFA معادلند به این معنا که هر ماشین NFA قابل شبیه‌سازی توسط یک ماشین DFA است.



شکل ۱: نحوه ساختن ماشینی که زبان $A \cup B$ را شناسایی کند.

اثبات: فرض کنید که $M = (\Sigma, Q, q_1, \delta, F)$ یک NFA باشد. در این صورت ماشین تعینی M' را به شکل زیر می سازیم:

$$M' := (\Sigma \cup \epsilon, P(Q), \{q_1\}, \delta', F), \quad (11)$$

که در آن $P(Q)$ مجموعه‌ی همه زیرمجموعه‌های Q است. کافی است که بگوییم این ماشین تعینی چه تابع انتقالی دارد. بازهم در این جا دریافت شهودی خود را از نحوه محاسبه‌ی ماشین نامتعیین به یک عبارت ریاضی دقیق ترجمه می کنیم. می دانیم که ماشین نامتعیین در هر لحظه به چند نمونه از خود تکثیر شده است و همه شاخه ها با هم در حال کارند. بنابراین هرگاه بخواهیم معادل این ماشین نامتعیین را پیدا کنیم باید ماشینی بسازیم که حالت های آن زیرمجموعه های Q باشند و تابع انتقال آن عمل موازی ماشین را نشان دهد. بنابراین می نویسیم حالت

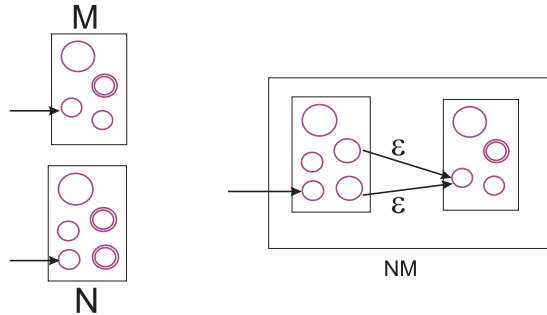
$$\delta'(s, \{q_i, q_j, q_k \dots\}) = \delta(s, q_i) \cup \delta(s, q_j) \cup \delta(s, q_k) \dots \quad (12)$$

به این ترتیب یک ماشین نامتعیین را همواره می توان به عنوان یک ماشین متعین ولی با تعداد حالت های بیشتر در نظر گرفت.

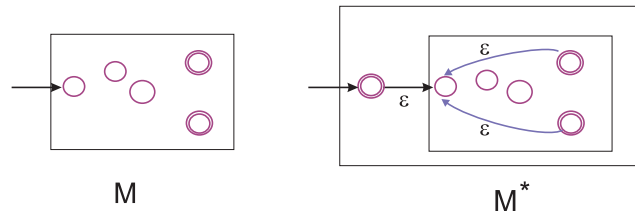
با استفاده از این قضیه می توانیم بسته بودن زبان های منظم تحت اعمال منظم را به آسانی نشان دهیم. برای این کار کافی است که نشان دهیم هرگاه دو زبان A و B منظم باشند، آنگاه ماشین های نامتعیینی وجود دارند که بتوانند زبان های $A \cup B$ ، $A \circ B$ و A^* را شناسایی کنند. از آنجا که این ماشین های نامتعیین را همواره می توان به صورت ماشین های متعین در آورد، نتیجه می گیریم که مجموعه زبان های منظم تحت اعمال منظم یک مجموعه بسته است.

قضیه: زبان های منظم تحت اعمال منظم بسته هستند.

اثبات: فرض کنید که A و B زبان های منظمی باشند. در این صورت حتماً ماشین هایی مثل M و N وجود دارند که این زبان ها را شناسایی کنند. شکل های ۱۱، ۱۱ و ۲۱ نشان می دهند که چگونه می توان ماشین هایی ساخت که به ترتیب زبان های $A \cup B$ ، $A \circ B$ و A^* را شناسایی کنند. این امراثبات قضیه را کامل می کند.



شکل ۱۱: نحوه ساختن ماشینی که زبان $B \circ A$ را شناسایی کند.



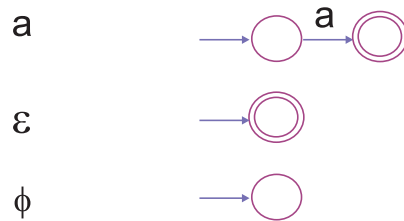
شکل ۲۱: نحوه ساختن ماشینی که زبان A^* را شناسایی کند.

۶ عبارت های منظم

تا کنون زبان های منظم و اعمال آنها را معرفی کرده ایم. در این بخش می خواهیم روشی را یاد بگیریم که با استفاده از آن یک زبان منظم را به صورت کوتاه معرفی کنیم. یک زبان A را با یک عبارت کوتاه R نشان می دهیم. R را یک عبارت منظم^{۱۴} می نامند که نشان دهنده آن زبان است. به عنوان مثال به ازای الفبای $\Sigma = \{a, b\}$ ، عبارت منظم Σ^* نشان دهنده تمام رشته های تشکیل شده از آن الفباست و عبارت منظم a^* نشان دهنده تمام رشته هایی است که از حرف a تشکیل شده است یعنی $a^* = \{a^n \mid n \geq 0\}$ و عبارت Σ^*b نشان دهنده تمام رشته هایی است که با حرف b ختم می شوند. در این نحوه نوشتن از قواعد مربوط به ترکیب زبان های منظم استفاده می کنیم. روابط زیر مثال های بیشتری را نشان می دهند. عبارت انگلیسی که روبروی هر عبارت منظم نوشته شده است زبان منظم متناظر با آن عبارت را نشان می دهد.

$\Sigma^*\Sigma^*$	=	all strings of even length
$a\Sigma^*b$	=	all strings begining with a and ending with b
$(0U1)0^*$	=	all strings which are purely made of 0 or have only one single 1 at their left

^{۱۴}Regular Expression



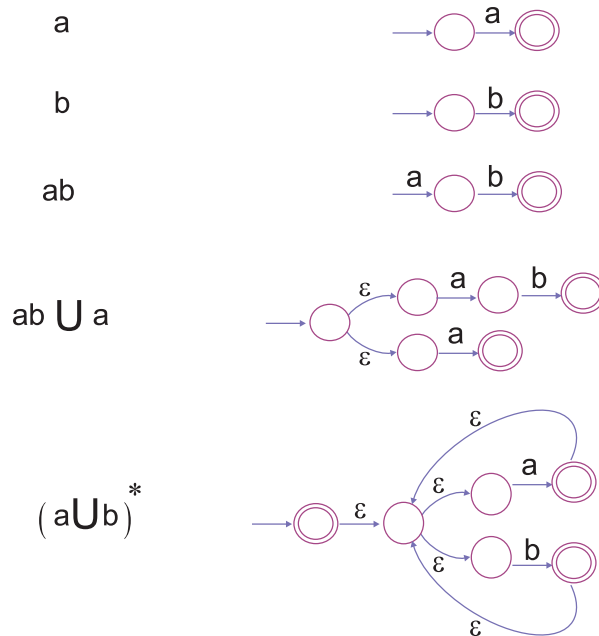
شکل ۳۱: عناصر اصلی ای برای تناظر عبارت های منظم و اتومات های محدود.

$$\begin{aligned}
 \Sigma^*1\Sigma^* &= \text{all strings which have at least a } 1 \\
 a^*b^* &= \text{all strings in which all } a\text{'s precede } b\text{'s,} \\
 a^*b^* \cup b^*a^* &= \text{all strings in which either all } a\text{'s precede all } b\text{'s or vice versa,} \\
 \Sigma\Sigma^*\Sigma^* &= \text{all strings whose length is odd.}
 \end{aligned}
 \tag{13}$$

خواننده می تواند بشمار مثال به فهرست فوق بیفزاید. آیا هر زبان منظم را می توان با یک عبارت منظم بیان کرد؟ آیا بین زبان های منظم و عبارت های منظم رابطه یک به یک وجود دارد؟ پاسخ این سوال ها مثبت است. در واقع نشان می دهیم که بین عبارت های منظم و ماشین های DFA تناظر یک به یک برقرار است و از آنجا که زبان منظم به زبانی گفته می شود که توسط یک ماشین DFA شناسایی شود، نتیجه می گیریم که بین زبان های منظم و عبارت های منظم تناظر یک به یک وجود دارد. برای اثبات این تناظر به ترتیب زیر عمل می کنیم. شکل ۳۱ نشان می دهد که چگونه به ساده ترین عبارت های منظم می توان ماشین های DFA نسبت داد. با استفاده از این شکل و قوانین ترکیب زبان ها و ماشین های DFA یعنی با اعمال منظم $\cup$, $\circ$, و $*$ می توانیم ماشین های DFA متناظر با هر عبارت منظمی را بسازیم. شکل ۴۱ نمونه هایی از این نحوه ساخت را نشان می دهد.

بنابراین نشان داده ایم که به ازای هر عبارت منظم یک ماشین DFA وجود دارد. بالعکس می توانیم نشان دهیم که به ازای هر ماشین DFA می توانیم یک عبارت منظم بسازیم. تفصیل این نحوه ساخت در مرجع شماره یک آمده است. در این جا فقط به ذکر ایده کلی آن می پردازیم. کلید اصلی ماشین تعمیم یافته ی نامنعین یا $GNFA$ ¹⁵ است. یک ماشین $GNFA$ درست مثل یک DFA عمل می کند با این تفاوت اصلی که در دیاگرام حالت آن بجای حروف الفبا عبارت های منظم نوشته می شود. شکل ۵۱ یک ماشین $GNFA$ را نشان می دهد. معنای این شکل آن است که هر عبارت از نوع aa یا bb ماشین را از حالت r_0 به حالت r_1 می برد و یا این که هر عبارت از نوع aa^*bb^* مثل $ab, aabb, aaaaabbbb, abbbb$ و نظایر آنها ماشین را از حالت r_{n-1} به حالت r_1 می برد. می توان با حذف کردن یک به یک حالت ها از یک ماشین $GNFA$ به یک ماشین $GNFA$ رسید که تنها یک حالت شروع و یک حالت پذیرش دارد و این دو حالت با یک عبارت منظم به هم مربوط اند. این عبارت منظم همان

¹⁵Generalized Non-deterministic Finite Automata



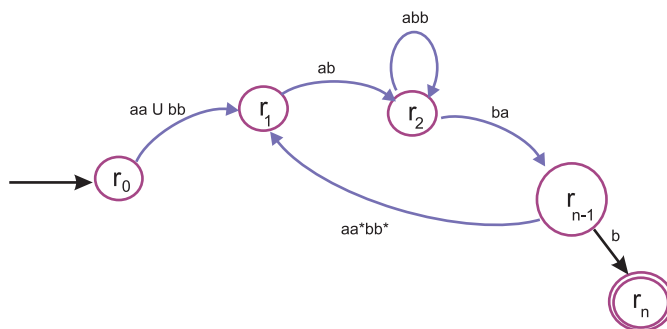
شکل ۴۱: مثال هایی از عبارت های منظم و اتومات های متناظر با آنها.

عبارتی است که ماشین DFA اولیه را بیان می کند. این فرایند در شکل ۶۱ نشان داده شده است. به این ترتیب نشان داده ایم که بین زبان های منظم، عبارت های منظم و ماشین های DFA تناظر یک به یک وجود دارد.

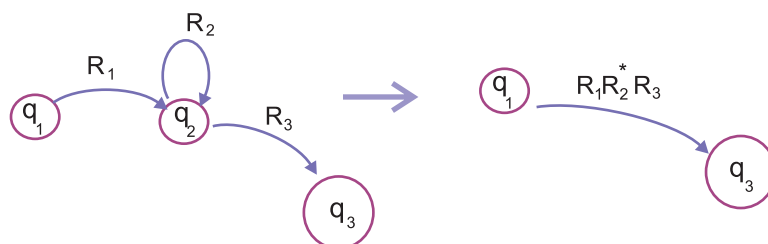
۷ محدودیت های اتومات های محدود و زبان های منظم

رابطه ای که بین عبارت های منظم و زبان های منظم و هم چنین ماشین های تعیینی محدود وجود دارد نشان می دهد که این ماشین ها می توانند گستره وسیعی از زبان ها را شناسایی کنند. باین وجود گستره وسیعی از زبان ها وجود دارند که توسط این ماشین ها قابل شناسایی نیستند. به زبان دیگر گستره وسیعی از مسایل وجود دارند که توسط این ماشین ها قابل حل نیستند. بازهم به زبان دیگر اگر فقط به چنین مدل محاسبه ای دسترسی داشتیم، دسته وسیعی از مسایل وجود می داشتند که نمی توانستیم برای حل آنها الگوریتم بنویسیم. محدودیت عمده ای که ماشین های تعیینی دارند آن است که حافظه ندارند یا به عبارت دیگر حافظه محدودی دارند. این حافظه محدود همانی است که در تعداد حالت های آنها وجود دارد. بنابراین هرگاه که زبانی داشته باشیم که برای شناسایی آن احتیاج به حافظه نامحدودی داشته باشیم یک ماشین تعیینی محدود قادر به شناسایی آن زبان نخواهد بود. به عنوان مثال زبان زیر را در نظر بگیرید:

$$A = \{0^n 1^n \mid n > 0\}. \quad (14)$$



شکل ۵۱: نمونه‌ای از یک اتومات تعمیم یافته نامتعیین $GNFA$.



شکل ۶۱: روشی که تعداد حالت های یک $GNFA$ را تقلیل داده و عبارت منظم متناظر با آن را پیدا می کنیم.

برای شناسایی رشته های مربوط به این زبان، یک ماشین باید بتواند نخست تعداد 0 را بشمارد و درجایی ذخیره کند و سپس تعداد 1 را بشمارد و آنها را با تعداد صفرها مقایسه کند. اگر این دو عدد مساوی بودند، ماشین رشته مربوطه را می پذیرد و درغیراین صورت آن را رد می کند. یک ماشین تعینی چنین امکانی ندارد. هم چنین زبان زیرتوسط یک ماشین تعینی محدود قابل شناسایی نیست:

$$B = \{w \in \Sigma^* \mid w \text{ contains equal number of } 0's \text{ } 1's\}. \quad (15)$$

چنین زبان هایی را زبان های نامنظم¹⁶ می گویند. یک نمونه دیگر از مسئله ای که یک ماشین تعینی محدود قادر به حل کردن آن نیست مسئله ی تست کردن پرانتزهاست¹⁷. فرض کنید که عبارتی مثل $((()()))$ داریم. دراین عبارت ساختار پرانتزها صحیح است ولی عبارتی مثل $((()_1))$ ساختارپرانتزی صحیحی ندارد. دقت کنید که کافی نیست که ماشین تنها تعداد پرانتزهای بسته و باز را بشمارد، زیرا که ترتیب باز و بسته شدن پرانتزها نیز مهم است. بنابراین مسئله تست پرانتزها با شناسایی زبان B متفاوت است.

برای آنکه تشخیص دهیم که آیا یک زبان منظم است یا نا منظم یک لم بسیار مفید و قدرتمند وجود دارد که به لم پُلمپ

¹⁶Non-Regular
¹⁷Paranthesis Chekcer Problem

کردن¹⁸ مشهور است.

لمپمپ کردن: اگر A یک زبان منظم باشد، آن گاه یک عدد مثبت p موسوم به طول پمپ کردن¹⁹ وجود دارد به قسمی که هر رشته‌ی s متعلق به A را که طولش بزرگ تر یا مساوی با p است، می توان به سه قسمت xyz چنان تقسیم کرد، که سه شرط زیر تماماً برآورده شوند:

الف: به ازای هر $xy^iz \in A, i \geq 0$

ب: $|y| > 0$

ج: $|xy| < p$

دقت کنید که در صورت قضیه قید چنان مهم است. یعنی هر نوع تقسیم بندی الزاماً به برآورده شدن شرط های سه گانه فوق نمی انجامد. بنابراین در صورت درست بودن این قضیه که صحت آن را کمی بعد اثبات می کنیم، راهی که برای اثبات نامنظم بودن یک زبان می توانیم طی کنیم چنین است.

روش استفاده از لمپمپ کردن: هرگاه زبان A چنان باشد که به ازای هر طول پمپ کردن، و هر نوع تقسیم کردنی از یک رشته به صورت $s = xyz$ ، همواره بتوان نشان داد که رشته‌ای مثل xy^iz (برای یک i) وجود دارد که متعلق به آن زبان نیست. قبل از استفاده از این لم به اثبات آن توجه می کنیم که چنان که خواهیم دید ساده است.

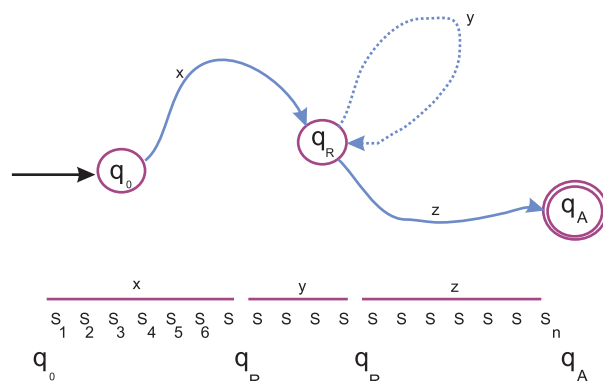
اثبات لمپمپ کردن: هرگاه A یک زبان منظم باشد، توسط یک ماشین تعینی محدود شناسایی می شود. فرض کنید که تعداد حالت های این ماشین برابر با p باشد. طول پمپ کردن را همین p می گیریم. در این صورت هرگاه رشته‌ای مثل $s = s_1s_2s_3 \dots s_n$ ، $n > p$ داشته باشیم که توسط ماشین پذیرفته شود، این رشته می بایست ماشین را مطابق با طرح زیر از حالت q_0 به حالت q_{accept} ببرد:

$$\begin{array}{ccccccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & \dots & \dots & s_n \\ q_0 & q_{i_1} & q_{i_2} & q_{i_3} & q_{i_5} & \dots & \dots & q_{accept} \end{array} \quad (16)$$

می دانیم که تعداد حالت های q_i ، p تا است که تعداد آنها از n کمتر است. بنابراین با استدلال لانه کبوتر می فهمیم که حداقل یکی از q_i ها تکرار شده است. این حالت را با q_R نشان می دهیم. اولین و دومین ظاهر شدن حالت q_R را در رشته‌ی حالت ها در نظر می گیریم و مطابق با شکل ۷۱، رشته‌ی s را به قسمت های xyz تقسیم می کنیم. اولاً واضح است که $|y| > 0$ ، ثانیاً $|xy| \leq p$ ، ثالثاً با توجه به شکل ۷۱ واضح است که هر رشته‌ای مثل xy^iz نیز متعلق به زبان A است.

دقت کنید که چون ابتدا و انتهای زیر رشته‌ی y محل اولین و دومین ظاهر شدن حالت تکراری q_R هستند و q_R اولین حالتی است که در رشته‌ی q_i ها تکرار شده است پس حتماً طول xy از p که تعداد کل حالت هاست کمتر یا با آن مساوی است، یعنی $|xy| \leq p$.

Pumping Lemma¹⁸
Pumping Length¹⁹



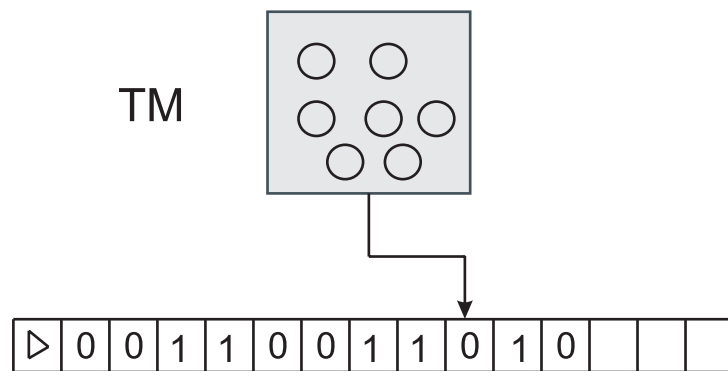
شکل ۷۱: لم پمپینگ. اگر این ماشین رشته xyz را بپذیرد، حتماً رشته های xy^iz را هم می پذیرد. ضمناً $|y| \geq 0$ و $|xy| \leq p$.

با استفاده از لم پمپ کردن می توان براحتی فهمید که یک زبان نامنظم است. به عنوان مثال زبان A در 14 را در نظر می گیریم. فرض کنید که p طول پمپینگ باشد. باید نشان دهیم که رشته ای وجود دارد با طول بیشتر از p که هر طور آن را به سه قسمت xyz تقسیم کنیم که در شرایط $|y| > 0$ و $|xy| \leq p$ صدق کند، بازهم بعضی از رشته های xy^iz خارج از زبان A خواهند افتاد. رشته ای مثل $0^p 1^p$ را در نظر بگیرید. هر طور تقسیم می که بخواهد شرط $|xy| \leq p$ را ارضا کند می بایست چنان باشد که x و y تماماً از صفرها تشکیل شده باشد. بنابراین واضح است که پمپ کردن این رشته یعنی تکرار y باعث افزایش 0 ها خواهد شد بدون اینکه تعداد 1 ها افزایش یابد. در نتیجه رشته های xy^iz خارج از زبان خواهند افتاد. بنابراین زبان A یک زبان نامنظم است.

همین استدلال نشان می دهد که زبان B در 15 نیز نامنظم است.

۸ ماشین تورینگ

محدودیت اساسی اتومات های تعیینی یا نامتعیین داشتن یک حافظه نامحدود است. به همین دلیل این ماشین ها می توانند تنها دسته خاصی از زبان ها یعنی زبان های منظم را شناسایی کنند که به معنای آن است که این ماشین ها تنها می توانند مسائل خاصی را حل کنند. همانطور که در مقدمه گفتیم انگیزش اولیه ریاضیدانان برای یافتن پاسخی برای مسئله هیلبرت آن بود که تعریف مشخصی از آگوریتم ارائه دهند. اگر مفهوم آگوریتم را منحصر به اتومات های محدود کنیم، تعریف محدودی از آن ارائه داده ایم. چگونه می توان این محدودیت را برداشت؟ چگونه می توان مدلی از یک ماشین ساخت که به یک حافظه نامحدود دسترسی داشته باشد؟ برای آنکه مفهوم آگوریتم را به شکل کلی اش بفهمیم یا آن را به طور دقیق صورت بندی کنیم بهتر است که خود را در جای یک ریاضیدان یا منطق دان یا یک انسان متفکر قرار دهیم که سعی می کند مسئله مشخصی را حل کند. بیایید به صورت بسیط و جدای از هر نوع شاخ و برگ اضافه که نقش اساسی ندارند، کارکرد این شخص را بررسی کنیم. از خود می پرسیم که چنین شخصی دقیقاً چه می کند؟ وی داده هایی (مثلاً اعداد) از روی یک کاغذ می خواند. مطابق با این داده های دریافتی حالت های ذهنی اش تغییر می کند. سپس مطابق با این حالت های ذهنی و داده هایی که دریافت کرده است داده های جدیدی را روی کاغذ می نویسد یا داده های قدیمی را پاک می کند. این که چگونه داده های خوانده شده و حالت های ذهنی منجر به داده های جدید می شوند، همان آگوریتم است. از آنجا که به نوع فونت و رسم الخط و غیره علاقه مند نیستیم، می توانیم فرض کنیم که تمامی داده ها به صورت علائمی از یک حروف الفبا در مربع های کوچکی نوشته می شوند و این مربع ها هم می توانند بجای اینکه در یک کاغذ دوبعدی نوشته شوند در یک نوار یک بعدی مرتب شوند. به این



شکل ۸۱: ماشین تورینگ.

ترتیب ماشینی درست می شود که آن را ماشین تورینگ می خوانیم ۸۱. ماشین تورینگ یک قلم^{۲۰} دارد که در هر لحظه روی یکی از مربع هاست. این قلم می تواند یک خانه به طرف راست یا چپ حرکت کند. می تواند یک حرف از الفبا را از روی نوار بخواند یا یک حرف را روی نوار بنویسد. از جمله می تواند یک حرف را از روی یک خانه از نوار پاک کند. آخرین خانه در سمت چپ نوار که خانه ی ابتدایی نوار است همیشه با علامت $\triangleright$ مشخص می شود. الفبای ماشین تورینگ با Σ ، خانه خالی با حرف b و اولین خانه سمت چپ با $\triangleright$ مشخص می شود. این خانه قابل نوشتن یا پاک کردن نیست. مجموعه ی حالت ها نیز با $Q = \{q_1, q_2, \dots, q_N\}$ مشخص می شوند. دقت کنید که ماشین تورینگ همچنان ماشینی است که تعداد حالت های آن محدود است. در بین این حالت ها دو حالت متمایزند. یکی حالت q_s که حالت آغازین است و دیگری حالت q_{Halt} ^{۲۱} هرگاه که ماشین در حالت q_{Halt} قرار بگیرد، اجرای برنامه متوقف می شود. توابع انتقال ماشین تورینگ به صورت زیر عمل می کنند:

$$\delta : Q \times \{b\} \cup \{\triangleright\} \cup \Sigma \longrightarrow Q \times \{b\} \cup \Sigma \times \{L, R\}. \quad (17)$$

علامت های L و R نیز جهت حرکت قلم را در ماشین تورینگ نشان می دهند. گاهی اوقات بجای حروف R و L از اعداد $+1$ و -1 استفاده می کنیم. به این ترتیب تحت اثر این تابع انتقال خواهیم داشت:

$$(q, s) \longrightarrow (q', s', \alpha) \quad (18)$$

که در آن $\alpha \in \{L, R\}$ جهت حرکت قلم را نشان می دهد. معمولاً اثر تابع انتقال به شکل فشرده ی زیر نوشته می شود:

$$(q, s, q', s', \alpha) \quad (19)$$

مجموعه ی پنج تایی هایی که به صورت فوق نوشته می شوند برنامه ماشین را تعیین می کنند. طرز کار ماشین تورینگ به این صورت است: در ابتدا یک رشته روی نوار نوشته شده است. اولین خانه نوار با حرف $\triangleright$ مشخص شده است. بعد از این علامت رشته قرار دارد و بعد از رشته علامت های b یعنی جای خالی تا انتهای نوار قرار گرفته اند. در ابتدا قلم ماشین روی حرف $\triangleright$ قرار دارد. حالت ماشین نیز در ابتدا q_s است. ماشین تورینگ شروع به خواندن یک به یک حروف می کند و مطابق با آن حالت جدیدی اختیار می کند، حرف جدیدی از جمله حرف b را، روی خانه خوانده شده می نویسد و قلم اش را

^{۲۰}Head از کلمه ی $Halt$ به معنای توقف آمده است.

به طرف راست حرکت می دهد تا برای خواندن خانه دوم آماده شود. حرکت قلم تنها وقتی که روی حرف $\triangleright$ در ابتدای نوار قرار گرفته است به طرف راست است ولی روی حروف دیگر در وسط نوار حرکت قلم می تواند به طرف چپ یا راست باشد که این دو جهت را با -1 و 1 نشان می دهیم. با خواندن حروف و تغییر آنها و هم چنین تغییر حالت ماشین، وقتی که ماشین به حالت q_{Halt} می رسد، محاسبه متوقف می شود و رشته ای که روی نوار نوشته شده است جواب ماشین تلقی خواهد شد. در اینجا ماشین تورینگ را توصیف کرده ایم که خروجی اش یک رشته است، یعنی ماشین تورینگ چیزی را محاسبه می کند. ماشین تورینگ که درباره یک رشته تنها تصمیم گیری می کند ساختمان اندکی متفاوت دارد. این چنین ماشینی بجای یک حالت q_{Halt} دو حالت q_{Accept} و q_{Reject} دارد و در پایان محاسبه تنها این مهم است که ماشین در کدام یک از این دو حالت قرار گرفته است نه این که روی نوار آن چه چیزی نوشته شده است. هرگاه در پایان محاسبه ماشین در حالت q_{Accept} قرار گرفته باشد، آن رشته پذیرفته شده و هرگاه ماشین در حالت q_{Reject} قرار بگیرد، آن رشته پذیرفته نشده است. وقتی که ماشین در هر کدام از این دو حالت قرار بگیرد عمل آن متوقف می شود ولی اگر ماشین در هیچ کدام از این دو حالت قرار نگیرد به معنای آن است که ماشین هیچ تصمیمی در مورد رشته مورد نظر نگرفته است. در این حالت ماشین نمی تواند متوقف شود.

۱.۸ انواع ماشین تورینگ

آنچه که در بالا توصیف کردیم، ساختمان یک ماشین تورینگ تعیینی و تک نواری بود. می توان ماشین های تورینگ پیچیده تری ساخت. به عنوان مثال ماشینی که قلم آن بجای دو انتخاب حرکت به راست و چپ، سه انتخاب حرکت به راست و چپ و توقف را دارد. دستورهای چنین ماشینی به صورت زیر خواهند بود:

$$(q, s, q', s', \alpha \in -1, 0, 1), \quad (20)$$

که در آن علامت 0 به معنای توقف قلم در خانه ی خوانده شده است. یامی توان ماشینی تصور کرد که چند تا نوار و در نتیجه چند تا قلم دارد. به عنوان مثال دستورهای یک ماشین دو نواری به صورت زیر خواهد بود:

$$(q, s_1, s_2; q', s'_1, s'_2, \alpha_1, \alpha_2), \quad (21)$$

که در آن همه علامت ها تعبیر روشنی دارند.

هم چنین می توان ماشین تورینگ نامتعیین را تعریف کرد که در آن در هر لحظه ماشین می تواند نسخه هایی از خودش را تولید کند و همه نسخه ها را به طور همزمان دنبال کند. هرگاه که یکی از نسخه ها یا شاخه ها به یک حالت q_{Accept} برسد به این معناست که ماشین رشته مورد نظر را پذیرفته است.

نکته مهم و اساسی در مورد همه این ماشین ها آن است که همه ی آنها توسط ماشین تورینگ ساده ای که در زیر بخش قبلی توصیف کردیم قابل شبیه سازی هستند. این امر به این معناست که این ماشین ها قدرت محاسباتی بیشتری ندارند و هر زبانی که توسط این ماشین های تعمیم یافته قابل شناسایی باشد، توسط آن ماشین تورینگ ساده نیز قابل شناسایی است اگر چه ممکن است توصیف برنامه ی محاسبه یا الگوریتم با این ماشین های توسعه یافته ساده تر باشد.

۲.۸ مثال هایی از ماشین تورینگ

در این بخش مثال های متفاوتی ارائه می دهیم تا با طرز کار ماشین تورینگ آشنا شویم. در هر مثال خواننده می تواند طرز کار ماشین تورینگ را با دنبال کردن حالت ها و برنامه بفهمد و تصدیق کند که آیا ماشین تورینگ کار خواسته شده را انجام می دهد یا خیر.

مثال: در این مثال یک ماشین تورینگ (یعنی یک آلگوریتم) طرح می کنیم که هر رشته ای روی نوار ماشین را پاک می کند. برنامه ی این ماشین به شکل زیر است:

$$\begin{aligned} &(q_s, \triangleright, q_1, \triangleright, +1) \\ &(q_1, 0, q_1, b, +1) \\ &(q_1, 1, q_1, b, +1) \\ &(q_1, b, q_{Halt}, b, +1). \end{aligned} \quad (22)$$

مثال: در این مثال یک ماشین تورینگ الفبای 0 و 1 را در هر رشته ای معکوس می کند. یعنی 0 را به 1 و 1 را به 0 تبدیل می کند.

$$\begin{aligned} &(q_s, \triangleright, q_1, \triangleright, +1) \\ &(q_1, 0, q_1, 1, +1) \\ &(q_1, 1, q_1, 0, +1) \\ &(q_1, b, q_{Halt}, b, +1). \end{aligned} \quad (23)$$

مثال: در این مثال یک ماشین تورینگ تابع ثابت $f(x) = 1$ را محاسبه می کند.

$$\begin{aligned} &(q_s, \triangleright, q_1, \triangleright, +1) \\ &(q_1, 0, q_2, 1, +1) \\ &(q_1, 1, q_2, 1, +1) \\ &(q_2, 0, q_2, b, +1) \\ &(q_2, 1, q_2, b, +1) \\ &(q_2, b, q_{Halt}, b, +1). \end{aligned} \quad (24)$$

در مثال هایی که تا کنون شرح داده ایم تمام توابع انتقال ماشین را به تفصیل نوشته ایم. این نحوه ی توصیف همواره لازم نیست و می توان شرح خلاصه تری از طرز کار ماشین تورینگ نوشت. همواره می توان در صورت لزوم این برنامه خلاصه شده

را که به زبان روزمره نزدیک ترند و اصطلاحاً یک زبان سطح بالا²² به همان زبان بنیادی برحسب توابع انتقال ماشین تورینگ یعنی برحسب زبان سطح پایین²³ نوشت. به عنوان مثال یک خط از برنامه مثل «قلم ماشین روی نوار به سمت راست حرکت می کند تا به پایان رشته یعنی به حرف b برسد و در این نقطه حالت ماشین از q_s به q_1 تغییر می کند» را در نظر می گیریم. این خط از برنامه را می توان به صورت زیر برحسب توابع انتقال نوشت.

$$\begin{aligned} & (q_s, \triangleright, q_s, \triangleright, +1) \\ & (q_s, 0, q_s, 0, +1) \\ & (q_s, 1, q_s, 1, +1) \\ & (q_s, b, q_1, b, 0). \end{aligned} \quad (25)$$

به این ترتیب اغلب اوقات برنامه ماشین تورینگ را می توانیم به زبان ساده تری بنویسیم. مثال های زیر این موضوع را روشن می کنند.

مثال: آلوگوریتمی می خواهیم که بتواند یک رشته را وارون کند به این معنا که رشته ی $s_1 s_2 \dots s_n$ را به رشته ی $s_n s_{n-1} \dots s_2 s_1$ تبدیل کند. این کار را می توان با یک ماشین تورینگ دو نوازی انجام داد. رشته ی $s_1 s_2 \dots s_n$ روی نوار اول نوشته شده است و نوار دوم خالی است. در ابتدا هر دو قلم ها در ابتدای نوارها و روی حروف $\triangleright$ هستند. قلم نوار اول روی نوار به طرف راست حرکت می کند تا به پایان رشته برسد. پایان رشته را از روی حرف b می فهمد. در این لحظه به سمت چپ حرکت می کند و هر حرفی را که قلم اول از یک خانه می خواند قلم دوم روی یک خانه می نویسد و به سمت راست حرکت می کند. وقتی که قلم اول به ابتدای نوار اول رسید ماشین به حالت q_{Halt} می رود و متوقف می شود. آنچه که روی نوار دوم نوشته شده است وارون رشته ای است که روی نوار اول نوشته شده بود. نوشتن این برنامه را برحسب توابع انتقال به عهده خواننده می گذاریم.

مثال: مسئله تست کردن پرانتزها را می توانیم با یک ماشین تورینگ براحتی حل کنیم. برنامه چنین ماشینی به شرح زیر است: الف — قلم به سمت راست می رود تا به اولین پرانتز (برخورد کند. این پرانتز را پاک می کند.

ب — قلم به سمت چپ برمی گردد تا به اولین پرانتز (برخورد کند. این پرانتز را نیز پاک می کند.

ج — ماشین مراحل الف و ب را تکرار می کند تا به انتهای رشته یعنی به حرف b برسد.

د — قلم به سمت چپ حرکت می کند و یک بار نوار را جاروب می کند. اگر به یک پرانتز (برخورد کرد، حالت ماشین به q_{Reject} تغییر کرده و رشته رد می شود، در غیر این صورت حالت ماشین به q_{Accept} تغییر می کند و رشته پذیرفته می شود.

High Level Language²²
Low level language²³

مثال: زبان زیر را در نظر بگیرید:

$$Add := \{x^i y^j z^k \mid k = i + j\}. \quad (26)$$

می خواهیم یک ماشین تورینگ طراحی کنیم که این زبان را تشخیص دهد. برنامه‌ی این ماشین به صورت زیر است:
الف — قلم به سمت راست می رود و به ازای هر حرف y یک حرف z را پاک می کند.

ب — ماشین مرحله الف را تکرار می کند تا اینکه تمام حروف y تمام شود.

ج — سپس ماشین به ازای هر حرف x یک حرف z را پاک می کند تا اینکه تمام حرف های x تمام شود.

د — قلم یک بار نوار را جاروب می کند. اگر هیچ حرف z یا x باقی نمانده باشد، رشته قبول می شود و اگر هر کدام از این حرف ها باقی مانده باشند رشته رد می شود.

مثال: زبان زیر را در نظر بگیرید:

$$Multiply := \{x^i y^j z^k \mid k = ij\}. \quad (27)$$

می خواهیم یک ماشین تورینگ طراحی کنیم که این زبان را تشخیص دهد. برنامه‌ی این ماشین به صورت زیر است:
الف — قلم به سمت راست می رود و به ازای هر حرف x تمام حروف y را پاک می کند و به ازای هر حرف y نیز یک حرف z را پاک می کند.

ب — قلم به سمت چپ حرکت می کند و تمام حروفی را که در قسمت y ها پاک کرده بود سر جایش می گذارد.

ج — مراحل الف و ب را ماشین آنقدر تکرار می کند تا تمام حروف x تمام شود.

د — اگر در این مرحله هیچ حرفی باقی نمانده باشد، رشته مورد نظر پذیرفته شده و در غیر این صورت رد می شود.
درسه مثال بالا خواننده می بایست جزئیات لازم برای تکمیل الگوریتم را اضافه کند.

۳.۸ تز چرچ — تورینگ

آیا می توان مدل محاسبه‌ای تصور کرد که توانایی اش بیشتر از ماشین تورینگ باشد؟ به بیان دیگر آیا یک مدل محاسبه یا ماشین انتزاعی وجود دارد که بتواند زبان هایی را تشخیص دهد که ماشین تورینگ از شناسایی آنها ناتوان باشد؟ اگر حل پذیری توسط یک ماشین تورینگ را به معنای حل پذیری توسط الگوریتم تلقی کنیم یک بیان دیگر این سوال این است که آیا مسایلی وجود دارند که به طور الگوریتمیک قابل حل باشند ولی نتوان این الگوریتم ها را در ماشین تورینگ پیاده سازی کرد؟

برای چنین سوالی پاسخ قطعی نمی توان ارایه داد و نمی توان آن را به صورت یک قضیه بیان کرد. پاسخ این سوال که منفی نیز هست چیزی است که به نام تز چرچ-تورینگ²⁴ شناخته می شود. این تز بیان می کند که هیچ مدل محاسبه ای دیگری وجود ندارد که قدرتمند تر از ماشین تورینگ باشد، به عبارت دیگر هر نوع ماشین دیگر، هر نوع مدل محاسبه ای دیگر قابل شبیه سازی توسط ماشین تورینگ است. در طول شش دهه گذشته از زمان وضع این تز توسط تورینگ و چرچ شاهدی بر نقض این تز بدست نیامده است. در واقع تز چرچ - تورینگ می گوید که هر نوع محاسبه ای که در چارچوب قوانین فیزیکی و توسط یک پدیده ی فیزیکی قابل انجام باشد، قابل پیاده شدن توسط یک ماشین تورینگ است. بنابراین از یک طرف این تز تعریف کننده آگوریتم است، (آگوریتم هر آن چیزی است که توسط ماشین تورینگ قابل شبیه سازی باشد) و از طرف دیگر نشان دهنده ی محدودیت های قوانین و پدیده های فیزیکی برای محاسبه کردن و استدلال آگوریتمی هستند. اگر به این نزاع اعتقاد راسخ داشته باشیم، آن گاه شکست این تز می تواند به این معنا باشد که گستره جدیدی از قوانین فیزیک کشف شده اند که رفتار آنها قابل پیا بعضی ها معتقدند که تز چرچ تورینگ رابطه خیلی مهمی با قوانین بنیادی فیزیک دارد، زیرا به یک معنا این رابطه می گوید که محدودیت های پدیده های طبیعی برای محاسبه کردن چه هستند؟

۴.۸ ماشین تورینگ جهان شمول

ماشین های تورینگی که تا کنون از آنها سخن گفته ایم همگی کارهای خاصی را انجام می دهند و برنامه آنها چنان نوشته شده است که یک محاسبه معین را انجام دهند یا یک زبان خاص را شناسایی کنند. اما می توان یک ماشین تورینگ جهان شمول²⁵ ساخت که بتواند هر ماشین تورینگ دیگری را شبیه سازی کند. این ماشین که اختصاراً آن را با UTM نشان می دهیم در واقع مدل نظری کامپیوترهای قابل برنامه ریزی امروزی است. در کامپیوترهای امروزی ما می توانیم با افزودن نرم افزارهای جدید به کامپیوتر کاری کنیم که کامپیوتر اولیه مان بتواند کارهای تازه تری انجام دهد. با نصب نرم افزارهای $Mathematica$ یا $Mathematica$ کاری می کنیم که کامپیوتر ساده ی اولیه توانایی محاسبه ریاضی و تحلیلی پیدا کند، یا با نصب یک نرم افزار دیگر این توانایی را به آن می دهیم که نقش یک کتاب لغت یا دایره المعارف گویا را نیز بازی کند یا این که بازی شطرنج و بازی های دیگر را به نحو استادانه ای بازی کند و هزاران کار متنوع دیگر. همه اینها نشان دهنده این هستند که لب تاپ روی میز ما یک ماشین تورینگ جهان شمول است یعنی می تواند ماشین های تورینگ دیگر را شبیه سازی کند. البته یک لب تاپ به ماشین تورینگ نزدیک است ولی یک ماشین تورینگ ایده آل نیست زیرا به هر حال حافظه محدودی در دسترس دارد و به اصطلاح طول نوارش محدود (اگر چه بسیار بزرگ) است. از لحاظ نظری یک ماشین جهان شمول تورینگ چگونه ساخته می شود؟ برای پاسخ به این سوال کافی است دقت کنیم که ماشین های تورینگ یک مجموعه شمارش پذیر را تشکیل می دهند. دلیل این امر آن است که یک ماشین تورینگ با تعداد حالت هایش یعنی اندازه مجموعه ی Q ، حروف الفبایش، و تعداد دستورهایش مشخص می شود. هر دستور نیز چیزی نیست جز یک نگاشت از یک مجموعه محدود به یک مجموعه ی محدود دیگر. شمارش پذیر بودن مجموعه ی تمام ماشین های تورینگ این امکان را به ما می دهد که به هر ماشین یک عدد طبیعی نسبت دهیم. این عدد، عدد تورینگ²⁶ آن ماشین خوانده می شود. بنابراین می توانیم از ماشین های تورینگ $M_1, M_2, \dots, M_K, \dots$ نام ببریم. برای آنکه ماشین جهان شمول تورینگ را بسازیم کافی است که علاوه بر رشته

Church-Turing Thesis²⁴
Universal Turing Machine²⁵
Turing Number²⁶

ورودی $x = x_1 x_2 \dots x_n$ ، برنامه مورد نظر (یعنی شماره ماشین تورینگ مورد نظر) را به صورت یک رشته به ماشین UTM بدهیم. بنابراین هرگاه خروجی ماشین M_i روی ورودی x را با $M_i(x)$ نشان دهیم، ماشین جهان شمول کار زیر را انجام می دهد:

$$\begin{aligned} UTM : (i \ b \ x) &\longrightarrow (i \ b \ M_i(x)) \\ UTM : (j \ b \ x) &\longrightarrow (j \ b \ M_j(x)) \\ UTM : (k \ b \ x) &\longrightarrow (k \ b \ M_k(x)) \\ &\dots \\ &\dots \end{aligned} \quad (28)$$

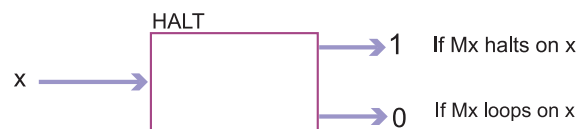
۵.۸ مسئله‌ی توقف

یکی از مفاهیم بسیار عمیق که پیامدهای گسترده‌ای دارد مسئله توقف یا ایستادن²⁷ است. برای پاسخ به مسئله هیلبرت در مورد وجود یک الگوریتم که بتواند همه مسائل ریاضیات را حل کند، تورینگ این پرسش را طرح کرد که آیا یک الگوریتم وجود دارد که بتواند بگوید یک ماشین تورینگ M_x روی یک ورودی y توقف می کند یا نه؟ برای پاسخ به این سوال تورینگ پرسش حتی ساده‌تری را پیش نهاد و آن اینکه آیا یک الگوریتم وجود دارد که بتواند بگوید یک ماشین تورینگ M_x روی یک ورودی x توقف می کند یا نه؟ یادآوری می کنیم که در این جا هر ماشین تورینگ را با شماره اش نشان داده‌ایم و از شمارش پذیر بودن ماشین های تورینگ استفاده کرده‌ایم. اگر چنین الگوریتمی وجود داشته باشد معنایش این است که تابع زیر یک تابع قابل محاسبه است:

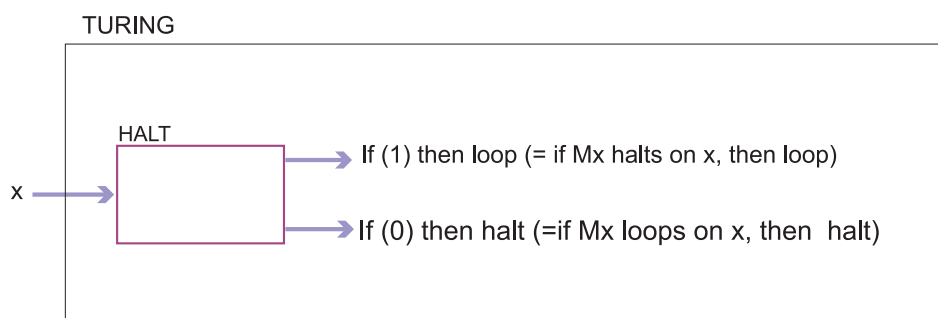
$$h(x) = \begin{cases} 1 & \text{if } M_x \text{ halts on } x, \\ 0 & \text{if } M_x \text{ doesn't halt on } x \end{cases} \quad (29)$$

معنای محاسبه پذیر بودن این است که یک ماشین تورینگ (الگوریتم) وجود دارد که می تواند این تابع را حساب کند و برای هر ورودی x مقدار $h(x)$ را که 0 است یا 1 تعیین کند. این ماشین تورینگ یا الگوریتم را $HALT$ می نامیم، شکل ۹۱. حال نشان می دهیم که فرض وجود چنین ماشینی به تناقض منطقی می انجامد. برای این کار به ترتیب زیر استدلال می کنیم. یک برنامه بزرگتر درست می کنیم که برنامه‌ی $HALT$ را فراخوانی کند. شکل ۹۲. طرز کار این برنامه را نشان می دهد. دقت کنید که این برنامه که آن را $Turing$ می نامیم باز هم ورودی اش یک عدد صحیح x است. اگر برنامه $Halt$ وجود داشته باشد، برنامه‌ی $Turing$ هم وجود خواهد داشت. تا اینجا هیچ تناقضی وجود ندارد. برای نشان دادن تناقض، دقت می کنیم که ماشین $Turing$ خودش می بایست یک شماره داشته باشد که با عدد صحیح t بیان شود. بنابراین می توان نوشت $Turing = M_t$. حال به این ماشین همان عدد t را می دهیم. حال همان طور که شکل ۹۲ نشان می دهد، به یک تناقض منطقی می رسیم. زیرا بنابراین شکل

²⁷ Halting Problem



شکل ۹۱: آگوریتیم $HALT$ برای محاسبه تابع h .



شکل ۹۲: اگر آگوریتیم یا ماشین تورینگ $HALT$ وجود داشته باشد، آگوریتیم یا ماشین تورینگ $Turing$ نیز وجود خواهد داشت.

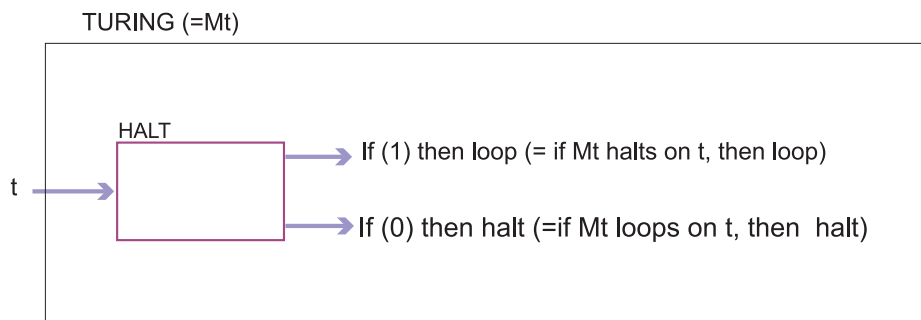
$$\begin{aligned} M_t \text{ halts on } t & \text{ if } M_t \text{ loops on } t, \\ M_t \text{ loops on } t & \text{ if } M_t \text{ halts on } t, \end{aligned} \quad (30)$$

این که ماشین $HALT$ وجود ندارد، به این معناست که تابع $h(x)$ محاسبه پذیر نیست. به عبارت دیگر هیچ آگوریتیمی وجود ندارد که این تابع را حساب کند.

اگر چنین ماشین تورینگی وجود داشته باشد به این معناست که می تواند بگوید آیا یک آگوریتیم کار می کند یا نه؟ برای آنکه اهمیت این مسئله را دریابیم یادآوری می کنیم که اگر چنین ماشینی وجود داشته باشد به این معناست که می توان تمام قضایای مهم ریاضی را به طور الگوریتمی حل کرد. به عنوان مثال یک مسئله مهم در نظریه اعداد مثل حدس گلدباخ²⁸ را در نظر بگیرید. این حدس بیان می کند که هر عدد زوج را حتماً می توان به صورت مجموع دو عدد اول نوشت. کافی است که خواننده به مثال های ساده ی زیر توجه کند:

$$4 = 2 + 2, \quad 6 = 3 + 3, \quad 8 = 3 + 5, \quad 10 = 5 + 5, \quad 20 = 7 + 13, \quad 30 = 23 + 7, \dots \quad (31)$$

Goldbach Conjecture²⁸



شکل ۱۲: می‌توانیم به ماشین Turing یا همان M_t ورودی t را بدهیم و تناقض منطقی وجود ماشین Turing و در نتیجه ماشین HALT را بنیم.

تا کنون کسی نتوانسته است این حدس را اثبات کند و یا مثال نقیضی بر علیه آن بیاورد و این مسئله همچنان یک مسئله باز در ریاضی است. حال فرض کنید که ما یک برنامه‌ی $M_{Goldbach}$ می‌نویسیم و این برنامه هر عدد زوج x را می‌گیرد و تست می‌کند که آیا این عدد مجموع دو عدد اول هست یا نه. اگر جواب مثبت بود ماشین می‌ایستد و جواب اری می‌دهد و اگر جواب منفی بود بازهم ماشین می‌ایستد و جواب نه می‌دهد. مسلم است که چنین برنامه‌ای را می‌توان نوشت. این برنامه یعنی برنامه $M_{Goldbach}$ تکلیف هر عددی را روشن می‌کند و ما می‌توانیم این برنامه را برای هر عددی از ۲ تا یک عدد زوج بسیار بسیار بزرگ چند هزار رقمی اجرا کنیم و می‌توانیم که مثلاً بگوییم که حدس گلدباخ تا اعداد ۱۰۰۰ رقمی صحیح است. ولی این به این معنی نیست که ما حدس گلدباخ را ثابت کرده‌ایم. (البته اگر یک عدد نقیض پیدا کنیم آنگاه حدس گلدباخ را نفی کرده‌ایم.) در واقع زبان این ماشین به این صورت است:

$$L = \{The \text{ set of all even integers which are the sum of two prime numbers}\}. \quad (32)$$

۹ کلاس‌های پیچیدگی محاسباتی

یکی از اهداف نظریه محاسبه، بررسی حداقل منابعی است که برای حل کلاس‌های مسائل مورد نیاز است. این منابع عبارتند از زمان²⁹، حافظه³⁰ و انرژی³¹. قبلاً درباره منبع انرژی بحث کرده‌ایم. در این درس دو منبع دیگر یعنی زمان و حافظه را بررسی می‌کنیم. یک مسئله یا یک الگوریتم معین را در نظر بگیرید. به عنوان مثال می‌خواهیم بنیم که آیا در یک گراف یک مسیر هامیلتونی وجود دارد یا نه؟³² ورودی این مسئله اطلاعات مربوط به یک گراف است. یک گراف عبارت است از مجموعه‌ای از نقاط که بعضی از آنها به هم وصل شده‌اند. مجموعه نقاط یا راس‌ها را با V و مجموعه یال‌ها را با E نشان می‌دهیم. در واقع $E \subset V \times V$ ، یعنی اینکه عناصر E جفت‌هایی هستند به صورت (v, w) که نشان دهنده این است که راس‌های v و w به هم وصل هستند. معمولاً یک گراف را به صورت $G = (V, E)$ نشان می‌دهیم و تمامی اطلاعات این گراف را به صورت یک رشته طولانی که در آن راس‌های V و یال‌های E با فاصله از هم جدا شده‌اند، به صورت یک رشته به ماشین تورینگ می‌دهیم. طول این رشته را اندازه‌ی این مسئله خاص می‌گیریم و با عدد صحیح n نشان می‌دهیم. سوالی که با آن روبرو هستیم

²⁹Time

³⁰Space

³¹Energy

³²این مسئله در بخش‌های بعدی این درس تعریف شده است.

این است که برای حل مسئله مورد نظر با اندازه n حداقل زمان لازم یا حداقل حافظه لازم چقدر است. تعیین این حداقل منابع اهمیت زیادی در نظریه محاسبه دارد. نکته‌ی مهم آن است که ما به رفتار مجانبی این زمان ها یا حافظه ها وقتی که اندازه مسئله بزرگ می شود توجه داریم و نه به منابع لازم برای حل یک مسئله خاص. می خواهیم بدانیم که وقتی گراف های بزرگ و بزرگ تر را بررسی می کنیم زمان لازم یا حافظه لازم برای حل مسئله چگونه رشد می کند؟ بنابراین نخست می بایست روی تعاریفی از رفتار مجانبی توابع توافق کنیم.

تعریف: الف: می گوییم که تابع $f(n)$ از نوع $O(g(n))$ است، اگر برای n های به اندازه کافی بزرگ، تابع $f(n)$ از مضربی از $g(n)$ کوچکتر باشد، به طور دقیق تر

$$f(n) \text{ is } O(g(n)), \quad \text{if } \exists c \text{ and } n_0 \quad \forall n > n_0 \quad f(n) \leq cg(n), \quad (33)$$

ب: می گوییم که تابع $f(n)$ از نوع $\Omega(g(n))$ است، اگر برای n های به اندازه کافی بزرگ، تابع $f(n)$ از مضربی از $g(n)$ بزرگتر باشد، به طور دقیق تر

$$f(n) \text{ is } \Omega(g(n)), \quad \text{if } \exists c \text{ and } n_0 \quad \forall n > n_0 \quad f(n) \geq cg(n), \quad (34)$$

می گوییم الف: می گوییم که تابع $f(n)$ از نوع $\Theta(g(n))$ است، اگر برای n های به اندازه کافی بزرگ $f(n)$ و $g(n)$ هم اندازه باشند، به طور دقیق تر

$$f(n) \text{ is } \Theta(g(n)), \quad \text{if } f(n) = O(g(n)), \quad \text{and } f(n) = \Omega(g(n)). \quad (35)$$

تعریف هرگاه ماشین تورینگ M به ازای ورودی x بعد از طی تعداد $t(x)$ مرحله (حرکت قلم) متوقف شود، می گوییم که زمان اجرا برای ورودی x ، t بوده است. هرگاه یک ماشین تورینگ داشته باشیم که یک زبان L را شناسایی کند، زمان اجرای آن مسئله را چنین تعریف می کنیم:

$$t(L) := \max_{x \in L} t(x). \quad (36)$$

به عبارت بهتر زمان اجرای یک مسئله را بزرگترین زمان اجرایی می گیریم که برای مثال های آن مسئله لازم بوده است. به عنوان مثال هرگاه L عبارت باشد از زبانی که تمام گراف های با n راس را توصیف می کند و یک ماشین تورینگ در باره هامیلتونی بودن یا نبودن این گراف ها تصمیم گیری می کند، زمان اجرا را برابر با بیشترین زمان اجرایی می گیریم که برای یک گراف در این کلاس به کار رفته است. این زمان اجرا تابعی است از n .

بعد از این تعاریف آماده ایم که چند کلاس پیچیدگی مهم را تعریف کنیم.

۱.۹ کلاس P

تعریف: به کلاس مسایلی که یک ماشین تورینگ با زمان اجرای $O(n^k)$ آنها را حل می کند، کلاس $DTime(n^k)$ می گوییم. لفظ $DTime$ به این خاطر آمده است که از یک ماشین تورینگ تعیینی استفاده کرده ایم. معنای این حرف این است که

حل دشوارترین مسائل در این کلاس نیز به زمان اجرایی کمتر از مضربی از n^k نیاز دارد. دقت کنید که $DTime(n^k)$ خانواده‌ای از مسئله‌ها یا زبان‌هاست و نه یک زبان خاص. حال می‌توانیم کلاس مسائل P ³³، یعنی مسائل چند جمله‌ای را تعریف کنیم:

$$P := \bigcup_{k \geq 0} DTime(n^k). \quad (37)$$

کلاس مسایل P ، مجموعه همه مسایل یا الگوریتم‌هایی است که در زمان چند جمله‌ای حل می‌شوند. طبیعی است که چنین مسائلی را مسائل آسان بنامیم. به عنوان مثال، مسئله‌ای مثل مرتب کردن یک مجموعه n تایی از اعداد، یا ضرب دو ماتریس n بعدی، یا جستجو در یک پایگاه داده‌ها که n عضو دارد، همه مسائلی هستند که برای حل آنها زمان چند جمله‌ای مورد نیاز است. بسیاری از مسائلی که نرم افزارهایی مثل *Maple* و *Mathematica* حل می‌کنند نیز چنین اند. البته نمی‌توان گفت که هر نوع مسئله‌ای که این برنامه‌ها و نظایر آن حل می‌کنند از نوع مسائل آسان یا P است، زیرا آنها فقط یک نمونه از مسئله را با اندازه معین حل می‌کنند که به ازای حل آن هم بعضاً زمان طولانی صرف می‌کنند. به عنوان مثال اگر وقت کافی به هر کدام از این برنامه‌ها بدهید قادر خواهند بود که یک عدد بزرگ را تجزیه کنند، ولی مسئله تجزیه عدد یک مسئله از نوع P نیست زیرا الگوریتمی که بر حسب اندازه مسئله (تعداد رقم‌های عدد اولیه) چند جمله‌ای باشد شناخته نشده است. (در اینجا فقط درباره الگوریتم‌های کلاسیک بحث می‌کنیم).

۲.۹ کلاس NP

تعریف: به کلاس مسایلی که یک ماشین تورینگ نامعین در زمان $O(n^k)$ آنها را حل می‌کند، کلاس $NDTime(n^k)$ می‌گوییم. لفظ $NDTime$ به این خاطر آمده است که از یک ماشین تورینگ نامعین استفاده کرده ایم. حال می‌توانیم کلاس مسائل NP ³⁴ را تعریف کنیم:

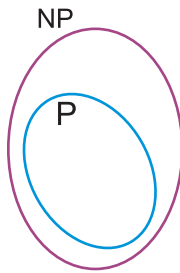
$$NP := \bigcup_{k \geq 0} NDTime(n^k). \quad (38)$$

از آنجا که یک ماشین تورینگ معین نوع خاصی ماشین تورینگ نامعین است نتیجه بدیهی تعاریف بالا آن است که $DTime(n^k) \subset NDTime(n^k)$ و در نتیجه

$$P \subset NP. \quad (39)$$

هرگاه به شکل ۷ نگاه کنیم که فلسفه کلی ماشین‌های نامعین را نشان می‌دهد، این موضوع روشن می‌شود که یک ماشین تورینگ نامعین وقتی یک مسئله را در زمان چند جمله‌ای حل می‌کند که یکی از شاخه‌های محاسبه به جواب برسد، یا به حالت q_{Accept} برسد. اگر بخواهیم کاری که یک ماشین تورینگ نامعین را با یک ماشین تورینگ معین شبیه سازی کنیم طبیعتاً می‌بایست همه شاخه‌ها را همزمان دنبال کنیم. این کار موجب افزایش نمایی تعداد حالت‌ها و تعداد مراحل محاسبه می‌شود. بنابراین به

Polynomial Time Problems or Polynomial Time Algorithms³³
Non-deterministic Polynomial Time Problems or Algorithms³⁴



شکل ۲۲: کلاس های P و NP .

طور شهودی وقتی که یک مسئله روی یک ماشین تورینگ نامعین زمان اجرای چندجمله‌ای نیاز دارد به این معناست که این مسئله روی ماشین تورینگ معین زمان اجرای نمایی از نوع $O(2^n)$ است. در نتیجه می توان گفت که این نوع مسائل، نوعاً مسایل دشوارند، زیر زمان لازم برای حل آنها با افزایش اندازه مسئله به صورت نمایی رشد می کند. البته ممکن است که بتوان برای همان مسئله روی ماشین تورینگ معین هم یک الگوریتم چندجمله‌ای پیدا کرد. این سوال که آیا کلاس NP واقعاً از کلاس P بزرگتر است، یک سوال باز و بسیار مهم در نظریه محاسبه است. در حال حاضر اعتقاد عمومی این است که $P \neq NP$ ، شکل ۲۲.

یک تعریف دیگر از کلاس NP که با تعریف بالا معادل است با شهود زیر بدست می آید. عموماً دشواری یک مسئله‌ی NP ناشی از آن است که برای یافتن جواب می بایست یک فضای بسیار بزرگ که اندازه آن نمایی است مورد جستجو قرار گیرد. به عنوان مثال تعیین اینکه یک گراف هامیلتونی است یا نه، (اگر هیچ قضیه‌ی کلی‌ای در مورد آن نداشته باشیم) نیازمند تست کردن تمام مسیرهای ممکن در این گراف است و تعداد این مسیرها یک تابع نمایی از تعداد راس هاست. یک ماشین تورینگ نامعین با شاخه شاخه شدن باعث کاهش زمان لازم برای این جستجو به یک زمان چندجمله‌ای می شود. با توجه به این مقدمه فرض کنید که در ماشین تورینگ نامعین، ما علاوه بر ورودی x مسیری را نیز که ماشین می بایست طی کند، تا به یک حالت پذیرش برسد به ماشین بدهیم. این مسیر را یا هر نوع اطلاع یا گواهی³⁵ را که منجر به یافتن آن می شود با y نشان می دهیم. در این صورت زمان لازم برای حل مسئله حتماً چندجمله‌ای خواهد بود. بنابراین می توانیم بگوییم که یک زبان L متعلق به کلاس NP است اگر یک ماشین تورینگ M وجود داشته باشد به نحوی که

$$\begin{aligned} & \text{if } x \in L \exists y \in L \mid TM \text{ halts in } q_{Accept} \text{ on the input } (x \parallel y) \text{ after time } (Poly(n)), \\ & \text{if } x \notin L \forall y \in L \mid TM \text{ halts in } q_{Reject} \text{ on the input } (x \parallel y) \text{ after time } (Poly(n)). \end{aligned} \quad (40)$$

در اینجا منظور از n اندازه ورودی x است.

³⁵Witness

۱۰ فروگاهش و مسایل کامل

می‌گوییم زبان A به زبان B کاهش پذیر است اگر یک ماشین تورینگ M وجود داشته باشد به نحوی که به ازای هر $x \in A$ ، یک رشته مثل $R(x) \in B$ را در زمان چند جمله‌ای حساب می‌کند و شرط زیر برقرار است:

$$R(x) \in B \leftrightarrow x \in A. \quad (41)$$

معنای این حرف آن است که به جای تصمیم‌گیری در مورد تعلق رشته‌ی x به زبان A می‌توانیم تنها با صرف زمان چند جمله‌ای در مورد مسئله معادل آن یعنی عضویت رشته‌ی $R(x)$ در زبان B تصمیم‌گیری کنیم. به عبارت دیگر مسئله بجای حل مسئله‌ی x می‌توانیم تنها با صرف زمان اضافی چند جمله‌ای، مسئله معادل آن یعنی $R(x)$ حل کنیم. هرگاه که این خاصیت برای همه‌ی $x \in A$ برقرار باشد، آنگاه مسئله یا زبان A به مسئله یا زبان B کاهش یافته‌است. فروگاهش یک خاصیت تراگذار³⁶ است به این معنا که اگر A به B کاهش پذیر باشد و B به C آنگاه A به C کاهش پذیر است.

دقت کنید که کاهش پذیری A به B علی‌رغم ظاهر متقارن رابطه‌ی $??$ برابر با معادل بودن B و A نیست، زیرا ممکن است که بتوان به ازای هر $x \in A$ ، $R(x) \in B$ را در زمان چند جمله‌ای حساب کرد ولی وارون آن ممکن است صحیح نباشد. زبان A و B وقتی معادل خواهند بود که هم A به B کاهش پذیر باشد و هم B به A .
تعریف: در یک خانواده از زبان‌ها مثل $\mathcal{L}$ می‌گوییم زبان C کامل یا $\mathcal{L}$ -complete است اگر همه زبان‌های متعلق به $\mathcal{L}$ به زبان C کاهش پذیر باشند.

مفهوم کامل بودن یکی از مفاهیم بسیار مهم در نظریه محاسبه و کلاس‌های پیچیدگی است. مهمترین زبان‌های کامل آنهایی هستند که در کلاس NP قرار دارند و به آنها زبان‌ها یا مسائل NP -Complete می‌گویند. در سال ۱۹۷۰ کوک و لوین³⁷ نشان دادند که مسئله $3-Sat$ یک مسئله NP -کامل است. از آن به بعد تعداد بسیار زیادی مسئله NP -کامل یافته شده‌اند که در زیر به آنها اشاره می‌کنیم. معنای این نتایج آن است که هرگاه یک الگوریتم چند جمله‌ای برای یکی از این مسئله‌ها یافته شود، آنگاه تمامی مسائل کلاس NP در زمان چند جمله‌ای حل خواهند شد و تساوی $P = NP$ برقرار خواهد شد. البته همانطور که در بالا گفتیم باور عمومی این است که کلاس مسائل NP از کلاس مسائل P بزرگتر است. در زیر چند نمونه از مسایل NP -کامل را معرفی می‌کنیم.

Transitive³⁶
Kook and Levin³⁷

۱۱ نمونه هایی از مسایل کامل در کلاس NP

۱.۱۱ مسئله تصدیق عبارت های منطقی

فرض کنید که $A = \{x_1, x_2, \dots, x_n\}$ متغیرهای منطقی هستند که می توانند مقادیر درست 1 یا نادرست 0 را اختیار کنند. یک عبارت k -تایی، عبارتی است مثل

$$x_{i_1} \vee x_{i_2} \vee x_{i_3} \vee \dots \vee x_{i_K}, \quad (42)$$

که در آن هر کدام از x_i ها یا یکی از متغیرهای A ست یا یکی از متغیرهای $\bar{A} = \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$. به عنوان مثال یک $4 - Clause$ عبارتی است مثل $x_1 \vee \bar{x}_3 \vee \bar{x}_5 \vee x_6$. حال یک جمله شامل تعدادی عبارت هم اندازه در نظر می گیریم مثل

$$S = C_1 \wedge C_2 \wedge C_3 \wedge \dots \wedge C_p. \quad (43)$$

سوال این است که آیا یک انتخاب از متغیرهای A وجود دارد که به ازای آنها عبارت S مقدار 1 را اختیار کند یعنی عبارت S ارزش منطقی درست داشته باشد یا خیر؟ لازمه این کار آن است که عبارت های C_1 تا C_p همه مقادیر 1 را اختیار کنند. بنابراین، این مسئله را که اصطلاحاً مسئله $K - Sat$ ³⁹ می نامند، می توان به عنوان یک مسئله p معادله و n مجهولی از متغیرهای منطقی نامید، به این معنا که می خواهیم ببینیم دستگاه معادلات منطقی زیر جواب دارند یا نه:

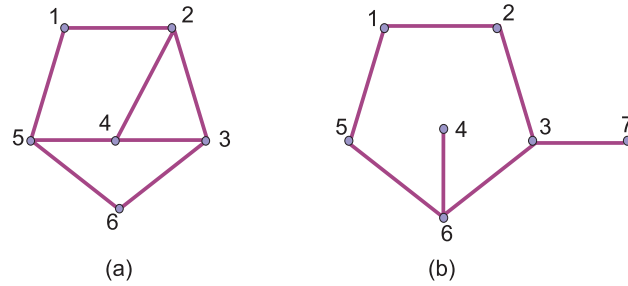
$$\begin{aligned} C_1 &= 1 \\ C_2 &= 1 \\ &\dots \\ C_p &= 1. \end{aligned} \quad (44)$$

بعد از آنکه اثبات کامل بودن مسئله تصدیق در کلاس NP توسط کوک و لوین در سال ۱۹۷۰، تعداد بسیار زیادی از مسائل کامل در همین کلاس کشف شدند. تعداد این نوع مسائل آنقدر زیاد شده است (حدود ۲۰۰۰ مسئله) که امروزه دیگر اثبات کامل بودن یک مسئله در کلاس NP هیجان چندانی ایجاد نمی کند. نمونه ای از این مسایل در ادامه می آید.

۲.۱۱ مسئله گراف های هامیلتونی

فرض می کنیم که خواننده با مفهوم گراف آشناست. در یک گراف $G = (V, E)$ که دارای راس های V و یال های E است یک مسیر، مسیر هامیلتونی⁴⁰ خوانده می شود، هرگاه این مسیر تمام راس های گراف را یک بار و فقط یک بار ملاقات کند (از آنها بگذرد). یک گراف هامیلتونی⁴¹، گرافی است که حداقل یک مسیر هامیلتونی داشته باشد. به عنوان مثال در شکل ۳۲ گراف a هامیلتونی است ولی گراف b هامیلتونی نیست.

Clause³⁸
K-Satisfiability³⁹
Hamiltonian Path⁴⁰
Hamiltonian Graph⁴¹



شکل ۳۲: گراف a هامیلتونی است ولی گراف b هامیلتونی نیست.

۳.۱۱ مسئله فروشنده دوره گرد

یک گراف وزن دار $G = (V, E)$ ^{۴۲} عبارت از گرافی است که به هر کدام از یال های آن یک عدد نامنفی نسبت داده شده است. می توان راس های این گراف را به عنوان شهرهای یک منطقه و وزن ها را مسافت بین این شهرها یا مثلاً بهای سفر بین این شهرها تلقی کرد. مسئله فروشنده دوره گرد عبارت است از پیدا کردن ارزان ترین مسیر بسته ای که از همه شهرهای این منطقه بگذرد. در این مسئله نیز سبایز مسئله همان سبایز گراف است.

۴.۱۱ مسئله پوشش راس ها در یک گراف

یک گراف $G = (V, E)$ در نظر می گیریم. یک پوشش راسی برای این گراف عبارت است از زیرمجموعه ای مثل $W \subset V$ ، به طوری که هر یال متعلق به E یکی یا هر دو سرش متعلق به W باشد. به عنوان مثال در شکل ۴.۱۱ یک پوشش راسی برای گراف است. مسئله پوشش راسی به این ترتیب است: یک گراف $G = (V, E)$ و یک عدد صحیح مثبت m داده شده است. آیا این گراف یک پوشش راسی با تعداد m راس دارد؟ به عنوان مثال در گراف a در شکل ۳۲ مجموعه $W = \{1, 4, 6\}$ یک پوشش راسی نیست ولی مجموعه $W' = \{1, 2, 4, 6\}$ یک پوشش راسی است.

۵.۱۱ مسئله زیرگراف های کامل

در یک گراف $G = (V, E)$ یک زیرگراف کامل یا $Clique$ به یک زیرگراف گفته می شود که همه راس هایش به هم وصل باشند. مسئله زیرگراف کامل می خواهیم ببینیم که اگر یک گراف با n راس داده شده باشد، آیا این گراف یک زیرگراف کامل با سبایز k دارد یا خیر؟ در شکل ۳۲ گراف شکل a بزرگترین زیرگراف کامل اش اندازه ۳ دارد و گراف شکل b بزرگترین زیرگراف کامل اش اندازه ۲ دارد.

۶.۱۱ مسئله جمع زیر مجموعه ها

یک مجموعه S از اعداد صحیح و یک عدد صحیح t داده شده است. آیا یک زیر مجموعه از S وجود دارد که مجموع عناصرش برابر با t باشد. این مسئله موسوم به مسئله $SubsetSum$ است و یک مسئله کامل در کلاس NP است.

^{۴۲}Weighted graph

۱۲ کلاس $PSPACE$

برای آنکه منابع حافظه‌ای ⁴³ را برای یک الگوریتم بررسی کنیم، احتیاج به تعریفی از میزان مصرف حافظه داریم. تعداد خانه‌هایی از نوار را که یک ماشین تورینگ مصرف می‌کند تا به حالت q_{Halt} برسد، حافظه یا فضای مصرف شده می‌خوانیم. بنابراین می‌گوییم یک زبان L در کلاس $PSPACE$ است اگر یک ماشین تورینگ TM وجود داشته باشد به نحوی که

$$\forall x \in L, TM \text{ halts on } x \text{ after the consumption of } P(n) \text{ bits of tape where } n \text{ is the length of } x. \quad (45)$$

رابطه زیر بین این کلاس‌ها وجود دارد:

$$P \subset PSPACE, \quad NP \subset PSPACE. \quad (46)$$

نامساوی اول واضح است زیرا اگر یک ماشین تعداد مراحل را با گرفتن ورودی x طی می‌کند تا به حالت پایانی برسد، یک چند جمله‌ای برحسب $|x| = n$ باشد، حتماً تعداد خانه‌هایی را نیز که از نوار مصرف می‌کند چند جمله‌ای خواهد بود. برای نامساوی دوم احتیاج به استدلال بیشتری داریم. می‌دانیم که کار یک ماشین تورینگ نامعین را می‌توانیم روی یک ماشین تورینگ معین شبیه‌سازی کنیم. اگر اندازه رشته‌ی ورودی را با n نشان دهیم، ماشین تورینگ نامعین تعداد $O(2^n)$ مسیر را طی می‌کند و لااقل در یکی از این مسیرها در زمان چند جمله $O(n^k)$ به جواب می‌رسد. می‌توانیم آدرس هر کدام از این مسیرها را به ماشین تورینگ معین بدهیم. این آدرس که همان آدرس نحوه شاخه شاخه شدن در شکل $??$ است، یک رشته کوتاه به طول $O(n)$ است. در هر بار که ماشین تورینگ معین کار با ورودی x و آدرس شاخه‌ی داده شده کار می‌کند در زمان چند جمله‌ای کار خود را تمام می‌کند و به حالت q_{Halt} می‌رسد. بنابراین در این زمان فقط تعداد $O(n^k)$ از خانه‌های نوار تغییر می‌کنند. برای ورودی دیگری یعنی x و آدرس یک شاخه دیگر خانه‌های نوار دوباره به حالت اول برمی‌گردند. بنابراین برای تست کردن کل شاخه‌ها باز هم تعداد خانه‌های نوار که تغییر می‌کنند یک چند جمله‌ای از n خواهد بود. بنابراین نتیجه می‌گیریم که $NP \subset PSPACE$.

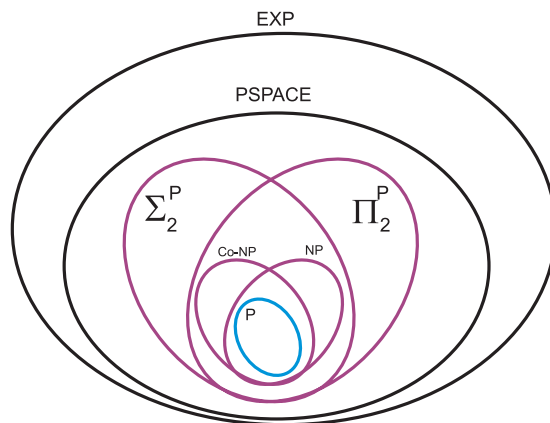
تعریف: کلاس L به کلاس مسائلی گفته می‌شود که در زمان لگاریتمی توسط یک ماشین تورینگ قابل تصمیم‌گیری هستند. به یک معنا این کلاس ساده‌ترین مسائل را دربردارد.

تعریف: کلاس EXP به کلاس مسائلی گفته می‌شود که در زمان نمایی توسط یک ماشین تورینگ قابل تصمیم‌گیری هستند. به یک معنا این کلاس سخت‌ترین مسائل را دربردارد.

رابطه کلاسهایی که تا کنون گفته ایم به شکل زیر است:

$$L \subset P \subset NP \subset PSPACE \subset EXP. \quad (47)$$

Memory Resources⁴³



شکل ۴۲: نمونه‌ای از کلاس‌های پیچیدگی و رابطه آنها.

علاوه بر کلاس‌های بالا ده‌ها کلاس پیچیدگی دیگر نیز وجود دارند که رابطه بین آنها از رابطه ساده بالا بسیار پیچیده‌تر است. پرداختن به این گونه کلاس‌ها و رابطه بین آنها از دایره بحث ما خارج است. خواننده علاقمند می‌تواند به کتب تخصصی این رشته مراجعه کند.

۱۳ تمرین‌ها:

۱ — برای هر کدام از زبان‌های زیر یک اتومات تعینی محدود طراحی کنید که آن زبان را تشخیص دهد. در همه زبان‌ها الفبا برابر است با $\Sigma = \{0,1\}$ بجز زبان‌های آخر که الفبای آن زبان انگلیسی است:

- $L_1 := \{w \mid w \text{ contains the substring } 1011\}$
- $L_2 := \{w \mid w \text{ contains an even number of } 0\text{'s and an odd number of } 1\text{'s}\}$
- $L_3 := \{w \mid w \text{ begins with } 0 \text{ and ends with } 1\}$
- $L_4 := \{w \mid w \text{ begins with } 00 \text{ and ends with } 11\}$
- $L_5 := \{w \mid w \text{ is of length } N\}$
- $L_6 := \{w \mid w \text{ does not contain the substring } 111\}$
- $L_7 := \{w \mid w \text{ begins and ends with the same letter}\}$
- $L_8 := \{w \mid w\text{'s length is a multiple of } 3\}$
- $L_9 := \{w \mid w\text{'s length is a multiple of } 3 \text{ and contains an even number of } 0\text{'s}\}$
- $L_{10} := \{w \mid w \text{ contains the word } WORK\}$
- $L_{11} := \{w \mid w \text{ contains the word } WORK \text{ twice}\}$ (48)

۲ — هر کدام از زبان‌های زیر اشتراک دو زبان ساده‌تر است. در هر مورد نخست ماشین‌های DFA ای بسازید که آن زبان‌های ساده‌تر را شناسایی کند. سپس ماشین DFA ای بسازید که زبان داده شده را شناسایی کند. در تمام موارد الفبای

زبان ها برابر است با $\Sigma := \{a, b\}$.

$$\begin{aligned} A &:= \{w \mid w \text{ contains at least two } a\text{'s and at least three } b\text{'s}\} \\ B &:= \{w \mid w \text{ contains at most two } a\text{'s and at least three } b\text{'s}\} \\ C &:= \{w \mid w \text{ contains at most two } a\text{'s and at most three } b\text{'s}\} \\ D &:= \{w \mid w \text{ contains the substring } aba \text{ and the substring } bab\} \\ E &:= \{w \mid w \text{ starts with } aaa \text{ and ends with } bbb\}. \end{aligned} \quad (49)$$

۳ – الف: فرض کنید که $A \subset \Sigma^*$ یک زبان باشد. در این صورت متمم آن زبان عبارت است از $A' := \Sigma^* / A$. نشان دهید که اگر M یک اتومات تعیینی محدود باشد که زبان A را شناسایی می کند آنگاه با عوض کردن حالت های $Accept$ و حالت های معمولی ماشینی ساخته می شود که زبان A' را شناسایی می کند؟ بنابراین نشان دهید که مجموعه زبان های منظم تحت عمل متمم گرفتن بسته است.

ب: نشان دهید که اگر M یک اتومات غیرتعیینی محدود باشد که زبان A را شناسایی می کند آنگاه با عوض کردن حالت های $Accept$ و حالت های معمولی ماشین ساخته شده الزاماً ماشینی نیست که زبان A' را شناسایی کند؟ یک مثال از این موضوع ارائه دهید.

۴ – در هر کدام از موارد زیر زبان داده شده متمم یک زبان ساده تر است. نخست یک اتومات تعیینی محدود بسازید که آن زبان ساده تر را شناسایی کند. سپس با استفاده از آنچه که در تمرین قبلی یاد گرفتید ماشینی بسازید که زبان داده شده را شناسایی کند:

$$\begin{aligned} A &:= \{w \mid w \text{ does not contain the substring } ab\} \\ B &:= \{w \mid w \text{ does not contain the substring } abab\} \\ C &:= \{w \mid w \text{ contains neither the substring } ab \text{ nor } ba\} \\ D &:= \{w \mid w \text{ is any string not in } a^*b^*\} \\ E &:= \{w \mid w \text{ is any string that does not contain exactly two } a\text{'s}\}. \end{aligned} \quad (50)$$

۵ – دیاگرام حالت ماشین های NFA ای را رسم کنید که هر کدام از زبان های زیر را تشخیص دهد. در تمام موارد الفبا عبارت است از $\Sigma = \{0, 1\}$.

$$A := 0^*1^*$$

$$\begin{aligned}
B &:= \sum 0 \sum \\
C &:= \sum 00 \sum \\
D &:= 0^*(10^*)1 \\
E &:= \sum 0^* \sum \\
F &:= (\sum \sum)^* \cup (\sum \sum \sum)^* \cup (01)^*.
\end{aligned} \tag{51}$$

۶ – ثابت کنید که هر NFA را می توان به یک NFA تبدیل کرد که تنها یک حالت قبولی داشته باشد.

الف: یک NFA بسازید که زبان $(01 \cup 001 \cup 010)^*$ را شناسایی کند.

ب: این NFA را به یک DFA معادل تبدیل کنید.

۸ – عبارت های منظم زیر را به NFA تبدیل کنید:

$$\begin{aligned}
&(0 \cup 1)^* 000 (0 \cup 1)^* \\
&(((00)^*(11)) \cup (01))^* \\
&0^*.
\end{aligned} \tag{52}$$

۹ – از لم پمپ کردن استفاده کنید و نشان دهید که زبان های زیر منظم نیستند:

$$\begin{aligned}
A &= \{0^n 1^n 2^n \mid n \geq 0\} \\
B &= \{www \mid w \in \{a, b\}^*\} \\
C &= \{a^{2^n} \mid n \geq 0\}.
\end{aligned} \tag{53}$$

۱۰ – هرگاه $\omega = \omega_1 \omega_2 \dots \omega_n$ یک رشته باشد معکوس آن رشته ی $\omega^R = \omega_n \dots \omega_2 \omega_1$ است. اگر A یک زبان باشد، $A^R = \{\omega^R \mid \omega \in A\}$ معکوس زبان A نامیده می شود. نشان دهید که اگر A منظم باشد، آنگاه A^R نیز یک زبان منظم است.