نظریه‌ی زبان‌ها و ماشین‌ها

درس‌نامه‌ی ۱۳

کاظم فولادی
http://kazim.fouladi.ir

ویراست اول: ۱۳۸۵
ویراست دوم: ۱۳۸۷
ویراست سوم: ۱۳۹۳

فهرست مطالب

۱۳ مدل‌های دیگر محاسبات

OTHER MODELS OF COMPUTATION

غیر از ماشین تورینگ، مدل‌های دیگری نیز برای محاسبه وجود دارند.
این مدل‌ها در نگاه اول به طور ریشه‌ای متفاوت با ماشین تورینگ به نظر می‌رسند، اما مشخص شده است که همه‌ی این مدل‌ها معادل هستند.
بیشتر کارهای اولیه در این زمینه مربوط به سال‌های ۱۹۳۰ تا ۱۹۴۰ است.
نتایج به دست آمده نه تنها بر مفهوم محاسبات مکانیکی بلکه بر مفهوم کل ریاضیات تاثیر گذاشت.
کار تورینگ در سال ۱۹۳۶ منتشر شد که در آن زمان هیچ کامپیوتر تجاری وجود نداشت.
اگرچه کار تورینگ سرانجام در علم کامپیوتر بسیار مهم شد، اما هدف اصلی او فراهم کردن اصولی برای مطالعه‌ی کامپیوترهای دیجیتال نبود. آنچه وی به دنبال آن بود، به وضعیت ریاضیات در آن دوران برمی‌گردد.
یافتن یک سیستم ریاضی قابل اثبات، هدف اصلی ریاضیدان‌ها در اواخر قرن نوزدهم میلادی بوده است.
این هدف با دو نگرانی همراه بود:

- **سازگاری (consistency)**. نباید گزاره‌ای موجود باشد که با یک دنباله از گام‌ها درستی و با دنباله‌ای دیگر نادرستی آن اثبات شود.
- **کامل بودن (completeness)**. درستی یا نادرستی هر گزاره‌ی قابل بیان در سیستم باید قابل اثبات باشد.

قضیه‌ی کامل نبودن گودل بیان می‌کند که هر سیستم سازگار مورد نظر باید ناکامل باشد (یعنی باید شامل گزاره‌هایی غیر قابل اثبات باشد) (1931).
پرسشی که توسط گودل پاسخ داده نشد این است که آیا گزاره‌های قابل اثبات از گزاره‌های غیر قابل اثبات قابل تشخیص هستند؟ در واقع، هنوز برای ریاضیدان‌ها امیدهایی وجود داشت که بتوانند درستی اثبات را به طور مکانیکی وارسی کنند.
برای پاسخ به این سؤال مدل‌های صوری مختلفی توسط تورینگ، چرچ، پُست، کِلین و دیگران ارائه شد.

تز چرچ: همه‌ی مدل‌های محاسباتی که به اندازه‌ی کافی گسترده باشند، معادل هستند.

تز چرچ ـ تورینگ: همه‌ی مدل‌های محاسباتی که به اندازه‌ی کافی گسترده باشند، می‌توانند هر تابع محاسبه‌پذیری را محاسبه کنند و لذا معادل با ماشین تورینگ هستند.

۱۳-۱ توابع بازگشتی

روش‌های مختلفی برای تعریف توابع وجود دارد که برخی از آنها بسیار متداول است و برخی دیگر کمتر مورد استفاده قرار می‌گیرد. یک روش آشنا استفاده از نمادگذاری تابعی مانند $f(n) = n^۲ + ۱$ است.

۱۳-۱-۱ توابع بازگشتی ابتدایی

برای سادگی بحث توابع یک یا دو متغیره‌ای را در نظر می‌گیریم که دامنه و برد آنها I، مجموعه‌ی اعداد صحیح نامنفی، است.

- **تابع صفر (zero)**، z، که برای هر $x \in I$، $z(x) = ۰$ است.
- **تابع مابعد (successor)**، s، که برای هر $x \in I$، عدد بعدی در دنباله را برمی‌گرداند، یعنی $s(x) = x + ۱$ است.
- **تابع تصویرگر (projector)**، p_k برای $k = ۱, ۲$، که $p_k(x_۱, x_۲) = x_k$ است.

دو راه برای ساخت توابع پیچیده‌تر از روی این توابع وجود دارد:

- **ترکیب (composition)**، که در آن f از روی $g_۱$، $g_۲$ و h تعریف شده، ساخته می‌شود:

$$f(x, y) = h(g_۱(x, y), g_۲(x, y))$$

- **بازگشت ابتدائی (primitive recursion)**، که به وسیله‌ی آن یک تابع می‌تواند به صورت بازگشتی تعریف شود:

$$\begin{aligned} f(x, ۰) &= g_۱(x) \\ f(x, succ(y)) &= h(g_۲(x, y), f(x, y)) \end{aligned}$$

مثال

تابع جمع دو عدد صحیح نامنفی $add(x, y)$:

$$\begin{aligned} add(x, ۰) &= x \\ add(x, y + ۱) &= add(x, y) + ۱ \end{aligned}$$

مثال

تابع ضرب دو عدد صحیح نامنفی $mult(x, y)$

$$\begin{aligned} mult(x, ۰) &= ۰ \\ mult(x, y + ۱) &= add(x, mult(x, y)) \end{aligned}$$

توجه کنید که $p_1(x,y) = x$ و $z(x) = 0$ توابع ابتدائی هستند.

مثال

عمل تفریق حسابی (monus)

$$x \dot{-} y = \begin{cases} x - y & , x \geq y \\ 0 & , x < y \end{cases}$$

با تعریف تابع ماقبل $pred(0) = 0$ و $pred(y+1) = y$ برای تعریف تابع تفریق حسابی داریم:

$$\begin{aligned} subtr(x, 0) &= x \\ subtr(x, y+1) &= pred(subtr(x, y)) \end{aligned}$$

تعریف یک تابع بازگشتی ابتدائی نامیده می‌شود اگر و فقط اگر بتواند از روی توابع پایه‌ی صفر (z)، مابعد (s) و تصویرگر (p_k) به وسیله‌ی ترکیب و بازگشت ابتدائی ساخته شود.

◄ **تذکر** اگر g_1، g_2 و h توابع تام باشند، در این صورت f تعریف شده با ترکیب و بازگشت ابتدائی نیز بر روی I یا $I \times I$ تام خواهد بود.
اکثر توابع متداول بازگشتی ابتدائی هستند، با این وجود برخی توابع وجود دارند که بازگشتی ابتدائی نمی‌باشند.

قضیه فرض کنید که F مجموعه‌ی همه‌ی توابع از I به I باشد. در این صورت تابعی در F وجود دارد که بازگشتی ابتدائی نیست.

◄ اثبات.
با روش قطری‌سازی انجام می‌شود. هر تابع بازگشتی ابتدائی با یک رشته‌ی متناهی روی الفبای شامل اعداد، عملگرها و علائمی مانند پرانتز باز و بسته بازنمایی می‌شود. از آنجایی که مجموعه‌ی همهِ‌ی رشته‌ها روی این الفبا شمارا است و با یک الگوریتم تجزیه می‌توان تعیین کرد که کدام رشته تابع بازگشتی ابتدائی را نشان می‌دهد و کدام خیر، پس مجموعه‌ی همه‌ی توابع بازگشتی ابتدائی، یک مجموعه‌ی شمارا است.
حال فرض می‌کنیم مجموعهِ‌ی همه‌ی توابع ممکن F، شمارا باشد، پس می‌توان ترتیبی چون

$$f_1, f_2, f_3, \ldots$$

را برای این توابع در نظر گرفت. سپس تابع g را به صورت زیر می‌سازیم:

$$g_i(i) = f_i(i) + 1 \qquad , i = 1, 2, \ldots$$

مشخص است که g خوش‌تعریف است و لذا در F قرار دارد، اما g با همه‌ی f_iها در موقعیت قطری متفاوت است. این تناقض نشان می‌دهد که F نمی‌تواند شمارا باشد. ترکیب دو نتیجه‌ی اخیر ثابت می‌کند که تابعی در F باید موجود باشد که بازگشتی ابتدائی نیست.

نه تنها توابعی وجود دارند که بازگشتی ابتدائی نیستند، بلکه توابع محاسبه‌پذیری وجود دارند که بازگشتی ابتدائی نمی‌باشند.

قضیه فرض کنید که C مجموعه‌ی همه‌ی توابع محاسبه‌پذیر تام از I به I باشد. در این صورت تابعی در C وجود دارد که بازگشتی ابتدائی نیست.

◄ **اثبات.**

بر اساس بحث انجام شده در قضیه‌ی قبلی، مجموعه‌ی همه‌ی توابع بازگشتی ابتدائی شمارا است. این توابع را به صورت

$$r_۱, r_۲, r_۳, \ldots$$

نامگذاری می‌کنیم و تابع g را به صورت

$$g(i) = r_i(i) + ۱$$

می‌سازیم. مشخص است که این تابع با همه‌ی r_iها متفاوت است و درنتیجه بازگشتی ابتدائی نیست؛ اما به وضوح مشخص است که g محاسبه‌پذیر است و این مثال قضیه را اثبات می‌کند.

تابع آکرمن تابع آکرمن (Ackermann's function)، $A : I \times I \to I$ است که با روابط زیر تعریف می‌شود:

$$\begin{aligned} A(۰, y) &= y + ۱ \\ A(x, ۰) &= A(x - ۱, ۱) \\ A(x, y + ۱) &= A(x - ۱, A(x, y)) \end{aligned}$$

A یک تابع محاسبه‌پذیر و تام است، اما این تابع بازگشتی ابتدائی نیست.
برای این که نشان دهیم این تابع بازگشتی ابتدائی نیست، نمی‌توانیم به طور مستقیم از تعریف A استفاده کنیم، بلکه باید خصوصیات عمومی کلاس توابع بازگشتی ابتدائی را در نظر بگیریم و نشان دهیم که تابع آکرمن یکی از این خصوصیات را ندارد.

یکی از این خصوصیات برای توابع بازگشتی ابتدائی نرخ رشد است. وقتی $i \to \infty$، برای سرعت رشد حدی وجود دارد و تابع آکرمن این حد را نقض می‌کند: **تابع آکرمن بسیار سریع رشد می‌کند.**

قضیه فرض کنید که f یک تابع بازگشتی ابتدائی باشد. در این صورت عدد صحیح n وجود دارد به طوری که برای هر $i = n, n+۱, \ldots$ داریم

$$f(i) < A(n,i)$$

با پذیرش این نتیجه به سادگی مشاهده می‌شود که تابع آکرمن بازگشتی ابتدائی نیست.

قضیه تابع آکرمن بازگشتی ابتدائی نیست.

◄ اثبات.

تابع

$$g(i) = A(i,i)$$

را در نظر می‌گیریم. اگر A بازگشتی ابتدائی باشد، آنگاه g نیز همین‌گونه است. اما، در این صورت طبق قضیه‌ی قبلی باید یک n وجود داشته باشد که برای هر i

$$g(i) < A(n,i)$$

اما اگر قرار دهیم $i = n$، به تناقض

$$g(n) = A(n,n) < A(n,n)$$

برخورد می‌کنیم. پس A نمی‌تواند بازگشتی ابتدائی باشد.

۱۳-۱-۲ توابع μ ـ بازگشتی

برای گسترش ایده‌ی توابع بازگشتی برای پوشش دادن تابع آکرمن و دیگر توابع محاسبه‌پذیر، باید به قواعد ساخت چیزی را بیفزاییم که با آن چنین توابعی بتوانند ساخته شوند. یک راه این کار معرفی عملگر μ یا عملگر می‌نیمم‌سازی (minimalization) است که با رابطه‌ی

$$\mu y(g(x,y)) = \text{smallest } y \text{ such that } g(x,y) = ۰$$

تعریف می‌شود. در این تعریف فرض می‌شود که g یک تابع تام باشد.

مثال

فرض کنید که $g(x,y) = x + y - ۳$ یک تابع تام باشد. در این صورت اگر $x \leq ۳$ باشد، نتیجه‌ی می‌نیمال‌سازی $y = ۳ - x$ خواهد بود.
اما اگر $x > ۳$ باشد، در این صورت هیچ $y \in I$ وجود ندارد که $x + y - ۳ = ۰$ باشد. بنابراین

$$\mu y(g(x,y)) = \begin{cases} ۳ - x & , x \leq ۳ \\ \text{undefined} & , x > ۳ \end{cases}$$

مشاهده می‌شود که اگرچه $g(x,y)$ یک تابع تام است، اما $\mu y(g(x,y))$ ممکن است یک تابع جزئی باشد.
عملگر می‌نیمال‌سازی امکان تعریف توابع جزئی را به صورت بازگشتی فراهم می‌کند، اما قدرت تعریف توابع تام را به گونه‌ای گسترش می‌دهد که همه‌ی توابع محاسبه‌پذیر را شامل گردد.

تعریف یک تابع μ ـ بازگشتی *(μ-recursive)* نامیده می‌شود اگر بتواند به کمک توابع پایه و به وسیله‌ی دنباله‌ای از بکارگیری عملگر μ و اعمال ترکیب و بازگشت ابتدائی ساخته شود.

قضیه یک تابع μ ـ بازگشتی است اگر و فقط اگر محاسبه‌پذیر باشد.

بنابراین توابع μ ـ بازگشتی یک مدل دیگر برای محاسبات الگوریتمی به ما می‌دهد.

۲-۱۳ سیستم‌های پست

یک سیستم پست (Post system) بسیار شبیه یک گرامر نامقید است که متشکل است از یک الفبا و تعدادی قاعده‌ی تولید که به کمک آنها رشته‌های پی در پی می‌توانند مشتق شوند؛ اما تفاوت‌های عمده‌ای در روش بکارگیری قواعد تولید آنها وجود دارد.

تعریف یک سیستم پست Π به وسیله‌ی $\Pi = (C, V, A, P)$ تعریف می‌شود که در آن
C مجموعه‌ای متناهی از ثوابت متشکل از دو مجموعه‌ی مجزای C_N (ثوابت ناپایانه) و C_T (ثوابت پایانه)،
V مجموعه‌ای متناهی از متغیرها،
A مجموعه‌ای متناهی از C^* که اصول موضوع *(axioms)* نام دارد و
P مجموعه‌ای متناهی از قواعد تولید است.

قواعد تولید در یک سیستم پست باید محدودیت‌های خاصی را ارضا کند. همه‌ی آنها باید به شکل

$$x_۱V_۱x_۲\dots V_nx_{n+۱} \to y_۱W_۱y_۲\dots W_my_{m+۱} \qquad (۱-۱۳)$$

باشند که در آن $x_i, y_i \in C^*$ و $V_i, W_i \in V$ در برابر این محدودیت قرار دارد که هر متغیر می‌تواند حداکثر یک مرتبه در سمت چپ ظاهر شود. بنابراین

$$V_i \neq V_j \qquad \text{for } i \neq j$$

و اینکه هر متغیر در سمت راست باید در سمت چپ ظاهر شود، یعنی

$$\bigcup_{i=۱}^{m}\{W_i\} \subseteq \bigcup_{i=۱}^{n}\{V_i\}$$

فرض کنید که رشته‌ای از ناپایانه‌ها به شکل $x_۱w_۱x_۲w_۲\dots w_nx_{n+۱}$ موجود باشد که در آن زیررشته‌های $x_۱$، $x_۲$، ... متناظر با رشته‌های ۱-۱۳ و $w_i \in C^*$ باشد. در این صورت می‌توانیم با تعیین $w_۱ = V_۱$، $w_۲ = V_۲$ و ...، این مقادیر را برای Wها در سمت راست ۱-۱۳ جایگزین کنیم. چون هر W یک V_i است که در سمت چپ ظاهر شده است، یک مقدار یکتا بدان نسبت داده می‌شود و رشته‌ی جدید تولید می‌شود:

$$x_۱w_۱x_۲w_۲\dots x_{n+۱} \Rightarrow y_۱w_iy_۲w_j\dots y_{m+۱}$$

همانند یک گرامر، می‌توانیم در مورد زبان مشتق شده توسط یک سیستم پست صحبت کنیم.

تعریف زبان تولید شده توسط سیستم پست $\Pi = (C, V, A, P)$ عبارت است از

$$L(\Pi) = \{w \in C_T^* : w_\circ \Rightarrow^* w \text{ for some } w_\circ \in A\}$$

مثال

سیستم پست زیر را در نظر بگیرید:

$$C_T = \{a, b\} \qquad C_N = \varnothing \qquad V = \{V_۱\} \qquad A = \{\lambda\}$$

با قاعده‌ی تولید

$$V_۱ \to aV_۱b$$

که اشتقاق زیر را مجاز می‌شمرد:

$$\lambda \Rightarrow ab \Rightarrow aabb$$

در گام اول، ۱-۱۳ را با تعیین $x_۱ = \lambda$، $V_۱ = \lambda$، $x_۲ = \lambda$، $y_۱ = a$، $W_۱ = V_۱$ و $y_۲ = b$ به کارمی‌گیریم. در گام دوم، $V_۱ = ab$ و سایر مقادیر به همان صورت قبلی باقی می‌مانند. اگر این کار را ادامه دهیم متوجه می‌شویم که زبان تولید شده با این سیستم پست خاص $\{a^nb^n : n \geq ۰\}$ است.

مثال

سیستم پست زیر را در نظر بگیرید:

$$C_T = \{۱, +, =\} \quad C_N = \{\varnothing\} \quad V = \{V_۱, V_۲, V_۳\} \quad A = \{۱ + ۱ = ۱۱\}$$

با قواعد تولید

$$V_۱ + V_۲ = V_۳ \quad \rightarrow \quad V_۱۱ + V_۲ = V_۳۱$$
$$V_۱ + V_۲ = V_۳ \quad \rightarrow \quad V_۱ + V_۲۱ = V_۳۱$$

این سیستم اشتقاق زیر را ممکن می‌سازد:

$$۱ + ۱ = ۱۱ \Rightarrow ۱۱ + ۱ = ۱۱۱ \Rightarrow ۱۱ + ۱۱ = ۱۱۱۱$$

تعبیر رشته‌های متشکل از ۱ها به عنوان بازنمایی یگانی اعداد صحیح است. بنابراین، این اشتقاق می‌تواند به صورت زیر نیز نوشته شود:

$$۱ + ۱ = ۲ \Rightarrow ۲ + ۱ = ۳ \Rightarrow ۲ + ۲ = ۴$$

زبان تولید شده با این سیستم پست مجموعه‌ی همه‌ی تساوی‌های جمع اعداد صحیح مانند $۲ + ۲ = ۴$ است که از اصل موضوع $۱ + ۱ = ۲$ مشتق شده است.

قضیه یک زبان شمارش‌پذیر بازگشتی است اگر و فقط اگر یک سیستم پست وجود داشته باشد که آن را تولید کند.

◄ اثبات.

نخست، از آنجا که اشتقاق با یک سیستم پست کاملاً مکانیکی است، می‌تواند با یک ماشین تورینگ انجام شود. بنابراین هر زبان تولید شده با یک سیستم پست، یک زبان شمارش‌پذیر بازگشتی است.

برای عکس قضیه به خاطر داریم که هر زبان شمارش‌پذیر بازگشتی با یک گرامر نامقید G تولید می‌شود که همه‌ی قواعد آن به شکل $x \rightarrow y$ می‌باشد که در آن $x, y \in (V \cup T)^*$ است. با داشتن هر گرامر نامقید G یک سیستم پست $\Pi = (V_\Pi, C, A, P_\Pi)$ می‌سازیم که در آن

$$A = \{S\} \quad C_T = T \quad C_N = V \quad V_\Pi = \{V_۱, V_۲\}$$

و برای هر قاعده‌ی تولید $x \rightarrow y$ در گرامر، قاعده‌ی تولید $V_۱xV_۲ \rightarrow V_۱yV_۲$ در سیستم پست وجود دارد.

در این صورت بسیار ساده خواهد بود که نشان دهیم یک رشته‌ی w می‌تواند توسط سیستم پست Π تولید شود اگر و فقط اگر در زبان تولید شده توسط G قرار داشته باشد.

۱۳-۳ سیستم‌های بازنویسی

سیستم‌های پست با انواع گرامرها اشتراک‌های زیادی دارند: همه‌ی آنها بر اساس یک الفبا هستند و به کمک آنها یک رشته می‌تواند از رشته‌ی دیگر به دست آید. حتی یک ماشین تورینگ را نیز می‌توان به این صورت ملاحظه کرد، چرا که توصیف بلافصل آن یک رشته است که پیکربندی آن را کاملاً مشخص می‌کند. برنامه‌ی ماشین تورینگ مجموعه‌ای از قواعد برای تولید رشته‌ی جدید از روی رشته‌ی قبلی است. این مشاهدات را می‌توانیم به طور رسمی در مفهوم **سیستم‌های بازنویسی** (rewriting systems) بیان کنیم.

به طور کلی یک سیستم بازنویسی از یک الفبای Σ و مجموعه‌ای از قواعد تولید تشکیل می‌شود که به کمک آنها یک رشته در Σ^+ می‌تواند رشته‌ی دیگری را تولید کند.

آنچه سیستم‌های بازنویسی را از هم متمایز می‌کند، طبیعت Σ و محدودیت‌های موجود برای به کارگیری قواعد تولید آنهاست.

در ادامه چند مدل از سیستم‌های بازنویسی ارائه می‌شود.

۱۳-۳-۱ گرامرهای ماتریسی

گرامرهای ماتریسی (matrix grammar) با گرامرهایی که تاکنون مطالعه کرده‌ایم (که اغلب گرامرهای phrase-structure نامیده می‌شوند) در چگونگی بکارگیری قواعد متفاوت هستند.

در گرامرهای ماتریسی، مجموعه‌ی قواعد متشکل است از زیرمجموعه‌های P_1، P_2، ...، P_n که هر یک از آنها یک دنباله‌ی متناهی مرتب از قواعد تولید به صورت

$$x_1 \to y_1, x_2 \to y_2, \ldots, x_m \to y_m,$$

است. هرگاه اولین قاعده‌ی یک زیرمجموعه‌ی P_i استفاده شد، سپس باید دومین قاعده نیز بر روی رشته اعمال شود، آنگاه سومی و به همین ترتیب. در واقع نمی‌توانیم نخستین قاعده‌ی تولید P_i را اعمال کنیم مگر اینکه همه‌ی قواعد دیگر در این مجموعه نیز بتواند اعمال شود.

مثال

گرامر ماتریسی زیر را در نظر بگیرید:

$$\begin{aligned} P_1 &: \quad S \to S_1 S_2 \\ P_2 &: \quad S_1 \to aS_1,\ S_2 \to bS_2c \\ P_3 &: \quad S_1 \to \lambda,\ S_2 \to \lambda \end{aligned}$$

یک اشتقاق با این گرامر عبارت است از:

$$S \Rightarrow S_1 S_2 \Rightarrow aS_1 bS_2 c \Rightarrow aaS_1 bbS_2 cc \Rightarrow aabbcc$$

توجه کنید که هرگاه قاعده‌ی اول P_2 برای ایجاد یک a استفاده شود، قاعده‌ی دوم نیز باید استفاده شود که یک b و c متناظر را تولید می‌کند. استفاده از این مطلب، مشاهده‌ی این که مجموعه‌ی رشته‌های پایانی

تولید شده توسط این گرامر ماتریسی چیست را بسیار ساده می‌کند:

$$L = \{a^n b^n c^n : n \geq 0\}.$$

گرامرهای ماتریسی گرامرهای phrase-structure را به عنوان یک حالت خاص دربر دارند که در آن هر P_i دقیقاً شامل یک قاعده‌ی تولید است.
همچنین، از آنجا که گرامرهای ماتریسی فرآیندهای الگوریتمی را بازنمایی می‌کنند، می‌توان با استفاده از تز چرچ نتیجه گرفت که گرامرهای ماتریسی و گرامرهای phrase-structure دارای قدرت یکسانی به عنوان یک مدل محاسبه هستند. اما مطابق مثال فوق گاهی اوقات استفاده از گرامرهای ماتریسی راه حل بسیار ساده‌تری را نسبت به گرامرهای نامقید به دست می‌دهد.

۱۳-۳-۲ الگوریتم‌های مارکوف

یک الگوریتم مارکوف (Markov algorithm) یک سیستم بازنویسی است که قواعد تولید آن به شکل

$$x \to y$$

دارای ترتیب در نظر گرفته می‌شوند. در یک اشتقاق، اولین قاعده‌ی تولید قابل بکارگیری باید استفاده شود. به علاوه سمت چپ‌ترین وقوع زیررشته‌ی x باید با y جایگزین شود. برخی از قواعد تولید می‌توانند به طور تکی به عنوان قواعد تولید پایانه‌ای خارج شوند که به صورت

$$x \to .y$$

نشان داده می‌شوند. یک اشتقاق با یک رشته‌ی $w \in \Sigma^+$ آغاز می‌شود و آن قدر ادامه می‌یابد تا اینکه یک قاعده‌ی پایانه‌ای مورد استفاده قرار گیرد یا تا وقتی که هیچ قاعده‌ی تولید قابل اعمالی وجود نداشته باشد.
برای پذیرش زبان، یک مجموعه‌ی $T \subseteq \Sigma$ از پایانه‌ها مشخص می‌شود. با شروع از یک رشته‌ی پایانی، قواعد تولید به کار می‌روند تا اینکه رشته‌ی تهی تولید شود.

تعریف

فرض کنید که M یک الگوریتم مارکوف با الفبای Σ و پایانه‌های T باشد. در این صورت زبان پذیرفته شده توسط M، مجموعه‌ی زیر خواهد بود:

$$L(M) = \{w \in T^* : w \Rightarrow^* \lambda\}$$

مثال

یک الگوریتم مارکوف M با $\Sigma = T = \{a, b\}$ و قواعد تولید

$$ab \to \lambda \qquad ba \to \lambda$$

را در نظر بگیرید. هر مرحله در اشتقاق یک زیررشته‌ی ab یا ba را از بین می‌برد. بنابراین

$$L(M) = \{w \in \{a,b\}^* : n_a(w) = n_b(w)\}$$

خواهد بود.

مثال

یک الگوریتم مارکوف برای زبان $L = \{a^n b^n : n \geq 0\}$ ارایه دهید.
پاسخ با مجموعه‌ی قواعد

$$ab \to S \qquad aSb \to S \qquad S \to .\lambda$$

داده می‌شود. اگر در این مثال دو قاعده‌ی اول را برداریم و سمت چپ و راست آن را عوض کنیم، یک گرامر مستقل از متن خواهیم داشت که زبان L را تولید می‌کند.

از یک نظر، الگوریتم‌های مارکوف، گرامرهای phrase-structure ی هستند که به صورت پسرو عمل می‌کنند. این حرف خیلی دقیق نیست، زیرا واضح نیست که با قاعده‌ی تولید آخر بایستی چه کرد. اما این مشاهده یک نقطه‌ی شروع برای اثبات قضیه‌ی زیر فراهم می‌کند که قدرت الگوریتم‌های مارکوف را مشخص می‌نماید.

قضیه یک زبان شمارش‌پذیر بازگشتی است اگر و فقط اگر یک الگوریتم مارکوف برای آن موجود باشد.

۳-۳-۱۳ سیستم‌های L

منشأ سیستم‌های L (L-system) با موارد قبلی تا حدی متفاوت است.
توسعه دهنده‌ی سیستم‌های L، لیندن‌مایر (A. Lindenmayer) از آنها برای مدل کردن الگوی رشد موجودات خاص استفاده می‌کرد.
سیستم‌های L اساساً سیستم‌های بازنویسی موازی هستند. در واقع منظور این است که در هر گام از یک اشتقاق هر نماد باید بازنویسی شود.
قواعد تولید یک سیستم L باید به شکل

$$a \to u$$

باشد که در آن $a \in \Sigma$ و $u \in \Sigma^*$ است.
وقتی یک رشته بازنویسی می‌شود، چنین قاعده‌ی تولیدی باید پیش از تولید رشته‌ی جدید بر روی هر نماد رشته‌ی فعلی اعمال شود.

مثال

فرض کنید $\Sigma = \{a\}$ و $a \to aa$ یک سیستم L را تعریف کند. با شروع از رشته‌ی a می‌توانیم اشتقاق زیر را داشته باشیم:

$$a \Rightarrow aa \Rightarrow aaaa \Rightarrow aaaaaaaa$$

مجموعه‌ی رشته‌هایی که به این صورت مشتق می‌شوند به طور واضح $L = \{a^{۲^n} : n \geq ۰\}$ است.

مجدداً توجه کنید که این سیستم بازنویسی قادر به برخورد ساده با مسایلی است که برای گرامرهای phrase-structure کاملاً دشوار است.

مشخص شده است که سیستم‌های L با قواعد تولید به شکل $a \to u$ به اندازه‌ی کافی برای توصیف همه‌ی محاسبات الگوریتمی کلی نیستند. یک توسعه از این ایده، تعمیم لازم را فراهم می‌کند.

در یک سیستم L توسعه‌یافته قواعد تولید به شکل

$$(x, a, y) \to u$$

هستند که در آن $a \in \Sigma$ و $x, y, u \in \Sigma^*$ است با این تعبیر که a می‌تواند با u جایگزین شود اگر به صورت بخشی از رشته‌ی xay ظاهر شود. مشخص شده است که چنین سیستم L توسعه یافته‌ای یک مدل کلی برای محاسبات است.

مراجع

[1] P. Linz, **An Introduction to Formal Languages and Automata**, 5th Ed., Jones and Bartletts, 2012.

[2] M. Sipser, **Introduction to the Theory of Computation**, 3rd Ed., Cengage Learning, 2013.