به نام خدا

نظریهٔ زبان‌ها و ماشین‌ها

آرش شفیعی

کتاب‌های مرجع

- مقدمه‌ای بر نظریهٔ زبان‌ها (زبان‌های صوری) و ماشین‌ها از پیتر لینز [1]
- مقدمه‌ای بر نظریهٔ محاسبات از مایکل سیپسر [2]
- مقدمه‌ای بر نظریهٔ ماشین‌ها، زبان‌ها، و محاسبات از جان هاپکرافت و همکاران [3]

---

[1] Introduction to Formal Languages and Automata، by Peter Linz
[2] Introduction to the Theory of Computation، by Michael Sipser
[3] Introduction to Automata Theory، Languages، and Computation، by John Hopcroft et al.

مقدمه

مباحث اصلی

- **زبان‌های صوری** [1] (فرمال یا قراردادی) و **گرامر** [2] (دستور زبان) آنها
- **ماشین‌ها** [3] (اتوماتا) برای شناسایی (پذیرفتن) زبان‌ها
- **محاسبه‌پذیری** [4]
- **پیچیدگی** [5]

[1] formal language
[2] grammar
[3] automata
[4] computability
[5] complexity

مباحث اصلی

- **زبان‌های صوری و گرامرها و ماشین‌ها** : چگونه برای شناسایی یک زبان، یک ماشین طراحی می‌کنیم؟
  به عبارت دیگر محاسبات را چگونه توسط کامپیوترها انجام می‌دهیم؟

- **محاسبه‌پذیری** : محدویت ماشین‌ها در شناسایی زبان‌ها چیست؟
  به عبارت دیگر چه نوع محاسباتی را با کامپیوترها می‌توانیم انجام دهیم و چه نوع محاسباتی را نمی‌توانیم؟

- **پیچیدگی** : چگونه زبان‌ها را از لحاظ پیچیدگی (درجهٔ سختی) آنها برای شناسایی توسط ماشین‌ها به دسته‌های متفاوت تقسیم‌بندی می‌کنیم؟
  به عبارت دیگر با چه مقدار و چه نوع حافظه‌ای و با چه سرعتی می‌توانیم محاسبات را انجام دهیم؟

مباحث اصلی

- ساختن یک نظریهٔ انتزاعی برای فهم اصول محاسبات و کامپیوترها
- پیدا کردن بینشی کلی در مورد چیستی و توانایی کامپیوترها
- کمک کردن به استفاده از کامپیوترها و زبان‌های برنامه‌نویسی برای حل مسائل
- به طور خلاصه: مدلسازی محاسبات برای استفاده در حل مسئله‌های محاسباتی

مباحث اصلی

- برای مدلسازی محاسبات از مفهوم **ماشین (اتوماتون)** استفاده می‌کنیم.
- یک ماشین یک ورودی را دریافت می‌کند، و تصمیم می‌گیرد که چگونه ورودی را به خروجی تبدیل کند.
- چگونگی حافظه و کنترل کنندهٔ (تصمیم‌گیرنده) هر ماشین را بیان می‌کنیم.
- پس یک ماشین انتزاعی مفهومی است برای تعریف یک محاسبه‌گر (کامپیوتر).

مباحث اصلی

- **زبان صوری** تشکیل شده است از تعدادی جملهٔ قابل قبول در آن زبان.
- برای ساختن یک جمله از تعدادی نماد (سمبول) و قوانینی برای ترکیب نمادها استفاده می‌کنیم.
- پس یک زبان صوری مفهومی انتزاعی از زبان‌های برنامه‌نویسی است.

پیش‌نیاز

- آشنایی با نظریهٔ مجموعه‌ها
- آشنایی با توابع و روابط
- آشنایی با گراف‌ها و درخت‌ها
- آشنایی با اثبات ریاضی

پیش‌نیاز

آشنایی با نظریهٔ مجموعه‌ها:

- اشتراک، اجتماع، تفاضل، متمم
- مجموعهٔ مرجع، مجموعهٔ تهی
- قوانین دمورگان
- زیر مجموعه، زیرمجموعهٔ محض، مجموعه‌های مجزا
- مجموعهٔ متناهی، مجموعهٔ نامتناهی
- مجموعهٔ توانی
- ضرب دکارتی
- افراز یک مجموعه

پیش‌نیاز

آشنایی با توابع و روابط:

- تابع و دامنه و برد
- تابع کامل و تابع جزئی
- مرتبۀ بزرگی توابع، حداکثر از مرتبه (کران پایین حدی)، حداقل از مرتبه (کران بالای حدی)، هم‌مرتبه
- رابطه
- رابطۀ هم‌ارزی، بازتابی، متقارن، ترایا

پیش‌نیاز

آشنایی با گراف‌ها و درخت‌ها

- گراف و رأس و یال
- گراف جهت‌دار و گراف معمولی (بدون جهت)
- گشت، مسیر، مسیر ساده، دور، حلقه
- درخت، ریشه، برگ، والد، فرزند
- سطح رأس درخت، ارتفاع درخت

پیش‌نیاز

آشنایی با اثبات ریاضی

- استدلال استقرایی
- برهان خلف

زبان‌ها

- زبان‌های طبیعی (مانند فارسی و انگلیسی و فرانسوی) مجموعه‌ای از نمادها (سمبول‌ها) و قوانین گرامری هستند که برای بیان مفاهیم و حقایق و ارتباط انسان‌ها به کار می‌روند.
- در این مبحث، برای مطالعهٔ علمی زبان‌های صوری (فرمال یا قراردادی) باید تعریف دقیق‌تری از زبان ارائه کنیم.

زبان‌ها

- یک زبان مجموعه ای است از **رشته‌ها** [1].
- یک رشته دنباله‌ای محدود است از **نمادها** (سمبول‌ها) [2].
- به یک مجموعه Σ از سمبول‌ها یک **الفبا** [3] می‌گوییم.

[1] string
[2] symbol
[3] alphabet

- برای مثال اگر الفبای $\Sigma = \{a, b\}$ را داشته باشیم،
- رشته‌های abab و aaabbba رشته‌هایی از الفبای Σ هستند.
- برای نمایش یک رشته با نام w و مقدار abaaa می‌نویسیم : $w = abaaa$.

زبان‌ها

- **الحاق** [1] رشتهٔ w و رشتهٔ v با افزودن نمادهای رشتهٔ v به سمت راست نمادهای رشتهٔ w به دست می‌آید.
$$w = a_۱ \cdots a_n, \quad v = b_۱ \cdots b_m, \quad wv = a_۱ \cdots a_n b_۱ \cdots b_m$$
- **معکوس** [2] یک رشته با نوشتن نمادهای آن با ترتیب معکوس (از آخر به اول) به دست می‌آید.
$$w^R = a_n \cdots a_۱$$
- **طول** [3] رشتهٔ w با $|w|$ نشان داده می‌شود که تعداد نمادهای آن رشته است.
- **رشتهٔ تهی** [4] رشته‌ای است که هیچ نمادی در آن نیست و با نماد لامبدا λ (یا نماد اپسیلون ϵ) نشان داده می‌شود.
$$|\lambda| = ۰, \quad \lambda w = w\lambda = w$$

---
[1] concatenation
[2] reverse
[3] length
[4] empty string

زبان‌ها

- یک **زیررشته** [1] از رشتهٔ w دنباله‌ای از نمادهای متوالی در w است.
- اگر $w = vu$ باشد، v و u به ترتیب **پیشوند** [2] و **پسوند** [3] نامیده می‌شوند.
- اگر $w = abbab$ باشد، $\{\lambda, a, ab, abb, abba, abbab\}$ مجموعهٔ همه پیشوندهای w است و $\{\lambda, b, ab, bab, bbab, abbab\}$ مجموعهٔ همه پسوندها.
- برای طول رشته‌ها این رابطه برقرار است : $|uv| = |u| + |v|$

[1] substring
[2] prefix
[3] suffix

زبان‌ها

- رشتهٔ w^n رشته‌ای است که از تکرار رشتهٔ w به تعداد n بار به دست می‌آید.
- تعریف می‌کنیم : $w^\circ = \lambda$
- مجموعهٔ همهٔ رشته‌هایی را که با الفبای Σ به دست می‌آید با Σ^* نشان می‌دهیم.
- Σ^* همیشه رشتهٔ λ را نیز در بر می‌گیرد.
- تعریف می‌کنیم : $\Sigma^+ = \Sigma^* - \{\lambda\}$
- در حالی که Σ طبق تعریف یک مجموعهٔ محدود است، Σ^* و Σ^+ همیشه نامحدود هستند.
- یک زبان L زیرمجموعه‌ای از Σ^* است.
- هر رشته از زبان L را یک **جمله** از زبان L می‌نامیم.

زبان‌ها

- اگر $\Sigma = \{a, b\}$ باشد، داریم : $\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, \cdots\}$
- مجموعهٔ $\{a, aa, aab\}$ یک زبان بر روی الفبای Σ است. این زبان یک زبان محدود است.
- مجموعهٔ $L = \{a^n b^n : n \geq 0\}$ نیز یک زبان بر روی الفبای Σ است. اما این زبان یک زبان نامحدود است. بعضی جملات آن برابرند با aabb و aaaabbbb . اما جملهٔ abb عضو این زبان نیست.

زبان‌ها

- از آنجایی که یک زبان، یک مجموعه است، همهٔ عملیات اجتماع و اشتراک و تفاضل بر روی آنها تعریف می‌شوند.
- با توجه به این که مجموعه مرجع، مجموعه‌ای شامل همه جملات بر روی یک الفبا است، متمم یک زبان بدین صورت تعریف می‌شود: $\overline{L} = \Sigma^* - L$
- معکوس یک زبان، معکوس همهٔ جملات آن است: $L^R = \{w^R : w \in L\}$
- الحاق دو زبان، الحاق همهٔ جملات آن دو زبان است: $L_۱L_۲ = \{xy : x \in L_۱, y \in L_۲\}$
- الحاق یک زبان به خودش را به تعداد n بار به صورت L^n تعریف می‌کنیم.
- اگر $L = \{a, ab\}$ باشد، داریم $L^۲ = \{aa, aab, aba, abab\}$
- تعریف می‌کنیم : $L^۰ = \{\lambda\}$

زبان‌ها

- **بستار-ستاره** [1] بر روی زبان L را بدین صورت تعریف می‌کنیم: $L^* = L^{۰} \cup L^{۱} \cup L^{۲} \cup \cdots$
- **بستار-مثبت** [2] بر روی زبان L را بدین صورت تعریف می‌کنیم: $L^+ = L^{۱} \cup L^{۲} \cup \cdots$
- اگر $L = \{a^n b^n : n \geq ۰\}$ باشد، داریم $L^{۲} = \{a^n b^n a^m b^m : n \geq ۰, m \geq ۰\}$.
- همچنین داریم : $L^R = \{b^n a^n : n \geq ۰\}$

[1] star-closure
[2] positive closure

گرامرها

- برای تعریف یک زبان صوری با استفاده از زبان ریاضی از نظریهٔ مجموعه‌ها استفاده کردیم.
- ولی مجموعه‌ها پاسخگوی همهٔ نیازهای ما نیستند و محدودیت‌هایی دارند.
- اکنون از گرامرها برای تعریف یک زبان استفاده می‌کنیم.

گرامرها

- گرامر در یک زبان طبیعی به ما می‌گوید که آیا ساختار یک جمله درست است یا غلط.
- به عبارت دیگر گرامر چگونگی ساختار جمله (نحوهٔ کنار هم قرار گرفتن واژه‌ها و تکواژها) را توصیف می‌کند.
- مثلا در زبان فارسی می‌توانیم گرامری به این صورت تعریف کنیم: <نهاد> <گزاره> → <جمله>
- حالا باید تعریف کنیم که نهاد و گزاره چه هستند.
- می‌توانیم تعریف کنیم: <گروه اسمی> → <نهاد> و <مفعول><فعل> → <گزاره> و <متمم><فعل> → <گزاره> و <مسند><فعل> → <گزاره> و ...
- مثلا در جملهٔ «خدا مهربان است»، «خدا» نهاد، «مهربان» مسند، و «است» فعل است. همچنین «مهربان است» یک گزاره است.

گرامرها

- پس در یک گرامر با یک جمله شروع می‌کنیم و اجزای آن را مشخص می‌کنیم.
- یک زبان تشکیل شده است از جملاتی که با آن گرامر ساختار بندی شده اند.
- برای زبان‌های صوری نیز به همین ترتیب عمل می‌کنیم.

گرامرها

- یک گرامر G به صورت یک چهارتایی $G = (V, T, S, P)$ تعریف می‌شود به طوری که:
- V مجموعه‌ای متناهی از **متغیر** [1] هاست.
- T مجموعه‌ای متناهی از **نماد های پایانی (پایانه)** [2] است.
- $S \in V$ نمادی است به نام **متغیر آغازی** [3] .
- P مجموعه‌ای متناهی از **قوانین تولید** [4] است.
- مجموعه‌های V و T غیرتهی و مجموعه‌هایی مجزا هستند.

---
[1] variable
[2] terminal symbols
[3] start variable
[4] production rules

گرامرها

- قوانین تولید تعیین می‌کنند چگونه گرامر یک رشته را به یک رشتهٔ دیگر تبدیل می‌کند.
- قوانین تولید را بدین صورت نمایش می‌دهیم : $x \rightarrow y$
- به طوری که : $x \in (V \cup T)^+$ و $y \in (V \cup T)^*$
- برای مثال رشتهٔ $w = uxv$ با استفاده از قانون تولید $x \rightarrow y$ به رشتهٔ $z = uyv$ تبدیل می‌شود.
- در این صورت می‌نویسیم $w \Rightarrow z$ و می‌خوانیم w نتیجه (به دست) می‌دهد z و یا z از w مشتق می‌شود.
- با اعمال قوانین تولید با ترتیب دلخواه، رشته‌های پی‌درپی مشتق می‌شوند.

گرامرها

- اگر داشته باشیم : $w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$ می‌گوییم w_1 در چند گام نتیجه می‌دهد w_n و می‌نویسیم

$$w_1 \overset{*}{\Rightarrow} w_n$$

- علامت $\overset{*}{\Rightarrow}$ به معنای نتیجه دادن با تعداد نامشخصی از گام‌ها است.

- مجموعهٔ همه رشته‌هایی که از یک گرامر به دست می‌آیند، و فقط از نمادهای پایانی تشکیل شده باشند (رشته‌های پایانی)، زبان مرتبط با آن گرامر را تعریف می‌کنند.

گرامرها

- فرض کنید $G = (V, T, S, P)$ یک گرامر باشد.
- آنگاه مجموعهٔ $L(G) = \{w \in T^* : S \overset{*}{\Rightarrow} w\}$ زبان تولید شده با استفاده از گرامر G است.
- اگر $w \in L(G)$ باشد، آنگاه دنبالهٔ $S \Rightarrow w_۱ \Rightarrow w_۲ \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$ یک **نتیجه‌گیری (اشتقاق)**[1] از جملهٔ w است.
- همهٔ رشته‌های $S, w_۱, \cdots w_n$ که از متغیرها و نمادهای پایانی تشکیل شده‌اند، **صورت‌های جمله‌ای** [2] از این نتیجه‌گیری هستند.

[1] derivation
[2] sentential form

- گرامر $G = (\{S\}, \{a, b\}, S, P)$ با این قوانین تولید را در نظر بگیرید: $S \rightarrow \lambda \quad , \quad S \rightarrow aSb$
- بنابراین می‌توانیم داشته باشیم : $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$
- می‌توانیم بنویسیم : $S \overset{*}{\Rightarrow} aabb$
- رشتهٔ aabb یک جملهٔ تولید شده توسط این گرامر است و رشتهٔ aaSbb یک صورت جمله‌ای است.
- می‌توانیم حدس بزنیم که گرامر G زبان $L(G)$ را بدین صورت تعریف می‌کند : $L(G) = \{a^n b^n : n \geq 0\}$
- می‌توانیم این حدس را با استفاده از اثبات استقرایی اثبات کنیم.

گرامرها

- گرامر $G = (\{S\}, \{a, b\}, S, P)$ با این قوانین تولید را در نظر بگیرید: $S \rightarrow aSb \quad , \quad S \rightarrow \lambda$
- می‌توانیم حدس بزنیم که گرامر G زبان $L(G)$ را بدین صورت تعریف می‌کند : $L(G) = \{a^n b^n : n \geq 0\}$
- اول اینکه باید نشان دهیم جملهٔ $a^n b^n$ از این گرامر به دست می‌آید.
- کافی است قانون $S \rightarrow aSb$ را n بار و قانون $S \rightarrow \lambda$ را یک بار اعمال کنیم.

- دوم اینکه باید نشان دهیم که گرامر G فقط زبان $L(G)$ را تولید می‌کند.
- پس باید نشان دهیم که همهٔ صورت‌های جمله‌ای بدین شکل هستند : $w_i = a^i S b^i \quad (۱)$
- فرض استقرا این است که تساوی (۱) برای $i = ۱$ برقرار است.
- حال اگر تساوی (۱) برای هر $n \geq i \geq ۰$ برقرار باشد، باید نشان دهیم برای $i = n + ۱$ نیز برقرار است.
- برای به دست آوردن $w_{n+۱}$ فقط می‌توانیم یک قانون را اعمال کنیم و آن هم قانون $S \rightarrow aSb$ است. پس الزاما داریم: $w_{n+۱} = a^{n+۱} S b^{n+۱}$
- نتیجه می‌گیریم همهٔ صورت های جمله‌ای این گرامر به شکل تساوی (۱) هستند، پس گرامر G فقط زبان $L(G)$ را تولید می‌کند.

- گرامری را بیابید که زبان L را تولید می‌کند : $L = \{a^n b^{n+1} : n \geq 0\}$

گرامرها

- گرامری را بیابید که زبان L را تولید می‌کند : $L = \{a^n b^{n+1} : n \geq 0\}$
- گرامر $G = (\{S, A\}, \{a, b\}, S, P\}$
- با قوانین تولید : $S \to Ab \quad , \quad A \to aAb \quad , \quad A \to \lambda$

گرامرها

- می‌توانیم در بسیاری مواقع جواب را به طور شهودی حدس بزنیم.
- برای اینکه نشان دهیم زبان L که حدس زده‌ایم توسط گرامر G تولید می‌شود، باید:
- (۱) نشان دهیم که همهٔ $w \in L$ توسط گرامر G با شروع صورت جمله‌ای S به دست می‌آیند،
- (۲) هر جملهٔ به دست آمده‌ای از گرامر G در زبان L است.

- الفبای $\Sigma = \{a, b\}$ و گرامر G با قوانین زیر را در نظر بگیرید:
$$S \to SS \quad , \quad S \to \lambda \quad , \quad S \to aSb \quad , \quad S \to bSa$$
- زبان $L(G)$ را تعیین کنید.

- الفبای $\Sigma = \{a, b\}$ و گرامر G با قوانین زیر را در نظر بگیرید:
$$S \to SS \;\; , \;\; S \to \lambda \;\; , \;\; S \to aSb \;\; , \;\; S \to bSa$$
- فرض کنید $n_a(w)$ و $n_b(w)$ تعداد تکرارهای a و b را در رشتهٔ w را نشان می‌دهند.
- در اینصورت داریم: $L = \{w : n_a(w) = n_b(w)\}$

گرامرها

- $S \rightarrow SS \;,\; S \rightarrow \lambda \;,\; S \rightarrow aSb \;,\; S \rightarrow bSa$
- $L = \{w : n_a(w) = n_b(w)\}$
- اول اینکه باید اثبات کنیم هر جمله‌ای توسط گرامر G تولید می‌شود، در زبان L وجو دارد.
- همهٔ صورت‌های جمله‌ای که در گرامر G تولید می‌شوند، تعداد a و b برابر دارند، چرا که قوانینی که یک a به صورت جمله‌ای اضافه می‌کنند، یک b نیز اضافه می‌کنند.
- بنابراین هر جمله‌ای که در $L(G)$ است، در L نیز وجود دارد.

- $S \to SS \ , \ S \to \lambda \ , \ S \to aSb \ , \ S \to bSa$
- $L = \{w : n_a(w) = n_b(w)\}$
- دوم اینکه باید اثبات کنیم همهٔ جمله‌هایی که در L وجود دارند، توسط گرامر G به دست می‌آیند.
- حالت‌های مختلف را در نظر می‌گیریم:
- اگر $w = aw_1b$ باشد، از آنجایی که تعداد a و b های w برابر است، تعداد a و b های w_1 هم برابر است. پس w_1 هم در L است. بنابراین w از این قانون به وجود آمده است : $S \to aSb$
- اگر $w = bw_1a$ باشد، به طور مشابه با حالت اول استدلال می‌کنیم.

- اگر $w = a \cdots a$ باشد، به صورت زیر استدلال می‌کنیم. فرض کنید به ازای مشاهدهٔ هر a یک واحد به یک شمارنده اضافه کنیم و به ازای مشاهدهٔ هر b یک واحد از شمارنده کم کنیم. در پایان این شمارش باید شمارنده به صفر برسد (چون تعداد a و b برابر است).

- پس در این حالت سوم که w با a شروع می‌شود و پایان می‌یابد، شمارنده باید قبل از اتمام رشته، یک بار به صفر رسیده باشد. در آن نقطه که شمارنده به صفر رسیده است، رشته‌ای به نام w_1 به دست آورده‌ایم که تعداد a و b در آن برابر است. دوباره از نقطه صفر در وسط رشته شروع می‌کنیم و در پایان شمارنده به صفر می‌رسد. پس در قسمت دوم رشته‌ای به نام w_2 به دست آورده‌ایم که تعداد a و b در آن برابر است.

- بنابراین در این حالت سوم $w = w_1 w_2$ جایی که هر دو رشتهٔ w_1 و w_2 در زبان هستند. پس برای این رشته می‌توانیم از قانون $S \to SS$ استفاده کنیم.

- در حالت چهارم $w = b \cdots b$ مشابه حالت سوم استدلال می‌کنیم.

- گرامر $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$ با قوانین تولید P_1 به صورت زیر را در نظر بگیرید.
- $S \to aAb|\lambda \quad , \quad A \to aAb|\lambda$
- علامت خط عمودی | به معنی «فصل منطقی» (یا ی منطقی) است. بدین معنی که سمت چپ یک قانون می‌تواند عبارت اول در سمت راست قانون «یا» عبارت دوم در سمت راست قانون را نتیجه دهد.
- پس در این گرامر چهار قانون تولید داریم.
- زبان تولید شده توسط این گرامر برابر است با: $L(G_1) = \{a^n b^n : n \geq 0\}$

- گرامر $S \to aS|\lambda$ چه زبانی را تولید می‌کند؟
- گرامر $S \to aS|a$ چه زبانی را تولید می‌کند؟
- گرامر $S \to aS$ چه زبانی را تولید می‌کند؟

گرامرها

- گرامر $S \to aS|\lambda$ چه زبانی را تولید می‌کند؟ $L_1 = \{\lambda, a, aa, \cdots\}$
- گرامر $S \to aS|a$ چه زبانی را تولید می‌کند؟ $L_2 = \{a, aa, \cdots\}$
- گرامر $S \to aS$ چه زبانی را تولید می‌کند؟ $L_3 = \{\} = \emptyset$

ماشین‌ها

- یک **ماشین** [1] (اتوماتون) مدلی انتزاعی از یک کامپیوتر (محاسبه‌گر) دیجیتال است.
- یک ماشین سازوکاری (مکانیزم) [2] برای خواندن رشتهٔ ورودی دارد (که این رشته‌ها بر روی الفبایی تعریف شده اند).
- رشتهٔ ورودی بر روی یک **فایل ورودی** [3] نوشته شده است که ماشین آن را می‌خواند.
- یک فایل ورودی از سلول [4] (خانه) هایی تشکیل شده است به طوری که هر نماد از رشتهٔ ورودی در یک سلول نوشته می‌شوند.

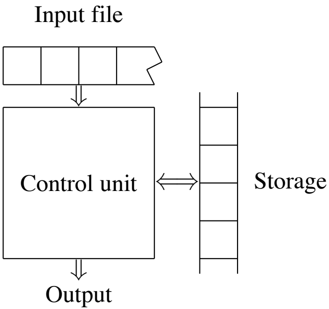
[1] automaton
[2] mechanism
[3] input file
[4] cell

ماشین‌ها

- سازوکار خواندن رشتهٔ ورودی قادر به تشخیص انتهای رشته است.
- یک ماشین می‌تواند یک خروجی نیز تولید کند.
- یک ماشین می‌تواند یک **حافظه** [1] موقت نیز داشته باشد، که تشکیل شده است از تعداد نامحدودی سلول برای نگهداری نمادهایی از یک الفبا (لزوما این الفبا با الفبای رشتهٔ ورودی یکسان نیست).
- ماشین می‌تواند محتوای حافظه را بخواند و تغییر دهد.
- ماشین همچنین یک **واحد کنترل** [2] دارد که می‌تواند در یکی از **حالت‌های داخلی** [3] باشد، به طوری که تعداد حالت‌ها محدود است.
- ماشین می‌تواند با یک روش تعیین‌شده از یک حالت به یک حالت دیگر برود.

[1] storage
[2] control unit
[3] internal state

- شکل زیر، یک ماشین را در حالت کلی نشان می‌دهد.



ماشین‌ها

- یک ماشین در بازه‌های زمانی گسسته عمل می‌کند.
- در هر نقطهٔ زمانی، واحد کنترل در یک حالت داخلی است و سازوکار ورودی، یک نماد را از فایل ورودی می‌خواند.
- حالت داخلی واحد کنترل در نقطهٔ زمانی بعدی، با یک **تابع گذار** [1] تعیین می‌شود.
- تابع گذار (انتقال)، حالت بعدی را بر اساس حالت فعلی، نماد فعلی بر روی ورودی، و اطلاعات فعلی روی حافظه تعیین می‌کند.
- در هر گذار از یک حالت به حالت دیگر، می‌تواند خروجی نیز تولید شود و یا اطلاعات روی حافظه تغییر کند.
- یک حالت از واحد کنترل، فایل ورودی، و حافظه را یک **پیکربندی** [2] می‌گوییم.
- گذار از یک پیکربندی به پیکربندی دیگر را یک **حرکت** [3] نامیم.

[1] transition function
[2] configuration
[3] move

ماشین‌ها

- همهٔ ماشین‌هایی که بررسی می‌کنیم، از این مدل کلی پیروی می‌کنند.
- همهٔ ماشین‌ها یک واحد کنترل دارند، اما تفاوت آنها در تولید خروجی و نوع حافظهٔ آنها است.
- خواهیم دید که نوع حافظه، توانایی ماشین‌ها را برای شناسایی زبان‌ها تعیین می‌کند.

ماشین‌ها

- ماشین‌ها می‌توانند **قطعی** [1] و یا **غیر قطعی** [2] باشند.
- یک ماشین قطعی، ماشینی است که در آن هر حرکت به طور منحصر به فرد به ازای پیکربندی فعلی تعیین شده است.
- بدین معنی که در یک ماشین قطعی، اگر حالت داخلی، ورودی، و محتوای حافظه را بدانیم، می‌توانیم دقیقا رفتار ماشین را در آینده پیش‌بینی کنیم.
- در یک ماشین غیرقطعی، در هر نقطهٔ زمانی، با توجه به پیکربندی فعلی، چندین امکان برای حرکت وجود دارد، پس تنها می‌توانیم مجموعه‌ای از حرکت‌های ممکن را پیش‌بینی کنیم.

[1] deterministic
[2] nondeterministic

ماشین‌ها

- یک ماشین که خروجی آن فقط «بله» و «خیر» است را یک **پذیرنده** [1] می‌نامیم.
- به ازای یک رشتهٔ داده شده، یک پذیرنده رشته را یا قبول می‌کند و یا رد می‌کند.
- ماشینی که توانایی تولید رشته‌ها در خروجی را داشته باشد، **مبدل** [2] می‌نامیم.

[1] acceptor
[2] transducer

کاربردها

- علاوه بر اینکه یادگیری مبحث زبان‌ها و ماشین‌ها به ما روش فکر کردن در مسائل محاسباتی را می‌آموزد، زبان‌های صوری و گرامرها به طور گسترده‌ای در طراحی زبان‌های برنامه‌نویسی و کامپایلرها کاربرد دارند.
- برای طراحی یک زبان برنامه‌نویسی و تولید یک کامپایلر ابتدا نیاز به تعریف گرامر آن زبان داریم.

کاربردها

- برای مثال، قوانین تعریف یک متغیر در زبان برنامه‌نویسی سی چنین است:
- نام متغیر (۱) دنباله‌ای از حروف انگلیسی (کوچک و بزرگ)، ارقام، و زیرخط [1] است و (۲) تنها می‌تواند با حروف و زیرخط آغاز شود.
- پس گرامر آن را می‌توانیم بدین صورت تعریف کنیم:

$$
\begin{aligned}
< \text{name} > &\rightarrow < \text{letter} >< \text{rest} > | < \text{uscore} >< \text{rest} > \\
< \text{rest} > &\rightarrow < \text{letter} >< \text{rest} > | < \text{digit} >< \text{rest} > | < \text{uscore} >< \text{rest} > |\lambda \\
< \text{letter} > &\rightarrow \text{a}|\text{b}|\cdots|\text{z}|\text{A}|\text{B}|\cdots|\text{Z} \\
< \text{digit} > &\rightarrow \text{۰}|\text{۱}|\cdots|\text{۹} \\
< \text{uscore} > &\rightarrow \_
\end{aligned}
$$

[1] underscore

کاربردها

$$
\begin{aligned}
< \text{name} > &\rightarrow < \text{letter} >< \text{rest} > \mid < \text{uscore} >< \text{rest} > \\
< \text{rest} > &\rightarrow < \text{letter} >< \text{rest} > \mid < \text{digit} >< \text{rest} > \mid < \text{uscore} >< \text{rest} > \mid \lambda \\
< \text{letter} > &\rightarrow \text{a}|\text{b}|\cdots|\text{z}|\text{A}|\text{B}|\cdots|\text{Z} \\
< \text{digit} > &\rightarrow ۰|۱|\cdots|۹ \\
< \text{uscore} > &\rightarrow \_
\end{aligned}
$$

- در مثال فوق مجموعهٔ متغیرها شامل $< \text{name} >, < \text{rest} >, < \text{letter} >, < \text{digit} >, < \text{uscore} >$ و مجموعهٔ پایانه‌ها شامل حروف و ارقام و زیرخط می‌شوند.

- نام متغیر a۰ را از این گرامر اینگونه به دست می‌آوریم:

$$< \text{name} > \Rightarrow < \text{letter} >< \text{rest} > \Rightarrow \text{a} < \text{rest} > \Rightarrow \text{a} < \text{digit} >< \text{rest} > \Rightarrow \text{a}۰ < \text{rest} > \Rightarrow \text{a}۰$$

کاربردها

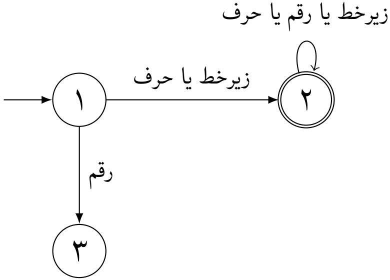
- همچنین می‌توانیم ماشینی طراحی کنیم که به ازای یک رشته داده شده پاسخ دهد که آیا رشتهٔ داده شده به عنوان یک نام متغیر مورد قبول است یا خیر.
- چنین ماشینی چنانکه اشاره شد یک پذیرنده است.

کاربردها

- واحد کنترل یک ماشین (پذیرنده و یا مبدل) معمولا به صورت یک گراف نمایش داده می‌شود.
- عملیات محاسبه (پذیرش یا تبدیل) از یکی از رأس‌ها (که با یک یال بدون مبدأ مشخص شده است) آغاز می‌شود.
- در هر واحد زمان، ماشین یک نماد را از ورودی می‌خواند.
- فرض کنید یالی با برچسب a از رأس x به رأس y وجود دارد. وجود این یال بدین معنی است که ماشین با خواندن نماد a از ورودی از حالت x به حالت y می‌رود.
- اگر خواندن یک رشته در حالتی به پایان برسد که با دو دایرهٔ تودرتو نشان داده شده است، رشته پذیرفته می‌شود.

کاربردها

- ماشین زیر نام یک متغیر را در زبان برنامه‌نویسی سی می‌پذیرد یا رد می‌کند.
- ماشین در حالت ۱ آغاز می‌شود. در هر گام، یک نماد از رشتهٔ ورودی خوانده می‌شود.
- اگر رشته‌ای در حالت ۲ پایان یابد، رشتهٔ مورد نظر پذیرفته می‌شود، در غیر اینصورت رشته رد می‌شود.



کاربردها

- در گراف واحد کنترل مربوط به یک مبدل، فرض کنید یالی با برچسب a/b از رأس x به y وجود دارد. وجود این یال بدین معنی است که با خواندن نماد a ماشین از حالت x به حالت y می‌رود و نماد b را در خروجی تولید می‌کند.
- برچسب a/λ بدین معنی است که با خواندن نماد a از ورودی، ماشین نماد λ را در خروجی تولید می‌کند و یا به عبارتی نمادی در خروجی نمی‌نویسد.

کاربردها

- فرض کنید یک عدد دودویی به صورت رشتهٔ $x = a_0 a_1 \cdots a_n$ از ورودی دریافت می‌شود به طوری که معادل عددی رشته در مبنای ده برابر است با: $\sum_{i=0}^{n} a_i 2^i$

- یک جمع‌کننده سریال دو رشتهٔ دودویی $x = a_0 \cdots a_n$ و $y = b_0 \cdots b_n$ را دریافت می‌کند و آنها را رقم به رقم با یکدیگر جمع کرده و حاصل را در خروجی می‌نویسد.

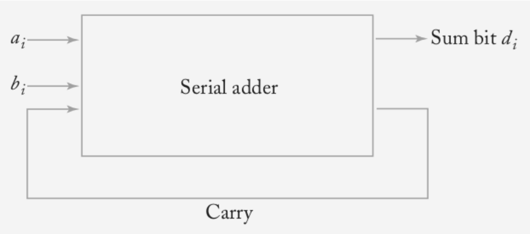
- عمل جمع با در نظر گرفتن رقم نقلی[1] طبق جدول زیر انجام می‌شود.

a_i \ b_i	0	1
0	0 No carry	1 No carry
1	1 No carry	0 Carry

[1] carry

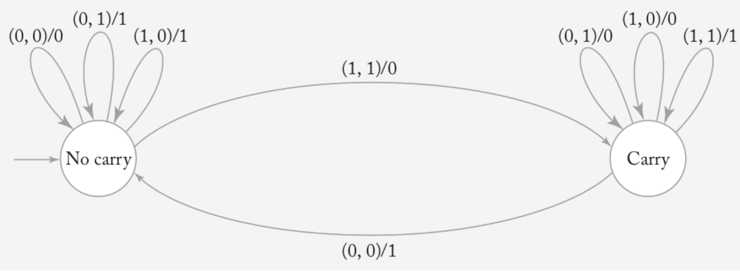
کاربردها

- شمای کلی یک جمع‌کننده را می‌توان به صورت زیر نشان داد.



کاربردها

- برای چنین جمع‌کننده‌ای می‌توان مبدلی به صورت زیر طراحی کرد.

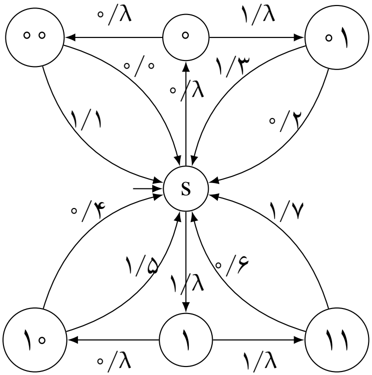


کاربردها

- یک مبدل طراحی کنید که یک عدد دودویی را به صورت یک رشته دریافت کند و معادل آن را در مبنای ۸ در خروجی بنویسد. برای مثال با دریافت رشتهٔ ۰۰۱۱۰۱۱۱۰ از ورودی، مبدل رشتهٔ ۱۵۶ را تولید می‌کند. فرض کنید طول رشتهٔ ورودی همیشه مضربی از ۳ است.

کاربردها

- یک مبدل طراحی کنید که یک عدد دودویی را به صورت یک رشته دریافت کند و معادل آن را در مبنای ۸ در خروجی بنویسد. برای مثال با دریافت رشتهٔ ۰۰۱۱۰۱۱۱۰ از ورودی، مبدل رشتهٔ ۱۵۶ را تولید می‌کند. فرض کنید طول رشتهٔ ورودی همیشه مضربی از ۳ است.



ماشین‌های حالات متناهی

مقدمه

- در این قسمت با یک ماشین ابتدایی به نام پذیرنده حالات متناهی [1] آشنا می‌شویم.
- این ماشین مجموعه‌ای متناهی از حالات داخلی دارد و حافظه جانبی ندارد (پس فقط حالتی که در آن قرار دارد را به یاد می‌آورد و حافظهٔ آن محدود است).
- این ماشین یک پذیرنده است، زیرا رشتهٔ ورودی را یا می‌پذیرد و یا رد می‌کند.
- ماشین‌های متناهی به دو دستهٔ قطعی و غیرقطعی تقسیم می‌شوند.

[1] finite state acceptor

مقدمه

- ابتدا با ماشین/پذیرندهٔ متناهی قطعی (دی‌اف‌ای) [1] آشنا می‌شویم.
- یک دی‌اف‌ای در هر حالت، با خواندن یک نماد فقط یک انتخاب برای تغییر حالت خود دارد.
- سپس با ماشین متناهی غیرقطعی (ان‌اف‌ای) [2] آشنا می‌شویم.
- با خواندن یک نماد در هر حالت، یک ان‌اف‌ای برای انتخاب حالت بعدی چندین گزینه دارد. این ماشین می‌تواند همهٔ حالت‌های بعدی ممکن را بررسی کند و یکی از حالات را انتخاب کند.
- دلیل اصلی استفاده از ان‌اف‌ای، سادگی آن در طراحی یک ماشین متناهی برای یک زبان است.
- از ماشین‌های حالات متناهی برای شناسایی و تعریف دسته‌ای از زبان‌ها به نام زبان‌های منظم استفاده خواهیم کرد.

[1] deterministic finite automaton/acceptor (dfa)
[2] nondeterministic finite automaton (nfa)

مقدمه

- اگر دو پذیرنده یک زبان واحد را شناسایی کنند می‌گوییم این دو پذیرنده معادل یکدیگرند.
- دستهٔ پذیرنده‌های قطعی و پذیرنده‌های غیرقطعی با یکدیگر معادل هستند، زیرا برای هر ان‌اف‌ای می‌توانیم یک دی‌اف‌ای معادل آن بیابیم.
- برای یک زبان منظم تعداد زیادی دی‌اف‌ای وجود دارد که معادل یکدیگرند. در اینصورت می‌توانیم دی‌اف‌ای کمینه (مینیمال) [1] را برای آن زبان بیابیم.

[1] minimal

ماشین متناهی قطعی

یک ماشین متناهی قطعی (دی‌اف‌ای):

- تعداد محدودی حالات داخلی دارد.
- یک رشتهٔ ورودی را با گرفتن یک نماد در واحد زمان پردازش می‌کند.
- با توجه به حالت داخلی فعلی و نماد ورودی به یک حالت دیگر گذار می‌کند.

ماشین متناهی قطعی

- یک پذیرندهٔ متناهی قطعی یا دی‌اف‌ای به صورت یک پنج‌تایی تعریف می‌شود: $M = (Q, \Sigma, \delta, q_\circ, F)$
- Q مجموعه‌ای متناهی از حالت‌های داخلی [1] است.
- Σ مجموعه‌ای متناهی از نمادها به نام الفبای ورودی [2] است.
- $\delta : Q \times \Sigma \rightarrow Q$ تابعی کامل به نام تابع گذار [3] است.
- $q_\circ \in Q$ حالت اولیه [4] است.
- $F \subseteq Q$ مجموعه‌ای از حالت‌های پایانی [5] است.

[1] internal states
[2] input alphabet
[3] transition function
[4] initial state
[5] final states

ماشین متناهی قطعی

یک دی‌اف‌ای بدین صورت عمل می‌کند:

- در نقطهٔ زمانی اولیه در حالت اولیهٔ q_0 قرار دارد و سازوکار ورودی خواندن رشتهٔ ورودی را از اولین نماد (از سمت چپ) رشتهٔ ورودی آغاز می‌کند.
- در هر حرکت ماشین، یک نماد از ورودی خوانده می‌شود، و سپس سازوکار ورودی یک سلول به سمت راست حرکت می‌کند.
- سازوکار ورودی فقط از چپ به راست حرکت می‌کند و فقط یک نماد در هر واحد زمان (در هر گام یا لحظه) از یک سلول خوانده می‌شود.
- گذار از یک حالت به حالت دیگر توسط تابع دلتا تعیین می‌شود، مثلا $\delta(q_0, a) = q_1$ بدین معنی که با خواندن نماد a در صورتی که ماشین در حالت q_0 باشد، به حالت q_1 می‌رود.
- بعد از خواندن پایان رشته، اگر ماشین در یکی از حالت‌های پایانی باشد، رشته پذیرش می‌شود، در غیراینصورت رد می‌شود.

ماشین متناهی قطعی

- برای نمایش یک دی‌اف‌ای از یک گراف گذار [1] استفاده می‌کنیم.
- رأس‌ها حالت‌ها و یال‌ها توابع گذار را نمایش می‌دهند. برچسب روی یک یال نمادی را نشان می‌دهد که توسط آن، ماشین از یک حالت به حالت دیگر گذار می‌کند.
- حالت اولیه با یک خط ورودی و حالت‌های پایانی با دو دایره مشخص می‌شوند.

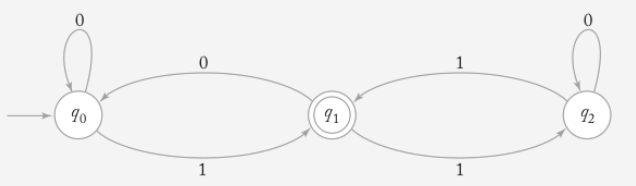
[1] transition graph

ماشین متناهی قطعی

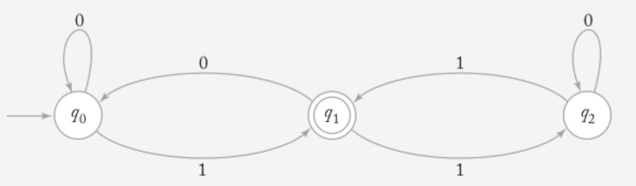
- پس ماشین M با گراف G_M نشان داده می‌شود، به طوری که گراف $|Q|$ حالت دارد و نام هر حالت $q_i \in Q$ است.
- به ازای هر تابع گذار $\delta(q_i, a) = q_j$ گراف یالی از q_i به q_j با برچسب a دارد.
- رأس q_0 رأس آغازی و رأس‌های $q_f \in F$ رأس‌های پایانی نام دارند.

ماشین متناهی قطعی

- گراف زیر دی‌اف‌ای $M = (\{q_۰, q_۱, q_۲\}, \{۰, ۱\}, \delta, q_۰, \{q_۱\})$ را نشان می‌دهد به طوری که :
- $\delta(q_۰, ۰) = q_۰ \quad , \quad \delta(q_۰, ۱) = q_۱ \quad , \quad \delta(q_۱, ۰) = q_۰ \quad , \quad \delta(q_۱, ۱) = q_۲ \quad , \quad \cdots$
- این ماشین چه رشته‌هایی را می‌پذیرد؟



- این ماشین رشته‌هایی را می‌پذیرد که تعداد یک‌های پی‌درپی در پایان آن فرد باشد.
- $\mathrm{L} = \{\mathrm{w}۱^{(۲\mathrm{n}+۱)} : \mathrm{w} \in \{۰, ۱\}^*, \mathrm{n} \geq ۰\}$



ماشین متناهی قطعی

- همچنین می‌توانیم یک تابع گذار تعمیم‌یافته بدین صورت تعریف کنیم: $\delta^* : Q \times \Sigma^* \to Q$
- در اینصورت دومین پارامتر تابع δ^* یک رشته است، به طوری که اگر $\delta(q_0, a) = q_1$ و $\delta(q_1, b) = q_2$ آنگاه $\delta^*(q_0, ab) = q_2$
- می‌توانیم به صورت بازگشتی تعریف کنیم: $\delta^*(q, \lambda) = q \quad , \quad \delta^*(q, wa) = \delta(\delta^*(q, w), a)$ به طوری که $q \in Q$ و $w \in \Sigma^*$ و $a \in \Sigma$

ماشین متناهی قطعی و زبان‌ها

- زبانی که توسط ماشین $M = (Q, \Sigma, \delta, q_\circ, F)$ پذیرفته می‌شود، مجموعهٔ همهٔ رشته‌ها است بر روی Σ که توسط ماشین M پذیرفته می‌شوند:
- $L(M) = \{w \in \Sigma^* : \delta^*(q_\circ, w) \in F\}$
- توجه کنید که δ و δ^* توابع کامل هستند، یعنی همیشه به ازای یک ورودی تنها یک خروجی برای تابع تعریف شده است. قطعیت در یک ماشین قطعی به همین دلیل است.
- به ازای هر رشتهٔ ورودی، ماشین یا رشته را قبول می‌کند و یا رد می‌کند.
- رشته‌ای که رد می‌شود در یک حالت غیر پایانی خاتمه می‌یابد و در زبان متمم $L(M)$ قرار دارد:
- $\overline{L(M)} = \{w \in \Sigma^* : \delta^*(q_\circ, w) \notin F\}$

ماشین متناهی قطعی و زبان‌ها

- ماشین زیر چه زبانی را می‌پذیرد؟

ماشین متناهی قطعی و زبان‌ها

- ماشین زیر چه زبانی را پذیرش می‌کند؟
- $\mathrm{L} = \{\mathrm{a}^{\mathrm{n}}\mathrm{b} : \mathrm{n} \geq ۰\}$
- در اینجا حالت $\mathrm{q}_{۲}$ حالت تله [1] (دام) نامید می‌شود.

---

[1] trap state

ماشین متناهی قطعی و زبان‌ها

- تابع گذار می‌تواند همچنین به صورت یک جدول نشان داده شود.
- در اینجا نام هر سطر حالت فعلی و نام هر ستون نماد خوانده شده از ورودی است.
- خانه $c_{i,j}$ از جدول، حالتی را نشان می‌دهد که ماشین بعد از مشاهده نماد ورودی j در صورتی که در حالت i باشد، به آن می‌رود.

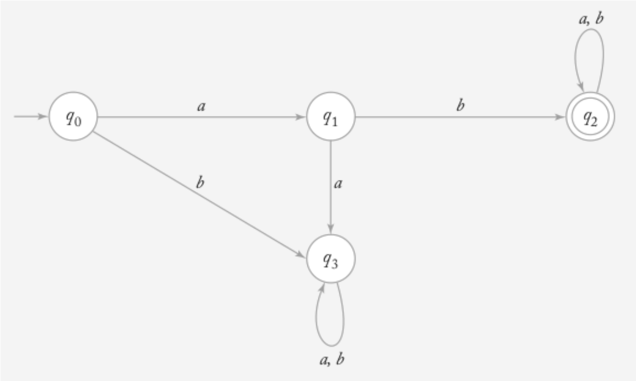
	a	b
$q_۰$	$q_۰$	$q_۱$
$q_۱$	$q_۲$	$q_۲$
$q_۲$	$q_۲$	$q_۲$

ماشین متناهی قطعی و زبان‌ها

- یک دی‌اف‌ای طراحی کنید که زبانی را شناسایی کند که بر روی الفبای $\Sigma = \{a, b\}$ تعریف شده است و پیشوند همهٔ جمله‌های آن ab باشد.

ماشین متناهی قطعی و زبان‌ها

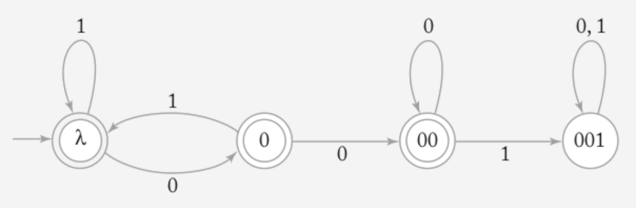
- یک دی‌اف‌ای طراحی کنید که زبانی را شناسایی کند که بر روی الفبای $\Sigma = \{a, b\}$ تعریف شده است و پیشوند همهٔ جمله‌های آن ab باشد.



ماشین متناهی قطعی و زبان‌ها

- یک دی‌اف‌ای طراحی کنید که زبانی را شناسایی کند که در آن هیچ جمله‌ای شامل زیررشتهٔ ۰۰۱ نباشد.

– یک دی‌اف‌ای طراحی کنید که زبانی را شناسایی کند که در آن هیچ جمله‌ای شامل زیررشتهٔ ۰۰۱ نباشد.



زبان‌های منظم

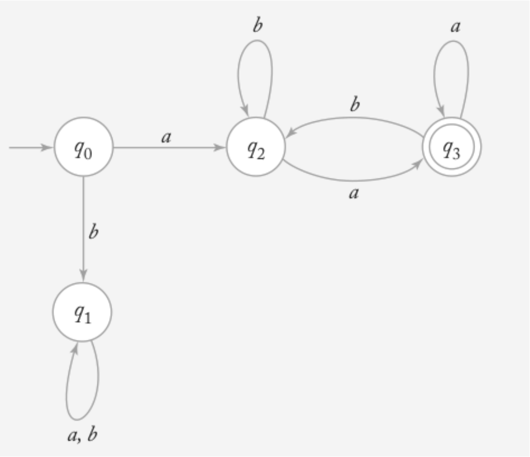
- ماشین متناهی قطعی خانواده‌ای از زبان‌ها را شناسایی می‌کند که به آنها زبان‌های منظم می‌گوییم.
- بنابراین زبان L منظم نامیده می‌شود اگر و تنها اگر یک پذیرندهٔ متناهی قطعی M وجود داشته باشد به طوری که $\mathrm{L} = \mathrm{L}(\mathrm{M})$.

زبان‌های منظم

- نشان دهید زبان $L = \{awa : w \in \{a, b\}^*\}$ منظم است.

زبان‌های منظم

- نشان دهید زبان $L = \{awa : w \in \{a, b\}^*\}$ منظم است.

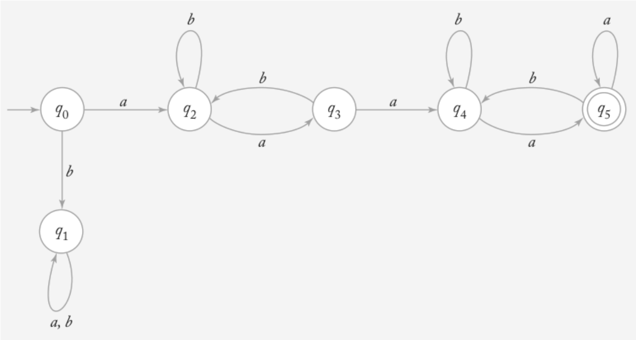


زبان‌های منظم

- نشان دهید زبان L^2 منظم است جایی که $L = \{awa : w \in \{a, b\}^*\}$.

زبان‌های منظم

- نشان دهید زبان $L^۲$ منظم است به طوری که $L = \{awa : w \in \{a, b\}^*\}$.
- $L^۲ = \{aw_۱aaw_۲a : w \in \{a, b\}^*\}$



ماشین متناهی غیرقطعی

- در ماشین متناهی قطعی، در هر حالت به ازای هر نماد فقط یک امکان برای گذار وجود دارد، به عبارت دیگر تابع δ یک تابع کامل است.
- در ماشین غیرقطعی به ازای یک نماد چندین گذار ممکن وجود دارد.

ماشین متناهی غیرقطعی

- یک ماشین یا پذیرندهٔ متناهی غیرقطعی (ان‌اف‌ای) [1] با یک پنج‌تایی تعریف می‌شود:
$M = (Q, \Sigma, \delta, q_۰, F)$
- به طوری که Q و Σ و $q_۰$ و F مانند تعریف ماشین متناهی قطعی تعریف می‌شوند، اما
$\delta : Q \times (\Sigma \cup \{\lambda\}) \to ۲^Q$
- ان‌اف‌ای و دی‌اف‌ای چند تفاوت عمده دارند : (۱) برد تابع دلتا عضوی است از مجموعه‌توانی حالت‌ها و (۲) ماشین توسط رشتهٔ تهی یا به عبارتی بدون خواندن ورودی نیز می‌تواند گذار انجام دهد، و (۳) برد تابع می‌تواند یک مجموعهٔ تهی باشد، بنابراین به ازای یک پیکربندی ممکن است گذاری تعریف نشده باشد.

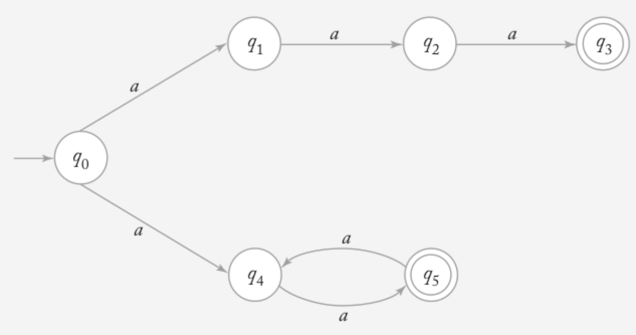
[1] nondeterministic finite acceptor/automata (nfa)

ماشین متناهی غیرقطعی

- گراف ان‌اف‌ای شبیه گراف دی‌اف‌ای است با این تفاوت که (۱) یال (q_i, q_j) با برچسب a موجود است اگر q_j عضو مجموعهٔ $\delta(q_i, a)$ باشد، و (۲) بر روی یال‌ها ممکن است برچسب λ باشد.
- یک رشته توسط ماشین پذیرفته می‌شود اگر حرکت‌هایی وجود داشته باشند که توسط آنها ماشین به یک حالت نهایی برسد، در غیر اینصورت رشته رد می‌شود.
- پس در صورتی که ماشینی رشته‌ای را بپذیرد، باید حرکت‌ها را برای رسیدن به حالت نهایی حدس زد.

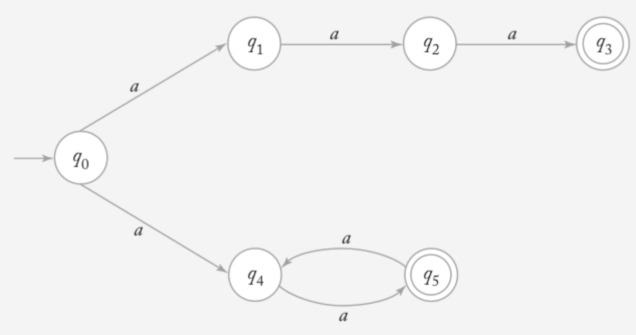
ماشین متناهی غیرقطعی

- ماشین زیر یک ماشین متناهی غیرقطعی است. این ماشین چه زبانی را می پذیرد؟



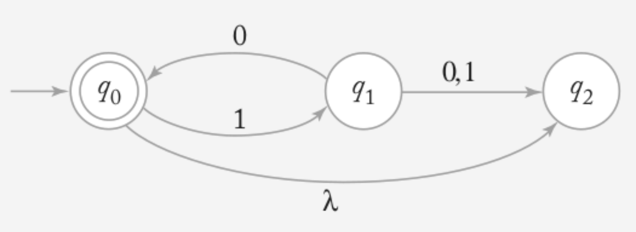
ماشین متناهی غیرقطعی

- این ماشین چه زبانی را می پذیرد؟ $L = \{a^{۳}\} \cup \{a^{۲n} : n \geq ۱\}$

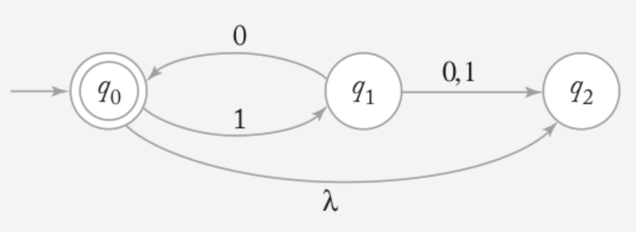


ماشین متناهی غیرقطعی

- ماشین زیر یک ماشین متناهی غیرقطعی است به دلیل (۱) وجود گذار با رشته تهی، (۲) بدون تابع گذار برای حالت q_2 و (۳) وجود دو گذار با نماد صفر از q_1.
- این ماشین چه رشته‌هایی را می‌پذیرد؟ آیا رشته ۱۰۱۰۰ را می‌پذیرد؟



- این ماشین چه رشته‌هایی را می‌پذیرد؟
- $L = \{(۱۰)^n : n \geq ۰\} = \{\lambda, ۱۰, ۱۰۱۰, ۱۰۱۰۱۰, \cdots\}$

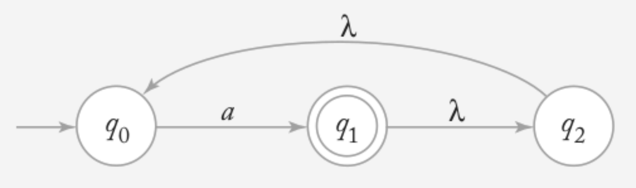


ماشین متناهی غیرقطعی

- برای یک ان‌اف‌ای، یک تابع گذار تعمیم یافته، به صورت $\delta^*(q_i, w)$ مجموعه‌ای است که شامل q_j می‌شود اگر و فقط اگر گشتی بر روی گراف گذار آن از q_i به q_j با خواندن رشتهٔ w وجود داشته باشد، به طوری که $q_i, q_j \in Q$ و $w \in \Sigma^*$.

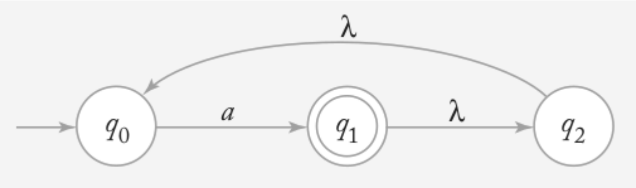
ماشین متناهی غیرقطعی

- برای ان‌اف‌ای زیر مقادیر $\delta^*(q_1, a)$، $\delta^*(q_2, \lambda)$، و $\delta^*(q_2, aa)$ را بیابید.



ماشین متناهی غیرقطعی

- برای ان‌اف‌ای زیر مقادیر $\delta^*(q_1, a)$، $\delta^*(q_2, \lambda)$، و $\delta^*(q_2, aa)$ را بیابید.
- $\delta^*(q_1, a) = \{q_0, q_1, q_2\}$
- $\delta^*(q_2, \lambda) = \{q_0, q_2\}$
- $\delta^*(q_2, aa) = \{q_0, q_1, q_2\}$



ماشین متناهی غیرقطعی

- در یک ان‌اف‌ای، طول یک گشت [1] بر روی گراف برای پیدا کردن $\delta^*(q_i, w)$ حداکثر چقدر می‌تواند باشد؟
- از آنجایی که گذارهای λ طول گشت را اضافه می‌کنند، باید برای طول گشت مقداری حداکثری پیدا کنیم، در غیر اینصورت الگوریتم جستجو نمی‌داند در چه نقطه‌ای متوقف شود.
- اگر بین دو رأس v_i و v_j گشتی با خواندن رشتهٔ w وجود داشته باشد، آنگاه طول این گشت در گراف گذار نمی‌تواند بزرگ‌تر از $|w|(\Lambda + ۱) + \Lambda$ باشد، به طوری که Λ تعداد یال‌ها با برچسب تهی در گراف است (با فرض اینکه هیچ دوری از یال‌های تهی به طور پی‌درپی در این گشت تکرار نمی‌شود).
- این رابطه را ثابت کنید.

[1] walk

ماشین متناهی غیرقطعی

- اگر بین دو رأس v_i و v_j گشتی با خواندن رشتهٔ w وجود داشته باشد، آنگاه طول این گشت در گراف گذار نمی‌تواند بزرگ‌تر از $|w|(\Lambda + ۱) + \Lambda$ باشد، به طوری که Λ تعداد یال‌ها با برچسب تهی در گراف است (با فرض اینکه هیچ دوری از یال‌های تهی به طور پی‌درپی در این گشت تکرار نمی‌شود).
- یک گشت برای رشتهٔ w به طول n به این شکل است: $e_۱w_۱e_۲w_۲ \cdots e_nw_ne_{n+۱}$
- به طوری که e_i یک دور است که روی همهٔ یال‌های آن برچسب تهی است و w_i یکی از نمادهای رشتهٔ w است.
- طول این گشت حداکثر برابر است با : $n + (n + ۱)\Lambda$
- $n + (n + ۱)\Lambda = \Lambda + (۱ + \Lambda)n$ به طوری که $n = |w|$

ماشین متناهی غیرقطعی

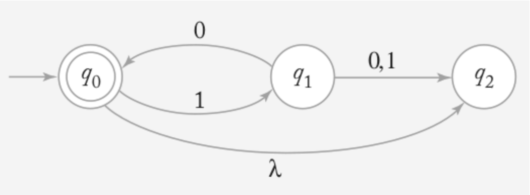
- زبان L که توسط ماشین متناهی غیرقطعی $M = (Q, \Sigma, \delta, q_{\circ}, F)$ پذیرفته می‌شود، مجموعه‌ای است از رشته‌ها که به صورت زیر تعریف می‌شود:

- $L(M) = \{w \in \Sigma^* : \delta^*(q_{\circ}, w) \cap F \neq \emptyset\}$

- به عبارت دیگر، زبانی که توسط یک ماشین متناهی غیرقطعی پذیرفته می‌شود، مجموعه ای از رشته‌های w است به طوری که گشتی با خواندن رشتهٔ w بر روی گراف گذار با شروع از رأس آغازی و پایان در یکی از رأس‌های پایانی وجود داشته باشد.

ماشین متناهی غیرقطعی

- اگر در یک ان‌اف‌ای، داشته باشیم $\delta^*(q_0, w) = \emptyset$ آنگاه می‌گوییم با دریافت رشتهٔ w ، ماشین به پیکربندی مرده [1] می‌رسد یا به بن‌بست برمی‌خورد.
- در ان‌اف‌ای زیر، با دریافت رشتهٔ ۰۰۱۰۱ یا ۱۱۰ ماشین به بن‌بست برمی‌خورد.
- وقتی ماشین به بن‌بست برمی‌خورد، رشته را رد می‌کند.



[1] dead configuration

ماشین متناهی غیرقطعی

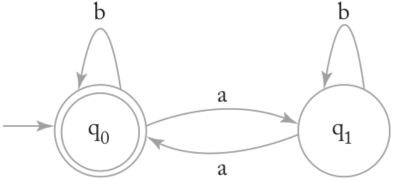
- دلیل مطالعهٔ عدم قطعیت چیست؟
- عدم قطعیت در بسیاری از مسائل محاسباتی کاربرد دارد. به طور مثال در یک بازی (مانند بازی شطرنج) بررسی تمام حالات ممکن امکان پذیر نیست، بنابراین پس از بررسی تعداد زیادی از حالات، یکی از گزینه‌های پیش رو انتخاب می‌شود و بازی ادامه پیدا می‌کند تا در آینده نتایج انتخاب مشخص‌تر شود.
- گاهی طراحی یک ماشین قطعی برای یک زبان ساده نیست، ولی طراحی ماشین غیرقطعی به راحتی امکان پذیر است. برای مثال برای طراحی یک ماشین برای زبانی که اجتماع دو زبان باشد، می‌توان ماشینی طراحی کرد که شروع آن به شروع ماشین هر دو زبان متصل شود.
- در آینده نشان می‌دهیم که ماشین‌های قطعی و غیرقطعی معادل یکدیگرند، بنابراین هر دو یک دسته از زبان‌ها را شناسایی می‌کنند. پس اگر ماشینی متناهی غیرقطعی برای یک زبان پیدا کنیم، ماشین متناهی قطعی آن نیز وجود خواهد داشت.

مثال

- به ازای $\Sigma = \{a, b\}$، یک ماشین متناهی قطعی طراحی کنید که همهٔ رشته‌ها با تعداد زوج a را بپذیرد.

مثال

- به ازای $\Sigma = \{a, b\}$، یک ماشین متناهی قطعی طراحی کنید که همهٔ رشته‌ها با تعداد زوج a را بپذیرد.

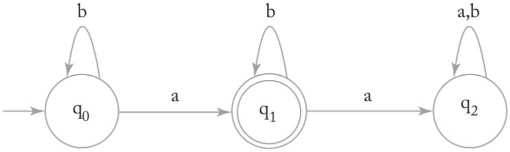


مثال

- به ازای $\Sigma = \{a, b\}$، یک ماشین متناهی قطعی طراحی کنید که همهٔ رشته‌هایی که فقط یک a دارند را بپذیرد.

مثال

- به ازای $\Sigma = \{a, b\}$، یک ماشین متناهی قطعی طراحی کنید که همهٔ رشته‌هایی که فقط یک a دارند را بپذیرد.

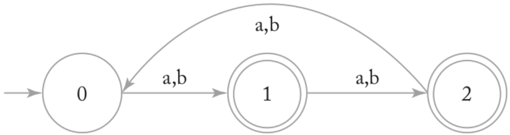


مثال

- به ازای $\Sigma = \{a, b\}$، یک ماشین متناهی قطعی طراحی کنید که زبان $\{w : |w| \bmod ۳ \neq ۰\}$ را بپذیرد.

مثال

- به ازای $\Sigma = \{a, b\}$، یک ماشین متناهی قطعی طراحی کنید که زبان $\{w : |w| \bmod ۳ \neq ۰\}$ را بپذیرد.

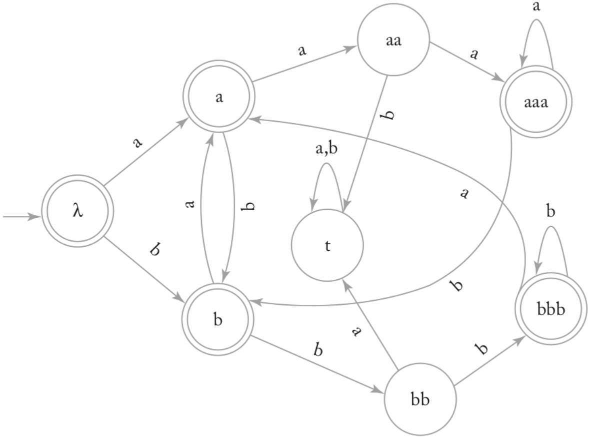


مثال

- به ازای $\Sigma = \{a, b\}$، یک ماشین متناهی قطعی طراحی کنید برای زبانی که در جملات آن طول هیچ زیر رشته‌ای از a های متوالی و همچنین b های متوالی برابر با ۲ نباشد.

مثال

- به ازای $\Sigma = \{a, b\}$، یک ماشین متناهی قطعی طراحی کنید برای زبانی که در جملات آن طول هیچ زیر رشته‌ای از a های متوالی و همچنین b های متوالی برابر با ۲ نباشد.

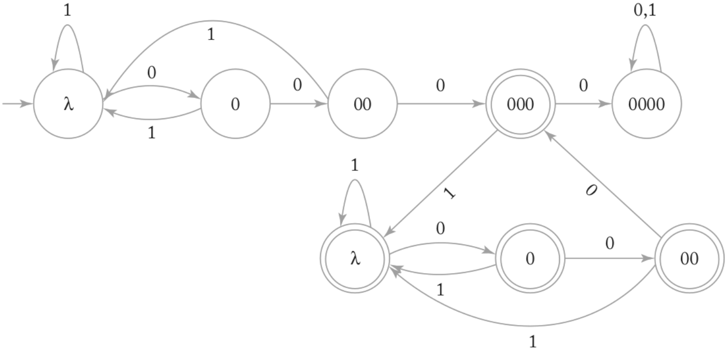


مثال

- به ازای $\Sigma = \{0, 1\}$، یک ماشین متناهی قطعی طراحی کنید برای زبانی که در جملات آن زیر رشتهٔ 000 وجود داشته باشد، ولی زیررشتهٔ 0000 وجود نداشته باشد.

مثال

- به ازای $\Sigma = \{۰, ۱\}$، یک ماشین متناهی قطعی طراحی کنید برای زبانی که در جملات آن زیر رشتهٔ ۰۰۰ وجود داشته باشد، ولی زیررشتهٔ ۰۰۰۰ وجود نداشته باشد.

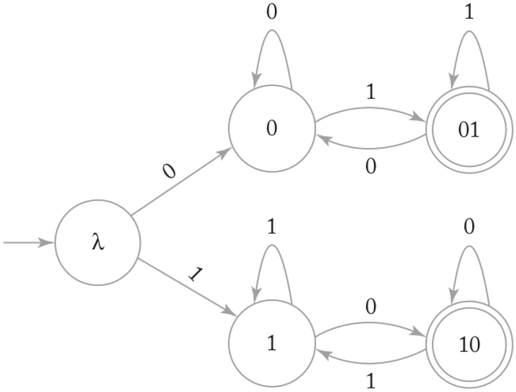


مثال

- به ازای $\Sigma = \{0, 1\}$، یک ماشین متناهی قطعی طراحی کنید برای زبانی که در هر جملهٔ آن اولین نماد و آخرین نماد متفاوت باشند.

مثال

- به ازای $\Sigma = \{۰, ۱\}$، یک ماشین متناهی قطعی طراحی کنید برای زبانی که در هر جملهٔ آن اولین نماد و آخرین نماد متفاوت باشند.

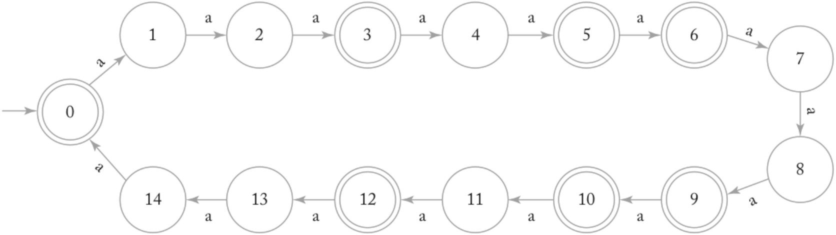


مثال

- به ازای $\Sigma = \{a\}$، یک ماشین متناهی قطعی طراحی کنید برای زبان $\{a^n\}$ به طوری که n مضرب ۳ یا ۵ باشد.

مثال

- به ازای $\Sigma = \{a\}$، یک ماشین متناهی قطعی طراحی کنید برای زبان $\{a^n\}$ به طوری که n مضرب ۳ یا ۵ باشد.

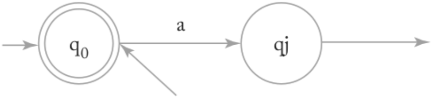


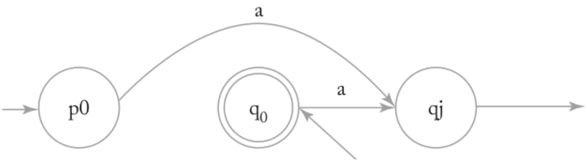
مثال

- نشان دهید اگر L منظم باشد، آنگاه $L - \{\lambda\}$ نیز منظم است.

مثال

- نشان دهید اگر L منظم باشد، آنگاه $L - \{\lambda\}$ نیز منظم است.
- نشان می‌دهیم همیشه برای زبان $L - \{\lambda\}$ یک ماشین متناهی قطعی وجود دارد.



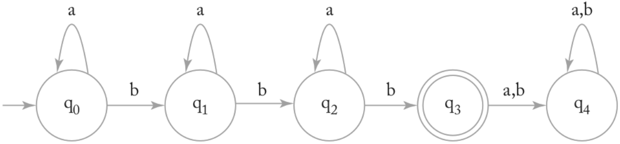


مثال

- فرض کنید $L = \{a^n b : n \geq 0\}$، آنگاه یک ماشین متناهی قطعی برای زبان L^3 طراحی کنید.

مثال

- فرض کنید $L = \{a^n b : n \geq ۰\}$، آنگاه یک ماشین متناهی قطعی برای زبان $L^۳$ طراحی کنید.
- $L^۳ = \{a^n b a^m b a^p b : n, m, p \geq ۰\}$

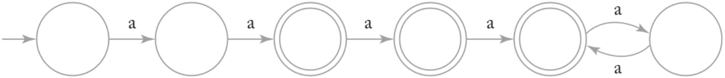


مثال

- یک ماشین متناهی قطعی برای زبان $L = \{a^{۳}\} \cup \{a^n : n \bmod ۲ = ۰\}$، طراحی کنید.

مثال

- یک ماشین متناهی قطعی برای زبان $L = \{a^۳\} \cup \{a^n : n \bmod ۲ = ۰\}$، طراحی کنید.

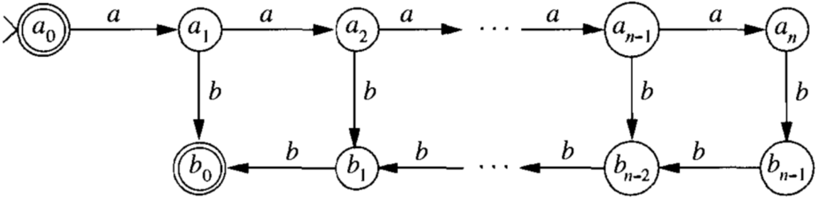


مثال

- برای زبان $L = \{a^k b^k : k \leq n\}$ یک ماشین متناهی قطعی (در حالت کلی به ازای n معلوم) طراحی کنید.

مثال

- برای زبان $L = \{a^k b^k : k \le n\}$ یک ماشین متناهی قطعی (در حالت کلی به ازای n معلوم) طراحی کنید.

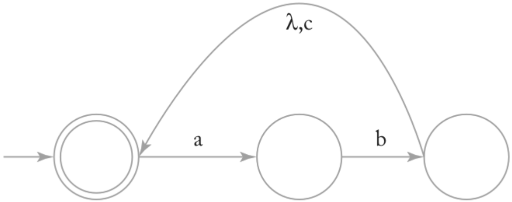


مثال

- برای زبان $L = \{ab, abc\}^*$ یک ماشین متناهی غیرقطعی با سه حالت طراحی کنید.

مثال

- برای زبان $L = \{ab, abc\}^*$ یک ماشین متناهی غیرقطعی با سه حالت طراحی کنید.



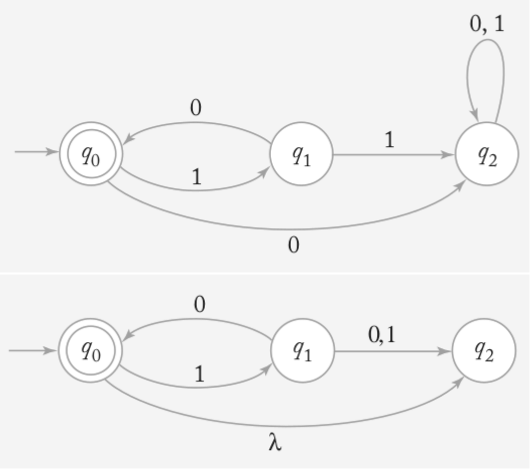
هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

- دو ماشین متناهی M_1 و M_2 هم‌ارز [1] هستند، اگر $L(M_1) = L(M_2)$
- پس دو ماشین هم‌ارز هستند اگر هر دو یک زبان را شناسایی کنند.

[1] equivalent

هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

- آیا دو ماشین متناهی قطعی و غیر قطعی زیر هم‌ارزند؟



هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

- وقتی دو رده (طبقه یا کلاس) [1] از ماشین‌ها را با هم مقایسه می‌کنیم، سؤالی که مطرح می‌شود این است که آیا یک رده از ماشین‌ها از ردهٔ دیگر قدرتمندتر است یا خیر.
- یک ماشین قدرتمندتر نسبت به ماشین دیگر، زبانی را می‌پذیرد که ابرمجموعهٔ زبان آن ماشین دیگر است.
- از آنجایی که دی‌اف‌ای نوع محدود شده‌ای از ان‌اف‌ای است، زبانی که با یک دی‌اف‌ای پذیرفته می‌شود، توسط یک ان‌اف‌ای نیز پذیرفته می‌شود.
- اما آیا برای زبانی که توسط یک ان‌اف‌ای شناسایی می‌شود، می‌توان یک دی‌اف‌ای طراحی کرد؟

[1] class

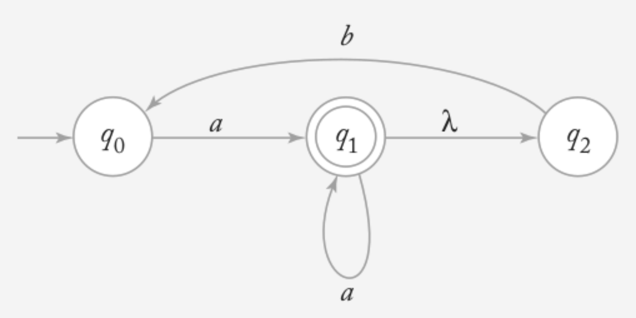
هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

- دو ماشین متناهی قطعی و غیرقطعی هم‌ارزند، و بنابراین ماشین غیرقطعی از ماشین قطعی قدرتمندتر نیست.
- این گزاره را با استفاده از برهان از طریق ساخت [1] (برهان با ساخت) اثبات می‌کنیم.
- روشی ارائه می‌کنیم که با آن هر ان‌اف‌ای را می‌توان به یک دی‌اف‌ای تبدیل کرد.
- به طور خلاصه، در یک ان‌اف‌ای، از هر حالت با خواندن یک نماد، به مجموعه‌ای از حالت‌ها می‌رویم. برای پیدا کردن دی‌اف‌ای معادل آن، هر یک از مجموعه حالت‌های ان‌اف‌ای باید یک حالت متمایز در دی‌اف‌ای باشد.
- پس معادل دی‌اف‌ای یک ان‌اف‌ای با $|Q|$ حالت، حداکثر $۲^{|Q|}$ حالت دارد.

[1] proof by construction (constructive proof)

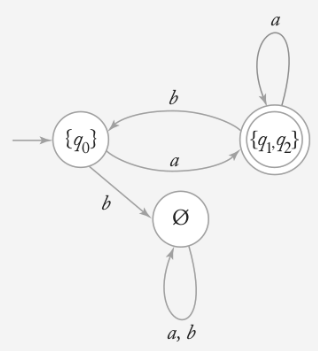
هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

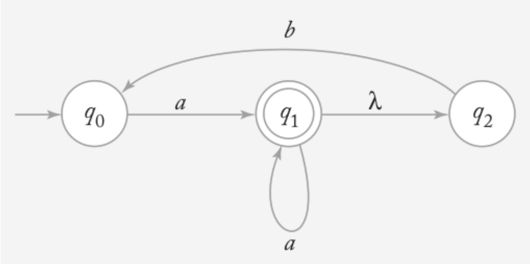
- ماشین متناهی قطعی معادل (هم‌ارز) ماشین متناهی غیرقطعی زیر را پیدا کنید.



هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

- ماشین‌های متناهی قطعی و غیرقطعی زیر معادل (هم‌ارز) یکدیگرند.





- قضیه: فرض کنید L زبانی باشد که توسط پذیرندهٔ متناهی غیرقطعی $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ پذیرفته می‌شود. در اینصورت یک پذیرندهٔ متناهی قطعی $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ وجود دارد، به طوری که $L = L(M_D)$

هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

- قضیه: فرض کنید L زبانی باشد که توسط پذیرندهٔ متناهی غیرقطعی $M_N = (Q_N, \Sigma, \delta_N, q_\circ, F_N)$ پذیرفته می‌شود. در اینصورت یک پذیرندهٔ متناهی قطعی $M_D = (Q_D, \Sigma, \delta_D, \{q_\circ\}, F_D)$ وجود دارد، به طوری که $L = L(M_D)$

- برای اثبات این قضیه از برهان با ساخت استفاده می‌کنیم.

- به ازای ماشین داده شدهٔ M_N از الگوریتم (روند) [1] تبدیل ان‌اف‌ای به دی‌اف‌ای استفاده می‌کنیم تا گراف گذار G_D را برای ماشین M_D بسازیم.

---

[1] procedure

هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

الگوریتم تبدیل ماشین غیرقطعی به ماشین قطعی:

۱. یک گراف G_D با رأس $\{q_0\}$ به عنوان رأس آغازی بسازید.

۲. گام‌های زیر را تکرار کنید تا جایی که گراف گذار ماشین قطعی کامل شود:

- یکی از رأس‌های $\{q_i, q_j, \cdots, q_k\}$ از گراف G_D را که هیچ یال خروجی برای یک نماد $a \in \Sigma$ ندارد را انتخاب کنید. همهٔ مقادیر $\delta_N^*(q_i, a), \delta_N^*(q_j, a), \cdots, \delta_N^*(q_k, a)$ را محاسبه کنید.
- اگر $\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \cdots \cup \delta_N^*(q_k, a) = \{q_l, q_m, \cdots, q_n\}$ باشد، رأسی با نام $\{q_l, q_m, \cdots, q_n\}$ برای گراف G_D بسازید، البته اگر این رأس موجود نیست.
- یالی با برچسب a از رأس $\{q_i, q_j, \cdots, q_k\}$ به رأس $\{q_l, q_m, \cdots, q_n\}$ در گراف G_D اضافه کنید.

۳. هر رأسی از گراف G_D که نام آن شامل $q_f \in F_N$ می‌شود را به عنوان یک رأس پایانی انتخاب کنید.

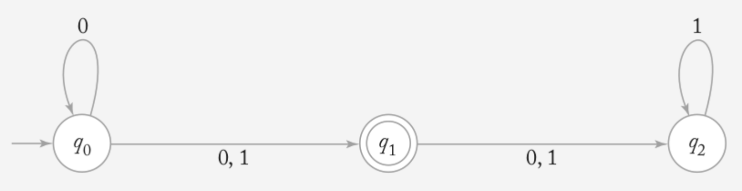
۴. اگر ماشین M_N رشتهٔ λ را می‌پذیرد، رأس $\{q_0\}$ در گراف G_D را به عنوان یک رأس پایانی انتخاب کنید.

هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

- این الگوریتم پایان می‌پذیرد و گرفتار حلقهٔ بی‌پایان نمی‌شود، زیرا گراف G_D حداکثر $|\Sigma|2^{|Q_N|}$ یال دارد، پس حلقه در نهایت متوقف می‌شود.
- برای اثبات درستی الگوریتم تبدیل ان‌اف‌ای به دی‌اف‌ای، می‌توان از برهان استقرایی (استقرا بر روی طول رشتهٔ ورودی) استفاده کرد.
- اگر برای رشتهٔ v با طول n ، وجود یک گشت از q_0 به q_i بر روی گراف G_N بر وجود گشتی بر روی گراف G_D از رأس $\{q_0\}$ به رأس $Q_i = \{\cdots q_i \cdots\}$ دلالت داشته باشد، آنگاه برای رشتهٔ $w = va$ با استفاده از الگوریتم تبدیل، وجود گشتی از q_0 به q_l به وجود یک گشتی از $\{q_0\}$ به یک رأس $Q_l = \{\cdots q_l \cdots\}$ دلالت خواهد داشت.

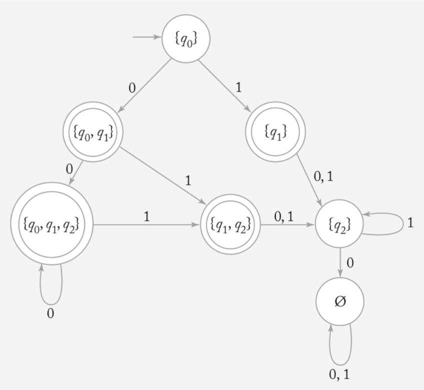
هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

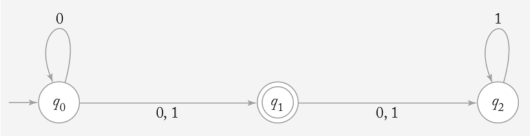
- یک دی‌اف‌ای هم‌ارز ان‌اف‌ای زیر طراحی کنید.



هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

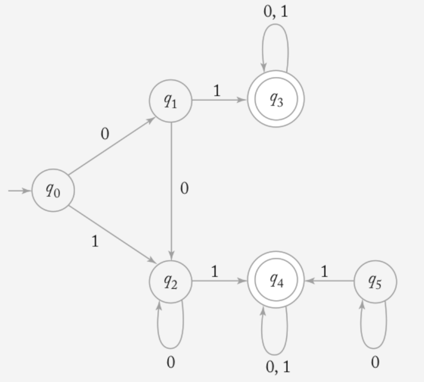
- ماشین‌های قطعی و غیرقطعی زیر با یکدیگر هم‌ارزند.

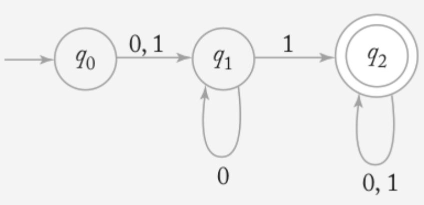




کاهش تعداد حالات در ماشین متناهی

- دو دی‌اف‌ای زیر با یکدیگر هم‌ارزند. اما در ماشین سمت چپ، حالت q_5 غیر قابل دسترسی است و در نتیجه می‌توان آن را حذف کرد. به علاوه، حالت‌های q_1 و q_2 و همچنین q_3 و q_4 در سمت چپ کاملا مشابه یکدیگرند.





کاهش تعداد حالات در ماشین متناهی

- از لحاظ نظری، دو ماشین که یک زبان را می‌پذیرند هیچ فرقی با یکدیگر ندارند.
- اما از لحاظ عملی، ماشین ساده‌تر و کوچک‌تر فضای کمتری را اشغال می‌کند.
- به علاوه، هر چه یک ماشین ساده‌تر نمایش داده شود، فهم عملکرد آن آسان‌تر می‌شود.

کاهش تعداد حالات در ماشین متناهی

- دو حالت p و q را در یک دی‌اف‌ای غیرمتمایز [1] می‌نامیم، اگر $\delta^*(q, w) \in F \implies \delta^*(p, w) \in F$ و همچنین $\delta^*(q, w) \notin F \implies \delta^*(p, w) \notin F$ به ازای هر $w \in \Sigma^*$
- اگر رشتهٔ $w \in \Sigma^*$ وجود داشته باشد به طوری که $\delta^*(p, w) \in F$ و $\delta^*(q, w) \notin F$ آنگاه حالت‌های p و q برای رشتهٔ w متمایز [2] هستند.

[1] indistinguishable
[2] distinguishable

کاهش تعداد حالات در ماشین متناهی

- بنابراین طبق تعریف، برای تشخیص دو حالت متمایز باید به ازای همهٔ رشته‌های w هر جفت حالت را با یکدیگر مقایسه کرد. یک جفت حالت غیرمتمایزند اگر از هر دو حالت با خواندن رشتهٔ w به یک حالت پایانی برسیم یا اگر از هر دو حالت با خواندن رشتهٔ w به یک حالت غیرپایانی برسیم.
- از آنجایی که تعداد رشته‌های ممکن زیاد است ($|\Sigma|^n$ برای رشته‌های با طول n)، لذا مسئله را ابتدا برای زیرمسئله‌ها با طول کوچکتر حل می‌کنیم.

کاهش تعداد حالات در ماشین متناهی

- برای کاهش حالات یک دی‌اف‌ای از دو الگوریتم (روند) استفاده می‌کنیم.
- ابتدا در الگوریتم اول به نام الگوریتم دسته‌بندی (علامت‌گذاری) حالات [1]، حالت‌های متمایز را مشخص می‌کنیم.
- سپس در الگوریتم دوم به نام الگوریتم کاهش تعداد حالات [2] دی‌اف‌ای کاهش یافته (مینیمال) را می‌سازیم.

[1] marking procedure
[2] reducing procedure

کاهش تعداد حالات در ماشین متناهی

- در الگوریتم دسته‌بندی حالات، ابتدا در گام صفر مسئله را برای رشته‌ها با طول صفر حل می‌کنیم.
- از نتایج گام صفر، در حل مسئله برای رشته‌های با طول یک استفاده می‌کنیم و به همین ترتیب از نتایج گام $n-1$ در حل مسئله در گام n برای رشته‌های با طول n استفاده می‌کنیم.
- بنابراین با استفاده از این روش گام به گام، دو حالت p و q در گام n غیرمتمایزند اگر با خواندن نماد a به ازای هر $a \in \Sigma$ از این دو حالت به دو حالت غیرمتمایز در گام $n-1$ برسیم.
- بنابراین برای دو حالت غیرمتمایز p و q در گام n داریم: $\delta(q,a) = q_a \quad , \quad \delta(p,a) = p_a$ به ازای هر $a \in \Sigma$ جایی که (p_a, q_a) یک جفت غیرمتمایز است در گام $n-1$.

کاهش تعداد حالات در ماشین متناهی

الگوریتم دسته‌بندی حالات:

۱. همهٔ حالت‌های غیرقابل‌دسترس را حذف کنید. پس از پیمایش گراف (جستجوی همهٔ مسیرها) از رأس آغازی، همهٔ رئوسی که غیر قابل پیمایش‌اند، رأس‌های غیرقابل دسترس‌اند.

۲. در گام $n = 0$ ، هر یک از جفت حالت p و q را در نظر بگیرید. اگر $p \in F$ و $q \notin F$ است، آنگاه این دو حالت را به عنوان حالت‌های متمایز در دو مجموعهٔ متفاوت قرار دهید.

کاهش تعداد حالات در ماشین متناهی

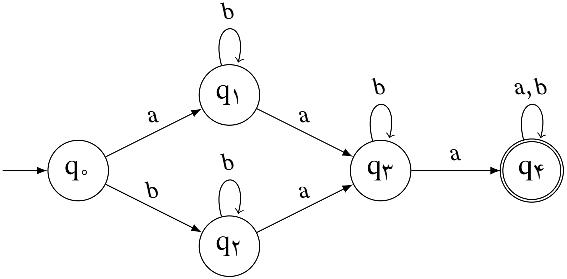
الگوریتم دسته‌بندی حالات:

۳. درگام $n+1$ همهٔ جفت‌هایی که غیرمتمایزند (درگام n در یک مجموعهٔ یکسان قرار دارند) را در نظر بگیرید. به ازای هر جفت غیرمتمایز (p,q) و به ازای هر $a \in \Sigma$ مقادیر $\delta(p,a) = p_a$ و $\delta(q,a) = q_a$ را محاسبه کنید. اگر جفت (p_a,q_a) متمایزند (درگام n در دو مجموعهٔ متفاوت قرار دارند)، جفت (p,q) را نیز به عنوان یک جفت متمایز در دو مجموعهٔ متفاوت قرار دهید.

۴. اگر هیچ تغییری در مجموعه‌های (دسته‌های) متمایز گام $n+1$ نسبت به مجموعه‌های متمایز گام n به وجود نیامد، همهٔ مجموعه‌های متمایز مشخص شده‌اند. بنابراین الگوریتم را خاتمه دهید. در غیراینصورت مقدار n را یک واحد افزایش دهید و به مرحلهٔ ۳ الگوریتم بازگرید.

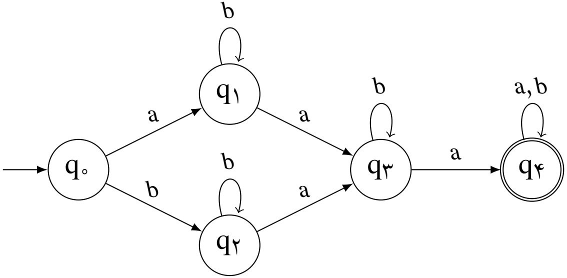
کاهش تعداد حالات در ماشین متناهی

- حالت‌های ماشین زیر را برای کاهش دادن تعداد حالات، دسته‌بندی کنید.



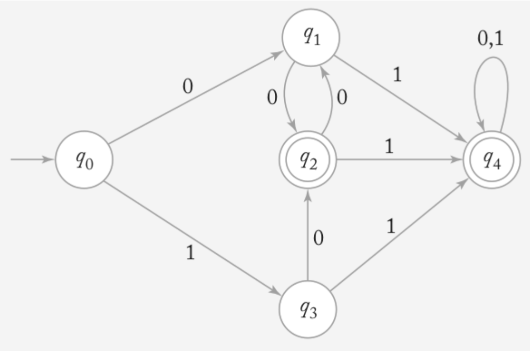
کاهش تعداد حالات در ماشین متناهی

$$n = ۰ : \quad \{q_۴\}, \quad \{q_۰, q_۱, q_۲, q_۳\}$$
$$n = ۱ : \quad \{q_۴\}, \quad \{q_۳\}, \quad \{q_۰, q_۱, q_۲\}$$
$$n = ۲ : \quad \{q_۴\}, \quad \{q_۳\}, \quad \{q_۰\}, \quad \{q_۱, q_۲\}$$
$$n = ۳ : \quad \{q_۴\}, \quad \{q_۳\}, \quad \{q_۰\}, \quad \{q_۱, q_۲\}$$



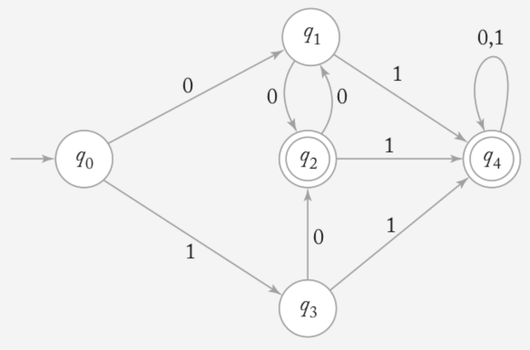
کاهش تعداد حالات در ماشین متناهی

- حالت‌های دی‌اف‌ای زیر را برای کاهش دادن تعداد حالات، دسته‌بندی کنید.



کاهش تعداد حالات در ماشین متناهی

- حالت‌های دی‌اف‌ای زیر را برای کاهش دادن تعداد حالات، دسته‌بندی کنید.
- $\{q_۰\}, \{q_۱, q_۳\}, \{q_۲\}, \{q_۴\}$



کاهش تعداد حالات در ماشین متناهی

الگوریتم کاهش تعداد حالات:

به ازای دی‌اف‌ای $M = (Q, \Sigma, \delta, q_0, F)$ دی‌اف‌ای کاهش‌یافتهٔ $\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\delta}, \widehat{q_0}, \widehat{F})$ را به صورت زیر می‌یابیم.

۱. با استفاده از الگوریتم دسته‌بندی حالات، همهٔ مجموعه‌های متمایز $\{q_i, q_j, \cdots, q_k\}$ را پیدا کنید.

۲. به ازای هر مجموعهٔ متمایز $\{q_i, q_j, \cdots, q_k\}$، یک حالت $ij\cdots k$ برای ماشین $\widehat{M}$ بسازید.

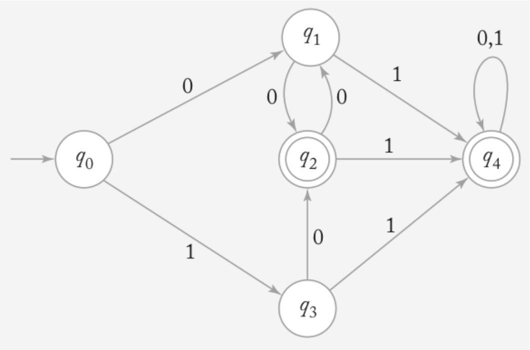
۳. به ازای هر گذار $\delta(q_r, a) = q_p$ در ماشین M مجموعه‌های متمایزی را بیابید که q_r و q_p متعلق به آنها هستند. اگر $q_r \in \{q_i, q_j, \cdots, q_k\}$ و $q_p \in \{q_l, q_m, \cdots, q_n\}$ باشند، برای تابع گذار $\widehat{\delta}$ چنین تعریف کنید: $\widehat{\delta}(ij\cdots k, a) = lm\cdots n$

۴. حالت آغازی $\widehat{q_0}$ در ماشین $\widehat{M}$ حالت $\cdots 0 \cdots$ است.

۵. مجموعهٔ حالات پایانی $\widehat{F}$ مجموعهٔ حالات $\cdots i \cdots$ است به طوری که $q_i \in F$.

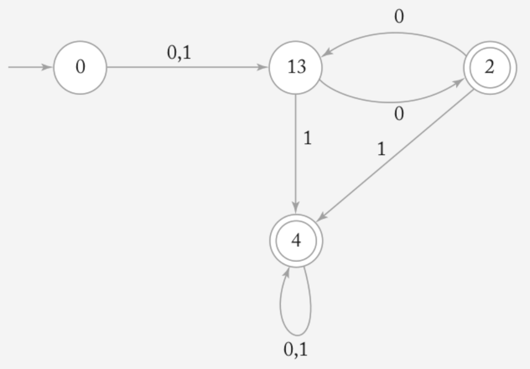
کاهش تعداد حالات در ماشین متناهی

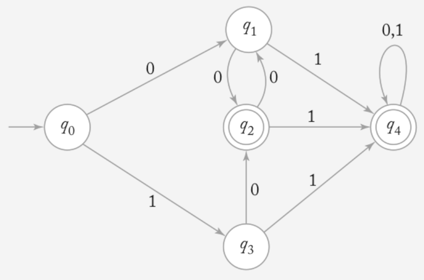
- پس از دسته‌بندی حالت‌های پذیرندهٔ قطعی زیر، مجموعه‌های متمایز $\{q_۰\}, \{q_۱, q_۳\}, \{q_۲\}, \{q_۴\}$ را به دست آوردیم. اکنون تعداد حالات این پذیرندهٔ متناهی قطعی را کاهش دهید.



کاهش تعداد حالات در ماشین متناهی

- با کاهش دادن حالات دی‌اف‌ای سمت راست، دی‌اف‌ای سمت چپ به دست می‌آید.



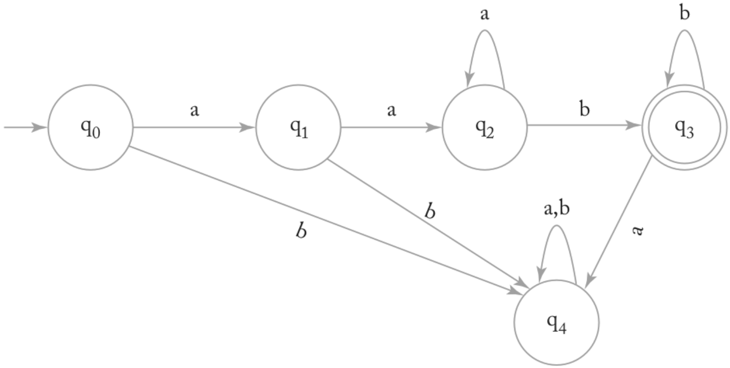


مثال

- برای زبان $L = \{a^n b^m : n \geq ۲, m \geq ۱\}$ یک ماشین متناهی قطعی با کمترین تعداد حالات طراحی کنید و سپس ثابت کنید که ماشین طراحی شده کمینه است.

مثال

- برای زبان $L = \{a^n b^m : n \geq ۲, m \geq ۱\}$ یک ماشین متناهی قطعی با کمترین تعداد حالات طراحی کنید و سپس ثابت کنید که ماشین طراحی شده کمینه است.
- الگوریتم تعداد کاهش حالات را بر روی این ماشین اعمال می‌کنیم و نشان می‌دهیم که همهٔ حالات متمایزند.



زبان‌های منظم

مقدمه

- پیشتر، از پذیرنده‌های متناهی برای شناسایی زبان‌های منظم استفاده کردیم.
- در این قسمت از عبارت‌های منظم و گرامرهای منظم برای توصیف زبان‌های منظم استفاده می‌کنیم.

زبان‌های منظم

- یکی از روش‌های توصیف زبان‌های منظم، استفاده از عبارت‌های منظم [1] است.
- یک عبارت منظم تشکیل شده است از رشته‌هایی بر روی الفبای Σ، پرانتزهای باز و بسته، و عملگرهای مثبت (+) و نقطه (.) و ستاره(*).
- برای مثال، زبان $\{a\}$ را با عبارت منظم a نشان می‌دهیم.
- عملگر مثبت (+) در عبارت‌های منظم به معنی اجتماع دو زبان (دو مجموعه) است. بنابراین زبان $\{a, b, c\}$ با عبارت منظم $a + b + c$ نمایش داده می‌شود.
- از عملگر نقطه (.) برای الحاق دو عبارت منظم و از عملگر ستاره (*) برای بستار-ستاره بر روی یک عبارت منظم استفاده می‌شود.
- عبارت $(a + (b \cdot c))^*$ به معنی بستار ستاره روی زبان $\{a\} \cup \{bc\}$ است، یعنی L^* جایی که $L = \{a, bc\}$. بنابراین زبان مورد نظر برابر است با $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \cdots\}$.

[1] regular expressions

- فرض کنید Σ یک الفبا باشد. آنگاه

۱. $\emptyset$، λ و $a \in \Sigma$ عبارت‌های منظم هستند که به آنها عبارت‌های منظم اولیه [1] می‌گوییم.

۲. اگر r_1 و r_2 دو عبارت منظم باشند، آنگاه $r_1 + r_2$، $r_1 \cdot r_1$، r_1^* و (r_1) نیز عبارت‌های منظم هستند.

۳. یک عبارت شامل عملگرها و نمادهای الفبا را یک عبارت منظم می‌گوییم اگر و فقط اگر بتوان آن را از عبارت‌های منظم اولیه با اعمال تعداد محدودی از قوانین مذکور در مرحلهٔ ۲ به دست آورد.

[1] primitive regular expressions

زبان‌های منظم

- اگر $\Sigma = \{a, b, c\}$ آنگاه رشتهٔ $(a + b \cdot c)^* \cdot (c + \emptyset)$ یک عبارت منظم است،
چرا که می‌توان آن را با اعمال چند قانون پی در پی از عبارت‌های منظم اولیه به دست آورد.
- ولی عبارت $(a + b+)$ یک عبارت منظم نیست.
- اگر r یک عبارت منظم باشد، آنگاه $L(r)$ نشان دهندهٔ زبان مربوط به عبارت منظم r است.

زبان‌های منظم

- زبان $L(r)$ که توسط عبارت منظم r تعیین شده است، با استفاده از قوانین زیر تعریف می‌شود:

۱. $\emptyset$ یک عبارت منظم است که نشان‌دهندهٔ زبان تهی $\{\}$ است.

۲. λ یک عبارت منظم است که نشان‌دهندهٔ زبان $\{\lambda\}$ است.

۳. به ازای هر $a \in \Sigma$، a یک عبارت منظم است که نشان‌دهندهٔ زبان $\{a\}$ است.

- اگر r_1 و r_2 دو عبارت منظم باشند، آنگاه

۴. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$

۵. $L(r_1 \cdot r_2) = L(r_1)L(r_2)$

۶. $L((r_1)) = L(r_1)$

۷. $L(r_1^*) = (L(r_1))^*$

زبان‌های منظم

- همچنین می‌توانیم از بستار مثبت روی یک عبارت منظم استفاده کنیم:
$$r^+ = rr^*$$

- برای الحاق و اجتماع و بستار–ستاره روابط زیر برقرار است:
$$r + \lambda = r + \lambda$$
$$r\lambda = r$$
$$\lambda^* = \lambda$$
$$r + \emptyset = r$$
$$r\emptyset = \emptyset$$
$\emptyset^* = \lambda$ (این رابطه یک قرارداد است مانند قرارداد $۰^۰ = ۱$)

زبان‌های منظم

- زبان $L(a^* \cdot (a+b))$ را با استفاده از مجموعه‌ها نشان دهید.

- زبان $L(a^* \cdot (a+b))$ را با استفاده از مجموعه‌ها نشان دهید.

$$
\begin{aligned}
L(a^* \cdot (a+b)) &= L(a^*)L((a+b)) \\
&= (L(a))^*(L(a) \cup L(b)) \\
&= \{\lambda, a, aa, \cdots\} \cdot \{a, b\} \\
&= \{a, aa, \cdots, b, ab, aab, \cdots\} \\
&= \{a^n : n \geq ۱\} \cup \{a^n b : n \geq ۰\}
\end{aligned}
$$

زبان‌های منظم

- زبان $L(a \cdot b + c)$ را با استفاده از مجموعه‌ها نشان دهید.

زبان‌های منظم

- زبان $L(a \cdot b + c)$ را با استفاده از مجموعه‌ها نشان دهید.
- بسته به تقدم عملگرها می‌توانیم این زبان را با $\{ab, c\}$ یا $\{ab, ac\}$ نشان دهیم.
- در تعریف عبارات منظم از تقدم صحبت نکردیم. با استفاده از تعریف داده شده عبارات منظم برای رفع ابهام یا باید کاملا پرانتزگذاری شده باشند، و یا اینکه عملگرهای سمت چپ تقدم بیشتری داشته باشند.
- برای سادگی، ما در اینجا تقدمی مانند عملیات ریاضی در نظر می‌گیریم. یعنی بستار ستاره (که مانند توان است) نسبت به الحاق (که مانند ضرب است) تقدم بیشتری دارد و همچنین الحاق نسبت به اجتماع (که مانند جمع است) تقدم بیشتری دارد.
- همچنین برای سادگی برای الحاق عملگر نقطه (.) را حذف می‌کنیم و می‌نویسیم ab به جای $a \cdot b$.

- زبان متناظر با عبارت منظم $(a + b)^*(a + bb)$ را با استفاده از مجموعه‌ها نشان دهید.

- زبان متناظر با عبارت منظم $(a+b)^*(a+bb)$ را با استفاده از مجموعه‌ها نشان دهید.
- قسمت اول یعنی $(a+b)^*$ نشان دهندهٔ همه رشته‌ها بر روی الفبای a و b است.
- این رشته‌ها در پایان به a یا bb الحاق می‌شوند.
- بنابراین زبان مورد نظر زبانی است از همهٔ رشته‌هایی که با a یا bb خاتمه پیدا می‌کنند.
- $\{wa : w \in \{a,b\}^*\} \cup \{wbb : w \in \{a,b\}^*\} = \{a, bb, aa, abb, ba, bbb, \cdots\}$

زبان‌های منظم

- زبان متناظر با عبارت منظم $r = (aa)^*(bb)^*b$ را با استفاده از مجموعه‌ها نشان دهید.

- زبان متناظر با عبارت منظم $r = (aa)^*(bb)^*b$ را با استفاده از مجموعه‌ها نشان دهید.
- $L(r) = \{a^{2n}b^{2m+1} : n \geq 0, m \geq 0\}$

زبان‌های منظم

- بر روی الفبای $\Sigma = \{0, 1\}$ عبارت منظم r را پیدا کنید به طوری که
- $L(r) = \{w \in \Sigma^* :$ حداقل دو صفر پی در پی داشته باشد $w\}$

- بر روی الفبای $\Sigma = \{۰, ۱\}$ عبارت منظم r را پیدا کنید به طوری که
- $L(r) = \{w \in \Sigma^* :$ حداقل دو صفر پی در پی داشته باشد $w\}$
- $r = (۰ + ۱)^*۰۰(۰ + ۱)^*$

- عبارت منظم r را پیدا کنید به طوری که
- $L(r) = \{w \in \{0, 1\}^* :$ دو صفر پی در پی ندارد $w\}$

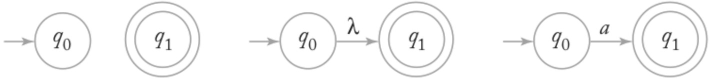
- عبارت منظم r را پیدا کنید به طوری که
- $L(r) = \{w \in \{۰, ۱\}^* :$ w دو صفر پی در پی ندارد$\}$
- $r = (۱ + ۰۱)^*(۰ + \lambda)$

عبارت‌ها و زبان‌های منظم

- فرض کنید r یک عبارت منظم باشد. آنگاه یک ماشین متناهی غیرقطعی وجود دارد که زبان $L(r)$ را می‌پذیرد. در اینصورت $L(r)$ یک زبان منظم است.

عبارت‌ها و زبان‌های منظم

- فرض کنید r یک عبارت منظم باشد. آنگاه یک ماشین متناهی غیرقطعی وجود دارد که زبان $L(r)$ را می‌پذیرد. در اینصورت $L(r)$ یک زبان منظم است.
- از برهان با ساخت استفاده می‌کنیم یعنی روشی ارائه می‌کنیم که از هر عبارت منظم بتوان یک ماشین غیرقطعی به دست آورد.
- با ماشین‌هایی شروع می‌کنیم که عبارت‌های منظم اولیهٔ $\emptyset$، λ و $a \in \Sigma$ را می‌پذیرند.
- این پذیرنده‌ها در زیر نشان داده شده‌اند.

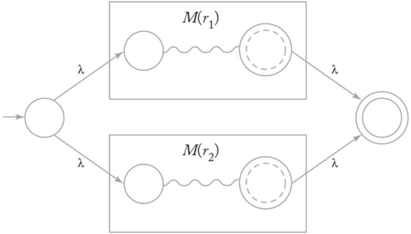


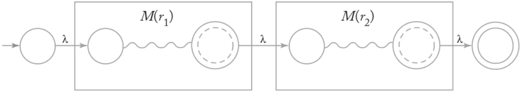
عبارت‌ها و زبان‌های منظم

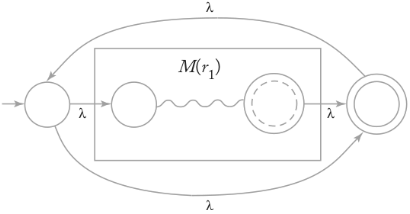
- اکنون فرض کنید ماشین‌های $M(r_1)$ و $M(r_2)$ زبان‌های منظمی را می‌پذیرند که با عبارت‌های منظم r_1 و r_2 نشان داده می‌شوند. برای سادگی در روند اثبات، هر ماشین ان‌اف‌ای را با معادل آن که تنها یک حالت پایانی دارد نشان می‌دهیم (با اتصال حالت‌های پایانی توسط رشته تهی به یک حالت پایانی واحد، این کار همیشه ممکن است).
- اکنون ماشین‌هایی می‌سازیم که زبان‌هایی را می‌پذیرند که با عبارت‌های منظم $r_1 + r_2$، r_1، r_1r_2 و r_1^* نمایش داده می‌شوند.

عبارت‌ها و زبان‌های منظم

- سه ماشین متناهی غیرقطعی برای پذیرفتن زبان‌های متناظر سه عبارت منظم $r_1 + r_2$، $r_1 r_2$ و r_1^* به صورت زیر ساخته می‌شوند.





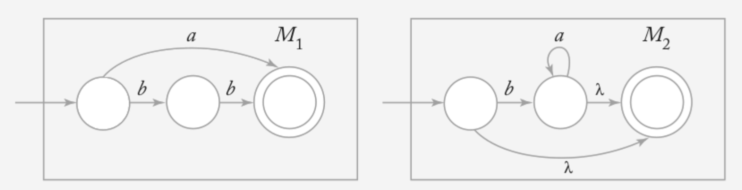


عبارت‌ها و زبان‌های منظم

- بنابراین برای هر عبارت منظم پیچیده‌ای می‌توانیم از ترکیب کردن این مراحل استفاده کنیم و ماشین متناهی غیرقطعی را که زبان متناظر با آن عبارت منظم را می‌پذیرد را بسازیم.

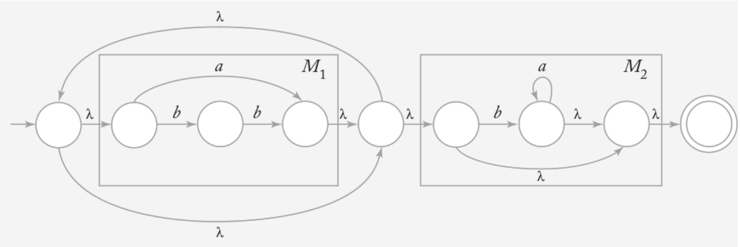
- یک ان‌اف‌ای طراحی کنید که زبان $L(r)$ را بپذیرد، به طوری که $r = (a + bb)^*(ba^* + \lambda)$

- یک ان‌اف‌ای طراحی کنید که زبان $L(r)$ را بپذیرد، به طوری که $r = (a + bb)^*(ba^* + \lambda)$
- ابتدا ماشین‌هایی طراحی می‌کنیم که عبارت‌های $(a + bb)$ و $(ba^* + \lambda)$ را بپذیرند.



عبارت‌ها و زبان‌های منظم

- یک ان‌اف‌ای طراحی کنید که زبان $L(r)$ را بپذیرد، به طوری که $r = (a + bb)^*(ba^* + \lambda)$
- ابتدا ماشین‌هایی طراحی می‌کنیم که عبارت‌های $(a + bb)$ و $(ba^* + \lambda)$ را بپذیرند.
- سپس با قوانین بستار و الحاق دو ماشین طراحی شده را با هم ترکیب می‌کنیم.



عبارت‌ها و زبان‌های منظم

- پس تا اینجا توانستیم برای هر عبارت منظم یک ماشین متناهی طراحی کنیم و از آنجایی که هر ماشین متناهی یک زبان منظم را می‌پذیرد، بنابراین برای هر عبارت منظم یک زبان منظم وجود دارد.
- آیا برای هر زبان منظم نیز یک عبارت منظم وجود دارد؟

عبارت‌ها و زبان‌های منظم

- از آنجایی که هر زبان منظم را می‌توان توسط یک ماشین متناهی نشان داد و هر ماشین متناهی یک گراف گذار دارد، پس برای به دست آوردن عبارت منظم متناظر آن کافی است همهٔ گشت‌ها از رأس آغازی به تمام حالات پایانی را با عبارات منظم وصف کرد.
- برای انجام این کار در اینجا از گراف گذار تعمیم‌یافته [1] استفاده می‌کنیم.
- ابتدا گراف گذار تعمیم‌یافته را تعریف می‌کنیم و سپس روشی برای تبدیل یک ماشین غیرقطعی به یک گراف گذار تعمیم‌یافته ارائه می‌کنیم.
- سپس روشی برای یافتن عبارت منظم متناظر با گراف گذار تعمیم‌یافته ارائه می‌کنیم.
- پس برای هر زبان منظم یک ماشین غیرقطعی وجود دارد و برای هر ماشین غیرقطعی یک گراف گذار تعمیم‌یافته و برای هر گراف گذار تعمیم‌یافته یک عبارت منظم.

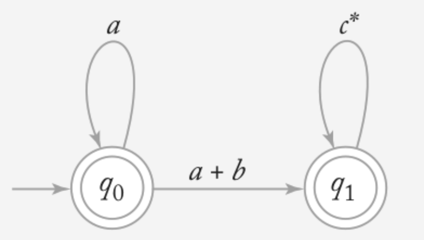
[1] generalized transition graph (GTG) or generalized nondeterministic finite automaton (GNFA)

عبارت‌ها و زبان‌های منظم

- گراف گذار تعمیم‌یافته یک گراف گذار است که برچسب یال‌های آن عبارت‌های منظم هستند.
- بنابراین برچسب هر گشت از حالت آغازی به یک حالت پایانی الحاق همهٔ عبارات منظم آن گشت است.
- تمام رشته‌هایی که توسط آن عبارات منظم به دست می‌آیند، عضوی از زبانی هستند که توسط گراف گذار شناسایی می‌شوند.
- مجموعهٔ همهٔ عبارات منظمی که در این گراف گذار پذیرفته می‌شوند، زبان متناظر با آن را می‌سازند.
- بنابراین هر زبانی که توسط یک گراف تعمیم یافته شناسایی می‌شود باید منظم باشد، زیرا جملات آن توسط عبارات منظم به دست می‌آیند.

- یک گراف گذار تعمیم‌یافته با دو حالت برای زبان منظم $L(a^* + a^*(a+b)c^*)$ بیابید.

- یک گراف گذار تعمیم‌یافته با دو حالت برای زبان منظم $L(a^* + a^*(a+b)c^*)$ بیابید.
- می‌توانیم به جای حلقه با برچسب a از برچسب a^* نیز استفاده کنیم.



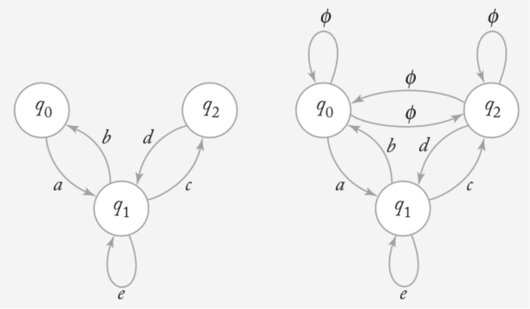
عبارت‌ها و زبان‌های منظم

- هر گراف گذار یک پذیرندهٔ متناهی غیرقطعی را می‌توانیم به یک گراف گذار تعمیم یافته (gtg) تبدیل کنیم.
- هر یالی که در nfa با نماد a نشان داده شده است را می‌توانیم در gtg با عبارت منظم a نشان دهیم.
- هر یالی که در nfa با نماد $a, b, \cdots$ نشان داده شده است را می‌توانیم در gtg با عبارت منظم $a + b + \cdots$ نشان دهیم.
- بنابراین برای هر زبان منظمی یک گراف تعمیم یافته وجود دارد.

- یک گراف گذار تعمیم‌یافتهٔ کامل گرافی است که همهٔ یال‌های آن حاضر باشند. بنابراین یال‌هایی که وجود ندارند را با عبارت منظم $\emptyset$ نشان می‌دهیم.
- بنابراین یک gtg کامل با n رأس تعداد n^2 یال دارد.

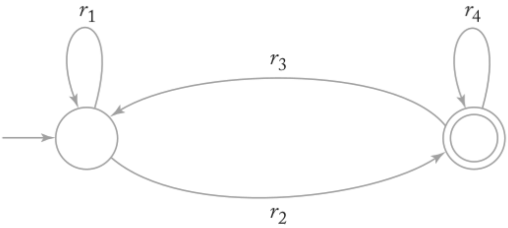
عبارت‌ها و زبان‌های منظم

- گراف گذار تعمیم یافتهٔ سمت چپ با معادل کامل آن در سمت راست نشان داده شده است.



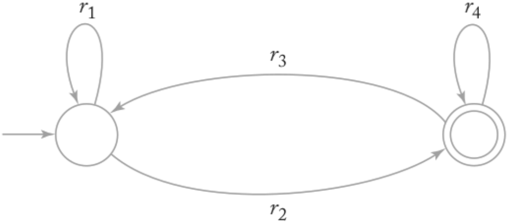
عبارت‌ها و زبان‌های منظم

- گراف گذار تعمیم یافتهٔ کامل زیر دو حالت دارد. عبارت منظمی را که این ماشین می‌پذیرد، تعیین کنید.



عبارت‌ها و زبان‌های منظم

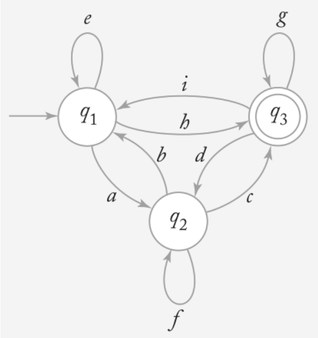
- گراف گذار تعمیم یافتهٔ کامل زیر دو حالت دارد.
- عبارت منظمی را که این ماشین می‌پذیرد، برابر است با : $r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*$



عبارت‌ها و زبان‌های منظم

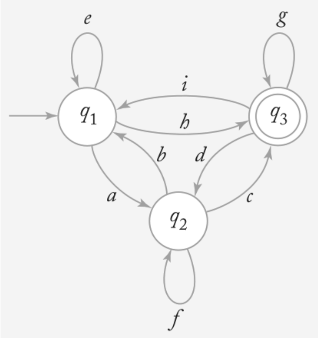
- وقتی یک gtg کامل بیشتر از دو حالت دارد، می‌توانیم در هر مرحله یک حالت آن را حذف کنیم تا در نهایت به یک gtg کامل با دو حالت برسیم.

- گراف گذار تعمیم‌یافته کامل زیر را در نظر بگیرید. چگونه می‌توانیم حالت q_2 را حذف کنیم؟



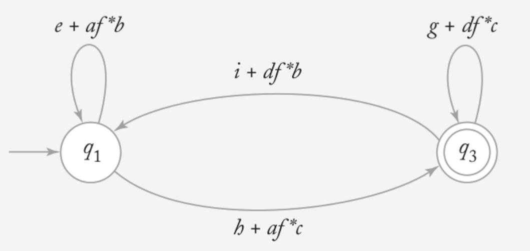
عبارت‌ها و زبان‌های منظم

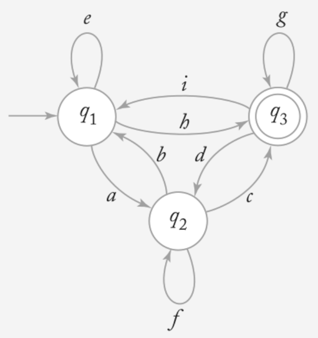
- گراف گذار تعمیم‌یافته کامل زیر را در نظر بگیرید. چگونه می‌توانیم حالت q_2 را حذف کنیم؟
- باید همهٔ مسیرهایی که برای گذار از q_2 عبور می‌کنند محاسبه کنیم تا بتوانیم q_2 را حذف کنیم.



عبارت‌ها و زبان‌های منظم

- گراف گذار تعمیم‌یافته کامل سمت چپ، کاهش یافتهٔ gtg کامل سمت راست است.





- پس به ازای هر gtg کامل می‌توانیم یک حالت در هر گام حذف کنیم تا به یک gtg کامل با دو حالت برسیم و عبارت منظم مربوط به آن گراف را محاسبه کنیم.

روند تبدیل ماشین متناهی غیرقطعی به عبارت منظم

۱. یک nfa را که از حالت‌های $q_0, q_1, \cdots, q_n$ تشکیل شده است و حالت پایانی آن متفاوت از حالت آغازی آن است، در نظر بگیرید.

۲. این nfa را به یک gtg کامل تبدیل کنید. فرض کنید برچسب یالی که از حالت q_i به q_j می‌رود برابر است با r_{ij}.

۳. اگر این gtg کامل فقط دو حالت داشته باشد و حالت q_i حالت آغازی و حالت q_j حالت پایانی آن باشد، آنگاه عبارت منظم آن برابر است با: $r = r_{ii}^* r_{ij} (r_{jj} + r_{ji} r_{ii}^* r_{ij})^*$

روند تبدیل ماشین متناهی غیرقطعی به عبارت منظم

۴. اگر gtg کامل سه حالت داشته باشد، حالت اولیه q_i، حالت پایانی q_j و حالت سوم q_k باشد، یال‌هایی با برچسب‌های $r_{pq} + r_{pk}r_{kk}^*r_{kq}$ به ازای $p = i, j$ و $q = i, j$ بسازید و حالت q_k و یال‌های قبلی را حذف کنید.

۵. اگر gtg کامل چهار حالت یا بیشتر دارد، حالت q_k را برای حذف کردن در نظر بگیرید. قوانین مرحله ۴ را برای هر زوج از حالت‌های $(q_i, q_j), i \neq k, j \neq k$ در نظر بگیرید. در هر مرحله برای ساده‌سازی عبارات می‌توانید قوانین زیر را به کار ببرید: $r + \emptyset = r$، $r\emptyset = \emptyset$ و $\emptyset^* = \lambda$.

۶. گام‌های ۳ تا ۵ را تکرار کنید تا عبارت منظم متناظر با گراف گذار نظر به دست بیاید.

عبارت‌ها و زبان‌های منظم

- عبارت منظمی برای این زبان بیابید:
$L = \{w \in \{a, b\}^* :$ تعداد نمادهای a زوج و تعداد نمادهای b فرد است $\}$

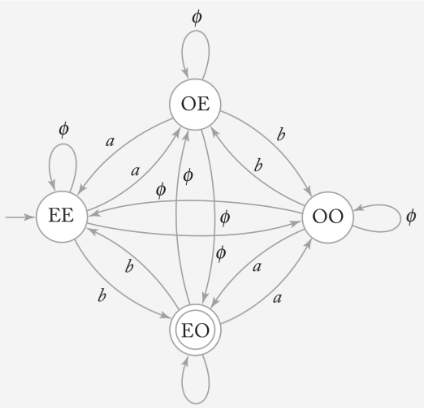
- عبارت منظمی برای این زبان بیابید:
$L = \{w \in \{a, b\}^* :$ تعداد نمادهای a زوج و تعداد نمادهای b فرد است $\}$

- ابتدا یک ماشین غیرقطعی با چهار حالت می‌سازیم که طوری که حالت EE حالتی است که در آن تعدادی زوج از a و b خوانده شده است، حالت OE حالتی است که در آن تعدادی فرد از a و تعدادی زوج از b خوانده شده، در حالت EO تعدادی زوج از a و تعدادی فرد از b خوانده شده، و در حالت OO تعدادی فرد از a و b خوانده شده است.

عبارت‌ها و زبان‌های منظم

- عبارت منظمی برای این زبان بیابید:
$L = \{w \in \{a, b\}^*$: تعداد نمادهای a زوج و تعداد نمادهای b فرد است $\}$

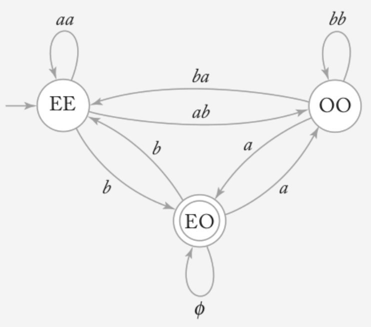
- گراف گذار کامل زیر را طراحی می‌کنیم.

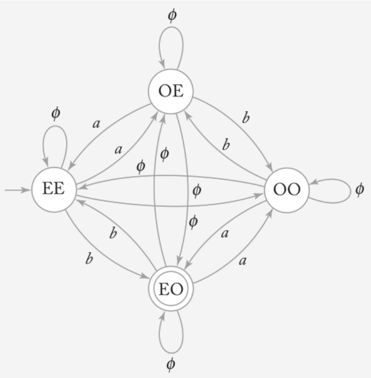


عبارت‌ها و زبان‌های منظم

- عبارت منظمی برای این زبان بیابید:
$L = \{w \in \{a, b\}^*$: تعداد نمادهای a زوج و تعداد نمادهای b فرد است $\}$

- یکی از حالات gtg کامل را حذف می‌کنیم.

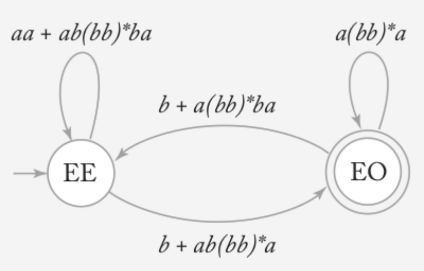


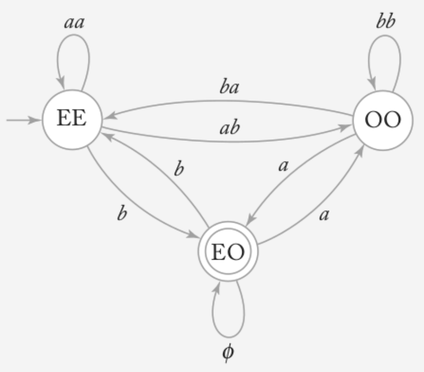


عبارت‌ها و زبان‌های منظم

- عبارت منظمی برای این زبان بیابید:
$L = \{w \in \{a, b\}^* :$ تعداد نمادهای a زوج و تعداد نمادهای b فرد است $\}$

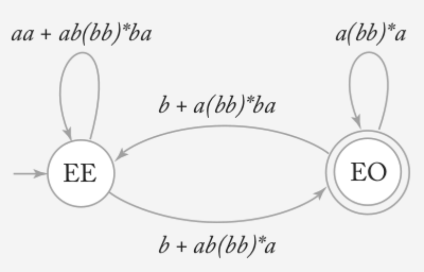
- یکی دیگر از حالات gtg کامل را حذف می‌کنیم.





عبارت‌ها و زبان‌های منظم

- عبارت منظمی برای این زبان بیابید:
$L = \{w \in \{a, b\}^*$: تعداد نمادهای a زوج و تعداد نمادهای b فرد است $\}$



عبارت‌ها و زبان‌های منظم

- استفاده از روش تبدیل nfa به gtg کامل به طوری دستی بسیار خسته کننده است و عبارتی که در پایان به دست می‌آید بسیار پیچیده است و در عمل نمی‌توان از آن استفاده کرد.
- تنها دلیل ارائه این روش، ارائه روشی رسمی برای اثبات قضایا است.

عبارت‌ها و زبان‌های منظم

- قضیه: فرض کنید L یک زبان منظم باشد. یک عبارت منظم r وجود دارد به طوری که $L = L(r)$
- اثبات: اگر L یک زبان منظم باشد، یک nfa برای آن وجود دارد. با استفاده از روند تبدیل ماشین متناهی غیرقطعی به عبارت منظم، یک عبارت منظم برای آن می‌سازیم که همان عبارت منظم مورد نظر است.

گرامرهای منظم

- زبان‌های منظم را علاوه بر نمایش با استفاده از مجموعه‌ها و عبارات منظم، می‌توان با گرامرهای منظم نیز نمایش داد.
- یک گرامر $G = (V, T, S, P)$ راست‌خطی [1] گفته می‌شود، اگر همهٔ قوانین تولید آن به صورت $A \to xB$ یا $A \to x$ باشد، به طوری که $A, B \in V$ و $x \in T^*$
- یک گرامر $G = (V, T, S, P)$ چپ‌خطی [2] گفته می‌شود، اگر همهٔ قوانین تولید آن به صورت $A \to Bx$ یا $A \to x$ باشد.
- یک گرامر منظم [3] گرامری است که راست‌خطی و یا چپ‌خطی باشد.

[1] right-linear
[2] left-linear
[3] regular grammar

- گرامر $G_1 = (\{S\}, \{a, b\}, S, P_1)$ با قوانین P_1 به صورت $S \rightarrow abS|a$ یک گرامر راست‌خطی است.
- عبارت منظمی برای زبان متناظر با این گرامر پیدا کنید.

گرامرهای منظم

- گرامر $G_1 = (\{S\}, \{a, b\}, S, P_1)$ با قوانین P_1 به صورت $S \to abS|a$ یک گرامر راست‌خطی است.
- عبارت منظمی برای زبان متناظر با این گرامر پیدا کنید.
- $r = (ab)^*a$

- گرامر $G_2 = (\{S, S_1, S_2\}, \{a, b\}, S, P_2)$ با قوانین P_2 به صورت $S \rightarrow S_1ab$، $S_1 \rightarrow S_1ab|S_2$، $S_2 \rightarrow a$ یک گرامر چپ‌خطی است.
- عبارت منظمی برای زبان متناظر با این گرامر پیدا کنید.

گرامرهای منظم

- گرامر $G_2 = (\{S, S_1, S_2\}, \{a, b\}, S, P_2)$ با قوانین P_2 به صورت $S \rightarrow S_1ab$، $S_1 \rightarrow S_1ab|S_2$، $S_2 \rightarrow a$ یک گرامر چپ‌خطی است.
- عبارت منظمی برای زبان متناظر با این گرامر پیدا کنید.
- $r = aab(ab)^*$

گرامرهای منظم

- گرامر $G = (\{S, A, B\}, \{a, b\}, S, P)$ با قوانین $S \rightarrow A$، $A \rightarrow aB|\lambda$، و $B \rightarrow Aa$ یک گرامر منظم نیست.
- گرچه برخی از قوانین راست‌خطی و برخی چپ‌خطی هستند، ولی گرامر نه چپ‌خطی است و نه راست‌خطی.
- این گرامر نمونه‌ای از یک گرامر خطی [1] است.
- گرامر خطی حداکثر یک متغیر در سمت راست هر قانون دارد و مکان آن متغیر بدون اهمیت است.
- هر گرامر منظم یک گرامر خطی نیز هست، ولی هر گرامر خطی منظم نیست.
- نشان می‌دهیم که گرامرهای منظم روش دیگری برای بیان زبان‌های منظم هستند.

[1] linear grammar

گرامرهای منظم

- اکنون نشان می‌دهیم که زبان‌های تولید شده توسط گرامرهای منظم، منظم هستند.
- ابتدا نشان می‌دهیم یک گرامر راست‌خطی، زبانی منظم تولید می‌کند.
- برای این کار، یک ان‌اف‌ای طراحی می‌کنیم که اشتقاق‌های یک گرامر راست‌خطی را شبیه‌سازی کند.
- در یک گرامر راست‌خطی، اشتقاق‌ها بدین صورت هستند : $ab \cdots cD \Rightarrow ab \cdots cdE$ که توسط قانون $D \to dE$ به وجود آمده است.
- ماشین غیرقطعی می‌تواند این گام را شبیه‌سازی کند، به طوری که در حالت D با خواندن نماد d به حالت E می‌رود.
- پس به طور کلی، متغیر گرامر را معادل یک حالت در نظر می‌گیریم و نمادهای پایانی (پیشوند صورت جمله‌ای) را نماد خوانده شده از ورودی بر روی یال در نظر می‌گیریم.

- قضیه: فرض کنید $G = (V, T, S, P)$ یک گرامر راست‌خطی باشد. آنگاه $L(G)$ یک زبان منظم است.

- اثبات: فرض می‌کنیم $V = \{V_۰, V_۱, \cdots\}$ به طوری که $S = V_۰$ باشد و قوانین تولید به صورت $V_۰ \rightarrow v_۱ V_i, V_i \rightarrow v_۲ V_j, \cdots V_n \rightarrow v_l, \cdots$ داشته باشیم.

- اگر w یک جمله در زبان $L(G)$ باشد، در یک اشتقاق خواهیم داشت:

$$V_۰ \Rightarrow v_۱ V_i \Rightarrow v_۱ v_۲ V_j \overset{*}{\Rightarrow} v_۱ v_۲ \cdots v_k V_n \Rightarrow v_۱ v_۲ \cdots v_k v_l = w$$

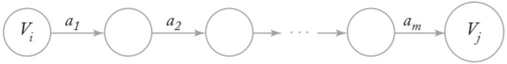
- ماشینی طراحی می‌کنیم که در ابتدا در حالت $V_۰$ قرار دارد. به ازای هر متغیر V_i یک حالت غیرپایانی با نام V_i در ماشین در نظر می‌گیریم.

گرامرهای منظم

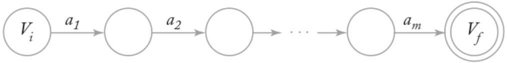
- به ازای هر قانون تولید $V_i \rightarrow a_1a_2\cdots a_mV_j$ ماشین گذارهایی برای اتصال V_i به V_j دارد. تابع گذار تعمیم‌یافته δ^* به ازای این قانون اینگونه تعریف می‌شود: $\delta^*(V_i, a_1a_2\cdots a_m) = V_j$

- برای هر قانون تولید $V_i \rightarrow a_1a_2\cdots a_m$ گذارهای ماشین به صورتی است که $\delta^*(V_i, a_1a_2\cdots a_m) = V_f$ به طوری که V_f یک حالت پایانی است.

- حالت‌های میانی را نیز برای شبیه‌سازی تابع گذار تعمیم‌یافته به صورت زیر می‌سازیم.



$V_i \rightarrow a_1a_2\ldots a_mV_j$



$V_i \rightarrow a_1a_2\ldots a_m$

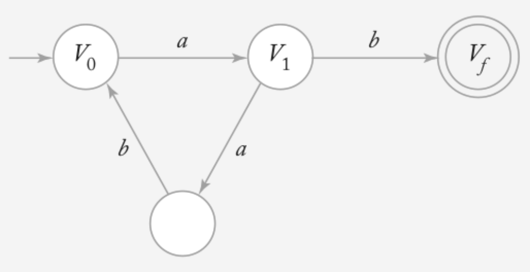
- پس فرض کردیم $w \in L(G)$ و یک ان‌اف‌ای معادل گرامر G طراحی کردیم به طوری که مسیری از $V_۰$ به V_i با برچسب $v_۱$ ، مسیری از V_i به V_j با برچسب $v_۲$ و الی آخر وجود دارد. پس $V_f \in \delta^*(V_۰, w)$ و بنابراین w توسط ماشین طراحی شده پذیرفته می‌شود. پس هر رشتهٔ تولید شده توسط این گرامر توسط یک ماشین متناهی غیرقطعی پذیرفته می‌شود.

- از طرف دیگر، فرض کنید رشتهٔ w توسط ماشین طراحی شده پذیرفته می‌شود. برای پذیرفتن w باید مسیری از حالت‌ها به صورت $V_۰, V_i, \cdots, V_f$ وجود داشته باشد که برچسب آن به صورت $v_۱, v_۲, \cdots$ باشد. بنابراین w باید بدین صورت باشد: $w = v_۱ v_۲ \cdots v_k v_l$ و بنابراین اشتقاق $V_۰ \Rightarrow v_۱ V_i \Rightarrow v_۱ v_۲ V_j \overset{*}{\Rightarrow} v_۱ v_۲ \cdots v_k V_k \Rightarrow v_۱ v_۲ \cdots v_k v_l$ امکان پذیر است. بنابراین w در $L(G)$ است. پس هر رشتهٔ پذیرفته شده توسط این ماشین غیرقطعی، توسط گرامر G نیز تولید می‌شود.

- پس برای $L(G)$ یک ان‌اف‌ای معادل آن طراحی کردیم و بنابراین $L(G)$ منظم است.

- یک ماشین متناهی طراحی کنید که زبان تولید شده توسط گرامر G با قوانین $V_0 \rightarrow aV_1 \quad , \quad V_1 \rightarrow abV_0|b$ را بپذیرد.

- ماشین متناهی زیر زبان تولید شده توسط گرامر G با قوانین $V_0 \rightarrow aV_1 \quad , \quad V_1 \rightarrow abV_0|b$ را می‌پذیرد.
- این ماشین زبان منظم $L((aab)^*ab)$ را می‌پذیرد.



گرامرهای منظم

- برای اینکه نشان دهیم هر زبان منظم می‌تواند توسط یک گرامر راست‌خطی تولید شود، از ماشین متناهی زبان منظم شروع می‌کنیم و نشان می‌دهیم که گذار از حالت‌های ماشین متناظر با اشتقاق توسط قوانین تولید یک گرامر راست‌خطی است.
- حالت‌های ماشین همان متغیرها در قوانین تولید هستند و برچسب روی یال‌ها همان پایانه‌ها در قوانین.
- می‌توانیم همانند قبل اثبات رسمی نیز ارائه کنیم.

گرامرهای منظم

- یک گرامر راست‌خطی برای زبان منظم $L(aab^*a)$ بسازید.

گرامرهای منظم

- یک گرامر راست‌خطی برای زبان منظم $L(aab^*a)$ بسازید.

$\delta(q_0, a) = \{q_1\}$	$q_0 \longrightarrow aq_1$
$\delta(q_1, a) = \{q_2\}$	$q_1 \longrightarrow aq_2$
$\delta(q_2, b) = \{q_2\}$	$q_2 \longrightarrow bq_2$
$\delta(q_2, a) = \{q_f\}$	$q_2 \longrightarrow aq_f$
$q_f \in F$	$q_f \longrightarrow \lambda$

گرامرهای منظم

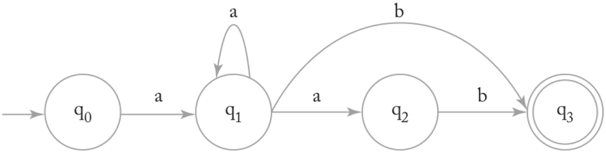
- زبان L منظم است اگر و فقط اگر گرامر منظم G برای آن وجود داشته باشد به طوری که $\mathrm{L} = \mathrm{L(G)}$.
- در این بخش الگوریتم‌هایی برای تبدیل عبارت‌های منظم و گرامرهای منظم به ماشین‌های متناهی و بالعکس ارائه کردیم.
- پس زبان‌های منظم را می‌توانیم توسط ماشین‌های متناهی قطعی و غیرقطعی، عبارت‌های منظم و گرامرهای منظم وصف کنیم.

مثال

- یک ماشین متناهی غیرقطعی برای زبان $L(aa^*(ab+b))$ طراحی کنید.

مثال

- یک ماشین متناهی غیرقطعی برای زبان $L(aa^*(ab + b))$ طراحی کنید.



مثال

- یک عبارت منظم برای زبان $L = \{a^n b^m : n \geq ۳, m \bmod ۲ = ۱\}$ پیدا کنید.

مثال

- یک عبارت منظم برای زبان $L = \{a^n b^m : n \geq ۳, m \bmod ۲ = ۱\}$ پیدا کنید.
- $aaaa^*(bb)^*b$

مثال

- یک عبارت منظم برای زبان $L = \{a^n b^m : n \geq ۳, m \leq ۴\}$ پیدا کنید.

مثال

- یک عبارت منظم برای زبان $L = \{a^n b^m : n \geq ۳, m \leq ۴\}$ پیدا کنید.
- $aaaa^*(\lambda + b + bb + bbb + bbbb)$

مثال

- یک عبارت منظم برای زبان $L = \{vwv : v, w \in \{a, b\}^*, |v| = ۲\}$ پیدا کنید.

مثال

- یک عبارت منظم برای زبان $L = \{vwv : v, w \in \{a, b\}^*, |v| = ۲\}$ پیدا کنید.
- $aa(a + b)^*aa + ab(a + b)^*ab + ba(a + b)^*ba + bb(a + b)^*bb$

مثال

- یک عبارت منظم برای زبان $L = \{uwv : u, v, w \in \{a, b\}^*, |u| = |v| = ۲\}$ پیدا کنید.

مثال

- یک عبارت منظم برای زبان $L = \{uwv : u, v, w \in \{a, b\}^*, |u| = |v| = ۲\}$ پیدا کنید.
- $(aa + ab + ba + bb)(a + b)^*(aa + ab + ba + bb)$
- $(a + b)(a + b)(a + b)^*(a + b)(a + b)$

مثال

- به ازای $\Sigma = \{a, b, c\}$، یک عبارت منظم برای زبانی بیابید که جملات آن فقط دو a دارند.

مثال

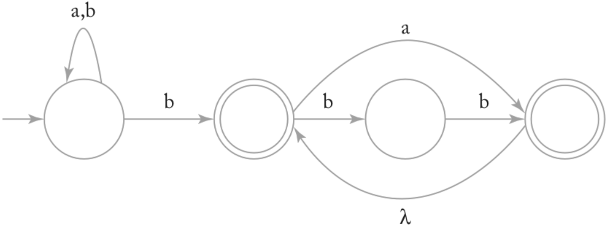
- به ازای $\Sigma = \{a, b, c\}$، یک عبارت منظم برای زبانی بیابید که جملات آن فقط دو a دارند.
- $(b + c)^*a(b + c)^*a(b + c)^*$

مثال

- یک ماشین متناهی غیرقطعی برای زبان $L((a + b)^*b(a + bb)^*)$ طراحی کنید.

مثال

- یک ماشین متناهی غیرقطعی برای زبان $L((a+b)^*b(a+bb)^*)$ طراحی کنید.

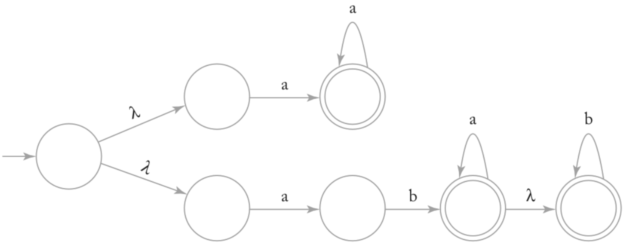


مثال

- یک ماشین متناهی برای زبان $L(aa^* + aba^*b^*)$ طراحی کنید.

مثال

- یک ماشین متناهی برای زبان $L(aa^* + aba^*b^*)$ طراحی کنید.

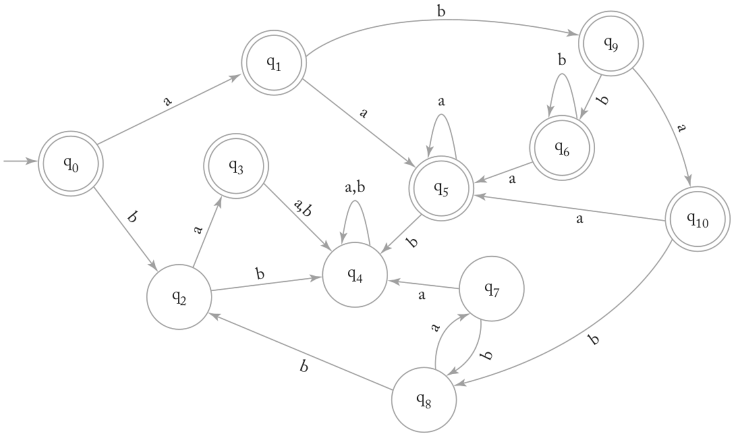


مثال

- یک ماشین متناهی قطعی برای زبان $L = L(ab^*a^*) \cup L((ab)^*ba)$

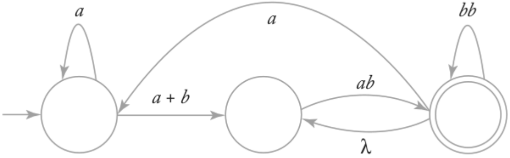
مثال

- یک ماشین متناهی قطعی برای زبان $L = L(ab^*a^*) \cup L((ab)^*ba)$



مثال

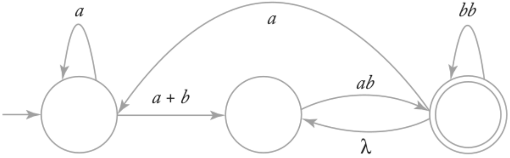
- گراف گذار تعمیم یافته زیر را به گرافی با دو حالت تبدیل کنید و عبارت منظم معادل آن را بنویسید.

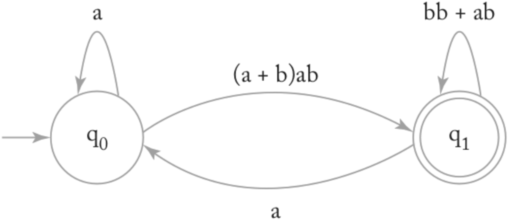


مثال

- گراف گذار تعمیم یافته زیر را به گرافی با دو حالت تبدیل کنید و عبارت منظم معادل آن را بنویسید.

- $r = a^*(a+b)ab(bb+ab+aa^*(a+b)ab)^*$





مثال

- به ازای $\Sigma = \{a, b\}$، عبارت منظمی برای زبانی بیابید طول جملات آن زوج باشد.

مثال

- به ازای $\Sigma = \{a, b\}$، عبارت منظمی برای زبانی بیابید طول جملات آن زوج باشد.
- $(aa + bb + ab + ba)^*$
- $((a + b)(a + b))^*$

مثال

- به ازای $\Sigma = \{a, b\}$، عبارت منظمی برای زبانی بیابید طول جملات آن بزرگتر یا مساوی ۳ باشد.

مثال

- به ازای $\Sigma = \{a, b\}$، عبارت منظمی برای زبانی بیابید طول جملات آن بزرگتر یا مساوی ۳ باشد.
- $(a + b)(a + b)(a + b)(a + b)^*$

مثال

- به ازای $\Sigma = \{a, b\}$، عبارت منظمی برای زبانی بیابید تعداد نمادهای a در آن فرد باشد.

مثال

- به ازای $\Sigma = \{a, b\}$، عبارت منظمی برای زبانی بیابید تعداد نمادهای a در آن فرد باشد.
- $b^*ab^*(ab^*ab^*)^*$
- $b^*(ab^*ab^*)^*ab^*$
- $b^*a(b + ab^*a)^*$
- $(b + ab^*a)^*ab^*$

مثال

- به ازای $\Sigma = \{0, 1\}$، عبارت منظمی برای همهٔ اعداد دودویی مساوی یا بیشتر از ۳۲ (100000) بیابید.

مثال

- به ازای $\Sigma = \{۰, ۱\}$، عبارت منظمی برای همهٔ اعداد دودویی مساوی یا بیشتر از ۳۲ (۱۰۰۰۰۰) بیابید.
- $۱(۰+۱)^۵(۰+۱)^*$

مثال

- به ازای $\Sigma = \{a, b\}$، عبارت منظمی برای زبانی بیابید که ابتدا و انتهای همهٔ جملات آن یکسان باشد.

مثال

- به ازای $\Sigma = \{a, b\}$، عبارت منظمی برای زبانی بیابید که ابتدا و انتهای همهٔ جملات آن یکسان باشد.
- $a(a+b)^*a + b(a+b)^*b + a + b$

مثال

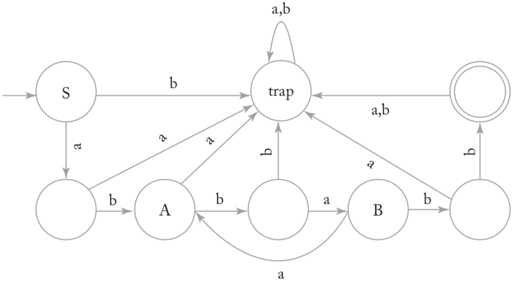
- یک ماشین متناهی قطعی برای گرامری با قوانین زیر بیابید. عبارت منظم و گرامر چپ خطی متناظر آن چیست؟ $S \rightarrow abA \quad , \quad A \rightarrow baB \quad , \quad B \rightarrow aA|bb$

مثال

- یک ماشین متناهی قطعی برای گرامری با قوانین زیر بیابید. عبارت منظم و گرامر چپ خطی متناظر آن چیست؟ $S \rightarrow abA \ , \ A \rightarrow baB \ , \ B \rightarrow aA|bb$

- $L(G) = L(abba(aba)^*bb)$

- $S \rightarrow Abb \ , \ A \rightarrow Aaba|B \ , \ B \rightarrow abba$



مثال

- یک گرامر راست‌خطی و یک گرامر چپ خطی برای زبان $L((aaab^*ab)^*)$ بیابید.

مثال

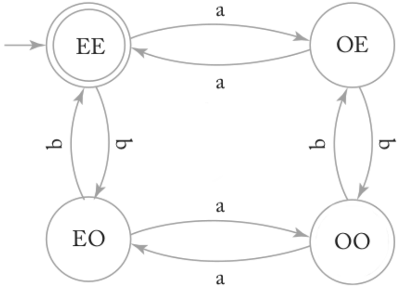
- یک گرامر راست‌خطی و یک گرامر چپ خطی برای زبان $L((aaab^*ab)^*)$ بیابید.
- $S \rightarrow aaaA|\lambda \;\; , \;\; A \rightarrow bA|B \;\; , \;\; B \rightarrow ab|abS$
- $S \rightarrow Aab|\lambda \;\; , \;\; A \rightarrow Ab|B \;\; , \;\; B \rightarrow Saaa$

مثال

- یک گرامر منظم برای زبان $L = \{w \in \{a, b\}^* : n_a(w) \bmod ۲ = ۰, n_b(w) \bmod ۲ = ۰\}$ بیابید.

مثال

- یک گرامر منظم برای زبان $L = \{w \in \{a, b\}^* : n_a(w) \bmod ۲ = ۰, n_b(w) \bmod ۲ = ۰\}$ بیابید.
- از ماشین متناهی این زبان استفاده می‌کنیم.
- $EE \rightarrow aOE|bEO|\lambda \ , \ OE \rightarrow aEE|bOO \ , \ EO \rightarrow aOO|bEE \ , \ OO \rightarrow aEO|bOE$



ویژگی‌های بستاری زبان‌های منظم

- اگر مجموعۀ همۀ زبان‌های منظم را در نظر بگیریم، آنگاه می‌خواهیم بررسی کنیم آیا این مجموعه در برابر عملگرهای مختلف مانند الحاق و اجتماع و اشتراک بسته است یا خیر.
- اگر یک مجموعه بر روی یک عملگر بسته باشد، نتیجه اعمال آن عملگر بر روی اعضای آن مجموعه عضوی از همان مجموعه است.
- پس اگر مجموعه‌ای از زبان‌های منظم داشته باشیم و زبان‌های منظم در برابر یک عملگر بسته باشند، آنگاه اعمال آن عملگر بر روی آن زبان‌ها زبانی منظم ایجاد می‌کند.
- در اینجا ویژگی‌های بستاری [1] زبان‌های منظم را بررسی می‌کنیم.

[1] closure properties

ویژگی‌های بستاری زبان‌های منظم

- اگر دو زبان L_1 و L_2 منظم باشند، زبان‌های $L_1 \cup L_2$، $L_1 \cap L_2$، L_1L_2، $\overline{L_1}$، L_1^* نیز منظم هستند.
- می‌گوییم خانوادهٔ زبان‌های منظم بر روی عملگرهای اجتماع، اشتراک، الحاق، متمم، و بستار–ستاره بسته است.

ویژگی‌های بستاری زبان‌های منظم

- اگر دو زبان L_1 و L_2 منظم باشند، زبان‌های $L_1 \cup L_2$، L_1L_2، و L_1^* نیز منظم هستند.
- اثبات: اگر دو زبان L_1 و L_2 منظم باشند، دو عبارت منظم r_1 و r_2 برای آنها وجود دارد که آن دو زبان را وصف می‌کند.
- پیشتر نشان دادیم اگر r_1 و r_2 منظم باشند، $r_1 + r_2$، r_1r_2، r_1^* نیز منظم هستند که زبان‌های $L_1 \cup L_2$، L_1L_2، L_1^* را وصف می‌کنند.

ویژگی‌های بستاری زبان‌های منظم

- اگر زبان L_1 منظم باشند، زبان $\overline{L_1}$ نیز منظم است.

- اثبات: ماشین متناهی قطعی متناظر با L_1 را در نظر می‌گیریم. متمم زبان در یک ماشین متناهی قطعی، ماشینی است که در آن جای حالت‌های پایانی و غیر پایانی عوض شده باشد. پس ماشینی برای متمم آن وجود دارد و بنابراین متمم یک زبان منظم نیز منظم است.

ویژگی‌های بستاری زبان‌های منظم

- اگر دو زبان L_1 و L_2 منظم باشند، زبان $L_1 \cap L_2$، نیز منظم است.

- اثبات: می‌توانیم از قوانین دمورگان استفاده کنیم. $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. از آنجایی که زبان‌های منظم بر روی متمم و اجتماع بسته هستند، بر روی اشتراک نیز بسته‌اند.

ویژگی‌های بستاری زبان‌های منظم

- برای اثبات بسته بودن زبان‌های منظم بر روی اشتراک همچنین می‌توانیم ماشینی بسازیم که هر یک از حالات آن متناظر با یک حالت از ماشین اول و یک حالت از ماشین دوم است.

- به عبارت دیگر فرض کنید $L_1 = L(M_1)$ و $L_2 = L(M_2)$ به طوری که $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ و $M_2 = (P, \Sigma, \delta_2, p_0, F_2)$. ماشینی به صورت $\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\delta}, (p_0, q_0), \widehat{F})$ که حالت‌های آن به صورت $\widehat{Q} = Q \times P$ شامل همهٔ حالت‌های (q_i, p_j) باشد به طوری که $\widehat{\delta}((q_i, p_j), a) = (q_k, q_l)$ وقتی $\delta_1(q_i, a) = q_k$ و $\delta_2(p_j, a) = q_l$. حالت‌های پایانی $\widehat{F}$ حالت‌های (q_i, p_i) هستند به طوری که $q_i \in F_1$ و $p_j \in F_2$. سپس نشان می‌دهیم هر رشته $w \in L_1 \cap L_2$ توسط ماشین $\widehat{M}$ پذیرفته می‌شود و بنابراین اشتراک دو زبان منظم است.

ویژگی‌های بستاری زبان‌های منظم

- نشان دهید اگر L_1 و L_2 دو زبان منظم باشند، آنگاه $L_1 - L_2$ نیز منظم است.

ویژگی‌های بستاری زبان‌های منظم

- نشان دهید اگر L_1 و L_2 دو زبان منظم باشند، آنگاه $L_1 - L_2$ نیز منظم است.
- می‌نویسیم $L_1 - L_2 = L_1 \cap \overline{L_2}$
- از آنجایی که متمم یک زبان منظم و اشتراک دو زبان منظم، یک زبان منظم است، پس تفاضل دو زبان منظم نیز یک زبان منظم است.

ویژگی‌های بستاری زبان‌های منظم

- خانوادهٔ زبان‌های منظم بر روی عملگر معکوس بسته است.
- برای اثبات می‌توانیم یک ان‌اف‌ای برای زبان L بسازیم. سپس حالت پایانی را تبدیل به حالت آغازی و حالت آغازی را تبدیل به حالت پایانی می‌کنیم و جهت یال‌های گذار را برعکس (از مقصد به مبدأ) می‌کنیم. این ماشین معکوس زبان L را می‌پذیرد.
- اگر ان‌اف‌ای بیشتر از یک حالت پایانی داشت، همهٔ حالت‌های پایانی را به یک حالت پایانی با گذار توسط نماد تهی متصل می‌کنیم.

محدودیت زبان‌های منظم

- زبان‌هایی وجود دارند که منظم نیستند و نمی‌توان برای آنها ماشین متناهی پیدا کرد.
- در اینجا برای اینکه نشان دهیم یک زبان منظم نیست از لم تزریق [1] (لم پمپاژ) استفاده می‌کنیم.

[1] pumping lemma

محدودیت زبان‌های منظم

- زبان‌های منظم توسط ماشین‌های متناهی شناسایی می‌شوند.
- ماشین‌های متناهی حافظهٔ محدود دارند، چرا که فقط می‌توانند به یاد بیاورند در چه حالتی هستند و حافظهٔ جانبی ندارند.
- به طور شهودی می‌توانیم حدس بزنیم که زبان‌هایی که برای شناسایی به ماشینی با حافظهٔ بیشتر نیاز دارند نمی‌توانند منظم باشند.
- این حدس را بیشتر بررسی می‌کنیم.

محدودیت زبان‌های منظم

- اصل لانه کبوتری : اگر n شیء (کبوتر) را در m جعبه (لانه کبوتر) قرار دهیم، و $n > m$ باشد، آنگاه حداقل یکی از جعبه‌ها باید بیشتر از یک شیء را شامل شود.

محدودیت زبان‌های منظم

- آیا زبان $L = \{a^n b^n : n \geq ۰\}$ منظم است؟

محدودیت زبان‌های منظم

- آیا زبان $L = \{a^n b^n : n \geq ۰\}$ منظم است؟

- با برهان خلف و با استفاده از اصل لانه کبوتری نشان می‌دهیم که این زبان منظم نیست.

- فرض کنید L یک زبان منظم باشد. آنگاه باید یک ماشین متناهی قطعی به صورت $M = (Q, \{a, b\}, \delta, q_۰, F)$ برای آن وجود داشته باشد.

- حال $\delta^*(q_۰, a^i)$ را به ازای $i = ۱, ۲, ۳, \cdots$ در نظر بگیرید. از آنجایی که تعداد نامحدودی a می‌تواند وجود داشته باشد ولی تعداد حالت‌های ماشین محدود است، طبق اصل لانه کبوتری حالتی به نام q وجود دارد به طوری که $\delta^*(q_۰, a^n) = q$ و $\delta^*(q_۰, a^m) = q$ به ازای $n \neq m$.

- از آنجایی که ماشین رشتهٔ $a^n b^n$ را می‌پذیرد، باید داشته باشیم $\delta^*(q, b^n) = q_f \in F$

- پس داریم: $\delta^*(q_۰, a^m b^n) = \delta^*(\delta^*(q_۰, a^m), b^n) = \delta^*(q, b^n) = q_f$

- این نتیجه با فرض اولیه تناقض دارد چراکه گفتیم ماشین فقط رشته‌های $a^n b^n$ را می‌پذیرد، پس L نمی‌تواند منظم باشد.

محدودیت زبان‌های منظم

- برای پذیرفتن رشتهٔ $a^n b^n$ یک ماشین متناهی باید قادر باشد بین پیشوندهای a^n و a^m تفاوت قائل شود.
- اما از آنجایی که فقط تعداد محدودی حالت داخلی وجود دارد، به ازای برخی n و m ها، ماشین متناهی نمی‌تواند تفاوتی قائل شود، بنابراین این زبان منظم نیست.

لم تزریق

- برای اثبات اینکه یک زبان منظم نیست، از لم تزریق استفاده می‌کنیم.
- در لم تزریق به طور شهودی از این خاصیت استفاده می‌کنیم: برای یک زبان نامحدود، در یک گراف گذار با n حالت، در هر گشتی در گراف که طول آن بیشتر از n باشد، از برخی از حالت‌ها چندین بار عبور می‌کنیم. بنابراین گراف گذار باید دارای دور باشد.
- زبان‌های محدود منظم هستند چرا که می‌توان یک ماشین غیرقطعی برای پذیرفتن همهٔ جملات آنها طراحی کرد.

لم تزریق

- قضیه: فرض کنید L یک زبان منظم نامحدود باشد. آنگاه، یک عدد صحیح مثبت m وجود دارد، به طوری که هر رشتهٔ $w \in L$ با طول $|w| \geq m$ می‌تواند به صورت $w = xyz$ به سه قسمت تقسیم شود به طوری که $|xy| \leq m$، $|y| \geq ۱$ و همچنین $w_i = xy^iz \in L$، به ازای همهٔ مقادیر $i = ۰, ۱, ۲, \cdots$.
- به عبارت دیگر هر رشته‌ای در زبان منظم L که به اندازهٔ کافی طولانی باشد، می‌تواند به سه قسمت تقسیم شود به طوری که از تکرار قسمت میانی، یک رشتهٔ دیگر در زبان L به دست می‌آید. می‌گوییم قسمت میانی پمپاژ (تزریق) می‌شود.

لم تزریق

- اثبات لم : اگر L یک زبان منظم باشد، یک دی‌اف‌ای برای آن وجود دارد که آن را می‌پذیرد. فرض کنید حالت‌های این ماشین $q_0, q_1, \cdots, q_n$ باشد.
- نشان می‌دهیم که اگر m را برابر با $n+1$ بگیریم، می‌توانیم رشتهٔ xyz را با شرایط گفته شده پیدا کنیم.
- حال رشتهٔ $w \in L$ را در نظر بگیرید به طوری که $|w| \geq m = n + 1$. از آنجایی که زبان L نامحدود است، این کار را می‌توانیم همیشه انجام دهیم.

لم تزریق

- دنباله‌ای از حالت‌ها را در نظر بگیرید که ماشین برای پذیرفتن w باید از آنها بگذرد : $q_0, q_i, q_j, \cdots, q_f$

- از آنجایی که این دنباله $|w| + 1$ حالت را در بر می‌گیرد و ماشین تنها $n + 1$ حالت دارد، بنابراین حداقل یکی از حالت‌ها باید تکرار شده باشد. پس دنباله باید بدین شکل باشد: $q_0, q_i, q_j, \cdots, q_r, \cdots, q_r, \cdots q_f$

- بنابراین باید زیررشته‌هایی وجود داشته باشند به طوری که $\delta^*(q_0, x) = q_r$، $\delta^*(q_r, y) = q_r$، و $\delta^*(q_r, z) = q_f$ به طوری که $|xy| \leq n + 1 = m$ و $|y| \geq 1$.

- همچنین داریم $\delta^*(q_0, xz) = q_f$، $\delta^*(q_0, xy^2z) = q_f$، $\delta^*(q_0, xy^3z) = q_f$ و الی‌آخر.

- با استفاده از لم تزریق نشان دهید که زبان $L = \{a^n b^n : n \geq 0\}$ منظم نیست.

لم تزریق

- با استفاده از لم تزریق نشان دهید که زبان $L = \{a^n b^n : n \geq 0\}$ منظم نیست.

- فرض کنید زبان L منظم باشد. پس لم تزریق باید برای آن برقرار باشد. مقدار m هرچه باشد می‌توانیم به ازای آن $w = a^m b^m$ را در نظر بگیریم که طول آن بزرگتر از m است.

- پس در رشتهٔ $xyz = a^m b^m$ زیررشتهٔ y فقط از a تشکیل شده است. فرض کنید طول رشتهٔ y برابر باشد با k.

- پس داریم: $w_i = xy^i z = a^{m-k}(a^k)^i b^m$

- به ازای $i = 0$ به دست می‌آوریم $w_0 = xy^0 z = a^{m-k} b^m$ که این رشته عضو زبان L نیست. پس فرض اولیه نادرست است و L منظم نیست.

لم تزریق

- نشان دهید زبان $L = \{ww^R : w \in \Sigma^*\}$ منظم نیست.

لم تزریق

- نشان دهید زبان $L = \{ww^R : w \in \Sigma^*\}$ منظم نیست.

- فرض کنیم که زبان منظم است. پس باید در لم تزریق صدق کند و یک m وجود داشته باشد که به ازای هر جملهٔ w در L شرایط لم تزریق برقرار باشد.

- اگر m هر مقداری داشته باشد، می‌توانیم یکی از جملات زبان را به صورت $w = a^m b^m b^m a^m$ در نظر بگیریم.

- از آنجایی که $|xy| \leq m$ پس y تنها از نمادهای a تشکیل شده است. فرض کنیم طول y برابر با k باشد.

- برای رشتهٔ $w_i = xy^iz = a^{m-k}(a^k)^i b^m b^m a^m$ اگر در نظر بگیریم $i = ۰$ آنگاه رشته‌ای داریم به صورت $a^{m-k} b^m b^m a^k$ که در زبان L نیست.

- پس فرض اولیه نادرست است و L منظم نیست.

لم تزریق

- فرض کنید $\Sigma = \{a, b\}$. با استفاده از لم تزریق نشان دهید $L = \{w \in \Sigma^* : n_a(w) < n_b(w)\}$ منظم نیست.

لم تزریق

- فرض کنید $\Sigma = \{a, b\}$. با استفاده از لم تزریق نشان دهید $L = \{w \in \Sigma^* : n_a(w) < n_b(w)\}$ منظم نیست.
- به ازای m داده شده، جملهٔ $w = a^m b^{m+1}$ را در نظر می‌گیریم.
- در اینصورت $y = a^k$ به ازای $1 \leq k \leq m$
- بنابراین $w_i = xy^iz = a^{m-k}(a^k)^i b^{m+1}$
- اکنون به ازای $i = 2$ داریم $w_2 = a^{m+k}b^{m+1}$ که در L نیست و L منظم نیست.

- نشان دهید زبان $L = \{a^n : \text{مربع کامل است } n\}$ منظم نیست.

لم تزریق

- نشان دهید زبان $L = \{a^n :$ مربع کامل است $n\}$ منظم نیست.
- به ازای m داده شده، جملهٔ $w = a^{m^۲}$ را انتخاب می‌کنیم.
- اگر $w = xyz$ آنگاه $y = a^k$ به ازای $۱ \leq k \leq m$.
- در اینصورت داریم $w_۰ = a^{m^۲-k}$ اما با توجه به اینکه $m^۲ - k > (m-۱)^۲$ بنابراین $w_۰$ در زبان L نیست و زبان L منظم نیست.

لم تزریق

- نشان دهید زبان $L = \{a^n b^l : n \neq l\}$ منظم نیست.

لم تزریق

- نشان دهید زبان $L = \{a^n b^l : n \neq l\}$ منظم نیست.
- فرض می‌کنیم زبان L منظم باشد، پس لم تزریق باید برای آن برقرار باشد.
- به ازای هر m با در نظر گرفتن جملهٔ $a^{m+۱}b^m$ نمی‌توانیم اثبات کنیم که زبان منظم نیست. گرچه با در نظر گرفتن $y = a$ و انتخاب $i = ۰$ می‌توانیم جملهٔ $a^m b^m$ را تولید کنیم که در زبان L نیست، ولی اگر y به گونه‌ای دیگر (مثلا $y = a^۲$) در نظر گرفته شود نمی‌توانیم به مطلوب دست پیدا کنیم.
- بنابراین برای اثبات غیرمنظم بودن این زبان با استفاده از لم تزریق باید جملهٔ دیگری را به ازای m داده شده در نظر بگیریم.

لم تزریق

- نشان دهید زبان $L = \{a^n b^l : n \neq l\}$ منظم نیست.
- فرض می‌کنیم زبان L منظم باشد، پس لم تزریق باید برای آن برقرار باشد.
- به ازای هر m جملهٔ $w = a^{m!} b^{(m+۱)!}$ را در نظر بگیرید.
- از آنجایی که $y = a^k$ است، داریم $w = xyz = a^{m!-k} a^k b^{(m+۱)!}$ و $w_i = xy^iz = a^{m!-k}(a^k)^i b^{(m+۱)!}$ بنابراین: $w_i = a^{m!+k(i-۱)} b^{(m+۱)!}$
- حال برای k هر مقداری در نظر گرفته شود، همیشه می‌توانیم در نظر بگیریم: $i = ۱ + \frac{mm!}{k}$ که به دست می‌دهد: $w_i = a^{m!+mm!} b^{(m+۱)!} = a^{(m+۱)!} b^{(m+۱)!}$
- پس در هر صورت w_i در زبان L نیست و بنابراین طبق لم تزریق فرض اولیه مبنی بر منظم بودن زبان نادرست بوده است.

لم تزریق

- لم تزریق یک شرط لازم است و یک شرط کافی نیست، بدین معنی که اگر یک زبان منظم باشد، شرایط گفته شده در لم برقرار است ولی اگر شرایط گفته شده برقرار بود، زبان الزاما منظم نیست.

- بنابراین از لم تزریق برای اثبات منظم نبودن زبان استفاده می‌کنیم و نه اثبات منظم بودن.

- لم تزریق می‌گوید اگر زبانی منظم باشد یک m **وجود دارد** به طوری که به ازای **هر** رشتهٔ w با طول بیشتر از m رشته را **می‌توان** با شرایط گفته شده به سه قسمت x و y و z تقسیم کرد به طوری که اگر قسمت میانی یعنی y به ازای **هر** i تکرار شود (به هر مقداری پمپاژ شود) رشتهٔ به دست آمده در زبان مورد نظر قرار می‌گیرد.

- پس برای اثبات نامنظم بودن زبان از طریق برهان خلف، فرض می‌کنیم زبانی منظم باشد. پس باید طبق لم تزریق یک m **وجود داشته باشد**. **یکی** از رشته‌هایی که طول آن بیشتر از m است را در نظر می‌گیریم. نشان می‌دهیم که **نمی‌توان** آن رشته را به سه قسمت تقسیم کرد، به طوری که از پمپاژ قسمت میانی، رشته‌ای در آن زبان به دست آید. به عبارت دیگر **به هر نحوی** رشته را تقسیم کنیم، یک مقدار i (تعداد تکرار معین از قسمت میانی) **وجود دارد** که به ازای آن لم تزریق **برقرار نباشد**. پس فرض اولیه نادرست بوده و آن زبان نامنظم است.

مثال

- با استفاده از ویژگی‌های بستاری زبان‌های منظم بر روی عملگرها، نشان دهید زبان $L_1 = \{a^n b^l : n \neq l\}$ منظم نیست.

مثال

- با استفاده از ویژگی‌های بستاری زبان‌های منظم بر روی عملگرها، نشان دهید زبان $L_۱ = \{a^n b^l : n \neq l\}$ منظم نیست.
- فرض می‌کنیم زبان $L_۱$ یک زبان منظم است، آنگاه زبان $\overline{L_۱} = \Sigma^* - L_۱$ نیز طبق ویژگی‌های بستاری منظم است.
- از طرفی می‌دانیم زبان $L_۲ = \{a^n b^l : n, l \geq ۰\}$ یک زبان منظم است زیرا یک پذیرندهٔ متناهی قطعی برای آن وجود دارد.
- طبق ویژگی‌های بستاری $L_۳ = \overline{L_۱} \cap L_۲$ باید منظم باشد.
- از آنجایی که $L_۳ = \{a^n b^n : n \geq ۰\}$ است و قبلا ثابت کردیم زبان $L_۳$ منظم نیست، پس فرض اولیه نادرست بوده، و $L_۱$ منظم نیست.

زبان‌های مستقل از متن

مقدمه

- حال که با محدودیت‌های زبان‌های منظم آشنا شدیم، یک دستهٔ دیگر از زبان‌ها را بررسی می‌کنیم که زبان‌های مستقل از متن نامید می‌شوند و محدودیت‌های زبان‌های منظم را ندارند.
- زبان‌های منظم دسته‌ای محدود شده از زبان‌های مستقل از متن و بنابراین زیرمجموعهٔ این دسته از زبان‌ها به شمار می‌آیند.
- یکی از زبان‌های سادهٔ غیرمنظم زبان $L = \{a^n b^n : n \geq 0\}$ بود. در این قسمت این دسته از زبان‌ها را به همراه گرامر آنها بررسی می‌کنیم.

گرامرهای مستقل از متن

- برای قوانین تولید در گرامرهای منظم محدودیتی در نظر گرفتیم. سمت چپ قانون در گرامرهای منظم همیشه یک متغیر و سمت راست ترکیبی از متغیرها و نمادهای پایانی است به طوری که متغیر فقط در ابتدا و یا فقط در انتهای عبارت سمت راست قرار می‌گیرد.
- در گرامرهای مستقل از متن از این محدودیت می‌کاهیم.

گرامرهای مستقل از متن

- گرامر $G = (V, T, S, P)$ یک گرامر مستقل از متن [1] نامید می‌شود اگر همهٔ قوانین تولید P در آن به صورت $A \to x$ باشند به طوری که $A \in V$ یک متغیر و $x \in (V \cup T)^*$ ترکیبی از متغیرها و پایانه‌ها باشد.
- زبان L یک زبان مستقل از متن نامیده می‌شود اگر و فقط اگر یک گرامر مستقل از متن وجود داشته باشد به طوری که $L = L(G)$.
- زبان‌های مستقل از متن به این دلیل اینگونه نامیده می‌شوند که در هر مرحله از اشتقاق برای به دست آوردن یک جمله از زبان، هر قانونی که یکی از متغیرهای صورت جمله‌ای را در سمت چپ داشته باشد می‌تواند اعمال شود و بنابراین اعمال قانون در فرایند اشتقاق به وابسته به متن نیست.

[1] context-free grammar

گرامرهای مستقل از متن

- گرامر $G = (\{S\}, \{a, b\}, S, P)$ با قوانین تولید $S \to \lambda \quad , \quad S \to bSb \quad , \quad S \to aSa$ یک گرامر مستقل از متن است.
- یک مثال از اشتقاق در این گرامر بدین صورت است: $S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabbaa$
- بنابراین زبان $L = \{ww^R : w \in \{a, b\}^*\}$ که توسط این گرامر وصف می‌شود یک زبان مستقل از متن است.

گرامرهای مستقل از متن

- گرامر G با قوانین تولید $S \rightarrow abB \quad , \quad A \rightarrow aaBb \quad , \quad B \rightarrow bbAa \quad , \quad A \rightarrow \lambda$ یک گرامر مستقل از متن است.
- این گرامر چه زبانی را توصیف می‌کند؟

گرامرهای مستقل از متن

- گرامر G با قوانین تولید $S \to abB \ , \ A \to aaBb \ , \ B \to bbAa \ , \ A \to \lambda$ یک گرامر مستقل از متن است.
- این گرامر زبان $L(G) = \{ab(bbaa)^n bba(ba)^n : n \geq ۰\}$ را توصیف می‌کند.
- این گرامر همچنین یک گرامر خطی [1] است. در یک گرامر خطی در سمت راست هر قانون حداکثر یک متغیر وجود دارد.

[1] linear grammar

- نشان دهید زبان $L = \{a^n b^m : n \neq m\}$ یک زبان مستقل از متن است.

گرامرهای مستقل از متن

- برای این که نشان دهیم زبان $L = \{a^n b^m : n \neq m\}$ یک زبان مستقل از متن است، باید یک گرامر مستقل از متن برای آن پیدا کنیم.

- ابتدا حالتی را در نظر می‌گیریم که در آن $n > m$ باشد. بنابراین ابتدا تعداد مساوی a و b تولید می‌کنیم و سپس تعداد b ها را می‌افزاییم. این کار را توسط گرامر $S \to AS_1 \ , \ S_1 \to aS_1b|\lambda \ , \ A \to aA|a$ انجام می‌دهیم.

- سپس حالتی را در نظر می‌گیریم که در آن $n < m$ باشد.

- در نتیجه داریم: $S \to AS_1|S_1B \ , \ S_1 \to aS_1b|\lambda \ , \ A \to aA|a \ , \ B \to bB|b$

- این گرامر یک گرامر مستقل از متن است ولی خطی نیست.

گرامرهای مستقل از متن

- گرامر $S \to aSb|bSa|SS|\lambda$ را در نظر بگیرید.
- این گرامر چه زبانی را توصیف می‌کند؟

گرامرهای مستقل از متن

- گرامر $S \to aSb|bSa|SS|\lambda$ را در نظر بگیرید.
- این گرامر زبانی را توصیف می‌کند که در همهٔ جملات آن تعداد a و b با هم برابرند.

- در یک گرامر مستقل از متن که خطی نیست، در فرایند اشتقاق، اگر یک صورت جمله‌ای بیش از یک متغیر داشته باشد، آنگاه چندین انتخاب برای اشتقاق وجود دارد.
- گرامر $G = (\{S, A, B\}, \{a, b\}, S, P)$ را با قوانین تولید P در نظر بگیرید:
- $۱.\ S \rightarrow AB \ , \ ۲.\ A \rightarrow aaA \ , \ ۳.\ A \rightarrow \lambda \ , \ ۴.\ B \rightarrow Bb \ , \ ۵.\ B \rightarrow \lambda$
- این گرامر چه زبانی را توصیف می‌کند؟

گرامرهای مستقل از متن

- در یک گرامر مستقل از متن که خطی نیست، در یک اشتقاق که در آن صورت جمله‌ای بیش از یک متغیر دارد، چندین انتخاب برای اشتقاق وجود دارد.
- گرامر $G = (\{S, A, B\}, \{a, b\}, S, P)$ را با قوانین تولید P در نظر بگیرید:
- $۱.\ S \rightarrow AB \ ,\ ۲.\ A \rightarrow aaA \ ,\ ۳.\ A \rightarrow \lambda \ ,\ ۴.\ B \rightarrow Bb \ ,\ ۵.\ B \rightarrow \lambda$
- این گرامر زبان $L = \{a^{۲n}b^m : n \geq ۰, m \geq ۰\}$ را توصیف می‌کند.
- اشتقاق یک جملهٔ واحد توسط این گرامر می‌تواند به اشکال گوناگون باشد. به طور مثال:
- $S \overset{۱}{\Rightarrow} AB \overset{۲}{\Rightarrow} aaAB \overset{۳}{\Rightarrow} aaB \overset{۴}{\Rightarrow} aaBb \overset{۵}{\Rightarrow} aab$
- $S \overset{۱}{\Rightarrow} AB \overset{۴}{\Rightarrow} ABb \overset{۲}{\Rightarrow} aaABb \overset{۵}{\Rightarrow} aaAb \overset{۳}{\Rightarrow} aab$

گرامرهای مستقل از متن

- یک اشتقاق را اشتقاق از چپ [1] (اشتقاق چپ یا اشتقاق چپ‌ترین) می‌نامیم اگر در هر مرحله از اشتقاق سمت چپ‌ترین متغیر در صورت جمله‌ای جایگزین شود.
- اگر در هر مرحله از اشتقاق سمت راست‌ترین صورت جمله‌ای جایگزین شود، آن اشتقاق را یک اشتقاق از راست [2] می‌نامیم.

[1] leftmost derivation
[2] rightmost derivation

گرامرهای مستقل از متن

- گرامری را با قوانین تولید $S \to aAB \quad , \quad A \to bBb \quad , \quad B \to A|\lambda$ در نظر بگیرید.
- اشتقاق $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB \Rightarrow abbbbB \Rightarrow abbbb$ یک اشتقاق از چپ است.
- اشتقاق $S \Rightarrow aAB \Rightarrow aA \Rightarrow aBbB \Rightarrow abAb \Rightarrow abbBbb \Rightarrow abbbb$ یک اشتقاق از راست است.

گرامرهای مستقل از متن

- گرامر $S \rightarrow aSb|bSa|SS|\lambda$ را در نظر بگیرید.
- یک اشتقاق چپ برای جملهٔ aaabbabb پیدا کنید.

گرامرهای مستقل از متن

- گرامر $S \to aSb|bSa|SS|\lambda$ را در نظر بگیرید.
- یک اشتقاق چپ برای جملهٔ aaabbabb به صورت زیر است.
- $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaSSbb \Rightarrow aaaSbSbb \Rightarrow aaabSbb \Rightarrow aaabbSabb \Rightarrow aaabbabb$

درخت اشتقاق

- یک روش دیگر برای نمایش اشتقاق که مستقل از ترتیب اعمال قوانین در فرایند اشتقاق است، استفاده از درخت اشتقاق [1] یا درخت تجزیه [2] است.

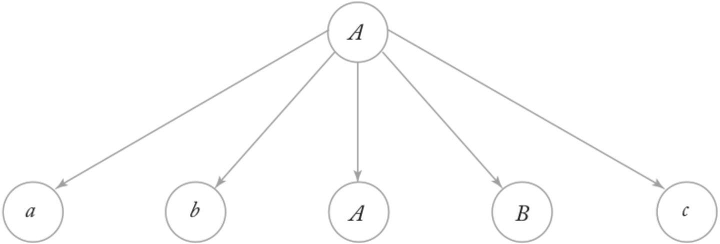
- درخت تجزیه یک درخت مرتب [3] است که در آن ریشه با متغیر آغازی برچسب زده شده است. فرزندان ریشه، نمادهایی از یک صورت جمله‌ای (با حفظ ترتیب) هستند که متغیر آغازی با آن جایگزین شده است. به همین ترتیب هر رأس میانی (رئوسی که برگ نیستند) با یک متغیر X برچسب زده شده است و فرزندان رأس X با نمادهای صورت جمله‌ای x (با حفظ ترتیب نمادهای x) برچسب زده شده‌اند به طوری که $X \to x$ یک قانون تولید در گرامر است. برگ‌های این درخت را پایانه‌ها تشکیل می‌دهند.

---

[1] derivation tree
[2] parse tree
[3] ordered tree

درخت اشتقاق

- برای مثال اگر قانون $A \rightarrow abABc$ را داشته باشیم، آنگاه در درخت اشتقاق می‌توانیم زیردرخت زیر را بیابیم.



درخت اشتقاق

- فرض کنید $G = (V, T, S, P)$ یک گرامر مستقل از متن باشد. یک درخت مرتب، درخت اشتقاق برای این گرامر است اگر و فقط اگر ویژگی‌های زیر را داشته باشد.

۱. ریشه با نماد S برچسب زده شده باشد.

۲. هر برگ با یکی از نمادهای $T \cup \{\lambda\}$ برچسب شده باشد.

۳. هر رأس میانی (رئوسی که ریشه و برگ نیستند) یک برچسب V داشته باشد.

۴. اگر رأسی برچسب $A \in V$ داشته باشد و فرزندان آن از چپ به راست برچسب‌های $a_1, a_2, \cdots, a_n$ داشته باشند، آنگاه یکی از قوانین تولید باید به صورت $A \to a_1 a_2 \cdots a_n$ باشد.

۵. برگی که با نماد λ برچسب زده شده است، هیچ همزادی (رأسی که با آن رأس از یک والد باشد) ندارد. به عبارت دیگر رأسی که فرزند آن λ است هیچ فرزند دیگری ندارد.

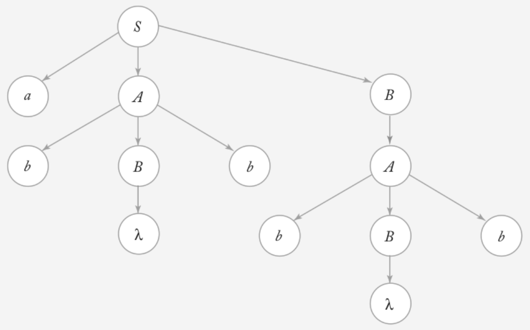
درخت اشتقاق

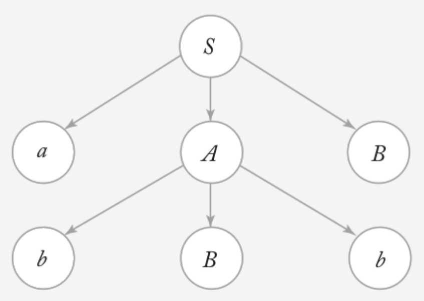
- حال اگر در یک درخت اشتقاق ریشه با یک متغیر غیر آغازی برچسب زده شده باشد و برگ‌ها متغیر یا نماد پایانی باشند، این درخت را یک درخت اشتقاق جزئی [1] می‌نامیم.
- اگر برچسب برگ‌های یک درخت را از چپ به راست به یک دیگر الحاق کنیم، محصول [2] درخت را به دست آورده‌ایم.
- در یک درخت اشتقاق محصول یک درخت، رشته‌ای از نمادهاست که توسط گرامر مربوط به آن درخت به دست آمده است.

[1] partial derivation tree
[2] yield

درخت اشتقاق

- گرامر G با قوانین تولید زیر را در نظر بگیرید: $S \rightarrow aAB \quad , \quad A \rightarrow bBb \quad , \quad B \rightarrow A|\lambda$
- درخت سمت راست یک درخت اشتقاق جزئی است که محصول آن صورت جمله‌ای abBbB است و درخت سمت چپ یک درخت اشتقاق است که محصول آن رشتهٔ abbbb است که جمله‌ای از زبان L(G) است.





درخت اشتقاق

- فرض کنید $G = (V, T, S, P)$ یک گرامر مستقل از متن باشد.
- آنگاه به ازای هر $w \in L(G)$ یک درخت اشتقاق وجود دارد که محصول آن w است. همچنین محصول یک درخت اشتقاق برای گرامر G رشته‌ای در $L(G)$ است.
- همچنین اگر t_G یک درخت اشتقاق جزئی برای گرامر G باشد که ریشه آن S باشد، آنگاه محصول درخت t_G یک صورت جمله‌ای است که از گرامر G به دست می‌آید (مشتق می‌شود).

درخت اشتقاق

- یک درخت اشتقاق نشان می‌دهد که جملات یک زبان چگونه از یک گرامر به دست آمده‌اند، اما چیزی در مورد ترتیب اعمال قوانین در یک اشتقاق به ما نمی‌گوید.
- بنابراین با پیمایش درخت اشتقاق با ترتیب‌های مختلف می‌توانیم یک اشتقاق از چپ یا یک اشتقاق از راست به دست بیاوریم.

مثال

- بر روی $\Sigma = \{0, 1\}$، یک گرامر مستقل از متن برای زبانی بیابید که نماد اول و آخر جملات آن یکسان باشند.

مثال

- بر روی $\Sigma = \{۰, ۱\}$، یک گرامر مستقل از متن برای زبانی بیابید که نماد اول و آخر جملات آن یکسان باشند.
- $S \to ۱T۱|۰T۰|۰|۱$
  $T \to ۰T|۱T|\lambda$

مثال

- بر روی $\Sigma = \{۰, ۱\}$، یک گرامر مستقل از متن برای زبانی بیابید که جملات آن حداقل سه نماد ۱ داشته باشند.

مثال

- بر روی $\Sigma = \{۰, ۱\}$، یک گرامر مستقل از متن برای زبانی بیابید که جملات آن حداقل سه نماد ۱ داشته باشند.
- $S \to T۱T۱T۱T \quad , \quad T \to ۱T|۰T|\lambda$

مثال

- بر روی $\Sigma = \{۰, ۱\}$، یک گرامر مستقل از متن برای زبانی بیابید که طول رشته‌های آن فرد است و نماد وسط رشته صفر است.

مثال

- بر روی $\{۰, ۱\} = \Sigma$، یک گرامر مستقل از متن برای زبانی بیابید که طول رشته‌های آن فرد است و نماد وسط رشته صفر است.
- $S \rightarrow ۰|۰S۰|۰S۱|۱S۰|۱S۱$

مثال

- یک گرامر مستقل از متن برای زبان $L = \{w \in \{a, b\}^* : n_a(w) \neq n_b(w)\}$ بیابید.

مثال

- یک گرامر مستقل از متن برای زبان $L = \{w \in \{a,b\}^* : n_a(w) \neq n_b(w)\}$ بیابید.
- دو حالت را در نظر می‌گیریم: (۱) تعداد نمادهای a بیشتر است و (۲) تعداد نمادهای b بیشتر است.
- راه حل اول:
- $$S \rightarrow S_1 a S_1 | S_2 b S_2$$
$$S_1 \rightarrow S_1 S_1 | a S_1 b | b S_1 a | A \quad , \quad A \rightarrow aA | \lambda$$
$$S_2 \rightarrow S_2 S_2 | a S_2 b | b S_2 a | B \quad , \quad B \rightarrow bB | \lambda$$
- راه حل دوم:
- $$S \rightarrow A | B$$
$$A \rightarrow AAb | AbA | bAA | aA | a$$
$$B \rightarrow BBa | BaB | aBB | bB | b$$

مثال

- یک گرامر مستقل از متن برای زبان $L = \{a^n b^m c^k : n = m \vee m \leq k\}$ بیابید.

مثال

- یک گرامر مستقل از متن برای زبان $L = \{a^n b^m c^k : n = m \vee m \leq k\}$ بیابید.
- مسأله را به دو قسمت تقسیم می‌کنیم: $L_1 = \{a^n b^m c^k : n = m\}$
  $L_2 = \{a^n b^m c^k : m \leq k\}$
  $L = L_1 \cup L_2$
- $S \to S_1 | S_2$
  $S_1 \to T_1 C \ , \ T_1 \to aT_1 b | \lambda \ , \ C \to cC | \lambda$
  $S_2 \to AT_2 \ , \ T_2 \to bT_2 c | C \ , \ A \to aA | \lambda$

مثال

- یک گرامر مستقل از متن برای زبان $L = \{a^n b^m c^k : k = n + ۲m\}$ بیابید.

مثال

- یک گرامر مستقل از متن برای زبان $L = \{a^n b^m c^k : k = n + ۲m\}$ بیابید.
- می‌توانیم زبان L را بدین شکل درآوریم: $L = \{a^n b^m c^{۲m} c^n\}$
- $S \rightarrow aSc|B \quad , \quad B \rightarrow bBcc|\lambda$

مثال

- یک گرامر مستقل از متن برای زبان $L = \{a^n ww^R b^n : w \in \Sigma^*, n \geq ۱\}$ بیابید.

- یک گرامر مستقل از متن برای زبان $L = \{a^n w w^R b^n : w \in \Sigma^*, n \geq ۱\}$ بیابید.
- $S \to aSb|aTb \quad , \quad T \to aTa|bTb|\lambda$

مثال

- بر روی $\Sigma = \{a, b\}$، یک گرامر مستقل از متن برای زبانی بیابید که همهٔ جمله‌های آن واروخوانه (پالیندروم[1]) باشند. واروخوانه به واژه‌هایی گفته می‌شود که خواندن آنها از چپ به راست و راست به چپ یکسان است. به طور رسمی تعریف می‌کنیم w یک جملهٔ واروخوانه است اگر $w = w^R$.

[1] palindrome

مثال

- بر روی $\Sigma = \{a, b\}$، یک گرامر مستقل از متن برای زبانی بیابید که همهٔ جمله‌های آن واروخوانه (پالیندروم [1]) باشند. واروخوانه به واژه‌هایی گفته می‌شود که خواندن آنها از چپ به راست و راست به چپ یکسان است. به طور رسمی تعریف می‌کنیم w یک جملهٔ واروخوانه است اگر $w = w^R$.
- $S \rightarrow aSa|bSb|a|b|\lambda$

[1] palindrome

مثال

- یک گرامر مستقل از متن برای زبانی بیابید که جملات آن واروخوانه (پالیندروم) نباشند: $L = \{w \in \{a, b\}^* : w \neq w^R\}$.

مثال

- یک گرامر مستقل از متن برای زبانی بیابید که جملات آن واروخوانه (پالیندروم) نباشند: $L = \{w \in \{a, b\}^* : w \neq w^R\}$.
- $S \rightarrow aSa|bSb|aTb|bTa \quad , \quad T \rightarrow aTa|bTb|aTb|bTa|a|b|\lambda$

مثال

- بر روی $\Sigma = \{a, b\}$، یک گرامر مستقل از متن برای زبان $L = \{a^n b^m : n \neq m\}$ بیابید.

مثال

- بر روی $\Sigma = \{a, b\}$، یک گرامر مستقل از متن برای زبان $L = \{a^n b^m : n \neq m\}$ بیابید.
- دو حالت را در نظر می‌گیریم: یا تعداد نمادهای a بیشتر است و یا تعداد نمادهای b .
- $S \to AS_۱|S_۱B \quad , \quad S_۱ \to aS_۱b|\lambda \quad , \quad A \to aA|a \quad , \quad B \to bB|b$

مثال

- یک گرامر مستقل از متن برای زبان $L = \{a^m b^n c^p d^q : m, n, p, q \geq 0, m + n = p + q\}$ بیابید.

مثال

- یک گرامر مستقل از متن برای زبان $L = \{a^m b^n c^p d^q : m, n, p, q \geq ۰, m + n = p + q\}$ بیابید.
- به ازای تولید هر a (و همینطور به ازای تولید هر b) باید یک c یا یک d تولید کنیم.
- دو حالت در نظر می‌گیریم:

$$m \geq q \ , \ n \leq p \ (۱)$$
$$m \leq q \ , \ n \geq p \ (۲)$$

- برای حالت اول داریم:

$$S \rightarrow aSd|A \ , \ A \rightarrow aAc|C \ , \ C \rightarrow bCc|\lambda$$

- برای حالت دوم داریم:

$$S \rightarrow aSd|B \ , \ B \rightarrow bBd|C \ , \ C \rightarrow bCc|\lambda$$

- از اجتماع این دو حالت داریم:

$$S \rightarrow aSd|A|B \ , \ A \rightarrow aAc|C \ , \ B \rightarrow bBd|C \ , \ C \rightarrow bCc|\lambda$$

مثال

- یک گرامر مستقل از متن برای متمم زبان $L = \{a^n b^n : n \geq ۰\}$ بیابید.

مثال

- یک گرامر مستقل از متن برای متمم زبان $L = \{a^n b^n : n \geq ۰\}$ بیابید.
- $\overline{L} = \{a^m b^n : m < n\} \cup \{a^m b^n : m > n\} \cup L((a+b)^* b (a+b)^* a (a+b)^*)$
- پس سه حالت در نظر می‌گیریم: یا $m < n$ یا $m > n$ و یا در جملهٔ مورد نظر حداقل یک b قبل از a قرار دارد.
- $$\begin{aligned} S &\rightarrow S_۱ | S_۲ | S_۳ \\ S_۱ &\rightarrow aS_۱b | aS_۱ | a \\ S_۲ &\rightarrow aS_۲b | S_۲b | b \\ S_۳ &\rightarrow TbTaT \\ T &\rightarrow aT | bT | \lambda \end{aligned}$$

مثال

- یک گرامر مستقل از متن برای زبان $L = \{a^m b^n c^p : m, n, p \geq ۰, n \neq m + p\}$ بیابید.

مثال

- یک گرامر مستقل از متن برای زبان $L = \{a^m b^n c^p : m, n, p \geq ۰, n \neq m + p\}$ بیابید.
- دو حالت در نظر می‌گیریم: $n < m + p$ یا $n > m + p$
- ابتدا گرامری برای زبان $L_۲ = \{a^m b^n c^p : m, n, p \geq ۰, n = m + p\}$ می‌نویسیم. سپس یا نمادهای a یا نمادهای c یا هر دو نماد a و c را افزایش می‌دهیم تا به دست آوریم $n < m + p$ یا نمادهای b را افزایش می‌دهیم تا به دست آوریم $n > m + p$
- همچنین زبان $L_۲$ را می‌توانیم به صورت $L_۲ = \{a^m b^m b^p c^p : m, p \geq ۰\}$ بازنویسی کنیم.

تجزیه

- اکنون به ازای یک رشتهٔ داده شده w می‌خواهیم بدانیم آیا اشتقاقی برای این رشته توسط قوانین تولید گرامر G وجود دارد یا خیر.
- به عبارت دیگر می‌خواهیم بدانیم آیا رشتهٔ w عضوی از زبان $L(G)$ است یا خیر.
- پیدا کردن دنباله‌ای از قوانین تولید گرامر G که توسط آن رشتهٔ w به دست می‌آید (مشتق می‌شود) را تجزیه [1] می‌گوییم.

[1] parsing

تجزیه

- بدیهی‌ترین راه برای تجزیه، جستجو در تمام اشتقاق‌های ممکن است. اگر از یکی از اشتقاق‌ها در گرامر G جملهٔ w به دست بیاید می‌گوییم w عضوی از زبان L(G) است.

- به عبارت دیگر، اگر $S \overset{*}{\Rightarrow} w$ آنگاه $w \in L(G)$.

- این نوع تجزیه را تجزیهٔ جستجوی کامل [1] یا تجزیهٔ از بالا به پایین [2] می‌نامیم.

---

[1] exhaustive search parsing
[2] top-down parsing

- یک الگوریتم ساده برای تجزیهٔ از بالا به پایین چنین است:

۱. از متغیر آغازی S شروع می‌کنیم و تمام قوانین تولید را اعمال می‌کنیم. با اعمال این قوانین تعدادی صورت جمله‌ای و یا جمله به دست می‌آید.

۲. در صورتی که در این مرحله یکی از جمله‌های به دست آمده رشتهٔ w باشد، به نتیجه رسیده‌ایم در غیراینصورت تمام قوانین تولید ممکن بر روی یکی از متغیرهای صورت‌های جمله‌ای به دست آمده در مرحلهٔ قبل اعمال می‌شود (این اعمال قانون می‌تواند به عنوان مثال بر روی چپ‌ترین متغیر صورت بگیرد).

۳. این روند ادامه پیدا می‌کند تا جایی که یا رشتهٔ w از این گرامر مشتق شود و یا همهٔ حالت‌ها بررسی شده و رشتهٔ w به دست نیاید.

تجزیه

- گرامر $S \to SS|aSb|bSa|\lambda$ را در نظر بگیرید. یک اشتقاق برای رشتهٔ $w = aabb$ با استفاده از تجزیهٔ جستجوی کامل پیدا کنید.

تجزیه

- گرامر $S \to SS|aSb|bSa|\lambda$ را در نظر بگیرید. یک اشتقاق برای رشتهٔ $w = aabb$ با استفاده از تجزیهٔ جستجوی کامل پیدا کنید.

- در مرحلهٔ اول داریم λ ⇒ ۱. S ⇒ SS, ۲. S ⇒ aSb, ۳. S ⇒ bSa, ۴. S اما اشتقاق سوم و چهارم حذف می‌شود.

- در مرحلهٔ دوم داریم
۱. S ⇒ SS ⇒ SSS, ۲. S ⇒ SS ⇒ aSbS, ۳. S ⇒ SS ⇒ bSaS, ۴. S ⇒ SS ⇒ S
و همچنین
۵. S ⇒ aSb ⇒ aSSb, ۶. S ⇒ aSb ⇒ aaSbb, ۷. S ⇒ aSb ⇒ abSab, ۸. S ⇒ aSb ⇒ ab

- در مرحلهٔ سوم از اشتقاق ششم مرحلهٔ قبل به دست می‌آوریم: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

تجزیه

- تجزیهٔ جستجوی کامل چندین نقص دارد.
- اول اینکه از لحاظ زمان اجرا پرهزینه است زیرا تمام مسیرها جستجو می‌شوند. پس در جایی که به تجزیهٔ کارامد نیاز است نمی‌توان از آن استفاده کرد.
- دوم اینکه برای جمله‌هایی که عضوی از زبان آن گرامر نیستند، الگوریتم ممکن است هیچ‌گاه به پایان نرسد، مگر اینکه راهی برای توقف آن بیابیم.

تجزیه

- برای حل مشکل خاتمه‌ناپذیری الگوریتم تجزیهٔ جستجوی کامل، باید محدودیتی بر روی شکل قوانین اعمال کنیم.
- برای مثال اگر قوانینی که به شکل $A \to \lambda$ و $A \to B$ هستند را حذف کنیم، در این صورت طول صورت‌های جمله‌ای همیشه افزایش پیدا می‌کند و بنابراین وقتی طول همهٔ صورت‌های جمله‌ای از جملهٔ w بیشتر شد می‌توانیم الگوریتم را متوقف کنیم.

تجزیه

- برای مثال گرامر $S \to SS|aSb|bSa|ab|ba$ معادل گرامر $S \to SS|aSb|bSa|\lambda$ است و زبانی که هر دو تولید می‌کنند برابر است (با این تفاوت که با استفاده از گرامر اول رشتهٔ تهی تولید نمی‌شود).
- تفاوت این دو گرامر در هنگام تجزیه این است که اگر از تجزیهٔ جستجوی کامل در گرامر اول استفاده کنیم در هر مرحله طول صورت‌های جمله‌ای افزایش پیدا می‌کند بنابراین بعد از چندین مرحله معین می‌توانیم الگوریتم را به پایان برسانیم.

تجزیه

- قضیه: فرض کنید $G = (V, T, S, P)$ یک گرامر مستقل از متن است که هیچ یک از قوانین آن به شکل $A \to \lambda$ یا $A \to B$ نیستند، جایی که $A, B \in V$. آنگاه الگوریتم جستجوی کامل یا رشتهٔ مورد نظر w را تولید می‌کند و یا بعد از چندین مرحله متوقف می‌شود.

- قضیه: فرض کنید $G = (V, T, S, P)$ یک گرامر مستقل از متن است که هیچ یک از قوانین آن به شکل $A \to \lambda$ یا $A \to B$ نیستند، جایی که $A, B \in V$. آنگاه الگوریتم جستجوی کامل یا رشتهٔ مورد نظر w را تولید می‌کند و یا بعد از چندین مرحله متوقف می‌شود.
- اثبات: در هر مرحله از فرایند اشتقاق یا یکی از متغیرها با یک پایانه جایگزین می‌شود و یا طول صورت جمله‌ای حداقل یک واحد افزایش پیدا می‌کند. این افزایش طول یا توسط یک متغیر است و یا توسط یک پایانه. در بدترین حالت در $|w|$ مرحله طول صورت جمله‌ای توسط متغیرها افزایش می‌یابد و در $|w|$ مرحله هر متغیر با یک پایانه جایگزین می‌شود، بنابراین در بدترین حالت بعد از $۲|w|$ مرحله می‌توان تعیین کرد یک جمله عضو زبان این گرامر است یا خیر.
- اگر تعداد قوانین $|P|$ باشد، حداکثر $|P|^{۲|w|}$ حالت مختلف باید بررسی شوند تا بتوانیم اشتقاق جملهٔ w را به دست آوریم.

تجزیه

- بهترین الگوریتمی که برای تجزیهٔ یک گرامر مستقل از متن وجود دارد در $|w|^3$ گام می‌تواند جملهٔ w را تجزیه کند.
- الگوریتم‌های بهینه‌تری نیز برای تجزیه در حالات خاص وجود دارند که در نظریهٔ کامپایلرها به این الگوریتم‌ها پرداخته می‌شود و از حوزهٔ نظریهٔ زبان‌ها و ماشین‌ها خارج است.

- یک گرامر مستقل از متن $G = (V, T, S, P)$ یک گرامر ساده [1] گفته می‌شود اگر همهٔ قوانین آن به شکل $A \to ax$ باشد به طوری که $A \in V, a \in T, x \in V^*$ و هر جفت (A, a) حداکثر یک بار در قوانین تولید تکرار شده باشد.

[1] simple grammar or s-grammar

- گرامر $S \to aS|bSS|c$ یک گرامر ساده است.
- گرامر $S \to aS|bSS|aSS|c$ یک گرامر ساده نیست زیرا زوج (S, a) در دو قانون تکرار شده است.

تجزیه

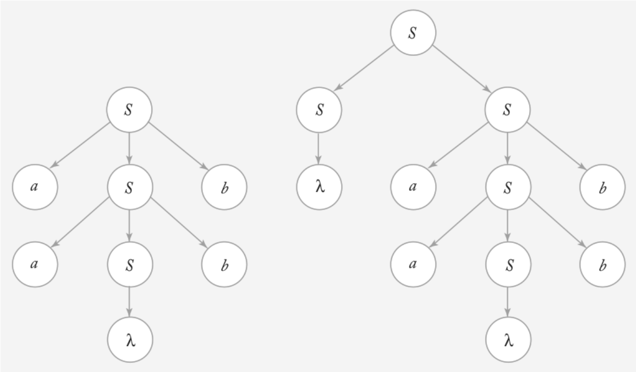
- برای یک گرامر ساده، الگوریتم تجزیه توسط جستجوی کامل برای جملهٔ w به زمانی با پیچیدگی $|w|$ نیاز دارد.
- فرض کنید داشته باشیم $w = a_۱a_۲ \cdots a_n$.
- از آنجایی که تنها یک قانون وجود دارد که در سمت چپش S و در سمت راستش با $a_۱$ آغاز می‌شود، بنابراین می‌توانیم اشتقاق $S \Rightarrow a_۱A_۱A_۲ \cdots A_m$ را در یک گام به دست آوریم.
- سپس باید متغیر $A_۱$ را جایگزین کنیم : $S \overset{*}{\Rightarrow} a_۱a_۲B_۱B_۲ \cdots A_۲ \cdots A_m$
- بدین ترتیب در هر مرحله یک پایانه به دست می‌آید و در $|w|$ گام جملهٔ w مشتق می‌شود.

- یک گرامر مستقل از متن مبهم [1] است اگر به ازای یک جمله $w \in L(G)$ حداقل دو درخت اشتقاق وجود داشته باشد. و یا به عبارت دیگر حداقل دو اشتقاق چپ یا راست برای یکی از جملات زبان آن گرامر وجود داشته باشد.

[1] ambiguous

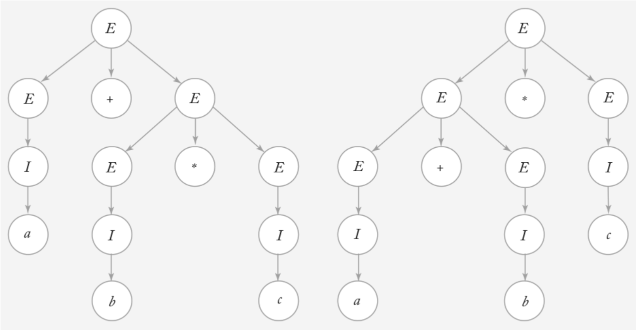
ابهام در گرامرها

- گرامر $S \rightarrow aSb|SS|\lambda$ مبهم است زیرا برای جملهٔ aabb دو درخت اشتقاق زیر وجود دارد.



ابهام در گرامرها

- گرامر $G = (V = \{E, I\}, T = \{a, b, c, +, *, (,)\}, E, P)$ را با قوانین تولید زیر در نظر بگیرید.
- $E \rightarrow I|E+E|E*E|(E) \quad , \quad I \rightarrow a|b|c$
- در این گرامر برای جملهٔ $a + b * c$ دو درخت اشتقاق وجود دارد.

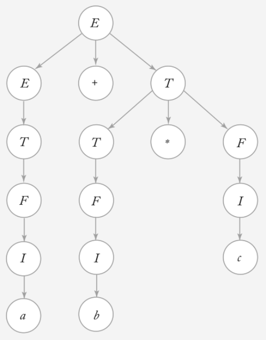


ابهام در گرامرها

- بنابراین گرامر قبل برای تولید جملات جبری با دو عملگر جمع و ضرب مبهم است.
- برای حل این مشکل، گرامر را به شکلی دیگر می‌نویسیم.

ابهام در گرامرها

- گرامر $G = (V = \{E, T, F, I\}, T = \{a, b, c, +, *, (,)\}, E, P)$ را با قوانین تولید زیر در نظر بگیرید.
- $E \rightarrow T|E + T \quad , \quad T \rightarrow F|T * F \quad , \quad F \rightarrow I|(E) \quad , \quad I \rightarrow a|b|c$
- در این گرامر برای جملهٔ $a + b * c$ فقط یک درخت اشتقاق وجود دارد.



ابهام در گرامرها

- گرچه توانستیم برای گرامری که جملات جبری تولید می‌کند یک گرامر غیرمبهم طراحی کنیم ولی الگوریتمی کلی برای تبدیل یک گرامر مبهم به یک گرامر غیرمبهم وجود ندارد.

ابهام در گرامرها

- اگر L یک زبان مستقل از متن باشد که برای آن یک گرامر غیرمبهم وجود داشته باشد، زبان L یک زبان غیرمبهم است.
- اگر هر گرامری که برای زبان L وجود دارد مبهم باشد، آنگاه زبان L ذاتا مبهم [1] است.
- اگر برای زبانی یک گرامر ساده وجود داشته باشد، آن زبان ذاتا مبهم نیست، چون با استفاده از گرامر ساده برای تجزیهٔ یک جمله تنها یک درخت می‌توان رسم کرد.

[1] inherently ambiguous

ابهام در گرامرها

- زبان $L = \{a^n b^n c^m : n, m \geq ۰\} \cup \{a^n b^m c^m : n, m \geq ۰\}$ ذاتا مبهم است.
- می‌توان گرامر $S \rightarrow S_۱ | S_۲ \quad , \quad S_۱ \rightarrow S_۱ c | A \quad , \quad A \rightarrow aAb | \lambda \quad , \quad S_۲ \rightarrow aS_۲ | B \quad , \quad B \rightarrow bBc | \lambda$ را برای آن طراحی کرد.
- چرا این گرامر مبهم است؟

ابهام در گرامرها

- زبان $L = \{a^nb^nc^m : n, m \geq ۰\} \cup \{a^nb^mc^m : n, m \geq ۰\}$ ذاتا مبهم است.
- می‌توان گرامر $S \to S_۱|S_۲ \quad , \quad S_۱ \to S_۱c|A \quad , \quad A \to aAb|\lambda \quad , \quad S_۲ \to aS_۲|B \quad , \quad B \to bBc|\lambda$ را برای آن طراحی کرد.
- این گرامر مبهم است، زیرا برای جملهٔ $a^nb^nc^n$ دو اشتقاق وجود دارد که یکی با $S \Rightarrow S_۱$ و دیگری با $S \Rightarrow S_۲$ آغاز می‌شود.
- اثبات این که این زبان ذاتا مبهم است اثباتی طولانی است که در مقاله‌ای در سال ۱۹۸۷ چاپ شده است.

ساده‌سازی گرامرهای مستقل از متن

- بررسی کردیم که تجزیهٔ جملات یک زبان توسط الگوریتم جستجوی کامل با استفاده از قوانین تولید گرامر مستقل از متن ممکن است خاتمه‌پذیر نباشد.
- برای حل مشکل خاتمه‌پذیری گفتیم می‌توانیم قوانینی را که تهی یا یک متغیر واحد تولید می‌کنند را حذف کنیم.
- در اینجا روشی برای حذف قوانین تولید تهی [1] (قانون تولید لامبدا یا اپسیلون) و همچنین حذف قوانین تولید یکه [2] (قانون تولید واحد یا تک‌متغیر) ارائه می‌کنیم.

[1] λ-productions
[2] unit-productions

ساده‌سازی گرامرهای مستقل از متن

- همچنین بررسی کردیم که الگوریتم جستجوی کامل، یک جمله را در زمانی از مرتبهٔ نمایی n به ازای جمله‌های با طول n تجزیه می‌کند.
- در اینجا در مورد دو فرم (شکل) نرمال [1] برای گرامرهای مستقل از متن، به نام فرم نرمال چامسکی [2] و فرم نرمال گریباخ [3] صحبت می‌کنیم.
- سپس الگوریتمی ارائه می‌کنیم که با استفاده از فرم نرمال چامسکی تجزیهٔ یک جمله را در زمان چندجمله‌ای $(n^۳)$ برای جمله‌های با طول n انجام می‌دهد.

[1] normal form
[2] Chomsky normal form
[3] Greibach normal form

ساده‌سازی گرامرهای مستقل از متن

- فرض کنید $G = (V, T, S, P)$ یک گرامر مستقل از متن است. فرض کنید P قانونی به شکل $A \to x_1 B x_2$ دارد.

- فرض کنید A و B دو متغیر متفاوتند و $B \to y_1|y_2|\cdots|y_n$ مجموعهٔ قوانین تولید در P است که در طرف چپ آنها متغیر B است.

- فرض کنید $\widehat{G} = (V, T, S, \widehat{P})$ گرامری است که $\widehat{P}$ به گونه‌ای ساخته شده است که در آن همهٔ قوانین $A \to x_1 B x_2$ در P با قوانین $A \to x_1 y_1 x_2|x_1 y_2 x_2|\cdots|x_1 y_n x_2$ جایگزین شده‌اند.

- آنگاه داریم: $L(\widehat{G}) = L(G)$

- اثبات: به ازای هر $w \in L(G)$ اگر یک اشتقاق از w را در نظر بگیریم این اشتقاق توسط گرامر $\widehat{G}$ نیز امکان پذیر است و بالعکس برای هر $w \in L(\widehat{G})$ اشتقاق آن در G نیز امکان‌پذیر است.

ساده‌سازی گرامرهای مستقل از متن

- گرامر $G = (\{A, B\}, \{a, b, c\}, A, P)$ با قوانین تولید $A \to a|aaA|abBc \quad , \quad B \to abbA|b$ را در نظر بگیرید.
- می‌توانیم متغیر B را در این گرامر بدین صورت جایگزین کنیم: $A \to a|aaA|ababbAc|abbc$

ساده‌سازی گرامرهای مستقل از متن

- در صورتی که یک قانون تولید هیچ نقشی در فرایند اشتقاق برای هیچ جمله‌ای نداشته باشد، می‌توانیم آن قانون را حذف کنیم.
- برای مثال در گرامری با قوانین تولید $S \to aSb|\lambda|A \quad , \quad A \to aA$ قانون $S \to A$ منجر به تولید هیچ جمله‌ای نمی‌شود و می‌توان آن را حذف کرد.

ساده‌سازی گرامرهای مستقل از متن

- فرض کنید $G = (V, T, S, P)$ یک گرامر مستقل از متن باشد. متغیر $A \in V$ مفید است اگر و تنها اگر جملهٔ $w \in L(G)$ وجود دارد به طوری که $S \overset{*}{\Rightarrow} xAy \overset{*}{\Rightarrow} w$ جایی که $x, y \in (V \cup T)^*$.
- به عبارت دیگر متغیر A مفید است اگر و تنها اگر در اشتقاق یکی از جملات حضور داشته باشد.
- اگر متغیری مفید نباشد آن را یک متغیر غیرمفید (بی‌استفاده) می‌خوانیم. می‌توانیم تمام قوانین تولیدی که شامل متغیرهای غیرمفید هستند را حذف کنیم.

ساده‌سازی گرامرهای مستقل از متن

- در گرامر زیر با متغیر آغازی S متغیر B غیر مفید است: $S \to A \ , \ A \to aA|\lambda \ , \ B \to bA$
- گرچه با شروع از متغیر B می‌توان رشته‌هایی تولید کرد اما با شروع از متغیر آغازی S هیچ رشته‌ای با استفاده از متغیر B تولید نمی‌شود و بنابراین متغیر B غیرمفید است.

ساده‌سازی گرامرهای مستقل از متن

- برای ساده سازی گرامر ابتدا متغیرهایی را که منجر به هیچ رشته‌ای نمی‌شوند (متغیرهای غیرسازنده) را حذف می‌کنیم.

- سپس متغیرهایی را که از متغیر آغازی قابل دسترسی نیستند (متغیرهای غیرقابل دسترس) حذف می‌کنیم.

- ترتیب حذف این متغیرها مهم است، زیرا بعد از حذف متغیرهای غیرسازنده ممکن است متغیرهای غیرقابل دسترس به وجود آید.

- برای مثال در قوانین تولید $S \to AB|a \ , \ A \to a \ , \ B \to Bc$ متغیر B غیرسازنده است. حذف متغیر B باعث حذف قانون $S \to AB$ می‌شود و بعد از این حذف، متغیر A غیرقابل دسترس می‌شود (گرچه در ابتدا متغیر A قابل دسترسی بود).

ساده‌سازی گرامرهای مستقل از متن

- در یک گرامر مستقل از متن به هر قانونی که به شکل $A \to \lambda$ باشد یک قانون تولید تهی [1] می‌گوییم.
- هر متغیری که از آن یک رشتهٔ تهی مشتق می‌شود، یعنی $A \overset{*}{\Rightarrow} \lambda$ یک متغیر تهی‌شدنی [2] می‌گوییم.
- یک گرامر ممکن است رشتهٔ تهی تولید نکند ولی شامل تعدادی قانون تولید تهی باشد. در چنین گرامری قوانین تولید تهی قابل حذف هستند.

[1] λ-production
[2] nullable

ساده‌سازی گرامرهای مستقل از متن

- برای مثال در گرامر $S \to aS_1b \ , \ S_1 \to aS_1b|\lambda$ می‌توان قانون تولید تهی را بدین صورت حذف کرد:
- $S \to aS_1b|ab \ , \ S_1 \to aS_1b|ab$

ساده‌سازی گرامرهای مستقل از متن

- فرض کنید G یک گرامر مستقل از متن برای زبان $L(G)$ باشد به طوری که رشتهٔ تهی در این زبان نباشد. در اینصورت گرامر $\widehat{G}$ وجود دارد که معادل گرامر G است و در آن قانون تولید تهی وجود ندارد.

- الگوریتم: ابتدا به ازای هر قانون تولید $A \to \lambda$ متغیر A را در مجموعهٔ V_N شامل همهٔ متغیرهای تهی‌شدنی اضافه می‌کنیم. سپس به ازای هر قانون تولید $B \to A_1 A_2 \cdots A_n$ که $A_1, A_2, \cdots A_n$ در مجموعه V_N قرار دارند، متغیر B را نیز به مجموعهٔ V_N اضافه می‌کنیم.

- سپس به ازای هر قانون $A \to x_1 x_2 \cdots x_m$ به طوری که $x_i \in V \cup T$ همهٔ حالت‌هایی را محاسبه می‌کنیم که یک یا تعدادی از متغیرهای تهی‌شدنی از طرف راست قانون حذف شوند. هر یک از این حالت‌ها را متغیر A ممکن است تولید کند، پس این حالت‌ها را به طرف راست قانون با علامت یا (خط عمودی |) اضافه می‌کنیم.

- برای مثال اگر دو متغیر x_i و x_i تهی‌شدنی باشند، آنگاه سه حالت وجود دارد: (۱) هر دو متغیر حذف شوند، (۲) متغیر x_i حذف شود، (۳) متغیر x_j حذف شود.

ساده‌سازی گرامرهای مستقل از متن

- یک گرامر مستقل از متن معادل گرامر زیر بدون قانون تهی پیدا کنید:
- $S \to ABaC \;\;,\;\; A \to BC \;\;,\;\; B \to b|\lambda \;\;,\;\; C \to D|\lambda \;\;,\;\; D \to d$

ساده‌سازی گرامرهای مستقل از متن

- یک گرامر مستقل از متن معادل گرامر زیر بدون قانون تهی پیدا کنید:
- $S \rightarrow ABaC \ , \ A \rightarrow BC \ , \ B \rightarrow b|\lambda \ , \ C \rightarrow D|\lambda \ , \ D \rightarrow d$
- ابتدا متغیرهای تهی‌شدنی را به مجموعهٔ V_N اضافه می‌کنیم. متغیرهای B و C و A تهی‌شدنی هستند.
- سپس قوانین تولید تهی را حذف می‌کنیم.
- در پایان همه حالت‌هایی را محاسبه می‌کنیم که یک یا چند متغیر تهی‌شدنی، در سمت راست قوانین حذف شوند. بنابراین داریم:

$$S \rightarrow ABaC|BaC|AaC|ABa|aC|Aa|Ba|a \ , \ A \rightarrow B|C|BC \ , \ B \rightarrow b \ , \ C \rightarrow D \ , \ D \rightarrow d$$

ساده‌سازی گرامرهای مستقل از متن

- بعد از حذف قوانین تولید تهی، قوانین تولید یکه را حذف می‌کنیم.
- در یک گرامر مستقل از متن قانون $A \to B$ به طوری که $A, B \in V$ باشند، قانون تولید یکه [1] نامیده می‌شود.

[1] unit-production

ساده‌سازی گرامرهای مستقل از متن

- فرض کنید $G = (V, T, S, P)$ یک گرامر مستقل از متن بدون قانون تولید تهی باشد. آنگاه یک گرامر مستقل از متن $\widehat{G} = (\widehat{V}, \widehat{T}, \widehat{S}, \widehat{P})$ وجود دارد که هیچ قانون تولید یکه ندارد.

- تمام قوانین تولید $A \rightarrow A$ می‌توانند بدون هیچ تأثیری در گرامر حذف شوند، پس فقط قوانین $A \rightarrow B$ را در نظر می‌گیریم، به طوری که $A \neq B$.

- سپس به ازای هر متغیر $A \in V$ متغیرهای $B \in V$ را محاسبه می‌کنیم به طوری که $A \overset{*}{\Rightarrow} B$.

- همهٔ قوانین تولید یکه به صورت $A \rightarrow B$ را حذف می‌کنیم.

- حال اگر داشته باشیم $B \rightarrow y_۱|y_۲|\cdots|y_n$ و $A \overset{*}{\Rightarrow} B$ آنگاه قوانین تولید $A \rightarrow y_۱|y_۲|\cdots|y_n$ را به قوانین تولید اضافه می‌کنیم.

- همهٔ قوانین تولید یکه را در گرامر $S \rightarrow Aa|B \;\; , \;\; B \rightarrow A|bb \;\; , \;\; A \rightarrow a|bc|B$ حذف کنید.

ساده‌سازی گرامرهای مستقل از متن

- همهٔ قوانین تولید یکه را در گرامر $A \rightarrow a|bc|B \quad , \quad B \rightarrow A|bb \quad , \quad S \rightarrow Aa|B$ حذف کنید.

- ابتدا به ازای هر متغیر در گرامر، همهٔ تک‌متغیرهایی را که تولید می‌کند را محاسبه می‌کنیم. در این گرامر داریم:
$S \overset{*}{\Rightarrow} B, S \overset{*}{\Rightarrow} A, B \overset{*}{\Rightarrow} A, A \overset{*}{\Rightarrow} B$

- سپس قوانین $S \rightarrow B, B \rightarrow A, A \rightarrow B$ را حذف و قوانین $S \rightarrow bb|a|bc, A \rightarrow bb, B \rightarrow a|bc$ را به قوانین اضافه می‌کنیم.

- در نهایت مجموعه قوانین $S \rightarrow a|bc|bb|Aa \quad , \quad A \rightarrow a|bb|bc \quad , \quad B \rightarrow a|bb|bc$ را به دست می‌آوریم.

- توجه کنید که بعد از حذف قوانین تولید یکه، متغیر B به یک متغیر غیرمفید تبدیل شد که می‌توان آن را حذف کرد.

ساده‌سازی گرامرهای مستقل از متن

- فرض کنید L یک زبان مستقل از متن باشد که شامل رشتهٔ تهی نباشد. آنگاه یک گرامر مستقل از متن برای آن وجود دارد که شامل هیچ قانون تولید غیرمفید، هیچ قانون تولید تهی، و هیچ قانون تولید یکه نشود.

- گرامر مستقل از متن G را برای زبان L در نظر می‌گیریم.

- (۱) ابتدا قوانین تولید تهی را با استفاده از الگوریتمی که اشاره شد حذف می‌کنیم. (۲) سپس قوانین تولید یکه را حذف می‌کنیم. (۳) در پایان قوانین تولید و متغیرهای غیرمفید (غیرسازنده و غیرقابل دسترس) را حذف می‌کنیم.

- توجه کنید که ترتیب حذف کردن مهم است، زیرا حذف قانون تولید تهی ممکن است قانون تولید یکه ایجاد کند اما حذف قانون تولید یکه، قانون تولید تهی ایجاد نمی‌کند. همچنین حذف قانون تولید یکه ممکن است قوانین غیرمفید ایجاد کند اما حذف قوانین غیرمفید، قوانین تولید تهی و یکه ایجاد نمی‌کند.

فرم نرمال

- برای گرامرهای مستقل از متن دو فرم نرمال وجود دارد که به طور گسترده‌ای استفاده شده‌اند:
- فرم نرمال چامسکی
- فرم نرمال گریباخ
- این فرم‌های نرمال برخی الگوریتم‌های تجزیه و اثبات‌ها را ساده می‌کنند و بدین دلیل به مطالعه آنها می‌پردازیم.

فرم نرمال چامسکی

- یک گرامر مستقل از متن به صورت فرم نرمال چامسکی است اگر همهٔ قوانین آن به صورت $A \to BC$ یا به صورت $A \to a$ باشند، به طوری که $A, B, C \in V$ و $a \in T$

- برای مثال گرامر $S \to AS|a \quad , \quad A \to SA|b$ به صورت فرم نرمال چامسکی است.

فرم نرمال چامسکی

- هر گرامر مستقل از متن $G = (V, T, S, P)$ به طوری که $\lambda \notin L(G)$ یک گرامر معادل به صورت فرم نرمال چامسکی دارد.

- اثبات: از آنجایی که در هر گرامر می‌توان قوانین تولید یکه و تهی و غیرمفید را حذف کرد، فرض می‌کنیم همهٔ این قوانین در G حذف شده‌اند. سپس گرامر $G_۱ = (V_۱, T, S, P_۱)$ را با استفاده از همهٔ قوانین تولید گرامر G که به صورت $A \to x_۱ x_۲ \cdots x_n$ هستند می‌سازیم. در اینجا $x_i \in (V \cup T)$.

- اگر $n = ۱$ آنگاه $x_۱$ باید یک نماد پایانی باشد، زیرا در گرامر G هیچ قانون تولید یکه‌ای وجود ندارد.

- اگر $n \geq ۲$ باشد، آنگاه به ازای هر نماد پایانی $x_i = a \in T$ یک متغیر B_a در نظر می‌گیریم. بنابراین داریم $A \to C_۱ C_۲ \cdots C_n$ به طوری که $C_i = x_i$ اگر $x_i \in V$ و $C_i = B_a$ اگر $x_i = a \in T$

- به ازای هر متغیر B_a قانون $B_a \to a$ را می‌سازیم.

- سپس همهٔ قوانین به صورت $A \to C_1C_2$ را به گرامر G_1 اضافه می‌کنیم.
- به ازای هر قانون $A \to C_1C_2\cdots C_n$ جایی که $n > 2$ است، متغیرهای $D_1, D_2, \cdots D_{n-2}$ را به متغیرهای گرامر G_1 اضافه می‌کنیم.
- سپس قوانین جدید به صورت فرم نرمال چامسکی را بدین صورت می‌سازیم:
  $A \to C_1D_1 \quad , \quad , D_1 \to C_2D_2 \quad , \quad \cdots D_{n-2} \to C_{n-1}C_n$

فرم نرمال چامسکی

- گرامر G با قوانین تولید $S \rightarrow ABa \quad , \quad A \rightarrow aab \quad , \quad B \rightarrow Ac$ را به صورت فرم نرمال چامسکی در آورید.

فرم نرمال چامسکی

- گرامر G با قوانین تولید $S \to ABa \ , \ A \to aab \ , \ B \to Ac$ را به صورت فرم نرمال چامسکی در آورید.

- این گرامر قانون تولید تهی، قانون تولید یکه، و قانون غیرمفید ندارد.

- ابتدا نمادهای پایانی را با متغیر جایگزین می‌کنیم: $S \to ABB_a \ , \ A \to B_aB_aB_b$
$B \to AB_c \ , \ B_a \to a \ , \ B_b \to b \ , \ B_c \to c$

- سپس گرامر را به صورت فرم نرمال چامسکی در می‌آوریم:
$S \to AD_۱ \ , \ D_۱ \to BB_a \ , \ A \to B_aD_۲ \ , \ D_۲ \to B_aB_b$
$B \to AB_c \ , \ B_a \to a \ , \ B_b \to b \ , \ B_c \to c$

فرم نرمال گریباخ

- یک گرامر مستقل از متن به صورت فرم نرمال گریباخ است اگر همهٔ قوانین تولید آن به صورت $A \to ax$ باشد به طوری که $a \in T$ و $x \in V^*$.

فرم نرمال گریباخ

- گرامر $S \to AB \quad , \quad A \to aA|bB|b \quad , \quad B \to b$ را به صورت فرم نرمال گریباخ درآورید.

فرم نرمال گریباخ

- گرامر $S \rightarrow AB \quad , \quad A \rightarrow aA|bB|b \quad , \quad B \rightarrow b$ را به صورت فرم نرمال گریباخ درآورید.
- قانون $S \rightarrow AB$ به صورت فرم نرمال گریباخ نیست و بنابراین می‌توانیم A را با مقادیر آن جایگزین کنیم.
- بنابراین داریم: $S \rightarrow aAB|bBB|bB \quad , \quad A \rightarrow aA|bB|b \quad , \quad B \rightarrow b$

فرم نرمال گریباخ

- گرامر $S \to abSb|aa$ را به صورت فرم نرمال گریباخ درآورید.
- می‌توانیم از تکنیک استفاده شده در تبدیل فرم نرمال چامسکی استفاده کنیم و تعدادی از نمادهای پایانی را با متغیرها جایگزین کنیم.
- بنابراین داریم: $S \to aBSB|aA \quad , \quad A \to a \quad , \quad B \to b$

الگوریتم سی‌وای‌کا

- الگوریتم سی‌وای‌کا [1] الگوریتمی است برای تجزیهٔ جملات با استفاده از یک گرامر مستقل از متن.
- فرض کنید $G = (V, T, S, P)$ یک گرامر مستقل از متن در فرم نرمال چامسکی باشد و $w = w_۱ w_۲ \cdots w_n$ رشته‌ای باشد که می‌خواهیم با استفاده از این گرامر تجزیه کنیم.
- تعریف می‌کنیم $w_{ij} = w_i \cdots w_j$ و همچنین $X_{ij} = \{X \in V : X \overset{*}{\Rightarrow} w_{ij}\}$.
- مجموعهٔ X_{ij} در واقع مجموعهٔ همهٔ متغیرهایی است که می‌توانند زیر رشتهٔ w_{ij} را تولید کنند.
- بنابراین داریم $w \in L(G)$ اگر و فقط اگر $S \in X_{۱n}$

[1] CYK (Cocke – Younger – Kasami) algorithm

الگوریتم سی‌وای‌کا

- می‌دانیم $X \in X_{ii}$ اگر و فقط اگر در گرامر G قانونی به صورت $X \rightarrow w_i$ وجود داشته باشد. به عبارت دیگر $X_{ii} = \{X : (X \rightarrow w_i) \in P\}$. بنابراین می‌توانیم X_{ii} را به ازای $۱ \leq i \leq n$ محاسبه کنیم.

- همچنین می‌دانیم $X \overset{*}{\Rightarrow} w_{ij}$ اگر و فقط اگر قانون تولید $X \rightarrow YZ$ وجود داشته باشد به طوری که $Y \overset{*}{\Rightarrow} w_{ik}$ و $Z \overset{*}{\Rightarrow} w_{(k+۱)j}$ به ازای $i \leq k < j$

- به عبارت دیگر داریم: $X_{ij} = \bigcup_{i \leq k < j} \{X : (X \rightarrow YZ) \in P, Y \in X_{ik}, Z \in X_{(k+۱)j}\}$

- بنابراین با استفاده از برنامه‌نویسی پویا [1] می‌توانیم مقدار $X_{۱n}$ را با استفاده از مقادیر به دست آمده از زیر مسئله‌ها به دست آوریم.

- برای این کار ابتدا $X_{۱۱}, X_{۲۲}, \cdots, X_{nn}$ را محاسبه می‌کنیم، سپس $X_{۱۲}, X_{۲۳}, \cdots, X_{(n-۱)n}$ و بعد از آن $X_{۱۳}, X_{۲۴}, \cdots, X_{(n-۲)n}$ و همینطور الی آخر تا مقدار $X_{۱n}$ محاسبه شود.

[1] dynamic programming

الگوریتم سی‌وای‌کا

- برای محاسبهٔ مقادیر X_{ij}، می‌توانیم جدولی به صورت زیر (برای مثال وقتی طول جمله ۶ است) تهیه کنیم:

۶	$X_{۱۶}$					
۵	$X_{۱۵}$	$X_{۲۶}$				
۴	$X_{۱۴}$	$X_{۲۵}$	$X_{۳۶}$			
۳	$X_{۱۳}$	$X_{۲۴}$	$X_{۳۵}$	$X_{۴۶}$		
۲	$X_{۱۲}$	$X_{۲۳}$	$X_{۳۴}$	$X_{۴۵}$	$X_{۵۶}$	
۱	$X_{۱۱}$	$X_{۲۲}$	$X_{۳۳}$	$X_{۴۴}$	$X_{۵۵}$	$X_{۶۶}$
	$w_{۱}$	$w_{۲}$	$w_{۳}$	$w_{۴}$	$w_{۵}$	$w_{۶}$

الگوریتم سی‌وای‌کا

- در این جدول، مقادیر X_{ii} مستقیما از مقادیر w_i به دست می‌آیند. به عبارت دیگر $X_{ii} = \{X : (X \to w_i) \in P\}$.

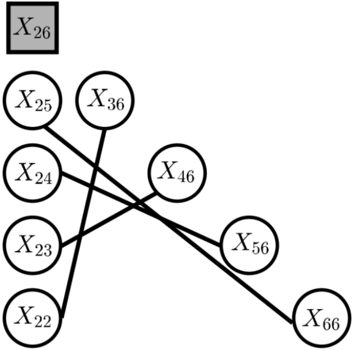
- همچنین مقادیر X_{ij} از اجتماع $j - i$ حالت به صورت $X_{ij} = \bigcup_{i \le k < j} \{X : (X \to YZ) \in P, Y \in X_{ik}, Z \in X_{(k+۱)j}\}$ به دست می‌آید که این جدول محاسبه این حالت‌ها را تسهیل می‌کند.

۶	$X_{۱۶}$					
۵	$X_{۱۵}$	$X_{۲۶}$				
۴	$X_{۱۴}$	$X_{۲۵}$	$X_{۳۶}$			
۳	$X_{۱۳}$	$X_{۲۴}$	$X_{۳۵}$	$X_{۴۶}$		
۲	$X_{۱۲}$	$X_{۲۳}$	$X_{۳۴}$	$X_{۴۵}$	$X_{۵۶}$	
۱	$X_{۱۱}$	$X_{۲۲}$	$X_{۳۳}$	$X_{۴۴}$	$X_{۵۵}$	$X_{۶۶}$
	$w_۱$	$w_۲$	$w_۳$	$w_۴$	$w_۵$	$w_۶$

الگوریتم سی‌وای‌کا

- برای مثال مقدار $X_{۲۶}$ را در این جدول، از اجتماع چهار حالت ممکن به دست می‌آوریم:

$$X_{۲۶} = \{X : (X \rightarrow YZ) \in P, Y \in X_{۲۲}, Z \in X_{۳۶}\} \cup \{X : (X \rightarrow YZ) \in P, Y \in X_{۲۳}, Z \in X_{۴۶}\}$$
$$\cup \{X : (X \rightarrow YZ) \in P, Y \in X_{۲۴}, Z \in X_{۵۶}\} \cup \{X : (X \rightarrow YZ) \in P, Y \in X_{۲۵}, Z \in X_{۶۶}\}$$



الگوریتم سی‌وای‌کا

- با استفاده از الگوریتم سی‌وای‌کا بررسی کنید که آیا رشتهٔ $w = aabbb$ متعلق به زبان تولید شده توسط گرامر $S \rightarrow AB \;\; , \;\; A \rightarrow BB|a \;\; , \;\; B \rightarrow AB|b$ است یا خیر.

- با استفاده از الگوریتم سی‌وای‌کا بررسی کنید که آیا رشتهٔ $w = aabbb$ متعلق به زبان تولید شده توسط گرامر $S \to AB \ , \ A \to BB|a \ , \ B \to AB|b$ است یا خیر.
- از آنجایی که $w_{11} = a$ بنابراین X_{11} شامل متغیرهایی است که از آنها a مشتق می‌شود و همینطور الی آخر.
- بنابراین: $X_{11} = \{A\}, X_{22} = \{A\}, X_{33} = \{B\}, X_{44} = \{B\}, X_{55} = \{B\}$
- سپس X_{12} را محاسبه می‌کنیم. از آنجایی که $X_{12} = \{X : (X \to YZ) \in P, Y \in X_{11}, Z \in X_{22}\}$ و $X_{11} = X_{22} = \{A\}$ بنابراین $X_{12} = \emptyset$.
- سپس X_{23} را محاسبه می‌کنیم: $X_{23} = \{X : (X \to YZ) \in P, Y \in X_{22}, Z \in X_{33}\} = \{S, B\}$

الگوریتم سی‌وای‌کا

- اگر همهٔ مقادیر X_{ij} را محاسبه کنیم داریم:

$$
\begin{array}{c|lllll}
۵ & X_{۱۵} = \{S, B\} & & & & \\
۴ & X_{۱۴} = \{A\} & X_{۲۵} = \{S, B\} & & & \\
۳ & X_{۱۳} = \{S, B\} & X_{۲۴} = \{A\} & X_{۳۵} = \{S, B\} & & \\
۲ & X_{۱۲} = \emptyset & X_{۲۳} = \{S, B\} & X_{۳۴} = \{A\} & X_{۴۵} = \{A\} & \\
۱ & X_{۱۱} = \{A\} & X_{۲۲} = \{A\} & X_{۳۳} = \{B\} & X_{۴۴} = \{B\} & X_{۵۵} = \{B\} \\
\hline
 & a & a & b & b & b
\end{array}
$$

- بنابراین $S \in X_{۱۵}$ و در نتیجه $w \in L(G)$

الگوریتم سی‌وای‌کا

- پیچیدگی الگوریتم سی‌وای‌کا برابر است با $O(n^۳)$ به طوری که n طول کلمهٔ w برای تجزیه است.
- برای محاسبه پیچیدگی الگوریتم تعداد همهٔ X_{ij} ها را می‌شماریم. به ازای هر X_{ij} باید $j - i$ مجموعه محاسبه شوند.
- تعداد X_{ii} ها برابر است با n و در این حالت مقادیر X_{ii} را مستقیما از w به دست می‌آوریم. تعداد $X_{i(i+۱)}$ ها برابر است با $n - ۱$ و در این حالت $j - i$ برابر است با ۱ ، تعداد $X_{i(i+۲)}$ ها برابر است با $n - ۲$ و در این حالت $j - i$ برابر است با ۲ ، و الی آخر. در نهایت تعداد $X_{۱n}$ برابر است با ۱ و در این حالت $j - i$ برابر است با $n - ۱$.

- بنابراین پیچیدگی الگوریتم برابر است با:

$$
\begin{aligned}
n + (n-۱) \times ۱ + (n-۲) \times ۲ + \cdots + ۱ \times (n-۱) &= n + \Sigma_{k=۱}^{n-۱}(n-k) \times k \\
&= n + \Sigma_{k=۱}^{n}(n-k) \times k \\
&= n + n\Sigma_{k=۱}^{n}k - \Sigma_{k=۱}^{n}k^۲ \\
&= n + n\frac{n(n+۱)}{۲} - \frac{n(n+۱)(۲n+۱)}{۶} \\
&= n + \frac{n(n+۱)(n-۱)}{۶} \\
&= \frac{n^۳ + ۵n}{۶} = O(n^۳)
\end{aligned}
$$

الگوریتم سی‌وای‌کا

- جملهٔ bbabb را توسط الگوریتم سی‌وای‌کا به ازای گرامر زیر تجزیه کنید.

$$S \to AB|AC|AA$$

$$A \to CB|a \quad , \quad B \to AC|b \quad , \quad C \to CC|b$$

الگوریتم سی‌وای‌کا

- جملهٔ bbabb را توسط الگوریتم سی‌وای‌کا به ازای گرامر زیر تجزیه کنید.

$S \rightarrow AB|AC|AA$

$A \rightarrow CB|a \quad , \quad B \rightarrow AC|b \quad , \quad C \rightarrow CC|b$

5	$\{S, A, B\}$				
4	$\{S, A\}$	$\{S, A, B\}$			
3	$\{S\}$	$\{A\}$	$\{S, B\}$		
2	$\{A, C\}$	$\emptyset$	$\{S, B\}$	$\{A, C\}$	
1	$\{B, C\}$	$\{B, C\}$	$\{A\}$	$\{B, C\}$	$\{B, C\}$
	1	2	3	4	5
	b	b	a	b	b

ماشین‌های پشته‌ای

ماشین‌های پشته‌ای

- علاوه بر گرامرهای مستقل از متن، ماشین‌های پشته‌ای [1] نیز برای پذیرش زبان‌های مستقل از متن به کار می‌روند.

- ماشین‌های پشته‌ای محدودیت ماشین‌های متناهی را که کمبود حافظهٔ آنها بود، با اضافه کردن یک پشته [2] نامحدود رفع می‌کنند.

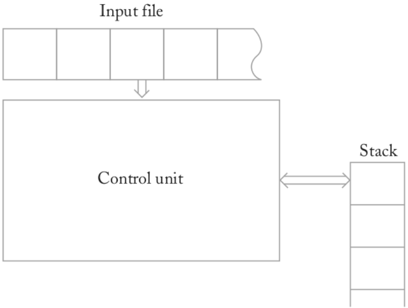
- برای مثال برای پذیرش زبان $L = \{ww^R : w \in \Sigma^*\}$ نیاز به ذخیرهٔ نمادهای رشتهٔ w داریم. از آنجایی که w نامحدود است، نیاز به یک حافظهٔ نامحدود داریم.

- ماشین‌های پشته‌ای نیز به دو صورت قطعی و غیرقطعی هستند. ماشین‌های پشته‌ای غیرقطعی زبان‌های مستقل از متن را پذیرش می‌کنند اما ماشین‌های پشته‌ای قطعی تنها زیرمجموعه‌ای از این زبان‌ها را می‌پذیرند.

[1] pushdown automata
[2] stack

ماشین‌های پشته‌ای

- شمای کلی ماشین پشته‌ای در زیر آمده است. در هر حرکت، ماشین یک نماد از ورودی و یک نماد از روی پشته می‌خواند. با توجه به نمادهای خوانده شده، ماشین حالت خود را تغییر می‌دهد و بر روی پشته می‌نویسد.



ماشین‌های پشته‌ای غیرقطعی

- پذیرندهٔ پشته‌ای غیرقطعی[1] (npda) به صورت یک هفت‌تایی $M = (Q, \Sigma, \Gamma, \delta, q_\circ, z, F)$ تعریف می‌شود، به طوری که :
- Q مجموعه‌ای است متناهی از حالات داخلی واحد کنترل
- Σ الفبای ورودی است
- Γ مجموعه‌ای متناهی از نمادهاست به نام الفبای پشته[2]
- $\delta : (Q \times (\Sigma \cup \{\lambda\}) \times \Gamma) \rightarrow (Q \times \Gamma^*)$ زیرمجموعه‌ای متناهی از) تابع گذار است.
- $q_\circ \in Q$ حالت آغازی واحد کنترل است
- $z \in \Gamma$ نماد آغازی پشته[3] است.
- $F \subseteq Q$ مجموعه‌ای از حالات پایانی است.

[1] nondeterministic pushdown accepter (npda)
[2] stack alphabet
[3] stack start symbol

ماشین‌های پشته‌ای غیرقطعی

- طبق تعریف ماشین پشته‌ای غیرقطعی، در هر حرکت با توجه به نماد ورودی، آخرین نماد بر روی پشته و حالت فعلی، حالت بعدی انتخاب می‌شود و یک رشته بر روی پشته نوشته می‌شود.
- نماد خوانده شده از ورودی می‌تواند تهی باشد که این یک گذار تهی [1] است.
- همچنین برای یک گذار، پشته نمی‌تواند خالی باشد.
- از آنجایی که این ماشین غیرقطعی است، پس در هر گذار، اگر چندین انتخاب وجود داشته باشد، یک کپی از ماشین به ازای هر انتخاب به اجرا ادامه می‌دهد و اگر یکی از کپی‌های ماشین، با اتمام خواندن رشته از ورودی، در یک حالت پایانی متوقف شد، رشته پذیرفته می‌شود.

[1] λ-transition

ماشین‌های پشته‌ای غیرقطعی

- فرض کنید یک قانون گذار در یک ماشین پشته‌ای غیرقطعی به صورت $\delta(q_1, a, b) = \{(q_2, cd), (q_3, \lambda)\}$ تعریف شده باشد.
- در صورتی که ماشین در حالت q_1 باشد و نماد a از ورودی خوانده شود و نماد b در بالای پشته باشد، آنگاه دو گذار ممکن خواهد بود.
- یا ماشین به حالت q_2 می‌رود و رشتهٔ cd در پشته نوشته می‌شود (نوشتن یک رشته نماد به نماد از راست به چپ رشته صورت می‌گیرد).
- یا ماشین به حالت q_3 می‌رود و چیزی در پشته نوشته نمی‌شود (در این صورت نماد b از بالای پشته حذف می‌شود).

ماشین‌های پشته‌ای غیرقطعی

- ماشین پشته‌ای غیرقطعی با $Q = \{q_0, q_1, q_2, q_3\}$، $\Sigma = \{a, b\}$، $\Gamma = \{0, 1\}$، $z = 0$، $F = \{q_3\}$ و حالت اولیهٔ q_0 و تابع گذار به صورت زیر را در نظر بگیرید.
- این ماشین چه زبانی را شناسایی می‌کند؟

$$\delta(q_0, a, 0) = \{(q_1, 10), (q_3, \lambda)\}$$
$$\delta(q_0, \lambda, 0) = \{(q_3, \lambda)\}$$
$$\delta(q_1, a, 1) = \{(q_1, 11)\}$$
$$\delta(q_1, b, 1) = \{(q_2, \lambda)\}$$
$$\delta(q_2, b, 1) = \{(q_2, \lambda)\}$$
$$\delta(q_2, \lambda, 0) = \{(q_3, \lambda)\}$$

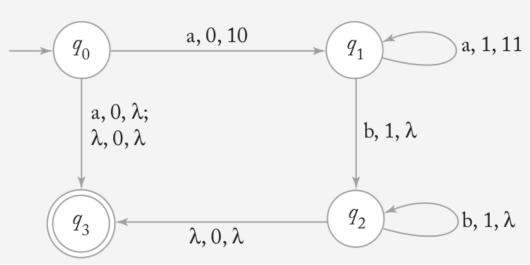
ماشین‌های پشته‌ای غیرقطعی

- دقت کنید که برخی از گذارها در این ماشین تعریف نشده‌اند، مثلا $\delta(q_0, b, 0)$. در صورتی که ماشین در این پیکربندی قرار بگیرد، به بن‌بست می‌خورد (به پیکربندی مرده می‌رود).
- در این ماشین در حالت q_1 با خواندن نماد a نماد ۱ به پشته اضافه می‌شود و با خواندن نماد b ماشین به حالت q_2 می‌رود و با خواندن نمادهای b متوالی در این حالت نماد ۱ از پشته حذف می‌شود.
- این ماشین تعداد نمادهای a و b را شمارش می‌کند و اگر تعداد این نمادها برابر باشد به حالت پایانی می‌رود.
- این ماشین زبان $L = \{a^n b^n : n \geq 0\} \cup \{a\}$ را می‌پذیرد.

ماشین‌های پشته‌ای غیرقطعی

- ماشین‌های پشته‌ای را می‌توانیم به صورت یک گراف گذار نیز نمایش دهیم.

- در اینصورت برچسب روی یال‌ها به صورت یک سه‌تایی a, b, c است که در آن a نماد خوانده شده از ورودی، b نماد برداشته شده از پشته، و c رشتهٔ نوشته شده بر روی پشته است.



ماشین‌های پشته‌ای غیرقطعی

- در هر لحظه، یک ماشین پشته‌ای در حالت q قرار دارد، رشتهٔ w از ورودی هنوز خوانده نشده است، و رشتهٔ u در پشته قرار دارد (نماد سمت چپ در رشتهٔ u در بالای پشته قرار دارد).
- این سه مقدار را به صورت یک سه‌تایی (q, w, u) نشان می‌دهیم و آن را توصیف لحظه‌ای [1] ماشین پشته‌ای می‌خوانیم.
- یک حرکت در ماشین پشته‌ای را با نماد $\vdash$ نشان می‌دهیم.
- بنابراین داریم $(q_۲, w, yx) \vdash (q_۱, aw, bx)$ اگر و تنها اگر $(q_۲, y) \in \delta(q_۱, a, b)$ باشد.

[1] instantaneous description

ماشین‌های پشته‌ای غیرقطعی

- اگر یک حرکت شامل چندین گام باشد آن را با $\overset{*}{\vdash}$ نشان می‌دهیم.
- بنابراین عبارت $(q_۱, w_۱, x_۱) \overset{*}{\vdash} (q_۲, w_۲, x_۲)$ بدین معناست که در حرکتی با چندگام ماشین می‌تواند از پیکربندی اول به پیکربندی دوم برسد.
- وقتی چندین ماشین را به طور همزمان بررسی می‌کنیم، می‌نویسیم $\vdash_M$ بدین معنا که حرکت در ماشین M مد نظر است.

ماشین‌های پشته‌ای غیرقطعی

- فرض کنید $M = (Q, \Sigma, \Gamma, \delta, q_\circ, z, F)$ یک ماشین پشته‌ای غیرقطعی باشد.
- زبانی که توسط M پذیرفته می‌شود $L(M)$ بدین صورت تعریف می‌شود:
$$L(M) = \{w \in \Sigma^* : (q_\circ, w, z) \overset{*}{\vdash}_M (p, \lambda, u), p \in F, u \in \Gamma^*\}$$
- به عبارت دیگر زبانی که توسط ماشین M پذیرفته می‌شود مجموعهٔ همهٔ رشته‌هایی است که ماشین M را در پایان رشته در یک حالت پایانی قرار دهد. محتوای پشتهٔ u در پایان خواندن رشته بی‌اهمیت است.

ماشین‌های پشته‌ای غیرقطعی

- یک npda برای زبان L طراحی کنید: $L = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$

ماشین‌های پشته‌ای غیرقطعی

- یک npda برای زبان L طراحی کنید: $L = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$

- یک راه حل ابتدایی: می‌توانیم با خواندن a یک نماد مانند نماد صفر به پشته اضافه کنیم و با خواندن b یک نماد صفر از بالای پشته حذف کنیم.

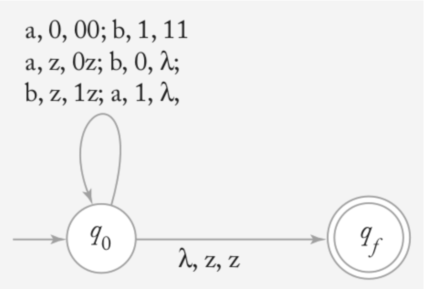
- مشکل این راه حل این است که اگر در ابتدا تعدادی b مشاهده کنیم پشته خالی می‌شود و ماشین متوقف می‌شود.

ماشین‌های پشته‌ای غیرقطعی

- یک npda برای زبان L طراحی کنید: $L = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$
- یک راه حل دیگر: می‌توانیم با خواندن b در صورتی که پشته خالی بود نماد ۱ را به ازای اعداد منفی به پشته اضافه کنیم.
- بنابراین با خواندن a اگر در محدودهٔ اعداد مثبت قرار داشتیم (نماد صفر در بالای پشته دیدیم)، نماد صفر را به پشته اضافه می‌کنیم. و اگر در محدودهٔ اعداد منفی قرار داشتیم (نماد ۱ در بالای پشته دیدیم)، نماد ۱ را از پشته حذف می‌کنیم.
- همچنین با خواندن b اگر در محدودهٔ اعداد مثبت قرار داشتیم (نماد صفر در بالای پشته دیدیم)، نماد صفر را از پشته حذف می‌کنیم. و اگر در محدودهٔ اعداد منفی قرار داشتیم (نماد ۱ در بالای پشته دیدیم)، نماد ۱ را به پشته اضافه می‌کنیم.

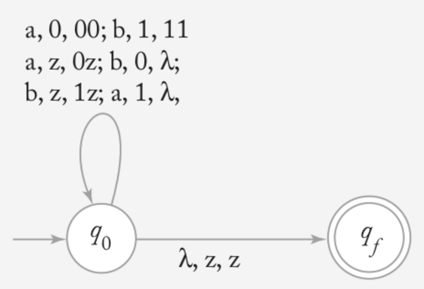
ماشین‌های پشته‌ای غیرقطعی

- یک npda برای زبان L طراحی کنید: $L = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$



ماشین‌های پشته‌ای غیرقطعی

- برای مثال برای رشتهٔ baab داریم:
- $(q_۰, baab, z) \vdash (q_۰, aab, ۱z) \vdash (q_۰, ab, z) \vdash (q_۰, b, ۰z) \vdash (q_۰, \lambda, z) \vdash (q_f, \lambda, z)$
- بنابراین رشته پذیرفته می‌شود.



ماشین‌های پشته‌ای غیرقطعی

- یک npda برای زبان L طراحی کنید: $L = \{ww^R : w \in \{a, b\}^+\}$

ماشین‌های پشته‌ای غیرقطعی

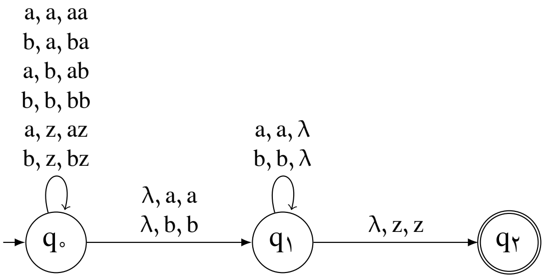
- یک npda برای زبان L طراحی کنید: $L = \{ww^R : w \in \{a,b\}^+\}$
- به ازای خواندن هر نماد از ورودی، آن نماد را در پشته ذخیره می‌کنیم. بعد از خواندن w برای بررسی w^R به ازای خواندن هر نماد باید آن نماد را با نماد بالای پشته مقایسه کنیم و در صورتی که دو نماد برابر بودند، نماد بالای پشته را حذف کنیم. اگر در پایان خواندن رشته به پشتهٔ خالی رسیدیم، رشته پذیرفته می‌شود.
- تنها مشکل این راه حل این است که نمی‌دانیم در کدام لحظه به وسط رشته رسیده‌ایم. ولی از آنجایی که ماشین غیرقطعی است، همهٔ مسیرهای ممکن برای رسیدن به یک حالت پایانی بررسی می‌شوند.

ماشین‌های پشته‌ای غیرقطعی

- یک npda برای زبان L طراحی کنید: $L = \{ww^R : w \in \{a, b\}^+\}$
- ماشین $M = (Q, \Sigma, \Gamma, \delta, q_۰, z, F)$ با $Q = \{q_۰, q_۱, q_۲\}$، $\Sigma = \{a, b\}$، $\Gamma = \{a, b, z\}$، $F = \{q_۲\}$ را در نظر می‌گیریم.
- برای خواندن قسمت w گذارهای زیر را تعریف می‌کنیم: $\delta(q_۰, a, a) = \{(q_۰, aa)\}$، $\delta(q_۰, b, a) = \{(q_۰, ba)\}$، $\delta(q_۰, a, b) = \{(q_۰, ab)\}$، $\delta(q_۰, b, b) = \{(q_۰, bb)\}$، $\delta(q_۰, a, z) = \{(q_۰, az)\}$، $\delta(q_۰, b, z) = \{(q_۰, bz)\}$
- برای حدس زدن وسط رشته و گذار از $q_۰$ به $q_۱$ گذارهای زیر را تعریف می‌کنیم: $\delta(q_۰, \lambda, a) = \{(q_۱, a)\}$، $\delta(q_۰, \lambda, b) = \{(q_۱, b)\}$
- برای مقایسه w^R در برابر محتوای پشته گذارهای زیر را تعریف می‌کنیم: $\delta(q_۱, a, a) = \{(q_۱, \lambda)\}$، $\delta(q_۱, b, b) = \{(q_۱, \lambda)\}$
- در نهایت برای پذیرفتن رشته گذار زیر را تعریف می‌کنیم: $\delta(q_۱, \lambda, z) = \{(q_۲, z)\}$

ماشین‌های پشته‌ای غیرقطعی

- یک npda برای زبان L طراحی کنید: $L = \{ww^R : w \in \{a, b\}^+\}$



ماشین‌های پشته‌ای غیرقطعی

- دنبالهٔ حرکت‌ها برای پذیرفتن رشتهٔ abba چنین است:
$(q_0, abba, z) \vdash (q_0, bba, az) \vdash (q_0, ba, baz) \vdash$
$(q_1, ba, baz) \vdash (q_1, a, az) \vdash (q_1, \lambda, z) \vdash (q_2, z)$

- در وسط رشته یعنی در جایی که توصیف لحظه‌ای ماشین (q_0, ba, baz) است، ماشین دو انتخاب برای حرکت خود دارد.

- یکی از انتخاب‌ها استفاده از گذار $\delta(q_0, b, b) = \{(q_0, bb)\}$ است که منجر به حرکت $(q_0, ba, baz) \vdash (q_0, a, bbaz)$ می‌شود.

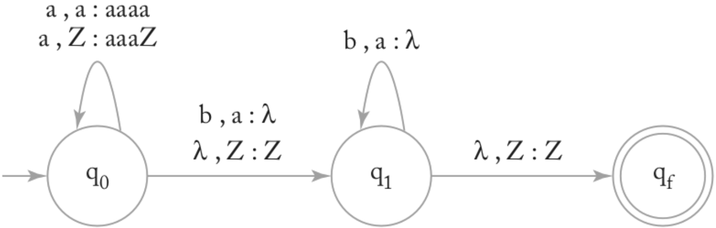
- انتخاب دوم استفاده از گذار $\delta(q_0, \lambda, b) = \{(q_1, b)\}$ است. این انتخاب منجر به پذیرفتن رشته می‌شود بنابراین ماشین این گذار را انتخاب می‌کند.

مثال

- یک ماشین پشته‌ای برای زبان $L = \{a^n b^{۳n} : n \geq ۰\}$ طراحی کنید.

مثال

- یک ماشین پشته‌ای برای زبان $L = \{a^n b^{۳n} : n \geq ۰\}$ طراحی کنید.

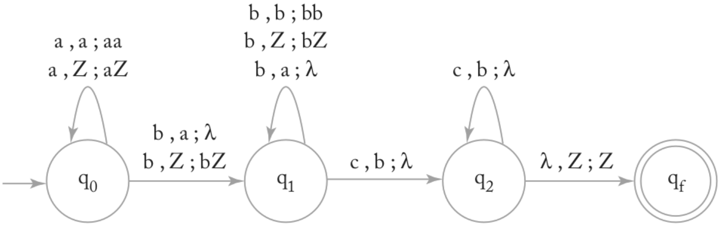


مثال

- یک ماشین پشته‌ای برای زبان $L = \{a^n b^{n+m} c^m : n \geq ۰, m \geq ۱\}$ طراحی کنید.

مثال

- یک ماشین پشته‌ای برای زبان $L = \{a^n b^{n+m} c^m : n \geq ۰, m \geq ۱\}$ طراحی کنید.



ماشین‌های پشته‌ای و زبان‌های مستقل از متن

- نشان می‌دهیم که برای هر زبان مستقل از متن یک ماشین پشته‌ای وجود دارد که آن را می‌پذیرد.
- برای سادگی اثبات فرض می‌کنیم که گرامر مستقل از متن به فرم گریباخ تبدیل شده است.
- به طور خلاصه، ماشین پشته‌ای، هر قانون گرامر به صورت $A \to ax$ را بدین گونه شبیه‌سازی می‌کند که با خواندن نماد پایانی سمت راست قانون $(a \in T)$ از ورودی و خواندن متغیر سمت چپ قانون $(A \in V)$ از پشته، متغیرهای سمت راست قانون $(x \in V^*)$ را در پشته ذخیره می‌کند.

ماشین‌های پشته‌ای و زبان‌های مستقل از متن

- یک ماشین پشته‌ای طراحی کنید که زبان تولید شده توسط یک گرامر با قوانین تولید $S \rightarrow aSbb|a$ را می‌پذیرد.

ماشین‌های پشته‌ای و زبان‌های مستقل از متن

- یک ماشین پشته‌ای طراحی کنید که زبان تولید شده توسط یک گرامر با قوانین تولید $S \to aSbb|a$ را می‌پذیرد.
- ابتدا گرامر را به فرم نرمال گریباخ تبدیل می‌کنیم، بنابراین قوانین $S \to aSA|a \ , \ A \to bB \ , \ B \to b$ را به دست می‌آوریم.
- ماشین پشته‌ای معادل آن سه حالت دارد: $\{q_0, q_1, q_2\}$ به طوری که حالت q_0 حالت آغازی و حالت q_2 یک حالت پایانی است.
- در حالت اولیه متغیر آغازی را به پشته اضافه می‌کنیم و به حالت q_1 گذار می‌کنیم:
$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$

ماشین‌های پشته‌ای و زبان‌های مستقل از متن

- قانون $S \rightarrow aSA|a$ را بدین صورت شبیه‌سازی می‌کنیم که با خواندن a از ورودی و خواندن S از پشته یا SA را به پشته اضافه می‌کنیم و یا به پشته چیزی اضافه نمی‌کنیم، بنابراین داریم:
$$\delta(q_۱, a, S) = \{(q_۱, SA), (q_۱, \lambda)\}$$
- همچنین برای قوانین دیگر گرامر داریم: $\delta(q_۱, b, A) = \{(q_۱, B)\}$ ، $\delta(q_۱, b, B) = \{(q_۱, \lambda)\}$
- اگر در ورودی نمادی باقی نماند و پشته خالی شود می‌توانیم گذار به حالت پایانی را انجام دهیم:
$$\delta(q_۱, \lambda, z) = \{(q_۲, \lambda)\}$$
- این الگوریتم را به حالت کلی تعمیم می‌دهیم.

ماشین‌های پشته‌ای و زبان‌های مستقل از متن

- برای هر زبان مستقل از متن L یک ماشین پشته‌ای غیرقطعی M وجود دارد به طوری که $\mathrm{L} = \mathrm{L}(\mathrm{M})$.
- اگر L یک زبان مستقل از متن بدون رشتهٔ تهی باشد، آنگاه یک گرامر مستقل از متن در فرم نرمال گریباخ برای آن وجود دارد.
- فرض کنید این گرامر $\mathrm{G} = (\mathrm{V}, \mathrm{T}, \mathrm{S}, \mathrm{P})$ باشد.
- می‌توانیم یک ماشین پشته‌ای طراحی کنیم که اشتقاق‌های چپ این گرامر را شبیه‌سازی کند.
- ماشین پشته‌ای $\mathrm{M} = (\{q_0, q_1, q_f\}, \mathrm{T}, \mathrm{V} \cup \{z\}, \delta, q_0, z, \{q_f\})$ را در نظر بگیرید، به طوری که $z \notin \mathrm{V}$.
- در این ماشین الفبای ورودی برابر با مجموعهٔ نمادهای پایانی گرامر G و الفبای پشته مجموعهٔ متغیرهای گرامر است.

ماشین‌های پشته‌ای و زبان‌های مستقل از متن

- تابع گذار را به ازای مقادیر اولیه تعریف می‌کنیم: $\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$

- بنابراین در حرکت اول، متغیر آغازی S را به پشته اضافه می‌کنیم. از نماد z برای تشخیص دادن خالی شدن پشته و پایان فرایند اشتقاق استفاده می‌کنیم.

- همچنین به ازای هر یک از قوانین $A \to au$ در گرامر، تابع گذار را بدین صورت تعریف می‌کنیم: $(q_1, u) \in \delta(q_1, a, A)$.

- برای شبیه‌سازی هر اشتقاق که A را به au تبدیل می‌کند، در ماشین a را از ورودی خوانده و متغیر A را از پشته حذف و متغیرهای u را به پشته اضافه می‌کنیم.

- در پایان در صورتی که رشته به پایان برسد، و پشته خالی از متغیرهای گرامر شود، به حالت پایانی گذار می‌کنیم: $\delta(q_1, \lambda, z) = \{(q_f, z)\}$

ماشین‌های پشته‌ای و زبان‌های مستقل از متن

- می‌توان نشان داد که برای هر اشتقاق در فرایند اشتقاق یک گرامر، یک حرکت متناظر در یک ماشین پشته‌ای غیرقطعی وجود دارد، و بنابراین هر جملهٔ w که از گرامر G به دست می‌آید را می‌توان توسط ماشین پشته‌ای متناظر آن پذیرفت.
- همینطور به ازای هر حرکت در ماشین پشته‌ای غیر قطعی M ، یک اشتقاق در گرامر G که ماشین M با قوانین تولید آن ساخته شده است وجود دارد، و بنابراین هر جملهٔ w که توسط ماشین M پذیرفته می‌شود را می‌توان توسط گرامر G به دست آورد.

- برای گرامر $S \to aA \;\;,\;\; A \to aABC|bB|a \;\;,\;\; B \to b \;\;,\;\; C \to c$ یک ماشین پشته‌ای بسازید.

- برای گرامر $S \to aA \quad , \quad A \to aABC|bB|a \quad , \quad B \to b \quad , \quad C \to c$ یک ماشین پشته‌ای بسازید.
- برای تابع گذار (گذار از حالت آغازی و گذار به حالت پایانی) چنین تعریف می‌کنیم:

$$\delta(q_۰, \lambda, z) = \{(q_۱, Sz)\}$$
$$\delta(q_۱, \lambda, z) = \{(q_f, z)\}$$

- سپس برای قوانین تولید تابع گذار را چنین تعریف می‌کنیم:

$$\delta(q_۱, a, S) = \{(q_۱, A)\}$$
$$\delta(q_۱, a, A) = \{(q_۱, ABC), (q_۱, \lambda)\}$$
$$\delta(q_۱, b, A) = \{(q_۱, B)\}$$
$$\delta(q_۱, b, B) = \{(q_۱, \lambda)\}$$
$$\delta(q_۱, c, C) = \{(q_۱, \lambda)\}$$

ماشین‌های پشته‌ای و زبان‌های مستقل از متن

- برای پذیرفتن رشتهٔ aaabc این ماشین حرکت‌های زیر را انجام می‌دهد:
$$(q_0, aaabc, z) \vdash (q_1, aaabc, Sz) \vdash (q_1, aabc, Az) \vdash (q_1, abc, ABCz)$$
$$\vdash (q_1, bc, BCz) \vdash (q_1, c, Cz) \vdash (q_1, \lambda, z) \vdash (q_f, \lambda, z)$$

- این جمله با استفاده از فرایند اشتقاق زیر مشتق می‌شود:
$$S \Rightarrow aA \Rightarrow aaABC \Rightarrow aaaBC \Rightarrow aaabC \Rightarrow aaabc$$

ماشین‌های پشته‌ای و زبان‌های مستقل از متن

- همچنین روشی برای تبدیل یک ماشین پشته‌ای به یک گرامر مستقل از متن وجود دارد که در اینجا به آن نمی‌پردازیم.
- پس برای هر گرامر مستقل از متن یک ماشین پشته‌ای و برای هر ماشین پشته‌ای یک گرامر مستقل از متن وجود دارد.

ماشین‌های پشته‌ای قطعی

- یک پذیرندهٔ پشته‌ای قطعی [1] (dpda) بر خلاف پذیرندهٔ پشته‌ای غیرقطعی هیچ‌گاه حق انتخاب ندارد.
- ماشین پشته‌ای $M = (Q, \Sigma, \Gamma, \delta, q_\circ, z, F)$ قطعی گفته می‌شود اگر تابع گذار آن نسبت به ماشین پشته‌ای غیرقطعی محدودیت‌های زیر را داشته باشد:

۱. $\delta(q, a, b)$ حداکثر یک عضو داشته باشد.

۲. اگر $\delta(q, \lambda, b)$ تهی نباشد، آنگاه $\delta(q, c, b)$ باید به ازای همهٔ مقادیر $c \in \Sigma$ تهی باشد.

- به طوری که $q \in Q, a \in \Sigma \cup \{\lambda\}, b \in \Gamma$

[1] deterministic pushdown accepter (dpda)

ماشین‌های پشته‌ای قطعی

- اولین محدودیت باعث می‌شود در هر حالت با خواندن هر یک از نمادهای الفبا و خواندن هر یک از نمادهای پشته ماشین فقط بتواند به حداکثر یک حالت برود. گرچه ماشین ممکن است به بن‌بست نیز برخورد کند.
- دومین محدودیت باعث می‌شود وقتی گذار تهی برای یک پیکربندی امکان‌پذیر است، هیچ گذار دیگری برای آن پیکربندی با خواندن هیچ نماد دیگری امکان‌پذیر نباشد.
- پس گرچه گذار تهی نیز وجود دارد و ماشین ممکن است به بن‌بست برخورد کند، اما در هر پیکربندی فقط یک گذار ممکن وجود دارد.
- زبان L یک زبان مستقل از متن قطعی [1] گفته می‌شود اگر توسط یک ماشین پشته‌ای قطعی M پذیرفته شود به طوری که $L = L(M)$.

[1] deterministic context-free language

ماشین‌های پشته‌ای قطعی

- زبان $L = \{a^n b^n : n \geq ۰\}$ یک زبان مستقل از متن قطعی است.
- زیرا ماشین پشته‌ای قطعی $M = (\{q_۰, q_۱, q_۲\}, a, b, ۰, ۱, \delta, q_۰, ۰, \{q_۰\})$ با گذارهای زیر وجود دارد که آن را می‌پذیرد.

$$\delta(q_۰, a, ۰) = \{(q_۱, ۱۰)\}$$
$$\delta(q_۱, a, ۱) = \{(q_۱, ۱۱)\}$$
$$\delta(q_۱, b, ۱) = \{(q_۲, \lambda)\}$$
$$\delta(q_۲, b, ۱) = \{(q_۲, \lambda)\}$$
$$\delta(q_۲, \lambda, ۰) = \{(q_۰, \lambda)\}$$

ماشین‌های پشته‌ای قطعی

- زبان $\{ww^R : w \in \Sigma^*\}$ یک زبان مستقل از متن است ولی یک زبان مستقل از متن قطعی نیست، زیرا هیچ ماشین مستقل از متن قطعی برای آن وجود ندارد.
- دلیل آن این است که برای تشخیص وسط رشته به عدم قطعیت نیاز داریم.
- زبان‌های مستقل از متن قطعی با گرامرهای مستقل از متن قطعی تولید می‌شوند و اهمیت این گرامرها در این است که تجزیه را در زمان چندجمله‌ای $O(n)$ به ازای جملات با طول n انجام می‌دهند.
- در فرایند تجزیه جملات با استفاده از گرامر مستقل از متن قطعی (همانند گرامر ساده)، همیشه برای یک اشتقاق تنها یک انتخاب وجود دارد.

لم تزریق برای زبان‌های مستقل از متن

- از لم تزریق برای زبان‌های مستقل از متن استفاده می‌کنیم برای اینکه نشان دهیم که یک زبان مستقل از متن نیست.
- نشان می‌دهیم که اگر یک زبان مستقل از متن باشد، آنگاه می‌توان هر رشتهٔ به اندازهٔ کافی طولانی از آن زبان را به پنج قسمت تقسیم کرد به طوری که از تکرار (پمپاژ) قسمت دوم و چهارم رشته‌ای به دست آید که در همان زبان است.

لم تزریق برای زبان‌های مستقل از متن

- فرض کنید L یک زبان نامحدود مستقل از متن باشد. آنگاه یک عدد صحیح مثبت m وجود دارد به طوری که هر جملهٔ $w \in L$ با طول $|w| \geq m$ می‌تواند به پنج قسمت تقسیم شود $w = uvxyz$ به طوری که $|vxy| \leq m$، $|vy| \geq ۱$ و همچنین $uv^ixy^iz \in L$ به ازای همهٔ مقادیر $i = ۰, ۱, ۲, \cdots$

لم تزریق برای زبان‌های مستقل از متن

- فرض کنید L یک زبان نامحدود مستقل از متن باشد. آنگاه یک عدد صحیح مثبت m وجود دارد به طوری که هر جملهٔ $w \in L$ با طول $|w| \geq m$ می‌تواند به پنج قسمت تقسیم شود $w = uvxyz$ به طوری که $|vxy| \leq m$، $|vy| \geq ۱$ و همچنین $uv^ixy^iz \in L$ به ازای همهٔ مقادیر $i = ۰, ۱, ۲, \cdots$

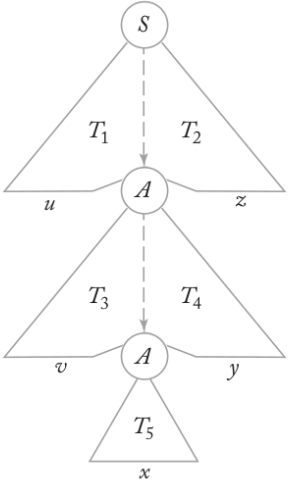
- از آنجایی که L نامحدود است، به ازای جملات طولانی، فرایندهای اشتقاق بسیار طولانی می‌تواند وجود داشته باشد که ارتفاع درخت اشتقاق آنها نیز بسیار زیاد است.

- حال یکی از این درخت‌های اشتقاق مرتفع را به همراه یک مسیر طولانی از ریشه تا یکی از برگ‌ها (به طوری که طول مسیر از تعداد متغیرهای گرامر بیشتر است) را در نظر بگیرید.

- از آنجایی که تعداد متغیرهای این گرامر محدود است، برخی از متغیرها باید در این مسیر تکرار شده باشند.

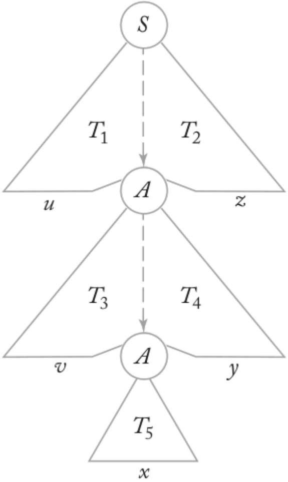
لم تزریق برای زبان‌های مستقل از متن

یک درخت اشتقاق مرتفع با تکرار متغیر A در زیر نشان داده شده است.



لم تزریق برای زبان‌های مستقل از متن

فرایند اشتقاق برای محصول این درخت بدین صورت است: $S \overset{*}{\Rightarrow} uAz \overset{*}{\Rightarrow} uvAyz \overset{*}{\Rightarrow} uvxyz$



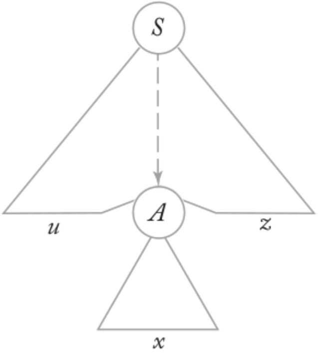
لم تزریق برای زبان‌های مستقل از متن

- پس فرایند اشتقاق برای جملهٔ uvxyz که محصول این درخت اشتقاق مرتفع است بدین صورت است: $S \overset{*}{\Rightarrow} uAz \overset{*}{\Rightarrow} uvAyz \overset{*}{\Rightarrow} uvxyz$ به طوری که u،v،x،y،z شامل تنها نمادهای پایانی هستند.

- از آنجایی که دو اشتقاق $A \overset{*}{\Rightarrow} vAy$ و $A \overset{*}{\Rightarrow} x$ در فرایند اشتقاق این گرامر امکان‌پذیر هستند، لذا می‌توان در اولین بار مشاهدهٔ متغیر A در فرایند اشتقاق، به جای $A \overset{*}{\Rightarrow} vAy$ از $A \overset{*}{\Rightarrow} x$ استفاده کرد و جملهٔ uxz را به دست آورد. همچنین در هر بار مشاهدهٔ متغیر A می‌توان برای i بار از اشتقاق $A \overset{*}{\Rightarrow} vAy$ استفاده کرد و جملهٔ uv^ixy^iz را به دست آورد.

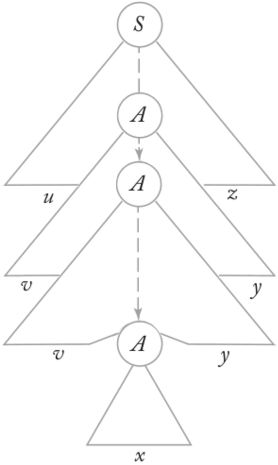
لم تزریق برای زبان‌های مستقل از متن

می‌توان در اولین بار مشاهدهٔ متغیر A در فرایند اشتقاق، به جای $\mathrm{A} \overset{*}{\Rightarrow} \mathrm{vAy}$ از $\mathrm{A} \overset{*}{\Rightarrow} \mathrm{x}$ استفاده کرد و جملهٔ uxz را به دست آورد.



لم تزریق برای زبان‌های مستقل از متن

همچنین در هر بار مشاهدهٔ متغیر A می‌توان برای i بار از اشتقاق $\mathrm{A} \overset{*}{\Rightarrow} \mathrm{vAy}$ استفاده کرد و جملهٔ $\mathrm{uv^i xy^i z}$ را به دست آورد.



لم تزریق برای زبان‌های مستقل از متن

- حال می‌خواهیم مقدار m را پیدا کنیم.
- باید تعیین کنیم برای رشته‌هایی با چه طولی حداقل یک متغیر در درخت اشتقاق تکرار می‌شود.
- در حالت کلی برای هر گرامر مستقل از متن داده شده، مقدار دقیق m به قوانین تولید گرامر بستگی دارد، اما می‌توانیم با استفاده از گرامر مستقل از متن در فرم نرمال چامسکی یک تقریب بالا برای مقدار m پیدا کنیم. از آنجایی که درخت اشتقاق در فرم نرمال چامسکی یک درخت دودویی است، استدلال بر روی این درخت ساده‌تر است.

لم تزریق برای زبان‌های مستقل از متن

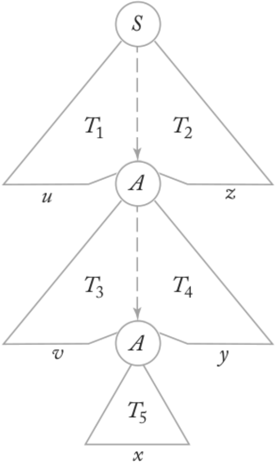
- فرض کنید گرامر G را به فرم نرمال چامسکی تبدیل می‌کنیم. در اینصورت درخت اشتقاق یک درخت دودویی است.

- اگر ارتفاع این درخت برابر باشد با تعداد متغیرهای گرامر، یعنی $|V|$، آنگاه حداقل یک مسیر از ریشه تا برگ با $|V| + ۱$ رأس وجود دارد، ولی از آنجایی که رأس آخر، یعنی برگ درخت، یک نماد پایانی است، بنابراین تعداد $|V|$ متغیر در یکی از مسیرها وجود دارد. طول جملات چنین درختی حداکثر برابر است با $۲^{|V|-۱}$ است (دقت کنید که در سطح آخر درخت یعنی جایی که متغیرها به نمادهای پایانی تبدیل می‌شوند، هر رأس تنها یک فرزند دارد).

- حال فرض کنید $m = ۲^{|V|}$. در اینصورت ارتفاع درخت اشتقاق برای جملاتی با طول برابر یا بیشتر از m باید حداقل $|V| + ۱$ باشد و بنابراین حداقل $|V| + ۲$ رأس در یکی از مسیرهای آن وجود دارد. آخرین رأس در این مسیر یک نماد پایانی است، پس حداقل $|V| + ۱$ متغیر در این مسیر وجود دارد و طبق اصل لانه کبوتری حداقل یکی از متغیرها در این مسیر تکرار شده است.

لم تزریق برای زبان‌های مستقل از متن

- از آنجایی که در این مسیر حداقل یک متغیر تکرار شده است، می توانیم رشته را طبق شکل زیر به پنج قسمت uvxyz تقسیم کنیم.



لم تزریق برای زبان‌های مستقل از متن

- همانطور که اشاره شد، از آنجایی که دو اشتقاق $A \overset{*}{\Rightarrow} vAy$ و $A \overset{*}{\Rightarrow} x$ در فرایند اشتقاق این گرامر امکان‌پذیر هستند:

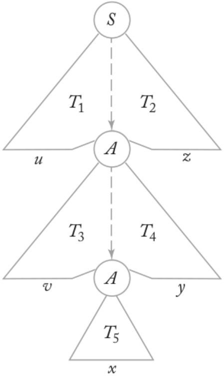
۱. می‌توان در اولین بار مشاهدهٔ متغیر A در فرایند اشتقاق، به جای $A \overset{*}{\Rightarrow} vAy$ از $A \overset{*}{\Rightarrow} x$ استفاده کرد و جملهٔ uxz را به دست آورد.

۲. همچنین در هر بار مشاهدهٔ متغیر A می‌توان برای i بار از اشتقاق $A \overset{*}{\Rightarrow} vAy$ استفاده کرد و جملهٔ uv^ixy^iz را به دست آورد.

لم تزریق برای زبان‌های مستقل از متن

- حال باید اطمینان حاصل کنیم که زیر رشته‌های v و y به طور همزمان نمی‌توانند تهی باشند.

- از آنجایی که G در فرم نرمال چامسکی است، لذا هیچ قانون تهی و یکه‌ای در آن وجود ندارد و بنابراین v و y نمی‌توانند به طور همزمان تهی باشند، بنابراین داریم $|vy| \geq ۱$.



لم تزریق برای زبان‌های مستقل از متن

- حال فرض کنیم متغیر A پایین‌ترین متغیر تکرار شونده در مسیری است که در نظر گرفته‌ایم و در مسیر ریشه تا برگ‌ها هیچ متغیر تکرار شونده‌ای پایین‌تر از A وجود ندارد.
- به عبارت دیگر، می‌توانیم فرض کنیم که در زیردرخت $T_۵$ هیچ متغیری تکرار نشده است. اگر متغیری در این زیر درخت تکرار شده بود می‌توانستیم آن متغیر را به جای متغیر A به عنوان متغیر تکرار شونده در نظر بگیریم.
- همچنین به طور مشابه می‌توانیم فرض کنیم که در زیر درخت‌های $T_۳$ و $T_۴$ هیچ متغیری تکرار نشده است.
- از آنجایی که در این زیردرخت‌ها هیچ متغیری تکرار نشده است، پس حداکثر طول رشتهٔ vxy را می‌توان با محاسبهٔ بلندترین طول مسیر ممکن از برگ‌ها تا دومین تکرار متغیر A محاسبه کرد. این مسیر حداکثر $|V| + ۱$ متغیر دارد، و آخرین رأس در مسیر نماد پایانی است پس ارتفاع آن $|V| + ۱$ است، پس طول vxy حداکثر برابر است با $m = ۲^{|V|}$. بنابراین خواهیم داشت $|vxy| \leq m$.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{a^n b^n c^n : n \geq 0\}$ مستقل از متن نیست.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{a^n b^n c^n : n \geq ۱\}$ مستقل از متن نیست.

- فرض می‌کنیم زبان L مستقل از متن باشد، پس لم تزریق باید برای آن برقرار باشد.

- به ازای m داده شده، جملهٔ $a^m b^m c^m$ را در نظر می‌گیریم.

- اگر زیر رشتهٔ vxy به نحوی انتخاب شود که فقط شامل نمادهای a باشد، آنگاه با پمپاژ کردن رشته به هر مقداری، رشتهٔ $a^k b^m c^m$ با $k \neq m$ را به دست می‌آوریم که در L نیست.

- اگر زیررشتهٔ vxy به نحوی انتخاب شود که شامل تعدادی مساوی a و b باشد، آنگاه با پمپاژ کردن، رشتهٔ $a^k b^k c^m$ با $k \neq m$ به دست می‌آید که آن هم در زبان L نیست.

- بدین علت که طول $|vxy| \leq m$ پس نمی‌توانیم رشتهٔ vxy را طوری انتخاب کنیم که شامل نمادهای a و b و c شود.

- پس در هر صورت رشتهٔ به دست آمده بعد از پمپاژ در زبان L نیست و بنابراین فرض اولیه مبنی بر مستقل از متن بودن زبان نادرست بوده است و زبان L مستقل از متن نیست.

لم تزریق برای زبان‌های مستقل از متن

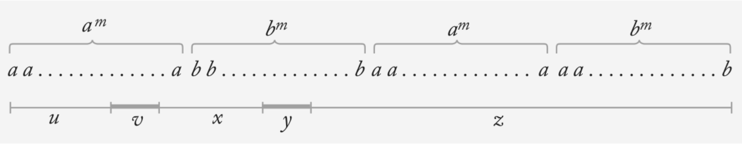
- حال فرض کنید می‌خواهیم با لم تزریق ثابت کنیم که زبان $L = \{a^n b^n\}$ مستقل از متن نیست (گرچه این زبان مستقل از متن است).
- فرض کنیم L مستقل از متن باشد، پس لم تزریق باید برای آن برقرار باشد. اگر رشته $a^m b^m$ را در نظر بگیریم زیر رشته vxy را می‌توان به نحوی انتخاب کرد که با پمپاژ کردن آن رشته‌ای به دست می‌آید که در زبان L است. می‌توان این زیر رشته را به طوری انتخاب کنیم که در آن تعداد a و b برابر باشد. در اینصورت همیشه با پمپاژ کردن آن رشته‌ای به دست می‌آید که در زبان L است. پس به تناقض نمی‌رسیم.
- گرچه به تناقض نمی‌رسیم ولی نمی‌توانیم نشان دهیم که L مستقل از متن است، زیرا لم تزریق یک شرط لازم است و کافی نیست. باید از روش دیگری استفاده کنیم تا نشان دهیم L مستقل از متن است، برای مثال یک گرامر مستقل از متن برای آن پیدا کنیم.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{ww : w \in \{a, b\}^*\}$ مستقل از متن نیست.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{ww : w \in \{a, b\}^*\}$ مستقل از متن نیست.
- فرض کنیم L مستقل از متن باشد، آنگاه باید لم تزریق برای آن برقرار باشد. به ازای m داده شده رشتهٔ $a^m b^m a^m b^m$ را در نظر می‌گیریم.
- زیر رشتهٔ vxy به هر نحوی که انتخاب شود، رشتهٔ به دست آمده با در نظر گرفتن $i = ۰$ در زبان L نیست.
- برای مثال اگر vxy را به صورت زیر انتخاب کنیم، با در نظر گرفتن $i = ۰$ داریم: $a^k b^j a^m b^m$ به طوری که $k < m, j < m$



لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{a^{n!} : n > ۲\}$ مستقل از متن نیست.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{a^{n!} : n > ۲\}$ مستقل از متن نیست.
- فرض می‌کنیم زبان L مستقل از متن باشد، پس لم تزریق باید برای آن برقرار باشد و می‌توان رشتهٔ $a^{m!}$ را به پنج قسمت $uvxyz$ تقسیم کرد.
- به ازای هر تقسیم بندی خواهیم داشت: $v = a^p$ و $y = a^q$
- در این صورت به ازای $i = ۰$ طول رشتهٔ uxz برابر است با $m! - (p+q)$
- این رشته در زبان L است تنها اگر $m! - (p+q) = j!$
- اما از آنجایی که $p + q \le m$ بنابراین پس $m! - (p+q) > (m-۱)!$ ، زیرا $m! - (m-۱)! = (m-۱)!(m-۱) > m \ge p+q$.
- بنابراین فرض اولیه نادرست بوده و زبان L نمی‌تواند مستقل از متن باشد.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{a^n b^j : n = j^۲\}$ مستقل از متن نیست.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{a^n b^j : n = j^۲\}$ مستقل از متن نیست.
- به ازای m داده شده، رشتهٔ $a^{m^۲} b^m$ را در نظر می‌گیریم.
- اگر زیررشتهٔ vxy با $|vy| = k$ به طور کامل در نمادهای a باشد، آنگاه بدیهی است که به ازای $i = ۰$ داریم $m^۲ - k < m^۲$ و اگر این زیررشته به طور کامل در نمادهای b باشد، به ازای $i = ۰$ داریم $m^۲ > (m-k)^۲$ و بنابراین رشتهٔ پمپاژ شده در زبان L نیست.
- حال اگر رشتهٔ v شامل تعداد $k_۱$ نماد a و رشتهٔ y شامل تعداد $k_۲$ نماد b باشد، آنگاه به ازای $i = ۰$ تعداد $m^۲ - k_۱$ نماد a و تعداد $m - k_۲$ نماد b به دست می‌آید. اما داریم:
$(m - k_۲)^۲ \leq (m-۱)^۲ = m^۲ - ۲m + ۱ < m^۲ - k_۱$ زیرا می‌دانیم $k_۱ < m$ و $k_۲ \geq ۱$
- پس رشتهٔ پمپاژ شده در L نیست و L نمی‌تواند مستقل از متن باشد.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{a^i b^j c^k : k = \max(i, j)\}$ مستقل از متن نیست.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{a^i b^j c^k : k = \max(i, j)\}$ مستقل از متن نیست.
- فرض می‌کنیم زبان L مستقل از متن است و به ازای m داده شده، رشتهٔ $a^m b^m c^m$ را در نظر می‌گیریم.
- چند حالت برای رشتهٔ vxy وجود دارد. یا این رشته حاوی نماد c است که در این صورت با در نظر گرفتن $i = ۰$ تعداد a بیشتر از c می‌شود و رشته‌ای به دست می‌آید که در L نیست. یا رشتهٔ vxy حاوی نماد c نیست که در این صورت با در نظر گرفتن $i = ۲$ رشته‌ای به دست می‌آید که تعداد نمادهای a یا نمادهای b یا تعداد هر دو نماد a و b در آن بیشتر از نماد c است و رشتهٔ به دست آمده در L نیست.
- پس فرض اولیه مبنی بر مستقل از متن بودن زبان نادرست بوده است.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{w \in \{a, b, c\}^* : n_a(w) < n_b(w) \wedge n_a(w) < n_c(w)\}$ مستقل از متن نیست.

لم تزریق برای زبان‌های مستقل از متن

- نشان دهید زبان $L = \{w \in \{a, b, c\}^* : n_a(w) < n_b(w) \wedge n_a(w) < n_c(w)\}$ مستقل از متن نیست.
- فرض می‌کنیم زبان L مستقل از متن است و به ازای m داده شده، رشتهٔ $a^m b^{m+۱} c^{m+۱}$ را در نظر می‌گیریم.
- چند حالت برای رشتهٔ vxy وجود دارد. یا این رشته حاوی نماد a است که در این صورت با در نظر گرفتن $i = ۲$ تعداد a بیشتر از c می‌شود و رشته‌ای به دست می‌آید که در L نیست. یا رشتهٔ vxy حاوی نماد a نیست که در این صورت با در نظر گرفتن $i = ۰$ رشته‌ای به دست می‌آید که تعداد نمادهای b یا نمادهای c یا تعداد هر دو نماد b و c در آن کمتر از نماد a است و رشتهٔ به دست آمده در L نیست.
- پس فرض اولیه مبنی بر مستقل از متن بودن زبان نادرست بوده است.

ویژگی‌های بستاری زبان‌های مستقل از متن

- مجموعهٔ زبان‌های مستقل از متن بر روی عملگرهای اجتماع، الحاق، و بستار–ستاره بسته، اما بر روی اشتراک و متمم بسته نیست.

ویژگی‌های بستاری زبان‌های مستقل از متن

- مجموعهٔ زبان‌های مستقل از متن بر روی عملگرهای اجتماع، الحاق، و بستار–ستاره بسته است.

- فرض کنید L_1 و L_2 دو زبان مستقل از متن باشند که به ترتیب با گرامرهای $G_1 = (V_1, T_1, S_1, P_1)$ و $G_2 = (V_2, T_2, S_2, P_2)$ تولید می‌شوند. فرض می‌کنیم V_1 و V_2 دو مجموعهٔ مجزا هستند.

- زبان $L(G_3)$ که توسط گرامر $G_3 = (V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, S_3, P_3)$ تولید می‌شود را در نظر بگیرید.

- فرض می‌کنیم S_3 عضو V_1 و V_2 نیست و قوانین تولید P_3 را به صورت $P_3 = P_1 \cup P_2 \cup \{S_3 \to S_1|S_2\}$ در نظر می‌گیریم.

ویژگی‌های بستاری زبان‌های مستقل از متن

- می‌توانیم نشان دهیم که $L(G_3) = L_1 \cup L_2$.

- به ازای هر $w \in L_1$ می‌توانیم فرایند اشتقاق $S_3 \Rightarrow S_1 \overset{*}{\Rightarrow} w$ را بنویسیم. همین استدلال را در مورد $w \in L_2$ نیز به کار می‌بریم.

- همچنین به ازای هر $w \in L(G_3)$ اولین مرحله در فرایند اشتقاق $S_3 \Rightarrow S_1$ یا $S_3 \Rightarrow S_2$ است. بدین ترتیب w یا در L_1 است و یا در L_2 .

- پس گرامری پیدا کردیم برای اجتماع دو زبان مستقل از متن و بنابراین اجتماع دو زبان مستقل از متن یک زبان مستقل از متن است.

ویژگی‌های بستاری زبان‌های مستقل از متن

- با استدلالی مشابه، گرامر G_4 را با قوانین تولید $P_4 = P_1 \cup P_2 \cup \{S_4 \to S_1S_2\}$ و نشان می‌دهیم $L(G_4) = L_1L_2$.
- پس گرامری پیدا کردیم برای الحاق دو زبان مستقل از متن و بنابراین الحاق دو زبان مستقل از متن یک زبان مستقل از متن است.
- در نهایت با استدلالی مشابه استدلال اجتماع، گرامر G_5 را با قوانین تولید $P_5 = P_1 \cup \{S_5 \to S_1S_5|\lambda\}$ و نشان می‌دهیم $L(G_5) = L_1^*$.
- پس گرامری پیدا کردیم برای بستار-ستاره بر روی یک زبان مستقل از متن و بنابراین بستار-ستارهٔ یک زبان مستقل از متن، یک زبان مستقل از متن است.

ویژگی‌های بستاری زبان‌های مستقل از متن

- مجموعهٔ زبان‌های مستقل از متن بر روی اشتراک و متمم بسته نیست.
- اثبات: از برهان خلف و یک مثال نقض استفاده می‌کنیم، یعنی فرض می‌کنیم مجموعهٔ زبان‌های مستقل از متن بر روی اشتراک بسته باشد. آنگاه دو زبان مستقل از متن را انتخاب می‌کنیم و نشان می‌دهیم که اشتراک آنها زبانی است که مستقل از متن نیست. پس فرض اولیه نادرست بوده و اشتراک دو زبان مستقل از متن همیشه یک زبان مستقل از متن نیست.
- دو زبان $\{a^n b^m c^m : n, m \geq ۰\}$ و $L_۱ = \{a^n b^n c^m : n, m \geq ۰\}$ $L_۲ =$ را در نظر می‌گیریم که هر دو مستقل از متن هستند زیرا یک گرامر مستقل از متن برای هر یک از آنها وجود دارد.
- می‌دانیم $L_۱ \cap L_۲ = \{a^n b^n c^n : n \geq ۰\}$ که توسط لم تزریق نشان دادیم مستقل از متن نیست. پس اشتراک دو زبان مستقل از متن مستقل از متن نیست.

ویژگی‌های بستاری زبان‌های مستقل از متن

- برای اثبات بسته نبودن زبان‌های مستقل از متن بر روی متمم از قانون دمورگان استفاده می‌کنیم:
$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

- اگر زبان‌های مستقل از متن بر روی متمم بسته بودند، آنگاه سمت راست عبارت بالا همیشه یک زبان مستقل از متن به دست می‌داد. اما نشان دادیم که سمت چپ عبارت بالا، یعنی اشتراک دو زبان مستقل از متن می‌تواند مستقل از متن نباشد.

- پس مجموعهٔ زبان‌های مستقل از متن بر روی عملگر متمم بسته نیست.

ویژگی بستاری

- فرض کنید $L_۱$ یک زبان مستقل از متن و $L_۲$ یک زبان منظم باشد. آنگاه $L_۱ \cap L_۲$ یک زبان مستقل از متن است.

ویژگی بستاری

- فرض کنید $L_۱$ یک زبان مستقل از متن و $L_۲$ یک زبان منظم باشد. آنگاه $L_۱ \cap L_۲$ یک زبان مستقل از متن است.

- فرض کنید $M_۱ = (Q, \Sigma, \Gamma, \delta_۱, q_۰, z, F_۱)$ یک ماشین پشته‌ای غیرقطعی باشد که زبان $L_۱$ را می‌پذیرد و $M_۲ = (P, \Sigma, \delta_۲, p_۰, F_۲)$ یک ماشین متناهی قطعی باشد که زبان $L_۲$ را می‌پذیرد.

- یک ماشین پشته‌ای $\widehat{M} = (\widehat{Q}, \Sigma, \Gamma, \widehat{\delta}, \widehat{q_۰}, z, \widehat{F})$ می‌سازیم که رشته‌هایی را می‌پذیرد که دو ماشین $M_۱$ و $M_۲$ می‌پذیرند.

- وقتی نمادی از ورودی خوانده می‌شود، ماشین $\widehat{M}$ حرکت‌های هر دو ماشین $M_۱$ و $M_۲$ را شبیه‌سازی می‌کند.

ویژگی بستاری

- برای این کار ماشین $\widehat{M}$ را به طوری طراحی می‌کنیم که $\widehat{Q} = Q \times P$ و $\widehat{q_0} = (q_0, p_0)$ و $\widehat{F} = F_1 \times F_2$.
- همچنین $\widehat{\delta}$ را طوری طراحی می‌کنیم که $((q_k, p_l), x) \in \widehat{\delta}((q_i, p_j), a, b)$ اگر و تنها اگر $(q_k, x) \in \delta_1(q_i, a, b)$ و همچنین $\delta_2(p_j, a) = p_l$.
- در صورتی که در ماشین پشته‌ای غیرقطعی گذار تهی وجود داشت و داشتیم $a = \lambda$، آنگاه قرار می‌دهیم $p_j = p_l$.
- به عبارت دیگر هر حالت در ماشین $\widehat{M}$ به نام (q_i, p_j) نمایندهٔ حالت‌های ماشین پشته‌ای M_1 و ماشین متناهی قطعی M_2 است و گذار برای آن حالت به ازای نمادهای الفبا برای هر دو ماشین محاسبه می‌شود.

ویژگی بستاری

- می‌توان نشان داد که $((q_0, p_0), w, z) \vdash^{*}_{\widehat{M}} ((q_r, p_s), \lambda, x)$ به ازای $q_r \in F_1$ و $p_s \in F_2$ اگر و تنها اگر $(q_0, w, z) \vdash^{*}_{\widehat{M}} (q_r, \lambda, x)$ و $\delta^*(p_0, w) = p_s$
- بنابراین یک رشته توسط ماشین $\widehat{M}$ پذیرفته می‌شود اگر و تنها اگر توسط ماشین M_1 و ماشین M_2 پذیرفته شود یا به عبارت دیگر آن رشته در $L_1 \cap L_2 = L(M_1) \cap L(M_2)$ باشد.
- می‌گوییم زبان‌های مستقل از متن بر روی اشتراک زبان‌های منظم بسته‌اند.

- نشان دهید زبان $L = \{a^n b^n : n \geq ۰, n \neq ۱۰۰\}$ مستقل از متن است.

ویژگی بستاری

- نشان دهید زبان $L = \{a^n b^n : n \geq ۰, n \neq ۱۰۰\}$ مستقل از متن است.
- فرض کنیم $L_۱ = \{a^{۱۰۰} b^{۱۰۰}\}$. می‌دانیم $L_۱$ منظم است و بنابراین $\overline{L_۱}$ نیز منظم است.
- زبان $L_۲ = \{a^n b^n : n \geq ۰\}$ نیز مستقل از متن است زیرا یک گرامر مستقل از متن برای آن وجود دارد.
- بنابراین $L = L_۲ \cap \overline{L_۱}$ نیز طبق ویژگی بستاری، مستقل از متن است.

ویژگی بستاری

- نشان دهید زبان $L = \{w \in \{a, b, c\}^* : n_a(w) = n_b(w) = n_c(w)\}$ مستقل از متن نیست.

ویژگی بستاری

- نشان دهید زبان $L = \{w \in \{a, b, c\}^* : n_a(w) = n_b(w) = n_c(w)\}$ مستقل از متن نیست.
- فرض کنیم زبان L یک زبان مستقل از متن باشد. آنگاه زبان $L \cap L(a^*b^*c^*) = L_۲ = \{a^n b^n c^n : n \geq ۰\}$ نیز طبق ویژگی بستاری باید مستقل از متن باشد.
- اما با استفاده از لم تزریق برای زبان‌های مستقل از متن نشان دادیم که زبان $L_۲$ مستقل از متن نیست.
- بنابراین فرض اولیه نادرست بوده و زبان L نمی‌تواند مستقل از متن باشد.

ماشین‌های تورینگ

مقدمه

- با استفاده از لم تزریق برای زبان‌های مستقل از متن، نشان دادیم که زبان‌هایی مانند $\{a^n b^n c^n\}$ و $\{ww\}$ مستقل از متن نیستند و بنابراین برای شناسایی آنها به ماشین‌های قدرتمندتری نیاز داریم.
- ماشین‌های پشته‌ای به دلیل داشتن پشته نسبت به ماشین‌های متناهی قدرت بیشتر پیدا کردند. می‌توانیم حدس بزنیم که با داشتن حافظه‌ای با انعطاف بیشتر نسبت به پشته می‌توانیم ماشینی بسازیم که از ماشین‌های پشته‌ای قوی‌ترند.
- در این قسمت ماشین‌های تورینگ را معرفی می‌کنیم و با استفاده از این ماشین‌ها مفهوم الگوریتم محاسباتی را تعریف می‌کنیم و در پایان نشان می‌دهیم که این ماشین‌ها هر نوع محاسباتی را می‌توانند انجام دهند.

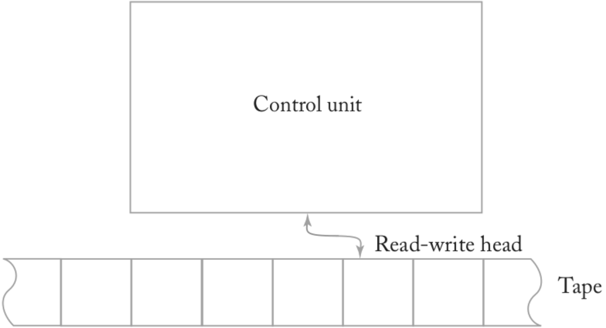
ماشین تورینگ استاندارد

- ماشین تورینگ ماشینی است که حافظه موقت آن یک نوار [1] است.
- این نوار از سلول‌ها [2] (خانه‌ها) یی تشکیل شده است که هر کدام یک نماد را در بر می‌گیرند.
- یک هد (کلاهک) خواندن و نوشتن [3] بر روی نوار قرارگرفته است که قادر است به سمت راست و چپ حرکت کند و در هر حرکت نمادی را از روی یکی از سلول‌های نوار بخواند و یا نمادی را بر روی یک سلول بنویسد.
- بنابراین نوار و هد بر روی آن، مکانیزم ورودی و خروجی این ماشین را تشکیل می‌دهند.

[1] tape
[2] cell
[3] read-write head

ماشین تورینگ استاندارد

- یک ماشین تورینگ را می‌توان بدین شکل نشان داد.



ماشین تورینگ استاندارد

- یک ماشین تورینگ M با یک هفت‌تایی $M = (Q, \Sigma, \Gamma, \delta, q_\circ, \square, F)$ تعریف می‌شود، به طوری که:
- Q مجموعهٔ حالات داخلی ماشین است.
- Σ الفبای ورودی است.
- Γ مجموعه‌ای متناهی از نمادهاست به نام الفبای نوار [1] .
- δ تابع گذار است.
- $\square \in \Gamma$ یک نماد ویژه به نام نماد نانوشته [2] است.
- $q_\circ \in Q$ حالت آغازی است.
- $F \subseteq Q$ مجموعه‌ای از حالت‌های پایانی است.

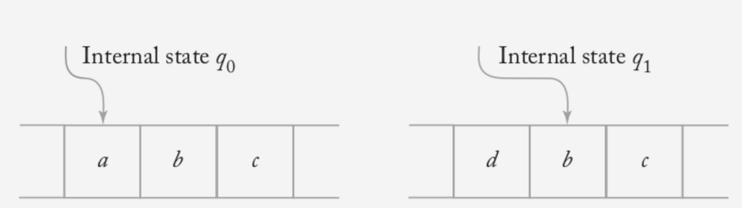
[1] tape alphabet
[2] blank

ماشین تورینگ استاندارد

- در ماشین تورینگ الفبای ورودی زیر مجموعه‌ای است از الفبای نوار و نماد نانوشته را در بر نمی‌گیرد. به عبارت دیگر $\Sigma \subseteq \Gamma - \{\square\}$
- همچنین تابع گذار یک تابع جزئی است که به صورت $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ تعریف می‌شود.
- بنابراین تابع گذار به طوری تعریف می‌شود که ماشین با خواندن یک نماد از نوار بر اساس حالتی که در آن قرار دارد به یک حالت دیگر گذار می‌کند و یک نماد دیگر بر روی نوار می‌نویسد. سپس هد نوار به سلول سمت چپ و یا به سلول سمت راست حرکت می‌کند.

ماشین تورینگ استاندارد

- برای مثال با تابع گذار $\delta(q_۰, a) = (q_۱, d, R)$ نوار از پیکربندی شکل سمت چپ به پیکربندی شکل سمت راست تغییر می‌کند.



ماشین تورینگ استاندارد

- ماشین تورینگ یک مدل اولیه برای کامپیوترهای امروزی است.
- این ماشین یک واحد پردازش دارد که حافظه محدود دارد و یک حافظهٔ جانبی دارد که از لحاظ نظری نامحدود است.
- تعداد دستورات این ماشین (که یک پردازنده است) بسیار محدود است. این ماشین تنها می‌تواند یک نماد را بخواند و تصمیم بگیرد که به چه حالتی برود، چه نمادی را بنویسد و هد خود را به چه سمتی حرکت دهد.

ماشین تورینگ استاندارد

- به نظر می‌رسد ماشین تورینگ بسیار ساده و ابتدایی باشد ولی می‌تواند عملیات پیچیده‌ای انجام دهد. تابع گذار این ماشین را برنامه [1] ماشین تورینگ می‌گوییم.

- ماشین از حالت آغازی شروع به کار می‌کند، با خواندن نمادها تعدادی سلول نوار را تغییر می‌دهد، و از حالتی به حالت دیگر می‌رود و در پایان در صورتی که در حالتی قرار بگیرد که هیچ گذاری تعریف نشده باشد، به حالت توقف [2] می‌رود.

- در یک حالت توقف هیچ گذاری تعریف نشده است. همچنین در حالت‌های پایانی در ماشین تورینگ هیچ گذاری تعریف نشده است، پس هر گاه یک ماشین تونیگ به یک حالت پایانی وارد می‌شود متوقف می‌شود.

[1] program
[2] halt state

ماشین تورینگ استاندارد

- یک ماشین تورینگ را که به صورت $Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{a, b, \Box\}, F = \{q_1\}$ تعریف شده است در نظر بگیرید.

- توابع گذار به صورت زیر هستند.
$$\delta(q_0, a) = (q_0, b, R)$$
$$\delta(q_0, b) = (q_0, b, R)$$
$$\delta(q_0, \Box) = (q_1, \Box, L)$$

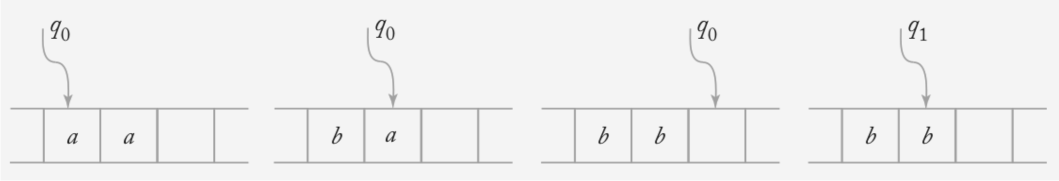
- این ماشین چه عملیاتی انجام می‌دهد؟

ماشین تورینگ استاندارد

- یک ماشین تورینگ را که به صورت $Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{a, b, \square\}, F = \{q_1\}$ تعریف شده است در نظر بگیرید.
- توابع گذار به صورت زیر هستند.
$$\delta(q_0, a) = (q_0, b, R)$$
$$\delta(q_0, b) = (q_0, b, R)$$
$$\delta(q_0, \square) = (q_1, \square, L)$$
- این ماشین به ازای هر نماد a خوانده شده، آن را به b تغییر می‌دهد و در نهایت با خواندن نماد نانوشته به حالت پایانی می‌رود و توقف می‌کند.

ماشین تورینگ استاندارد

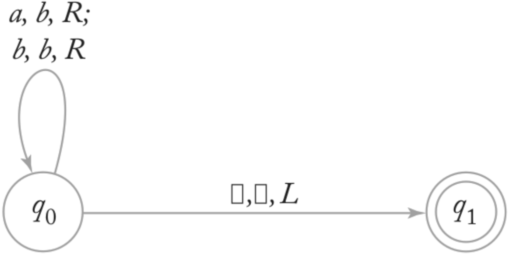
- این ماشین به ازای هر نماد a خوانده شده، آن را به b تغییر می‌دهد و در نهایت با خواندن نماد نانوشته به حالت پایانی می‌رود و توقف می‌کند.
- $\delta(q_0, a) = (q_0, b, R)$ ،$\delta(q_0, b) = (q_0, b, R)$ ،$\delta(q_0, \square) = (q_1, \square, L)$



ماشین تورینگ استاندارد

- همانند قبل، ماشین تورینگ را می‌توانیم توسط یک گراف گذار نشان دهیم. بر روی یال‌ها به ترتیب، نماد خوانده شده از روی نوار، نماد نوشته شده بر روی نوار، و جهت حرکت هد درج شده است.

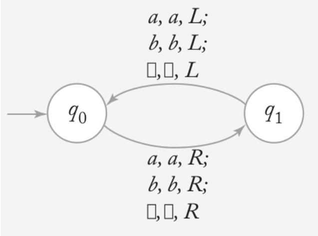
- $\delta(q_0, a) = (q_0, b, R)$ ، $\delta(q_0, b) = (q_0, b, R)$ ، $\delta(q_0, \square) = (q_1, \square, L)$



ماشین تورینگ استاندارد

- یک ماشین تورینگ ممکن است هیچ‌گاه متوقف نشود. در این صورت می‌گوییم ماشین در یک حلقهٔ بی‌پایان افتاده است.[1]

- ماشین تورینگ زیر در زمان اجرا در یک حلقهٔ بی‌پایان می‌افتد.



[1] infinite loop

ماشین تورینگ استاندارد

- ماشین تورینگ را می‌توان به چندین طریق تعریف کرد. یک ماشین تورینگ استاندارد [1] را به صورت زیر تعریف می‌کنیم.

۱. یک ماشین تورینگ نواری دارد که از دو طرف نامحدود است و می‌تواند به طور نامحدود به سمت چپ و راست حرکت کند.

۲. ماشین تورینگ قطعی است، بدین معنی که در هر حالت برای یک نماد در تابع گذار فقط یک حرکت تعریف شده است.

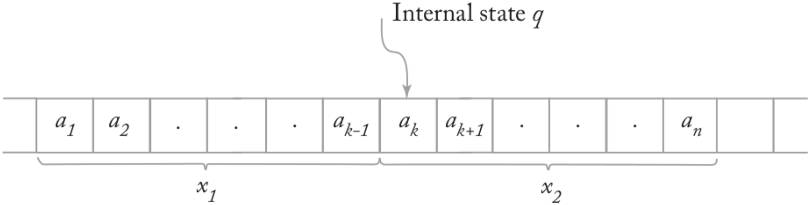
۳. ماشین فایل ورودی و خروجی جداگانه‌ای ندارد. فرض می‌کنیم که قبل از آغاز به کار ماشین، ورودی بر روی نوار نوشته شده باشد. همچنین بعد از توقف ماشین، خروجی بر روی نوار نوشته شده است.

- در آینده با انواع دیگر ماشین تورینگ آشنا می‌شویم.

[1] standard Turing machine

ماشین تورینگ استاندارد

- همانند ماشین‌های پشته‌ای، در ماشین تورینگ از توصیف لحظه‌ای [1] استفاده می‌کنیم.
- هر پیکربندی با توجه به حالت داخلی فعلی ماشین، محتوای نوار، و موقعیت هد ماشین تعیین می‌شود.
- توصیف لحظه‌ای ماشین را با $x_۱qx_۲$ یا $a_۱ \cdots a_{k-۱}qa_k \cdots a_n$ نشان می‌دهیم که بدین معناست که ماشین در حالت q قرار دارد، محتوای نوار $x_۱x_۲ = a_۱ \cdots a_n$ است، و هد ماشین بر روی اولین نماد $x_۲$ یعنی a_k قرار دارد.



[1] instantaneous description

ماشین تورینگ استاندارد

- فرض می‌کنیم که محتوای نوار قبل از x_1 و بعد از x_2 را نمادهای نانوشته تشکیل داده‌اند.
- در صورتی که نماد نانوشته در میانهٔ رشتهٔ ورودی با اهمیت بود آن را نشان می‌دهیم، برای مثال $w\Box q$.

ماشین تورینگ استاندارد

- یک حرکت از یک پیکربندی به یک پیکربندی دیگر را با علامت $\vdash$ نشان می‌دهیم.

- بنابراین اگر داشته باشیم $\delta(q_۱, c) = (q_۲, e, R)$ آنگاه حرکت $abq_۱cd \vdash abeq_۲d$ انجام می‌شود، در صورتی که محتوای نوار abcd باشد و ماشین در حالت $q_۱$ قرار داشته باشد و هد ماشین بر روی حرف c باشد.

- نماد $\overset{*}{\vdash}$ برای حرکت در چند گام نشان داده می‌شود.

- همچنین می‌نویسیم $\vdash_M$ اگر حرکت برای ماشین M را در نظر داشته باشیم.

ماشین تورینگ استاندارد

- فرض کنید $M = (Q, \Sigma, \Gamma, \delta, q_۰, \square, F)$ یک ماشین تورینگ باشد.
- آنگاه رشتهٔ $a_۱ \cdots a_{k-۱} q_۱ a_k \cdots a_n$ یک توصیف لحظه‌ای از ماشین M است به طوری که $a_i \in \Gamma$ و $q_۱ \in Q$.
- حرکت $a_۱ \cdots a_{k-۱} q_۱ a_k a_{k+۱} \cdots a_n \vdash a_۱ \cdots a_{k-۱} b q_۲ a_{k+۱} \cdots a_n$ امکان پذیر است اگر و تنها اگر داشته باشیم: $\delta(q_۱, a_k) = (q_۲, b, R)$.
- حرکت $a_۱ \cdots a_{k-۱} q_۱ a_k a_{k+۱} \cdots a_n \vdash a_۱ \cdots q_۲ a_{k-۱} b a_{k+۱} \cdots a_n$ امکان پذیر است اگر و تنها اگر داشته باشیم: $\delta(q_۱, a_k) = (q_۲, b, L)$.
- با شروع از پیکربندی $x_۱ q_i x_۲$ ماشین متوقف می‌شود اگر در یک یا چند گام به پیکربندی $y_۱ q_j a y_۲$ برود $x_۱ q_i x_۲ \overset{*}{\vdash} y_۱ q_j a y_۲$ به طوری که $\delta(q_j, a)$ تعریف نشده باشد.
- محاسبه [1] دنباله‌ای از پیکربندی‌های ماشین است که به توقف می‌انجامد.

[1] computation

ماشین تورینگ استاندارد

- اگر یک ماشین با شروع از پیکربندی x_1qx_2 هیچ‌گاه متوقف نشود و در یک حلقهٔ بی‌پایان بیافتد می‌نویسیم $x_1qx_2 \overset{*}{\vdash} \infty$.

ماشین تورینگ استاندارد

- ماشین تورینگ می‌تواند به عنوان یک پذیرنده نیز در نظر گرفته شود.
- اگر رشتهٔ w روی نوار ماشین نوشته شود و بقیهٔ نوار را نمادهای نانوشته تشکیل دهند، ماشین می‌تواند در حالت آغازی با هدی بر روی اولین نماد رشتهٔ w آغاز به کار کند. در صورتی که بعد از تعدادی حرکت، ماشین به یک حالت پایانی رفته و توقف کند، رشته پذیرفته می‌شود.
- فرض کنید $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ یک ماشین تورینگ باشد. زبان پذیرفته شده توسط این زبان برابر است با:
- $L(M) = \{w \in \Sigma^+ : q_0 w \overset{*}{\vdash} x_1 q_f x_2, q_0 \in Q, q_f \in F, x_1, x_2 \in \Gamma^*\}$
- رشته پذیرفته نمی‌شود اگر ماشین در یک حالت غیر پایانی توقف کند و یا اگر ماشین هیچ گاه توقف نکند و در حلقهٔ بی‌پایان بیافتد.

ماشین تورینگ استاندارد

- پس برای محدود کردن رشتهٔ ورودی آن را با نمادهای نانوشته از چپ و راست محصور می‌کنیم.
- بدین ترتیب می‌توانیم در نوار نامحدود، مکان رشته را مشخص کنیم. در غیر اینصورت ماشین هیچ راهی جز جستجو بر روی نوار نامحدود برای نمادهای ورودی نداشت و هیچ گاه نمی‌توانستیم مشخص کنیم آیا حرکت‌های ماشین پایان می‌پذیرد یا خیر.

ماشین تورینگ استاندارد

- یک ماشین تورینگ طراحی کنید که زبان $L = \{a^n b^n : n \geq ۱\}$ را بپذیرد.

ماشین تورینگ استاندارد

- یک ماشین تورینگ طراحی کنید که زبان $L = \{a^n b^n : n \geq ۱\}$ را بپذیرد.
- ماشین را بدین صورت طراحی می‌کنیم که ابتدا با خواندن نماد a آن را با نماد x جایگزین می‌کنیم و هد را به سمت راست حرکت می‌دهیم تا به اولین نماد b برخورد کنیم. نماد b را با نماد y جایگزین می‌کنیم و هد را به سمت چپ حرکت می‌دهیم تا به نماد اولین نماد a بعد از یک نماد x برخورد کنیم. دوباره نماد a را با نماد x جایگزین می‌کنیم و این کار را آنقدر ادامه می‌دهیم تا همه نمادهای a با x و همهٔ b با y جایگزین شود. اگر هیچ نماد a یا b باقی نماند، تعداد نمادهای a و b مساوی بوده است و رشته باید پذیرفته شود.

ماشین تورینگ استاندارد

- یک ماشین تورینگ طراحی کنید که زبان $L = \{a^n b^n : n \geq ۱\}$ را بپذیرد.
- $Q = \{q_۰, q_۱, q_۲, q_۳, q_۴\}, F = \{q_۴\}, \Sigma = \{a, b\}, \Gamma = \{a, b, x, y, \square\}$
- ابتدا با حرکت دادن هد به سمت راست، به ازای یک نماد a یک نماد b پیدا می‌کنیم، a را با x و b را با y جایگزین می‌کنیم:

$$\delta(q_۰, a) = (q_۱, x, R)$$
$$\delta(q_۱, a) = (q_۱, a, R)$$
$$\delta(q_۱, y) = (q_۱, y, R)$$
$$\delta(q_۱, b) = (q_۲, y, L)$$

- سپس به حالت $q_۲$ می‌رویم و در این حالت هد را آنقدر به سمت چپ حرکت می‌دهیم تا اولین نماد a را پیدا کنیم:

$$\delta(q_۲, y) = (q_۲, y, L)$$
$$\delta(q_۲, a) = (q_۲, a, L)$$
$$\delta(q_۲, x) = (q_۰, x, R)$$

ماشین تورینگ استاندارد

- در پایان وقتی همهٔ نمادهای a و b جایگزین شدند، ماشین در حالت q_0 با یک نماد y مواجه می‌شود. بنابراین باید از همهٔ نمادهای y عبور کند تا به نماد نانوشته برسد و رشته را بپذیرد.

$$\delta(q_0, y) = (q_3, y, R)$$
$$\delta(q_3, y) = (q_3, y, R)$$
$$\delta(q_3, \Box) = (q_4, \Box, R)$$

ماشین تورینگ استاندارد

- در صورتی که رشته‌ای متعلق به این زبان نباشد، چند احتمال ممکن است وجود داشته باشد:

۱. رشتهٔ a^*b^* دریافت شده و تعداد نمادهای a از تعداد b بیشتر است. در اینصورت با حرکت به سمت راست در حالت q_1 ماشین هیچ نماد b به ازای یک نماد a پیدا نمی‌کند و با خواندن نماد نانوشته در حالت q_1 متوقف می‌شود.

۲. رشتهٔ a^*b^* دریافت شده و تعداد نمادهای a از نمادهای b کمتر است و در اینصورت ماشین در حالت q_3 به نماد b برخورد می‌کند و متوقف می‌شود.

۳. رشته‌ای غیر از a^*b^* دریافت شده و ماشین در یکی از حالت‌ها متوقف می‌شود.

- در صورتی که رشتهٔ aabb دریافت شود، دنبالهٔ حرکت‌های ماشین به صورت زیر خواهد بود:
- $q_0aabb \vdash xq_1abb \vdash xaq_1bb \vdash xq_2ayb$
$\vdash q_2xayb \vdash xq_0ayb \vdash xxq_1yb$
$\vdash xxyq_1b \vdash xxq_2yy \vdash xq_2xyy$
$\vdash xxq_0yy \vdash xxyq_3y \vdash xxyyq_3\square$
$\vdash xxyy\square q_4\square$

ماشین تورینگ استاندارد

- یک ماشین تورینگ طراحی کنید که زبان $L = \{a^n b^n c^n : n \geq ۱\}$ را بپذیرد.

ماشین تورینگ استاندارد

- یک ماشین تورینگ طراحی کنید که زبان $L = \{a^n b^n c^n : n \geq ۱\}$ را بپذیرد.
- مشابه مسأله قبل به ازای یک نماد a یک نماد b و یک نماد c پیدا می‌کنیم. نماد a را با x ، نماد b را با y و نماد c را با z جایگزین می‌کنیم و در پایان بررسی می‌کنیم که هیچ نماد a یا b یا c باقی نمانده باشد.
- پس ماشین تورینگ علاوه بر زبان $\{a^n b^n\}$ که مستقل از متن است، زبان $\{a^n b^n c^n\}$ را که مستقل از متن نیست را می‌پذیرد و بنابراین قدرت آن از ماشین پشته‌ای بیشتر است.

ماشین تورینگ استاندارد

- از آنجایی که ماشین تورینگ علاوه بر خواندن ورودی از روی نوار، می‌تواند یک خروجی نیز تولید کند، بنابراین این ماشین نه تنها به عنوان یک پذیرنده، بلکه به عنوان یک مبدل نیز می‌تواند مورد استفاده قرار بگیرد.
- ماشین مبدل تورینگ M تابع f را که با رابطهٔ $\widehat{w} = f(w)$ تعریف شده است، پیاده‌سازی می‌کند، اگر $q_0 w \overset{*}{\vdash}_M q_f \widehat{w}$ به طوری که q_f یک حالت پایانی است.

ماشین تورینگ استاندارد

- تابع f با دامنهٔ D تورینگ-محاسبه‌پذیر [1] یا محاسبه‌پذیر [2] نامیده می‌شود اگر یک ماشین تورینگ $M = (Q, \Sigma, \Gamma, \delta, q_\circ, \square, F)$ وجود داشته باشد به طوری که

$$q_\circ w \overset{*}{\vdash}_M q_f f(w) \quad , \quad q_f \in F$$

به ازای هر $w \in D$

[1] Turing-computable
[2] computable

ماشین تورینگ استاندارد

- به ازای دو عدد صحیح مثبت x و y داده شده، یک ماشین تورینگ طراحی کنید که $x + y$ را محاسبه کند.

ماشین تورینگ استاندارد

- به ازای دو عدد صحیح مثبت x و y داده شده، یک ماشین تورینگ طراحی کنید که $x + y$ را محاسبه کند.
- ابتدا باید روشی برای نمایش یک عدد صحیح ارائه کنیم که بتوان عمل جمع را با استفاده از آن به سادگی انجام داد. برای این کار از نمایش یگانی [1] استفاده می‌کنیم.
- در نمایش یگانی به ازای عدد صحیح مثبت x داریم $w(x) \in \{۱\}^+$ به طوری که $x = |w(x)|$
- همچنین برای عملگر جمع از نماد صفر استفاده می‌کنیم و دو عدد را با یک نماد صفر از یکدیگر جدا می‌کنیم.
- در نهایت نتیجهٔ جمع دو عدد را با یک نماد صفر در پایان بر روی نوار می‌نویسیم.
- پس داریم $q_۰ w(x) ۰ w(y) \overset{*}{\vdash} q_f w(x + y) ۰$

[1] unary notation

ماشین تورینگ استاندارد

- پس ماشین تورینگ را طوری طراحی می‌کنیم که نماد صفر بین دو عدد را به یک تبدیل کند و آخرین نماد یک در عدد y را به صفر تبدیل کند.

- $M = (Q, \Sigma, \Gamma, \delta, q_۰, \square, F), Q = \{q_۰, q_۱, q_۲, q_۳, q_۴\}, F = \{q_۴\}$

- $$\begin{aligned}\delta(q_۰, ۱) &= (q_۰, ۱, R)\\ \delta(q_۰, ۰) &= (q_۱, ۱, R)\\ \delta(q_۱, ۱) &= (q_۱, ۱, R)\\ \delta(q_۱, \square) &= (q_۲, \square, L)\\ \delta(q_۲, ۱) &= (q_۳, ۰, L)\\ \delta(q_۳, ۱) &= (q_۳, ۱, L)\\ \delta(q_۳, \square) &= (q_۴, \square, R)\end{aligned}$$

- بنابراین برای جمع دو عدد ۳ و ۲ حرکت‌های زیر را در ماشین داریم:

$$q_{۰}۱۱۱۰۱۱ \vdash ۱q_{۰}۱۱۰۱۱ \vdash ۱۱q_{۰}۱۰۱۱ \vdash ۱۱۱q_{۰}۰۱۱$$
$$\vdash ۱۱۱۱q_{۱}۱۱ \vdash ۱۱۱۱۱q_{۱}۱ \vdash ۱۱۱۱۱۱q_{۱}\square$$
$$\vdash ۱۱۱۱۱q_{۲}۱ \vdash ۱۱۱۱q_{۳}۱۰$$
$$\overset{*}{\vdash} q_{۳}\square۱۱۱۱۱۰ \vdash q_{۴}۱۱۱۱۱۰$$

ماشین تورینگ استاندارد

- یک ماشین تورینگ طراحی کنید که به ازای دو عدد x و y داده شده، در یک حالت پایانی q_y توقف کند اگر $x \geq y$ و در یک حالت غیرپایانی q_n توقف کند، اگر $x < y$.

- به عبارت دیگر

اگر $x \geq y$ آنگاه $q_\circ w(x) \circ w(y) \overset{*}{\vdash} q_y w(x) \circ w(y)$

اگر $x < y$ آنگاه $q_\circ w(x) \circ w(y) \overset{*}{\vdash} q_n w(x) \circ w(y)$

ماشین تورینگ استاندارد

- یک ماشین تورینگ طراحی کنید که به ازای دو عدد x و y داده شده، در یک حالت پایانی q_y توقف کند اگر $x \geq y$ و در یک حالت غیرپایانی q_n توقف کند، اگر $x < y$

- همانند ماشینی که برای زبان $a^n b^n$ طراحی کردیم، این بار ماشینی برای زبان $۱^n ۰ ۱^m$ طراحی می‌کنیم. در پایان، اگر تعداد یک باقیمانده قبل از صفر بیشتر بود عدد x بزرگتر است و در غیر اینصورت عدد y بزرگتر است.

- پس در صورتی که $x > y$ بر روی نوار خواهیم داشت: $\square x \cdots xx ۰ ۱۱ \cdots xx$
و در صورتی که $x < y$ بر روی نوار خواهیم داشت: $\square ۱۱x \cdots xx ۰ xx \cdots xx$

- اگر x بزرگتر باشد، به ازای یک نماد یک در قسمت اول رشته، نماد یک در قسمت دوم پیدا نمی‌کنیم و به حالت q_y می‌رویم.

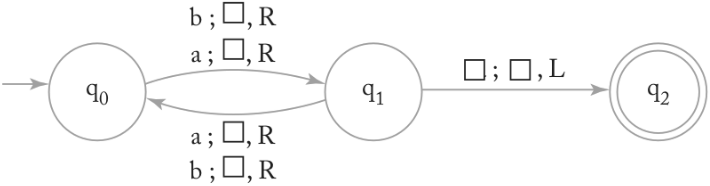
- اگر y بزرگتر باشد، وقتی همهٔ نمادهای یک قسمت اول تبدیل شدند حداقل یک نماد یک در قسمت دوم پیدا می‌کنیم و به حالت q_n می‌رویم.

مثال

- ماشین تورینگی طراحی کنید که زبان $L = \{w : |w| \text{ فرد است}\}$ را بپذیرد.

مثال

- ماشین تورینگی طراحی کنید که زبان $L = \{w : |w| \text{ فرد است}\}$ را بپذیرد.

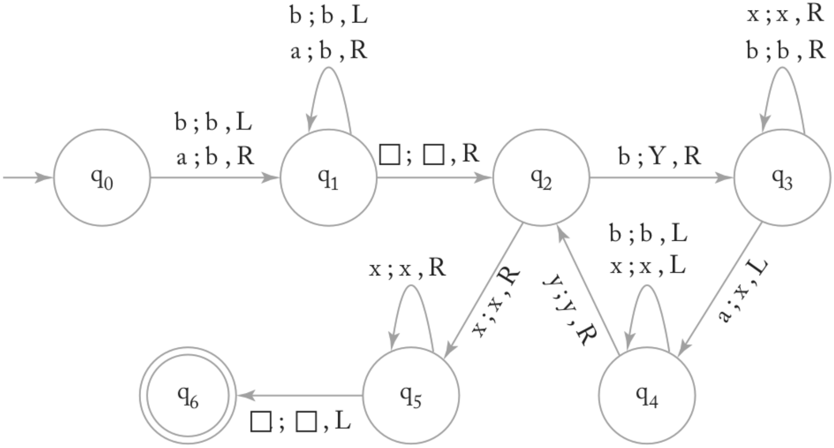


مثال

- ماشین تورینگی طراحی کنید که زبان $L = \{a^n b^m a^{n+m} : n \geq ۰, m \geq ۱\}$ را بپذیرد.

مثال

- ماشین تورینگی طراحی کنید که زبان $L = \{a^n b^m a^{n+m} : n \geq ۰, m \geq ۱\}$ را بپذیرد.

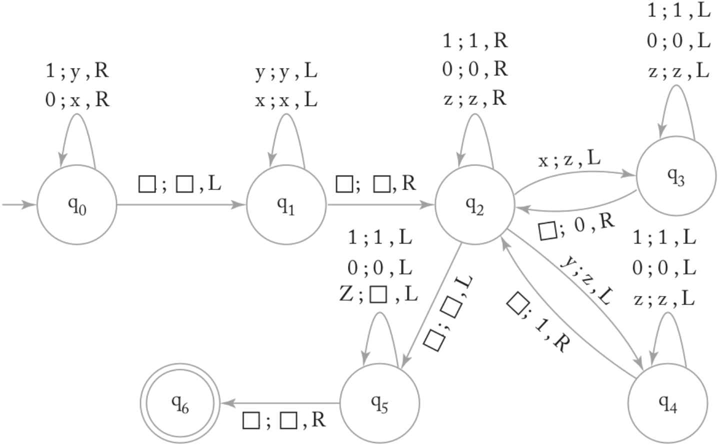


مثال

- ماشین تورینگی طراحی کنید که تابع $f(w) = w^R$ را به ازای $w \in \{۰, ۱\}^+$ محاسبه کند.

مثال

- ماشین تورینگی طراحی کنید که تابع $f(w) = w^R$ را به ازای $w \in \{۰, ۱\}^+$ محاسبه کند.



ترکیب ماشین‌های تورینگ

- دیدیم چگونه می‌توانیم عملیات ساده‌ای را توسط ماشین تورینگ انجام دهیم.
- برای عملیات پیچیده‌تر می‌توانیم این عملیات ساده را با یکدیگر ترکیب کنیم.

- یک ماشین تورینگ طراحی کنید که تابع زیر را محاسبه کند:

اگر $x \geq y$ آنگاه $f(x, y) = x + y$

اگر $x < y$ آنگاه $f(x, y) = ۰$

ترکیب ماشین‌های تورینگ

- یک ماشین تورینگ طراحی کنید که تابع زیر را محاسبه کند:
  اگر $x \geq y$ آنگاه $f(x, y) = x + y$
  اگر $x < y$ آنگاه $f(x, y) = ۰$

- از آنجایی که ماشین تورینگ را برای مقایسه و جمع طراحی کرده‌ایم از این پس به جای آن ماشین‌ها می‌توانیم از توصیف سطح بالا استفاده کنیم و آنها را توسط نام عملیاتشان نشان دهیم.

- بعد از اینکه دو عدد را مقایسه کردیم به حالتی می‌رویم که آن حالت، حالت آغازی برای ماشین تورینگی است که عملیات بعدی را انجام می‌دهد.

- پس می‌توانیم دو عدد را توسط ماشین تورینگ مقایسه‌گر C مقایسه کنیم و سپس اگر عدد اول بزرگتر بود یا دو عدد مساوی بودند، عملیات ماشین تورینگ جمع‌کنندهٔ A را برای محاسبهٔ مجموع آغاز می‌کنیم و اگر عدد دوم بزرگتر بود عملیات ماشین تورینگ صفرکننده E را برای تولید خروجی صفر آغاز می‌کنیم.

ترکیب ماشین‌های تورینگ

- پس بعد از مقایسهٔ دو عدد توسط ماشین تورینگ C ، اگر عدد x از عدد y بزرگتر بود، به حالت $q_{A,0}$ می‌رویم تا دو عدد را با یکدیگر جمع کنیم و در غیر اینصورت به حالت $q_{E,0}$ تا اعداد روی نوار را پاک کرده و عدد صفر را بر روی نوار بنویسیم.

- اگر $x \geq y$ آنگاه $q_{C,0}w(x)0w(y) \overset{*}{\vdash} q_{A,0}w(x)0w(y)$

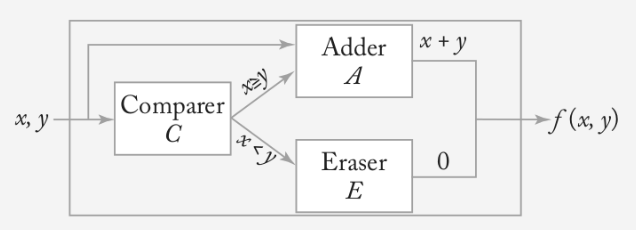
- اگر $x < y$ آنگاه $q_{C,0}w(x)0w(y) \overset{*}{\vdash} q_{E,0}w(x)0w(y)$

- $q_{A,0}w(x)0w(y) \overset{*}{\vdash} q_{A,f}w(x+y)0$

- $q_{E,0}w(x)0w(y) \overset{*}{\vdash} q_{E,f}0$

ترکیب ماشین‌های تورینگ

- یک ماشین تورینگ طراحی کنید که تابع زیر را محاسبه کند:
  اگر $x \geq y$ آنگاه $f(x, y) = x + y$
  اگر $x < y$ آنگاه $f(x, y) = ۰$



ترکیب ماشین‌های تورینگ

- برای توصیف سطح بالای یک ماشین تورینگ همچنین می‌توانیم از شبه‌کد [1] استفاده کنیم.
- شبه‌کد روشی است برای توصیف عملکرد یک ماشین توسط یک زبان سطح بالا نزدیک به زبان طبیعی انسان.
- گرچه این شبه‌کدها توسط کامپیوترها قابل فهم نیستند اما فرض می‌کنیم که روشی برای تبدیل آنها به زبان ماشین وجود دارد.

[1] pseudocode

ترکیب ماشین‌های تورینگ

- همچنین می‌توانیم از مفهوم زیربرنامه استفاده کنیم، بدین معنی که ماشین اول (یا به طور دقیق‌تر زیرماشین اول) ماشین دوم را برای اجرا فراخوانی می‌کند. ماشین دوم مقدار مورد نیاز ماشین اول را محاسبه می‌کند و نتیجه را روی نوار می‌نویسد. ماشین اول مجددا محاسبات خود را از سر می‌گیرد.

ترکیب ماشین‌های تورینگ

- یک ماشین تورینگ طراحی کنید که دو عدد را در هم ضرب می‌کند.

ترکیب ماشین‌های تورینگ

- یک ماشین تورینگ طراحی کنید که دو عدد را در هم ضرب می‌کند.
- در اینجا از توصیف سطح بالا استفاده می‌کنیم.
- دو عدد x و y را در نمایش یگانی در نظر می‌گیریم. برای ضرب عدد x در عدد y باید به ازای هر نماد یک در عدد y را روی نوار بنویسیم. سپس همهٔ عددهای y نوشته شده بر روی نوار را با هم جمع کنیم. x
- پس به طور دقیق‌تر (۱) به ازای هر نماد یک در عدد x آن را با نماد a جایگزین می‌کنیم و به حالتی می‌رویم که در آن عدد y را روی نوار تکرار می‌کنیم و به حالت اولیه بازمی‌گردیم. (۲) این کار را ادامه می‌دهیم تا هیچ یک از نمادهای یک در عدد x باقی نماند، سپس به حالتی می‌رویم که در آن همهٔ اعداد y نوشته شده بر روی نوار را با هم جمع کنیم. (۳) در پایان برای بازیابی عدد x می‌توانیم همهٔ نمادهای a را با نماد یک جایگزین کنیم.

مثال

- ماشین تورینگی طراحی کنید که یک عدد یگانی را به یک عدد دودویی تبدیل کند. به عبارت دیگر این ماشین باید طول یک رشته شامل نمادهای ۱ را بشمارد و حاصل را در مبنای دو بر روی نوار بنویسد.

مثال

- ماشین تورینگی طراحی کنید که یک عدد یگانی را به یک عدد دودویی تبدیل کند. به عبارت دیگر این ماشین باید طول یک رشته شامل نمادهای ۱ را بشمارد و حاصل را در مبنای دو بر روی نوار بنویسد.
- برای تبدیل یک عدد به معادل آن در مبنای ۲ ، آن عدد را به ۲ تقسیم می‌کنیم و باقیمانده را مرتبهٔ $۲^۰$ می‌نویسیم. سپس خارج قسمت به دست آمده را به ۲ تقسیم می‌کنیم و باقیماندهٔ بعدی را در مرتبهٔ $۲^۱$ می‌نویسیم. این کار را ادامه می‌دهیم تا جایی که خارج قسمت به دست آمده ۰ شود.
- به عبارت دیگر برای به دست آوردن معادل عدد x در مبنای ۲ چنین عمل می‌کنیم: $x = ۲x_۰ + p_۰, x_۰ = ۲x_۱ + p_۱, x_۱ = ۲x_۲ + p_۲, \cdots, x_n = p_n$
- از بسط دادن عدد x به دست می‌آوریم: $x = ۲^n p_n + ۴p_۲ + ۲p_۱ + p_۰$
- $(x)_{۱۰} = (p_n p_{n-۱} \cdots p_۲ p_۱ p_۰)_۲$

مثال

- بنابراین ماشین تورینگی که طراحی می‌کنیم، بر روی ورودی ۱۱۱۰۰۰۱۱۱ چنین عمل می‌کند:

۱. در هر گام i (با شروع از $i = 0$) به ازای هر دو نماد یک، یکی را حذف می‌کنیم. برای حذف کردن یک نماد، آن را با علامتی جایگزین می‌کنیم. در این مرحله در واقع عدد را بر دو تقسیم می‌کنیم.

۲. اگر تعداد نمادهای ۱ زوج بود، در پایان تقسیم، باقیماندهٔ تقسیم صفر می‌شود (یعنی به ازای هر نماد ۱ ، یک زوج حذف شونده وجود دارد)، پس $p_i = 0$، بنابراین صفر را در سمت چپ خروجی یادداشت می‌کنیم. اگر تعداد نمادهای ۱ فرد بود، باقیماندهٔ تقسیم ۱ می‌شود، پس یک نماد ۱ باقی می‌ماند که برای آن زوج حذف شونده وجود ندارد، پس $p_i = 1$ و بنابراین عدد ۱ را در سمت چپ خروجی یادداشت می‌کنیم.

۳. به مرحلهٔ ۱ می‌رویم و دوباره نمادهای ۱ را به دو تقسیم می‌کنیم و این کار را ادامه می‌دهیم تا همهٔ نمادهای ورودی حذف شوند و ماشین را در حالت پایانی متوقف می‌کنیم.

مثال

- ماشین تورینگی طراحی کنید که یک عدد دودویی را به یک عدد یگانی تبدیل کند.

مثال

- ماشین تورینگی طراحی کنید که یک عدد دودویی را به یک عدد یگانی تبدیل کند.
- الگوریتم کلی بدین ترتیب عمل می‌کند: در هر مرحله n امین نماد از رشتهٔ دودویی را از سمت راست می‌خوانیم. در ابتدا قرار می‌دهیم $n = ۱$. مرتبهٔ نماد (بیت) خوانده شده برابر است با $m = ۲^i$. در ابتدا قرار می‌دهیم $i = ۰$. مقدار اولیه m را که برابر با ۱ است در قسمتی از نوار می‌نویسیم. حاصل به دست آمده از تبدیل عدد دودویی برابر است با r . در ابتدا قرار می‌دهیم $r = ۰$.

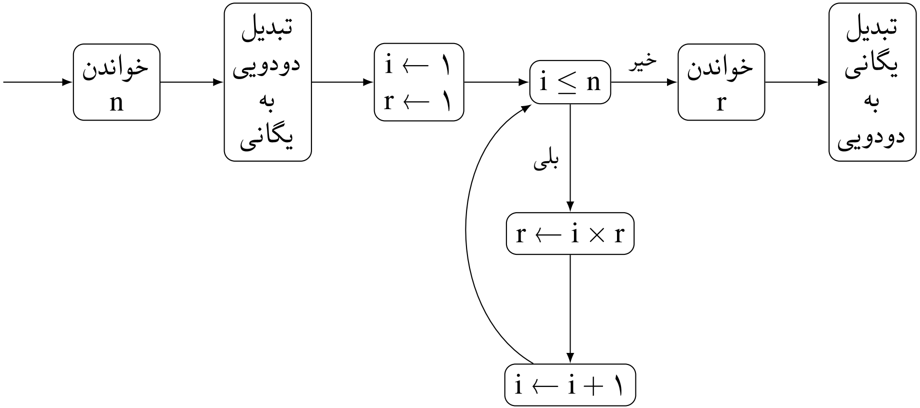
۱. بیت n ام را می‌خوانیم. مقدار آن را در m ضرب می‌کنیم. حاصل به دست آمده را به مقدار حاصل خروجی r اضافه می‌کنیم.

۲. مقدار n را یک واحد می‌افزاییم (یک سلول به سمت چپ می‌رویم). مقدار m را دو برابر می‌کنیم (تعداد نمادهای یک آن را کپی می‌کنیم).

۳. اگر مقدار بیت n ام برابر با نانوشته نبود به مرحله ۱ می‌رویم، در غیر اینصورت ماشین در حالت پایانی متوقف می‌شود.

مثال

- ماشین تورینگی را توصیف کنید که عدد n فاکتوریل را محاسبه کند.



تز تورینگ

- نشان دادیم که ماشین تورینگ نه تنها برای محاسبات ساده، بلکه برای محاسبات پیچیده‌تر نیز با ترکیب ماشین‌های مختلف می‌تواند مورد استفاده قرار بگیرد.
- تا اینجا متوجه شدیم که ماشین تورینگ از ماشین‌های پشته‌ای قدرتمند تر است چنانچه با یک مثال نشان دادیم که ماشین تورینگ زبانی را می‌پذیرد که ماشین‌های پشته‌ای نمی‌پذیرند.
- همچنین نشان دادیم که عملیات ساده مانند مقایسه، جمع و ضرب، و کپی کردن یک مقدار با استفاده از ماشین تورینگ امکان پذیر است و از آنجایی که عملیات پیچیده‌تر از ترکیب این عملیات مقدماتی به دست می‌آیند، می‌توانیم حدس بزنیم که ماشین تورینگ هر محاسبهٔ پیچیده‌ای را می‌تواند انجام دهد.
- اما آیا ماشین تورینگ همهٔ محاسباتی را که با هر ماشین دیگری قابل انجام است می‌تواند انجام دهد؟

تز تورینگ

- اگر بخواهیم ماشین تورینگ را با یک کامپیوتر دیجیتال مقایسه کنیم، کافی است که دستورات کامپیوتر مورد نظر را یک به یک با ماشین‌های تورینگ متناظرشان مقایسه کنیم. از آنجایی که یک محاسبهٔ پیچیده از ترکیب تعدادی دستور تشکیل شده است که توسط ماشین‌های تورینگ قابل شبیه‌سازی هستند، و از آنجایی که ماشین‌های تورینگ را نیز می‌توانیم با هم ترکیب کنیم، پس قدرت ماشین تورینگ باید به اندازهٔ ماشین دیجیتال مورد نظر باشد.
- اگر بتوانیم محاسباتی پیدا کنیم و نشان دهیم که آن محاسبات توسط ماشین دیگری قابل انجام است، ولی هیچ ماشین تورینگی برای آن وجود ندارد، آنگاه به این نتیجه می‌رسیم که ماشینی قدرتمندتر از ماشین تورینگ وجود دارد.
- اما کسی تاکنون چنین محاسباتی پیدا نکرده است، پس حدس می‌زنیم که ماشین تورینگ می‌تواند هر محاسبه‌ای را که توسط هر ماشین دیگری انجام شود را انجام دهد.

تز تورینگ

- تز تورینگ [1] که در سال ۱۹۳۶ توسط آلن تورینگ [2] بیان شد، می‌گوید که هر محاسبه‌ای که توسط یک محاسبه‌گر مکانیکی انجام شود، می‌تواند توسط ماشین تورینگ نیز انجام شود.
- این فرضیه را نمی‌توان اثبات کرد، زیرا باید به طور دقیق و رسمی بیان کنیم منظور از محاسبه‌گر مکانیکی چیست. قبل از کامپیوترهای دیجیتال که در سال ۱۹۴۵ به وجود آمدند، محاسبه‌گرهای مکانیکی برای محاسبات استفاده می‌شدند. همچنین می‌توان محاسبه بر روی کاغذ توسط انسان را یک محاسبهٔ مکانیکی محسوب کرد.
- اما آیا کامپیوترهایی که در آینده به وجود می‌آیند قدرتمندتر از ماشین تورینگ خواهند بود و آیا محاسباتی یافت خواهد شد که توسط ماشین تورینگ قابل انجام نباشند؟

---

[1] Turing thesis
[2] Alan Turing

تز تورینگ

- در حال حاضر می‌دانیم که:

۱. هر محاسبه‌ای که توسط یک کامپیوتر دیجیتال قابل انجام است توسط ماشین تورینگ نیز قابل انجام است.

۲. هیچ کس هیچ مسألهٔ محاسباتی پیدا نکرده است که توسط ماشین تورینگ قابل انجام نباشد.

۳. مدل‌های محاسباتی دیگری (مانند حساب لامبدا [1]) ارائه شده‌اند که هیچ کدام از مدل محاسباتی تورینگ قدرتمندتر نیستند و ثابت شده است که همه با یکدیگر هم‌ارزند.

[1] lambda-calculus

تز تورینگ

- با این حال هنوز تز تورینگ در حد فرضیه است.
- می‌توان تز تورینگ را با قوانین فیزیک نیوتون مقایسه کرد. درستی قوانین نیوتن اثبات شدنی نیست. تنها می‌دانیم که همهٔ آزمایش‌ها و مشاهدات درستی آنها را تأیید می‌کنند، اما ممکن است شرایطی وجود داشته باشد که در آن قوانین نیوتن نقض شوند. پس این قوانین اثبات نمی‌شوند، و تنها ممکن است با مشاهداتی نقض شوند.
- پس تز تورینگ می‌تواند به منزلهٔ قوانین پایه‌ای علوم کامپیوتر در نظر گرفته شود.

- یک الگوریتم [1] برای تابع $f : D \to R$ یک ماشین تورینگ M است، که به ازای دریافت ورودی $d \in D$ خوانده شده از روی نوار خود، در نهایت با جواب $f(d) \in R$ نوشته شده بر روی نوار خود، متوقف می‌شود.
می‌نویسیم $q_f \in F$, $q_\circ d \overset{*}{\vdash}_M q_f f(d)$ به ازای همهٔ مقادیر $d \in D$
- از این پس می‌توانیم از یک شبه‌کد یا یک زبان سطح بالا برای توصیف محاسبات استفاده کنیم با این اطمینان که برای آن یک ماشین تورینگ وجود دارد.

[1] algorithm

ماشین‌های تورینگ دیگر

- حال به بررسی ماشین‌های تورینگ دیگر می‌پردازیم که گرچه امکانات بیشتری (مانند حافظهٔ بیشتر) دارند ولی قدرت آنها از ماشین تورینگ بیشتر نیست.
- در این بخش به بررسی ماشین‌های تورینگ با تعداد بیشتری نوار، نوار در ابعادی بیشتر از یک بعد، ماشین تورینگ غیرقطعی [1]، و ماشین تورینگ جهانی [2] می‌پردازیم.
- در پایان ماشین‌های کراندار خطی [3] را بررسی می‌کنیم که یک ماشین تورینگ محدود شده است و برای شناسایی زبان‌های حساس به متن [4] به کار می‌رود.

[1] nondeterministic Turing machine
[2] universal Turing machine
[3] linear bounded automata
[4] context-sensitive languages

ماشین‌های تورینگ دیگر

- دو ماشین هم‌ارز [1] یکدیگرند اگر هر دو یک زبان را شناسایی کنند.

- یک دسته (طبقه یا کلاس [2]) از ماشین‌ها، ماشین‌هایی هستند که همگی یک تعریف یکسان دارند. برای مثال هر ماشین متناهی قطعی متعلق به دستهٔ ماشین‌های متناهی قطعی است. تاکنون با سه دسته از ماشین‌ها آشنا شدیم: ماشین‌های متناهی، ماشین‌های پشته‌ای، و ماشین‌های تورینگ. هر کدام از این دسته‌ها دو زیر دستهٔ قطعی و غیرقطعی نیز دارند.

[1] equivalent
[2] class

ماشین‌های تورینگ دیگر

- حال دو دستهٔ $C_۱$ و $C_۲$ از ماشین‌ها را در نظر بگیرید.
- اگر برای هر ماشین $M_۱$ در دستهٔ $C_۱$ یک ماشین $M_۲$ در دستهٔ $C_۲$ وجود داشته باشد به طوری که $L(M_۱) = L(M_۲)$ باشد، آنگاه می‌گوییم کلاس $C_۲$ قدرتی حداقل برابر با کلاس $C_۱$ دارد.
- اگر همچنین برای هر ماشین $M_۲$ در دستهٔ $C_۲$ یک ماشین $M_۱$ در دستهٔ $C_۱$ وجود داشته باشد به طوری که $L(M_۱) = L(M_۲)$ باشد، آنگاه می‌گوییم کلاس $C_۱$ و $C_۲$ هم‌ارزند.

ماشین‌های تورینگ با انتخاب توقف

- در ماشین تورینگ استاندارد، همیشه هد به چپ یا راست حرکت می‌کند. گاهی نیاز به توقف نیز می‌باشد که می‌توان گزینهٔ توقف را به ماشین تورینگ افزود. ماشین تورینگ با انتخاب توقف [1] این امکان را به ماشین تورینگ می‌افزاید.
$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

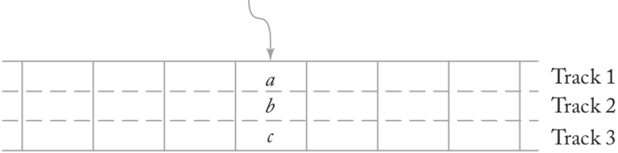
- در اینجا S به معنی توقف هد در یک سلول است.

- این ماشین هم‌ارز ماشین تورینگ استاندارد است. ایدهٔ اثبات: توقف را می‌توان با یک حرکت به چپ و یک حرکت به راست شبیه‌سازی کرد.

[1] Turing machine with a stay-option

ماشین‌های تورینگ با چند شیار

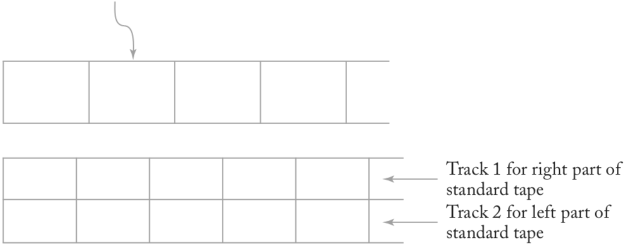
- در تعریف ماشین تورینگ گاهی به جای نوشتن یک نماد در یک سلول، یک رشته در یک سلول می‌نویسند. می‌توان نشان داد که این تعریف با تعریف ماشین تورینگ استاندارد یکسان است.
- حال اگر به جای نوشتن یک نماد در یک سلول، یک سه‌تایی در یک سلول بنویسیم، همانند این است که یک سلول را شیاربندی کرده‌ایم.
- چنین ماشینی را ماشین تورینگ با چند شیار [1] می‌نامیم که هم‌ارز ماشین تورینگ استاندارد است.



[1] Turing machine with multiple track

ماشین‌های تورینگ با نوار نیمه‌نامحدود

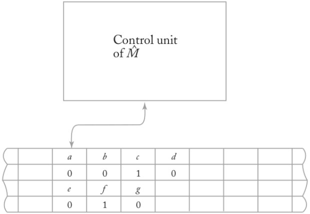
- اگر یک ماشین تورینگ با نوار نیمه‌نامحدود [1] داشته باشیم، می‌توانیم یک ماشین معادل با نوار نیمه‌نامحدود با دو شیار در نظر بگیریم. سپس فرض می‌کنیم هرگاه ماشین از کران سمت چپ نوار به سمت راست حرکت می‌کند، فقط در شیار بالایی می‌نویسد، و هرگاه به کران سمت چپ نوار رسید و با حرکت به چپ از کران گذر کرد در شیار پایینی می‌نویسد. بدین ترتیب این ماشین تورینگ ماشین تورینگ استاندارد را شبیه‌سازی می‌کند.

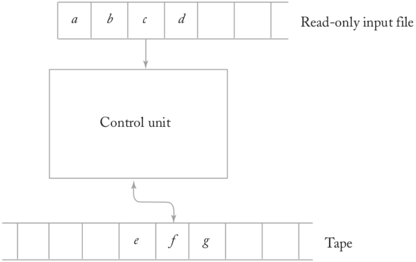


[1] Turing machine with semi-infinite tape

ماشین تورینگ آفلاین

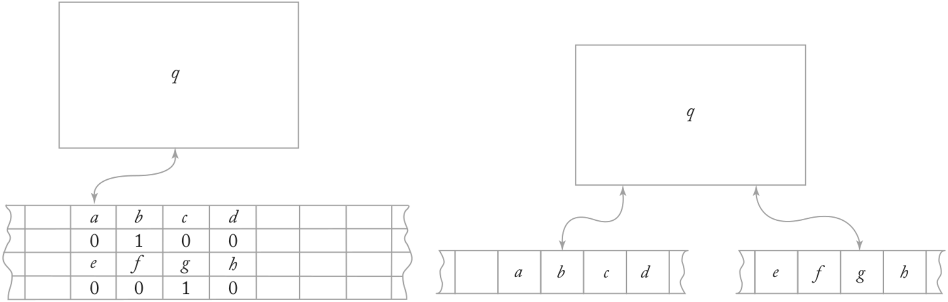
- یک ماشین تورینگ آفلاین ماشینی است که در آن ورودی از هد خواندن و نوشتن جدا شده است. برای شبیه‌سازی این ماشین توسط یک ماشین تورینگ استاندارد از یک ماشین تورینگ با یک نوار چهار شیاری استفاده می‌کنیم. شیار اول ورودی را در بر می‌گیرد، شیار دوم مکان هد خواندن از ورودی، شیار سوم محتوای نوار، و شیار چهارم مکان هد خواندن نوشتن بر روی نوار را در بر می‌گیرد. پس ماشین تورینگ آفلاین را می‌توان توسط یک ماشین تورینگ استاندارد (با نوار چهار شیاری) شبیه‌سازی کرد. بدینگونه می‌توان نشان داد که این ماشین نیز هم‌ارز ماشین تورینگ استاندارد است.





ماشین‌های تورینگ با چند نوار

– یک ماشین تورینگ با چند نوار [1] و به طور مشخص با n نوار را می‌توان توسط یک ماشین تورینگ با یک نوار ۲n شیاری شبیه‌سازی کرد. به طوری که شیار n محتوای نوار n ام، و شیار ۲n موقعیت هد در شیار n ام را نشان می‌دهد.



[1] multitape Turing machine

ماشین‌های تورینگ با نوار چندبعدی

- یک ماشین تورینگ با یک نوار چندبعدی [1] را می‌توان توسط یک ماشین تورینگ با یک نوار دوشیاری شبیه‌سازی کرد به طوری که شیار اول محتوای نوار چندبعدی و شیار دوم مکان محتوا را در نوار چندبعدی مشخص می‌کند.
- به روشی دیگر می‌توان یک ماشین تورینگ با نوار n بعدی را با یک ماشین تویِنگ با یک نوار شامل $n + ۱$ شیار شبیه‌سازی کرد، به طوری که شیار اول محتوای نوار n بعدی و n شیار دیگر هر کدام مختصات یکی از ابعاد را در برمی‌گیرد.

[1] multidimentional Turing machine

ماشین‌های تورینگ غیرقطعی

- یک ماشین تورینگ غیرقطعی ماشینی همانند ماشین تورینگ قطعی است، با این تفاوت که تابع گذار آن به صورت زیر تعریف می‌شود: $\delta : Q \times \Gamma \to ۲^{Q \times \Gamma \times \{L,R\}}$
- این بدین معنی است که در هر حرکت، ماشین می‌تواند یکی از گذارهای ممکن را انتخاب کند.
- یک ماشین تورینگ غیرقطعی رشتهٔ w را می‌پذیرد اگر دنباله‌ای از حرکت‌ها وجود داشته باشد که ماشین را با شروع از حالت آغازی و خواندن رشتهٔ w به یک حالت پایانی ببرد.
- با خواندن رشتهٔ w ممکن است دنباله‌های دیگری از حرکت‌ها وجود داشته باشند که ماشین را به حلقهٔ بی‌پایان یا حالت غیرپایانی ببرند.

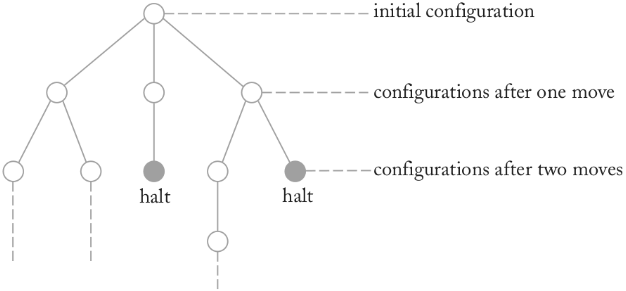
ماشین‌های تورینگ غیرقطعی

- برای اینکه نشان دهیم قدرت ماشین تورینگ غیرقطعی به اندازهٔ قدرت ماشین تورینگ قطعی است باید بتوانیم برای هر ماشین غیرقطعی یک ماشین قطعی ارائه کنیم.

ماشین‌های تورینگ غیرقطعی

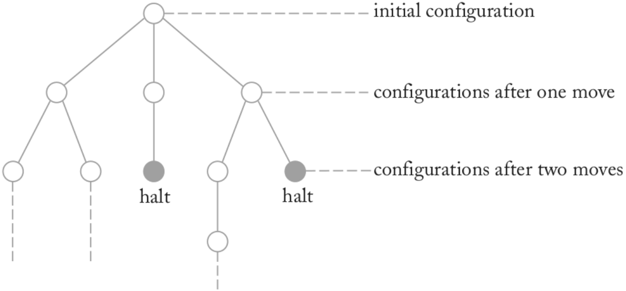
- فرض کنید می‌خواهیم یک الگوریتم (که در واقع یک ماشین تورینگ استاندارد است) را توصیف کنیم که یک ماشین تورینگ غیرقطعی را اجرا می‌کند.

- در هر حرکت از ماشین غیرقطعی (که در آن چندین انتخاب وجود دارد)، الگوریتم مورد نظر، به ازای هر انتخاب، ماشین تورینگ غیرقطعی را در قسمتی از نوار کپی می‌کند و هر کپی از ماشین را جداگانه اجرا می‌کند. اگر یکی از کپی‌ها به حالت پایانی رسید و توقف کرد، الگوریتم مورد نظر رشته ورودی را می‌پذیرد.



ماشین‌های تورینگ غیرقطعی

- اگر تعداد انتخاب‌های یک ماشین در هر حرکت حداکثر k باشد، آنگاه حداکثر تعداد پیکربندی های ایجاد شده بعد از n حرکت برابر است با k^n .



ماشین‌های تورینگ غیرقطعی

- حال که برای اجرای یک ماشین تورینگ غیرقطعی الگوریتمی ارائه کردیم، همین الگوریتم را می‌توانیم به صورت یک ماشین تورینگ قطعی ارائه کنیم. این ماشین تورینگ قطعی معادل ماشین تورینگ غیرقطعی مورد نظر است.
- ماشین تورینگ قطعی طراحی شده، در هر حرکت توصیف لحظه‌ای ماشین غیرقطعی را روی نوار خود می‌نویسد. سپس حرکت‌های مختلف را به ازای انتخاب‌های ماشین غیرقطعی محاسبه می‌کند و این روند را ادامه می‌دهد تا به یک حالت پایانی برسد.
- پس این ماشین تورینگ قطعی، معادل ماشین تورینگ غیرقطعی است و بنابراین به ازای هر ماشین غیرقطعی یک ماشین قطعی وجود دارد و این دو طبقه از ماشین‌ها هم‌ارز یکدیگرند.

ماشین‌های تورینگ غیرقطعی

- یک ماشین تورینگ غیرقطعی زبان L را می‌پذیرد [1]، اگر به ازای هر جملهٔ $w \in L$ یک دنباله از حرکت‌ها وجود داشته باشد (یک پیکربندی برای ماشین پس از چند گام اجرا وجود داشته باشد) که رشتهٔ w را بپذیرد. ممکن است دنباله‌ای از حرکت‌ها در حلقهٔ بی‌پایان بیافتند و دنباله‌ای دیگر بدون پذیرفتن متوقف شود، که چنین رفتاری در پذیرفتن رشته تأثیری ندارد و فقط وجود داشتن دنباله‌ای که به پذیرفتن رشته ختم می‌شود برای پذیرفتن رشته اهمیت دارد.

- یک ماشین تورینگ غیرقطعی می‌تواند زبان L را تصمیم‌بگیرد [2]، (بر روی زبان L تصمیم‌پذیر است) اگر به ازای هر جملهٔ $w \in \Sigma^*$ دنباله‌ای از حرکت‌ها وجود داشته باشد که یا جمله را بپذیرد یا جمله را رد کند. (برای رد کردن جمله، هیچ دنباله‌ای نمی‌تواند به حلقهٔ بی‌پایان برود، زیرا در حلقهٔ بی‌پایان نمی‌توانیم مطمئن باشیم که جمله پذیرفته نخواهد شد.)

[1] accept
[2] decide

ماشین تورینگ جهانی

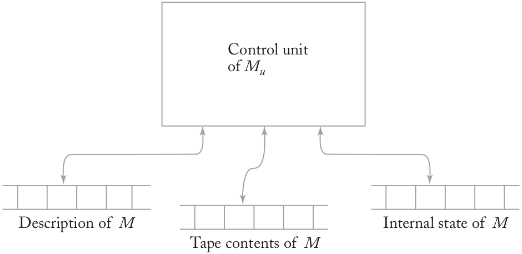
- ماشین تورینگ استاندارد معادل یک الگوریتم است که یک تابع خاص را محاسبه می‌کند.
- بنابراین ماشین تورینگ استاندارد را نمی‌توان به عنوان یک محاسبه‌گر جامع معادل یک کامپیوتر دیجیتال دانست.
- بدین جهت ماشین تورینگ جهانی [1] را تعریف می‌کنیم که یک محاسبه‌گر جامع است و می‌تواند هر محاسبه‌ای را بر روی هر جمله‌ای انجام دهد.
- یک ماشین تورینگ جهانی M_u ماشینی است که با گرفتن توصیف یک ماشین تورینگ M و رشتهٔ w محاسبات M بر روی w را شبیه‌سازی می‌کند.

[1] universal Turing machine

ماشین تورینگ جهانی

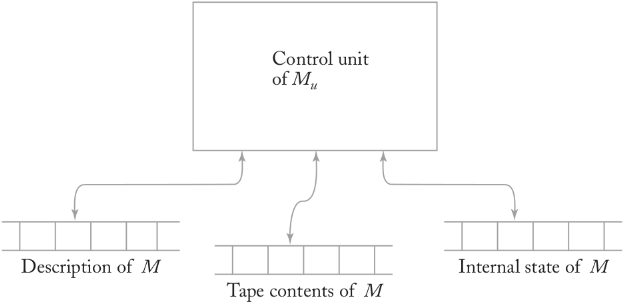
- یک ماشین تورینگ جهانی سه نوار دارد. بر روی نوار اول توصیف یک ماشین تورینگ را دریافت می‌کند، بر روی نوار دوم رشتهٔ ورودی را دریافت می‌کند، و بر روی نوار سوم حالت داخلی ماشین ورودی را ذخیره می‌کند.

- بنابراین با استفاده از نوار دوم و سوم، ماشین تورینگ جهانی توصیف لحظه‌ای ماشین تورینگ دریافتی خود را تولید می‌کند، سپس با استفاده از توابع گذار نوشته شده بر روی نوار اول، ماشین تورینگ جهانی، ماشین تورینگ دریافتی را اجرا می‌کند.



ماشین تورینگ جهانی

- پس همانطور که یک کامپیوتر دیجیتال، یک الگوریتم و مقدار ورودی آن را دریافت می‌کند و خروجی الگوریتم را محاسبه می‌کند، یک ماشین تورینگ جهانی نیز، یک ماشین تورینگ و یک رشتهٔ ورودی را دریافت کرده و محاسبات ماشین تورینگ را بر روی رشتهٔ ورودی محاسبه می‌کند.



ماشین تورینگ جهانی

- ماشین تورینگ ورودی را می‌توان به صورت یک رشته شامل صفر و یک دریافت کرد.
- بدین منظور باید حالت‌های ماشین را با اعداد دودویی و الفبای نوار را نیز با اعداد دودویی کدگذاری کنیم. بنابراین یک تابع گذار برای ماشین تورینگ می‌تواند با یک رشتهٔ دودویی مشخص شود و کل ماشین تورینگ با رشته های دودویی از توابع گذار.
- برای مثال تابع گذار $\delta(q_۱, a_۲) = (q_۲, a_۳, L)$ را می‌توان به صورت ۱۰۱۱۰۱۱۰۱۱۱۰۱۰ کدگذاری کرد به طوری که همهٔ صفر ها علامت فاصله هستند، اولین ۱ کدگذاری برای $q_۱$، ۱۱ بعدی کدگذاری برای $a_۲$، ۱۱ بعدی کدگذاری برای $q_۲$، ۱۱۱ بعدی کدگذاری برای $a_۳$ و ۱ نهایی کدگذاری برای L می‌باشد.

ماشین تورینگ جهانی

- از آنجایی که مجموعهٔ همهٔ رشته‌های دودویی قابل شمارش است، پس مجموعهٔ همهٔ ماشین‌های تورینگ نیز قابل شمارش است.
- برای شمارش همهٔ ماشین‌های تورینگ، با شمارش رشته‌های دودویی شروع می‌کنیم. به ازای هر رشتهٔ دودویی در $\{۰, ۱\}^+$، بررسی می‌کنیم آیا رشتهٔ دودویی معادل یک ماشین تورینگ است یا خیر. اگر جواب مثبت بود، ماشین تورینگ مورد نظر را در مجموعهٔ همهٔ ماشین‌های تورینگ شمارش می‌کنیم، در غیر اینصورت به رشتهٔ بعدی می‌رویم.

ماشین‌های کراندار خطی

- اگر ماشین تورینگ را محدود کنیم به طوری که نوار آن مانند یک پشته عمل کند از قدرت آن می‌کاهیم و آن را به یک ماشین پشته‌ای تبدیل می‌کنیم.

- اگر نوار نامحدود را به نوار محدود تبدیل کنیم، یک ماشین متناهی به دست می‌آوریم.

- اگر ماشین را محدود کنیم به طوری که فقط از قسمتی از نوار استفاده کند که رشتهٔ ورودی بر روی آن نوشته شده است، یک ماشین کراندار خطی [1] به دست آوریم.

- ماشین کراندار خطی مانند ماشین تورینگ یک نوار نامحدود دارد، با این تفاوت که فقط از قسمتی از نوار استفاده می‌کند که ورودی بر روی آن قرار دارد.

- بنابراین رشتهٔ ورودی با دو نشانهٔ سمت چپ و سمت راست [2] مشخص می‌شود و ماشین نمی‌تواند از آن محدوده خارج شود.

[1] linear bounded automata (lba)

[2] left-end and right-end marker

ماشین‌های کراندار خطی

- ماشین کراندار خطی، یک ماشین تورینگ غیر قطعی $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ است، با این تفاوت که الفبای Σ دو نماد ویژه برای نشانهٔ سمت چپ و راست رشته ($[,]$) دارد.
- همهٔ اعضای $\delta(q_i, [)$ به شکل $(q_j, [, R)$ و همهٔ اعضای $\delta(q_i,])$ به شکل $(q_j,], L)$ هستند.
- رشتهٔ w توسط یک ماشین کراندار خطی پذیرفته می‌شود اگر دنباله‌ای از حرکت‌ها وجود داشته باشد به طوری که $q[w] \overset{*}{\vdash} [x_1 q_f x_2]$ به ازای $q_f \in F$ و $x_1, x_2 \in \Gamma^*$.
- یک زبان پذیرفته شده توسط یک ماشین کراندار خطی مجموعه‌ای از تمام رشته‌های پذیرفته شده توسط آن ماشین است.

ماشین‌های کراندار خطی

- می‌توان نشان داد که قدرت ماشین کراندار خطی بیشتر از ماشین پشته‌ای است زیرا زبان $\{a^n b^n c^n : n \geq ۱\}$ را می‌پذیرد.
- نشان داده شده است که قدرت ماشین کراندار خطی کمتر از ماشین تورینگ است و طبقهٔ زبان‌هایی که می‌پذیرد زیرمجموعهٔ زبان‌هایی است که توسط ماشین‌های تورینگ پذیرفته می‌شوند. در آینده با این دسته از زبان‌ها آشنا می‌شویم.

سلسله‌مراتب زبان‌ها

سلسله‌مراتب زبان‌ها

- حال که با چندین دسته از زبان‌ها و ماشین‌ها آشنا شدیم به دسته‌بندی آنها می‌پردازیم.
- ماشین‌های تورینگ دسته‌ای از زبان‌ها را شناسایی می‌کنند که به آنها زبان‌های شمارش‌پذیر بازگشتی [1] می‌گوییم و ماشین‌های کراندار خطی زبان‌های حساس به متن [2] را شناسایی می‌کنند.
- بدین ترتیب چهار دسته از زبان‌ها را بررسی کردیم: زبان‌های منظم، مستقل از متن، حساس به متن، و شمارش‌پذیر بازگشتی.
- ماشین‌هایی که این چهار دسته از زبان‌ها را شناسایی می‌کنند به ترتیب ماشین‌های حالات متناهی، پشته‌ای، کراندار خطی، و تورینگ هستند.
- همچنین خواهیم دید که تعداد زبان‌ها از تعداد ماشین‌های تورینگ بیشتر است، پس باید زبان‌هایی وجود داشته باشند که هیچ ماشین تورینگی برای آنها وجود ندارد.

[1] recursively enumerable language
[2] context-sensitive language

زبان‌های شمارش‌پذیر بازگشتی

- زبان L یک زبان شمارش‌پذیر بازگشتی [1] یا تشخیص پذیر [2] نامیده می‌شود، اگر یک ماشین تورینگ وجود داشته باشد که آن را بپذیرد.

- بنابراین اگر ماشین تورینگ M وجود داشته باشد به طوری که برای هر $w \in L$ داشته باشیم
$q_0 w \vdash_M^* x_1 q_f x_2$ به طوری که q_f یک حالت پایانی باشد، آنگاه زبان L را یک زبان شمارش‌پذیر بازگشتی می‌نامیم.

- برای رشته‌هایی که توسط ماشین M پذیرفته نمی‌شوند، ماشین ممکن است به حالت غیرپایانی برود یا در یک حلقهٔ بی‌پایان بیافتد.

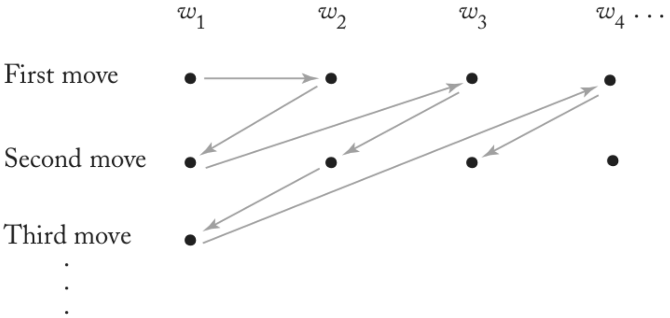
---
[1] recursively enumerable language
[2] recognizable or Turing-recongnizable

زبان‌های شمارش‌پذیر بازگشتی

- زبان‌های شمارش‌پذیر بازگشتی به این دلیل شمارش‌پذیر خوانده می‌شوند که روشی برای شمارش جمله‌های آنها وجود دارد.
- برای شمارش جمله‌های آنها، همهٔ رشته‌های الفبا را مرتب می‌کنیم. به ترتیب رشتهٔ اول را به ماشین تورینگی که برای آن زبان طراحی شده برای اجرا می‌دهیم، ولی از آنجایی که ماشین ممکن است بر روی رشته هیچ‌گاه توقف نکند، بنابراین فقط یک قدم را بر روی رشتهٔ اول اجرا می‌کنیم و سپس به رشتهٔ دوم می‌رویم، یک قدم را اجرا می‌کنیم و الی آخر. این روند را ادامه می‌دهیم و به ترتیب هر رشته‌ای را که توسط ماشین پذیرفته شد، شمارش می‌کنیم.

زبان‌های شمارش‌پذیر بازگشتی

- زبان‌های شمارش‌پذیر بازگشتی به این دلیل شمارش‌پذیر خوانده می‌شوند که روشی برای شمارش جمله‌های آنها وجود دارد.



زبان‌های شمارش‌پذیر بازگشتی

- زبان‌های شمارش‌پذیر بازگشتی به این دلیل بازگشتی خوانده می‌شوند که در گذشته مباحث نظریهٔ محاسبات در حوزه‌ای به نام نظریهٔ بازگشت [1] مطالعه می‌شدند و تعاریف بازگشتی در تعریف محاسبات نقش مهمی داشتند.
- در حال حاضر نظریهٔ محاسبه‌پذیری [2] و نظریهٔ بازگشت به یک معنی به کار می‌روند.
- نظریهٔ محاسبات شاخه‌ای از منطق و علوم کامپیوتر است که به بررسی محاسبه‌پذیری توابع می‌پردازد. همچنین توابع محاسبه‌پذیر را در نظریهٔ محاسبات از حیث زمان و حافظهٔ مورد نیاز آنها دسته‌بندی می‌کنیم.

[1] recursion theory
[2] computability theory

زبان‌های بازگشتی

- زبان L بر روی الفبای Σ یک زبان بازگشتی [1] یا تصمیم‌پذیر [2] نامیده می‌شود، اگر یک ماشین تورینگ وجود داشته باشد که L را بپذیرد و برای هر رشتهٔ $w \in \Sigma^+$ توقف کند.
- به عبارت دیگر یک زبان بازگشتی است، اگر و تنها اگر برای آن الگوریتمی وجود داشته باشد که تعیین کند آیا هر رشتهٔ داده شده عضو آن زبان است یا خیر.

---
[1] recursive language
[2] decidable or Turing-decidable

زبان‌های شمارش‌پذیر بازگشتی

- زبان‌های بازگشتی نیز شمارش‌پذیر هستند.
- برای شمارش جمله‌های آنها، همهٔ رشته‌های الفبا را مرتب می‌کنیم. به ترتیب همهٔ رشته‌ها را به ماشین تورینگی که برای آن زبان طراحی شده می‌دهیم. اگر رشته پذیرفته شد، آن را شمارش می‌کنیم و اگر پذیرفته نشد، به رشتهٔ بعدی می‌رویم و الی آخر.

تشخیص‌پذیری و تصمیم‌پذیری

- برای مثال مسألۀ دهم هیلبرت می‌پرسد: آیا الگوریتمی وجود دارد که بتواند به ازای هر چندجمله‌ای تعیین کند آیا آن چندجمله‌ای ریشه‌های صحیح دارد یا خیر؟
- نشان داده شده است که هیچ الگوریتم تصمیم‌پذیری (ماشین تورینگی که روی همۀ ورودی‌ها متوقف شود) برای این مسأله وجود ندارد.
- به عبارت دیگر زبان (مجموعۀ) همۀ چندجمله‌ای ها با ریشۀ صحیح، یک زبان تشخیص‌پذیر (شمارش‌پذیر بازگشتی) است و تصمیم‌پذیر (بازگشتی) نیست.

- اگر زبان D را چنین تعریف کنیم:
$D = \{p :$ یک چندجمله‌ای با ریشه‌های صحیح است $p\}$
مسأله دهم هیلبرت می‌پرسد آیا D تصمیم‌پذیر است؟
- زبان D تصمیم‌پذیر نیست، ولی تشخیص‌پذیر است.
- به عبارت دیگر می‌توانیم به ازای یک چندجمله‌ای داده شدهٔ p ، همهٔ اعداد صحیح را بررسی کنیم. اگر به ازای تعدادی عدد صحیح ریشه‌ای برای چندجمله‌ای p پیدا شد، p پذیرفته می‌شود، ولی اگر ریشه‌ای برای p پیدا نشد، الگوریتم در حلقهٔ بی‌پایان می‌افتد و نمی‌توان تصمیم گرفت که ریشهٔ صحیح وجود ندارد.

زبان‌های غیر بازگشتی

- حال می‌خواهیم نشان دهیم زبان‌هایی وجود دارند که تشخیص پذیر نیستند.
- قضیه: اگر S یک مجموعهٔ نامتناهی شمارا (شمارش‌پذیر) باشد، آنگاه مجموعهٔ توانی ۲^S ناشمارا (شمارش‌ناپذیر) است.

زبان‌های غیر بازگشتی

- قضیه: اگر S یک مجموعهٔ نامتناهی شمارا (شمارش‌پذیر) باشد، آنگاه مجموعهٔ توانی 2^S ناشمارا (شمارش‌ناپذیر) است.

- اثبات: فرض کنید مجموعهٔ t_i یکی از اعضای مجموعهٔ توانی 2^S باشد. می‌نویسیم $t_i = 110...$ اگر t_i اولین عضو و دومین عضو S را در شامل شود اما سومین عضو S را شامل نشود، و الی آخر.

- حال فرض کنید مجموعهٔ همهٔ زیرمجموعه‌های S یعنی 2^S شمارا باشد. آنگاه می‌توانیم با استفاده از جدولی مشابه جدول زیر اعضای آن را بشماریم.

t_1	1	0	0	0	0	...
t_2	1	1	0	0	0	...
t_3	1	1	0	1	0	...
t_4	1	1	0	0	1	...
$\vdots$					$\ddots$	

زبان‌های غیر بازگشتی

- حال رشته‌ای که در جدول زیر روی قطر است را در نظر بگیرید. همهٔ صفرهای آن را به یک و همهٔ یک‌های آن را به صفر تبدیل کنید.

- اگر ۲^S شمارا باشد، آنگاه باید متمم رشته‌ای که روی قطر قرار دارد نیز در این جدول شمارش شود و یکی از t_i ها باشد. اما این رشته نمی‌تواند هیچ‌کدام از t_i ها باشد، زیرا نه $t_۱$ است (چون نماد اول آن با نماد اول $t_۱$ متفاوت است) و نه $t_۲$ است (چون نماد دوم آن با نماد دوم $t_۲$ متفاوت است) و الی آخر.

- تناقض به وجود آمده نشان می‌دهد که فرض اولیه مبنی بر شمارا بودن ۲^S نادرست بوده است.

t_1	1	0	0	0	0	. . .
t_2	1	1	0	0	0	. . .
t_3	1	1	0	1	0	. . .
t_4	1	1	0	0	1	. . .
⋮				⋱		

زبان‌های غیر بازگشتی

- قضیه: بر روی الفبای Σ زبان‌هایی وجود دارند که شمارش‌پذیر بازگشتی (تشخیص‌پذیر) نیستند.
- اثبات: یک زبان بر روی الفبای Σ یک زیرمجموعه است از مجموعهٔ Σ^*. بنابراین مجموعهٔ همهٔ زبان‌ها برابر است با مجموعهٔ توانی 2^{Σ^*}. این مجموعه طبق قضیه‌ای که بیان شد، ناشمارا است. پس مجموعهٔ همهٔ زبان‌ها بر روی یک الفبای معین ناشمارا است.
  از طرف دیگر اثبات کردیم که مجموعهٔ همهٔ ماشین‌های تورینگ شمارا است. پس تعداد زبان‌ها از تعداد ماشین‌های تورینگ بیشتر است و بنابراین زبان‌هایی وجود دارند که برای آنها هیچ ماشین تورینگی وجود ندارد.
- اکنون باید یک زبان پیدا کنیم که شمارش‌پذیر بازگشتی نباشد.

زبان‌های غیر بازگشتی

- قضیه: یک زبان تشخیص‌پذیر (شمارش‌پذیر بازگشتی) وجود دارد که متمم آن تشخیص‌پذیر نیست.
- اثبات: مجموعهٔ همهٔ ماشین‌های تورینگ بر روی الفبای $\Sigma = \{a\}$ را در نظر بگیرید.
- این مجموعه شمارا است (قبلا روشی برای شمارش همهٔ ماشین‌های تورینگ بیان کردیم). پس مجموعهٔ شمارای $\{M_۱, M_۲, \cdots\}$ را در نظر می‌گیریم. برای هر یک از این ماشین‌های تورینگ M_i یک زبان تشخیص‌پذیر (شمارش‌پذیر بازگشتی) $L(M_i)$ وجود دارد. همچنین برای هر زبان تشخیص‌پذیر بر روی Σ یک ماشین تورینگ وجود دارد.
- حال زبان $L = \{a^i : a^i \in L(M_i)\}$ را در نظر بگیرید.
به ازای هر $i \geq ۱$، رشتهٔ a^i عضو زبان L است اگر و فقط اگر $a^i \in L(M_i)$.

زبان‌های غیر بازگشتی

- ابتدا باید نشان دهیم که L یک زبان تشخیص‌پذیر است.
- می‌توانیم ماشین تورینگ جهانی M_u را طراحی کنیم که همهٔ ماشین‌های تورینگ را شمارش می‌کند. ماشین M_u، رشتهٔ a^i را به ماشین M_i به عنوان ورودی می‌دهد. اگر رشته توسط ماشین M_i پذیرفته شد، آنگاه توسط ماشین M_u نیز پذیرفته می‌شود و بنابراین رشته عضو زبان L است. پس برای زبان L ماشین M_u را پیدا کردیم، و در نتیجه زبان L یک زبان تشخیص‌پذیر است.

زبان‌های غیر بازگشتی

- حال متمم زبان L را در نظر بگیرید: $\overline{L} = \{a^i : a^i \notin L(M_i)\}$

- نشان می‌دهیم که زبان $\overline{L}$ شمارش‌پذیر بازگشتی (تشخیص‌پذیر) نیست.

- فرض کنید $\overline{L}$ تشخیص پذیر باشد. پس باید برای آن ماشین تورینگ M_k وجود داشته به طوری که $\overline{L} = L(M_k)$.

- حال رشتهٔ a^k را در نظر بگیرید. این رشته عضو زبان L است یا زبان $\overline{L}$ ؟

- فرض کنید رشتهٔ $a^k \in \overline{L}$ آنگاه داریم $a^k \in L(M_k)$ زیرا فرض کردیم $\overline{L} = L(M_k)$. از طرف دیگر طبق تعریف $\overline{L}$، رشتهٔ a^k عضو زبان $\overline{L}$ است اگر $a^k \notin L(M_k)$. پس به تناقض می‌رسیم و بنابراین $a^k \notin \overline{L}$.

- حال فرض کنید $a^k \in L$ پس $a^k \notin \overline{L}$ پس $a^k \notin L(M_k)$. اما در این صورت طبق تعریف $\overline{L}$، اگر $a^k \notin L(M_k)$ آنگاه داریم $a^k \in \overline{L}$. پس به تناقض رسیدیم و فرض اولیه نادرست بوده و برای تشخیص $\overline{L}$ هیچ ماشینی وجود ندارد و بنابراین $\overline{L}$ نمی‌تواند تشخیص‌پذیر باشد.

زبان‌های بازگشتی

- حال می‌خواهیم نشان دهیم زبان‌های بازگشتی شمارش پذیر (تشخیص‌پذیر) وجود دارند که بازگشتی (تصمیم‌پذیر) نیستند.

زبان‌های بازگشتی

- قضیه: اگر L یک زبان تصمیم‌پذیر باشد، متمم آن نیز تصمیم‌پذیر است.
- اگر L تصمیم‌پذیر باشد پس به ازای هر رشته با استفاده از یک ماشین تورینگ می‌توان تعیین کرد که آیا رشته عضو L است یا خیر. همین ماشین تورینگ را برای متمم L نیز می‌توانیم به کار ببریم. اگر رشته‌ای توسط این ماشین پذیرفته شد، رشته عضو $\overline{L}$ نیست، در غیر اینصورت رشته عضو $\overline{L}$ است.

زبان‌های بازگشتی

- قضیه: زبان‌های تشخیص‌پذیر وجود دارند که تصمیم‌پذیر نیستند. به عبارت دیگر خانوادۀ زبان‌های تصمیم‌پذیر زیرمجموعۀ خانوادۀ زبان‌های تشخیص‌پذیر است.
- اثبات: زبان L که قبلا به صورت $L = \{a^i : a^i \in L(M_i)\}$ توصیف کردیم در نظر بگیرید. ثابت کردیم این زبان تشخیص‌پذیر است. حال فرض کنید این زبان تصمیم‌پذیر باشد. پس متمم آن نیز باید تصمیم‌پذیر باشد. ولی ثابت کردیم متمم آن غیربازگشتی است. پس زبانی وجود دارد که تشخیص‌پذیر است ولی تصمیم‌پذیر نیست.

- قضیه: اگر L و متمم آن $\overline{L}$ هر دو تشخیص‌پذیر باشند، آنگاه هر دو تصمیم‌پذیرند.
- اثبات: فرض کنیم ماشین تورینگ M زبان L و ماشین تورینگ $\widehat{M}$ زبان $\overline{L}$ را می‌پذیرد. به ازای هر رشتهٔ زبان L می‌توانیم یک ماشین تورینگ طراحی کنیم که رشته را به ماشین M و $\widehat{M}$ می‌دهد. اگر M رشته را پذیرفت، رشته پذیرفته می‌شود و اگر $\widehat{M}$ رشته را پذیرفت، رشته رد می‌شود. پس L یک زبان تصمیم‌پذیر است. به همین ترتیب $\overline{L}$ نیز تصمیم پذیر است.

گرامرهای بدون محدودیت

- گرامر $G = (V, T, S, P)$ را یک گرامر بدون محدودیت [1] می‌گوییم اگر همهٔ قوانین تولید آن به صورت $u \to v$ باشد، به طوری که $u \in (V \cup T)^+$ و $v \in (V \cup T)^*$.
- در یک گرامر بدون محدودیت، هیچ محدودیتی برای شکل قوانین در نظر گرفته نشده است. هر تعداد متغیر و نماد پایانی می‌توانند در سمت چپ و راست یک قانون قرار داشته باشند و تنها محدودیت این است که سمت چپ قانون تهی نباشد.

[1] unrestricted grammar

گرامرهای بدون محدودیت

- هر زبان تولید شده توسط یک گرامر بدون محدودیت یک زبان شمارش‌پذیر بازگشتی (تشخیص‌پذیر) است.
- همچنین برای هر زبان تشخیص‌پذیر L ، یک گرامر بدون محدودیت G وجود دارد، به طوری که $L = L(G)$.
- روشی برای شبیه‌سازی یک فرایند اشتقاق توسط یک ماشین تورینگ وجود دارد که در اینجا به آن نمی‌پردازیم.

گرامرهای بدون محدودیت

- یک گرامر بدون محدودیت برای زبان $L = \{a^{2^k} : k \geq 0\}$ بنویسید.

- یک گرامر بدون محدودیت برای زبان $L = \{a^{2^k} : k \geq 0\}$ بنویسید.
- می‌توانیم جملات L را به صورت بازگشتی تعریف کنیم. می‌دانیم $a \in L$ و به ازای هر $n \geq 1$ اگر $a^n \in L$ آنگاه $a^{2n} \in L$.
- پس باید گرامری بنویسیم که بتواند در هر مرحله تعداد نمادهای a را دو برابر کند.

گرامرهای بدون محدودیت

- پس باید گرامری بنویسیم که بتواند در هر مرحله تعداد نمادهای a را دو برابر کند.
- برای این کار از یک متغیر D استفاده می‌کنیم که می‌تواند هر نماد a در یک صورت جمله‌ای را با دو نماد a جایگزین کند.
- فرض کنید در یک صورت جمله‌ای تعدادی a تولید شده است، آنگاه متغیر D از چپ شروع می‌کند و هر a را با دو a جایگزین می‌کند.
- به عبارت دیگر ما به دنبال قانونی می‌گردیم که اشتقاق زیر را تولید کند:
$\mathrm{Daaaa} \Rightarrow \mathrm{aaDaaa} \Rightarrow \mathrm{aaaaDaa} \Rightarrow \mathrm{aaaaaaDa} \Rightarrow \mathrm{aaaaaaaaD}$
- این قانون را به صورت زیر می‌نویسیم:
$\mathrm{Da} \rightarrow \mathrm{aaD}$

گرامرهای بدون محدودیت

- حال به دنبال یک قانون اولیه می‌گردیم که برای ما یک نماد a تولید کند و همچنین بتواند در هر بار یک متغیر D برای دو برابر کردن تعداد a ها تولید کند. این قوانین را بدین صورت می‌نویسیم:
$S \rightarrow LaR$
$L \rightarrow LD$

- سپس به دنبال قانونی می‌گردیم که متغیر D را پس از این که تعداد نمادهای a را دو برابر کرد، حذف کند. این قانون را بدین صورت می‌نویسیم:
$DR \rightarrow R$

- در پایان باید متغیرهای L و R را حذف کنیم:
$L \rightarrow \lambda$
$R \rightarrow \lambda$

گرامرهای بدون محدودیت

- برای به دست آوردن جملهٔ aaaa فرایند اشتقاقی به صورت زیر داریم:
$S \Rightarrow LaR \Rightarrow LDaR \Rightarrow LaaDR \Rightarrow LaaR \Rightarrow LDaaR \Rightarrow$
$LaaDaR \Rightarrow LaaaaDR \Rightarrow LaaaaR \Rightarrow aaaaR \Rightarrow aaaa$

گرامرهای بدون محدودیت

- یک گرامر بدون محدودیت برای زبان $L = \{ww : w \in \{a, b\}^*\}$ بنویسید.

گرامرهای بدون محدودیت

- یک گرامر بدون محدودیت برای زبان $L = \{ww : w \in \{a, b\}^*\}$ بنویسید.
- با تولید رشته‌های ww^R آشنا هستیم، پس ابتدا گرامری می‌نویسیم که رشته‌هایی را تولید کند که نیمهٔ اول آنها w و نیمهٔ دوم آنها معکوس w باشد، با این تفاوت که در نیمهٔ دوم، به جای نمادهای پایانی، متغیر داریم.
متغیر A به جای نماد a و متغیر B به جای نماد b . بنابراین ابتدا یک صورت جمله‌ای به شکل wW^R تولید می‌کنیم، به طوری که W فقط از متغیرها تشکیل شده است.
- برای این کار قوانینی به صورت زیر می‌نویسیم:
$S \rightarrow T]$
$T \rightarrow aTA|bTB|[R$

گرامرهای بدون محدودیت

- پس در یک دنباله از اشتقاق‌ها خواهیم داشت: $S \overset{*}{\Rightarrow} w[RW^R]$.

- سپس باید صورت جمله‌ای $[RW^R]$ را تبدیل به w کنیم.

- در هر بار با استفاده از یک متغیر به پایان رشتهٔ W^R می‌رویم. آخرین متغیر این رشته را تشخیص می‌دهیم. اگر آخرین متغیر در رشته A بود، آن را به L_a تبدیل می‌کنیم و اگر B بود، آن را به L_b تبدیل می‌کنیم. سپس متغیر L_a یا L_b را به ابتدای W^R می‌آوریم و آن را از براکت خارج کرده و به نماد پایانی تبدیل می‌کنیم.

- برای مثال برای به دست آوردن جملهٔ aabaaaba فرایند اشتقاق زیر را داریم:

$$S \Rightarrow T] \overset{*}{\Rightarrow} aabaTABAA] \Rightarrow aaba[RABAA]$$
$$\overset{*}{\Rightarrow} aaba[ABAAR] \Rightarrow aaba[ABAL_a] \overset{*}{\Rightarrow} aaba[L_aABA] \Rightarrow aabaa[RABA]$$
$$\overset{*}{\Rightarrow} aabaa[ABAR] \Rightarrow aabaa[ABL_a] \overset{*}{\Rightarrow} aabaa[L_aAB] \Rightarrow aabaaa[RAB]$$
$$\overset{*}{\Rightarrow} aabaaaba[R] \Rightarrow aabaaaba$$

- این گرامر را می‌توانیم به صورت زیر بنویسیم:
- $V = \{S, T, A, B, R, L_a, L_b, [,]\}$

$$S \to T] \ , \ T \to aTA|bTB|[R$$
$$RA \to AR \ , \ RB \to BR$$
$$AR] \to L_a] \ , \ BR] \to L_b]$$
$$AL_a \to L_aA \ , \ AL_b \to L_bA \ , \ BL_a \to L_aB \ , \ BL_b \to L_bB$$
$$[L_a \to a[R \ , \ [L_b \to b[R$$
$$[R] \to \lambda$$

گرامرهای حساس به متن

- گرامر $G = (V, T, S, P)$ حساس به متن گفته می‌شود، اگر همهٔ قوانین آن به صورت $x \to y$ باشند، به طوری که $x, y \in (V \cup T)^+$ و $|x| \leq |y|$.

- این گرامرها حساس به متن نامیده می‌شوند، زیرا اثبات شده است که چنین گرامرهایی را می‌توان به یک فرم نرمال تبدیل کرد به طوری که همهٔ قوانین آن به شکل $xAy \to xvy$ باشند. بنابراین این قانون مانند قانون تولید $A \to v$ است با این تفاوت که متن (یعنی نمادهای اطراف A در یک صورت جمله‌ای در فرایند اشتقاق) در اعمال قانون مؤثر است، بر خلاف گرامرهای مستقل از متن که بدون در نظر گرفتن متن، قوانین را اعمال می‌کند.

زبان‌های حساس به متن

- زبان L حساس به متن گفته می‌شود اگر یک گرامر حساس به متن G برای آن وجود داشته باشد، به طوری که $L = L(G)$.

- از آنجایی که گرامر حساس به متن اجازهٔ تولید رشتهٔ تهی را نمی‌دهد، یک استثنا در نظر می‌گیریم برای حالتی که در زبان مورد نظر L نیاز به تولید رشتهٔ تهی داشته باشیم، بنابراین $L = L(G) \cup \{\lambda\}$.

زبان‌های حساس به متن

- زبان $L = \{a^n b^n c^n : n \geq ۱\}$ یک زبان حساس به متن است. برای آن یک گرامر حساس به متن بنویسید.

زبان‌های حساس به متن

- زبان $L = \{a^n b^n c^n : n \geq ۱\}$ یک زبان حساس به متن است. برای آن یک گرامر حساس به متن بنویسید.
- این گرامر را به دو روش می‌نویسیم.
- در روش اول ابتدا یک صورت جمله‌ای به شکل $a^n(BC)^n$ تولید می‌کنیم. برای این کار از قوانین تولید $S \rightarrow aSBC|aBC$ استفاده می‌کنیم.
- سپس همهٔ متغیرهای B را به قبل از متغیرهای C انتقال می‌دهیم. برای این کار از قانون زیر استفاده می‌کنیم:
$$CB \rightarrow BC$$
- در پایان باید همهٔ متغیرها را به نمادهای پایانی تبدیل کنیم. برای این کار از قوانین زیر استفاده می‌کنیم:
$$aB \rightarrow ab \ , \ bB \rightarrow bb \ , \ bC \rightarrow bc \ , \ cC \rightarrow cc$$
- دقت کنید که در اینجا از قوانین $B \rightarrow b$ و $C \rightarrow c$ استفاده نمی‌کنیم، زیرا می‌خواهیم در مرحلهٔ اول همهٔ متغیرهای B را انتقال دهیم و سپس در مرحلهٔ بعد همهٔ متغیرها را به نمادهای پایانی تبدیل کنیم.

- برای به دست آوردن رشتهٔ aaabbbccc با استفاده از این گرامر فرایند اشتقاق زیر را داریم:

$$S \Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaaBCBCBC$$
$$\Rightarrow aaaBBCCBC \Rightarrow aaaBBCBCC \Rightarrow aaaBBBCCC$$
$$\Rightarrow aaabBBCCC \overset{*}{\Rightarrow} aaabbbCCC \Rightarrow aaabbbcCC \overset{*}{\Rightarrow} aaabbbccc$$

- در روش دوم از دو متغیر A و B استفاده می‌کنیم. متغیر A در سمت چپ رشته تولید می‌شود، سپس به سمت راست رشته حرکت می‌کند و به ازای هر a یک نماد b و یک نماد c می‌سازد. سپس این متغیر تبدیل به متغیر B می‌شود و به ابتدای رشته بازمی‌گردد. وقتي متغیر B به ابتدای رشته رسید، تبدیل به A می‌شود و این کار ادامه پیدا می‌کند تا رشتهٔ مورد نظر به دست آید.
- یک فرایند اشتقاق برای به دست آوردن رشتهٔ aaabbbcc در این روش به شکل زیر است:

$$\begin{aligned} &S \Rightarrow aAbc \Rightarrow abAc \Rightarrow abBbcc \\ &\Rightarrow aBbbcc \Rightarrow aaAbbcc \Rightarrow aabAbcc \\ &\Rightarrow aabbAcc \Rightarrow aabbBbccc \\ &\Rightarrow aabBbbccc \Rightarrow aaBbbbccc \Rightarrow aaabbbccc \end{aligned}$$

زبان‌های حساس به متن

- قوانین تولید را بدین صورت می‌نویسیم:

$$S \to abc|aAbc$$
$$Ab \to bA$$
$$Ac \to Bbcc$$
$$bB \to Bb$$
$$aB \to aa|aaA$$

زبان‌های حساس به متن

- برای هر زبان حساس به متن L که شامل رشتهٔ تهی نیست، یک ماشین ماشین کراندار خطی M وجود دارد به طوری که $L = L(M)$.
- اگر زبان L توسط یک ماشین کراندار خطی پذیرفته شود، آنگاه یک گرامر حساس به متن وجود دارد که L را تولید می‌کند.
- برای اثبات این دو قضیه می‌توان فرایند اشتقاق در گرامرهای مستقل از متن را با حرکت‌های یک ماشین کراندار خطی شبیه‌سازی کرد که در اینجا به آن نمی‌پردازیم.

زبان‌های حساس به متن

- می‌توان نشان داد که هر زبان حساس به متن بازگشتی (تصمیم‌پذیر) است و همچنین می‌توان نشان داد که یک زبان بازگشتی وجود دارد که حساس به متن نیست.
- نتیجه می‌گیریم زبان‌های حساس به متن زیرمجموعهٔ زبان‌های بازگشتی هستند و بنابراین قدرت ماشین‌های کراندار خطی کمتر از ماشین‌های تورینگ است.

سلسله‌مراتب زبان‌ها

- تا اینجا با چند دسته از زبان‌ها آشنا شدیم:
- L_{RE} : زبان‌های شمارش‌پذیر بازگشتی (تشخیص‌پذیر)
- L_{CS} : زبان‌های حساس به متن
- L_{CF} : زبان‌های مستقل از متن
- L_{REG} : زبان‌های منظم

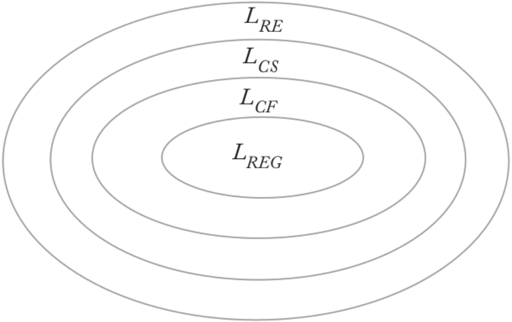
سلسله‌مراتب زبان‌ها

- چامسکی، یکی از پایه‌گذاران زبان‌های صوری، همهٔ زبان‌های صوری را به چهار نوع (نوع ۰، نوع ۱، نوع ۲، و نوع ۳) [1] تقسیم کرده است.
- زبان‌های نوع صفر زبان‌هایی هستند که با گرامرهای بدون محدودیت تولید می‌شوند، یعنی زبان‌های شمارش‌پذیر بازگشتی. زبان‌های نوع یک زبان‌های حساس به متن، نوع دو زبان‌های مستقل از متن، و نوع سه زبان‌های منظم هستند.
- زبان‌های نوع i در این طبقه‌بندی، زیرمجموعهٔ زبان‌های نوع $i - ۱$ هستند.

[1] type ۰ ، type ۱ ، type ۲ ، type ۳

سلسله‌مراتب زبان‌ها

- بر اساس طبقه‌بندی چامسکی، سلسله‌مراتب زبان‌ها را می‌توان به صورت زیر نشان داد.

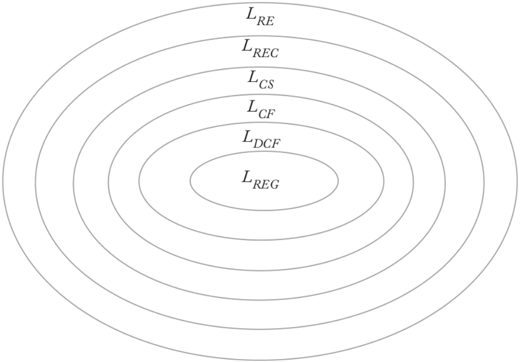


سلسله‌مراتب زبان‌ها

- همچنین با دو طبقه دیگر از زبان‌ها آشنا شدیم.
- L_{DCF} : زبان‌های مستقل از متن قطعی
- L_{REC} : زبان‌های بازگشتی (تصمیم‌پذیر)

سلسله‌مراتب زبان‌ها

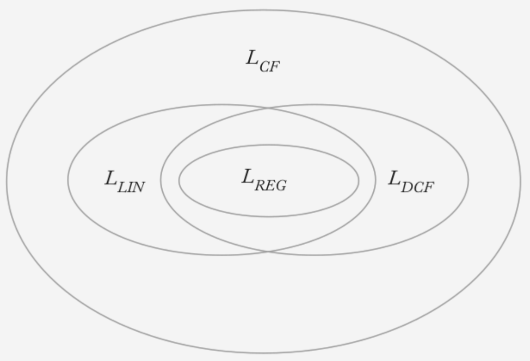
- سلسله‌مراتب زبان‌ها را بر اساس آنچه مطالعه کردیم، می‌توان به صورت زیر نشان داد.



سلسله‌مراتب زبان‌ها

- همچنین با زبان‌های خطی L_{LIN} آشنا شدیم که توسط گرامرهای خطی تولید می‌شوند. زبان‌های خطی زیرمجموعهٔ زبان‌های مستقل از متن هستند و هر زبان منظم یک زبان خطی است، پس این زبان‌ها ابرمجموعهٔ زبان‌های منظم هستند.
- از طرف دیگر زبان‌های خطی وجود دارند که زبان مستقل از متن قطعی نیستند و زبان‌های مستقل از متن قطعی وجود دارند که خطی نیستند، اما این دو زبان اشتراک‌هایی دارند.

- جایگاه زبان‌های خطی را می‌توان به صورت زیر نشان داد.



سلسله‌مراتب زبان‌ها

- همچنین ماشین‌های کراندار خطی را غیرقطعی تعریف کردیم. جایگاه زبان‌هایی که با ماشین‌های کراندار خطی قطعی پذیرفته می‌شوند تاکنون شناخته نشده است.
- به عبارت دیگر تاکنون اثباتی ارائه نشده است مبنی بر اینکه زبان‌های حساس به متن قطعی زیر مجموعهٔ زبان‌های حساس به متن هستند یا خیر.

سلسله‌مراتب زبان‌ها

مخفف	زبان	گرامر	ماشین
REG نوع ۳	منظم	$A \rightarrow aB, A \rightarrow Ba, A \rightarrow a$ $A, B \in V, a \in T^*$	متناهی قطعی و متناهی غیرقطعی
CF نوع ۲	مستقل از متن	$A \rightarrow \alpha$ $A \in V, \alpha \in (V \cup T)^*$	پشته‌ای
CS نوع ۱	حساس به متن	$x \rightarrow y$ $x, y \in (V \cup T)^+, \lvert y \rvert \geq \lvert x \rvert$	کراندار خطی
RE نوع ۰	شمارش‌پذیر بازگشتی	$x \rightarrow y$ $x \in (V \cup T)^+, y \in (V \cup T)^*$	تورینگ

محاسبه‌پذیری

تصمیم‌پذیری

- برای برخی از مسأله‌های محاسباتی هیچ الگوریتمی وجود ندارد. به طور مثال الگوریتمی وجود ندارد که با دریافت یک گرامر مستقل از متن تعیین کند که گرامر مبهم است. به عبارت دیگر این مسأله یک مسأله تصمیم‌ناپذیر است و هیچ ماشین تورینگی وجود ندارد که به ازای همهٔ ورودی‌ها (گرامرهای مستقل از متن) در یک حالت پایانی (در صورتی که گرامر مبهم است) یا یک حالت غیرپایانی (در صورتی که گرامر غیرمبهم است) متوقف شود.
- مفهوم تصمیم‌پذیری را در این قسمت معرفی می‌کنیم و خواهیم دید که چه کارهایی را با ماشین تورینگ نمی‌توان انجام داد. به عبارت دیگر محدودیت ماشین‌ها را در محاسبات بررسی می‌کنیم.
- بسیاری از مسائل تصمیم‌ناپذیر گرچه به آسانی قابل بیان هستند، ولی الگوریتمی برای آنها وجود ندارد.

تصمیم‌پذیری

- یک مسألهٔ تصمیم‌گیری یک عبارت یا گزاره است که پاسخ آن بلی یا خیر است. به عبارت دیگر یک مسأله تصمیم‌گیری، جمله‌ای است که جواب آن بلی است اگر عضو یک زبان تعیین شده باشد و پاسخ آن خیر است اگر عضو آن زبان نباشد.
- برای مثال این مسأله را در نظر بگیرید: به ازای گرامر مستقل از متن G ، آیا G مبهم است؟ پاسخ این سؤال برای برخی از گرامرها بلی است و برای برخی از گرامرها خیر است. دامنهٔ این مسأله، مجموعهٔ همهٔ گرامرهای مستقل از متن است.
- یک مسأله تصمیم‌پذیر است اگر ماشین تورینگی وجود داشته باشد که به ازای همهٔ اعضای دامنه تعیین کند که جواب بلی است (جمله عضو زبان است) یا خیر (جمله عضو زبان نیست).
- دقت کنید که یک مسأله می‌تواند بر روی یک دامنه تصمیم پذیر باشد و بر روی یک دامنهٔ دیگر تصمیم‌پذیر نباشد.

تصمیم‌پذیری

- مسألهٔ دهم هیلبرت را در نظر بگیرید.

- آیا الگوریتمی وجود دارد که به ازای یک چندجمله‌ای داده شده، تعیین کند آیا ریشه‌های چندجمله‌ای اعداد صحیح هستند یا خیر؟

- این مسأله را می‌توانیم به صورت یک مسأله تعیین عضویت یک جمله در یک زبان بنویسیم:

- به ازای چند جمله‌ای p تعیین کنید آیا p عضو زبان D است یا خیر.
$D = \{p :$ p یک چندجمله‌ای با ریشه‌های صحیح است $\}$

- به عبارت دیگر آیا ماشین تورینگی برای زبان D وجود دارد که به ازای همهٔ چندجمله‌ای ها در یک حالت پایانی یا یک حالت غیر پایانی متوقف شود؟

تصمیم‌پذیری

- $D = \{p :$ یک چندجمله‌ای با ریشه‌های صحیح است $p\}$

- برای تشخیص زبان D ماشین تورینگ زیر را در نظر بگیرید:

- ماشین تورینگ M به ازای ورودی p و همهٔ زیر مجموعه‌های مجموعهٔ اعداد صحیح، عبارت p را محاسبه می‌کند. اگر حاصل چندجمله‌ای به ازای مقادیری از اعداد صحیح صفر بود ورودی p پذیرفته می‌شود.

- دقت کنید که اگر p ریشهٔ چندجمله‌ای نداشت ماشین تورینگ M در یک حلقهٔ بی‌پایان می‌افتد، زیرا به جواب نمی‌رسد ولی نمی‌دانیم آیا در نهایت به جواب خواهد رسید یا خیر.

- در اینصورت می‌گوییم ماشین M یک تشخیص‌دهنده [1] است ولی تصمیم‌گیرنده [2] نیست.

- اگر برای دامنهٔ مقادیر ریشه‌های یک چندجمله‌ای در این مسأله یک محدوده تعریف کنیم آنگاه مسأله تصمیم‌پذیر خواهد شد، زیرا در حلقهٔ بی‌پایان نمی‌افتیم.

---

[1] recognizer

[2] decider

- برای وارد کردن یک مفهوم به ورودی ماشین تورینگ ابتدا باید آن را کدگذاری کنیم. برای مثال اگر بخواهیم گراف G را به ورودی ماشین تورینگ بدهیم، ابتدا گراف را به صورت یک رشته در می‌آوریم. این کدگذاری را با علامت $\langle G \rangle$ نشان می‌دهیم.

تصمیم‌پذیری

- مسأله همبندی گراف را در نظر بگیرید. این مسأله به ازای یک گراف داده شدهٔ G می‌پرسد آیا G همبند است یا خیر.
- حال زبان A را به صورت زیر در نظر بگیرید. $A = \{\langle G \rangle : \text{یک گراف همبند است } G\}$
- مسأله همبندی گراف G را به یک مسأله عضویت در زبان G تغییر دادیم. اینک می‌پرسیم به ازای یک رشتهٔ کدگذاری شدهٔ $\langle G \rangle$ برای گراف G آیا $\langle G \rangle$ عضو زبان A است یا خیر.
- برای تشخیص دادن این زبان یک ماشین تورینگ طراحی می‌کنیم که زبان A را بپذیرد. اگر به ازای هر $\langle G \rangle$ ماشین تورینگ رشته را بپذیرد یا رد کند، آنگاه ماشین طراحی شده یک تصمیم‌گیرنده است و زبان A یک زبان تصمیم‌پذیر است.

تصمیم‌پذیری

- زبان A تصمیم‌پذیر است، زیرا الگوریتمی برای آن وجود دارد. این الگوریتم یک توصیف سطح بالا است برای عملکرد یک ماشین تورینگ که A را تصمیم‌می‌گیرد، یا به عبارت دیگر به ازای هر جمله w (که یک گراف کدگذاری شده است) در مجموعهٔ همهٔ گراف‌ها تصمیم می‌گیرد که جمله w عضو زبان A هست یا خیر.
- به ازای گراف G الگوریتم بدین صورت است: (۱) یک رأس را علامت‌گذاری می‌کنیم. (۲) همهٔ همسایه‌های رأس‌های علامت‌گذاری شده را علامت‌گذاری می‌کنیم. (۳) اگر همهٔ رئوس علامت‌گذاری شدند، گراف G همبند است و $\langle G \rangle$ عضو زبان A است، پس $\langle G \rangle$ پذیرفته می‌شود، در غیر اینصورت گراف G همبند نیست و $\langle G \rangle$ عضو زبان A نیست، پس $\langle G \rangle$ رد می‌شود.

تصمیم‌پذیری

- در مبحث الگوریتم‌ها معمولا یک الگوریتم برای یک مسأله پیدا می‌کنیم و با پیدا کردن یک الگوریتم به طور ضمنی نشان می‌دهیم که مسأله قابل حل است.
- ولی اگر الگوریتمی برای مسأله‌ای پیدا نکنیم، چگونه می‌توانیم با قاطعیت بگوییم که مسأله غیرقابل حل است و الگوریتمی وجود ندارد؟
- در اینجا روشی برای اثبات تصمیم‌ناپذیری ارائه می‌کنیم. اگر ثابت کنیم یک مسأله تصمیم‌ناپذیر است، اثبات کرده‌ایم که الگوریتمی برای آن وجود ندارد.
- اگر بتوانیم اثبات کنیم مسأله‌ای غیرقابل حل است، دیگر به دنبال الگوریتم برای آن نمی‌گردیم.

تصمیم‌پذیری

- مسألهٔ پذیرفتن یک رشته توسط یک ماشین متناهی را در نظر بگیرید. به ازای رشتهٔ w آیا رشته توسط ماشین متناهی قطعی B پذیرفته می‌شود؟

- این مسأله را می‌توانیم به صورت یک زبان نمایش دهیم:
$A_{DFA} = \{\langle B, w \rangle$: یک ماشین متناهی قطعی است که رشتهٔ w را می‌پذیرد B$\}$

- این زبان یک زبان تصمیم‌پذیر است، زیرا الگوریتمی وجود دارد که به ازای ماشین B و رشتهٔ w تعیین کند رشته توسط ماشین پذیرفته می‌شود یا خیر. این الگوریتم می‌تواند توسط یک ماشین تورینگ اجرا شود. به عبارت دیگر ماشین تورینگ M وجود دارد که اجرای رشتهٔ w بر روی ماشین B را شبیه‌سازی می‌کند. اگر رشتهٔ w توسط ماشین B پذیرفته شود، ماشین M در یک حالت پایانی متوقف می‌شود و در غیراینصورت ماشین M در یک حالت غیرپایانی متوقف می‌شود.

- پس ماشین M که برای زبان A_{DFA} طراحی شده است، به ازای هر ورودی $\langle B, w \rangle$ متوقف می‌شود و بنابراین زبان A_{DFA} تصمیم‌پذیر است.

تصمیم‌پذیری

- مسألهٔ تولید یک رشته توسط یک گرامر مستقل از متن را در نظر می‌گیریم. به ازای رشتهٔ w آیا رشته توسط گرامر مستقل از متن G تولید می‌شود؟

- این مسأله را می‌توانیم به صورت یک زبان نمایش دهیم:
$A_{CFG} = \{\langle G, w \rangle$: G یک گرامر مستقل از متن است که رشتهٔ w را تولید می‌کند$\}$

- آیا زبان A_{CFG} تصمیم‌پذیر است؟

- $A_{CFG} = \{\langle G, w\rangle :$ یک ماشین گرامر مستقل از متن است که رشتهٔ w را تولید می‌کند $G\}$
- زبان A_{CFG} تصمیم‌پذیر است، زیرا یک الگوریتم وجود دارد که تعیین کند آیا رشته عضو زبان است یا خیر.
- گرامر G را به فرم نرمال چامسکی تبدیل می‌کنیم. همهٔ اشتقاق‌ها را با طول $|w|$ به دست می‌آوریم. اگر رشتهٔ w به دست آمد رشته عضو زبان است، در غیراینصورت رشته عضو زبان نیست.

تصمیم‌پذیری

- زبان زیر را در نظر بگیرید: $E_{DFA} = \{\langle A\rangle : L(A) = \emptyset$ که به طوری است قطعی متناهی ماشین یک $A\}$

- آیا این زبان یک زبان تصمیم‌پذیر است؟ به عبارت دیگر آیا الگوریتمی وجود دارد که به ازای یک ماشین متناهی قطعی داده شده، تعیین کند آیا زبان ماشین تهی است یا خیر؟

- و باز به عبارتی آیا ماشین تورینگی وجود دارد که به ازای هر ماشین متناهی قطعی داده شده (به صورت کدگذاری شده) در حالت پایانی متوقف شود اگر زبان ماشین تهی است و در حالت غیرپایانی متوقف شود اگر زبان ماشین غیرتهی است؟

- این زبان یک زبان تصمیم‌پذیر است، زیرا الگوریتمی برای آن وجود دارد. الگوریتم بدین صورت است که اگر حداقل یک مسیر غیرتهی از حالت آغازی ماشین متناهی A به یک حالت پایانی وجود داشته باشد، آنگاه زبان ماشین A غیرتهی است.

تصمیم‌پذیری

- زبان زیر را در نظر بگیرید: $E_{\text{CFG}} = \{\langle G\rangle : L(G) = \emptyset$ یک گرامر مستقل از متن است به طوری که $G\}$
- این زبان نیز یک زبان تصمیم‌پذیر است زیر الگوریتمی برای آن وجود دارد.
- ابتدا همهٔ نمادهای پایانی را علامت‌گذاری می‌کنیم. سپس متغیر A که قانونی به صورت $A \to U_1 U_2 \cdots U_n$ دارد را علامت‌گذاری می‌کنیم، اگر همهٔ نمادهای (پایانی یا غیرپایانی) U_i علامت‌گذاری شده باشند.
- اگر در نهایت S علامت‌گذاری نشد، ورودی را قبول می‌کنیم و در غیراینصورت آن را رد می‌کنیم.

- آیا زبان A_{LBA} تصمیم‌پذیر است؟
$A_{LBA} = \{\langle M, w\rangle :$ M یک ماشین کراندار خطی است که رشتهٔ w را می‌پذیرد$\}$

- اگر M یک ماشین کراندار خطی با q حالت باشد و الفبای نوار آن g نماد داشته باشد آنگاه به ازای یک ورودی با طول n این ماشین می‌تواند در qng^n پیکربندی مختلف قرار بگیرد، زیرا محتوای نوار می‌تواند g^n حالت مختلف داشته باشد و هد خواندن نوشتن می‌تواند در n مکان مختلف قرار بگیرد.

تصمیم‌پذیری

- آیا زبان A_{LBA} تصمیم‌پذیر است؟
$A_{LBA} = \{\langle M, w\rangle$: یک ماشین کراندار خطی است که رشتهٔ w را می‌پذیرد $M\}$
- می‌توانیم الگوریتمی برای پذیرفتن یک رشته توسط یک ماشین کراندار خطی بدین صورت طراحی کنیم:

۱. اجرای ماشین M را بر روی رشتهٔ w شبیه‌سازی می‌کنیم. از آنجایی که تعداد کل پیکربندی‌های ممکن qng^n است، شبیه‌سازی را تا qng^n گام ادامه می‌دهیم.

۲. اگر ماشین قبل از اتمام شبیه‌سازی متوقف شد و رشته را پذیرفت ورودی $\langle M, w\rangle$ را می‌پذیریم. اگر ماشین M رشته را نپذیرفت و یا تعداد گام‌های شبیه‌سازی پایان یافت ورودی $\langle M, w\rangle$ را رد می‌کنیم.

- الگوریتمی برای تصمیم‌گیری این زبان وجود دارد، پس این زبان تصمیم‌پذیر است.

تصمیم‌ناپذیری

- حال مسألهٔ زیر را در نظر بگیرید: به ازای ماشین تورینگ M و رشتهٔ w تعیین کنید آیا رشته توسط ماشین پذیرفته می‌شود یا خیر.
- به عبارت دیگر $A_{TM} = \{\langle M, w \rangle : M$ یک ماشین تورینگ است که رشتهٔ w را می‌پذیرد$\}$
- آیا زبان A_{TM} تصمیم‌پذیر است؟

تصمیم‌ناپذیری

- فرض می‌کنیم زبان A_{TM} تصمیم‌پذیر باشد و به تناقض می‌رسیم. فرض کنیم H یک ماشین تصمیم‌پذیر است که زبان A_{TM} را می‌پذیرد.

- با دریافت ورودی $\langle M, w \rangle$ به طوری که M یک ماشین تورینگ دلخواه و w یک جمله است، در صورتی که ماشین M جملهٔ w را بپذیرد و متوقف شود، آنگاه H ورودی $\langle M, w \rangle$ را می‌پذیرد، در غیر اینصورت H ورودی را نمی‌پذیرد.
  اگر M جملهٔ w را بپذیرد $H(\langle M, w \rangle) = \text{accept};$
  اگر M جملهٔ w را نپذیرد $H(\langle M, w \rangle) = \text{reject};$

- حال با استفاده از ماشین تورینگ H ماشین تورینگ D را طراحی می‌کنیم.
  اگر $H(\langle M, \langle M \rangle \rangle) = \text{accept};$ آنگاه $D(\langle M \rangle) = \text{reject};$
  اگر $H(\langle M, \langle M \rangle \rangle) = \text{reject};$ آنگاه $D(\langle M \rangle) = \text{accept};$

- به عبارت دیگر:
  اگر M جملهٔ $\langle M \rangle$ را بپذیرد $D(\langle M \rangle) = \text{reject};$
  اگر M جملهٔ $\langle M \rangle$ را نپذیرد $D(\langle M \rangle) = \text{accept};$

- حال اگر به ماشین تورینگ D ورودی $\langle D \rangle$ را بدهیم، چه اتفاقی می‌افتد؟
- اگر $D(\langle D \rangle) = \text{accept}$ آنگاه $H(\langle D, \langle D \rangle \rangle) = \text{reject}$ آنگاه $D(\langle D \rangle) = \text{reject}$
- اگر $D(\langle D \rangle) = \text{reject}$ آنگاه $H(\langle D, \langle D \rangle \rangle) = \text{accept}$ آنگاه $D(\langle D \rangle) = \text{accept}$
- پس در هر صورت به تناقض می‌رسیم و بنابراین فرض اولیه مبنی بر وجود ماشین تورینگ H نادرست است و چنین ماشینی وجود ندارد.

کاهش

- کاهش [1] روشی است برای تبدیل یک مسأله به یک مسألهٔ دیگر به طوری که راه حل مسألهٔ دوم بتواند برای مسألهٔ اول مورد استفاده قرار بگیرد.
- پس وقتی مسألهٔ A را به مسألهٔ B کاهش دهیم جواب مسألهٔ B را می‌توانیم برای مسألهٔ A نیز استفاده کنیم.
- برای مثال در ریاضی مسألهٔ حل کردن یک دستگاه معادلهٔ چند مجهولی را به پیدا کردن وارون یک ماتریس کاهش می‌دهیم.

[1] reduction

کاهش

- در نظریهٔ محاسبات، اگر مسألهٔ A قابل کاهش [1] به مسألهٔ B باشد و B تصمیم‌پذیر باشد، آنگاه A نیز تصمیم‌پذیر است.

- مسألهٔ A را به B کاهش می‌دهیم. اگر به ازای یک ورودی B ورودی را پذیرفت، A نیز ورودی را می‌پذیرد، در غیراینصورت ورودی را نمی‌پذیرد.

- اگر A تصمیم‌ناپذیر باشد و قابل کاهش به B باشد، آنگاه B نیز تصمیم‌ناپذیر است. (فرض کنید B تصمیم‌پذیر باشد، آنگاه الگوریتمی تصمیم‌پذیر برای آن وجود دارد. مسأله A را به B کاهش می‌دهیم و الگوریتمی تصمیم‌پذیر برای A با استفاده از الگوریتم B پیدا می‌کنیم. ولی می‌دانیم A تصمیم‌ناپذیر است، پس B نمی‌تواند تصمیم‌پذیر باشد.)

[1] reducable

کاهش

- حال مسألهٔ توقف [1] را در نظر بگیرید.
- مسألهٔ توقف می‌پرسد آیا ماشین تورینگ M بر روی رشتهٔ w توقف می‌کند یا خیر؟
- $\mathrm{HALT_{TM}} = \{\langle M, w\rangle$: M یک ماشین تورینگ است و بر روی رشتهٔ w توقف می‌کند$\}$
- دقت کنید که می‌دانیم $\mathrm{HALT_{TM}}$ تشخیص‌پذیر است زیرا برای آن ماشین تورینگی وجود دارد که آن را می‌پذیرد. در اینجا ثابت می‌کنیم که این زبان تصمیم‌پذیر نیست.

[1] halting problem

کاهش

- فرض کنیم مسأله توقف تصمیم‌پذیر باشد. پس ماشین تورینگ R باید وجود داشته باشد که $\mathrm{HALT_{TM}}$ را تصمیم می‌گیرد.

- با استفاده از R ماشین S را می‌سازیم و نشان می‌دهیم که S یک ماشین تصمیم‌پذیر برای مسألهٔ پذیرش $\mathrm{A_{TM}}$ است. از آنجایی که مسألهٔ $\mathrm{A_{TM}}$ تصمیم‌ناپذیر است فرض اولیه نادرست بوده و مسألهٔ توقف نمی تواند تصمیم‌پذیر باشد.

- ماشین S را بدین صورت می‌سازیم: ماشین تورینگ R را روی ورودی $\langle M, w\rangle$ اجرا می‌کنیم. اگر R ورودی را رد، کرد S نیز رد می‌کند. اگر R ورودی را پذیرفت، آنگاه اجرای ماشین M را با ورودی w شبیه‌سازی می‌کنیم. اگر M رشتهٔ w را پذیرفت S ورودی را می‌پذیرد درغیراینصورت S ورودی را رد می‌کند

- پس فرض کردیم $\mathrm{HALT_{TM}}$ تصمیم‌پذیر است و بدین نتیجه رسیدیم که $\mathrm{A_{TM}}$ نیز تصمیم‌پذیر است. ولی قبلا ثابت کردیم که $\mathrm{A_{TM}}$ تصمیم‌ناپذیر است، پس $\mathrm{HALT_{TM}}$ تصمیم‌ناپذیر است.

تصمیم‌ناپذیری

- آنچه به طور رسمی با استدلال منطقی نشان دادیم را می‌توانیم به طور غیررسمی به زبان برنامه نویسی بیان کنیم:
- فرض کنید تابعی به نام `halt` وجود دارد که به ازای هر تابع داده شدهٔ `f` تعیین می‌کند آیا تابع `f` متوقف می‌شود یا خیر.
- حال تابع `f` را بدین صورت در نظر بگیرید:

```
void f() { if (halt(&f)) loop_forever(); }
```

- اگر `halt` مقدار درست را بازگرداند آنگاه تابع `f` باید متوقف شود ولی متوقف نمی‌شود. اگر `halt` مقدار نادرست را بازگرداند آنگاه تابع `f` نباید متوقف شود ولی متوقف می‌شود. پس فرض اولیه مبنی بر وجود تابع `halt` نادرست بوده و چنین تابعی وجود ندارد.

کاهش

- ثابت کنید مسألهٔ $E_{TM} = \{\langle M \rangle : L(M) = \emptyset$ و M یک ماشین تورینگ است$\}$ تصمیم‌ناپذیر است.
- فرض کنید ماشین تورینگ R زبان E_{TM} را تصمیم می‌گیرد. با استفاده از ماشین R ماشین S را می‌سازیم که زبان A_{TM} را تصمیم می‌گیرد. در نتیجه به تناقض می‌رسیم.
- به ازای ورودی M و w در ماشین S چنین تصمیم می‌گیریم:

۱. ماشین تورینگ $M_۱$ را طوری می‌سازیم که به ازای ورودی x اگر $x \neq w$ آنگاه، رشته را رد می‌کند و اگر $x = w$ آنگاه x را می‌پذیرد اگر M رشتهٔ w را بپذیرد و درغیراینصورت x را رد می‌کند.

۲. ماشین R را با ورودی $M_۱$ اجرا می‌کنیم.

۳. اگر R ورودی را بپذیرد، ماشین S ورودی را رد می‌کند و اگر R ورودی را رد کند، ماشین S ورودی را می‌پذیرد.

کاهش

- دقت کنید که در این اثبات ماشین S در هر بار اجرا باید بتواند ماشین $M_۱$ را با استفاده از توصیف ماشین M و رشتهٔ w بسازد. این کار همیشه ممکن است، زیرا $M_۱$ یک کپی از M است با این تفاوت که یک قسمت برای تست $x = w$ دارد.

- پس فرض کردیم ماشین R یک تصمیم‌گیرنده برای زبان E_{TM} است و یک تصمیم‌گیرندهٔ S برای A_{TM} ساختیم. از آنجایی که قبلا اثبات کردیم هیچ تصمیم‌گیرنده‌ای برای A_{TM} وجود ندارد پس به تناقض می‌رسیم و بنابراین فرض اولیه مبنی بر تصمیم‌پذیر بودن E_{TM} نادرست بوده است و E_{TM} تصمیم‌ناپذیر است.

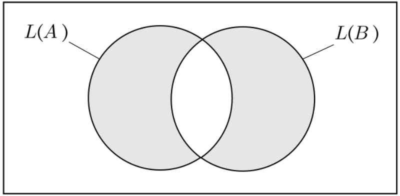
کاهش

- ثابت کنید زبان $\mathrm{EQ_{DFA}} = \{\langle \mathrm{A}, \mathrm{B} \rangle : \mathrm{L(A)} = \mathrm{L(B)}$ و A و B دو ماشین متناهی قطعی هستند و$\}$ تصمیم‌پذیر است.

- باید ماشین تورینگی برای این زبان پیدا کنیم که این زبان را بپذیرد.

- با استفاده از زبان A و B زبان C را می‌سازیم به طوری که
$$\mathrm{L(C)} = (\mathrm{L(A)} \cap \overline{\mathrm{L(B)}}) \cup (\overline{\mathrm{L(A)}} \cap \mathrm{L(B)})$$

- اگر $\mathrm{L(C)} = \emptyset$ آنگاه $\mathrm{L(A)} = \mathrm{L(B)}$



کاهش

- می‌دانیم زبان $\mathrm{E_{DFA}} = \{\langle \mathrm{A} \rangle : \mathrm{L(A)} = \emptyset$ و A یک ماشین متناهی قطعی است$\}$ یک زبان تصمیم‌پذیر است، پس برای آن یک ماشین تورینگ تصمیم‌گیرندهٔ T وجود دارد.
- حال ماشین F را می‌سازیم به طوری که با دریافت دو ماشین متناهی قطعی A و B :

۱. ماشین متناهی قطعی C را می‌سازد به طوری که $\mathrm{L(C)} = (\mathrm{L(A)} \cap \overline{\mathrm{L(B)}}) \cup (\overline{\mathrm{L(A)}} \cap \mathrm{L(B)})$

۲. با استفاده از ماشین T که یک تصمیم‌گیرنده برای زبان $\mathrm{E_{DFA}}$ تصمیم می‌گیریم که آیا زبان $\mathrm{L(C)}$ تهی است یا خیر.

۳. اگر T ورودی $\langle \mathrm{C} \rangle$ را بپذیرد آنگاه ماشین F ورودی $\langle \mathrm{A}, \mathrm{B} \rangle$ را قبول می‌کند و اگر T ورودی $\langle \mathrm{C} \rangle$ را نپذیرد، آنگاه F نیز ورودی را رد می‌کند.

کاهش

- توجه کنید که استدلال قبل را به دو صورت می‌توانیم تعبیر کنیم:
- (۱) برای اینکه نشان دهیم $\mathrm{EQ_{DFA}}$ تصمیم پذیر است، کافی است یک الگوریتم تصمیم‌پذیر (یک ماشین تورینگ تصمیم‌گیرنده) برای آن پیدا کنیم که در استدلال قبل چنین الگوریتمی یافتیم.
- (۲) برای اینکه نشان دهیم $\mathrm{EQ_{DFA}}$ تصمیم‌پذیر است، از ماشین تصمیم‌گیرنده برای زبان $\mathrm{E_{DFA}}$ استفاده کردیم، پس مسأله را به یک مسأله تصمیم‌پذیر کاهش دادیم.

- ثابت کنید زبان $EQ_{TM} = \{\langle M_۱, M_۲\rangle : L(M_۱) = L(M_۲)$ و $M_۱$ و $M_۲$ دو ماشین تورینگ هستند $\}$ تصمیم‌ناپذیر است.

- فرض می‌کنیم زبان EQ_{TM} تصمیم‌پذیر باشد، آنگاه نشان می‌دهیم که در آن صورت زبان E_{TM} باید تصمیم‌پذیر باشد. می‌دانیم که E_{TM} تصمیم‌ناپذیر است، پس فرض اولیه نادرست بوده و EQ_{TM} تصمیم‌ناپذیر است.

کاهش

- ثابت کنید زبان $\mathrm{EQ_{TM}} = \{\langle \mathrm{M}_۱, \mathrm{M}_۲\rangle : \mathrm{L}(\mathrm{M}_۱) = \mathrm{L}(\mathrm{M}_۲)$ و $\mathrm{M}_۱$ و $\mathrm{M}_۲$ دو ماشین تورینگ هستند$\}$ تصمیم‌ناپذیر است.

- فرض می‌کنیم زبان $\mathrm{EQ_{TM}}$ تصمیم‌پذیر باشد، آنگاه ماشین تورینگ تصمیم‌گیرندهٔ R برای آن وجود دارد. با استفاده از ماشین R ماشین S را برای $\mathrm{E_{TM}}$ می‌سازیم.

- به ازای ورودی $\langle \mathrm{M}\rangle$ (رشتهٔ متناظر با ماشین تورینگ M) در ماشین S برای زبان $\mathrm{E_{TM}}$ چنین می‌کنیم:

۱. ماشین R را با ورودی $\langle \mathrm{M}, \mathrm{M}_۱\rangle$ اجرا می‌کنیم به طوری که $\mathrm{M}_۱$ ماشینی است که همهٔ ورودی‌ها را رد می‌کند.

۲. اگر ماشین R ورودی $\langle \mathrm{M}, \mathrm{M}_۱\rangle$ را پذیرفت، این بدین معناست که زبان ماشین M تهی است، پس ورودی $\langle \mathrm{M}\rangle$ در ماشین S پذیرفته می‌شود و در غیراینصورت ورودی رد می‌شود.

- اگر R تصمیم‌گیرنده‌ای برای زبان $\mathrm{EQ_{TM}}$ باشد، آنگاه S تصمیم‌گیرنده‌ای برای زبان $\mathrm{E_{TM}}$ است. اما قبلا نشان دادیم $\mathrm{E_{TM}}$ تصمیم‌ناپذیر است، پس فرض اولیه نادرست بوده و $\mathrm{EQ_{TM}}$ باید تصمیم‌ناپذیر باشد.

کاهش

- ثابت کنید زبان $ALL_{DFA} = \{\langle A \rangle : L(A) = \Sigma^*$ یک ماشین متناهی قطعی است و $A\}$ تصمیم‌پذیر است.
- با استفاده از ماشین تصمیم‌گیرندهٔ T برای زبان EQ_{DFA} یک ماشین تصمیم‌گیرنده برای زبان ALL_{DFA} می‌سازیم.
- یک ماشین متناهی قطعی B با یک حالت آغازی و پایانی q_0 می‌سازیم. به ازای هر $a \in \Sigma$ گذار $\delta(q_0, a) = q_0$ را تعریف می‌کنیم. این ماشین همهٔ رشته‌های Σ^* را می‌پذیرد.
- حال از ماشین تصمیم‌گیرندهٔ T استفاده می‌کنیم و ماشین‌های قطعی A و B را به عنوان ورودی به ماشین T می‌دهیم. اگر T ورودی را پذیرفت ماشین A پذیرفته می‌شود و درغیراینصورت پذیرفته نمی‌شود.

مسألهٔ تناظر پست

- تا اینجا محاسبه‌پذیری را برای مسائلی در حوزهٔ نظریهٔ زبان‌ها و ماشین‌ها بررسی کردیم. انواع دیگری از مسأله‌های محاسباتی وجود دارند که می‌توان تصمیم‌پذیری و تصمیم‌ناپذیری آنها را با استفاده از روش‌های ذکر شده اثبات کرد.
- مسألهٔ تناظر پست [1] نوعی پازل است. این مسأله، یک مسألهٔ تصمیم‌ناپذیر است.

[1] Post Correspondence Problem (PCP)

مسألهٔ تناظر پست

- یک دومینو [1] تشکیل شده است از یک رشتهٔ بالایی x و یک رشتهٔ پایینی y که آن را به صورت $\begin{bmatrix} x \\ - \\ y \end{bmatrix}$ نشان می‌دهیم.
- یک مجموعه از دومینوها دارای یک تطابق [1] است اگر دنباله‌ای از دومینوها وجود داشته باشد به طوری که الحاق رشته‌های بالایی دومینوها در دنباله برابر با الحاق رشته‌های پایینی دومینوها باشد.

[1] domino
[1] match

مسألهٔ تناظر پست

- برای مثال مجموعهٔ P را در نظر بگیرید: $P = \left\{ \left[\frac{a}{ab}\right], \left[\frac{b}{ca}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}$
- به ازای اعضای مجموعهٔ P دنبالهٔ دومینوهای زیر دارای یک تطابق است: $\left[\frac{a}{ab}\right] \left[\frac{b}{ca}\right] \left[\frac{ca}{a}\right] \left[\frac{a}{ab}\right] \left[\frac{abc}{c}\right]$
- این تطابق را می‌توانیم به صورت زیر نشان دهیم:

```
| a | b | c a | a | a b c |
| a b | c a | a | a b | c |
```

- توجه کنید که اعضای مجموعهٔ P می‌توانند به هر تعداد دلخواه در دنبالهٔ دومینوهای دارای تطابق تکرار شوند. مثلا دومینوی $\left[\frac{a}{ab}\right]$ در دنبالهٔ بالا دو بار تکرار شده است.

مسألهٔ تناظر پست

- برای برخی از مجموعه‌ها تطابقی وجود ندارد. برای مثال در مجموعهٔ $\left\{ \left[\frac{\text{abc}}{\text{ab}}\right], \left[\frac{\text{ca}}{\text{a}}\right], \left[\frac{\text{acc}}{\text{ba}}\right] \right\}$ هیچ تطابقی نمی‌توان یافت.
- مسأله تناظر پست می‌پرسد آیا به ازای یک مجموعهٔ داده شدهٔ P یک دنبالهٔ حاوی تطابق وجود دارد یا خیر؟
- دقت کنید یک دومینو می‌تواند به تعداد نامحدودی در یک دنبالهٔ دارای تطابق استفاده شود، پس مجموعهٔ همهٔ پیکربندی‌ها نامحدود است.
- با استفاده از کاهش مسألهٔ A_{TM} به مسألهٔ تناظر پست، می‌توان نشان داد این مسأله تصمیم‌ناپذیر است. به عبارت دیگر برای این مسأله هیچ الگوریتمی وجود ندارد.

مسألهٔ تناظر پست

- یک نمونه از مسألهٔ تناظر پست، یک مجموعهٔ P از دومینوها به صورت $\left\{\left[\frac{t_۱}{b_۱}\right], \left[\frac{t_۲}{b_۲}\right], \cdots, \left[\frac{t_n}{b_n}\right]\right\}$ است.
- یک تطابق دنباله‌ای به صورت $i_۱, i_۲, \cdots, i_k$ است، به طوری که $t_{i_۱}, t_{i_۲}, \cdots, t_{i_k} = b_{i_۱}, b_{i_۲}, \cdots, b_{i_k}$.
- مسأله این است که آیا مجموعهٔ P دارای یک تطابق است یا خیر.
- می‌توانیم مسألهٔ تطابق پست را بدین صورت بیان کنیم:
- $PCP = \{\langle P \rangle :$ یک نمونه از مسألهٔ تناظر پست است که دارای یک تطابق است $P\}$

مسائل تصمیم‌ناپذیر در مورد زبان‌های مستقل از متن

- حال با استفاده از مسألهٔ تناظر پست، تصمیم‌ناپذیر بودن دو مسأله در مورد زبان‌های مستقل از متن را نشان می‌دهیم.
- فرض کنید $P = \{(t_۱, b_۱), (t_۲, b_۲), \cdots, (t_n, b_n)\}$ یک نمونه از مسألهٔ تناظر پست به صورت P است به طوری که t_i رشتهٔ بالایی یک دومینو و b_i رشتهٔ پایینی یک دومینو بر روی الفبای Σ است.
- دو گرامر مستقل از متن G_t و G_b را به طوری می‌سازیم که ویژگی آنها شبیه مسألهٔ تناظر پست باشد.

مسائل تصمیم‌ناپذیر در مورد زبان‌های مستقل از متن

- فرض کنید نمادهای $c_۱, c_۲, \cdots, c_n$ در الفبای Σ نباشند.
- گرامر G_t را با متغیر آغازی S_t و قوانین تولید $(۱ \leq i \leq n)\; S_t \rightarrow t_i S_t c_i | t_i c_i$ می‌سازیم.
- به همین ترتیب گرامر G_b را با متغیر آغازی S_b و قوانین تولید $(۱ \leq i \leq n)\; S_b \rightarrow b_i S_b c_i | b_i c_i$ می‌سازیم.
- به ازای هر رشتهٔ $c_{i_۱} c_{i_۲} \cdots c_{i_k}$ یک رشتهٔ x در زبان $L(G_t)$ و یک رشتهٔ y در زبان $L(G_b)$ وجود دارد، به طوری که $x = t_{i_k} \cdots t_{i_۲} t_{i_۱} c_{i_۱} c_{i_۲} \cdots c_{i_k}$ و $y = b_{i_k} \cdots b_{i_۲} b_{i_۱} c_{i_۱} c_{i_۲} \cdots c_{i_k}$

مسائل تصمیم‌ناپذیر در مورد زبان‌های مستقل از متن

- حال نشان می‌دهیم دو مسألهٔ زیر تصمیم‌ناپذیرند.
- $\mathrm{EMI_{CFG}}$: به ازای دو گرامر مستقل از متن داده شدهٔ G_1 و G_2 آیا $L(G_1) \cap L(G_2) \neq \emptyset$ ؟

$\mathrm{EMI_{CFG}} = \{\langle G_1, G_2\rangle :$ G_1 و G_2 دو گرامر مستقل از متن هستند و $L(G_1) \cap L(G_2) \neq \emptyset\}$

- $\mathrm{AMB_{CFG}}$: به ازای گرامر مستقل از متن داده شدهٔ G آیا G مبهم است؟

$\mathrm{AMB_{CFG}} = \{\langle G\rangle :$ G یک گرامر مستقل از متن مبهم است$\}$

مسائل تصمیم‌ناپذیر در مورد زبان‌های مستقل از متن

- مسألهٔ تناظر پست PCP را به دو مسألهٔ $\mathrm{EMI_{CFG}}$ و $\mathrm{AMB_{CFG}}$ کاهش می‌دهیم.

- فرض کنید I یک نمونه از مسألهٔ تناظر پست باشد به طوری که جواب مسألهٔ I بلی باشد. به عبارت دیگر $\mathrm{I} \in \mathrm{PCP}$.

- آنگاه دنبالهٔ $i_1, i_2, \cdots, i_k$ وجود دارد به طوری که $b_{i_k} b_{i_{k-1}} \cdots b_{i_1} = t_{i_k} t_{i_{k-1}} \cdots t_{i_1}$. بنابراین داریم: $x = t_{i_k} t_{i_{k-1}} \cdots t_{i_1} c_{i_1} \cdots c_{i_k} = b_{i_k} b_{i_{k-1}} \cdots b_{i_1} c_{i_1} \cdots c_{i_k}$.

مسائل تصمیم‌ناپذیر در مورد زبان‌های مستقل از متن

- حال گرامر G را در نظر بگیرید به طوری که $L(G) = L(G_t) \cup L(G_b)$ بنابراین اگر S متغیر آغازی G باشد، داریم: $S \to S_t | S_b$.

- رشتهٔ x هم از گرامر G_t به دست می‌آید و هم از گرامر G_b و بنابراین دو اشتقاق دارد که یکی با $S \Rightarrow S_t$ آغاز می‌شود و دیگری با $S \Rightarrow S_b$. بنابراین G یک گرامر مبهم است. پس نشان دادیم به ازای هر نمونه از مسألهٔ PCP می‌توان گرامری ساخت که مبهم است.

- مسألهٔ PCP را به مسألهٔ $\mathrm{AMB_{CFG}}$ کاهش دادیم و می‌دانیم PCP تصمیم‌ناپذیر است، پس $\mathrm{AMB_{CFG}}$ تصمیم‌ناپذیر است.

- همچنین اگر رشتهٔ x وجود داشته باشد، این بدین معناست که $L(G_1) \cap L(G_2) \neq \emptyset$. پس به ازای هر نمونه از مسألهٔ PCP دو گرامر وجود دارند که اشتراک آنها غیر تهی است. بنابراین می‌توانیم بدین طریق PCP را به $\mathrm{EMI_{CFG}}$ کاهش دهیم، پس مسألهٔ $\mathrm{EMI_{CFG}}$ نیز تصمیم‌ناپذیر است.

تصمیم‌ناپذیری

- ثابت کنید زبان
$\mathrm{REGULAR_{TM}} = \{\langle M \rangle :$ یک زبان منظم است $L(M)$ یک ماشین تورینگ است و $M\}$ تصمیم‌ناپذیر است.

- فرض می‌کنیم ماشین تورینگ R زبان $\mathrm{REGULAR_{TM}}$ را تصمیم می‌گیرد. با استفاده از R ماشین S را می‌سازیم که زبان $\mathrm{A_{TM}}$ را تصمیم می‌گیرد. از آنجایی که می‌دانیم $\mathrm{A_{TM}}$ تصمیم‌ناپذیر است، پس فرض اولیه نادرست بوده و $\mathrm{REGULAR_{TM}}$ باید تصمیم‌ناپذیر باشد.

به ازای ورودی $\langle M, w\rangle$ در ماشین S به طوری که M یک ماشین تورینگ و w یک رشته است، چنین می‌کنیم:

- ابتدا یک ماشین $M_۲$ می‌سازیم که چنین عمل می‌کند. به ازای ورودی x به ماشین $M_۲$ اگر $x = ۰^n۱^n$ آنگاه ماشین $M_۲$ رشتهٔ ورودی را می‌پذیرد و اگر x به فرم دیگری بود، آنگاه ماشین $M_۲$ رشتهٔ x را می‌پذیرد اگر و تنها اگر M رشتهٔ w را بپذیرد. دقت کنید که اگر ماشین R ورودی $\langle M_۲\rangle$ را بپذیرد، آنگاه زبان ماشین $M_۲$ منظم است و این تنها در صورتی ممکن است که $M_۲$ همهٔ رشته‌های ورودی را بپذیرد، یعنی $L(M_۲) = \Sigma^*$. تنها در صورتی $M_۲$ همهٔ رشته‌های ورودی را می‌پذیرد که M رشتهٔ w را بپذیرد. اما اگر ماشین R ورودی $\langle M_۲\rangle$ را نپذیرد، آنگاه ماشین $M_۲$ برخی از مقادیر ورودی x را می‌پذیرد و برخی را رد می‌کند. همچنین ماشین $M_۲$ رشته‌های ورودی به شکل $۰^n۱^n$ را می‌پذیرد پس زبان آن منظم نیست. این تنها در صورتی ممکن است که M رشتهٔ w را نپذیرد.

تصمیم‌ناپذیری

- حال ماشین R را بر روی ورودی $\langle M_2 \rangle$ اجرا می‌کنیم.
- اگر R ورودی $\langle M_2 \rangle$ را پذیرفت ماشین S ورودی $\langle M, w \rangle$ را می‌پذیرد و اگر R ورودی $\langle M_2 \rangle$ را نپذیرفت، آنگاه S ورودی $\langle M, w \rangle$ را نمی‌پذیرد.

قضیهٔ رایس

- در حالت کلی، مسألهٔ تعیین کردن هر گونه ویژگی از زبانی که یک ماشین تورینگ M می‌پذیرد، یک مسألهٔ تصمیم‌ناپذیر است.

- بنابراین همهٔ مسائل به شکل زیر تصمیم‌ناپذیر هستند : تعیین کنید زبانی که توسط یک ماشین تورینگ تشخیص داده می‌شود، ویژگی P را دارد.

- ویژگی P در اینجا می‌تواند مستقل از متن بودن زبان، یا حتی متناهی بودن زبان باشد.

- قضیهٔ رایس [1]: فرض کنید P یک ویژگی از زبانی باشد که توسط یک ماشین تورینگ M پذیرفته می‌شود. مسألهٔ تعیین کردن اینکه ماشین تورینگ M ویژگی P را دارد تصمیم‌ناپذیر است.

- بنابراین زبان $P_{TM} = \{\langle M \rangle$: ویژگی P را دارد $L(M)$ یک ماشین تورینگ است و $M\}$ تصمیم‌ناپذیر است.

[1] Rice's theorem

مقایسهٔ زبان‌ها

	منظم	مستقل از متن	حساس به متن	شمارش‌پذیر بازگشتی
بسته‌بودن بر روی:				
الحاق	بله	بله	بله	بله
اجتماع	بله	بله	بله	بله
بستار–ستاره	بله	بله	بله	بله
متمم	بله	خیر	بله	خیر
اشتراک	بله	خیر	بله	بله
اشتراک با منظم	بله	بله	بله	بله
تصمیم‌پذیری:				
عضویت رشته در زبان	بله	بله	بله	خیر
تهی‌بودن زبان	بله	بله	خیر	خیر
محدود بودن زبان	بله	بله	خیر	خیر
مرجع بودن زبان	بله	خیر	خیر	خیر
برابری دو زبان	بله	خیر	خیر	خیر

پیچیدگی محاسباتی

پیچیدگی

- در مبحث محاسبه‌پذیری تصمیم‌پذیری مسائل محاسباتی را بررسی کردیم. به عبارت دیگر بررسی کردیم آیا برای یک مسأله الگوریتمی وجود دارد یا خیر.
- حتی اگر مسأله‌ای تصمیم‌پذیر باشد، ممکن است الگوریتمی که برای آن وجود دارد نیاز به زمان و حافظه‌ای بسیار زیاد داشته و الگوریتم در عمل غیرقابل استفاده باشد.
- در مبحث پیچیدگی بررسی می‌کنیم که محاسبات در عمل نیازمند چه میزان زمان و حافظه هستند. به عبارت دیگر بررسی می‌کنیم در عمل محاسبات چقدر کارآمد هستند.
- کارآمدی محاسبات را با توجه به میزان زمان و حافظهٔ مورد نیاز آنها توسط پیچیدگی زمانی و پیچیدگی حافظه [1] می‌سنجیم. در اینجا تنها در مورد پیچیدگی زمانی صحبت خواهیم کرد.

[1] time and space complexity

پیچیدگی

- برای محاسبهٔ پیچیدگی زمانی یک مسأله چند نکته را باید در نظر بگیریم: (۱) مدل محاسباتی که در آن محاسبه انجام می‌شود ماشین تورینگ است. (۲) اندازهٔ مسأله را با n نشان می‌دهیم. (۳) معمولا می‌خواهیم کارآمدی یک الگوریتم را به ازای n های بسیار بزرگ بسنجیم.
- می‌توانیم فرض کنیم که ماشین تورینگ یک حرکت در واحد زمان انجام می‌دهد، بنابراین می‌خواهیم محاسبه کنیم که برای انجام یک محاسبات معین برای مسأله‌ای با اندازهٔ n ماشین تورینگ چند حرکت انجام می‌دهد و چه اتفاقی می‌افتد وقتی n به سمت اعداد بسیار بزرگ میل می‌کند.
- پیچیدگی زمانی یک تابع از n است، بنابراین می‌گوییم پیچیدگی مسأله $T(n)$ است یا به عبارت دیگر برای مسأله با اندازهٔ n ماشین تورینگ $T(n)$ حرکت انجام می‌دهد.

پیچیدگی

- فرض کنید M یک ماشین تورینگ قطعی باشد که بر روی همهٔ ورودی‌ها متوقف می‌شود. پیچیدگی زمانی M تابعی است به صورت $f : \mathbb{N} \rightarrow \mathbb{N}$، به طوری که $f(n)$ حداکثر تعداد گام‌هایی است که M برای توقف بر روی یک ورودی با اندازهٔ n نیاز دارد.
- می‌گوییم زمان اجرای ماشین M (یا الگوریتم M) برای ورودی‌های با اندازهٔ n برابر است با $f(n)$.

پیچیدگی

- معمولا برای اندازه‌گیری زمان اجرای الگوریتم‌ها از تحلیل مجانبی [1] استفاده می‌کنیم. تحلیل مجانبی (تحلیل حدی) روشی برای توصیف حدی توابع است.

- در تحلیل مجانبی الگوریتم‌ها، می‌خواهیم زمان اجرای الگوریتم را به ازای مقادیر بسیار بزرگ n بدانیم.

- فرض کنید f و g دو تابع باشند. می‌گوییم $f(n) = O(g(n))$ اگر عدد صحیح c و $n_۰$ وجود داشته باشند به طوری که به ازای $n > n_۰$ داشته باشیم: $f(n) \leq cg(n)$.

- وقتی $f(n) = O(g(n))$ می‌گوییم $g(n)$ کران بالای مجانبی $f(n)$ است.

- برای مثال اگر داشته باشیم $f(n) = ۳n^۲ + ۲n + ۱$ آنگاه $f(n) = O(n^۲)$.

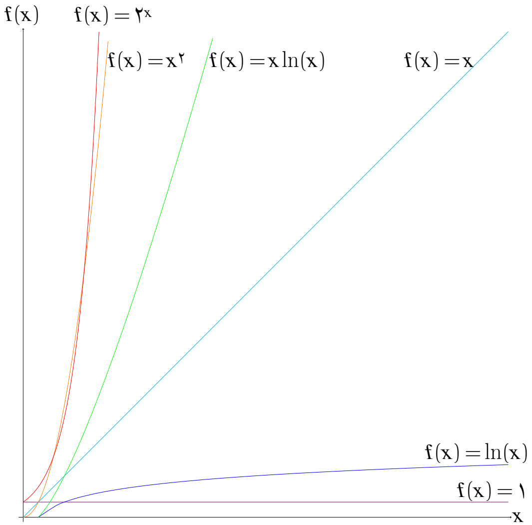
[1] asymptotic analysis

پیچیدگی

- فرض کنید $t : \mathbb{N} \to \mathbb{R}^+$ یک تابع باشد. کلاس پیچیدگی زمانی [1] $\mathrm{TIME}(t(n))$ مجموعهٔ همهٔ زبان‌هایی است که در زمان $O(t(n))$ قابل محاسبه هستند.

[1] time complexity class

مقایسه رشد توابع



مقایسه رشد توابع پیچیدگی

اگر هر گام فقط یک میکروثانیه زمان ببرد:

اندازهٔ n	۲۰	۴۰	۶۰
تابع پیچیدگی f(n)			
n	۰/۰۰۰۰۲ ثانیه	۰/۰۰۰۰۴ ثانیه	۰/۰۰۰۰۶ ثانیه
$n^۲$	۰/۰۰۰۴ ثانیه	۰/۰۰۱۶ ثانیه	۰/۰۰۳۶ ثانیه
$n^۳$	۰/۰۰۸ ثانیه	۰/۰۶۴ ثانیه	۰/۲۱۶ ثانیه
$n^۵$	۳/۲ ثانیه	۱/۷ دقیقه	۱۳ دقیقه
۲^n	۱ ثانیه	۱۲/۷ روز	۳۶۶ قرن
۳^n	۵۸ دقیقه	۳۸۵۵ قرن	$۱/۳ \times ۱۰^{۱۳}$ قرن

پیچیدگی

- از مقایسهٔ رشد توابع می‌توانیم نتیجه بگیریم رشد چندجمله‌ای برای پیچیدگی یک الگوریتم مطلوب و رشد نمایی بسیار نامطلوب است، چراکه حتی برای اندازه‌های بسیار کوچک (برای مثال ۶۰)، الگوریتمی با رشد نمایی به چندین قرن زمان نیاز دارد.

کلاس پی

- کلاس پی [1] رده‌ای از زبان‌ها است که در زمان چندجمله‌ای توسط ماشین تورینگ قطعی تصمیم‌پذیر هستند.
- به عبارت دیگر $P = \bigcup_k \mathrm{TIME}(n^k)$.
- کلاس پی کلاس مسائلی است که در عمل توسط کامپیوترها قابل حل شدن هستند.
- وقتی مسأله‌ای در کلاس پی باشد، آن مسأله در زمان چندجمله‌ای n^k به ازای عدد ثابت k قابل حل شدن است.

[1] class P

کلاس پی

- مسألهٔ پیدا کردن یک مسیر از یک رأس به رأس دیگر در یک گراف جهت‌دار در کلاس پی است.
- $\text{PATH} = \{\langle G, s, t\rangle :$ G یک گراف جهت‌دار است و یک مسیر از s به t وجود دارد $\}$
- $\text{PATH} \in \text{P}$ زیرا می‌توان از الگوریتم جستجوی سطح–اول [1] برای حل آن استفاده کرد که پیچیدگی آن از مرتبه $O(|V| + |E|)$ است. می‌توانیم اندازهٔ گراف را مجموع تعداد رأس‌ها و یال‌ها در نظر بگیریم، و بنابراین پیچیدگی الگوریتم برابر است با $O(n)$.

[1] Breadth–First Search (BFS)

کلاس پی

- آیا الگوریتمی در زمان چندجمله‌ای وجود دارد که به ازای دو عدد داده شده تعیین کند آیا آن دو عدد نسبت به هم اول‌اند یا خیر؟

- به عبارت دیگر زبان $\text{RELPRIME} = \{\langle x, y \rangle :$ نسبت به هم اول‌اند y و $x\}$ را در نظر بگیرید. آیا $\text{RELPRIME} \in P$ ؟

- الگوریتم اقلیدس، الگوریتمی است که به ازای دو عدد x و y در زمان $\log b$ بزرگترین مقسوم علیه مشترک دو عدد را می‌یابد، به طوری که $b = \min(x, y)$. بنابراین این زبان در کلاس پی است.

کلاس پی

- چند زبان (مسألهٔ) دیگر در کلاس پی:
- $\text{CONNECTED} = \{\langle G \rangle :$ یک گراف همبند است $G\}$
- $\text{EULERIAN\_CIRCUIT} = \{\langle G \rangle :$ یک گراف است که شامل دور اویلری است $G\}$
  دور اویلری یک دور ساده است که در آن هیچ یالی تکرار نشده است.
- $\text{PRIMES} = \{\langle x \rangle :$ یک عدد صحیح اول است $x\}$

کلاس ان‌پی

- برای برخی از مسأله‌ها هیچ الگوریتمی در زمان چند جمله‌ای یافت نشده است. یک احتمال این است که چنین الگوریتمی وجود دارد ولی هنوز کسی آن را نیافته است. احتمال دیگر این است که چنین الگوریتمی وجود ندارد و مسأله در زمان چندجمله‌ای قابل حل نیست.

- مسأله مسیر همیلتونی [1] را در نظر بگیرید. این مسأله می‌پرسد به ازای گراف داده شدهٔ G آیا گراف دارای یک مسیر همیلتونی از رأس s به رأس t است؟

- $\mathrm{HAMPATH} = \{\langle \mathrm{G}, \mathrm{s}, \mathrm{t} \rangle :$ دارای یک مسیر همیلتونی از رأس s به رأس t است $\mathrm{G}\}$

- مسیر همیلتونی مسیری است که از هر رأس گراف دقیقا یک بار عبور می‌کند.

- گرچه برای مسأله مسیر همیلتونی الگوریتمی در زمان چندجمله‌ای وجود ندارد، ولی به ازای یک مسیر داده شده می‌توانیم در زمان چندجمله‌ای بررسی کنیم آیا مسیر همیلتونی است یا خیر. پس بررسی (راستی آزمایی) [2] جواب مسألهٔ همیلتونی آسان‌تر از پیدا کردن جواب آن است.

[1] Hamiltonian path
[2] verifying

کلاس ان‌پی

- فرض کنید الگوریتم A ورودی x را دریافت می‌کند و تصمیم می‌گیرد که x عضو A است و سپس خروجی y را تولید می‌کند. یک تصدیق‌کنند [1] برای الگوریتم A الگوریتمی است که x و y را به عنوان ورودی دریافت می‌کند و بررسی می‌کند آیا A با ورودی x خروجی y را تولید می‌کند یا خیر.
- یک تصدیق کننده را تصدیق‌کنندهٔ چندجمله‌ای [2] می‌نامیم اگر به ازای ورودی x و y در زمان چندجمله‌ای نسبت به طول x تعیین کند آیا y جواب x است یا خیر.
- یک زبان را تصدیق‌پذیر چندجمله‌ای [3] می‌نامیم اگر یک تصدیق‌کنندهٔ چندجمله‌ای داشته باشد.
- برای مثال در مسألهٔ مسیر همیلتونی ورودی تصدیق کننده، $\langle G, s, t \rangle$ است و یک مسیر y . الگوریتم تصدیق کننده باید تصمیم بگیرد آیا y یک مسیر همیلتونی در گراف G از s به t است یا خیر.

---
[1] verifier
[2] polynomial time verifier
[3] polynomially verifiable

کلاس ان‌پی

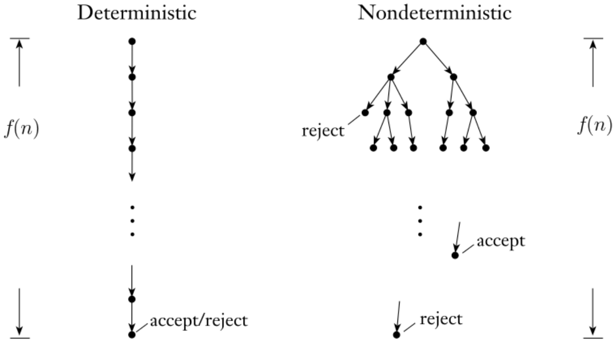
- کلاس ان‌پی [1] مجموعه‌ای است شامل همهٔ زبان‌های تصدیق‌شوندهٔ چندجمله‌ای.
- این دسته از زبان‌ها را بدین دلیل ان‌پی می‌نامیم که توسط یک ماشین تورینگ غیرقطعی [2] در زمان چندجمله‌ای [3] می‌توان آنها را تصمیم گرفت.
- فرض کنید یک ماشین تورینگ غیرقطعی در یک گام همهٔ جواب‌ها را حدس بزند، آنگاه در زمان چندجمله‌ای می‌توان جواب را بررسی کرد.

[1] class NP
[2] Nondeterministic Turing machine
[3] Polynomial time

کلاس ان‌پی

- دقت کنید که پیچیدگی زمانی برای ماشین غیرقطعی از شروع آغاز به کار ماشین است تا وقتی که همهٔ کپی‌های ماشین در همهٔ مسیرها به پایان برسند.

- در شکل زیر پیچیدگی زمانی برای هر دو ماشین قطعی و غیرقطعی $f(n)$ است. گرچه ماشین غیرقطعی محاسبات بیشتری انجام می‌دهد اما در زمان $f(n)$ به پایان می‌رسد.



کلاس ان‌پی

- کلاس پیچیدگی زمانی غیرقطعی[1] $\mathrm{NTIME}(t(n))$ مجموعهٔ همهٔ زبان‌هایی است که در زمان $O(t(n))$ توسط یک ماشین تورینگ غیرقطعی تصمیم‌پذیر باشند.

- بنابراین تعریف می‌کنیم: $\mathrm{NP} = \bigcup_k \mathrm{NTIME}(n^k)$

- برای اثبات ان‌پی بودن یک مسأله باید نشان دهیم که یک الگوریتم تصدیق‌کننده چندجمله‌ای برای آن وجود دارد.

- برای مثال مسأله پیدا کردن کلیک در یک گراف در کلاس ان‌پی است. یک کلیک مجموعه‌ای از رئوس در یک گراف است که هر جفت از آنها توسط یک یال به یکدیگر متصل هستند.
$\mathrm{CLIQUE} = \{\langle G, k\rangle$: یک گراف بی‌جهت است و شامل یک کلیک با k رأس است $\}$

[1] nondeterministic time complexity class

مسألهٔ پی در مقابل ان‌پی

- مسائل پی مسائلی هستند که برای تصمیم‌گیری آنها یک الگوریتم در زمان چندجمله‌ای وجود دارد.
- مسائل ان‌پی مسائلی هستند که برای تصدیق جواب آنها یک الگوریتم در زمان چندجمله‌ای وجود دارد.
- برای مسائل ان‌پی الگوریتم چندجمله‌ای تاکنون پیدا نشده است و هیچ‌کس نیز اثبات نکرده است که برای آنها الگوریتم چندجمله‌ای هرگز وجود نخواهد داشت.
- مسأله پی در مقابل ان‌پی [1] می‌پرسد آیا کلاس پی و ان‌پی برابر هستند یا خیر؟ به عبارت دیگر آیا $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ ؟

[1] P versus NP problem

مسألهٔ پی در مقابل ان‌پی

- آیا برای مسائل ان‌پی، که برای آنها تصدیق‌کنندهٔ چندجمله‌ای وجود دارد، تصمیم‌گیرندهٔ چندجمله‌ای نیز پیدا خواهد شد؟
- این مسأله یکی از هفت مسأله جایزهٔ هزاره [1] است.
- در سال ۲۰۱۰ یکی از مسائل جایزهٔ هزاره به نام مسألهٔ حدس پوانکاره [2] توسط ریاضیدان روسی به نام گریگوری پرلمان [3] حل شده است.

---

[1] Millennium prize problems
[2] Poincaré conjecture
[3] Grigori Perelman

ان‌پی کامل

- در دهۀ ۱۹۷۰ استیون کوک [1] و لئونید لوین [2] که بر روی مسألۀ پی در مقابل ان‌پی کار می‌کردند، متوجه شدند تعدادی از مسائل ان‌پی قابل تبدیل کردن به یکدیگرند، چنانچه اگر الگوریتمی چندجمله‌ای برای یکی از آنها پیدا شود، بقیۀ آنها نیز توسط یک الگوریتم چندجمله‌ای حل خواهند شد. این دسته از مسائل ان‌پی‌کامل [3] نامیده شدند.

[1] Stephen Cook
[2] Leonid Levin
[3] NP-complete problems

ان‌پی کامل

- فرض کنید تابع f تابعی است از جملات زبان A به جملات زبان B به طوری که $w \in A$ اگر و تنها اگر $f(w) \in B$.

- زبان A کاهش‌پذیر در زمان چندجمله‌ای به زبان B است، اگر الگوریتمی در زمان چندجمله‌ای برای محاسبه تابع f وجود داشته باشد.

- زبان B ان‌پی‌کامل است، اگر (۱) در کلاس ان‌پی باشد و (۲) همهٔ زبان‌های A در ان‌پی در زمان چندجمله‌ای قابل کاهش به B باشند.

- اولین مسأله‌ای که اثبات شد ان‌پی‌کامل است، مسألهٔ صدق‌پذیری [1] بود.

[1] satisfiability problem (SAT)

مسألهٔ صدق‌پذیری

- یک متغیر بولی متغیری است که مقدار آن درست [1] یا نادرست [2] باشد. می‌توانیم مقدار درست را با ۱ و مقدار نادرست را با ۰ نشان دهیم.
- با استفاده از متغیرهای بولی [3] و عملگرهای منطقی [4] مانند عطف و فصل و نقیض [5] می‌توانیم یک عبارت منطقی [6] بسازیم.

[1] true
[2] false
[3] Boolean variables
[4] Boolean operations
[5] conjunction (AND)، disjunction (OR)، negation (NOT)
[6] Boolean expression or Boolean formula

مسألهٔ صدق‌پذیری

- یک عبارت منطقی، در فرم نرمال عطفی [1] است اگر عطف چندین عبارت فصلی به صورت $e = (p_{1,1} \vee \cdots \vee p_{1,m_1}) \wedge (p_{2,1} \vee \cdots \vee p_{2,m_2}) \wedge \cdots \wedge (p_{n,1} \vee p_{n,2} \vee \cdots \vee p_{n,m_n})$ باشد.
- مسألهٔ صدق‌پذیری بدین صورت بیان می‌شود: برای یک عبارت منطقی e در فرم نرمال عطفی، تعیین کنید آیا مقادیری از متغیرها وجود دارند به طوری که به ازای آن مقادیر، مقدار عبارت منطقی e درست باشد.
- برای مثال مقدار عبارت $e_1 = (\overline{x_1} \vee x_2) \wedge (x_1 \vee x_3)$ به ازای $x_1 = 0$، $x_2 = 1$، $x_3 = 1$ درست است، اما عبارت $e_2 = (x_1 \vee x_2) \wedge \overline{x_1} \wedge \overline{x_2}$ به ازای هیچ مقداری صدق‌پذیر نیست.

[1] conjunctive normal form (CNF)

مسألهٔ صدق‌پذیری

- مسألهٔ صدق‌پذیری بدین صورت بیان می‌شود: به ازای عبارت منطقی داده شدهٔ e در فرم نرمال عطفی، آیا e صدق‌پذیر است؟
- $\text{SAT} = \{\langle e \rangle :$ یک عبارت منطقی صدق‌پذیر در فرم نرمال عطفی است e$\}$
- می توانستیم زبان SAT را بدین صورت نیز تعریف کنیم:
- $\text{SAT} = \{\langle e \rangle :$ یک عبارت منطقی صدق‌پذیر است e$\}$
- از عبارت منطقی در فرم نرمال عطفی بدین دلیل استفاده می‌کنیم که به یک فرم ساده و استاندارد است و همهٔ عبارت‌های منطقی می‌توانند در زمان چندجمله‌ای بدین فرم تبدیل شوند.

مسائل ان‌پی‌کامل

- یک حالت خاص مسأله SAT هنگامی است که عبارت منطقی داده شده در فرمال نرمال عطفی سه‌تایی به صورت $e = (p_{۱,۱} \vee p_{۱,۲} \vee p_{۱,۳}) \wedge (p_{۲,۱} \vee p_{۲,۲} \vee p_{۲,۳}) \wedge \cdots \wedge (p_{n,۱} \vee p_{n,۲} \vee p_{n,۳})$ باشد.
- زبان ۳SAT یک حالت خاص از زبان SAT و بنابراین ان‌پی‌کامل است.
- $\text{۳SAT} = \{\langle e \rangle :$ یک عبارت منطقی صدق‌پذیر در فرم نرمال عطفی سه‌تایی است $e\}$

مسائل ان‌پی‌کامل

- می‌توان در زمان چندجمله‌ای مسألهٔ ۳SAT را به CLIQUE کاهش داد. بنابراین مسأله CLIQUE نیز ان‌پی‌کامل است.

- یک پوشش رأسی با k رأس مجموعه‌ای از k رأس است که همهٔ یال‌ها را در یک گراف پوشش می‌دهد.

VERTEX-COVER =

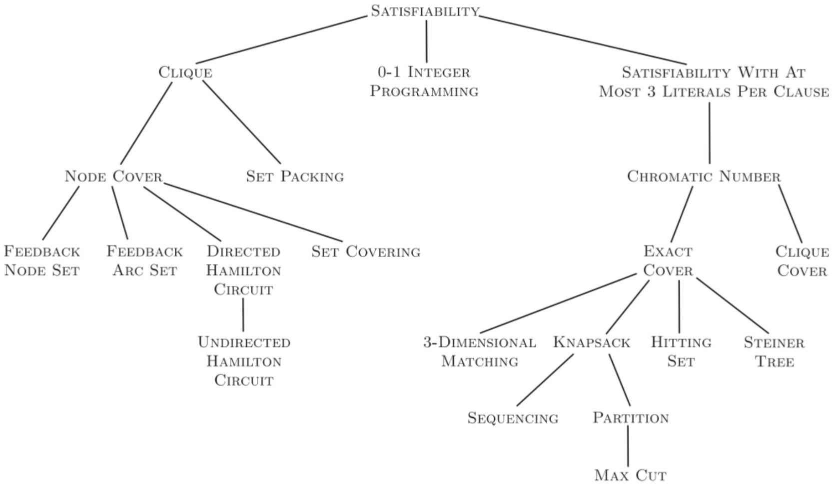
$\{\langle G, k \rangle$: G یک گراف بدون‌جهت است که شامل یک پوشش رأسی با k رأس است$\}$

می‌توان ۳SAT را به VERTEX-COVER نیز کاهش داد و بنابراین VERTEX-COVER نیز ان‌پی‌کامل است.

- همچنین مسألهٔ ۳SAT قابل کاهش به HAMPATH است و بنابراین مسألهٔ پیدا کردن مسیر همیلتونی نیز ان‌پی‌کامل است.

مسائل ان‌پی‌کامل

- در سال ۱۹۷۲ ریچارد کارپ بیست مسأله را کاهش داد و نشان داد ۲۱ مسألهٔ محاسباتی ان‌پی‌کامل هستند.



مثال

- اثبات کنید مسألهٔ CLIQUE ان‌پی کامل است.
- یک کلیک مجموعه‌ای از رئوس در یک گراف است که هر جفت از آنها توسط یک یال به یکدیگر متصل هستند.
  $\text{CLIQUE} = \{\langle G, k\rangle :$ یک گراف بی‌جهت است و شامل یک کلیک با k رأس است $\}$

مثال

- اثبات کنید مسألهٔ CLIQUE ان‌پی کامل است.
- برای اثبات ان‌پی کامل بودن مسألهٔ CLIQUE ، مسأله ۳SAT را با یک الگوریتم چندجمله‌ای به آن کاهش می‌دهیم.
- فرض کنید ϕ یک عبارت منطقی در فرم نرمال عطفی سه‌تایی به صورت $\phi = (a_۱ \vee b_۱ \vee c_۱) \wedge (a_۲ \vee b_۲ \vee c_۲) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$ باشد.
- الگوریتم کاهش f در زمان چندجمله‌ای، با استفاده از عبارت ϕ، رشتهٔ $\langle G, k \rangle$ را تولید می‌کند به طوری G یک گراف بی‌جهت باشد.
- به ازای هر یک از متغیرها در یک گروه $(a_i \vee b_i \vee c_i)$ یک رأس در نظر می‌گیریم. بنابراین با استفاده از عبارت منطقی ϕ یک گراف دارای ۳k رأس تولید می‌کنیم.

مثال

- دقت کنید یک عبارت منطقی بر روی تعدادی متغیر $x_۱, \cdots, x_n$ تعریف شده است. بنابراین متغیرها به صورت x_m یا $\overline{x_m}$ هستند.

- همچنین توجه کنید برای گراف‌هایی که دارای ۳k رأس نیستند، عبارت $(a_i \vee b_i \vee c_i)$ می‌تواند شامل دو یا سه متغیر تکراری باشد. به طور مثال در یک عبارت می‌توانیم داشته باشیم $(x_۱ \vee x_۱ \vee \overline{x_۳})$ و یا $(x_۲ \vee x_۲ \vee x_۲)$.

- متغیر x_p در گروه i را به صورت $(x_p)_i$ نشان می‌دهیم.

مثال

- حال برای رسم یال‌ها، اولا هیچ یالی از یک رأس در یک گروه به رأسی دیگر در همان گروه متصل نمی‌کنیم.
- دوما، از هر یک از رئوس گروه i به هر یک از رئوس گروه j یال‌های زیر را متصل می‌کنیم:

۱. $(x_p)_j$ $\bullet\!\!-\!\!\bullet$ $(x_p)_i$ ، $(\overline{x_p})_j$ $\bullet\!\!-\!\!\bullet$ $(\overline{x_p})_i$

۲. $(x_q)_j$ $\bullet\!\!-\!\!\bullet$ $(x_p)_i$ ،$(\overline{x_q})_j$ $\bullet\!\!-\!\!\bullet$ $(\overline{x_p})_i$ ،$(x_q)_j$ $\bullet\!\!-\!\!\bullet$ $(\overline{x_p})_i$ ،$(\overline{x_q})_j$ $\bullet\!\!-\!\!\bullet$ $(x_p)_i$ به طوری که $p \neq q$

- پس همهٔ یال‌ها به غیر از یال‌های $(\overline{x_p})_j$ $\bullet\!\!-\!\!\bullet$ $(x_p)_i$ ، $(x_p)_j$ $\bullet\!\!-\!\!\bullet$ $(\overline{x_p})_i$ را رسم می‌کنیم.

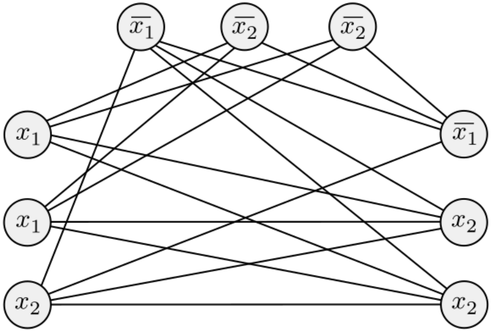
مثال

- حال توجه کنید که اگر عبارت ϕ تصدیق‌پذیر باشد، آنگاه مجموعه‌ای از k متغیر در k گروه وجود دارند که مقدار آنها درست است و این مجموعه که شامل k متغیر است، شامل هیچ دو متغیر x_m و $\overline{x_m}$ نیست، چون x_m و $\overline{x_m}$ نمی‌توانند همزمان هر دو درست باشند.
- این مجموعه که عبارت ϕ را تصدیق‌پذیر می‌کند، یک کلیک با k رأس را در گراف تولید شدهٔ G نشان می‌دهد. بدین ترتیب مسأله ۳SAT را به مسألهٔ CLIQUE کاهش دادیم. پس CLIQUE ان‌پی‌کامل است.

مثال

- عبارت $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$ را می‌توان به صورت گراف زیر درآورد.

- عبارت ϕ به ازای $\overline{x_1} = 1$ و $x_2 = 1$ صدق پذیر است. همچنین در گراف تولید شده کلیک $(x_2, \overline{x_1}, \overline{x_1})$ وجود دارد.



مسائل ان‌پی‌سخت

- یک مسأله ان‌پی‌سخت [1] است اگر بتوان همهٔ مسائل ان‌پی را به آن مسأله کاهش داد، گرچه آن مسأله خود در ان‌پی نباشد.

- بنابراین ممکن است برای یک مسألهٔ ان‌پی‌سخت هیچ تصدیق‌کننده‌ای در زمان چندجمله‌ای وجود نداشته باشد.

[1] NP-hard

مسائل بهینه‌سازی

- تاکنون تنها در مورد مسائل تصمیم‌گیری [1] صحبت کردیم. پاسخ مسائل تصمیم‌گیری بلی یا خیر است. به عبارت دیگر می‌پرسیم آیا الگوریتمی وجود دارد که تعیین کند یک ورودی متعلق به یک زبان است یا خیر. الگوریتم‌های تصمیم‌گیری را از نظر درجهٔ سختی بر اساس زمان مورد نیاز برای تصمیم‌گیری دسته‌بندی کردیم.

- دسته‌ای دیگر از مسائل به نام مسائل بهینه‌سازی [2] وجود دارند که در اینگونه مسائل به دنبال بهترین یا بهینه‌ترین پاسخ از بین مجموعه‌ای از پاسخ‌ها می‌گردیم.

- برای مثال مسألهٔ VERTEX-COVER یک مسألهٔ تصمیم‌گیری است که در آن می‌پرسیم آیا به ازای یک گراف داده شدهٔ G یک مجموعه از k رأس وجود دارد که همهٔ یال‌ها را پوشش دهد یا خیر. در مسألهٔ MIN-VERTEX-COVER که یک مسألهٔ بهینه‌سازی است به دنبال کوچک‌ترین مجموعه از رئوس می‌گردیم که همهٔ یال‌ها را پوشش دهند.

---

[1] decision problem
[2] optimization problems

مسائل بهینه‌سازی

- هر مسألهٔ بهینه‌سازی یک مسألهٔ تصمیم‌گیری متناظر آن دارد که از لحاظ درجهٔ سختی با آن در یک کلاس قرار دارند.

- در مسائل تصمیم‌گیری می‌پرسیم به ازای یک مقدار k برای یک مسأله پاسخی وجود دارد یا خیر. در مسائل بهینه‌سازی می‌پرسیم به ازای همهٔ مقادیر k بهینه‌ترین (کوچک‌ترین، بزرگ‌ترین، کوتاهترین، بلندترین، ...) مقدار چیست؟

- برای مثال در مسألهٔ تصمیم‌گیری رنگ‌آمیزی گراف می‌پرسیم آیا رئوس یک گراف با ۴ رنگ قابل رنگ‌آمیزی هستند به طوری که هیچ دو رأس مجاوری هم‌رنگ نباشند. در مسألهٔ بهینه‌سازی رنگ‌آمیزی گراف می‌پرسیم کمترین تعداد رنگ‌هایی که با آن می‌توان یک گراف را رنگ‌آمیزی کرد چیست؟

- از آنجایی که برای حل یک مسألهٔ بهینه‌سازی ممکن است به زمانی از مرتبه نمایی احتیاج داشته باشیم، لذا معمولا در اینگونه مسائل به دنبال یک پاسخ تقریبی توسط یک الگوریتم تقریبی [1] می‌گردیم.

[1] approximation algorithm