

مقدمه ای بر نظریه رایانش

وحید کریمی پور- دانشکده فیزیک - دانشگاه صنعتی شریف

۷ دی ۱۳۹۹

۱ مقدمه

سال ۱۹۰۰ در کنگره ریاضیدانان دیوید هیلبرت ۲۳ مسئله معروف خود را طرح کرد که مسیر عمده ریاضیات را در قرن بیستم تعیین کرد و بسیاری از پیشرفت های ریاضیات در قرن بیستم ناشی از تلاش نسل های مختلف ریاضیدانان برای حل این مسایل بوده است. مسئله دهم هیلبرت چنین است:

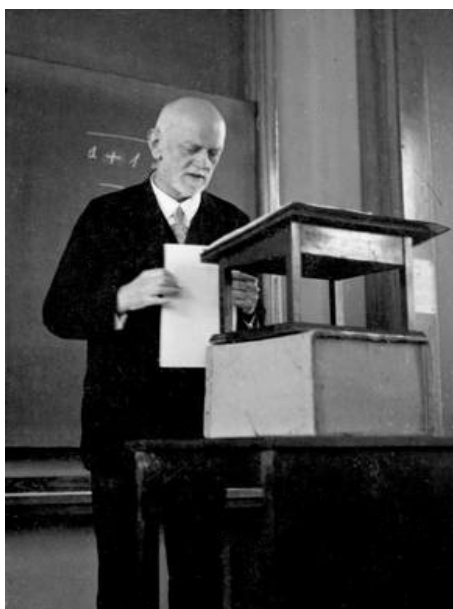
■ **مسئله دهم هیلبرت:** به ازای هر معادله دیوفانتی با ضرایب صحیح آیا می توان فرایندی را طراحی کرد که با طی کردن مراحل محدودی تعیین کرد که این معادله پاسخ صحیح دارد یا خیر؟

معادله دیوفانتی با ضرایب صحیح یک معادله چند جمله ای از یک یا چند متغیر است. منظور هیلبرت چنین معادله هایی است:

$$\begin{aligned}x^2 + y^2 &= z^2 \\x^n + y^n &= z^n \quad n > 2 \\x^n + y &= z^2 + x + y^3.\end{aligned}\tag{1}$$

امروزه می دانیم که معادله اول (یا همان قضیه فیثاغورث) بی نهایت جواب صحیح دارد و معادله دوم یا همان قضیه آخر فرما هیچگونه جواب صحیحی ندارد. منظور هیلبرت به زبان امروزی این است که آیا می توان با استفاده از یک الگوریتم پاسخی برای این سوال یافت یا خیر. البته او

از کلمه آگوریتیم که امروزه رایج شده استفاده نمی کند و به همین دلیل از « فرایندی که با طی کردن مراحل محدودی به پاسخ برسد » استفاده می کند. تلاش برای پاسخ گویی به این مسئله توسط آلن تورینگ منجر به ابداع نظریه رایانش و کامپیوترها به شکلی که امروزه می شناسیم شده است. پاسخ این سوال به یک معنا به کشف مهم دیگری در باره مبانی ریاضیات و منطق نیز مرتبط است که توسط کورت گودل منطق دان بزرگ اتریشی به یک انقلاب در ریاضیات و منطق و فلسفه منجر شده است. این درس مقدمه ای بسیار ابتدایی بر این موضوع است.



شکل ۱: دیوید هیلبرت (۱۸۶۲-۱۹۴۳ م):

دیوید هیلبرت یکی از بزرگترین و تاثیرگذارترین ریاضیدانان نیمه دوم قرن نوزدهم و نیمه اول قرن بیستم است. او سهم عمده ای در توسعه چندین شاخه از ریاضیات داشت، از جمله نظریه ناوردادا، حساب تغییرات، جبر جابجایی، نظریه جبری اعداد، مبانی هندسه، نظریه طیفی اپراتورها و کاربرد آن در معادلات انتگرال، ریاضی فیزیک و مبانی ریاضیات. میزان تاثیر او را می توان از سخنرانی اش در کنگره ریاضیدانان در سال ۱۹۰۰ فهمید که فهرستی از مسایل را طرح کرد که به تنهایی عمده خط سیر ریاضیات را در قرن بیستم تعیین کرد. این مسایل امروزه به ۲۳ مسئله هیلبرت مشهور هستند.

David Hilbert^۱

۲ تعریف شهودی از الگوریتم

بسیاری از مسائل ریاضیات را ما به روش الگوریتمی 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 الگوریتم 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 الگوریتم 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 الگوریتمی تنها برای بعضی از مسائل خاص ریاضیات وجود دارند و بعضی از مسائل تن به راه حل الگوریتمی نمی دهند و برای حل آنها همچنان باید فکر و ابتکار و خلاقیت به خرج داد. مثلاً 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 انتگرال نامعین را نیز می توان به صورت الگوریتمی محاسبه کرد. این کار را نخستین بار رابرت هنری ریش^۱ در سال ۱۹۶۸ انجام داد. از آن موقع تا کنون الگوریتم های سریع تری برای دسته وسیع تری از توابع حتی توابع خاص ابداع شده اند به طوری که امروزه می توانیم بگوییم محاسبه انتگرال را نیز می توان به صورت الگوریتمی 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 عموماً مسائل هندسه را می بایست با ابتکارات خاص و با روشهای نه چندان سراسر 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 الگوریتمی 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 الگوریتمی حل کرد؟ آیا می توان قضیه های ریاضیات را نیز به روش الگوریتمی ثابت یا رد کرد؟ برای مثال به قضیه فرما توجه کنیم: صورت این قضیه این است که معادله زیر

$$x^n + y^n = z^n, \quad n > 2 \quad (۲)$$

Robert Henry Risch^۱

هیچ پاسخ صحیحی ندارد. آیا الگوریتمی 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 الگوریتم در آورد و آن را ثابت یا رد کرد؟

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

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 الگوریتم 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 الگوریتم 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:

Goldbach Conjecture^۲

Alan Turing^۳

Alonzo Church^۴

Decision Problems^۵

Computational Complexity^۶

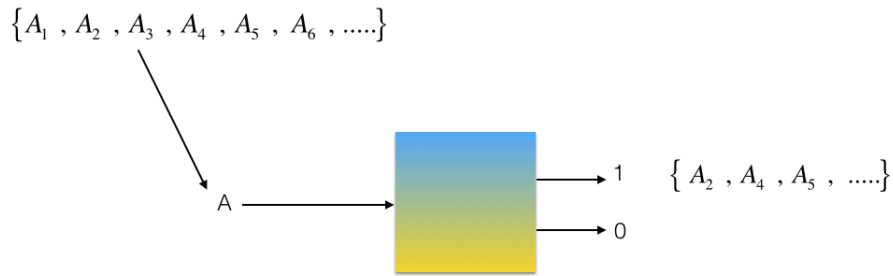
۳ تعاریف مقدماتی

تقریباً 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. مثلاً می پرسیم که آیا یک عدد اول است یا نه؟ آیا یک عدد بر عدد دیگر بخش پذیر است یا نه؟ آیا یک ماتریس وارون پذیر است 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 مثلاً پیدا کردن 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. مثلاً 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 اول 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 مثلاً 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 (مثلاً 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

^۷Decision Problems



شکل ۲: شمای کلی یک ماشین که یک مسئله تصمیم گیری را حل می کند .

P هستند. همانطور که شکل (۲) نشان می دهد، ماشین می تواند عناصری از یک مجموعه بی نهایت را دریافت کرده و فقط بعضی از آنها را بپذیرد.

مجموعه $\{A_1, A_2, A_3, \dots\}$ و خاصیت P مجموعاً یک مسئله A را تعریف می کنند و A_i ها نمونه های مسئله A خوانده می شوند. هر مسئله را می توان همواره به صورت مجموعه ای از رشته های 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 و ارون 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 ها در آورد. درک این موضوع ما را به تعاریف زیر می رساند.

۱.۳ الفبا و زبان

تعریف الفبا و رشته ها : هر مجموعه ای از علائم 10 یک الفبا 11 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 12 یا یک کلمه می گوئیم. 000111 و 010101 هر دو 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 بی نهایت عضوی است. بنابراین رشته های $u = 000111$ و $v = 010101$ هر دو عضو Σ^* هستند. رشته تهی رشته ای است که شامل هیچ حرفی نباشد. این رشته بانماد ϵ نشان داده می شود. طول رشته ϵ برابر

Problem^۸
 Instants of the problem^۹
 Symbols^{۱۰}
 Alphabet^{۱۱}
 String^{۱۲}

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

$$\begin{aligned} L_1 &= \{w \in \Sigma^* | w \text{ ends with } 0\} \\ L_2 &= \{w \in \Sigma^* | w \text{ contains at least three zeros}\} \\ L_3 &= \{w \in \Sigma^* | w \text{ ends with } 1010\} \end{aligned} \quad (3)$$

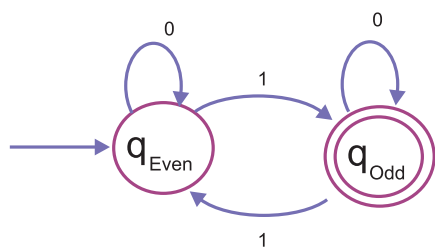
یک زبان می تواند یک مجموعه متناهی یا یک مجموعه نامتناهی باشد. به عنوان مثال مجموعه $L_4 = \{00, 11\}$ نیز یک زبان روی الفبای Σ است.

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 الگوریتم و یک ماشین استفاده کرد؟

۴ 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_1, q_2, \dots, q_N نشان می دهیم. وقتی که این ماشین علامت s را دریافت می کند تغییر حالت 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 ای 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

A language over Σ^*



شکل ۳: یک اتومات تعیینی که رشته‌های شامل تعدادی فرد از 1 را می‌پذیرد.

یک مثال ساده را مطالعه می‌کنیم. شکل ۳ یک اتومات محدود را نشان می‌دهد که کارش آن است که پاریته رشته‌های متشکل از علائم 0 و 1 را تعیین کند. هرگاه رشته‌ای مثل 0011 به این اتومات داده شود، درحالت q_{Even} و هرگاه رشته‌ای مثل 01011011 به آن داده شود در حالت q_{Odd} قرار می‌گیرد. شکل ۳ نشان می‌دهد که این اتومات وقتی که در حالت‌های مختلف است چگونه به علائم دریافتی عکس‌العمل نشان می‌دهد. در واقع طرز کار این اتومات را می‌توان در توابع زیر که تابع انتقال خوانده می‌شوند خلاصه کرد:

$$\begin{aligned} \delta(q_{Even}, 0) &= q_{Even}, & \delta(q_{Even}, 1) &= q_{Odd}, \\ \delta(q_{Odd}, 0) &= q_{Odd}, & \delta(q_{Odd}, 1) &= q_{Even}. \end{aligned} \quad (4)$$

این روابط نشان دهنده یک برنامه هستند که اتومات برطبق آنها عمل می‌کند. این برنامه به سادگی در شکل ۳ که اصطلاحاً دیاگرام حالت^{۱۴} نامیده می‌شود نشان داده شده است. می‌توان گفت که این اتومات رشته‌های با تعدادی فرد از 1 را پذیرفته و بقیه رشته‌ها را رد می‌کند. می‌گوییم که زبان این ماشین مجموعه تمام رشته‌هایی است که تعداد فردی عدد 1 دارند. قبل از آنکه مثال‌های بیشتری از اتومات‌های محدود ارائه دهیم، بهتر است که تعریف دقیق و رسمی آن را مشخص کنیم.

۱.۴ تعریف دقیق از یک اتومات متعین و محدود

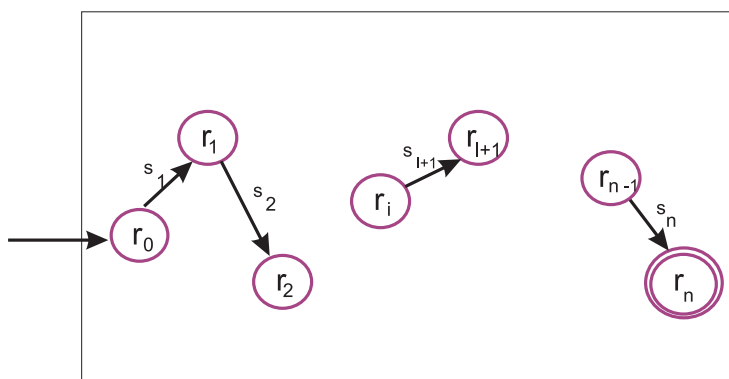
تعریف: یک اتومات تعیینی محدود^{۱۵} دستگاهی است متشکل از پنج عنصر $DFA = \{Q, \Sigma, \delta, q_0, F\}$ که در آن:

الف: Q : یک مجموعه محدود است. $Q = \{q_0, q_1, \dots, q_n\}$. هر عضو Q یک حالت^{۱۶} خوانده می‌شود.

^{۱۴} State Diagram

^{۱۵} Deterministic Finite Automata (DFA)

^{۱۶} State



شکل ۴: پذیرش یک رشته توسط ماشین DFA .

ب : Σ یک الفباست.

ج : $\delta : Q \times \Sigma \rightarrow Q$ 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 آغازین^{۱۷} است و

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

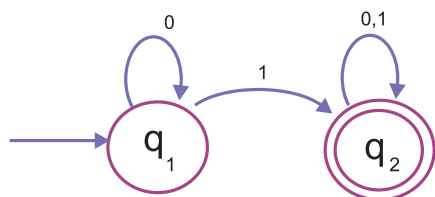
هرگاه $w = w_1 w_2 \dots w_n \in \Sigma^*$ یک رشته باشد می گوییم ماشین M ، یا اتومات تعینی M این رشته را می پذیرد هرگاه ماشین که ابتدا در حالت q_0 است بتواند با خواندن یکی یکی حرف های این رشته از چپ به راست، تغییر حالت داده و سرانجام پس از خواندن آخرین حرف یعنی w_n در یکی از حالت های پذیرش قرار گیرد. ۴

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 :

$$r_0 = q_0 - ۱$$

$$\delta(r_i, w_i) = r_{i+1} - ۲$$

Start State^{۱۷}
Accept States^{۱۸}



شکل ۵: این اتومات تعیینی رشته‌هایی را می‌پذیرد که حداقل شامل یک نماد 1 باشند.

$$r_n \in F - ۳$$

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

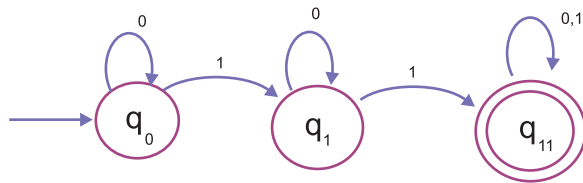
■ چگونه طرز کار ماشین تعیینی را دنبال کنیم: بهترین کار برای دنبال کردن محاسبه در یک ماشین تعیینی این است که انگشت خود را روی حالت کنونی ماشین قرار دهیم و ببینیم ورودی‌های متمادی انگشت ما را به کجا می‌برند. هر موقع که رشته تمام شد چنانچه انگشت ما روی یکی از حالت‌های قبولی وجود داشته باشد، آن رشته پذیرفته می‌شود و در غیر این صورت رشته رد خواهد شد.

مجموعه تمام رشته‌هایی که توسط ماشین پذیرفته می‌شوند زبان آن ماشین خوانده می‌شود و با $L(M)$ نمایش داده می‌شود. در این حالت می‌گوییم که ماشین M آن زبان را تشخیص می‌دهد.

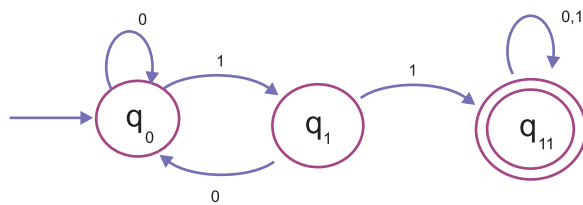
۲.۴ مثالهایی از اتومات‌های متعین و محدود

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



شکل ۶: این اتومات محدود رشته هایی را می پذیرد که حداقل شامل دو نماد 1 باشند.



شکل ۷: این اتومات تعیینی رشته هایی را می پذیرد که حداقل شامل دو نماد 1 به صورت متوالی باشند.

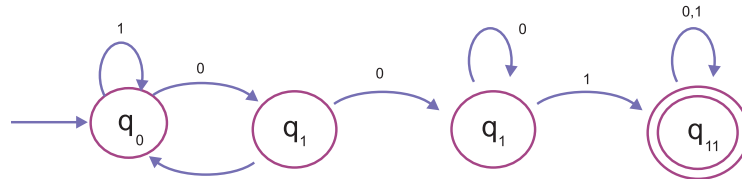
اند اگر هر دو یک زبان را تشخیص دهند.

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

■ تمرین: در تمام قسمت های زیر الفبای زبان ها عبارت است از $\{a, b\}$. در هر قسمت یک ماشین DFA 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. $\{w \mid w \text{ has at least two a's.}\}$
- b. $\{w \mid w \text{ has exactly two a's }\}$
- c. $\{w \mid w \text{ has at most two a's. }\}$



شکل ۸: این اتومات تعینی رشته هایی را می پذیرد که شامل زیر رشته‌ی 001 باشند.

- d. $\{w \mid w \text{ has at least two consecutive a's.}\}$
- e. $\{w \mid w \text{ has exactly one set of two consecutive a's and at least one set of three consecutive b's.}\}$
- f. $\{w \mid w \text{ has an even number of a's.}\}$
- g. $\{w \mid w \text{ has the sub string 'abab' }\}$
- h. $\{w \mid w \text{ has an even number of substrings 'ab'.}\}$
- k. $\{w \mid w \text{ starts with a and ends with b.}\}$
- l. $\{w \mid w \text{ starts with aa and ends with bb.}\}$ (۵)

۵ 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 آن 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

^{۱۹}decision problem

^{۲۰}Instants of the problem

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.

۱.۵ اعمال روی زبان ها

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

$$A \cup B = \{x \in \Sigma^*, x \in A \text{ or } x \in B\}$$

$$A \circ B = \{xy \in \Sigma^*, x \in A \text{ and } y \in B\} \quad (۶)$$

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

$$A = \{aa, bb, cc\}$$

$$B = \{00, 11\}, \quad (۷)$$

آنگاه

$$A \cup B = \{aa, bb, cc, 00, 11\}$$

^{۲۱}Union

^{۲۲}Concatenation

$$A \circ B = \{aa00, aa11, bb00, bb11, cc00, cc11\}$$

$$B \circ A = \{00aa, 11aa, 00bb, 11bb, 00cc, 11cc\}. \quad (8)$$

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

$$L^* = \{x_1x_2 \cdots x_n | n \geq 0 \text{ and } x_i \in A\}. \quad (9)$$

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

$$A^* = \{\epsilon, aa, bb, cc, aaaa, bbbb, cccc, aabb, aacc, bbcc, bbaa, ccaa, aaaaaa, \dots\}. \quad (10)$$

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

■ می توانیم اشتراک دو زبان را نیز تعریف کنیم

$$A \cap B = \{w | w \in A, \text{ and } w \in B\}. \quad (11)$$

■ **تعریف:** هرگاه L یک زبان باشد، متمم آن را با \bar{L} نمایش می دهیم. بنابر تعریف

$$\bar{L} := \{w \in \Sigma^* | w \notin L\} \quad (12)$$

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

■ **مثال:** ۱ - هرگاه L زبانی باشد که رشته های آن طول زوج دارند \bar{L} زبانی است که رشته های آن طول فرد دارند.

۲ - هرگاه هرگاه L زبانی باشد که رشته های حد اقل دارای یک زیر رشته 001100 باشند، \bar{L} زبانی است که رشته های آن شامل زیر رشته بالا نیستند.

۲ - هرگاه هرگاه L زبانی باشد که حرف آغاز و پایان رشته های آن با هم مساوی باشند، \bar{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 متمم 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.

- $\{w \mid w \text{ does not contain the substring } ab.\}$
- $\{w \mid w \text{ length}(w) \text{ is not a multiple of } 3.\}$
- $\{w \mid w \text{ does not start with } aa.\}$
- $\{w \mid w \text{ does not end with } bb.\}$
- $\{w \mid w \text{ does not contain consecutive } b\text{'s}.\}$ (۱۳)

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

- $\{w \mid w \text{ every odd position of } w \text{ is } 1.\}$
- $\{\text{The empty set.}\}$
- $\{\text{All strings except the empty string.}\}$ (۱۴)

دقت کنید که با ترکیب زبان ها می توانیم زبان های خیلی پیچیده بسازیم. حال ببینیم آیا ماشینی 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.

^{۲۳}Regular Language

■ **قضیه:** مجموعه زبان های منظم تحت اعمال اجتماع، اشتراک، متمم گیری، چسباندن و ستاره یک مجموعه بسته است.

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 قبلاً (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 نامتعیین را بفهمیم. در آن 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 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 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 ماشین M_1 بدهیم و یک نسخه‌اش 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 هرکدام که به یک حالت پذیرش رسید از روی آن می فهمیم که رشته‌ی 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 $A \cup B$ نبوده‌است. با این توصیف ماشینی که زبان $A \cup B$ را شناسایی می کند عبارت است از:

$$M = (\Sigma, Q_1 \times Q_2, (q_1, q_2), \delta, F) \quad (15)$$

که در آن

$$\delta : \Sigma \times Q_1 \times Q_2 \longrightarrow Q_1 \times Q_2, \quad \delta(s, (q_1, q_2)) = (\delta_1(s, q_1), \delta_2(s, q_2)). \quad (16)$$

این رابطه بیان دقیق این امرشهودی است که هر دو 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 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 $A \cup B$ بوده است. بنابراین حالت های پذیرش نهایی عبارتند از:

$$F = (F_1 \times Q_2) \cup (Q_1 \times F_2). \quad (17)$$

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.

■ 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

$$M_A = \{\Sigma, q_a, Q_a, \delta_a, F_a\} \quad (18)$$

و

$$M_B = \{\Sigma, q_b, Q_b, \delta_b, F_b\}. \quad (19)$$

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

$$M_{A \cap B} = \{\Sigma, (q_a, q_b), Q_a \times Q_b, \delta, F_a \times F_b\} \quad (20)$$

که در آن

$$\delta((q_1, q_2), s) := (\delta_a(q_1, s), \delta_b(q_2, s)) \quad (21)$$

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.

۶ عدم تعین و اتومات های نامتعین و محدود

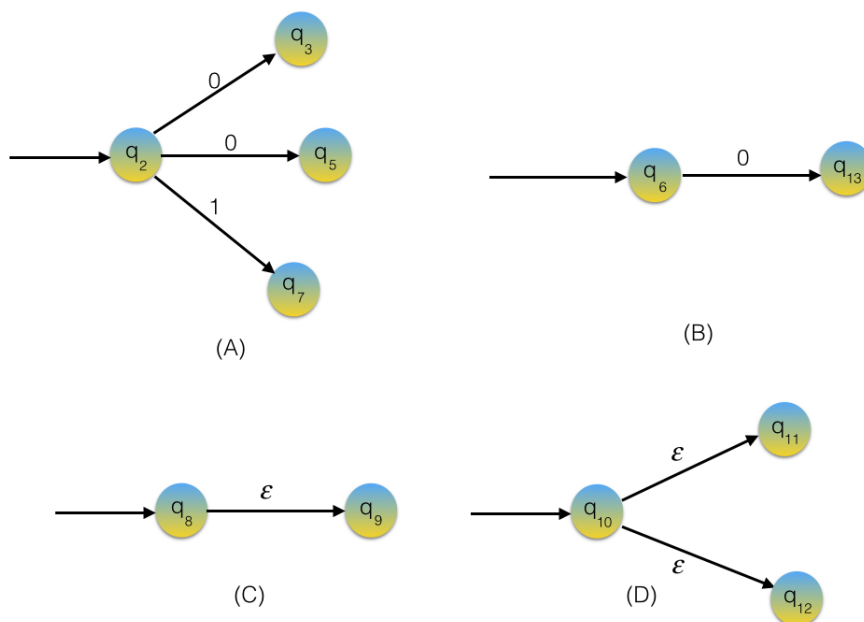
اگر بخواهیم مسئله پیچیده ای حل کنیم یک راه آن است که به جای یک رایانه از دو یا چند رایانه کمک بگیریم که به طور همزمان کار می کنند و به جستجوی حل مسئله مورد نظر ما می پردازند. این کاری است که امروزه خیلی مرسوم است. اتومات های نامتعین و محدود^{۲۴} در واقع مدلی نظری برای کار همزمان چند رایانه با هم هستند. 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 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 قواعد یک ماشین نامتعین مطابق شکل ۹ 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 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 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 بازم تکرار شود.

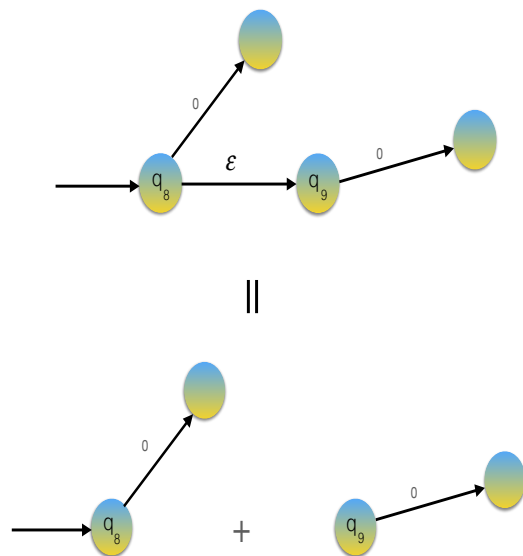
وقتی که از یک حالت یک فلش خارج می شود، به این معناست که حالت بعدی ماشین به صورت یکتا تعیین می شود. اگر ماشین در این حالت باشد، با گرفتن ورودی مورد نظر حالت بعدی ماشین به صورت یکتا تعیین می شود.

^{۲۴}Non-deterministic Finite Automata (NFA)



شکل ۹: قواعد کار برای یک ماشین نامتعیین. بالا سمت چپ: با گرفتن 0 یک حالت از ماشین ممکن است به دو حالت برود. بالا سمت راست: اگر وقتی که ماشین در حالت q_6 است حرف 1 را دریافت کند، ماشین یا این شاخه از ماشین متوقف می شود. پایین سمت چپ و راست: وقتی ماشین به حالت q_8 می رسد، به دو یا سه ماشین مجزا تبدیل می شود که به کار خود ادامه می دهند. (شکل بعدی را ببینید.)

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.



شکل ۱۰: معنای ϵ . وقتی که ماشین به حالت q_8 می‌رسد، به دو ماشین جداگانه تبدیل می‌شود که هر کدام راه خود را جداگانه طی می‌کنند. بقیه حالت‌ها برای سادگی نشان داده نشده‌اند.

سوم: وقتی که یک شاخه از ماشین به حالتی می‌رسد که به ازای یک ورودی خاص به هیچ حالتی نمی‌رود، آن کپی از ماشین دیگر متوقف می‌شود و کار خود را پایان می‌دهد. البته کپی‌های دیگر کارشان را ادامه می‌دهند. این وضعیت برای حالت q_{11} رخ می‌دهد وقتی که ورودی 1 را دریافت کند.

و بالاخره یک ویژگی مهم این ماشین‌ها این است که به طور خود به خود نیز شاخه شاخه می‌شوند. این کیفیت در شکل (۹) وقتی که ماشین در حالت q_{10} است نشان داده شده است. علامت ϵ به این معنای این است که وقتی ماشین به این حالت می‌رسد خود بخود به چند شاخه تبدیل می‌شود و همه شاخه‌ها از جمله همان شاخه قبلی همزمان شروع به کار می‌کنند. بنابراین در اینجا ماشین به طور خود به خود سه شاخه شده است و دو شاخه آن در حالت‌های q_{13} و q_{14} به کار خود ادامه می‌دهند و شاخه فعلی نیز در همان حالت قبلی یعنی حالت q_{10} به کار خود ادامه می‌دهد. باید دقت کنیم که ϵ یکی از حروف الفبای ماشین نیست و در رشته‌هایی که به ماشین وارد می‌شوند هیچ‌کجا علامت ϵ وجود ندارد. این علامت تنها به معنای انشعاب ماشین است وقتی که یک حالت خاص می‌رسد. بعد از این توضیحات آماده‌ایم که ماشین نامتعیین را به صورت دقیق و فرمال تعریف کنیم:

تعریف: یک اتومات محدود و نامتعیین^{۲۵} 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 $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

دقت کنید که اگر چه برای سهولت نوشتار از نماد Σ_ϵ استفاده کرده ایم، بازهم تاکید می کنیم که این نماد جدید در رشته های ورودی به ماشین وجود ندارد و تنها خاصیتی از بعضی حالت های ماشین است که وقتی ماشین در آنها قرار می گیرد کپی هایی از خود را بوجود می آورد.

ج : $\delta : Q \times \Sigma_\epsilon \rightarrow P(Q)$ 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.

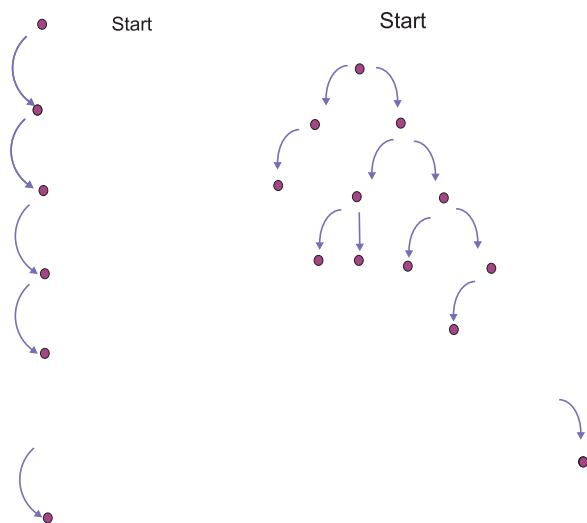
د: q_0 یک حالت ویژه از Q موسوم به حالت آغازین است و

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. بازهم تاکید می کنیم که این نماد جدید در رشته های ورودی به ماشین وجود ندارد و تنها خاصیتی از بعضی حالت های ماشین است که وقتی ماشین در آنها قرار می گیرد کپی هایی از خود را بوجود می آورد. مثلاً عبارت $\delta(q_1, \epsilon) = (q_2, q_3)$ به معنای این است که ماشین وقتی در حالت q_1 قرار گرفت، بلافاصله سه کپی از ماشین درست می شود که هر کدام در یکی از حالت های q_1, q_2, q_3 قرار می گیرند و آماده دریافت ورودی های جدید صفر یا یک از رشته ورودی می شوند تا به طور مستقل به کار خود ادامه دهند.

^{۲۵} Non-deterministic Finite Automata (NFA)

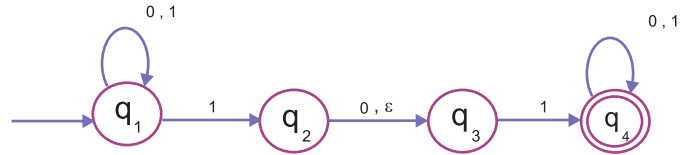


شکل ۱۱: در یک اتومات نامعین، یک رشته مسیره‌های متعددی را طی می‌کند و هر مسیری که به حالت قبولی برسد آن رشته پذیرفته می‌شود. اگر هیچ مسیری به یک حالت قبولی نرسد، آنگاه رشته مورد نظر پذیرفته نخواهد شد.

ثانیاً خروجی تابع δ می‌تواند 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 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 مسیره‌های ممکن را همزمان طی می‌کند و بالاخره هرگاه که خروجی تابع مجموعه‌ی تهی باشد به این معناست که عمل ماشین در این شاخه متوقف می‌شود و به اصطلاح ماشین در این شاخه می‌میرد. بنابراین در دیاگرام حالت یک ماشین نامتعیین از هر دایره که نشان دهنده‌ی یک حالت است می‌تواند هیچ خط، یک خط، یا چندین خط خارج شده باشند. این کیفیت در شکل‌های مربوط به مثال‌ها بخوبی دیده می‌شود.

■ چگونه طرز کار ماشین نامتعیین را دنبال کنیم: می‌دانیم که ماشین نامتعیین در هر لحظه می‌تواند در چند حالت قرار بگیرد. بنابراین طرز دنبال کردن محاسبه در این ماشین نیز مثل ماشین تعیینی است با این تفاوت که ممکن است مجبور شویم از بیش از یک انگشت برای دنبال کردن محاسبه استفاده کنیم. به این ترتیب که انگشتان خود را روی حالت‌های کنونی ماشین قرار دهیم و ببینیم ورودی‌های متمادی هر انگشت ما را به کجا می‌برند. هر موقع که رشته تمام شد چنانچه یکی از انگشت‌های ما روی یکی از حالت‌های قبولی وجود داشته باشد، آن رشته پذیرفته می‌شود و در غیر این صورت رشته رد خواهد شد.

نحوه محاسبه‌ای که در بالا شرح دادیم هرگاه به صورت دقیق درآید به شکل زیر خواهد بود:



شکل ۱۲: این اتومات نامعین رشته‌هایی را می‌پذیرد که یا شامل زیر رشته‌های 11 باشند یا شامل رشته‌های 101. نحوه کار این ماشین به طور دقیق‌تر در شکل (۱۳) نشان داده شده است.

محاسبه توسط ماشین NFA : یک رشته‌ی $s = s_1s_2 \dots s_n$ توسط یک ماشین NFA پذیرفته می‌شود هرگاه زنجیره‌ای از حالت‌های

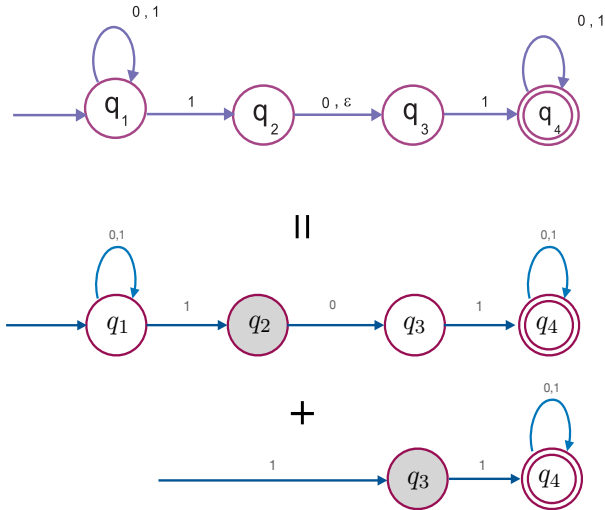
ماشین مثل $r_0, r_1, r_2, \dots, r_n \in Q$ وجود داشته باشند به قسمی که:

$$r_0 = q_0 - ۱$$

$$r_{i+1} \in \delta(r_i, w_i) - ۲$$

$$r_n \in F - ۳$$

شکل‌های ۱۲ و ۱۴ نمونه‌هایی از ماشین‌های نامعین و محدود هستند.



شکل ۱۳: طرز کار ماشین شکل (۱۲). وقتی که ماشین به حالت q_2 می رسد، دو ماشین پایینی به طور موازی شروع به کار می کنند. ماشین بالایی رشته های شامل 101 و ماشین پایین رشته های شامل 11 را شناسایی می کند.

■ تمرین: ماشین شکل ۱۲ 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.

00

010

1000

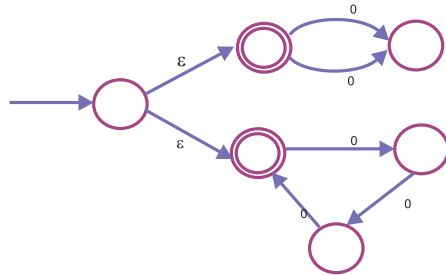
11

101

111

100011

(۲۲)



شکل ۱۴: این اتومات نامعین رشته هایی را می پذیرد که فقط شامل مضربهای ۲ یا ۳ از نماد 0 باشند.

■ 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 $\Sigma = \{0, 1\}$.

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

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

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

$D := 0^*$ with one states

$E := \{w \mid w \text{ contains the substring } 0101\}$ with five state

$F := \{w \mid w \text{ contains either } 000 \text{ or } 111\}$ with the least possible number of states (۲۳)

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 بسیارزادای حالت طراحی کنیم. این امر ضمن آنکه نشان دهنده سادگی و کارایی 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 هر ماشین NFA قابل شبیه سازی توسط یک ماشین DFA است.

■ **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:

$$M' := (\Sigma \cup \epsilon, P(Q), \{q_1\}, \delta', F), \quad (24)$$

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 نامتعین به یک عبارت ریاضی دقیق ترجمه می کنیم. می دانیم که ماشین نامتعین 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 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

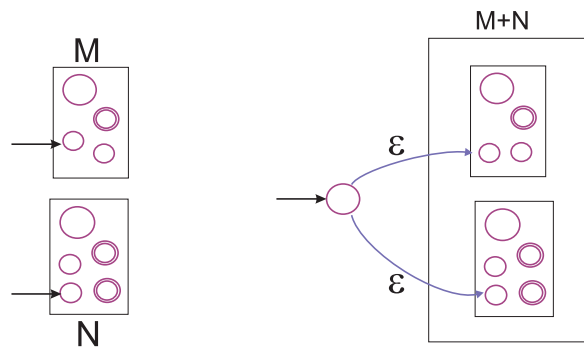
$$\delta'(s, \{q_i, q_j, q_k \dots\}) = \delta(s, q_i) \cup \delta(s, q_j) \cup \delta(s, q_k) \dots. \quad (25)$$

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

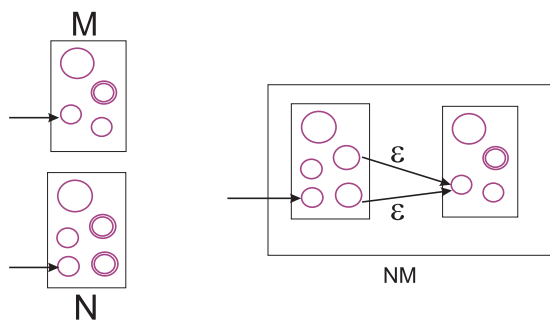
با استفاده از این قضیه می توانیم بسته بودن زبان های منظم تحت اعمال منظم را به آسانی نشان دهیم. برای این کار کافی است که نشان دهیم هرگاه دو زبان A و B منظم باشند، آنگاه ماشین های نامتعینی وجود دارند که بتوانند زبان های $A \cup B$ ، $A \circ B$ و A^* را شناسایی کنند. از آنجا که این ماشین های نامتعین را 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.

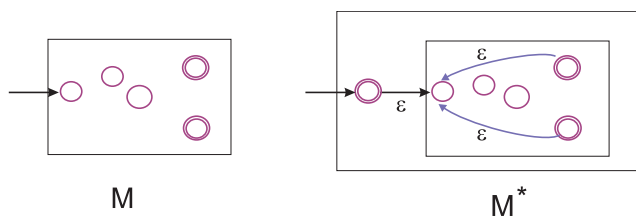
■ **اثبات:** فرض کنید که A و B زبان های منظمی باشند. در این صورت حتماً ماشین هایی مثل M و N وجود دارند که این زبان ها را شناسایی کنند. شکل های ۱۵، ۱۶ و ۱۷ نشان می دهند که چگونه می توان ماشین هایی ساخت که به ترتیب زبان های $A \circ B$ ، $A \cup B$ و A^* را شناسایی کنند. این امر اثبات قضیه را کامل می کند.



شکل ۱۵: نحوه ساختن ماشینی که زبان $A \cup B$ را شناسایی کند.



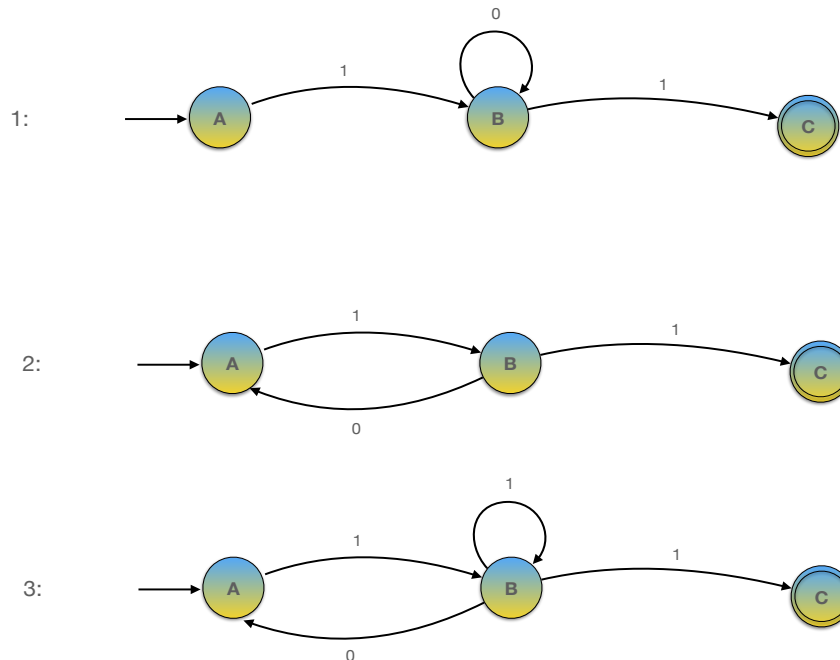
شکل ۱۶: نحوه ساختن ماشینی که زبان $B \circ A$ را شناسایی کند.



شکل ۱۷: نحوه ساختن ماشینی که زبان A^* را شناسایی کند.

۷ عبارت های منظم

از این به بعد هر وقت از ماشین صحبت می کنیم منظور ما یک ماشین متعین است. این کار از این جهت درست است که می دانیم نهایتاً هر ماشین نامتعین را می توانیم توسط یک ماشین متعین شبیه سازی کنیم. فرض کنید که یک ماشین خاص را، مثلاً با دیاگرام حالت آن به شما نشان می دهند و از شما می پرسند که این ماشین خاص چه نوع زبانی را شناسایی می کند؟ یا به عبارت دیگر می پرسند این ماشین خاص چه نوع مسئله ای را حل می کند؟ چگونه می توانید به این سوال پاسخ دهید؟ به عنوان مثال های ماشین نشان داده شده در شکل (۱۸) را در نظر بگیرید. سوال این است که این ماشین ها چه نوع زبانی را شناسایی می کنند؟



شکل ۱۸: این ماشین ها چه نوع زبانی را شناسایی می کنند؟

با کمی تمرین می توان فهمید که هرکدام از این ماشین ها چه زبانی را شناسایی می کنند. این زبان ها را به ترتیب با L_1 ، L_2 و L_3 نشان می دهیم. در این صورت می توانیم با کمی تمرین خود را قانع کنیم که زبان های شناسایی شده این ها هستند:

$$L_1 = \{11, 101, 1001, 10001, 100..001, \dots\}$$

$$L_2 = \{11, 1011, 101011, 10101011, \dots\}$$

$$L_3 = \{11, 1011, 10111, 101111, 1010111, 101110111, \dots\}. \quad (26)$$

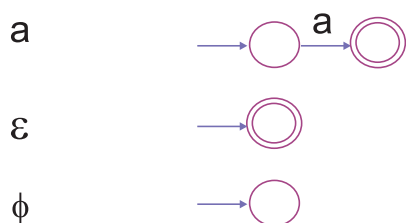
تا کنون زبان های منظم و اعمال آنها را معرفی کرده ایم. در این بخش می خواهیم روشی را یاد بگیریم که با استفاده از آن یک زبان منظم را به صورت کوتاه معرفی کنیم.

یک زبان A را با یک عبارت کوتاه R نشان می دهیم. R را یک عبارت منظم 26 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.

$$\begin{aligned} \Sigma^*\Sigma^* &= \text{all strings of even length} \\ a\Sigma^*b &= \text{all strings beginning with } a \text{ and ending with } b \\ (0U1)0^* &= \text{all strings which are purely made of } 0 \text{ or have only one single } 1 \text{ at their left} \\ \Sigma^*1\Sigma^* &= \text{all strings which have at least a } 1 \\ a^*b^* &= \text{all strings in which all } a\text{'s precede } b\text{'s,} \\ a^*b^* \cup b^*a^* &= \text{all strings in which either all } a\text{'s precede all } b\text{'s or vice versa,} \\ \Sigma\Sigma^*\Sigma^* &= \text{all strings whose length is odd.} \end{aligned} \quad (27)$$

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 ۱۹ 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

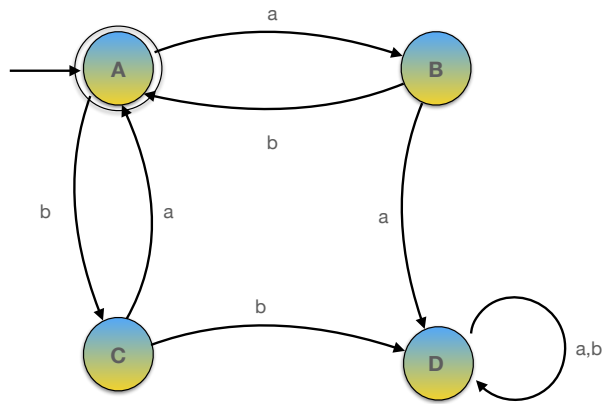
²⁶Regular Expression



شکل ۱۹: عناصر اصلی ای برای تناظر عبارت های منظم و اتومات های محدود.

های DFA یعنی با اعمال منظم \circ , U و $*$ می توانیم ماشین های DFA متناظر با هر عبارت منظمی را بسازیم. شکل ۲۳ نمونه هایی از این نحوه ساخت را نشان می دهد.

البته قواعد نشان داده شده در این شکل به تنهایی نمی توانند نشان دهند که چگونه از یک دیاگرام حالت عبارت منظم مربوطه را استخراج کنیم. به همین دلیل در زیر این کار را به صورت یک الگوریتم عرضه می کنیم و با ذکر یک مثال آن را توضیح می دهیم. دیاگرام حالت نشان داده شده در شکل (۲۰) در نظر بگیرید.



شکل ۲۰: شیوه تعیین زبان منظم مربوط به این ماشین به صورت یک الگوریتم توضیح داده شده است.

نخست یک قرارداد را می پذیریم و آن اینکه از نماد + برای اجتماع استفاده می کنیم. این الگوریتم شامل مراحل زیر است:

یک- هر حالتی از ماشین را در نظر می گیریم و توجه می کنیم که از کدام حالت ها و با چه حرف الفبایی به آن وارد شده ایم. این کار را

برای همه حالت ها انجام می دهیم ، مطابق با مثال زیر:

$$A = Bb + Ca + \epsilon$$

$$B = Aa$$

$$C = Ab$$

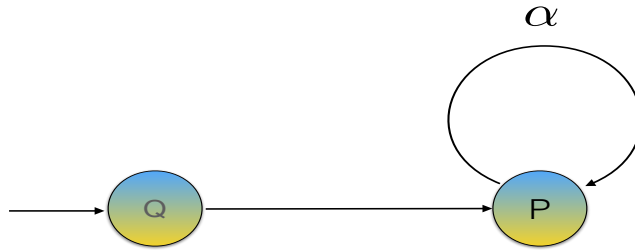
$$D = Ba + Cb + Da + Db. \quad (۲۸)$$

حال سعی می کنیم این عبارتها را ترکیب کنیم و نهایتاً حالت قبولی یعنی D را به صورت یک عبارت بر حسب حروف الفبا و حالت ورودی یعنی

A بنویسیم. اما در این راه از یک اتحاد باید استفاده کنیم و آن اتحاد این است:

$$P = Q + P\alpha \quad \rightarrow \quad P = Q\alpha^* \quad (29)$$

این اتحاد خیلی مهم قضیه آردن^{۲۷} نام دارد. در واقع دلیل شهودی این رابطه دیاگرام (۲۱) است.



شکل ۲۱: این دیاگرام و اتحاد (۲۹) بیان می کند که هم حالت Q به P می رسد و هم حالت های $Q\alpha, Q\alpha^2, Q\alpha^3, \dots$.

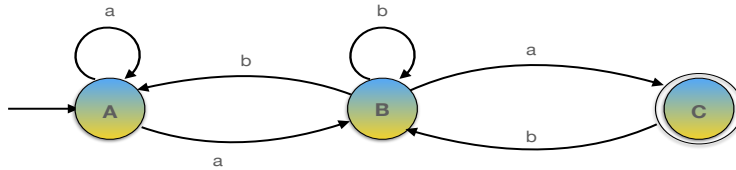
حال سعی می کنیم که روابط (۲۸) را ترکیب و خلاصه کنیم. با استفاده از قضیه آردن بدست می آوریم:

$$A = \epsilon + Aab + Aba = \epsilon + A(ab + ba) = \epsilon(ab + ba)^* = (ab + ba)^* \quad (30)$$

خواننده از نظر شهودی می تواند خود را قانع کند که چرا عبارت منظم وابسته به این دیاگرام حالت چنین است.

■ **یک مثال دیگر:** دیاگرام حالت نشان داده شده در شکل (۲۲) را در نظر بگیرید.

Theorem Arden's^{۲۷}



شکل ۲۲: دیاگرام حالت مربوط به مثال این صفحه.

برای این دیاگرام حالت به همان شیوه می نویسیم:

$$A = Aa + Bb + \epsilon$$

$$B = Aa + Bb + Cb$$

$$C = Ba. \quad (۳۱)$$

از رابطه اول بدست می آوریم:

$$A = (Bb + \epsilon)a^*$$

و سپس با جایگذاری در رابطه دوم

$$B = (Bb + \epsilon)a^*a + Bb + Cb = Bba^*a + a^*a + Bb + Cb = B(ba^*a + b) + Cb + a^*a$$

و با استفاده از قضیه آردن:

$$B = (Cb + a^*a)(ba^*a + b)^* \quad (۳۲)$$

با استفاده از رابطه سوم:

$$B = Bab(ba^*a + b)^* + (a^*a)(ba^*a + b)^* \quad (۳۳)$$

و با استفاده دوباره از قضیه آردن:

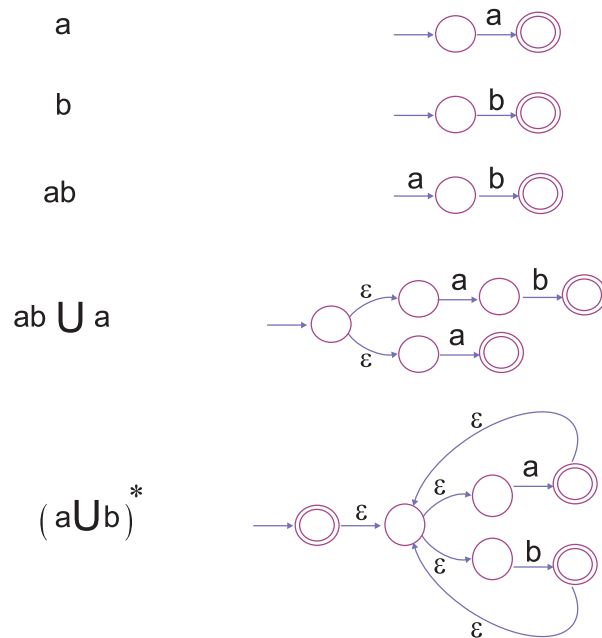
$$B = a^* a (ba^* a + b)^* [(ab)(ba^* + b)]^* \quad (۳۴)$$

و سرانجام جواب نهایی برابر می شود با:

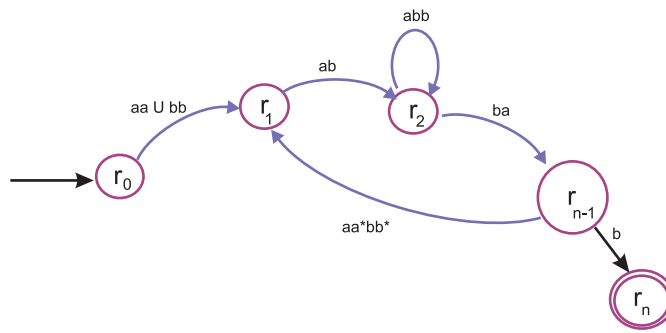
$$C = a^* a (ba^* a + b)^* [(ab)(ba^* + b)]^* a. \quad (۳۵)$$

■ **تمرین:** با استفاده از عبارت فوق چهار نوع از رشته هایی که توسط این ماشین پذیرفته می شوند را مشخص کنید و سپس روی دیاگرام حالت نشان دهید که چرا این ماشین این رشته ها را قبول می کند. آیا رشته های زیر جزئی از زبان بیان شده توسط این عبارت منظم هستند؟ آیا این ماشین رشته های زیر را می پذیرد؟

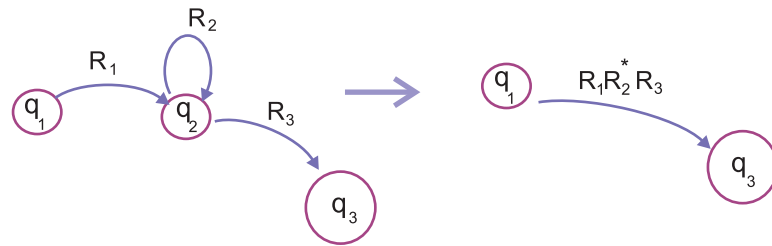
$$a^2 \quad a^n a, \quad a^2 b^2 a. \quad (۳۶)$$



شکل ۲۳: مثال هایی از عبارت های منظم و اتومات های متناظر با آنها.

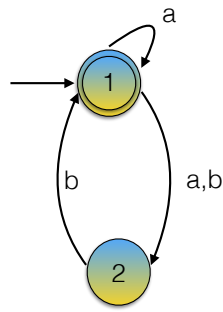


شکل ۲۴: نمونه‌ای از یک اتومات تعمیم یافته نامتعیین $GNFA$.

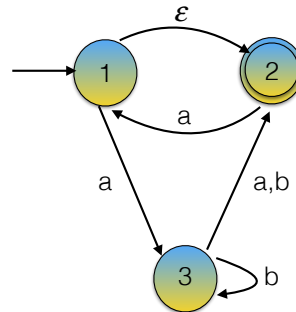


شکل ۲۵: روشی که تعداد حالت های یک $GNFA$ را تقلیل داده و عبارت منظم متناظر با آن را پیدا می کنیم.

بنابراین نشان داده‌ایم 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 نامتعیین یا $GNFA$ ^{۲۸} 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 ۲۴ 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 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 عبارتی است که ماشین DFA اولیه را بیان می کند. این فرایند در شکل ۲۵ 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.



(A)



(B)

شکل ۲۶: مربوط به تمرین ها: این دو ماشین نامتعیین را به ماشین های متعین تبدیل کنید.

ماشین های نامتعیین نشان داده شده در شکل (۲۶) 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. $\{w \mid w \text{ has at least two a's and at least three b's.}\}$
- b. $\{w \mid w \text{ has exactly two a's and at least three b's.}\}$
- c. $\{w \mid w \text{ has exactly two two a's and exactly two b's.}\}$
- d. $\{w \mid w \text{ has at least two consecutive a's and at least three consecutive b's.}\}$
- e. $\{w \mid w \text{ has an even number of a's and each a is followed by at least one b.}\}$
- f. $\{w \mid w \text{ has an even number of a's and each a is followed by exactly one b.}\}$
- g. $\{w \mid w \text{ starts with a and ends with b.}\}$
- h. $\{w \mid w \text{ starts with aaa and ends with bbb.}\}$
- k. $\{w \mid w \text{ starts with aa and has at least one b.}\}$
- l. $\{w \mid w \text{ has even length and has an odd number of b's.}\}$ (۳۷)

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

$$\begin{aligned}
 & a^*b^* \\
 & \Sigma^*aa\Sigma^*bb \\
 & a^* \cup b^* \\
 & (aaa)^* \cup (bbb)^* \\
 & (aUab)\Sigma^* \\
 & (\epsilon Ua)b^* \\
 & (a \cup b)^*a^*(a \cup b)^* \\
 & a^*b^* \cup b^*a^*.
 \end{aligned} \tag{۳۸}$$

۸ 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 الگوریتم 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 تعداد 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:

$$A = \{0^n1^n \mid n > 0\}. \tag{۳۹}$$

برای شناسایی رشته های مربوط به این زبان، یک ماشین باید بتواند نخست تعداد 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 در غیر 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:

$$B = \{w \in \Sigma^* \mid w \text{ contains equal number of } 0's \text{ } 1's\}. \tag{۴۰}$$

چنین زبان هایی را زبان های نامنظم^{۲۹} می گویند. یک نمونه دیگر از مسئله ای که یک ماشین تعینی محدود قادر به حل کردن آن نیست مسئله ی تست کردن پرانتز هاست^{۳۰}. 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 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 پمپ کردن^{۳۱} مشهور است.

■ **لم پمپ کردن:** اگر A یک زبان منظم باشد، آن گاه یک عدد مثبت p موسوم به طول پمپ کردن^{۳۲} وجود دارد به قسمی که هر رشته ی 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 $i \geq 0$ ، $xy^iz \in A$ ،

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 لم پمپ 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 رشته ای مثل xy^iz (برای یک i) وجود دارد که متعلق به آن زبان نیست.

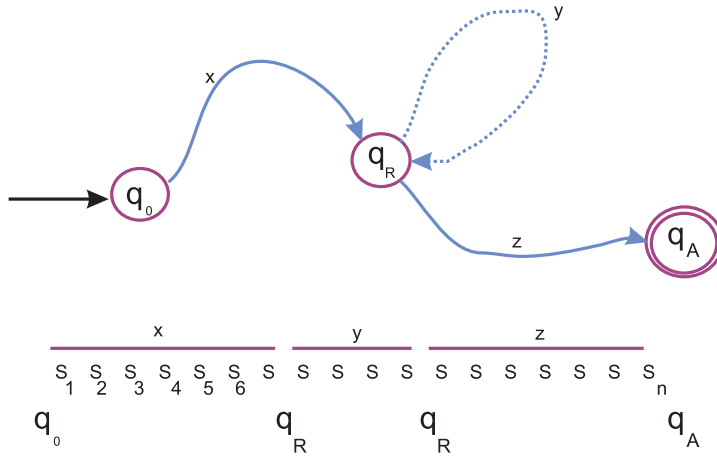
قبل از استفاده از این لم به اثبات آن توجه می کنیم که چنان که خواهیم دید ساده است.

^{۲۹}Non-Regular

^{۳۰}Paranthesis Cheker Problem

^{۳۱}Pumping Lemma

^{۳۲}Pumping Length



شکل ۲۷: لم پمپینگ. اگر این ماشین رشته xyz را بپذیرد، حتماً رشته های $xy^i z$ را هم می پذیرد. ضمناً $|y| \geq 0$ و $|xy| \leq p$.

اثبات لم پمپ کردن: هرگاه A یک زبان منظم باشد، توسط یک ماشین تعیین 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 $s = s_1 s_2 s_3 \dots s_n$ ، $n > p$ 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:

$$\begin{matrix}
 s_1 & s_2 & s_3 & s_4 & s_5 & \dots & \dots & s_n \\
 q_0 & q_{i_1} & q_{i_2} & q_{i_3} & q_{i_5} & \dots & \dots & q_{accept}
 \end{matrix} \quad (41)$$

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 ۲۷، 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 ۲۷ واضح است که هر رشته‌ای مثل $xy^i z$ 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 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 ۳۹ 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 قسمت xyz تقسیم کنیم که در شرایط $|y| > 0$ و $|xy| \leq p$ صدق کند، باز هم بعضی از رشته های $xy^i z$ خارج از زبان A خواهند افتاد. رشته‌ای مثل $0^p 1^p$ را در نظر بگیرید. هر طور تقسیمی که بخواهد شرط $|xy| \leq p$ 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 تکرار y باعث افزایش 0 ها خواهد شد بدون اینکه تعداد 1 ها افزایش یابد. در نتیجه رشته های $xy^i z$ خارج از زبان خواهند افتاد. بنابراین زبان A یک زبان نامنظم است.

همین استدلال نشان می دهد که زبان B در \mathcal{C}_0 نیز نامنظم است.

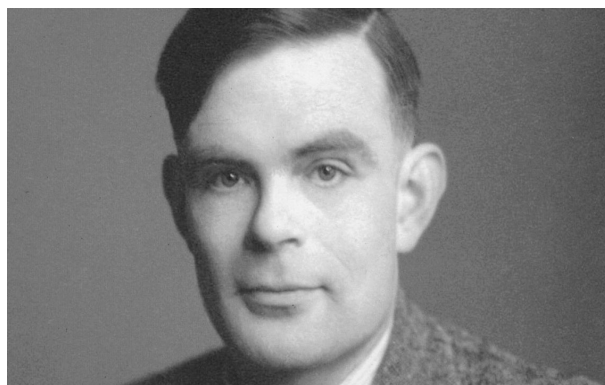
■ از لم پمپینگ استفاده کنید و نشان دهید که زبان های زیر منظم نیستند:

$$A = \{0^n 1^{2n}\}$$

$$B = \{www \mid w \in a, b^*\}$$

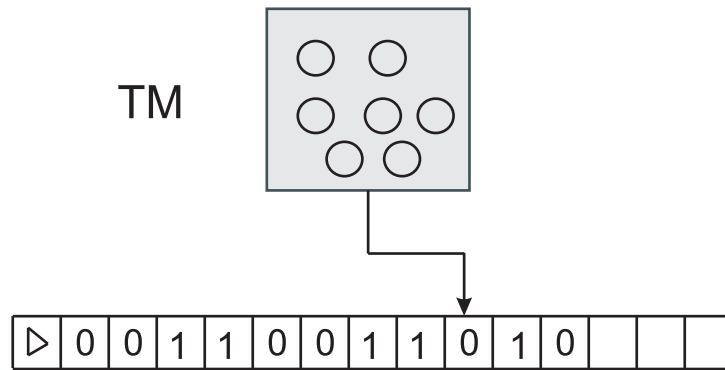
$$C = \{0^n 1^n 2^n \mid n \geq 0\} \quad (۴۲)$$

۹ ماشین تورینگ



شکل ۲۸: آلن تورینگ (۱۹۱۲-۱۹۵۴ م):

دیوید هیلبرت آلن تورینگ، ریاضیدان و دانشمند علوم کامپیوتر انگلیسی است که او را بنیانگذار علوم نظری کامپیوتر و هم چنین هوش مصنوعی می دانند. البته شهرت او نزد عامه مردم به خاطر نقش مهم اش در پروژه رمزگشایی برای سیستم رمزنگاری پیچیده انیگماست که توسط قوای آلمان در جنگ دوم جهانی استفاده می شد. تخمین زده می شود که تلاش های او برای این رمزگشایی که نهایتاً به شکست آلمان ها منجر شد باعث کوتاه شدن جنگ جهانی دوم به مدت دوسال شده که در اثر آن جان حدود ۲۵ میلیون انسان از مرگ در اثر جنگ نجات



شکل ۲۹: ماشین تورینگ.

یافته است.

Turing Alan^{۳۳}

محدودیت اساسی اتومات های تعینی یا نامتعین نداشتن 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 الگوریتم ارائه دهند. اگر مفهوم الگوریتم 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 الگوریتم 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. بیا 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 الگوریتم 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 ترتیب ماشینی درست می شود که آن را ماشین تورینگ می خوانیم ۲۹.

ماشین تورینگ یک قلم ۳۳ 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 ،

Head^{۳۳}

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 $Q = \{q_1, q_2, \dots, q_N\}$ 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} ^{۳۴} هرگاه که ماشین در حالت q_{Halt} قرار بگیرد، اجرای برنامه متوقف می شود. توابع انتقال ماشین تورینگ به صورت زیر عمل می کنند:

$$\delta : Q \times \{b\} \cup \{\triangleright\} \cup \Sigma \rightarrow Q \times \{b\} \cup \Sigma \times \{L, R\}. \quad (۴۳)$$

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

$$(q, s) \rightarrow (q', s', \alpha) \quad (۴۴)$$

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

$$(q, s, q', s', \alpha) \quad (۴۵)$$

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 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 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 حرف 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 حالت q_{Halt} می رسد، محاسبه متوقف می شود و رشته‌ای که روی نوار نوشته شده است جواب ماشین تلقی خواهد شد. در اینجا ماشین تورینگ را توصیف کرده‌ایم که خروجی اش یک رشته است، یعنی ماشین تورینگ چیزی را محاسبه می کند. ماشین تورینگ که در باره 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

^{۳۴} حرف h از کلمه‌ی $Halt$ به معنای توقف آمده است.

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.

۱.۹ انواع ماشین تورینگ

آنچه که در بالا توصیف کردیم، ساختمان یک ماشین تورینگ تعینی 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:

$$(q, s, q', s', \alpha \in -1, 0, 1), \quad (46)$$

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:

$$(q, s_1, s_2; q', s'_1, s'_2, \alpha_1, \alpha_2), \quad (47)$$

که در آن همه علامت ها تعبیر روشنی دارند.

هم چنین می توان ماشین تورینگ نامتعین 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.

۲.۹ 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 آیا ماشین تورینگ کار خواسته شده را انجام می دهد یا خیر.

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

$$\begin{aligned}(q_s, \triangleright, q_1, \triangleright, +1) \\ (q_1, 0, q_1, b, +1) \\ (q_1, 1, q_1, b, +1) \\ (q_1, b, q_{Halt}, b, +1).\end{aligned}\tag{۴۸}$$

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.

$$\begin{aligned}(q_s, \triangleright, q_1, \triangleright, +1) \\ (q_1, 0, q_1, 1, +1) \\ (q_1, 1, q_1, 0, +1) \\ (q_1, b, q_{Halt}, b, +1).\end{aligned}\tag{۴۹}$$

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

$$\begin{aligned}(q_s, \triangleright, q_1, \triangleright, +1) \\ (q_1, 0, q_2, 1, +1) \\ (q_1, 1, q_2, 1, +1) \\ (q_2, 0, q_2, b, +1) \\ (q_2, 1, q_2, b, +1) \\ (q_2, b, q_{Halt}, b, +1).\end{aligned}\tag{۵۰}$$

در مثال هایی که تا کنون شرح داده‌ایم تمام توابع انتقال ماشین را به تفصیل نوشته‌ایم. این نحوه‌ی توصیف همواره لازم نیست و می‌توان شرح خلاصه‌تری از طرز کار ماشین تورینگ نوشت. همواره می‌توان در صورت لزوم این برنامه خلاصه شده را که به زبان روزمره نزدیک ترند و اصطلاحاً یک زبان سطح بالا^{۳۵} به همان زبان بنیادی برحسب توابع انتقال ماشین تورینگ یعنی بر حسب زبان سطح پایین^{۳۶} نوشت. به عنوان مثال یک خط از برنامه مثل « قلم ماشین روی نوار به سمت راست حرکت می‌کند تا به پایان رشته یعنی به حرف b برسد و در این نقطه حالت ماشین از q_s به q_1 تغییر می‌کند» را در نظر می‌گیریم. این خط از برنامه را می‌توان به صورت زیر برحسب توابع انتقال نوشت.

$$\begin{aligned} & (q_s, \triangleright, q_s, \triangleright, +1) \\ & (q_s, 0, q_s, 0, +1) \\ & (q_s, 1, q_s, 1, +1) \\ & (q_s, b, q_1, b, 0). \end{aligned} \quad (51)$$

به این ترتیب اغلب اوقات برنامه ماشین تورینگ را می‌توانیم به زبان ساده‌تری بنویسیم. مثال‌های زیر این موضوع را روشن می‌کنند.

مثال: الگوریتمی می‌خواهیم که بتواند یک رشته را وارون کند به این معنا که رشته‌ی $s_1s_2 \dots s_n$ را به رشته‌ی $s_1s_2 \dots s_{n-1} \dots s_2s_1$ تبدیل کند. این کار را می‌توان با یک ماشین تورینگ دو نوازی انجام داد. رشته‌ی $s_1s_2 \dots s_n$ روی نوار اول نوشته شده است و نوار دوم خالی است. در ابتدا هر دو قلم‌ها در ابتدای نوارها و روی حروف \triangleright هستند. قلم نوار اول روی نوار به طرف راست حرکت می‌کند تا به پایان رشته برسد. پایان رشته را از روی حرف b می‌فهمد. در این لحظه به سمت چپ حرکت می‌کند و هر حرفی را که قلم اول از یک خانه می‌خواند قلم دوم روی یک خانه می‌نویسد و به سمت راست حرکت می‌کند. وقتی که قلم اول به ابتدای نوار اول رسید ماشین به حالت q_{Halt} می‌رود و متوقف می‌شود. 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 می‌گذاریم.

مثال: مسئله تست کردن پرانتزها را می‌توانیم با یک ماشین تورینگ براهتی حل کنیم. برنامه چنین ماشینی به شرح زیر است: الف - قلم به سمت راست می‌رود تا به اولین پرانتز (برخورد کند. این پرانتز را پاک می‌کند.

High Level Language^{۳۵}
Low level language^{۳۶}

ب - قلم به سمت چپ برمی گردد تا به اولین پرانتزِ 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 پرانتزِ 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.

مثال: زبان زیر را در نظر بگیرید:

$$Add := \{x^i y^j z^k \mid k = i + j\}. \quad (52)$$

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 هرکدام از این حرف ها باقی مانده باشند رشته رد می شود.

مثال: زبان زیر را در نظر بگیرید:

$$Multiply := \{x^i y^j z^k \mid k = ij\}. \quad (53)$$

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 سرجایش 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.

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

۳.۹ 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 - تورینگ^{۳۷} شناخته می شود. این تز بیان می کند که هیچ مدل محاسبه‌ای دیگری وجود 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 شاهی بر نقض این تز بدست نیامده است. در واقع تز چرچ - 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 الگوریتم است، (الگوریتم 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 الگوریتمی 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

Church-Turing Thesis^{۳۷}

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؟

۴.۹ 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^{۳۸} 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 ساده‌ی 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 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 ایده‌آل 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 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 محدود دیگر. شمارش پذیر بودن مجموعه‌ی تمام ماشین های تورینگ این امکان را به ما می دهد که به هر ماشین یک عدد طبیعی نسبت دهیم. این عدد، عدد تورینگ^{۳۹} 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 ورودی $x = x_1x_2 \dots x_n$ ، 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:

$$UTM : (i b x) \longrightarrow (i b M_i(x))$$

Universal Turing Machine^{۳۸}
Turing Number^{۳۹}

$$UTM : (j b x) \longrightarrow (j b M_j(x))$$

$$UTM : (k b x) \longrightarrow (k b M_k(x))$$

...

...

(۵۴)

۵.۹ مسئله‌ی توقف

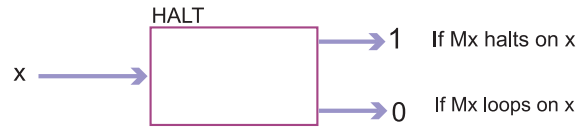
یکی از مفاهیم بسیار عمیق که پیامدهای گسترده‌ای دارد مسئله توقف یا ایستادن^{۴۰} است. برای پاسخ به مسئله هیلبرت در مورد وجود یک الگوریتم که بتواند همه مسائل ریاضیات را حل کند، تورینگ این پرسش را طرح کرد که آیا یک الگوریتم وجود دارد که بتواند بگوید یک ماشین تورینگ M_x روی یک ورودی y توقف می‌کند یا نه؟ برای 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 می‌کند یا نه؟ یادآوری می‌کنیم 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:

$$h(x) = \begin{cases} 1 & \text{if } M_x \text{ halts on } x, \\ 0 & \text{if } M_x \text{ doesn't halt on } x \end{cases} \quad (۵۵)$$

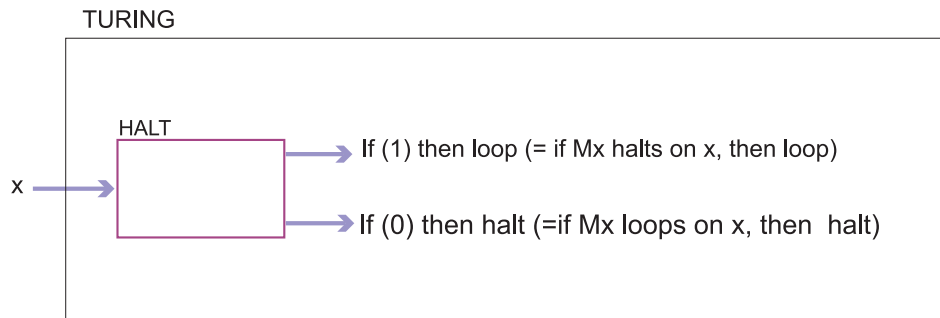
معنای محاسبه پذیر بودن این است که یک ماشین تورینگ (الگوریتم) وجود دارد که می‌تواند این تابع را حساب کند و برای هر ورودی x مقدار $h(x)$ را که یا 0 است یا 1 تعیین کند. این ماشین تورینگ یا الگوریتم را $HALT$ می‌نامیم، شکل ۳۰. حال نشان می‌دهیم که فرض وجود چنین ماشینی به تناقض منطقی می‌انجامد. برای این کار به ترتیب زیر استدلال می‌کنیم. یک برنامه بزرگتر درست می‌کنیم که برنامه‌ی $HALT$ را فراخوانی کند. شکل ۳۱ طرز کار این برنامه را نشان می‌دهد. دقت کنید که این برنامه که آن را $Turing$ می‌نامیم بازهم 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 می‌کنیم که ماشین $Turing$ خودش می‌بایست یک شماره داشته باشد که با عدد صحیح t بیان شود. بنابراین می‌توان نوشت $Turing = M_t$. حال به این ماشین همان عدد t را می‌دهیم. حال همان طور که شکل ۳۲ نشان می‌دهد، به یک تناقض منطقی می‌رسیم. زیرا

بنابراین شکل

Halting Problem^{۴۰}



شکل ۳۰: آگوریتم $HALT$ برای محاسبه تابع h .



شکل ۳۱: اگر آگوریتم یا ماشین تورینگ $HALT$ وجود داشته باشد، آگوریتم یا ماشین تورینگ $Turing$ نیز وجود خواهد داشت.

$$M_t \text{ halts on } t \text{ if } M_t \text{ loops on } t,$$

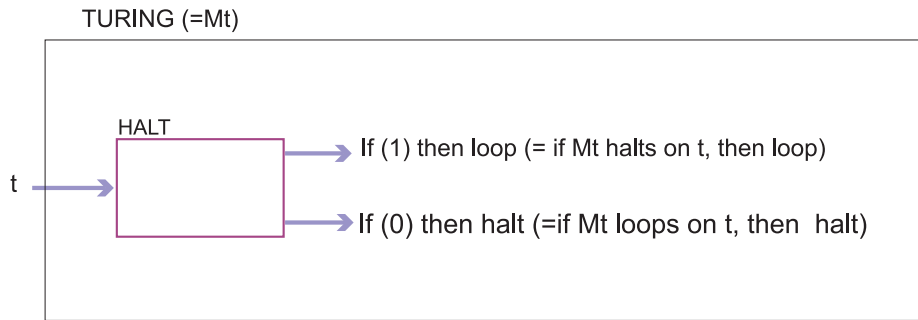
(۵۶)

$$M_t \text{ loops on } t \text{ if } M_t \text{ halts on } t,$$

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

اگر چنین ماشین تورینگی وجود داشته باشد به این معناست که می تواند بگوید آیا یک آگوریتم 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^{۴۱} را در نظر بگیرید. این حدس بیان می کند که هر عدد زوج را حتماً

^{۴۱}Goldbach Conjecture



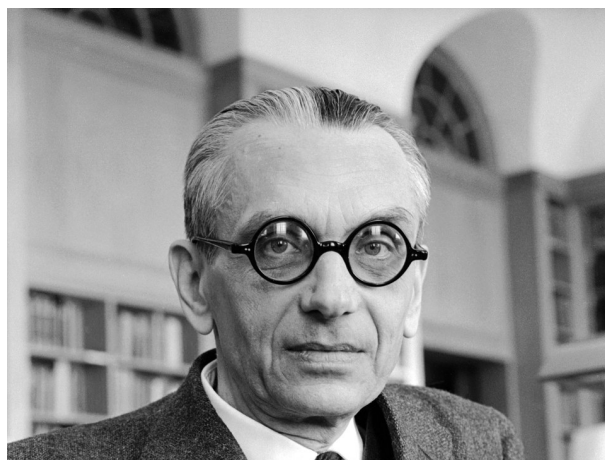
شکل ۳۲: می توانیم به ماشین *Turing* یا همان M_t ورودی t را بدهیم و تناقض منطقی وجود ماشین *Turing* و در نتیجه ماشین *HALT* را ببینیم.

می توان به صورت مجموع دو عدد اول نوشت. کافی است که خواننده به مثال های ساده ی زیر توجه کند:

$$4 = 2 + 2, \quad 6 = 3 + 3, \quad 8 = 3 + 5, \quad 10 = 5 + 5, \quad 20 = 7 + 13, \quad 30 = 23 + 7, \dots \quad (57)$$

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 برنامه ی $M_{Goldbach}$ می نویسیم و این برنامه هر عدد 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 و اگر جواب منفی بود باز هم 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 بسیار بسیار بزرگ چند هزار رقمی اجرا کنیم و می توانیم که مثلاً بگوییم که حدس گلدباخ تا اعداد ۱۰۰۰ رقمی صحیح است. ولی این به این معنی نیست که ما حدس گلدباخ را ثابت کرده ایم. (البته اگر یک عدد نقیض پیدا کنیم آنگاه حدس گلدباخ را نفی کرده ایم.) در واقع زبان این ماشین به این صورت است:

$$L = \{The\ set\ of\ all\ even\ integers\ which\ are\ the\ sum\ of\ two\ prime\ numbers\}. \quad (58)$$



شکل ۳۳: کورت فردریش گودل (۱۹۰۶-۱۹۷۸ م).

کورت فردریش گودل (۱۹۰۶-۱۹۷۸ م).^آ ریاضیدان، منطق دان و فیلسوف تحلیلی اتریشی است که به همراه ارسطو و فرگه مهم ترین منطقی دانان تاریخ به شمار می روند. کار گودل تاثیر عمیقی بر تفکر علمی و فلسفی قرن بیستم نهاده است. مهم ترین کار او یعنی دو قضیه مهم او در ناکامل بودن مبانی ریاضیات در بیست و پنج سالگی اش منتشر شده. دو قضیه مهم او به اختصار بیان می کنند که :

یک - در هر سیستم اصل موضوعی سازگار همواره قضایایی وجود خواهند داشت که درستی یا نادرستی آن ها را نمی توان بر اساس آن اصول موضوع ثابت کرد.

دو- سازگاری یک مجموعه اصول موضوع را نمی توان با استفاده از خود آن اصول موضوع ثابت کرد.

جان فون نویمان در باره قضایای گودل گفته است که این قضایا بناهای رفیعی هستند که از اعماق فضا و زمان قابل رویت هستند. وی قبل از شروع جنگ دوم جهانی به امریکا مهاجرت کرد و عضو دائم موسسه مطالعات پیشرفته در پرنستون شد، جایی که شاهد دوستی ماندگارش با اینشتین و پیاده روی های طولانی این دو طی سالیان دراز شد. اینشتین به یک همکار اقتصاددان در موسسه به نام اسکار مونگسترن^ب گفته است که کار علمی اش در موسسه پایان یافته و تنها دلیل آمدنش به موسسه به شوق همراهی گودل در بازگشت پیاده به خانه است. گودل حوالی سال ۱۹۴۶ یعنی سی و دو سال قبل از مرگش از انتشار مقاله و کتاب دست کشید اگرچه هم چنان فعالیت علمی خود را ادامه داد. در سال ۱۹۵۳ به مقام استادی در موسسه رسید. در سالهای بعدی علایق او متوجه فیزیک و فلسفه شد. در نسبت عام به حل هایی از معادله اینشتین دست یافت که منحنی های بسته زمان گونه و سفر به گذشته را امکان پذیر می کرد. گفته می شود که او این حل را به عنوان هدیه هفتاد سالگی اینشتین به او هدیه کرد. در سالهای آخر عمر گودل به نوعی وسواس و ترس از مسمومیت غذایی شد. وی که تنها غذاهای طبخ

شده توسط همسرش را می خورد به دنبال بستری شدن شش هفته ای همسرش در بیمارستان به دلیل امتناع از خوردن غذا درگذشت.

Godel Friedrich Kurt^{۴۲}
Oskar Morgenstern^{۴۳}

۱۰ 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^{۴۲}، حافظه^{۴۳} و انرژی^{۴۴}. قبلاً در باره منبع انرژی بحث کرده ایم. در این درس دو منبع دیگر یعنی زمان و حافظه را بررسی می کنیم. یک مسئله یا یک الگوریتم معین را در نظر بگیرید. به عنوان مثال می خواهیم ببینیم که آیا در یک گراف یک مسیر هامیلتونی وجود دارد یا نه؟^{۴۵} ورودی این مسئله اطلاعات مربوط به یک گراف است. یک گراف عبارت است از مجموعه ای از نقاط که بعضی از آنها به هم وصل شده اند. 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 دهنده این 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 اندازه ی 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 چقدر است. تعیین این حداقل منابع اهمیت زیادی در نظریه محاسبه دارد. نکته ی 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 گوئیم 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

$$f(n) \text{ is } O(g(n)), \quad \text{if } \exists c \text{ and } n_0 \forall n > n_0 \quad f(n) \leq cg(n), \quad (59)$$

Time^{۴۲}

Space^{۴۳}

Energy^{۴۴}

^{۴۵} این مسئله در بخش های بعدی این درس تعریف شده است.

ب: می‌گوییم 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

$$f(n) \text{ is } \Omega(g(n)), \quad \text{if } \exists c \text{ and } n_0 \quad \forall n > n_0 \quad f(n) \geq cg(n), \quad (60)$$

می‌گوییم الف: می‌گوییم 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

$$f(n) \text{ is } \Theta(g(n)), \quad \text{if } f(n) = O(g(n)), \quad \text{and } f(n) = \Omega(g(n)). \quad (61)$$

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

$$t(L) := \max_{x \in L} t(x). \quad (62)$$

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

۱.۱۰ کلاس P

تعریف: به کلاس مسایلی که یک ماشین تورینگ با زمان اجرای $O(n^k)$ آنها را حل می‌کند، کلاس $DTime(n^k)$ می‌گوییم. 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 مسئله ها یا زبان هاست و نه یک زبان خاص. حال

می توانیم کلاس مسائل P ،^{۴۶} یعنی مسائل چند جمله‌ای را تعریف کنیم:

$$P := \bigcup_{k \geq 0} DTime(n^k). \quad (۶۳)$$

کلاس مسایل P ، مجموعه همه مسایل یا الگوریتم‌هایی است که در زمان چندجمله‌ای 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 داده n که 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 نظایران 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 هرکدام 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 برحسب اندازه مسئله (تعداد رقم‌های عدد اولیه) چند جمله‌ای باشد شناخته نشده است. (در اینجا فقط در باره الگوریتم‌های کلاسیک بحث می‌کنیم.)

۲.۱.۰ 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 ^{۴۷} را تعریف کنیم:

$$NP := \bigcup_{k \geq 0} NDTIME(n^k). \quad (۶۴)$$

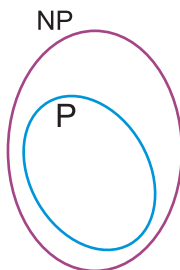
از آنجا که یک ماشین تورینگ معین نوع خاصی ماشین تورینگ نامعین است نتیجه بدیهی تعاریف بالا آن است که $DTIME(n^k) \subset NDTIME(n^k)$ و در نتیجه

$$P \subset NP. \quad (۶۵)$$

هرگاه به شکل ۱۱ 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 کار

^{۴۶} Polynomial Time Problems or Polynomial Time Algorithms

^{۴۷} Non-deterministic Polynomial Time Problems or Algorithms



شکل ۳۴: کلاس های P و NP .

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 چندجمله ای 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، زیرا 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 چندجمله ای 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 ۳۴.

یک تعریف دیگر از کلاس NP که با تعریف بالا معادل است با شهود زیر بدست می آید. عموماً دشواری یک مسئله 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 قضیه کلی ای 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 چندجمله ای 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 48 را که منجر به یافتن آن می شود با y نشان می دهیم. در این صورت زمان لازم برای حل مسئله حتماً چندجمله ای 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

$$\begin{aligned} & \text{if } x \in L \exists y \in L \mid TM \text{ halts in } q_{Accept} \text{ on the input } (x \parallel y) \text{ after time } (Poly(n)), \\ & \text{if } x \notin L \forall y \in L \mid TM \text{ halts in } q_{Reject} \text{ on the input } (x \parallel y) \text{ after time } (Poly(n)). \end{aligned} \quad (66)$$

در اینجا منظور از n اندازه ورودی x است.

⁴⁸Witness

۱۱ 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:

$$R(x) \in B \leftrightarrow x \in A. \quad (۶۷)$$

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 تراگذار^{۴۹} 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 لوین^{۵۰} 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.

^{۴۹}Transitive
^{۵۰}Kook and Levin

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

$$x_{i_1} \vee x_{i_2} \vee x_{i_3} \vee \dots \vee x_{i_K}, \quad (68)$$

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

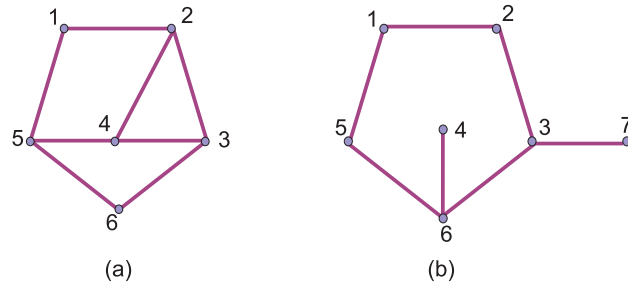
$$S = C_1 \wedge C_2 \wedge C_3 \wedge \dots \wedge C_p. \quad (69)$$

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 $K - Sat$ ^{۵۲} 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:

$$\begin{aligned} C_1 &= 1 \\ C_2 &= 1 \\ &\dots \\ C_p &= 1. \end{aligned} \quad (70)$$

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 همچنان چندانای ایجاد نمی کند. نمونه ای از این مسایل در ادامه می آید.

^{۵۱} Clause
^{۵۲} K-Satisfiability



شکل ۳۵: گراف a هامیلتونی است ولی گراف b هامیلتونی نیست.

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 هامیلتونی^{۵۳} 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^{۵۴}، گرافی است که حداقل یک مسیر هامیلتونی داشته باشد. به عنوان مثال در شکل ۳۵ گراف a هامیلتونی است ولی گراف b هامیلتونی نیست.

۳.۱۱ مسئله فروشنده دوره گرد

یک گراف وزن دار^{۵۵} $G = (V, E)$ ، 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 سائز همان سائز 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 ۴۱ 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 ۳۵ مجموعه $W = \{1, 4, 6\}$ یک پوشش راسی نیست ولی مجموعه $W' = \{1, 2, 4, 6\}$ یک

^{۵۳}Hamiltonian Path

^{۵۴}Hamiltonian Graph

^{۵۵}Weighted graph

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 سایز k دارد یا خیر؟. در شکل ۳۵ 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.

۱۲ کلاس $PSPACE$

برای آنکه منابع حافظه ای ^{۵۶} را برای یک الگوریتم 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 گوئیم یک زبان L در کلاس $PSPACE$ است اگر یک ماشین تورینگ TM وجود داشته باشد به نحوی که

$$\forall x \in L, TM \text{ halts on } x \text{ after the consumption of } P(n) \text{ bits of tape where } n \text{ is the length of } x. \quad (71)$$

رابطه زیر بین این کلاس ها وجود دارد:

$$P \subset PSPACE, \quad NP \subset PSAPCE. \quad (72)$$

Memory Resources^{۵۶}

نامساوی اول واضح است زیرا اگر یک ماشین تعداد مراحل را با گرفتن ورودی x طی می کند تا به حالت پایانی برسد، یک چند جمله‌ای برحسب $|x| = n$ 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 تعداد $O(2^n)$ 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 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 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 بازم تعداد خانه های نوار که تغییر می کنند یک چند جمله‌ای از n خواهد بود. بنابراین نتیجه می گیریم که $NP \subset PSPACE$.

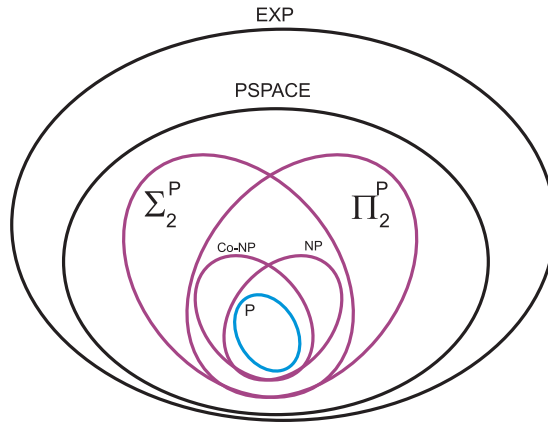
تعریف: کلاس L به کلاس مسائلی گفته می شود که در زمان لگاریتمی توسط یک ماشین تورینگ قابل تصمیم گیری هستند. به یک معنا این کلاس ساده ترین مسائل را در بردارد.

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 زیر است:

$$L \subset P \subset NP \subset PSPACE \subset EXP. \quad (۷۳)$$

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



شکل ۳۶: نمونه‌ای از کلاس‌های پیچیدگی و رابطه آنها.

۱۳ تمرین‌ها:

■ برای هرکدام از زبان‌های زیر یک اتومات تعیین 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:

$$L_1 := \{w \mid w \text{ contains the substring } 1011\}$$

$$L_2 := \{w \mid w \text{ contains an even number of } 0\text{'s and an odd number of } 1\text{'s}\}$$

$$L_3 := \{w \mid w \text{ begins with } 0 \text{ and ends with } 1\}$$

$$L_4 := \{w \mid w \text{ begins with } 00 \text{ and ends with } 11\}$$

$$L_5 := \{w \mid w \text{ is of length } N\}$$

$$L_6 := \{w \mid w \text{ does not contain the substring } 111\}$$

$$L_7 := \{w \mid w \text{ begins and ends with the same letter}\}$$

$$L_8 := \{w \mid w \text{'s length is a multiple of } 3\}$$

$$L_9 := \{w \mid w \text{'s length is a multiple of } 3 \text{ and contains an even number of } 0\text{'s}\}$$

$$L_{10} := \{w \mid w \text{ contains the word } \text{WORK}\}$$

$$L_{11} := \{w \mid w \text{ contains the word } \text{WORK twice}\} \quad (۷۴)$$

■ هرکدام از زبان‌های زیر اشتراک دو زبان ساده تر است. در هر مورد نخست ماشین‌های DFA ای بسازید که آن زبان‌های 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 $\Sigma := \{a, b\}$.

$$\begin{aligned} A &:= \{w \mid w \text{ contains at least two } a\text{'s and at least three } b\text{'s}\} \\ B &:= \{w \mid w \text{ contains at most two } a\text{'s and at least three } b\text{'s}\} \\ C &:= \{w \mid w \text{ contains at most two } a\text{'s and at most three } b\text{'s}\} \\ D &:= \{w \mid w \text{ contains the substring } aba \text{ and the substring } bab\} \\ E &:= \{w \mid w \text{ starts with } aaa \text{ and ends with } bbb\}. \end{aligned} \quad (75)$$

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 غیر تعیینی محدود باشد که زبان A را شناسایی می کند آنگاه با عوض کردن حالت های $Accept$ و حالت های معمولی ماشینی 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 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 یادگرفتید ماشینی بسازید که زبان داده شده را شناسایی کند:

$$\begin{aligned} A &:= \{w \mid w \text{ does not contain the substring } ab\} \\ B &:= \{w \mid w \text{ does not contain the substring } abab\} \\ C &:= \{w \mid w \text{ contains neither the substring } ab \text{ nor } ba\} \\ D &:= \{w \mid w \text{ is any string not in } a^*b^*\} \\ E &:= \{w \mid w \text{ is any string that does not contain exactly two } a\text{'s}\}. \end{aligned} \quad (76)$$

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

$$\begin{aligned}
A &:= 0^*1^* \\
B &:= \sum 0 \sum \\
C &:= \sum 00 \sum \\
D &:= 0^*(10^*)1 \\
E &:= \sum 0^* \sum \\
F &:= (\sum \sum)^* \cup (\sum \sum \sum)^* \cup (01)^*. \tag{۷۷}
\end{aligned}$$

■ 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.

■ عبارت های منظم زیر را به NFA تبدیل کنید:

$$\begin{aligned}
&(0 \cup 1)^*000(0 \cup 1)^* \\
&(((00)^*(11)) \cup (01))^* \\
&0^*. \tag{۷۸}
\end{aligned}$$

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

$$\begin{aligned}
A &= \{0^n 1^n 2^n \mid n \geq 0\} \\
B &= \{www \mid w \in \{a, b\}^*\} \\
C &= \{a^{2^n} \mid n \geq 0\}. \tag{۷۹}
\end{aligned}$$

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