مقدمه ای بر نظریه محاسبه

وحیدکریمی پور- دانشکده فیزیک ـ دانشگاه صنعتی شریف

۱۰ دی ۱۳۹۳

۱ مقدمه

بسیاری از مسائل ریاضیات را ما به روش آلگوریتمی حل می کنیم به این معنا که مجموعه ای متناهی از مراحل را مطابق با یک ترتیب معین دنبال می کنیم و مطمئن هستیم که در پایان این مراحل به پاسخ مسئله می رسیم. به عنوان مثال محاسبه مشتق یک تابع، محاسبه وارون یک ماتریس، بزرگترین مقسوم علیه مشترک دو عدد صحیح را هرکدام با دنبال کردن یک آلگوریتم معین حل می کنیم. هرگاه بتوانیم با استفاده از یک ماشین (خواه مکانیکی، خواه الکتریکی یا مغناطیسی) این مراحل را با سرعت انجام بدهیم می توانیم این مسائل را برای نمونه های خیلی بزرگ یا دشوار نیز در زمان کوتاه انجام دهیم. این فایده آلگوریتم است یعنی وقتی که حل یک مسئله را به تکرار یک مجموعه اعمال مشخص تقلیل بدهیم همواره می توانیم بدون نیاز به فکر کردن از یک ماشین برای انجام این مجموعه اعمال کمک بگیریم و نمونه های خیلی بزرگ مسئله را حل کنیم. در نگاه اول به نظر می رسد که حل های آلگوریتمی تنها برای بعضی از مسائل خاص ریاضیات وجود دارند و بعضی از مسائل تن به راه حل آلگوریتمی نمی دهند و برای حل آنها همچنان باید فکر و ابتکار و خلاقیت به خرج داد. مثلا به نظر می رسد که در برای محاسبه انتگرال نامعین از یک تابع باید از نبوغ و فکر استفاده کرد، تغییر متغیر عجیب و غریبی داد، ابتکار ویژه ای به کار برد تا بتوان انتگرال را محاسبه کرد. برخلاف این تصور اولیه امروز معلوم شده است که انتگلرال نامعین را نیز می توان به صورت

آلگوریتمی محاسبه کرد. این کار را نخستین بار رابرت هنری ریش [۱] در سال ۱۹۶۸ انجام داد. از آن موقع تا کنون آلگوریتم های سریع تری برای دسته وسیع تری از توابع حتی توابع خاص ابداع شده اند به طوری که امروزه می توانیم بگوییم محاسبه آنتگرال را نیز می توان به صورت آلگوریتمی انجام داد که همانطور که دیدیم معنایش این است که این کار را می توان به ماشین ها سپرد چرا که آن را با سرعت خیلی بیشتر و بدون اشتباه می توانند انجام دهند.

همه ما از درس های اولیه هندسه در دبیرستان به یاد می آوریم که عموما مسائل هندسه را می بایست با ابتکارات خاص و با روشهای نه چندان سرراست حل کرد. و به یاد می آوریم که گاهی اوقات کشیدن یک خط اضافه راه رسیدن به اثبات یک قضیه را به ناگهان شفاف می کند. اما با اختراع هندسه تحلیلی اکنون می دانیم که همه مسائل هندسه را می توان با روش آلگوریتمی حل کرد. تساوی دو زاویه ، دو پاره خط، و نظایر آنها را همه می توان به راحتی تحقیق کرد و نیازی به هیچ نوع تفکر و ابتکار ویژه ندارد.

به این نقطه که می رسیم می توانیم یک سوال مهم طرح کنیم: آیا هر سوالی از ریاضیات را می توان به روش آلگوریتمی حل کرد؟ آیا می توان قضیه های ریاضیات را نیز به روش آلگوریتمی ثابت یا رد کرد؟ برای مثال به قضیه فرما توجه کنیم: صورت این قضیه این است که معادله زیر

$$x^n + y^n = z^n, \qquad n > 2 \tag{۱}$$

هیچ پاسخ صحیحی ندارد. آیا آلگوریتمی وجود دارد که تعیین کند این قضیه درست است؟ یا اگر این قضیه نادرست است، آیا الگوریتمی وجود دارد که یک مثال نقض برای آن پیدا کند؟ به یک مثال دیگر توجه کنیم: حدس گلدباخ [۲] بیان می کند که هر عدد زوجی را می توان به صورت مجموع دو عدد اول نوشت. این حدس تا کنون برای اعداد بسیار بزرگ نیز تایید شده ولی هنوز هیچ اثباتی برای آن وجود ندارد. سوال این است که آیا می توان اثبات این قضیه را به شکل یک آلگوریتم در آورد و آن را ثابت یا رد کرد؟

■ حدس گلدباخ را برای اعداد از ۲ تا ۵۰ تست کنید.

---

[۱] Robert Henry Risch

[۲] Goldbach Conjecture

در واقع سوال این است که آیا همان کاری را که دکارت برای هندسه تحلیلی انجام داد می توان برای کل ریاضیات انجام داد؟ سالهای اولیه قرن بیستم، سالهایی بوده است که این نوع سوال ها مورد توجه ویژه ریاضیدانان بوده است. تلاش برای پاسخ گویی به این سوال سالها قبل از اختراع اولین کامپیوترهای واقعی منجر به ابداع نظریه محاسبه توسط آلن تورینگ [۳] و آلونزو چرچ [۴] شد. تورینگ و چرچ به دو روش متفاوت ولی معادل اولین مدل های نظری محاسبه را ساختند و مهم تر از هر چیز نشان دادند که محدودیت های هر نوع مدل محاسبه ای چیست. به خصوص نتیجه کار آنها پاسخی منفی به مسئله دهم هیلبرت بود که می پرسید آیا می توان همه ریاضیات را به صورت آلگوریتم درآورد یا خیر؟

در این درس ما با مقدمات نظریه محاسبه آشنا می شویم. نخست با مسایل تصمیم گیری [۵] یعنی مسائلی که پاسخ آنها تنها آری یا خیر است آشنا می شویم و سپس نشان می دهیم که هر مسئله تصمیم گیری را می توان به عنوان شناسایی یا پذیرش رشته هایی از حروف الفبا یا به اصطلاح شناسایی یک زبان در نظر گرفت. سپس با ساده ترین آلگوریتم ها یا ماشین ها یعنی اتومات های محدود آشنا می شویم . سپس نشان می دهیم که این اتومات ها کلی ترین و قدرت مند ترین مدل های محاسبه نیستند. سرانجام با ماشین تورینگ آشنا می شویم که کامل ترین مدل محاسبه ای است که می شناسیم. سپس نشان می دهیم که چرا حتی ماشین تورینگ نیز قادر به پاسخ گویی به بعضی سوالات نیست. سرانجام بر اساس آنچه که آموخته ایم مختصری در باره نظریه پیچیدگی محاسباتی [۶] سخن خواهیم گفت. خواننده ای که بخواهد مطالب این فصل را به صورت گسترده تر و عمیق تر بخواند می تواند به کتاب زیر مراجعه کند:

Introduction to the theory of Computation, Michael Sipser, 2006.

---

[۳] Alan Turing

[۴] Alonzo Church

[۵] Decision Problems

[۶] Computational Complexity

۲ تعاریف مقدماتی

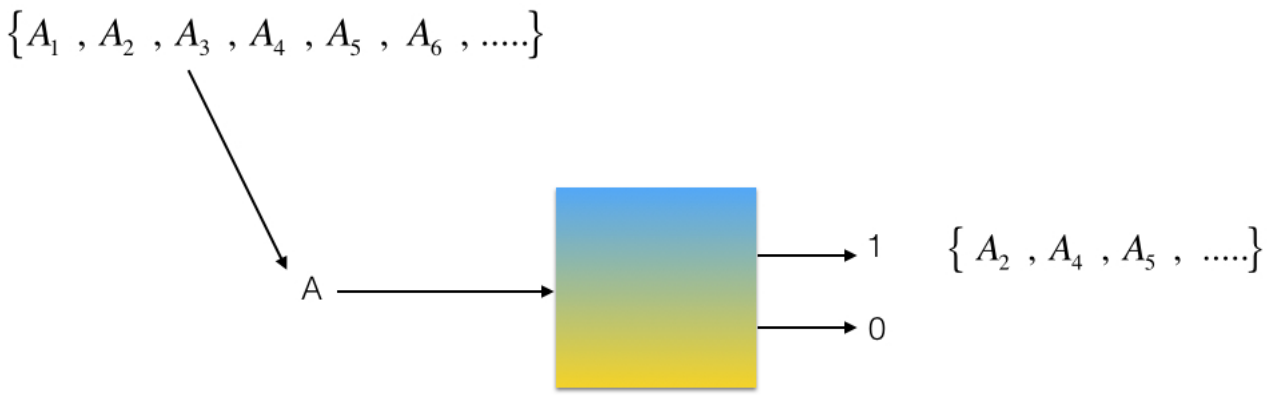
تقریبا هر مسئله ای که با آن سروکار پیدا می کنیم یا یک مسئله تصمیم گیری [۷] است یا اینکه قابل تبدیل به مسائلی از این نوع هستند. بنابر تعریف مسائل تصمیم گیری آنهایی هستند که پاسخ آنها فقط آری یا خیر است. مثلا می پرسیم که آیا یک عدد اول ا ست یا نه؟ آیا یک عدد بر عدد دیگر بخش پذیراست یا نه؟ آیا یک ماتریس وارون پذیراست یا نه؟ آیا یک معادله چندجمله ای مثل $x^3 + y^3 = z^3$ در محدوده اعداد صحیح حل دارد یا نه؟ تمام این مسائل از نوع تصمیم گیری هستند و هر الگوریتمی تنها باید به آنها پاسخ آری یا نه بدهد. ممکن است فکر کنیم که خیلی از مسائل دیگر چنین نیستند مثلا پیداکردن فاکتورهای یک عدد صحیح یک مسئله تصمیم گیری نیست. ولی نگاه دقیق تر نشان می دهد که این نوع مسائل را نیز می توان به مسائل دیگری تنها با پاسخ های آری یا نه تبدیل کرد. مثلا عددی مثل N را در نظر بگیرید. هدف ما یافتن فاکتوری از این عدد است. می پرسیم آیا این عدد فاکتوری کوچکتر از $\frac{N}{2}$ دارد یا نه؟ اگر پاسخ منفی باشد این سوال را در باره اعداد بزرگتر از $\frac{N}{2}$ می پرسیم . به این ترتیب متوجه می شویم که آیا این عدد فاکتور دارد یا نه و اگر دارد در کدام یک از دو نیمه است. سپس این کار را می توانیم در باره بازه ای که در سوال آول به آن راهنمایی شده ایم تکرار می کنیم تا به پاسخ نهایی برسیم. این مثال نشان می دهد که چگونه مسئله های دیگر به مسائل تصمیم گیری قابل تبدیل هستند.

یک مسئله تصمیم گیری را در نظر بگیرید. صورت کلی چنین مسئله ای به این شکل است که اشیایی از یک کلاس معین به ما داده می شود و ما قرار است تشخیص دهیم که آیا اعضای این کلاس خاصیتی مثل P را دارند یا خیر؟

کلاس معین مثلا می تواند مجموعه تمام ماتریس های $N \times N$ باشد که یک مجموعه بی نهایت عضوی است و خاصیت P می تواند این باشد که آیا ماتریس وارون پذیر است یا نه؟

آلگوریتم یا رایانه ای که قرار است این مسئله را حل کند می بایست به ازای یک ورودی معین مثل A هرگاه A وارون پذیر است پاسخ مثبت دهد (مثلا خروجی 1) را نشان دهد در غیر این صورت خروجی 0 را نشان دهد. به این ترتیب ورودی آلگوریتم یا ماشین ما عناصری از یک مجموعه بزرگ و بی نهایت است که ماشین تنها زیرمجموعه خاصی از این ورودی ها را می پذیرد به این معنا که تایید می کند آنها دارای خاصیت P هستند. همانطور که شکل (؟؟) نشان می دهد، ماشین می تواند عناصری از یک مجموعه بی نهایت را دریافت کرده و فقط بعضی از آنها را بپذیرد.

[۷] Decision Problems



شکل ۱: شمای کلی یک ماشین که یک مسئله تصمیم گیری را حل می کند .

مجموعه $\{A_1, A_2, A_3, \cdots\}$ و خاصیت P مجموعاً یک مسئله [۸] را تعریف می کنند و A_i ها نمونه های مسئله [۹] خوانده می شوند. هر مسئله را می توان همواره به صورت مجموعه ای از رشته ها ی $0, 1$ نوشت.

- تمرین: می خواهیم تشخیص دهیم که کدام یک از ماتریس های ۲ در ۲ با درایه های صحیح که هرکدام از درایه ها نیز مثبت بوده از عدد ۱۶ کوچکتر هستند وارون پذیر هستند. اطلاعات این مسئله را به صورت رشته هایی از $0, 1$ نشان دهید.

با کمی تامل و فکر کردن در باره مثال های مختلف می توان هر مسئله ای را به صورت شناسایی یک زیر مجموعه از رشته

[۸] Problem

[۹] Instants of the problem

ها در آورد. درک این موضوع ما را به تعاریف زیر می رساند.

۱.۲ الفبا و زبان

تعریف الفبا و رشته ها : هرمجموعه ای ازعلائم [۱۰] یک الفبا[۱۱] خوانده می شود. یکی از ساده ترین الفبا ها از مجموعه دوحرفی $\Sigma = \{0,1\}$ تشکیل می شود. به هرکدام از اعضای این مجموعه یک حرف و به هردنباله ای از حروف یک رشته[۱۲] یا یک کلمه می گوییم. 000111 و 010101 هردو رشته هایی از الفبای Σ هستند. معمولاً برای نشان دادن رشته ها از حروف $w,\ s$ و نظایر آن استفاده می کنیم و می نویسیم $s = s_1 s_2 s_3 \cdots s_n$ که درآن s_i ها حرف های الفبا هستند. عددِ 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 مجموعه‌ی 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 رشته‌ی ϵ برابر با صفراست. 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 [۱۳] عبارت است از یک زیرمجموعه از Σ^*.

درزیر چند مثال از زبان های تشکیل شده از الفبای Σ آورده شده است:

$$\begin{aligned} L_1 &= \{w \in \Sigma^* | w \quad ends\ with\ 0\} \\ L_2 &= \{w \in \Sigma^* | w \quad contains \quad at \quad least \quad three \quad zeros\} \\ L_3 &= \{w \in \Sigma^* | w \quad ends\ with\ 1010\} \end{aligned} \qquad (۲)$$

یک زبان می تواند یک مجموعه‌ی متناهی یا یک مجموعه‌ی نامتناهی باشد. به عنوان مثال مجموعه‌ی $L_4 = \{00, 11\}$

[۱۰] Symbols

[۱۱] Alphabet

[۱۲] String

[۱۳] A language over Σ

نیز یک زبان روی الفبای Σ است.

مطالب گفته شده تا کنون باید ما را قانع کرده باشد که هر مسئله تصمیم گیری را می توان به صورت شناسایی یک زبان خاص در آورد. بنابر این سوال این است که آیا برای شناسایی یک زبان می توان از یک آلگوریتم و یک ماشین استفاده کرد؟

---

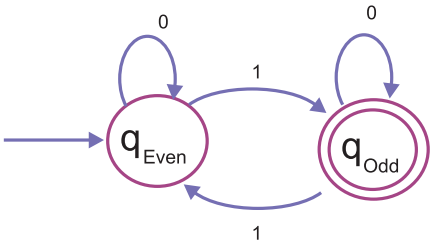
۳ اتومات های متعین و محدود

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 $q_N \cdots, q_2, q_1$ نشان می دهیم. وقتی که این ماشین علامتِ s را دریافت می کند تغییرحالت 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 کننده‌ی 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 ۲ 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 رشته‌ای مثل 0011 به این اتومات داده شود، درحالتِ q_{Even} و هرگاه رشته‌ای مثل 01011011 به آن داده شود در حالت q_{Odd} قرار می گیرد. شکلِ ۲ 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:

$$\begin{aligned} \delta(q_{Even}, 0) &= q_{Even}, \qquad \delta(q_{Even}, 1) = q_{Odd}, \\ \delta(q_{Odd}, 0) &= q_{Odd}, \qquad \delta(q_{Odd}, 1) = q_{Even}. \end{aligned} \tag{۳}$$

این روابط نشان دهنده یک برنامه هستند که اتومات برطبق آنها عمل می کند. این برنامه به سادگی در شکل ۲ که اصطلاحاً دیاگرامِ حالت [۱۴] نامیده می شود نشان داده شده است. می توان گفت که این اتومات رشته‌های با تعدادی فرد از 1 را پذیرفته

[۱۴] State Diagram



شکل ۲: یک اتومات تعینی که رشته های شامل تعدادی فرد از 1 را می پذیرد.

و بقیه رشته ها را ردّ می کند. می گوییم که زبان این ماشین مجوعه تمام رشته هایی است که تعداد فردی عددِ ۱ دارند.
قبل از آنکه مثال های بیشتری از اتومات های محدود ارائه دهیم، بهتراست که تعریف دقیق و رسمی آن را مشخص کنیم.

۱.۳ تعریف دقیق از یک اتومات متعین و محدود

تعریف: یک اتومات تعینی محدود [۱۵] دستگاهی است متشکل از پنج عنصر $DFA = \{Q, \Sigma, \delta, q_0, F\}$ که درآن :

الف : Q یک مجموعه محدود است. $Q = \{q_0, q_1, \cdots q_n\}$. هرعضو Q یک حالت [۱۶] خوانده می شود.

ب : Σ یک الفباست.

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

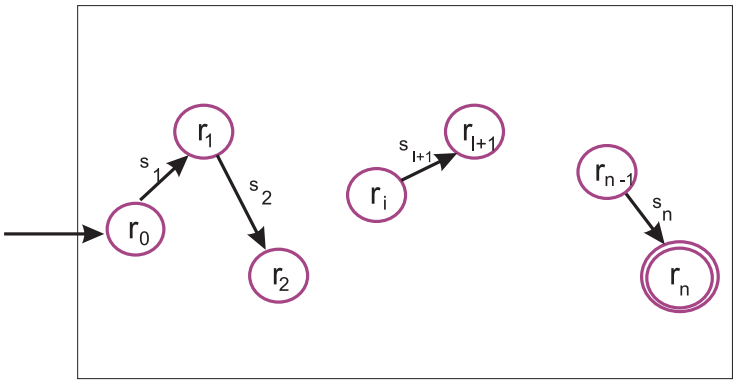
د: q_0 یک حالت ویژه از Q موسوم به حالت آغازین [۱۷] است و

---

[۱۵] Deterministic Finite Automata (DFA)

[۱۶] State

[۱۷] Start State



شکل ۳: پذیرش یک رشته توسط ماشین DFA .

ه :$F \subset Q$ یک زیرمجموعه از حالات است که مجموعه حالات پذیرش[۱۸] نامیده می شود.

هرگاه $w = w_1 w_2 \cdots w_n \in \Sigma^*$ یک رشته باشد می گوییم ماشین M، یا اتومات تعینی ِ 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 دریکی از حالت های پذیرش قرار گیرد. ۳

به بیان دیگر این رشته پذیرفته می شود هرگاه زنجیره ای از حالت های ماشین مثل $r_0, r_1, r_2, \cdots r_n \in Q$ وجود داشته باشند به قسمی که :

۱ ـ $r_0 = q_0$.

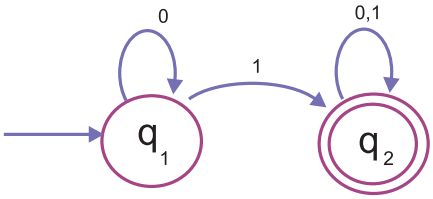
۲ ـ $\delta(r_i, w_i) = r_{i+1}$.

۳ ـ $r_n \in F$.

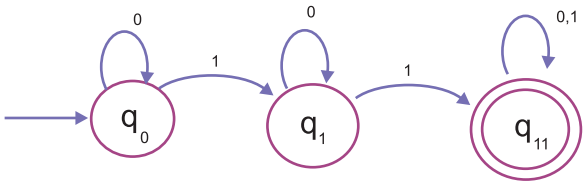
دقت کنید که در مجموعه حالت های r_0 تا r_n ممکن است حالت های تکراری وجود داشته باشد، زیرا تعداد حالت های ماشین محدود است ولی طول رشته های پذیرفته شده ممکن است خیلی بیشتر از تعداد حالت های ماشین باشد.

مجموعه تمام رشته هایی که توسط ماشین پذیرفته می شوند زبان آن ماشین خوانده می شود و با $L(M)$ نمایش داده می

[۱۸] Accept States



شکل ۴: این اتومات تعینی رشته هایی را می پذیرد که حداقل شامل یک نمادِ 1 باشند.



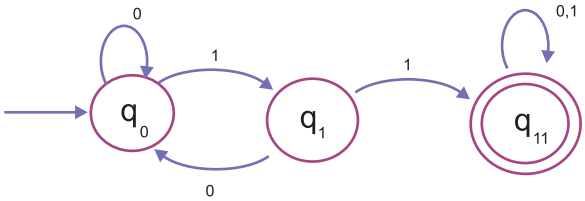
شکل ۵: این اتومات محدود رشته هایی را می پذیرد که حداقل شامل دو نمادِ 1 باشند.

شود. دراین حالت می گوییم که ماشینِ M آن زبان را تشخیص می دهد.

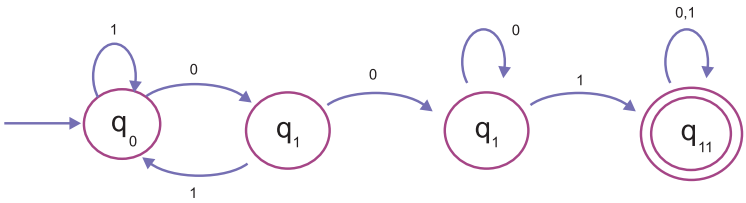
۲.۳ مثالهایی از اتومات های متعین و محدود

در این زیربخش با چند نمونه از اتومات های محدود آشنا می شویم. خواننده برای آنکه با طرز کار این اتومات ها آشنا شود و هم چنین برای اینکه بتواند اتومات هایی را برای زبان های دلخواه طراحی کند می بایست خود را به جای ماشین بگذارد و به خود بگوید که اگر من بخواهم رشته های 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 آلگوریتم وی را به خودی خود به سوی طراحی یک اتومات سوق خواهد داد. این روش یعنی هم ذات پنداری با ماشین در بخش های آینده نیز به خواننده کمک خواهد کرد.

ممکن است برای شناسایی یک زبان اتومات های متفاوتی به کاربرده شود. به عبارت بهتر می توان برای شناسایی یک زبان یا حل یک مسئله آلگوریتم های متفاوتی نوشت. بنابراین بجاست که دراین جاتعریفی از ماشین های معادل بدست دهیم. می گوییم که دو ماشینِ M_1 و M_2 معادل اند اگر هردو یک زبان را تشخیص دهند.



شکل ۶: این اتومات تعینی رشته هایی را می پذیرد که حداقل شامل دو نمادِ 1 به صورت متوالی باشند.



شکل ۷: این اتومات تعینی رشته هایی را می پذیرد که شامل زیر رشته‌ی 001 باشند.

تعریف: دوماشین M_1 و M_2 **معادل اند** اگر $L(M_1) = L(M_2)$.

■ تمرین: در تمام قسمت های زیر الفبای زبان ها عبارت است از $\{a, b\}$. در هر قسمت یک ماشینDFA بسازید که زبان مربوطه را شناسایی کند.

راهنمایی: در هر مورد سعی کنید که خود را به جای ماشین قرار دهید و در نظر بگیرید که اگر بخواهید رشته های گفته شده را شناسایی کنید چکار باید بکنید. هم چنین سعی کنید برای حالت های خود (یا ماشین) اسامی با معنا انتخاب کنید.

$a.$ $\{w \mid w$ w has at least two a's.$\}$

$b.$ $\{w \mid w$ w has exactly two a's $\}$

$c.$ $\{w \mid w$ w has at most two a's. $\}$

$d.$ $\{w \mid w$ w has at least two consecutive a's.$\}$

$e.$ $\{w \mid w$ w has exactly one set of two consecutive a's and at least one set of three consecutive b's .$\}$

$f.$ $\{w \mid w$ w has an even number of a's. $\}$

$g.$ $\{w \mid w$ w has the sub string 'abab' $\{$

$h.$ $\{w \mid w$ w has an even number of substrings 'ab'. $\}$

$k.$ $\{w \mid w$ w starts with a and ends with b.$\}$

$l.$ $\{w \mid w$ w starts with aa and ends with bb. $\}$ (۴)

۴ زبان های منظم

تا کنون یاد گرفته ایم که چگونه هر مسئله تصمیم گیری [۱۹] کلی را می توان به صورت پذیرش جملات یک زبان در آوریم. یک مسئله متناظر با یک زبان خاص و نمونه های آن مسئله [۲۰] متناظر با عناصر موجود در آن زبان هستند. هم چنین آموخته ایم که برای تشخیص یک زبان می توانیم یک اتومات تعینی ساده یا یک ماشین بسازیم . این ماشین در واقع چیزی نیست جز یک رایانه خیلی ساده. وقتی که ماشین زبان را شناسایی می کند معنایش این است که به نمونه های مسئله تعریف شده واکنش نشان می دهد و می گوید که کدام نمونه از از مسئله ها دارای خاصیت معین در نظر گرفته شده هستند یا نیستند. به عنوان مثال مسئله ما ممکن است این باشد که تشخیص دهیم آیا یک ماتریس مربعی با بعد مشخص وارون پذیر هست یا نه. هر ماتریس را می توانیم به صورت رشته ای از یک الفبا معمولا الفبای $\{0,1\}$ در آوریم. در این صورت زبان ما صورت کلی مسئله و عناصر این زبان تمام ماتریس های با آن بعد مشخص هستند. ماشینی که این زبان را شناسایی می کند کارش این است که ماتریس های وارون پذیر را قبول می کند و ماتریس های غیروارون پذیر را رد می کند.

به نظر می رسد که با طرح ماشین های پیچیده تر می توانیم مسائل پیچیده تر را نیز حل کنیم. کافی است که تعداد حالت های ماشین را زیاد کرده و توابع انتقال پیچیده تری نیز تعریف کنیم. می توانیم ماشینی را طراحی کنیم با هزاران حالت که یک مسئله خیلی پیچیده را حل کند. می توانیم به طور سیستماتیک تری به این مسئله فکر کنیم. نخست بیایید در باره پیچیده تر کردن زبان ها فکر کنیم و سعی کنیم که زبان ها (یا مسئله ها) را به روش های گوناگون باهم ترکیب کنیم.

[۱۹] decision problem

[۲۰] Instants of the problem

۱.۴ اعمال روی زبان ها

■ **تعریف اعمال مقدماتی روی زبان ها:** فرض کنید که A و B دو زبان باشند. دراین صورت اجتماع[۲۱] این دوزبان $A \cup B$ و ترکیب [۲۲] این دو زبان $A \circ B$ به صورت زیر تعریف می شوند:

$$\begin{aligned} A \cup B &= \{x \in \Sigma^*, \ \ x \in A \ \ or \ \ x \in B\} \\ A \circ B &= \{xy \in \Sigma^*, \ \ x \in A \ \ and \ \ y \in B\} \end{aligned} \tag{۵}$$

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 رشته‌هایی که از پشت سرهم نهادن کلمات اولی و دومی بدست می آید. دقت کنید که لزومی ندارد که هردو زبان روی یک الفبا تعریف شده باشند.

مثال: هرگاه داشته باشیم

$$\begin{aligned} A &= \{aa, bb, cc\} \\ B &= \{00, 11\}, \end{aligned} \tag{۶}$$

آنگاه

$$\begin{aligned} A \cup B &= \{aa, bb, cc, 00, 11\} \\ A \circ B &= \{aa00, aa11, bb00, bb11, cc00, cc11\} \\ B \circ A &= \{00aa, 11aa, 00bb, 11bb, 00cc, 11cc\}. \end{aligned} \tag{۷}$$

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

$$L^* = \{x_1 x_2 \cdots x_n | n \geq 0 \ \ and \ \ x_i \in A\}. \tag{۸}$$

---

[۲۱] Union

[۲۲] Concatenation

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

$$A* = \{\epsilon, aa, bb, cc, aaaa, bbbb, cccc, aabb, aacc, bbcc, bbaa, ccaa, aaaaaa, \cdots\}. \quad (۹)$$

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

- می توانیم اشتراک دو زبان را نیز تعریف کنیم

$$A \cap B = \{w \mid w \in A, \quad and \quad w \in B\}. \quad (۱۰)$$

- تعریف: هرگاه L یک زبان باشد ، متتم آن را با $\tilde{L}$ نمایش می دهیم . بنابر تعریف

$$\tilde{L} := \{w \in \sum{}^* \mid w \notin L\} \quad (۱۱)$$

یعنی متمم یک زبان شامل تمام رشته هایی است که در آن زبان نیستند.

- مثال : ۱ـ هرگاه L زبانی باشد که رشته های آن طول زوج دارند $\tilde{L}$ زبانی است که رشته های آن طول فرد دارند.
۲ ـ هرگاه هرگاه L زبانی باشد که رشته های حد اقل دارای یک زیر رشته 001100 باشند، $\tilde{L}$ زبانی است که رشته های آن شامل زیر رشته بالا نیستند.

۲ ـ هرگاه هرگاه L زبانی باشد که حرف آغاز و پایان رشته های آن با هم مساوی باشند، $\tilde{L}$ زبانی است که رشته های آن یا با صفر شروع شده و با صفر ختم می شوند، یا با یک شروع شده و با یک نیز ختم می شوند.

- با چند مثال ساده خود را قانع کنید که هرگاه در یک ماشین DFA جای حالت های معمولی و حالت های $Accept$ یک ماشین را عوض کنید، ماشینی بدست می آید که زبان متتم ماشین قبلی را شناسایی کند.

- تمرین: هرکدام از زبان های زیر متمم یک زبان ساده است. نخست ماشینی بسازید که آن زبان ساده را شناسایی کند و سپس با استفاده از آن ماشینی بسازید که زبان متمم را شناسایی کند.

$a.$ $\{w \mid w$ w does not contain the substring ab.$\}$

$b.$ $\{w \mid w$ length(w) is not a multiple of 3. $\}$

$c.$ $\{w \mid w$ w does not start with aa.$\}$

$d.$ $\{w \mid w$ w does not end with bb .$\}$

$e.$ $\{w \mid w$ w does not contain consecutive b's. .$\}$ (۱۲)

■ تمرین: ماشین های DFA ای بسازید که زبان های زیر را شناسایی کند. در هر مورد الفبا عبارت است ا ز $\sum = \{0, 1\}$.

$a.$ $\{w \mid w$ every odd position of w is 1.$\}$

$b.$ { The empty set. }

$c.$ { All strings except the empty string. } (۱۳)

دقت کنید که با ترکیب زبان ها می توانیم زبان های خیلی پیچیده بسازیم. حال ببینیم آیا ماشین هایی وجود دارند که این زبان های ترکیبی را بسازند. برای پاسخ به این سوال نخست به یک تعریف احتیاج داریم.

■ تعریف: زبان منظم [۲۳] به زبانی گفته می شود که توسط یک اتومات محدود قابل شناسایی باشد. به عبارت بهتر تناظر یک به یک بین زبان های منظم و اتومات های محدود وجود دارد.

تاکنون با زبان های منظم و هم چنین اتومات های بسیار ساده آشنا شدیم. ولی بسته به تعداد حالت های یک اتومات و هم چنین نوع توابع آن یک اتومات و در نتیجه زبان شناسایی شده توسط آن می 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، اشتراک، متتم گیری، چسباندن و ستاره یک مجموعه بسته است.

[۲۳] Regular Language

با دانش فعلی مان می توانیم بسته بودن زبان های منظم تحت عمل اجتماع و اشتراک را اثبات کنیم. اثبات بسته بودن تحت عمل متمم گیری را قبلا (در قالب یک تمرین انجام داده ایم). ولی برای اثبات بسته بودن این زبان ها تحت دو عمل دیگر بهتراست صبر کنیم تا اتومات های نامتعین را بفهمیم. درآن موقع اثبات این دو قسمت بسیار آسان خواهد بود. روشی که ما برای اثبات قسمت اول و دوم این قضیه به کار می بریم خود به خود ما را به ابداع مفهوم جدیدی از اتومات رهنمون خواهد شد.

■ **اثبات بسته بودن زبان ها تحت عمل اجتماع :** فرض کنید که ماشینِ $M_1 = (\Sigma, Q_1, q_1, \delta_1, F_1)$ زبانِ A و ماشینِ $M_2 = (\Sigma, Q_2, q_2, \delta_2, F_2)$ زبانِ B را شناسایی می کند. می خواهیم ماشینی بسازیم که زبان $A \cup B$ را شناسایی کند. می دانیم که هر رشته‌ی متعلق به $A \cup B$ یا متعلق به A است یا متعلق به B. اگر مطابق با فلسفه‌ای که برای فهم کارکرد ماشین ها و طراحی آنها به کار برده‌ایم عمل کنیم، راه بدیهی برای آنکه تشخیص دهیم رشته‌ای متعلق به $A \cup B$ هست یا نه آن است که هر رشته‌ای که به دستمان می رسد، یک نسخه‌اش را به ماشینِ M_1 بدهیم و یک نسخه‌اش را به ماشینِ M_2. اگر این رشته متعلق به A باشد، ماشینِ 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 هرکدام که به یک حالت پذیرش رسید از روی آن می فهمیم که رشته‌ی مورد نظر متعلق به زبانِ $A \cup B$ بوده است و اگر هیچ کدام به حالت پذیرش نرسیدند نتیجه می گیریم که رشته مورد نظر متعلق به زبانِ $A \cup B$ نبوده‌است. با این توصیف ماشینی که زبانِ $A \cup B$ را شناسایی می کند عبارت است از:

$$M = (\Sigma, Q_1 \times Q_2, (q_1, q_2), \delta, F) \qquad (۱۴)$$

که درآن

$$\delta : \Sigma \times Q_1 \times Q_2 \longrightarrow Q_1 \times Q_2, \qquad \delta(s, (q_1, q_2)) = (\delta_1(s, q_1), \delta_2(s, q_2)). \qquad (۱۵)$$

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 دقیقِ 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 دیگر 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 $A \cup B$ بوده‌است. بنابراین حالت های پذیرش نهایی عبارتند از:

$$F = (F_1 \times Q_2) \cup (Q_1 \times F_2). \quad (۱۶)$$

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

- اثبات بسته بودن زبان های منظم تحت عمل اشتراک: هرگاه M_A یک ماشین باشد DFA که زبان A را شناسایی کند و M_B یک ماشین DFA باشد که زبان B را شناسایی کند، همواره می توان یک ماشین تعینی ساخت که زبان $A \cap B$ را شناسایی کند.

اثبات این امر خیلی ساده است: فرض کنید

$$M_A = \{\Sigma,\ q_a,\ Q_a,\ \delta_a,\ F_a\} \quad (۱۷)$$

و

$$M_B = \{\Sigma,\ q_b,\ Q_b,\ \delta_b,\ F_b\}. \quad (۱۸)$$

در این صورت ماشین زیر زبان $A \cap B$ را شناسایی می کند.

$$M_{A\cap B} = \{\Sigma,\ (q_a, q_b)\ Q_a \times Q_b,\ \delta,\ F_a \times F_b\} \quad (۱۹)$$

که در آن

$$\delta((q_1, q_2)\ ,\ s) := (\delta_a(q_1, s)\ , \delta_b(q_2, s)) \quad (۲۰)$$

براحتی و یا با مطالعه چند مثال می توانید درستی این ادعا را ثابت کنید.

به این ترتیب ثابت کرده ایم که اشتراک دو زبان منظم نیز یک زبان منظم است.

۵ عدم تعین و اتومات های نامتعین و محدود

اگر بخواهیم مسئله پیچیده ای حل کنیم یک راه آن است که به جای یک رایانه از دو یا چند رایانه کمک بگیریم که به طور همزمان کار می کنند و به جستجوی حل مسئله مورد نظر ما می پردازند. این کاری است که امروزه خیلی مرسوم است. اتومات های نامتعین و محدود [۲۴] در واقع مدلی نظری برای کار همزمان چند رایانه با هم هستند. البته این مدل نظری توانایی اش خیلی بیشتر از چند رایانه است که از ابتدا به طور همزمان با هم کار می کنند چرا که دو ویژگی خیلی مهم دارند. نخست آنکه در این مدل نظری هرگاه که لازم باشد با توجه به حالت کنونی ماشین و ورودی ای که در آن لحظه ماشین می گیرد، کپی های جدیدی از ماشین شروع به کار می کنند و این ماشین ها هرکدام راه خود را طی می کنند. دوم این که هیچ محدودیتی برای تعداد کپی ها وجود ندارد و با گذشت زمان تعداد کپی های ماشین که مشغول کار هستند به طور نمایی نیز زیاد شود. در واقع این کار موازی منشاء توان خیلی زیاد این ماشین هاست.

در واقع روشی که برای اثبات قضیه پیشین به کار بردیم از نمونه خیلی ساده ای از این موازی سازی استفاده کرد. 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 مرحله‌ای از کار خود می تواند نمونه‌هایی 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 قواعد یک ماشین نامتعین مطابق شکلِ ۸ هستند. این قواعد ویژگی های زیر را دارند:

اول: همانطور که شکل A نشان می دهد، در بعضی از حالت ها به ازای یک حرف الفبا ممکن است که ماشین به چند تا حالت مختلف برود. این قاعده به این معناست که از آن به سه کپی از از ماشین که در حالت های $\{q_3, q_5, q_7\}$ هستند ساخته می شوند و شروع به کار می کنند. به این ترتیب برخلاف ماشین تعینی ماشین غیر تعینی در هر لحظه ممکن است در بیش از

[۲۴] Non-deterministic Finite Automata (NFA)

یک حالت باشد. بنابر شکل A مثل این است که در این لحظه سه نسخه یکسان از ماشین اولیه در اختیار داریم و یکی از آنها در حالت q_3 ، دیگری در حالت q_5 و سومی در حالت q_7 است و هر سه ماشین محاسبه را مطابق با دیاگرام نشان داده شده دنبال می کنند. به این ترتیب توانایی این ماشین برای محاسبه بیشتر شده است زیرا درست مثل این است که به جای یک ماشین داریم با سه ماشین محاسبه می کنیم. این پردازش موازی یا شاخه شاخه شدن می تواند بازهم تکرار شود.

وقتی که از یک حالت یک فلش خارج می شود، به این معناست که حالت بعدی ماشین به صورت یکتا تعیین می شود. اگر ماشین در این حالت باشد، با گرفتن ورودی مورد نظر حالت بعدی ماشین به صورت یکتا تعیین می شود.

دوم: وقتی که از یک حالت ماشین به ازای یک حرف الفبا مثلا 1 هیچ فلشی خارج نشود به این معناست که وقتی که ماشین در آن حالت است ، اگر ورودی 1 را دریافت کند، ماشین در آن شاخه متوقف می شود. این وضعیت در شکل B نشان داده شده است.

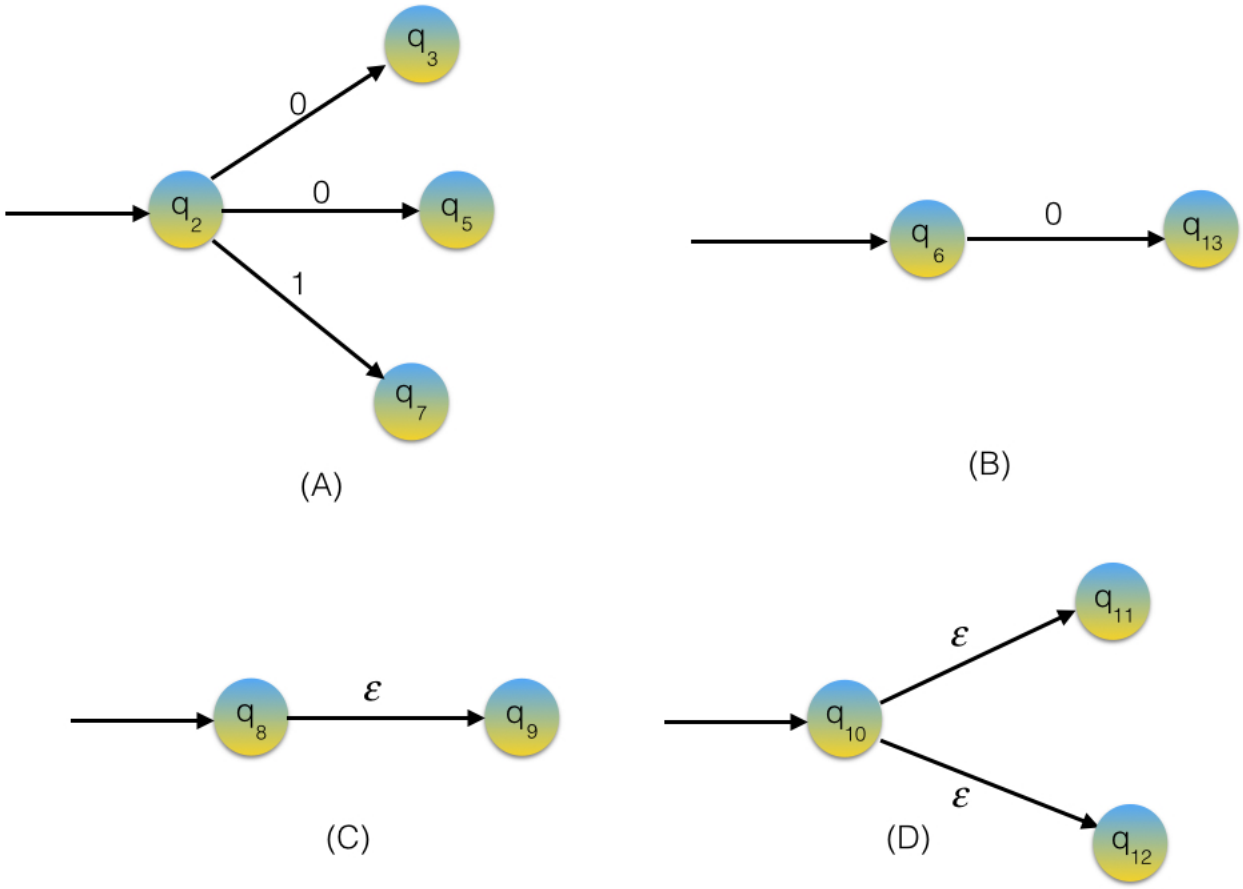
سوم: وقتی که یک شاخه از ماشین به حالتی می رسد که به ازای یک ورودی خاص به هیچ حالتی نمی رود، آن کپی از ماشین دیگر متوقف می شود و کار خود را پایان می دهد. البته کپی های دیگر کارشان را ادامه می دهند. این وضعیت برای حالت q_{11} رخ می دهد وقتی که ورودی 1 را دریافت کند.

و بالاخره یک ویژگی مهم این ماشین ها این است که به طور خود به خود نیز شاخه شاخه می شوند. این کیفیت در شکل () وقتی که ماشین در حالت q_{10} است نشان داده شده است. علامت ϵ به این معنای هیچ است یعنی این که ماشین به طور خود به خود سه شاخه شده است و دو شاخه آن در حالت های q_{13} و q_{14} به کار خود ادامه می دهند و شاخه فعلی نیز در همان حالت قبلی یعنی حالت q_{10} به کار خود ادامه می دهد.

بعد از این توضیحات آماده ایم که ماشین نامتعین را به صورت دقیق و فرمال تعریف کنیم:

تعریف: یک اتومات محدود و نامتعین [۲۵] دستگاهی است متشکل از پنج عنصر $NFA = \{Q, \Sigma_\epsilon, \delta, q_0, F\}$ که درآن :

[۲۵] Non-deterministic Finite Automata (NFA)



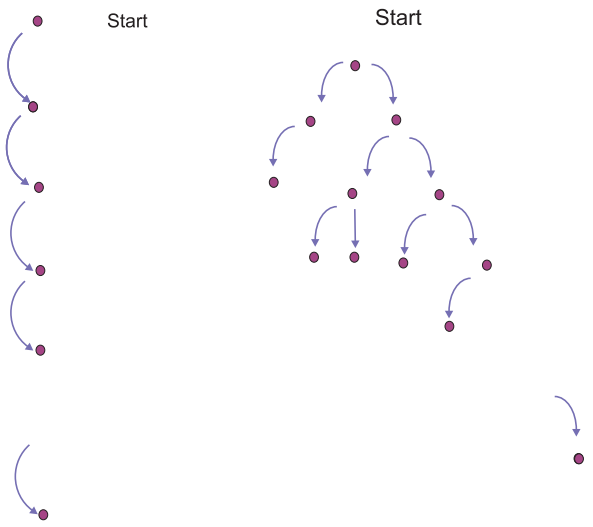
شکل ۸: قواعد کار برای یک ماشین نامتعین.

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

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 هرحالت و به هرحرف از Σ_ϵ یک زیر مجموعه از حالت های Q را نسبت می دهد.

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

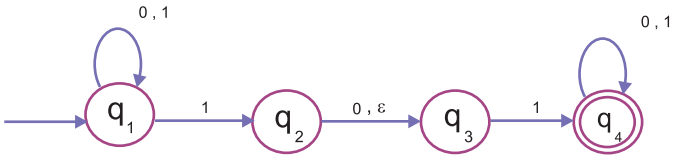


شکل ۹: در یک اتومات نامعین، یک رشته مسیرهای متعددی را طی می کند و هر مسیری که به حالت قبولی برسد آن رشته پذیرفته می شود. اگر هیچ مسیری به یک حالت قبولی نرسد، آنگاه رشته مورد نظر پذیرفته نخواهد شد.

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

دراین تعریف منظور از $P(Q)$ مجموعه تمام زیرمجموعه‌های Q است.

دقت کنید که چند تفاوت عمده بین تابع انتقال یک ماشین DFA و یک ماشین NFA وجود دارد. اولا تابع δ حالتِ 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 مجموعه‌ی تهی باشد. هرگاه که این خروجی یک مجموعه تک عضوی باشد، عمل ماشین در این مرحله کاملا تعینی است و هرگاه خروجی ماشین یک مجموعه چند عضوی باشد، عمل ماشین در این مرحله نامتعین است به این معنی که ماشین دراین مرحله همه مسیرهای ممکن را همزمان طی می کند و بالاخره هرگاه که خروجی تابع مجموعه‌ی تهی باشد به این معناست که عمل ماشین در این شاخه متوقف می شود و به اصطلاح ماشین دراین شاخه می میرد. بنابراین در دیاگرام حالتِ یک ماشین نامتعین از هر دایره که نشان دهنده‌ی 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 محاسبه‌ای که در بالا شرح دادیم هرگاه به صورت دقیق درآید به شکل زیر



شکل ۱۰: این اتومات نامعین رشته هایی را می پذیرد که یا شامل زیر رشته های 11 باشند یا شامل رشته های 101.

خواهد بود:

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

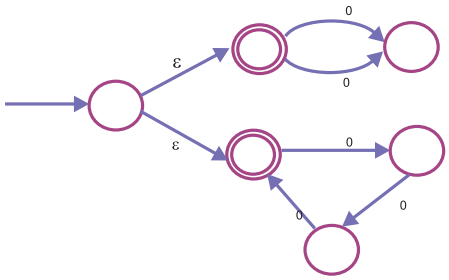
۱ ـ $r_0 = q_0$.

۲ ـ $r_{i+1} \in \delta(r_i, w_i)$.

۳ ـ $r_n \in F$.

شکل های ۱۰ و ۱۱ نمونه‌هایی از ماشین های نامتعین و محدود هستند.

■ تمرین: ماشین شکل ۱۰ را در نظر بگیرید. تعیین کنید که مسیری که ماشین می پیماید (یعنی رشته حالت هایی را که در آنها قرار می گیرد) وقتی که هرکدام از رشته های زیر به آن داده می شود چه هستند. دقت کنید که ممکن است ماشین در هر لحظه در بیش از یک حالت باشد.



شکل ۱۱: این اتومات نامعین رشته هایی را می پذیرد که فقط شامل مضربهای ۲ یا ۳ از نماد 0 باشند.

010

1000

11

101

111

100011 (۲۱)

■ تمرین: در هرکدام از موارد زیر یک ماشین NFA با تعداد حالت های خواسته شده بسازید که زبان مورد نظر را شناسایی کند: در تمام موارد نیز الفبا عبارت است از $\Sigma = \{0, 1\}$.

A := $\{w$ w ends with 11$\}$ with three states

B := $0^*1^*0^*$ with three states

C := $\{\epsilon\}$ with one state

D := 0^* with one states

E := $\{w$ |w contains the substring 0101$\}$ with five state.

$$F := \{w \mid \text{w contains either 000 or 111}\} \quad \text{with the least possible number of states} \qquad (۲۲)$$

اگر چه ظاهر امر آن است که ماشین های نامتعین کارایی بیشتری نسبت به ماشین های متعین دارند، ولی می توان ثابت کرد که این دو نوع ماشین در واقع یک نوع زبان را تشخیص می دهند به این معنا که هر ماشین نامتعین را می توان با یک ماشین متعین شبیه سازی کرد. البته ممکن است که برای شبیه سازی یک ماشین نامتعین که تعداد خیلی کمی حالت دارد مجبور شویم که ماشین متعینی با تعداد بسیارزیادی حالت طراحی کنیم. این امر ضمن آنکه نشان دهنده سادگی و کارآیی ماشین های NFA است نشان می دهد که ماشین های NFA و DFA با هم معادل اند.

■ **قضیه:** ماشین های NFA با ماشین های DFA معادلند به این معنا که هرماشین NFA قابل شبیه سازی توسط یک ماشینِ DFA است.

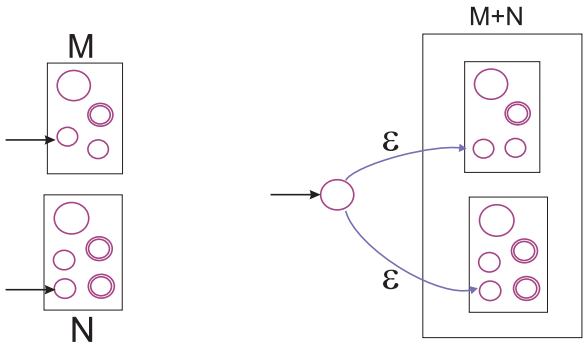
■ **اثبات:** فرض کنید که $M = (\Sigma, Q, q_1, \delta, F)$ یک NFA باشد. دراین صورت ماشین تعینی M' را به شکل زیر می سازیم:

$$M' := (\Sigma \cup \epsilon, P(Q), \{q_1\}, \delta', F), \qquad (۲۳)$$

که درآن $P(Q)$ مجموعه‌ی همه زیرمجموعه‌های Q است. کافی است که بگوییم این ماشین تعینی چه تابع انتقالی دارد. بازهم دراین جا دریافت شهودی خود را از نحوه محاسبه‌ی ماشین نامتعین به یک عبارت ریاضی دقیق ترجمه می کنیم. می دانیم که ماشین نامتعین در هرلحظه به چند نمونه از خود تکثیر شده است و همه شاخه ها با هم در حال کارند. بنابراین هرگاه بخواهیم معادل این ماشین نامتعین 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

$$\delta'(s, \{q_i, q_j, q_k \cdots\}) = \delta(s, q_i) \cup \delta(s, q_j) \cup \delta(s, q_k) \cdots . \qquad (۲۴)$$

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

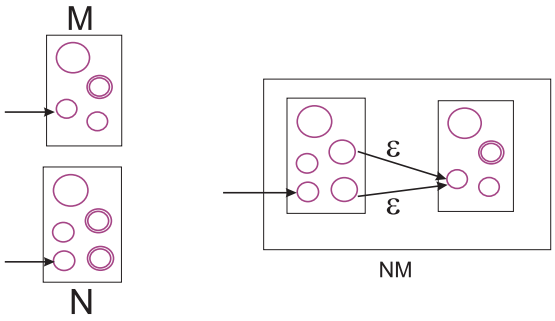


شکل ۱۲: نحوه ساختن ماشینی که زبان $A \cup B$ را شناسایی کند.

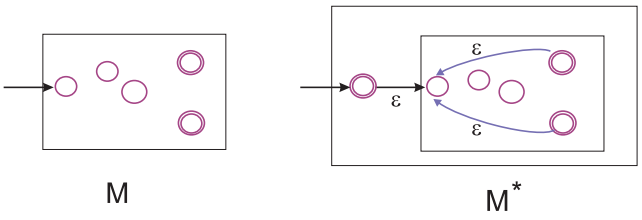
با استفاده از این قضیه می توانیم بسته بودن زبان های منظم تحت اعمال منظم را به آسانی نشان دهیم. برای این کار کافی است که نشان دهیم هرگاه دو زبان A و B منظم باشند، آنگاه ماشین های نامتعینی وجود دارند که بتوانند زبان های $A \cup B$، $A \circ B$ و A^* را شناسایی کنند. از آنجا که این ماشین های نامتعین را همواره می توان به صورت ماشین های متعین درآورد، 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. دراین صورت حتماً ماشین هایی مثل N و M وجود دارند که این زبان ها را شناسایی کنند. شکل های ۱۲، ۱۳ و ۱۴ 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 امر اثبات قضیه را کامل می کند.



شکل ۱۳: نحوه ساختن ماشینی که زبان $B \circ A$ را شناسایی کند.

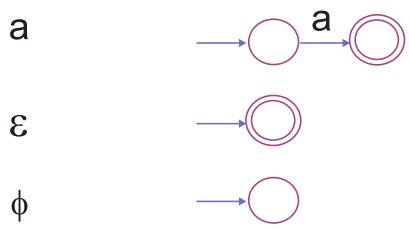


شکل ۱۴: نحوه ساختن ماشینی که زبان A^* را شناسایی کند.

۶ عبارت های منظم

تا کنون زبان های منظم و اعمال آنها را معرفی کرده‌ایم. دراین بخش می خواهیم روشی را یاد بگیریم که با استفاده از آن یک زبان منظم را به صورت کوتاه معرفی کنیم. یک زبانِ A را با یک عبارت کوتاهِ R نشان می دهیم. R را یک عبارت منظم [۲۶] می نامند که نشان دهنده آن زبان است. به عنوان مثال به ازای الفبای $\Sigma = \{a, b\}$، عبارت منظمِ 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 $\Sigma^* b$ نشان دهنده‌ی 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

[۲۶] Regular Expression

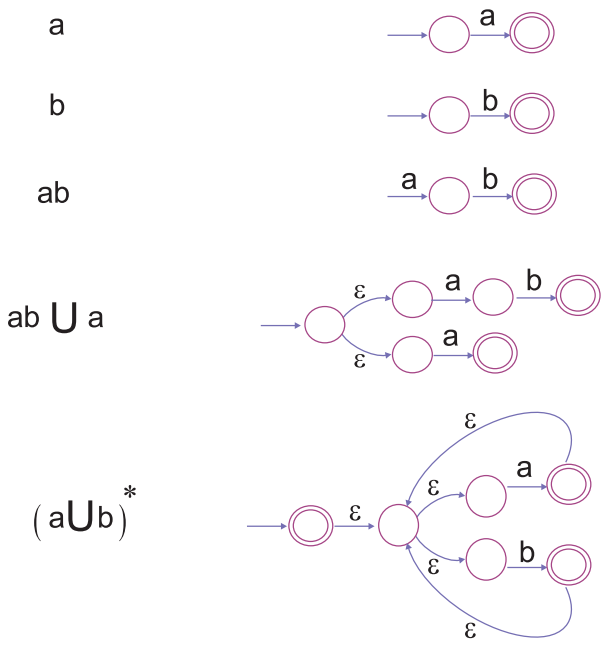


شکل ۱۵: عناصراصلی ای برای تناظر عبارت های منظم و اتومات های محدود.

روبروی هر عبارت منظم نوشته شده است زبان منظم متناظر با آن عبارت را نشان می دهد.

$$
\begin{aligned}
\Sigma^*\Sigma^* &= \textit{all strings of even length} \\
a\Sigma^*b &= \textit{all strings begining with } a \textit{ and ending with } b \\
(0U1)0^* &= \textit{all strings which are purely made of } 0 \textit{ or have only one single } 1 \textit{ at their left} \\
\Sigma^*1\Sigma^* &= \textit{all strings which have at least a } 1 \\
a^*b^* &= \textit{all strings in which all } a's \textit{ precede } b's, \\
a^*b^* \cup b^*a^* &= \textit{all strings in which either all } a's \textit{ precede all } b's \textit{ or vice versa}, \\
\Sigma\Sigma^*\Sigma^* &= \textit{all strings whose length is odd .}
\end{aligned}
\tag{۲۵}
$$

خواننده می تواند بیشمار 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 برقراراست 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 ِ ۱۵ 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

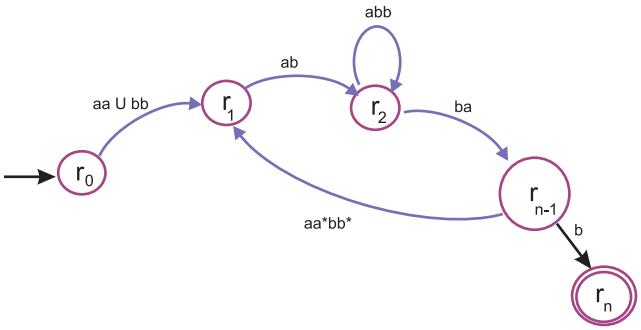


شکل ۱۶: مثال هایی از عبارت های منظم و اتومات های متناظر با آنها.

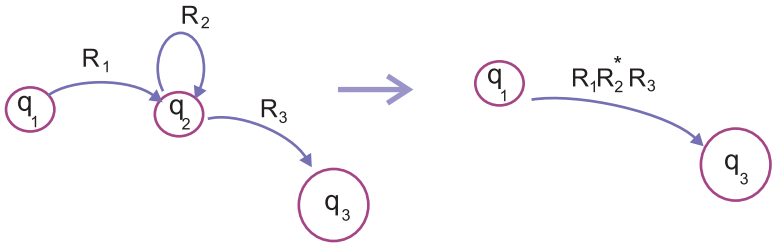
منظم $\cup, \circ,$ و $*$ می توانیم ماشین های DFA متناظر با هر عبارت منظمی را بسازیم. شکل ۱۶ نمونه هایی از این نحوه ساخت را نشان می دهد.

بنابراین نشان داده‌ایم که به ازای هر عبارت منظم یک ماشین DFA وجود دارد. بالعکس می توانیم نشان دهیم که به ازای هر ماشینِ DFA می توانیم یک عبارت منظم بسازیم. تفصیل این نحوه ساخت در مرجع شماره یک آمده است. دراین جا فقط به ذکر ایده کلی آن می پردازیم. کلید اصلی ماشین تعمیم یافته‌ی نامتعین یا $GNFA$ [۲۷] است. یک ماشینِ $GNFA$ درست مثل یک DFA عمل می کند با این تفاوت اصلی که در دیاگرام حالتِ آن بجای حروف الفبا عبارت های منظم نوشته می شود. شکل ۱۷ یک ماشینِ $GNFA$ را نشان می دهد. معنای این شکل آن است که هر عبارت از نوع aa یا bb ماشین را از حالتِ r_0 به حالت r_1 می برد و یا این که هر عبارتِ از نوع aa^*bb^* مثل $ab, aabb, aaaaabbbb, abbbb$ و نظایر آنها ماشین را از حالتِ 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

[۲۷] Generalized Non-deterministic Finite Automata



شکل ۱۷: نمونه‌ای از یک اتومات تعمیم یافته نامتعین $GNFA$.



شکل ۱۸: روشی که تعداد حالت های یک $GNFA$ را تقلیل داده و عبارت منظم متناظر با آن را پیدا می کنیم.

$GNFA$ 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 عبارتی است که ماشین DFA اولیه را بیان می کند. این فرایند در شکل ۱۸ نشان داده شده است.

به این ترتیب نشان داده‌ایم که بین زبان های منظم، عبارت های منظم و ماشین های DFA تناظر یک به یک وجود دارد.

ماشین های نامتعین نشان داده شده در شکل (۱۹) را به ماشین های تعینی تبدیل کنید.

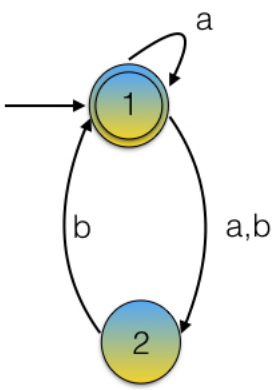
■ زبان های زیر را در نظر بگیرید. در هر مورد عبارت منظمی که آن زبان را توصیف می کند بنویسید:

a. $\{w \mid w$ w has at least two a's and at least three b's.$\}$

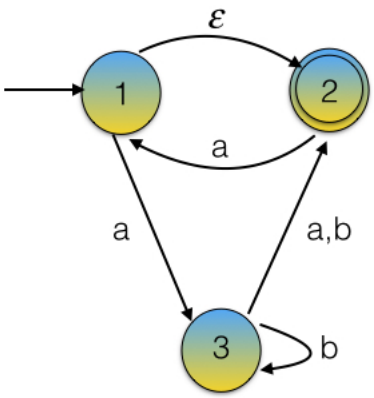
b. $\{w \mid w$ w has exactly two a's and at least three b's.$\}$

c. $\{w \mid w$ w has exactly two two a's and exactly two b's.$\}$

d. $\{w \mid w$ w has at least two consecutive a's and at least three consecutive b's .$\}$



(A)



(B)

شکل ۱۹: مربوط به تمرین ها: این دو ماشین نامتعین را به ماشین های متعین تبدیل کنید.

$e.$ $\{w \mid w$ w has an even number of a's and each a is followed by at least one b .$\}$

$f.$ $\{w \mid w$ w has an even number of a's and each a is followed by exactly one b.$\}$

$g.$ $\{w \mid w$ w starts with a and ends with b.$\{$

$h.$ $\{w \mid w$ w starts with aaa and ends with bbb.$\}$

$k.$ $\{w \mid w$ w starts with aa and has at least one b.$\}$

$l.$ $\{w \mid w$ w has even length and has an odd number of b's. $\}$ (۲۶)

■ عبارت های منظم زیر را به ماشین های نامتعین تبدیل کنید:

a^*b^*

$\Sigma^*aa\Sigma^*bb$

$a^* \cup b^*$

$(aaa)^* \cup (bbb)^*$

$(aUab)\Sigma^*$

$$(\epsilon U a)b^*$$
$$(a \cup b)^* a^* (a \cup b)^*$$
$$a^* b^* \cup b^* a^*. \qquad (۲۷)$$

۷ محدودیت های اتومات های محدود و زبان های منظم

رابطه‌ای که بین عبارت های منظم و زبان های منظم و هم چنین ماشین های تعینی محدود وجود دارد نشان می دهد که این ماشین ها می توانند گستره وسیعی از زبان ها را شناسایی کنند. بااین 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 آلگوریتم بنویسیم. محدودیت عمده‌ای که ماشین های تعینی دارند آن است که حافظه ندارند یا به عبارت دیگر حافظه محدودی دارند. این حافظه محدود همانی است که در تعدادِ حالت های آنها وجود دارد. بنابراین هرگاه که زبانی داشته باشیم که برای شناسایی آن احتیاج به حافظه نامحدودی داشته باشیم یک ماشین تعینی محدود قادر به شناسایی آن زبان نخواهد بود. به عنوان مثال زبان زیر را در نظر بگیرید:

$$A = \{0^n 1^n \mid n > 0 \}. \qquad (۲۸)$$

برای شناسایی رشته های مربوط به این زبان، یک ماشین باید بتواند نخست تعدادِ 0 را بشمارد و در جایی 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 درغیر این صورت آن را رد می کند. یک ماشین تعینی چنین امکانی ندارد. هم چنین زبان زیر توسط یک ماشین تعینی محدود قابل شناسایی نیست:

$$B = \{w \in \Sigma^* \mid w \; contains \; equal \; number \; of \; 0's \; 1's\}. \qquad (۲۹)$$

چنین زبان هایی را زبان های نامنظم [۲۸] می گویند. یک نمونه دیگر از مسئله‌ای که یک ماشین تعینی محدود قادر به حل

[۲۸] Non-Regular

کردن آن نیست مسئله‌ی تست کردن پرانتز هاست [۲۹]. فرض کنید که عبارتی مثل ((()())()) داریم. در این عبارت ساختار پرانتزها صحیح است ولی عبارتی مثل ()(()()) ساختار پرانتزی صحیحی ندارد. دقت کنید که کافی نیست که ماشین تنها تعداد پرانتز های بسته و باز را بشمارد، زیرا که ترتیب باز و بسته شدن پرانتز ها نیز مهم است. بنابراین مسئله تستِ پرانتزها با شناسایی زبان B متفاوت است.

برای آنکه تشخیص دهیم که آیا یک زبان منظم است یا نا منظم یک لمِ بسیار مفید و قدرتمند وجود دارد که به لمِ پُمپ کردن [۳۰] مشهوراست.

- **لِم پمپ کردن:** اگر A یک زبان منظم باشد، آن گاه یک عددِ مثبت p موسوم به طول پمپ کردن [۳۱] 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 قسمتِ xyz *چنان* تقسیم کرد، که سه شرط زیر تواماً برآورده شوند:

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

  ب: $|y| > 0$،

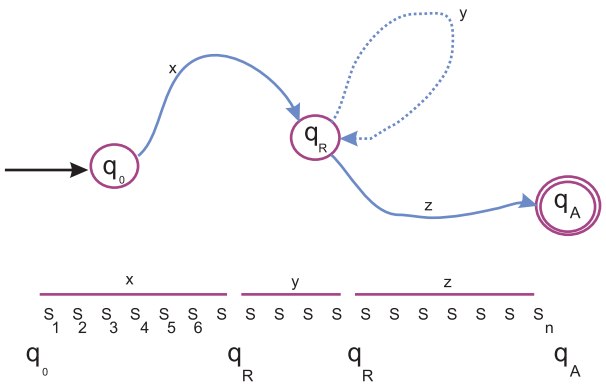
  ج: $|xy| < p$.

  دقت کنید که در صورت قضیه قیدِ *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.

[۲۹] Paranthesis Chekcer Problem

[۳۰] Pumping Lemma

[۳۱] Pumping Length



شکل ۲۰: لم پمپینگ. اگر این ماشین رشته xyz را بپذیرد، حتماً رشته های xy^iz را هم می پذیرد. ضمناً $|y| \geq 0$ و $|xy| \leq p$.

روش استفاده از لم پمپ کردن: هرگاه زبان A چنان باشد که به ازای هر طول پمپ کردن، و هرنوع 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 لمِ پمپ 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. دراین صورت هرگاه رشته‌ای مثل $s = s_1 s_2 s_3 \cdots s_n, \quad n > p$ داشته باشیم که توسط ماشین پذیرفته شود، این رشته می بایست ماشین را مطابق با طرح زیر از حالتِ q_0 به حالت q_{accept} ببرد:

$$\begin{array}{ccccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & \cdots & \cdots & s_n & \\ q_0 & q_{i_1} & q_{i_2} & q_{i_3} & q_{i_5} & \cdots & \cdots & q_{accept}. & \end{array} \qquad (۳۰)$$

می دانیم که تعداد حالت های q_i ، p تاست که تعداد آنها از n کمتراست. 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 شدنِ حالت q_R را در رشته‌ی حالت ها در نظر می گیریم و مطابق با شکل ۲۰ ، 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 ۲۰ واضح است که هر رشته‌ای مثل xy^iz نیز متعلق به زبانِ A است.

دقت کنید که چون ابتدا و انتهای زیر رشته‌ی y محل اولین و دومین ظاهرشدن حالتِ 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 $|xy| \leq p$.

با استفاده از لم پمپ کردن می توان براحتی فهمید که یک زبان نامنظم است. به عنوان مثال زبان A در ۲۸ را در نظر می گیریم. فرض کنید که p طول پمپینگ باشد. باید نشان دهیم که رشته‌ای وجود دارد با طول بیشتر از p که هر طور آن را به سه قسمتِ xyz تقسیم کنیم که در شرایطِ $|y| > 0$ و $|xy| \leq p$ صدق کند، بازهم بعضی از رشته های xy^iz خارج از زبانِ A خواهند افتاد. رشته‌ای مثل 0^p1^p را در نظربگیرید. هر طور تقسیمی که بخواهد شرطِ $|xy| \leq p$ را ارضا کند می بایست چنان باشد که x و y تماماً از صفر ها تشکیل شده باشد. بنابراین واضح است که پمپ کردن این رشته یعنی تکرارِ y باعث افزایش 0 ها خواهد شد بدون اینکه تعدادِ 1 ها افزایش یابد. در نتیجه رشته های xy^iz خارج از زبان خواهند افتاد. بنابراین زبانِ A یک زبان نامنظم است.

همین استدلال نشان می دهد که زبانِ B در ۲۹ نیز نامنظم است.

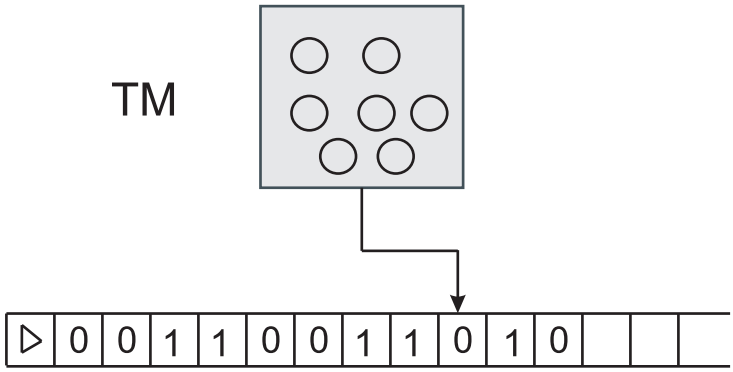
■ از لم پمپینگ استفاده کنید و نشان دهید که زبان های زیر منظم نیستند:

$$\begin{aligned} A &= \{0^n1^{2n}\} \\ B &= \{www \mid w \in a, b^*\} \\ C &= \{0^n1^n2^n \mid n \geq 0\} \end{aligned} \qquad (۳۱)$$

---

۸ ماشین تورینگ

محدودیت اساسی اتومات های تعینی یا نامتعین نداشتن یک حافظه نامحدوداست. به همین دلیل این ماشین ها می توانند تنها دسته‌خاصی 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



شکل ۲۱: ماشین تورینگ.

که تعریف مشخصی از آلگوریتم ارائه دهند. اگر مفهوم آلگوریتم را منحصر به اتومات های محدود کنیم، تعریف محدودی از آن ارایه داده‌ایم. 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 آلگوریتم را به شکل کلی اش بفهمیم یا آن را به طور دقیق صورت بندی کنیم بهتراست که خود را درجای 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 برگِ اضافه که نقش اساسی ندارند، کارکرد این شخص را بررسی کنیم . از خود می پرسیم که چنین شخصی دقیقاً چه می کند؟ وی داده‌هایی (مثلاً اعداد) از روی یک کاغذ می خواند. مطابق با این داده های دریافتی حالت های ذهنی‌اش 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 آلگوریتم است. از آنجا که به نوع فونت و رسم الخط و غیر علاقه مند نیسیتم، می توانیم فرض کنیم که تمامی داده ها به صورت علائمی از 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 ترتیب ماشینی درست می شود که آن را ماشین تورینگ می خوانیم ۲۱.

ماشین تورینگ یک قلم [۳۲] دارد که در هرلحظه 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 بنویسید. از جمله می تواند یک حرف را از روی یک خانه از نوار پاک کند. آخرین خانه در سمت چپ نوار که خانه‌ی ابتدایی نوار است همیشه با علامتِ $\triangleright$ 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. مجموعه‌ی حالت ها نیز با $Q = \{q_1, q_2, \cdots q_N\}$ مشخص می شوند. دقت کنید که ماشین

[۳۲] Head

تورینگ همچنان ماشینی است که تعداد حالت های آن محدوداست. در بین این حالت ها دو حالت متمایزند. یکی حالتِ q_s که حالتِ آغازین است و دیگری حالتِ q_{Halt} [۳۳] هرگاه که ماشین در حالتِ q_{Halt} قرار بگیرد، اجرای برنامه متوقف می شود. توابع انتقال ماشین تورینگ به صورت زیر عمل می کنند:

$$\delta : Q \times \{b\} \cup \{\triangleright\} \cup \Sigma \longrightarrow Q \times \{b\} \cup \Sigma \times \{L, R\}. \tag{۳۲}$$

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

$$(q, s) \longrightarrow (q', s', \alpha) \tag{۳۳}$$

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

$$(q, s, q', s', \alpha) \tag{۳۴}$$

مجموعه‌ی 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 حرفِ $\triangleright$ مشخص شده است. بعد از این علامت رشته قرار دارد و بعد از رشته علامت های b یعنی جای خالی تا انتهای نوار قرارگرفته اند. درابتدا قلم ماشین روی حرفِ $\triangleright$ حرف 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 حرفِ b را، روی خانه خوانده شده می نویسد و قلم اش را 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 حالتِ q_{Halt} می رسد، محاسبه متوقف می شود و رشته‌ای که روی نوار نوشته شده است جواب ماشین تلقی خواهد شد. دراینجا ماشین تورینگی را توصیف کرده‌ایم که خروجی اش یک رشته است، یعنی ماشین تورینگ چیزی را محاسبه می کند. ماشین تورینگی که در باره یک رشته تنها تصمیم گیری می کند ساختمان اندکی متفاوت دارد. این چنین ماشینی بجای یک حالتِ

[۳۳] حرفِ h از کلمه‌ی $Halt$ به معنای توقف آمده است.

q_{Halt} دو حالت q_{Accept} و q_{Reject} دارد و درپایان 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 نشده‌است. وقتی که ماشین درهرکدام ازاین 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. دراین حالت ماشین نمی تواند متوقف شود.

۱.۸ انواع ماشین تورینگ

آنچه که در بالا توصیف کردیم، ساختمان یک ماشین تورینگ تعینی و تک نواری بود. می توان ماشین های تورینگ پیچیده‌تری 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:

$$(q, s, q', s', \alpha \in -1, 0, 1), \tag{۳۵}$$

که در آن علامت 0 به معنای توقف قلم در خانه‌ی 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:

$$(q, s_1, s_2; q', s'_1, s'_2, \alpha_1, \alpha_2), \tag{۳۶}$$

که در آن همه علامت ها تعبیر روشنی دارند.

هم چنین می‌توان ماشین تورینگ نامتعین را تعریف کرد که در آن در هرلحظه ماشین می تواند نسخه‌هایی از خودش را تولید کند و همه نسخه ها را به طور همزمان دنبال کند. هرگاه که یکی از نسخه‌ها یا شاخه‌ها 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 هاآن است که همه‌ی آنها توسط ماشین تورینگ ساده‌ای 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 برنامه‌ی محاسبه یا آلگوریتم با این ماشین های توسعه یافته ساده تر باشد.

۲.۸ مثال هایی از ماشین تورینگ

در این بخش مثال های متفاوتی ارایه 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 که آیا ماشین تورینگ کار خواسته شده را انجام می دهد یا خیر.

مثال: دراین مثال یک ماشین تورینگ (یعنی یک آلگوریتم) طرح می کنیم که هر رشته‌ای روی نوار ماشین را پاک می کند. برنامه‌ی این ماشین به شکل زیراست:

$$\begin{aligned}&(q_s, \triangleright, q_1, \triangleright, +1)\\&(q_1, 0, q_1, b, +1)\\&(q_1, 1, q_1, b, +1)\\&(q_1, b, q_{Halt}, b, +1).\end{aligned} \qquad (۳۷)$$

مثال: دراین 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.

$$\begin{aligned}&(q_s, \triangleright, q_1, \triangleright, +1)\\&(q_1, 0, q_1, 1, +1)\\&(q_1, 1, q_1, 0, +1)\\&(q_1, b, q_{Halt}, b, +1).\end{aligned} \qquad (۳۸)$$

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

$$\begin{aligned}&(q_s, \triangleright, q_1, \triangleright, +1)\\&(q_1, 0, q_2, 1, +1)\\&(q_1, 1, q_2, 1, +1)\\&(q_2, 0, q_2, b, +1)\end{aligned}$$

$$(q_2, 1, q_2, b, +1)$$
$$(q_2, b, q_{Halt}, b, +1). \qquad (۳۹)$$

در مثال هایی که تا کنون شرح داده‌ایم تمام توابع انتقال ماشین را به تفصیل نوشته‌ایم. این نحوه‌ی 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 بالا [۳۴] به همان زبان بنیادی برحسب توابع انتقال ماشین تورینگ یعنی بر حسبِ زبان سطح پایین [۳۵] 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.

$$(q_s, \triangleright, q_s, \triangleright, +1)$$
$$(q_s, 0, q_s, 0, +1)$$
$$(q_s, 1, q_s, 1, +1)$$
$$(q_s, b, q_1, b, 0). \qquad (۴۰)$$

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: آلگوریتمی می خواهیم که بتواند یک رشته را وارون کند به این معنا که رشته‌ی $s_1 s_2 \cdots s_n$ را به رشته‌ی $s_n s_{n-1} \cdots s_2 s_1$ تبدیل کند. این کار را می توان با یک ماشین تورینگ دو نواری انجام داد. رشته‌ی $s_1 s_2 \cdots s_n$ روی نوار اول نوشته شده است و نوار دوم خالی است. در ابتدا هر دو قلم ها درابتدای نوارها و روی حروفِ $\triangleright$ 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. دراین 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

[۳۴] High Level Language

[۳۵] Low level language

که قلم اول به ابتدای نوار اول رسید ماشین به حالتِ q_{Halt} می رود و متوقف می شود. آنچه که روی نوار دوم نوشته شده است وارون رشته‌ای است که روی نوار اول نوشته شده بود. نوشتن این برنامه را بر حسب توابع انتقال به عهده خواننده می گذاریم.

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

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

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

د ـ 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 ردّ می شود، در غیر این صورت حالت ماشین به q_{Accept} تغییر می کند و رشته پذیرفته می شود.

مثال: زبان زیر را در نظر بگیرید:

$$Add := \{x^i y^j z^k \mid k = i + j\}. \qquad (۴۱)$$

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

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

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

د ـ قلم یک بار نوار را جاروب می کند. اگر هیچ حرف z یا x با قی نمانده باشد، رشته قبول می شود و اگر هرکدام از این حرف ها باقی مانده باشند رشته ردّ می شود.

مثال: زبان زیر را در نظر بگیرید:

$$Mulitply := \{x^i y^j z^k \mid k = ij\}. \tag{۴۲}$$

می خواهیم یک ماشین تورینگ طراحی کنیم که این زبان را تشخیص دهد. برنامه‌ی این ماشین به صورت زیراست:

الف ـ 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 ـ قلم به سمت چپ حرکت می کند و تمام حروفی را که در قسمت ِ y ها پاک کرده بود سرجایش می گذارد.

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

د ـ اگر در این مرحله هیچ حرفی باقی نمانده باشد، رشته مورد نظر پذیرفته شده و درغیر این صورت ردّ می شود.

در سه مثال بالا خواننده می بایست جزییات لازم برای تکمیل آلگوریتم را اضافه کند.

۳.۸ تز چرچ ـ 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 آلگوریتم تلقی کنیم یک بیان دیگر این سوال این است که آیا مسایلی وجود دارند که به طور الگوریتمیک قابل حل باشند ولی نتوان این آلگوریتم 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–تورینگ [۳۶] شناخته می شود. این تز بیان می کند که هیچ مدل محاسبه‌ای دیگری وجود

[۳۶] Church-Turing Thesis

ندارد که قدرتمند تر از ماشین تورینگ باشد، به عبارت دیگر هر نوع ماشین دیگر، هر نوع مدل محاسبه‌ی 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 ـ تورینگ می گوید که هر نوع محاسبه‌ای که درچارچوب قوانین فیزیکی و توسط یک پدیده‌ی فیزیکی قابل انجام باشد، قابل پیاده شدن توسط یک ماشین تورینگ است. بنابراین از یک طرف این تز تعریف کننده آلگوریتم است، (آلگوریتم هر آن چیزی است که توسط ماشین تورینگ قابل شبیه سازی باشد) و از طرف دیگر نشان دهنده‌ی محدودیت های قوانین و پدیده‌های فیزیکی برای محاسبه کردن و استدلال آلگوریتمی هستند. اگر به این تز اعتقاد راسخ داشته باشیم ، آن گاه شکست این تز می تواند به این معنا باشد که گستره جدیدی از قوانین فیزیک کشف شده‌اند 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 شمول [۳۷] شاخت که بتواند هر ماشین تورینگ دیگری را شبیه سازی کند. این ماشین که اختصاراً آن را با UTM نشان می دهیم درواقع مدل نظری کامپیوترهای قابل برنامه ریزی امروزی است. در کامپیوترهای امروزی ما می توانیم با افزودن نرم افزارهای جدید به کامپیوتر کاری کنیم که کامپیوتراولیه مان بتواند کارهای تازه تری انجام دهد. با نصب نرم افزارهای $Mathematica$ یا $Mathematica$ کاری می کنیم که کامپیوترساده‌ی 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 استادانه‌ای بازی کند و هزاران کار متنوع دیگر. همه اینها نشان دهنده‌این 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 ایده‌آل نیست زیرا به هرحال 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؟

[۳۷] Universal Turing Machine

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 مجموعه‌ی Q، حروف الفبایش، و تعداد دستورهایش مشخص می شود. هر دستور نیز چیزی نیست جز یک نگاشت از یک مجموعه محدود به یک مجموعه‌ی محدودِ دیگر. شمارش پذیر بودن مجموعه‌ی تمام ماشین های تورینگ این امکان را به ما می دهد که به هرماشین یک عدد طبیعی نسبت دهیم. این عدد، عدد تورینگ [۳۸] آن ماشین خوانده می شود. بنابراین می توانیم از ماشین های تورینگ $M_1, M_2, \cdots M_K, \cdots$ نام ببریم. برای آنکه ماشین جهان شمول تورینگ را بسازیم کافی است که علاوه بر رشته ورودی‌ِ $x = x_1 x_2 \cdots x_n$، 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:

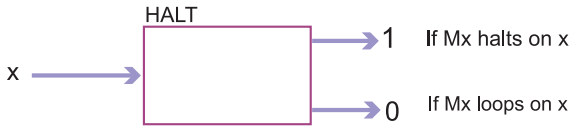
$$
\begin{aligned}
&UTM : (i\ b\ x) \longrightarrow (i\ b\ M_i(x)) \\
&UTM : (j\ b\ x) \longrightarrow (j\ b\ M_j(x)) \\
&UTM : (k\ b\ x) \longrightarrow (k\ b\ M_k(x)) \\
&\cdots \\
&\cdots
\end{aligned}
\qquad (۴۳)
$$

۵.۸ مسئله‌ی توقف

یکی از مفاهیم بسیار عمیق که پیامدهای گسترده‌ای دارد مسئله توقف یاایستادن [۳۹] است. برای پاسخ به مسئله هیلبرت در مورد وجود یک آلگوریتم که بتواند همه مسائل ریاضیات را حل کند، تورینگ این پرسش را طرح کرد که آیا یک آلگوریتم وجود دارد که بتواند بگوید یک ماشین تورینگ‌ِ 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 آلگوریتم وجود دارد که بتواند بگوید یک ماشین تورینگ‌ِ 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

[۳۸] Turing Number

[۳۹] Halting Problem



شکل ۲۲: آلگوریتم $HALT$ برای محاسبه تابع h.

پذیربودن ماشین های تورینگ استفاده کرده‌ایم. اگر چنین آلگوریتمی وجود داشته باشد معنایش این است که تابع زیر یک تابع قابل محاسبه است:

$$h(x) = \begin{cases} 1 & if \;\; M_x \;\; halts \;\; on \;\; x, \\ \\ 0 & if \;\; M_x \;\; doesn't \; halt \;\; on \;\; x \end{cases} \qquad (۴۴)$$

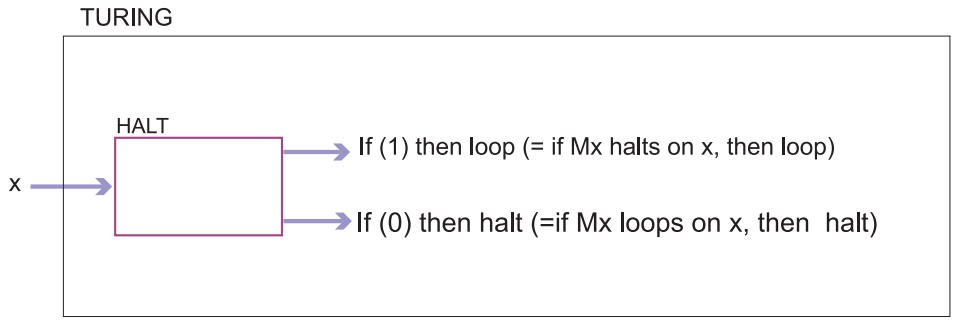
معنای محاسبه پذیر بودن این است که یک ماشین تورینگ (آلگوریتم) وجود دارد که می تواند این تابع را حساب کند و برای هر ورودی x مقدار $h(x)$ را که یا 0 است یا 1 تعیین کند. این ماشین تورینگ یا آلگوریتم را $HALT$ می نامیم، شکل ۲۲. حال نشان می دهیم که فرض وجود چنین ماشینی به تناقض منطقی می انجامد. برای این کار به ترتیب زیر استدلال می کنیم. یک برنامه بزرگتر درست می کنیم که برنامه‌ی $HALT$ را فراخوانی کند. شکلِ ۲۳ طرز کار این برنامه را نشان می دهد. دقت کنید که این برنامه که آن را $Turing$ می نامیم بازهم ورودی اش یک عدد صحیحِ 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 عددِ t را می دهیم. حال همان طور که شکل ۲۴ نشان می دهد، به یک تناقض منطقی می رسیم. زیرا بنابراین شکل

$$M_t \;\; halts \;\; on \; t \; if \;\; M_t \;\; loops \;\; on \;\; t,$$

(۴۵)

$$M_t \;\; loops \;\; on \; t \; if \;\; M_t \;\; halts \;\; on \;\; t,$$

این که ماشینِ $HALT$ وجود ندارد، به این معناست که تابع $h(x)$ محاسبه پذیرنیست. به عبارت دیگر هیچ آلگوریتمی وجود ندارد که این تابع را حساب کند.



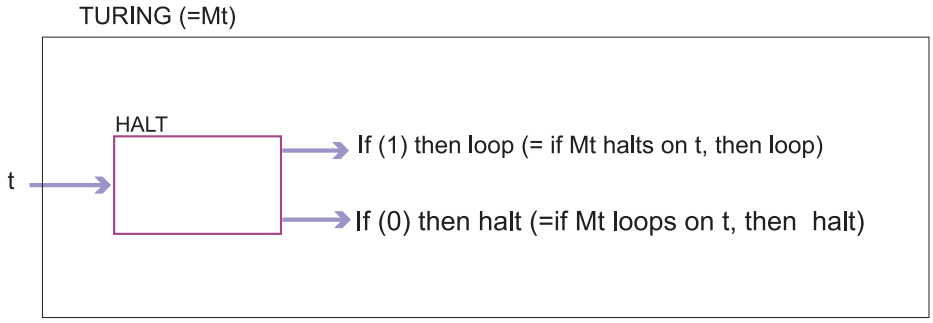
شکل ۲۳: اگر آلگوریتم یا ماشین تورینگ $HALT$ وجود داشته باشد، آلگوریتم یا ماشین تورینگ $Turing$ نیز وجود خواهد داشت.

اگر چنین ماشین تورینگی وجود داشته باشد به این معناست که می تواند بگوید آیا یک آلگوریتم 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 گلدباخ [۴۰] 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 ساده‌ی زیر توجه کند:

$$4 = 2 + 2, \qquad 6 = 3 + 3, \qquad 8 = 3 + 5, \qquad 10 = 5 + 5, \qquad 20 = 7 + 13, \qquad 30 = 23 + 7, \cdots. \quad (۴۶)$$

تا کنون کسی نتوانسته است این حدس را اثبات کند و یا مثال نقیضی برعلیه آن بیاورد و این مسئله همچنان یک مسئله باز در ریاضی است. حال فرض کنید که ما یک برنامه‌ی $M_{Goldbach}$ می نویسیم و این برنامه هر عددِ زوج x را می گیرد و تست می کند که آیا این عدد مجموع دو عدد اول هست یا نه. اگر جواب مثبت بود ماشین می ایستد و جواب اری می دهد واگر جواب منفی بود بازهم ماشین می ایستد و جواب نه می دهد. مسلم است که چنین برنامه ای را می توان نوشت. این برنامه یعنی برنامه $M_{Goldbach}$ تکلیف هر عددی را روشن می کند و ما می توانیم این برنامه را برای هر عددی از ۲ تا یک عدد زوج

[۴۰] Goldbach Conjecture



شکل ۲۴: می توانیم به ماشین $Turing$ یا همان M_t ورودی t را بدهیم و تناقض منطقی وجود ماشین $Turing$ و درنتیجه ماشین $HALT$ را ببنیم.

بسیار بسیار بزرگ چند هزار رقمی اجرا کنیم و می توانیم که مثلاً بگوییم که حدس گلدباخ تا اعدادِ 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:

$$L = \{The\ set\ of\ all\ even\ integers\ which\ are\ the\ sum\ of\ two\ prime\ numbers\}. \qquad (۴۷)$$

---

۹ کلاس های پیچیدگی محاسباتی

یکی از اهداف نظریه محاسبه، بررسی حداقل منابعی است که برای حل کلاس های مسائل مورد نیاز است. این منابع عبارتند از زمان[۴۱]، حافظه[۴۲] و انرژی[۴۳]. قبلاً در باره منبع انرژی بحث کرده‌ایم. دراین درس دو منبع دیگر یعنی زمان و حافظه را بررسی می‌کنیم. یک مسئله یا یک آلگوریتم معین را در نظر بگیرید. به عنوان مثال می خواهیم ببنیم که آیا در یک گراف

[۴۱] Time

[۴۲] Space

[۴۳] Energy

یک مسیر هامیلتونی وجود دارد یا نه؟ [۴۴] 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 راس ها را با V و مجموعه یال ها را با E نشان می دهیم. در واقع $E \subset V \times V$، یعنی اینکه عناصر E جفت هایی هستند به صورت (v, w) که نشان دهنده‌این است که راس های v و w به هم وصل هستند. معمولاً یک گراف را به صورت $G = (V, E)$ نشان می دهیم و تمامی اطلاعات این گراف را به صورت یک رشته طولانی که در آن راس های V و یال های E با فاصله از هم جدا شده اند، به صورت یک رشته به ماشین تورینگ می دهیم. طول این رشته را اندازه‌ی این مسئله خاص می گیریم و با عدد صحیح n نشان می دهیم. سوالی که با آن روبرو هستیم این است که برای حل مسئله مورد نظر با اندازه n حداقل زمان لازم یا حداقل حافظه لازم چقدراست. 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

$$f(n) \ \ is \ \ O(g(n)), \qquad if \quad \exists\, c \, and \, n_0 \quad \forall n > n_0 \quad f(n) \leq cg(n), \tag{۴۸}$$

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

$$f(n) \ \ is \ \ \Omega(g(n)), \qquad if \quad \exists\, c \, and \, n_0 \quad \forall n > n_0 \quad f(n) \geq cg(n), \tag{۴۹}$$

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

$$f(n) \ \ is \ \ \Theta(g(n)), \qquad if \quad f(n) = O(g(n)), \quad and \quad f(n) = \Omega(g(n)). \tag{۵۰}$$

[۴۴] این مسئله در بخش های بعدی این درس تعریف شده است.

تعریف هرگاه ماشین تورینگ M به ازای ورودی x بعد از طی تعدادِ $t(x)$ 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 (یامسئله)ی L را شناسایی کند، زمان اجرای آن مسئله را چنین تعریف می کنیم:

$$t(L) := max_{x \in L} t(x). \qquad (۵۱)$$

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 آماده‌ایم 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. لفظِ $DTime$ به این خاطر آمده است که از یک ماشین تورینگ تعینی استفاده کرده ایم. معنای این حرف این است که حلِ 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 مسئله ها یا زبان هاست و نه یک زبان خاص. حال می توانیم کلاس مسائل P،[۴۵] یعنی مسائل چند جمله‌ای را تعریف کنیم:

$$P := \bigcup_{k \geq 0} DTime(n^k). \qquad (۵۲)$$

کلاس مسایل P ، مجموعه همه مسایل یا آلگوریتم هایی است که در زمان چندجلمه‌ای حل می شوند. 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 داد ه ها که n عضو دارد، همه مسائلی هستند که برای حل آنها زمان چند جمله‌ای مورد نیاز است. بسیاری از مسائلی که نرم افزار هایی مثل $Maple$ و $Mathematica$ حل می کنند نیز چنین اند. البته نمی توان گفت که

[۴۵] Polynomial Time Problems or Polynomial Time Algorithms

هرنوع مسئله‌ای که این برنامه ها و نظایرآن حل می کنند از نوع مسائل آسان یا P است، زیرا آنها فقط یک نمونه از مسئله را با اندازه معین حل می کنند که به ازای حل آن هم بعضاً زمان طولانی صرف می کنند. به عنوان مثال اگر وقت کافی به هرکدام از این برنامه ها بدهید قادر خواهند بود که یک عدد بزرگ را تجزیه کنند، ولی مسئله تجزیه عدد یک مسئله از نوع P نیست زیرا آلگوریتمی که برحسب اندازه مسئله (تعداد رقم های عدد اولیه) چند جمله‌ای باشد شناخته نشده است. (دراینجا فقط در باره آلگوریتم های کلاسیک بحث می کنیم.)

۲.۹ کلاس NP

تعریف: به کلاس مسایلی که یک ماشین تورینگ نامعین در زمان $O(n^k)$ آنها را حل می کند، کلاس $NDTime(n^k)$ می گوییم. لفظِ $NDTime$ به این خاطر آمده است که از یک ماشین تورینگ نامعین استفاده کرده ایم. حال می توانیم کلاس مسائل NP [۴۶] را تعریف کنیم:

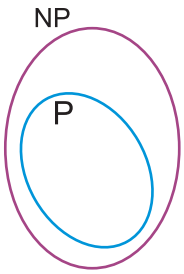
$$NP := \bigcup_{k\geq 0} NDTime(n^k). \tag{۵۳}$$

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

$$P \subset NP. \tag{۵۴}$$

هرگاه به شکلِ ۹ نگاه کنیم که فلسفه کلی ماشین های نامتعین را نشان می دهد، این موضوع روشن می شود که یک ماشین تورینگ نامعین وقتی یک مسئله را در زمان چند جمله‌ای حل می کند که یکی از شاخه‌های محاسبه به جواب برسد، یا به حالت q_{Accept} برسد. اگر بخواهیم کار یک ماشین تورینگ نامعین را با یک ماشین تورینگ معین شبیه سازی کنیم طبیعتاً می بایست همه شاخه ها را همزمان دنبال کنیم. این کار موجب افزایش نمایی تعداد حالت ها و تعداد مراحل محاسبه می شود. بنابراین به 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. درنتیجه 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

[۴۶] Non-deterministic Polynomial Time Problems or Algorithms



شکل ۲۵: کلاس های P و NP.

برای همان مسئله روی ماشین تورینگ معین هم یک آلگوریتم 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 ِ ۲۵.

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 یانه، (اگر هیچ قضیه‌ی کلی‌ای درمورد آن نداشته باشیم) نیازمند تست کردن تمام مسیرهای ممکن دراین 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 گواهی [۴۷] را که منجر به یافتن آن می شود با y نشان می دهیم. دراین 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

$$\begin{aligned} &if \;\; x \in L \;\; \exists \;\; y \in L \mid TM \;\; haltsinq_{Accept} \;\; ontheinput(x\ b\ y) \;\; after\ time(Poly(n)), \\ &if \;\; x \notin L \;\; \forall \;\; y \in L \mid TM \;\; haltsinq_{Reject} \;\; ontheinput(x\ b\ y) \;\; after\ time(Poly(n)). \end{aligned} \tag{۵۵}$$

دراینجا منظور از n اندازه ورودی x است.

[۴۷] Witness

۱۰ فروکاهش و مسایل کامل

می گوییم زبانِ A به زبانِ B کاهش پذیر است اگر یک ماشین تورینگِ 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 می کند و شرط زیر برقرار است:

$$R(x) \in B \leftrightarrow x \in A. \qquad (۵۶)$$

معنای این حرف آن است که به جای تصمیم گیری در مورد تعلق رشته‌ی x به زبان A می توانیم تنها با صرف زمان چند جمله‌ای در مورد مسئله معادل آن یعنی عضویت رشته‌ی $R(x)$ در زبانِ B تصمیم گیری کنیم. به عبارت دیگر مسئله بجای حل مسئله‌ی x می توانیم تنها با صرف زمان اضافیِ چند جمله‌ای، مسئله معادل آن یعنی $R(x)$ حل کنیم. هرگاه که این خاصیت برای همه‌ی $x \in A$ برقرار باشد، آنگاه مسئله یا زبانِ A به مسئله یا زبانِ B کاهش یافته‌است. فروکاهش یک خاصیت تراگذار[۴۸] است به این معنا که اگر A به B کاهش پذیر باشد و B به C آنگاه A به C کاهش پذیراست.

دقت کنید که کاهش پذیری A به B علیرغم ظاهر متقارن رابطه‌ی ؟؟ برابر با معادل بودنِ 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 زبان ها مثل $\mathcal{L}$ می گوییم زبانِ C کامل یا $\mathcal{L} - complete$ است اگر همه زبان های متعلق به $\mathcal{L}$ به زبان C کاهش پذیرباشند.

مفهوم کامل بودن یکی از مفاهیم بسیار مهم در نظریه محاسبه و کلاس های پیچیدگی است. مهمترین زبان های کامل آنهایی هستند که در کلاس NP قرار دارند و به آنها زبان ها یا مسائل $NP - Complete$ می گویند. در سال ۱۹۷۰ کوک و لوین [۴۹] نشان دادند که مسئله $3 - Sat$ یک مسئله $NP-$ کامل است. از آن به بعد تعداد بسیار زیادی مسئله $NP-$ کامل یافته شده اند که در زیر به آنها اشاره می کنیم. معنای این نتایج آن است که هرگاه یک آلگوریتم 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

[۴۸] Transitive

[۴۹] Kook and Levin

نمونه از مسایل $NP-$ کامل را معرفی می کنیم.

نمونه هایی از مسایل کامل در کلاس NP

۱.۱۰ مسئله تصدیق عبارت های منطقی

فرض کنید که $A = \{x_1, x_2, \cdots x_n\}$ متغیرهای منطقی هستند که می توانند مقادیر درست 1 یا نادرست 0 را اختیار کنند. یک عبارت k تایی[۵۰]، عبارتی است مثل

$$x_{i_1} \vee x_{i_2} \vee x_{i_3} \vee \cdots \vee x_{i_K}, \tag{۵۷}$$

که درآن هرکدام از x_i ها یا یکی از متغیرهای A ست یا یکی از متغیرهای $\overline{A} = \{\overline{x_1}, \overline{x_2}, \cdots \overline{x_n}\}$. به عنوان مثال یک $4-Clause$ عبارتی است مثل $x_1 \vee \overline{x_3} \vee \overline{x_5} \vee x_6$. حال یک جمله شامل تعدادی عبارت هم اندازه در نظر می گیریم مثل

$$S = C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_p. \tag{۵۸}$$

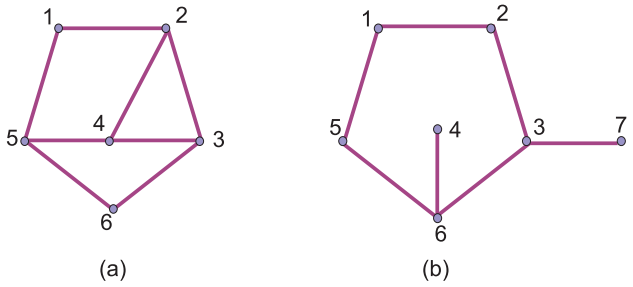
سوال این است که آیا یک انتخاب از متغیرهای A وجود دارد که به ازای آنها عبارتِ S مقدارِ ۱ را اختیار کند یعنی عبارت S ارزش منطقی درست داشته باشد یا خیر؟ لازمه این کار آن است که عبارت های C_1 تا C_p همه مقادیر ۱ را اختیار کنند. بنابراین، این مسئله را که اصطلاحاً مسئله $K-Sat$ [۵۱] می نامند، می توان به عنوان یک مسئله‌ی p معادله و n مجهولی از متغیرهای منطقی نامید، به این معنا که می خواهیم ببینیم دستگاه معادلات منطقی زیر جواب دارند یا نه:

$$\begin{aligned} C_1 &= 1 \\ C_2 &= 1 \\ &\cdots \\ C_p &= 1. \end{aligned} \tag{۵۹}$$

---

[۵۰] Clause

[۵۱] K-Satisfiability



شکل ۲۶: گراف a هامیلتونی است ولی گراف b هامیلتونی نیست.

بعداز 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. نمونه‌ای از این مسایل در ادامه می آید.

۲.۱۰ 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 هامیلتونی [۵۲] خوانده می شود، هرگاه این مسیر تمام راس های گراف را یک بار و فقط یک بار ملاقات کند (از آنها بگذرد). یک گراف هامیلتونی [۵۳]، گرافی است که حداقل یک مسیر هامیلتونی داشته باشد. به عنوان مثال در شکل ۲۶ گراف a هامیلتونی است ولی گراف b هامیلتونی نیست.

۳.۱۰ مسئله فروشنده دوره گرد

یک گراف وزن دار [۵۴] $G = (V, E)$، عبارت از گرافی است که به هرکدام 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 بسته‌ای که از همه شهرهای این

[۵۲] Hamiltonian Path

[۵۳] Hamiltonian Graph

[۵۴] Weighted graph

منطقه بگذرد. در این مسئله نیز سایز مسئله همان سایز گراف است.

۴.۱۰ 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 زیرمجموعه‌ای مثل $W \subset V$، به طوری که هر یالِ متعلق به E یکی یا هر دو سرش متعلق به W باشد. به عنوان مثال در شکل ؟؟ W یک پوشش راسی برای گراف است. مسئله پوشش راسی به این ترتیب است: یک گراف $G = (V, E)$ و یک عددِ 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 ۲۶ مجموعه‌ی $W = \{1, 4, 6\}$ یک پوشش راسی نیست ولی مجموعه‌ی $W' = \{1, 2, 4, 6\}$ یک پوشش راسی است.

۵.۱۰ مسئله زیرگراف های کامل

دریک 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 سایز k دارد یا خیر؟. درشکل ۲۶ گراف شکلِ a بزرگترین زیرگرافِ کامل اش اندازه ۳ دارد و گراف شکل b بزرگترین زیرگراف کامل اش اندازه‌ی ۲ دارد.

۶.۱۰ مسئله جمع زیر مجموعه ها

یک مجموعه‌ی 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.

۱۱ کلاس $PSPACE$

برای آنکه منابع حافظه‌ای [۵۵] را برای یک آلگوریتم بررسی کنیم، احتاج به تعریفی از میزان مصرف حافظه داریم. تعداد خانه هایی از نوار را که یک ماشین تورینگ مصرف می کند تا به حالت q_{Halt} برسد، حافظه یا فضای مصرف شده می خوانیم. بنابراین می گوییم یک زبان L در کلاس $PSPACE$ است اگر یک ماشین تورینگِ TM وجود داشته باشد به نحوی که

$$\forall\, x \in L,\ \ TM \ halts \ on \ x \ \ after \ the \ consumption \ of \ P(n) \ bits \ of \ tape \ where \ n \ is \ the \ length \ of \ x. \quad (۶۰)$$

رابطه زیر بین این کلاس ها وجود دارد:

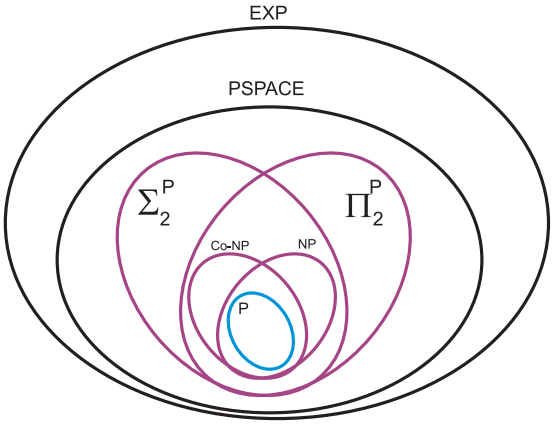
$$P \subset PSPACE, \qquad NP \subset PSAPCE. \quad (۶۱)$$

نامساوی اول واضح است زیرا اگر یک ماشین تعداد مراحلی را باگرفتن ورودی x طی می کند تا به حالت پایانی برسد، یک چند جمله‌ای برحسب $n := |x|$ باشد، حتماً تعداد خانه هایی را نیز که از نوار مصرف می کند چند جمله‌ای خواهد بود. برای نامساوی دوم احتیاج به استدلال بیشتری داریم. می دانیم که کار یک ماشین تورینگ نامعین را می توانیم روی یک ماشین تورینگ معین شبیه سازی کنیم. اگر اندازه رشته‌ی ورودی را با n نشان دهیم، ماشین تورینگ نامعین تعدادِ $O(2^n)$ مسیر را طی می کند و لااقل در یکی از این مسیرها در زمان چندجمله $O(n^k)$ به جواب می رسد. می توانیم آدرس هرکدام از این مسیرها را به ماشین تورینگ معین بدهیم. این آدرس که همان آدرس نحوه شاخه شاخه شدن در شکلِ ؟؟ 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 می کند در زمان چند جمله‌ای کار خود را تمام می کند و به حالت q_{Halt} می رسد. بنابراین در این زمان فقط تعداد $O(n^k)$ از خانه‌های نوار تغییر می کنند. برای ورودی دیگر یعنی x و آدرس یک شاخه دیگر خانه های نوار دوباره به حالت اول برمی گردند. بنابراین برای تست کردن کل شاخه‌ها بازهم تعداد خانه های نوار که تغییر می کنند یک چند جمله‌ای از n خواهد بود. بنابراین نتیجه می گیریم که $NP \subset PSPACE$.

تعریف: کلاس L به کلاس مسائلی گفته می شود که در زمان لگاریتمی توسط یک ماشین تورینگ قابل تصمیم گیری هستند. به یک معنا این کلاس ساده ترین مسائل را در بردارد.

---

[۵۵] Memory Resources



شکل ۲۷: نمونه‌ای از کلاس های پیچیدگی و رابطه آنها.

تعریف: کلاس EXP به کلاس مسائلی گفته می شود که در زمان نمایی توسط یک ماشین تورینگ قابل تصمیم گیری هستند. به یک معنا این کلاس سخت ترین مسائل را در بردارد.

رابطه کلاسهایی که تا کنون گفته ایم به شکل زیراست:

$$L \subset P \subset NP \subset PSPACE \subset EXP. \tag{۶۲}$$

علاوه بر کلاس های بالا ده ها کلاس پیچیدگی دیگر نیز وجود دارند که رابطه بین آنها از رابطه ساده بالا بسیار پیچیده تراست. پرداختن به این گونه کلاس ها و رابطه بین آنها از دایره بحث ما خارج است. خواننده علاقمند می‌تواند به کتب تخصصی این رشته مراجعه کند.

۱۲ تمرین ها:

- برای هرکدام از زبان های زیر یک اتومات تعینی محدود طراحی کنید که آن زبان را تشخیص دهد. در همه زبان ها الفبا برابر است با $\sum = \{0,1\}$ بجز زبان های آخر که الفبای آن زبان انگلیسی است:

$$
\begin{aligned}
&L_1 := \{w \mid w \text{ contains the substring } 1011\}\\
&L_2 := \{w \mid w \text{ contains an even number of } 0'\text{s and an odd number of } 1'\text{s }\}\\
&L_3 := \{w \mid w \text{ begins with 0 and ends with 1 }\}\\
&L_4 := \{w \mid w \text{ begins with 00 and ends with 11 }\}\\
&L_5 := \{w \mid w \text{ w is of length N }\}\\
&L_6 := \{w \mid w \text{ does not contain the substring 111 }\}\\
&L_7 := \{w \mid w \text{ begins and ends with the same letter }\}\\
&L_8 := \{w \mid w \text{ w}'\text{s length is a multiple of 3 }\}\\
&L_9 := \{w \mid w \text{ w}'\text{s length is a multiple of 3 and contains an even number of } 0'\text{s }\}\\
&L_{10} := \{w \mid w\text{contains the word WORK }\}\\
&L_{11} := \{w \mid w\text{contains the word WORK twice }\} \qquad (۶۳)
\end{aligned}
$$

■ هرکدام از زبان های زیر اشتراک دو زبان ساده تر است. در هر مورد نخست ماشین های DFA ای بسازید که آن زبان های ساده تر را شناسایی کند. سپس ماشین DFA ای بسازید که زبان داده شده را شناسایی کند. در تمام موارد الفبای زبان ها برابر است با $\sum := \{a, b\}$.

$$
\begin{aligned}
&A := \{w \mid w \text{ contains at least two a}'\text{s and at least three b}'\text{s}\}\\
&B := \{w \mid w \text{ contains at most two a}'\text{s and at least three b}'\text{s}\}\\
&C := \{w \mid w \text{ contains at most two a}'\text{s and at most three b}'\text{s}\}\\
&D := \{w \mid w \text{ contains the substring aba and the substring bab}\}\\
&E := \{w \mid w \text{ starts with aaa and ends with bbb}\}. \qquad (۶۴)
\end{aligned}
$$

■ الف: فرض کنید که $A \subset \sum^*$ یک زبان باشد. دراین صورت متمم آن زبان عبارت است از $A' := \sum^* /A$. نشان دهید که اگر M یک اتومات تعینی 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 غیر تعینی 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.

- در هرکدام از موارد زیر زبان داده شده متمم یک زبان ساده‌تر است. نخست یک اتومات تعینی محدود بسازید که آن زبان ساده‌تر را شناسایی کند. سپس با استفاده از آنچه که در تمرین قبلی یادگرفتید ماشینی بسازید که زبان داده شده را شناسایی کند:

$$
\begin{aligned}
A &:= \{w \mid w \text{ does not contain the substring ab}\} \\
B &:= \{w \mid w \text{ does not contain the substring abab}\} \\
C &:= \{w \mid w \text{ contains neither the substring ab nor ba}\} \\
D &:= \{w \mid w \text{ is any string not in } \mathrm{a}^*\mathrm{b}^*\} \\
E &:= \{w \mid w \text{ is any string that does not contain exactly two a's}\}.
\end{aligned} \tag{۶۵}
$$

- دیاگرام حالتِ ماشین های NFA ای را رسم کنید که هرکدام از زبان های زیر را تشخیص دهد. در تمام موارد الفبا عبارت است از $\sum = \{0, 1\}$.

$$
\begin{aligned}
A &:= 0^*1^* \\
B &:= \sum 0 \sum \\
C &:= \sum 00 \sum \\
D &:= 0^*(10^*)1 \\
E &:= \sum 0^* \sum
\end{aligned}
$$

$$F := (\sum\sum)^* \cup (\sum\sum\sum)^* \cup (01)^*. \tag{۶۶}$$

■ 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 زبانِ $(01 \cup 001 \cup 010)^*$ را شناسایی کند.
ب: این NFA را به یک DFA معادل تبدیل کنید.

■ عبارت های منظم زیر را به NFA تبدیل کنید:

$$\begin{aligned} &(0 \cup 1)^* 000 (0 \cup 1)^* \\ &(((00)^*(11)) \cup (01))^* \\ &0^*. \end{aligned} \tag{۶۷}$$

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

$$\begin{aligned} A &= \{0^n 1^n 2^n \mid n \geq 0\} \\ B &= \{www \mid w \in \{a, b\}\} \\ C &= \{a^{2^n} \mid n \geq 0\}. \end{aligned} \tag{۶۸}$$

■ هرگاه $\omega = \omega_1\omega_2 \cdots \omega_n$ یک رشته باشد معکوس آن رشته‌ی $\omega^R = \omega_n \cdots \omega_2\omega_1$ 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.