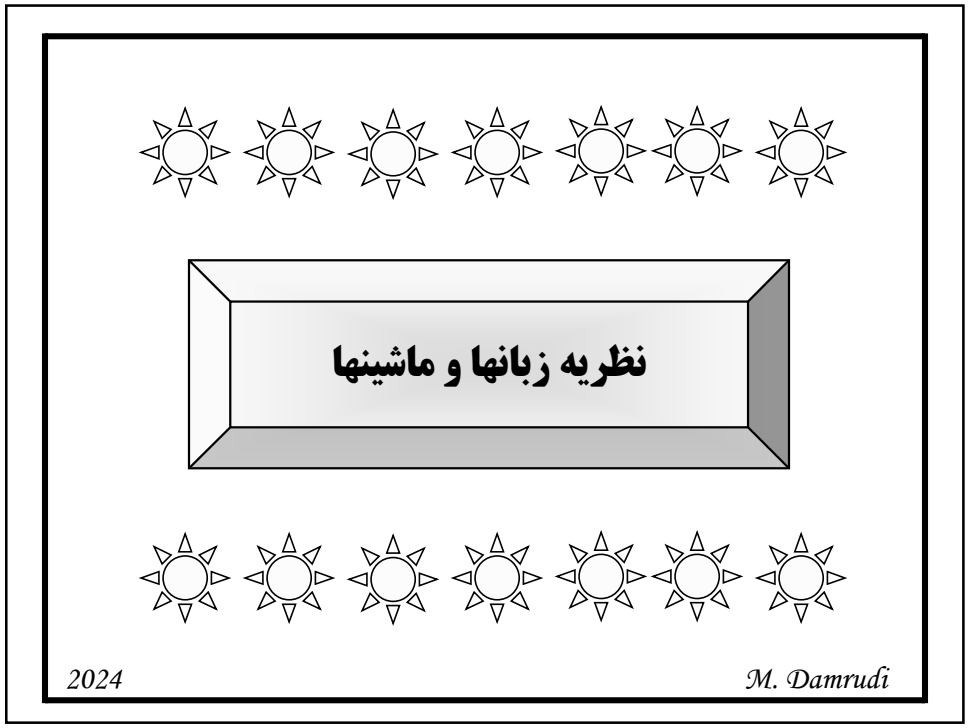


زبانها و عبارات با قاعده

زبان: یک زبان یک مجموعه از رشته ها روی یک الفبا است.

رشته: یک رشته روی یک مجموعه x یک دنباله متناهی از عناصرمجموعه x است.

الفبای زبان: به مجموعه عناصری که رشته‌ها از آن ساخته می‌شوند الفبای زبان گویند.

رشته تهی: رشته فاقد عنصر را رشته تهی نامیده و با λ نشان می دهیم.

تعریف: فرض کنید $\sum$ مجموعه الفبایی باشد در آنصورت $\sum^*$ شامل رشته هایی روی مجموعه $\sum$ است که به صورت زیر تعریف می شود.

۱) پایه: همواره $\lambda \in \sum^*$

۲) خاصیت بازگشتی: اگر $w \in \sum^*$ و $a \in \sum$ در آنصورت $wa \in \sum^*$

۳) خاصیت بسته بودن: $w \in \sum^*$ اگر و فقط اگر بتوان آن را از λ و با استفاده متناهی از قانون بازگشتی بدست آورد.

زبانها و عبارات با قاعده

فرض کنید که $\sum = \{a,b,c\}$ باشد. عناصر $\sum^*$ شامل:

λ : طول ۰

a b c: طول ۱

aa ab ac ba bb bc ca cb cc: طول ۲

aaa aab aac aba abb abc aca acb acc: طول۳

baa bab bac bba bbb bbc bca bcb bcc

caa cab cac cba cbb cbc cca ccb ccc

$\sum^*=\{\lambda,a,b,c,aa,ab,ac,\ldots\}$

۳۲ دو جمله ای-۳۱ یک جمله ای

اگر $\sum$ شامل n عنصر باشد، n^k رشته به طول k در $\sum^*$ وجود دارد.

یک زبان شامل رشته هایی روی الفبا است.

تعریف زبان: یک زبان روی یک الفبای $\sum$ یک زیر مجموعه از $\sum^*$ است.

زبانها و عبارات با قاعده

فرض کنید که $u,v \in \sum^*$ باشد. الحاق (concatination) uوv، که به صورت uv نوشته می شود یک عملگر دوتایی (دو عملوند نیاز دارد) روی $\sum^*$ است که به صورت زیر تعریف می شود:

۱) پایه: اگر length(v)=0 باشد. آنگاه $v=\lambda$ و uv=u خواهد بود.

۲) قانون بازگشتی: اگرv یک رشته با طول length(v)=n›0 باشد. در اینصورت، به ازای برخی رشته های w با طول length(w)=(n-1) n-1 و $a \in \sum$، v=wa و در نتیجه uv=(uw)a=u(wa) خواهد بود.

الحاق: الحاق یک عملگر دو تایی است که دو رشته را به عنوان ورودی گرفته و با چسباندن آنها در کنار هم یک رشته جدید ایجاد می کند.

الحاق عمل اصلی در تولید رشته هاست.

زبانها و عبارات با قاعده

مثال: $\sum=\{a,b,c\}$ و $u=abb$ و $v=bcc$

$uv=(abb)(bcc)=(abb(bc))c=(abb(b))cc=(abb(\lambda))bcc=abbbcc$

مثال: فرض کنید که $u=ab$, $v=ca$ و $w=bb$ باشد. در این صورت:

$uv=abca$ $uw=abbb$

$(uv)w=abcabb$ $u(vw)=abcabb$

نکته:

$uv \neq vu$ $u^3=uuu$

$u=ab \Rightarrow u^2=abab \neq a^2b^2=aabb$

فرض کنید که $u,v,w \in \sum^*$ باشد. در این صورت $(uv)w= u(vw)$ خواهد بود.

زبانها و عبارات با قاعده

تعریف زیررشته یا substring:

u زیر رشته ای از v است اگر رشته های $x,y \in \sum^*$ موجود باشند به گونه ای که $v= xuy$

اگر $x=\lambda$ باشد در آنصورت u پیشوندی (prefix) از رشته v است.

اگر $y=\lambda$ باشد در آنصورت u پسوندی (suffix) از رشته v است.

معکوس رشته:

فرض کنید که $w \in \sum^*$ درآنصورت معکوس w (reversal) که با w^R نشان می دهیم به صورت زیر تعریف می شود:

۱) حالت پایه: $length(w)=0$. در این صورت $u=\lambda$ و $\lambda^R=\lambda$ خواهد بود.

۲) قانون بازگشتی: اگر $length(w)=n\geq 1$ باشد، در اینصورت $length(u)=(n-1)$ و $w=ua$, $a \in \sum$ معکوس رشته برابرخواهد بود: $w^R=(ua)^R=au^R$

قضیه: فرض کنید که $u,v \in \sum^*$ باشد. در این صورت $(uv)^R= v^Ru^R$ است.

زبانها و عبارات با قاعده

الحاق دو زبان X,Y:

الحاق زبانهای x وy که به صورت xy نشان می دهیم، زبان

$xy=\{uv \mid u \in x \text{ and } v \in y\}$ است.

x^n مرتبه الحاق x با خودش را به صورت x^n نشان می دهیم. x^0 به صورت $\{\lambda\}$ تعریف می شود.

فرض کنید که x یک مجموعه باشد. در این صورت:

$$x^* = \bigcup_{i=0}^{\infty} x^i \qquad x^+ = \bigcup_{i=1}^{\infty} x^i$$

x^* شامل تمامی رشته هایی است که می توانند از عناصر x ساخته شوند.

x^+ مجموعه رشته های غیر تهی ایجاد شده از x است.

نکته: $u^+=u^*u$

زبانها و عبارات با قاعده

مثال: فرض کنید که $x=\{a,b,c\}$ و $y=\{abb,ba\}$ است. در اینصورت

$xy=\{aabb,babb,cabb,aba,bba,cba\}$

$x^0=\{\lambda\}$

$x^1=x=\{a,b,c\}$

$x^2=xx=\{aa,ab,ac,ba,bb,bc,ca,cb,cc\}$

$x^3=x^2x=\{aaa,aab,aac,aba,abb,abc,aca,acb,acc,baa,bab,bac,$
$bba,bbb,bbc,bca,bcb,bcc,caa,cab,cac,cba,cbb,cbc,cca,ccb,ccc\}$

مثال: مجموعه L یک زبان است که شامل رشته هایی روی $\sum=\{a,b\}$ است که هر رشته دارای حداقل یک زیررشته bb است.

$L=\{a,b\}^*\{bb\}\{a,b\}^*$

الحاق مجموعه {bb} ما را از وجود bb در هر رشته از L مطمئن می سازد. مجموعه های $\{a,b\}^*$ مشخص می کنند که هر تعدادی a وb با هر ترتیبی می تواند بعد یا قبل از bb قرار بگیرند.

زبانها و عبارات با قاعده

مجموعه منظم (باقاعده) Regular set: مجموعه ای با قاعده است که بتواند با استفاده از عملیات اجتماع، الحاق و * از مجموعه تهی، مجموعه شامل رشته تهی و اعضای مجموعه الفبا تولید شود.

فرض کنید که∑ یک الفبا باشد. مجموعه های باقاعده روی ∑ به صورت بازگشتی زیر تعریف می شوند:

۱) پایه: Ø، {λ}و{a} ،به ازاء هر a ϵ ∑، مجموعه هایی باقاعده روی∑ هستند.

۲) حالت بازگشتی: فرض کنید که X وY مجموعه هایی باقاعده روی ∑ باشند. مجموعه هایXY و XUY و X* مجموعه هایی باقاعده روی ∑ هستند.

۳) خاصیت بسته بودن: X یک مجموعه باقاعده روی ∑ است اگر بتواند با تکرار متناهی از مرحله گام بازگشت از عناصر پایه بدست آید.

مثال: مجموعه رشته هایی که با یک a شروع شده و شامل حداقل یک b هستند و به a ختم شود، مجموعه ای با قاعده روی {a,b} است.
{a}{a,b}*{b}{a,b}*{a}

زبانها و عبارات با قاعده

فرض کنید که∑ یک الفبا است. عبارات باقاعده روی∑ به صورت بازگشتی زیر تعریف می شوند:

۱) پایه:Ø، λ وa ،به ازاء هر a ϵ ∑، عباراتی باقاعده روی∑ هستند.

۲) قانون بازگشتی: فرض کنید که u,v عباراتی باقاعده روی∑ باشند. در این صورت عبارات uv, uUv و u* نیز عباراتی باقاعده روی∑ هستند.

۳) خاصیت بسته بودن: u یک عبارت باقاعده روی∑ است اگر و فقط اگر بتوان آن را از عناصر حالت پایه و با استفاده متناهی از قانون بازگشتی بدست آورد.

مثال: عبارت باقاعده ای بنویسید روی ∑={a,b} که رشته های تولید شده توسط آن حداقل دارای ۲ تا b باشد.

(i a*ba*b(aUb)*

(ii (aUb)*ba*ba*

(iii (aUb)*b(aUb)*b(aUb)*

زبانها و عبارات با قاعده

مثال: عبارت منظمی روی $\sum=\{a,b\}$ بنویسید که دقیقا دارای 2 تا b باشد.

$a^*ba^*ba^*$

مثال: عبارت منظمی روی $\sum=\{a,b\}$ بنویسید که مولد رشته هایی با تعداد زوج b باشد.

$a^* (ba^*ba^*)^*$ یا $a^* (a^*ba^*ba^*)^*$

مثال: عبارت منظمی بنویسید که شامل زیررشته {aa} نباشد.

$(b \cup ab)^* (a \cup \lambda)$

مثال: عبارت باقاعده‌ای روی $\sum=\{a,b,c\}$ بنویسید که شامل حداقل زیر رشته bc باشد.

$(a \cup b \cup c)^* bc (a \cup b \cup c)^*$

مثال: عبارت باقاعده ای روی $\sum=\{a,b,c\}$ بنویسید که شامل زیررشته bc نباشد.

$c^*(b \cup (ac^*))^*$

مثال: مجموعه $\{ba\ w\ ab \mid w \in \{a,b\}^*\}$ روی $\sum=\{a,b\}$ منظم است عبارت منظم معادل با آن به صورت $ba\ (a \cup b)^*\ ab$ بیان می شود.

زبانها و عبارات با قاعده

مثال: عبارت های منظم زیر بیانگر چه رشته هایی است.

$(a \cup b)^* aa (a \cup b)^*$ حداقل یک aa دارد

$(a \cup b)^* bb (a \cup b)^*$ حداقل یک bb دارد

$(a \cup b)^* aa (a \cup b)^* \cup (a \cup b)^* bb (a \cup b)^*$

$(a \cup b)^* (aa \cup bb) (a \cup b)^*$

حداقل یک aa یا bb دارد

ویژگیهایی در عبارات باقاعده:

$u(v \cup w)=uv \cup uw$	$(u \cup v)w = uw \cup vw$	$(u \cup v)^* = (u^*v^*)^*$
$u \cup v=v \cup u$	$(uv)^*u = u(vu)^*$	$u^*u=uu^*$
$uu^* \cup \lambda=u^*$	$u^* = (u^*)^*$	$u \cup u=u$
$(u \cup v)^* = u^*(v\ u^*)^*$	$(u \cup v)^* = (u^* \cup v)^*$	$(u \cup v)^* = (u^* \cup v^*)^*$

گرامرها و زبانهای مستقل از متن

به یک رشته درست از لحاظ نحوی، یک **جمله (sentence)** از زبان اطلاق می‌شود.

عناصر الفبا به عناصر پایانی(ترم) زبان موسومند.

عناصر اضافی مورد استفاده در فرآیند تولید جملات جهت اجرای محدودیتهای نحوی زبان به متغیرها یا عناصر غیر پایانی موسومند.

گرامر مستقل از متن: یک گرامر مستقل از متن، یک چهارتایی $(V,\sum,P,S)$ است

V یک مجموعه متناهی از متغیرها

$\sum$ (الفبا)یک مجموعه متناهی از نمادهای پایانی

P یک مجموعه متناهی از قوانین

S عضوی از مجموعه V می باشد که به آن سرترم گرامر گویند

فرض می شود که Vو$\sum$مجموعه هایی غیر الحاقی (جدا از هم)هستند.

گرامرها و زبانهای مستقل از متن

قانون(**rule**):

هر قانون عنصری از مجموعه $V\times(VU\sum)^*$ است. قانون $[A,w]$ معمولاً به صورت $A\rightarrow w$ نوشته می شود.

نکته: از آنجائیکه رشته تهی در $(VU\sum)^*$ وجود دارد، لذا λ نیز ممکن است در سمت راست یک قانون قرار گیرد.

قانونی به شکل $A\rightarrow \lambda$ به قانون تهی یا قانون لامبدا موسوم است.

مرحله اصلی در فرایند تولید، تبدیل یک رشته با استفاده از قانون است.

مثال: بکارگیری قانون $A \rightarrow w$ برای متغیر A در uAv رشته uwv را تولید می کندکه آن را به صورت $uAv \Rightarrow uwv$ نشان می دهیم.

یک رشته w از v قابل اشتقاق است اگر یک دنباله متناهی از قوانین که v را به w تبدیل می کنند وجود داشته باشد. $v \xrightarrow{*} w$

گرامرها و زبانهای مستقل از متن

فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن و $u \in (VU\sum)^*$ باشد. مجموعه رشته های قابل اشتقاق از u به صورت زیر تعریف می شود:

۱) پایه: u از u قابل اشتقاق است.

۲) قانون بازگشتی: اگر $w=xAy$ از u قابل اشتقاق بوده و $A \rightarrow w_1 \in P$ باشد، آنگاه xw_1y از u قابل اشتقاق خواهد بود.

۳) خاصیت بسته بودن: تمامی رشته های ایجاد شده از u با بکارگیری تعداد متناهی قانون بازگشتی از u قابل اشتقاق هستند. تعداد قوانین مورد استفاده طول اشتقاق را تعیین می کند.

با n بار استفاده از قانون گرامر G می توانیم رشته V را به W اشتقاق دهیم.

نکته: $V \xrightarrow{n} W$

یک گرامر شامل یک الفبا یک روش تولید رشته ها است. این رشته ها ممکن است شامل متغیرها و عناصر پایانی باشند.

گرامرها و زبانهای مستقل از متن

فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن باشد.

فرم جمله ای: یک رشته $w \in (VU\sum)^*$ یک فرم جمله ای (sentential form) از G است اگر یک اشتقاق $S \overset{*}{\Longrightarrow} w$ در G وجود داشته باشد.

جمله: یک رشته $w \in \sum^*$ یک جمله از G است اگر یک اشتقاق $S \overset{*}{\Longrightarrow} w$ در G وجود داشته باشد. هر جمله ای حتما فرم جمله ای است.

زبان G: که آنرا با $L(G)$ نشان می دهند، مجموعه زیراست.

$$L(G)=\{w \in \sum^* \mid S \overset{*}{\Rightarrow} w\}$$

فرم جمله ای ها، رشته هایی قابل اشتقاق ازعنصرابتدایی گرامرهستند.

جملات فرم جمله ای هایی هستند که تنها شامل عناصر پایانی می باشند.

به مجموعه ای از رشته ها روی یک مجموعه الفبا ($\sum$) زبان مستقل از متن گفته می شود.

گرامرها و زبانهای مستقل از متن

یک قانون A به شکل $u \rightarrow uAv$ را بازگشت مستقیم می نامیم.

$A \rightarrow bA$

به یک اشتقاق $A \overset{+}{\rightarrow} W \rightarrow uAv$ که A در w نیست، بازگشت غیر مستقیم می گوییم.

$A \rightarrow aB$

$B \rightarrow CA$

تعریف: مجموعه ای از رشته های روی الفبا $\sum$ یک زبان مستقل از متن (context free) نامیده می شود اگر گرامر مستقل از متنی وجود داشته باشد که آن زبان را تولید کند.

دو گرامر معادل یا یکسان گفته می شود اگر هر دو زبان یکسانی را تولید کند.

یکی از مزایای بسیار مهم گرامرهای مستقل از متن قابلیت انعطاف پذیری در مراحل اشتقاق است.

گرامرها و زبانهای مستقل از متن

مثال: گرامر مستقل از متن زیر را در نظر بگیرید. $G=(V,\sum,P,S)$

$V=\{S,A\}$

$\sum=\{a,b\}$

P: $S \rightarrow AA$

$A \rightarrow AAA \mid bA \mid Ab \mid a$

- ✓ منظور از متغیر این است که A می تواند تغییر کند.
- ✓ به عناصر V متغیر یا ترم های ناپایانی گویند.
- ✓ به عناصر $\sum$ ترم های پایانی گویند.
- ✓ همیشه از سرترم گرامر شروع کرده با انجام عملیات اشتقاق آنقدر ادامه می دهیم که به λ می رسیم یا تمام عناصر جز $\sum$ است.

گرامرها و زبانهای مستقل از متن

مثال: رشته w=ababaa را در نظر بگیرید. اشتقاق آن را از گرامر فوق بدست آورید.

S=>AA
=>aA
=>aAAA
=>abAAA
=>abaAA
=>ababAA
=>ababaA
=>ababaa

اشتقاق چپ (left most derivation): اشتقاقی که در آن همواره سمت چپ ترین متغیر فرم جمله ای بسط داده می شود.

اشتقاق راست (right most derivation): اشتقاقی که در آن همواره سمت راست ترین متغیر فرم جمله ای بسط داده می شود.

استقاقهای چپ و راست یکتا نیستند.

گرامرها و زبانهای مستقل از متن

درخت اشتقاق:

فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن بوده و $S \Rightarrow^{*} w$ یک اشتقاق باشد.درخت اشتقاق (derivation tree) آن با نماد DT نمایش داده می شود. یک درخت مرتب است که برای هر اشتقاق به صورت زیر ساخته می شود:

۱) درخت اشتقاق (DT)را با ریشه S مقداردهی کنید.

۲) اگر $A \rightarrow x_1x_2x_3....x_n$ با $x_1 \epsilon (V \cup \sum)$ قانونی در اشتقاق بکار رفته برای رشته uAv باشد، آنگاه $x_1x_2x_3....x_n$ را به عنوان فرزندان گره A به درخت اضافه کنید.

۳) اگر $A \rightarrow \lambda$ قانونی در اشتقاق بکار رفته برای رشته uAv باشد، آنگاه λ را به عنوان تنها فرزند A به درخت اضافه کنید.

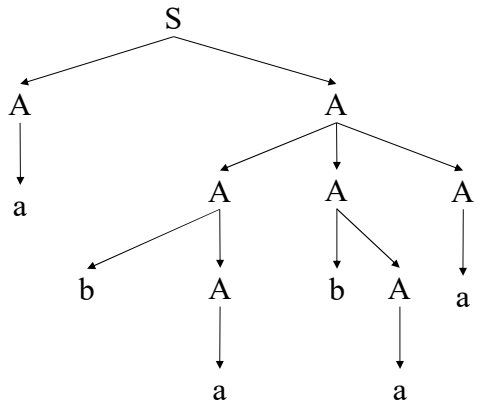
هر درخت اشتقاق با یک اشتقاق متناظر است. ممکن است اشتقاقهای مختلف درخت اشتقاق یکسان داشته باشند.

گرامرها و زبانهای مستقل از متن

اشتقاق

S=>AA
=>aA
=>aAAA
=>abAAA
=>abaAA
=>ababAA
=>ababaA
=>ababaa

درخت اشتقاق



گرامرها و زبانهای مستقل از متن

مثال: فرض کنید که G گرامری با قوانین زیر باشد:

$S \rightarrow aSa \mid aBa$

$B \rightarrow bB \mid b$

زبان گرامر را بدست آورید.

$S \Rightarrow a^nSa^n$

$\Rightarrow a^{n+1}Ba^{n+1}$

$\Rightarrow a^{n+1}b^mBa^{n+1}$

$\Rightarrow a^{n+1}b^{m+1}a^{n+1}$

$L(G)=\{a^{n+1}b^{m+1}a^{n+1} \mid n, m \geq 0\}$

$L(G)=\{a^nb^ma^n \mid n, m > 0\}$

گرامرها و زبانهای مستقل از متن

گرامرهای باقاعده

گرامر باقاعده (منظم): یک گرامر مستقل از متن است که هر قانون آن دارای یکی از حالات زیر است:

۱) $A \rightarrow a$

۲) $A \rightarrow \lambda$

۳) $A \rightarrow aB$

که در آن $A,B \in V$ و $a \in \sum$ می باشد.

یک زبان با قاعده است اگر بتواند توسط یک گرامر با قاعده تولید شود.

مثال:

گرامر با قاعده	گرامر بی قاعده
$S \rightarrow aA \mid \lambda$	$G: \ S \rightarrow abSA \mid \lambda$
$A \rightarrow bS \mid bB$	$A \rightarrow Aa \mid \lambda$
$B \rightarrow aB \mid \lambda$	

$S \Rightarrow (ab)^nSA^n$

$\Rightarrow (ab)^nA^n$

$\Rightarrow (ab)^n(a^*)^n$ $\quad n \geq 0$

$L(G)=(ab)^+a^* \cup \lambda$

گرامرها و زبانهای مستقل از متن

- ✓ هر گرامر مستقل از متن یک زبان مستقل از متن تولید می کند.
- ✓ در عبارت با قاعده توان معنا ندارد.

مثال: دو گرامر G_1 و G_2 را در نظر بگیرید. آیا این دو گرامر معادلند.

G_1: $S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid \lambda$ $\quad L(G_1)=a^+b^*$

G_2: $S \rightarrow aS \mid aB$

$B \rightarrow bB \mid \lambda$ $\quad L(G_2)=a^+b^*$

مثال: فرض کنید که G گرامری با قوانین زیر باشد. زبان گرامر را بدست آورید.

$S \rightarrow abScB \mid \lambda$ $\quad S \Rightarrow (ab)^nS(cB)^n$

$B \rightarrow bB \mid b$ $\quad \Rightarrow (ab)^n(cb^m)^n$

$L(G)=\{(ab)^n(cb^m)^n \mid n \geq 0, m>0\}$

گرامرها و زبانهای مستقل از متن

مثال: نشان دهید دو گرامر G_1 و G_2 معادلند.

G_1: $S \rightarrow AbAbA$ $\quad a^n b a^n b a^n$

$A \rightarrow aA \mid \lambda$ $\quad L(G_1)=a^*ba^*ba^*$ مستقل از متن

G_2: $S \rightarrow aS \mid bA$

$A \rightarrow aA \mid bC$ $\quad a^n b a^m b a^k$

$C \rightarrow aC \mid \lambda$ $\quad L(G_2)=a^*ba^*ba^*$ با قاعده

مثال: نشان دهید دو گرامر G_1 و G_2 معادلند.

G_1: $S \rightarrow AbAbA$ مستقل از متن $\quad$ G_2: $S \rightarrow aS \mid bA$ با قاعده

$A \rightarrow aA \mid bA \mid \lambda$ $\quad$ $A \rightarrow aA \mid bC$

$L(G_1)=(aUb)^*b(aUb)^*b(aUb)^*$ $\quad$ $C \rightarrow aC \mid bC \mid \lambda$

هر دو حداقل دو b دارند. $\quad$ $L(G_2)=a^*ba^*b(aUb)^*$

گرامرها و زبانهای مستقل از متن

مثال: گرامری بنویسید که زبان زیر را تولید کند.

رشته هایی به طول زوج روی $\sum=\{a,b\}$.

$S \rightarrow aaS \mid B$ $\quad$ $S \rightarrow aA \mid bA \mid \lambda$

$B \rightarrow abB \mid C$ $\quad$ $A \rightarrow aS \mid bS$

$C \rightarrow baC \mid D$ $\quad$ با قاعده

$D \rightarrow bbD \mid S \mid \lambda$

مستقل از متن

مثال: گرامری برای تولید زبان زیر طراحی کنید.

$L(G)=\{w \in \{a,b\}^* \mid w=w^R\}$

$S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$

مثال: گرامر مستقل از متنی برای تولید زبان زیر بنویسید.

گرامر	زبان
$S \rightarrow aSdd \mid A$ $A \rightarrow bAc \mid bc$	$\{a^n b^m c^m d^{2n} \mid n \geq 0, m > 0\}$

گرامرها و زبانهای مستقل از متن

مثال:گرامری برای تولید زبان زیر طراحی کنید. رشته ها فاقد زیر رشته abc باشد.

$S \rightarrow bS|cS|aA|\lambda$

$A \rightarrow bB|cS|aA|\lambda$

$B \rightarrow bS|aA|\lambda$

مثال: زبان گرامر زیر را بدست آورید.

زبان	گرامر
$L=\{a^{2n}b^{3n} \mid n>0\}$	$G: S \rightarrow AASB \mid AAB$ $A \rightarrow a$ $B \rightarrow bbb$

اثبات:

قانون به کار رفته	اشتقاق
$S \rightarrow AASB$ $S \rightarrow AAB$ $A \rightarrow a$ $B \rightarrow bbb$	$S \Rightarrow (AA)^n SB^n$ $\Rightarrow (AA)^{n+1}B^{n+1}$ $\Rightarrow (aa)^{n+1}\ B^{n+1}$ $\Rightarrow (aa)^{n+1}\ (bbb)^{n+1} \quad n\geq 0$ $\Rightarrow (aa)^{n}\ (bbb)^{n} \quad n>0$ $= a^{2n}\ b^{3n}$

گرامرها و زبانهای مستقل از متن

مثال: زبان گرامر زیر را بدست آورید.

زبان	گرامر
$L(G)=a^*(a^*ba^*ba^*)^*$	$S \rightarrow aS \mid bB \mid \lambda$ $B \rightarrow aB \mid bS \mid bC$ $C \rightarrow aC \mid \lambda$

زمانی گرامر باقاعده ای معادل با گرامر مستقل از متن می توان نوشت اگر بتوان زبان گرامر مستقل از متن را با استفاده از عبارت باقاعده بیان کرد. زبان یک گرامر با قاعده حتما یک عبارت با قاعده است.

اشتقاق در یک گرامر مستقل از متن مکانیزمی برای تولید رشته های زبان گرامر ایجاد می کند.

زبان یک گرامر: مجموعه ای از رشته های (ترم های) پایانی است که می توانند به هر روشی از عنصر ابتدائی مشتق شوند.

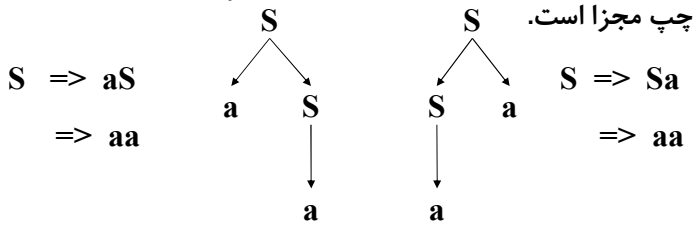
گرامرها و زبانهای مستقل از متن

قضیه: فرض کنید $G=(V,\sum,P,S)$ یک گرامر مستقل از متن باشد. یک رشته w در $L(G)$ است اگر و تنها اگر یک اشتقاق چپ w از S وجود داشته باشد و بالعکس.

$w \in L(G) \Leftrightarrow S \overset{*}{\Rightarrow} w$

نکته: گرامر مستقل از متن G مبهم (ambiguous) است، اگر یک رشته $w \in L(G)$ وجود داشته باشد بطوریکه با بیش از یک اشتقاق چپ مجزا تولید شود.

مثال: گرامر $G: S \rightarrow aS \mid Sa \mid a$ یک گرامر مبهم است زیرا رشته aa دارای دو اشتقاق چپ مجزا است.



گرامر $S \rightarrow aS|a$ گرامر غیرمبهم است.

$L(G)=a^{+}$

گرامرها و زبانهای مستقل از متن

تعریف: اگر برای تولید یک زبان نتوان هیچ گرامر غیرمبهمی نوشت در آنصورت آن زبان مبهم می باشد.

نکته: یک گرامر غیر مبهم است اگر در هر مرحله از یک اشتقاق چپ تنها یک قانون وجود داشته باشد که ما را به اشتقاق یک رشته کامل هدایت کند.

$S \rightarrow aSb \mid A \mid \lambda$
$A \rightarrow aAbb \mid abb$

مثال: گرامر $G: S \rightarrow bS \mid Sb \mid a$

زبان G، b^*ab^* است. اشتقاقهای چپ زیر ابهام G را نشان می دهند.

S => bS
=> bSb
=> bab

S => Sb
=> bSb
=> bab

گرامرها و زبانهای مستقل از متن

گراف یک گرامر

اشتقاقهای چپ یک گرامر مستقل از متن G می تواند به وسیله یک گراف جهت دار g(G) (گراف چپ گرامرG) نشان داده شود. گره های گراف، فرم‌جمله‌ای های چپ گرامر هستند.

یک فرم جمله ای چپ، رشته ای است که می تواند بوسیله یک اشتقاق چپ از عنصر ابتدایی نتیجه شود.

از آنجا که دو اشتقاق چپ مختلف دارای درختهای اشتقاق متمایز می‌باشند بنابراین گرامری مبهم است که برای رشته$w\epsilon L(G)$بتوان دو درخت اشتقاق مختلف رسم کرد.

گراف گرامر می تواند چپ یا راست یا کامل باشد. در گراف کامل گرامر تمامی فرم جمله ای های حاصل از یک جمله را (اشتقاقهای راست یا چپ یا...) باید در نظر گرفت.

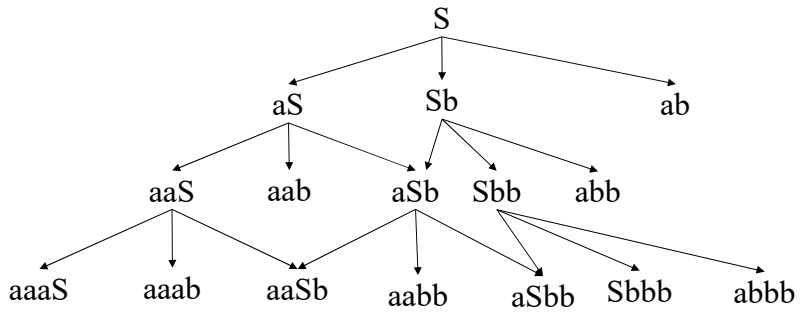
اگر گراف گرامر درخت باشد(درجه ورودی بیش از یک نباشد) گرامر مبهم نیست.

گرامر مستقل از متن دارای تعداد محدودی قانون می باشد، لذا هر گره نیز دارای تعداد محدودی فرزند خواهد بود، به این گراف، گراف متناهی محلی گوییم.

گرامرها و زبانهای مستقل از متن

مثال:گراف چپ گرامر زیر را تا چهار سطح رسم کنید.

$S \rightarrow aS|Sb|ab$



درخت نیست بنابراین مبهم است.

پارسر

در بحث parsing فقط گراف چپ گرامر مدنظر خواهد بود.

تجزیه(parsing): یک سوال مهم اینست که چگونه می توان تعیین کرد که آیا رشته ای متعلق به زبان گرامر می باشد یا خیر. آیا یک رشته ورودی از قوانین گرامر قابل اشتقاق است یا خیر.

الگوریتمهای مورد استفاده برای پاسخ به این سوال را پارسر (تجزیه کننده یا تحلیل گر) گویند.

انواع الگوریتم های parsing:

۱- پارسر بالا به پائین(top-down): جستجو از گره S شروع شده و تا یافتن رشته موردنظر ادامه می یابد. جستجو در پهنا(سطح)/ جستجو در عمق

۲- پارسر پائین به بالا(Buttom-up): شروع جستجو با رشته پایانی موردنظر و ادامه آن تا رسیدن به S است. جستجو در پهنا(سطح)/ جستجو در عمق

جستجو در پهنا با صف پیاده سازی می شود و جستجو در عمق با پشته پیاده سازی می شود.

پارسر

terminal prefix: اشتقاق چپ uAv=>uwv را در نظر بگیرید که با استفاده از قانون $A \rightarrow w$ انجام گرفته است. اگر رشته u فقط از حروف الفبا ($\sum$) تشکیل شده باشد (یا λ باشد)، به آن terminal prefix می‌گویند. پیش رشته ای که فقط دارای ترم پایانی باشد.

هدف استفاده از terminal prefix محدود کردن رشد درخت جستجو است.

در درخت جستجو هرجا شاخه‌ای تولید شود که پیشوند آن با terminal prefix مورد نظر یکسان نباشد، رشد آن متوقف می‌شود.

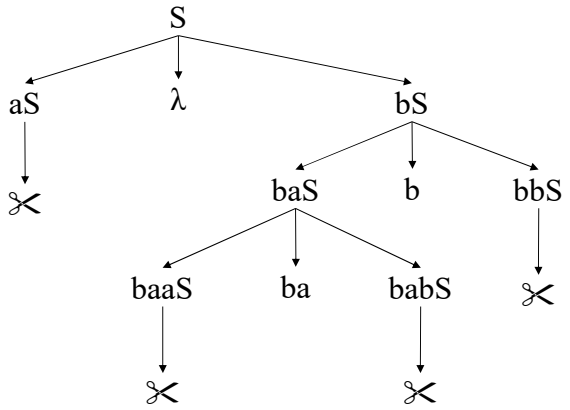
dead-end string: رشته بن بست رشته ای است که پارسر می تواند تشخیص دهد در اشتقاق، رشته مورد نظر نمی تواند تولید شود. پارسر با استفاده از terminal prefix رشته بن بست را تشخیص می دهد.

پارسر

مثال: گرامر **G** را در نظر بگیرید:

1)S→aS
2)S→bS
3)S→λ

با استفاده از گراف اشتقاق تعیین کنید که آیا جمله ba به گرامر تعلق دارد یا خیر.



پارسر

پارسر بالا به پایین سطحی (breadth-first top-down)

پارسر بالا به پایین: اشتقاقهایی را با استفاده از قوانین روی متغیرهای چپ یک فرم جمله ای ایجاد می کند.

مسیرهایی که با S در گراف یک گرامر شروع می شوند بیانگر اشتقاق چپ گرامر هستند.

این پارسر درخت جستجو را در سطح تولید می‌کند. برای تولید درخت جستجو از صف استفاده می‌شود.

شرایط پایانی الگوریتم:

۱) رسیدن به رشته مورد نظر(جواب پارسر مثبت)

۲) خالی شدن محتوای صف (جواب پارسر منفی)

گرامرهایی که طول فرم‌جمله‌ای‌های ایجاد شده طی مراحل اشتقاق آن از مرحله‌ای به مرحله بعد کاهش پیدا نکند گرامر غیرکاهشی (noncontracting) گویند.

پارسر

مثال: پارسر بالا به پایین سطحی (breadth-first top-down)

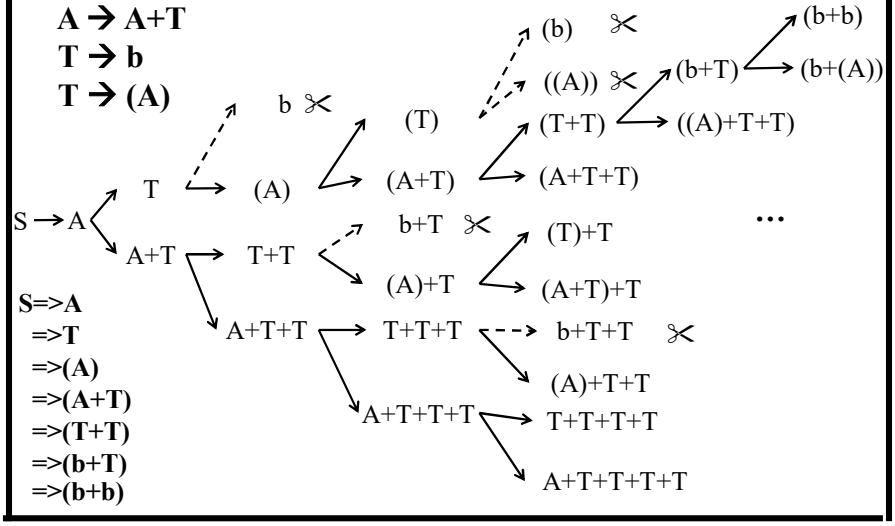
P: S → A ∑={b,+,(,)} V={S,A,T} w=(b+b)

A → T

A → A+T

T → b

T → (A)



S=>A
=>T
=>(A)
=>(A+T)
=>(T+T)
=>(b+T)
=>(b+b)

پارسر

پارسر بالا به پایین سطحی (breadth-first top-down)

ویژگیهای این الگوریتم:

۱- این امکان وجود دارد که رشته متعلق به زبان گرامر نبوده ولی الگوریتم در حلقه نامتناهی قرار گیرد. (به دلیل وجود قوانین بازگشت به چپ)

۲- اشتقاق تولید شده توسط این الگوریتم اشتقاق چپ است. (تمام الگوریتم های بالا به پایین این ویژگی را دارند)

۳- حجم درخت جستجوی تولید شده زیاد است به خصوص اگر قوانین زیاد باشد.

برای رفع مشکل قرارگیری در حلقه نامتناهی می توان به صورتهای زیر عمل کرد:

۱- قبل از انجام الگوریتم جمع آوری اطلاعاتی در مورد رشته مورد بحث از قبیل تعداد + و ... و سپس شرایط پایانی الگوریتم را براساس این اطلاعات تغییر داد.

۲- استفاده از پارسرهای چند گذره (multipass): هر گذر یکبار خواندن کل برنامه است. یعنی رشته ورودی چندبار از ابتدا تا انتها خوانده شود. به عنوان مثال در گذر اول طول رشته را بدست آورید و در گذر دوم فرم جمله ایهایی را که طول آنها از طول رشته بیشتر است داخل صف قرار ندهید.

پارسر

پارسر بالا به پایین عمقی (depth-first top-down)

در این الگوریتم عمل بازگشت backtracking وجود دارد بنابراین نیاز به استفاده از پشته است.

شرایط پایانی الگوریتم:

۱) رسیدن به رشته مورد نظر(جواب پارسر مثبت)

۲) خالی شدن محتوای پشته(جواب پارسر منفی)

ویژگیهای این الگوریتم:

۱- این امکان وجود دارد که رشته متعلق به زبان گرامر باشد ولی الگوریتم در حلقه نامتناهی قرار گیرد.

۲- اشتقاق تولید شده توسط این الگوریتم اشتقاق چپ است.

۳- حجم درخت جستجوی تولید شده کم است.

۴- زمان پاسخگویی این الگوریتم نسبت به الگوریتم قبل بسته به مکان رشته در گراف گرامر می تواند کمتر یا بیشتر باشد.

۵- این الگوریتم پیاده سازی و منطق ساده ای دارد.

پارسر

پارسر بالا به پایین عمقی (depth-first top-down)

هر خانه پشته ساختاری به شکل [u,i] دارد که درآن u پشته ای است که با قانون شماره i بسط داده خواهد شد. هر backtrack معادل pop کردن است.

P: 1)S → A ∑={b,+,(,)} V={S,A,T} w=(b+b)
2)A → T
3)A → A+T
4)T → b
5)T → (A)

[S,Ø]	[S,1]			
	[A,2]			
	~~[T,4]~~	[T,5]		
S=>A	b	~~[(A),2]~~		[(A),3]
=>T		~~[(T),4]~~	~~[(T),5]~~	[(A+T),2]
=>(A)		(b)	((A))	[(T+T),4]
=>(A+T)				[(b+T),4]
=>(T+T)				(b+b)
=>(b+T)				
=>(b+b)				

پارسر

پارسر پایین به بالا سطحی (breadth-first bottom-up)

کاهش (reduction): فرض کنید اشتقاق راست وجود دارد. جایگزینی رشته w با متغیر A را کاهش می گوییم (عکس عمل اشتقاق).

ویژگیهای این الگوریتم:

۱- این امکان وجود دارد که رشته متعلق به زبان گرامر نبوده ولی الگوریتم در حلقه نامتناهی قرار گیرد. (به دلیل وجود قوانین بازگشت به چپ)

۲- مسیر موجود از رشته به سرترم S بیانگر مراحل کاهش و عکس آن بیانگر مراحل اشتقاق است.

۳- همواره اشتقاق راست تولید می شود.

شرایط پایانی الگوریتم:

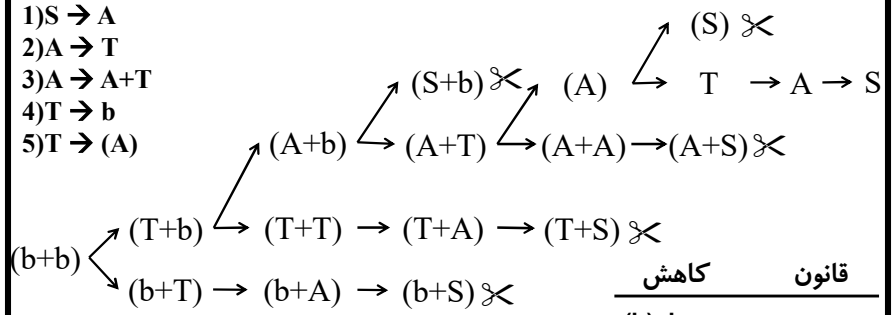
۱) رسیدن به سرترم گرامر(جواب پارسر مثبت)

۲) خالی شدن محتوای صف (جواب پارسر منفی)

پارسر

پارسر پایین به بالا سطحی (breadth-first bottom-up)

1)S → A
2)A → T
3)A → A+T
4)T → b
5)T → (A)



کاهش	قانون
(b)+b	
(T)+b	T → b
(A)+b	A → T
T+b	T → (A)
A+b	A → T
A+T	T → b
A	A → A+T
S	S → A

S=>A
=>T
=>(A)
=>(A+T)
=>(A+b)
=>(T+b)
=>(b+b)

S=>A
=>A+T
=>A+b
=>T+b
=>(A)+b
=>(T)+b
=>(b)+b

پارسر

پارسر پایین به بالا عمقی (depth-first bottom-up)

شرایط پایانی الگوریتم:

۱) رسیدن به سرترم گرامر(جواب پارسر مثبت)

۲) خالی شدن محتوای پشته(جواب پارسر منفی)

ویژگیهای این الگوریتم:

۱- این الگوریتم روی گرامرهایی که سرترم گرامر بازگشتی نباشد عمل می کند.

۲- همواره اشتقاق راست تولید می کند.

۳- محتوای پشته مراحل اشتقاق را نشان می دهد.

۴- به دلیل کارایی و سرعت بالای این الگوریتم اکثر کامپایلرهای زبانهای برنامه نویسی با این الگوریتم پیاده سازی می شوند.

پارسر

پارسر پایین به بالا عمقی (depth-first bottom-up)

در این الگوریتم اولویت انجام عملیات به شرح زیر است:

۱- کاهش (reduction)

۲- انتقال (shift)

۳- برداشتن عضوی از پشته (pop)

- **هر عمل reduction معادل با قرار دادن عنصری در پشته (push) می باشد. کاهش به سرترم زمانی انجام گیرد که $v=\lambda$ و u به s بتواند کاهش داده شود.**
- **عمل shift به معنای انتقال یک الفبا از سمت چپ رشته v و الحاق آن در سمت راست رشته u است.**
- **عناصر پشته در این الگوریتم به صورت [u,i,v] می باشد که در آن w=uv رشته ای است که با قانون i کاهش داده شده است.**

پارسر

پارسر پایین به بالا عمقی (depth-first bottom-up)

عمل	v	i	u	پشته
pop	(b+b)	Ø	λ	[λ,0,(b+b)]
shift	b+b)	Ø	(	
shift	+b)	4	(b	
push	+b)	2	(T	[(b,4,+b)]
push	+b)	Ø	(A	[(T,2,+b)] [(b,4,+b)]
shift	b)	Ø	(A+	[(T,2,+b)] [(b,4,+b)]
shift	)	4	(A+b	[(T,2,+b)] [(b,4,+b)]
push	)	2	(A+T	[(A+b,4,)] [(T,2,+b)] [(b,4,+b)]
push	)	Ø	(A+A	[(A+T,2,)] [(A+b,4,)] [(T,2,+b)] [(b,4,+b)]

پارسر

پارسر پایین به بالا عمقی (depth-first bottom-up)

عمل	v	i	u	پشته
shift	λ	Ø	(A+A)	[(A+T,2,)] [(A+b,4,)] [(T,2,+b)] [(b,4,+b)]
pop***	)	3	(A+T	[(A+b,4,)] [(T,2,+b)] [(b,4,+b)]
push	)	Ø	(A	[(A+T,3,)] [(A+b,4,)] [(T,2,+b)] [(b,4,+b)]
shift	λ	5	(A)	[(A+T,3,)] [(A+b,4,)] [(T,2,+b)] [(b,4,+b)]

پارسر

پارسر پایین به بالا عمقی (depth-first bottom-up)

عمل	v	i	u	پشته
push	λ	2	T	[(A),5, λ] [(A+T,3,)] [(A+b,4,)] [(T,2,+b)] [(b,4,+b)]
push	λ	1	A	[T,2, λ] [(A),5, λ] [(A+T,3,)] [(A+b,4,)] [(T,2,+b)] [(b,4,+b)]
push	λ	Ø	S	[A,1, λ] [T,2, λ] [(A),5, λ] [(A+T,3,)] [(A+b,4,)] [(T,2,+b)] [(b,4,+b)]

فرمهای نرمال Normal forms

یک فرم نرمال با اعمال محدودیتهایی روی شکل قوانین یک گرامر تعریف می شود. گرامرها در یک فرم نرمال، مجموعه کاملی از زبانهای مستقل از متن را تولید می کنند.

برای گرامرهای مستقل از متن دو فرم نرمال مهم وجود دارد:

۱- فرم های نرمال شومسکی (Chomsky)

۲- فرم های نرمال گریباش (Gribach)

هر گرامر مستقل از متن را می توان به این دو نوع فرم نرمال تبدیل کرد و برای انجام این کار الگوریتم های مشخصی وجود دارد که از یک سری تبدیلات پایه ای استفاده می کند.

- ☜ حذف خاصیت بازگشتی سرترم
- ☜ حذف قوانین لامبدا
- ☜ حذف قوانین زنجیره ای (chain rules)
- ☜ حذف نمادهای زائد (useless symbols)

فرمهای نرمال Normal forms

حذف خاصیت بازگشتی سرترم

قضیه: فرض کنید که $G=(V,\sum,P,S)$یک گرامر مستقل از متن باشد. یک گرامر $G'=(V',\sum,P',S')$وجود دارد به گونه ای که در شرایط زیر صدق کند.

الف) $L(G)=L(G')$

ب) قوانین P' به شکل $A\rightarrow w$ می باشند که در آن $w\epsilon((V'-\{S'\})U\sum)^*$ و $A\epsilon V'$ است.

کافیست قانون جدیدی به صورت $S\rightarrow S'$ به گرامر اضافه شود. در اینجا S' سرترم گرامر است.

مثال:

$G:S' \rightarrow S$

$G:S \rightarrow aS \mid AB \mid AC$

$A \rightarrow aA \mid \lambda$

$B \rightarrow bB \mid bS$

$C \rightarrow cC \mid \lambda$

$\Rightarrow$

$S \rightarrow aS \mid AB \mid AC$

$A \rightarrow aA \mid \lambda$

$B \rightarrow bB \mid bS$

$C \rightarrow cC \mid \lambda$

فرمهای نرمال Normal forms

حذف قوانین لامبدا

قضیه: فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن باشد. الگوریتمی وجود دارد که می تواند مجموعه متغیرهای **nullable** گرامرG را تولید می کند.

به گرامرهای بدون لامبدا گرامر **noncontracting** گویند.

قضیه: متغیر A، **nullable** است اگر اشتقاق زیر وجود داشته باشد. $A\overset{*}{\Rightarrow}\lambda$

ورودی: یک گرامر مستقل از متن $G=(V,\sum,P,S)$

```
NULL : = {A l A→ λ ϵP}
repeat
        PREV := NULL
        for each variable A ϵ V do
          If there is an A rule A → w and  w ϵ PREV* then
                NULL : = NULL U {A}
until NULL = PREV
```

قضیه: فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن باشد. اگر $A\overset{*}{\underset{G}{\Rightarrow}}w$ باشد، آنگاه گرامر $G'=(V,\sum,P\ U\{A\rightarrow w\},S)$ معادل با گرامر G است.

فرمهای نرمال Normal forms

حذف قوانین لامبدا

قضیه: فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن باشد. الگوریتمی برای ایجاد یک گرامر مستقل از متن $G_L=(V_L,\sum,P_L,S_L)$ وجود دارد به گونه ای که در شرایط زیر صدق کند.

۱) $L(G_L)=L(G)$

۲) $\lambda \epsilon L(G) \Rightarrow S_L \rightarrow \lambda \epsilon P_L$

۳) با استفاده از قضیه قبل می توان قوانین جدیدی را به گرامر G اضافه نمود به گونه ای که اثر null بودن متغیرها را در برگیرد.

- ✓ شرط اجرای این الگوریتم عدم وجود خاصیت بازگشتی برای سرترم گرامر است.
- ✓ اکنون با درنظر گرفتن محتوای مجموعه null اثر null بودن آنها را مستقیما در قوانین اعمال می کنیم.
- ✓ گرامری که فاقد قوانین لامبدا است G_L نامیده می شود.
- ✓ اگر سرترم در مجموعه Null وجود داشته باشد ناگزیریم $S\rightarrow\lambda$ را در G_L بنویسیم.

فرمهای نرمال Normal forms

حذف قوانین لامبدا

مثال:

$G:S \rightarrow ACA$
$A \rightarrow aAa \mid B \mid C$
$B \rightarrow bB \mid b$
$C \rightarrow cC \mid \lambda$

$G_L:S \rightarrow ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \lambda$
$A \rightarrow aAa \mid aa \mid B \mid C$
$B \rightarrow bB \mid b$
$C \rightarrow cC \mid c$

NULL={C,A,S}

مثال: w=aba با هر دو گرامر

G:S => ACA
=> aAaCA
=> aBaCA
=> abaCA
=> abaA
=> abaC
=> aba

G_L:S => A
=> aAa
=> aBa
=> aba

فرمهای نرمال Normal forms

حذف قوانین زنجیره

بکارگیری یک قانون $A \rightarrow B$ $(A,B \in V)$ نه تنها طول رشته مشتق شده را افزایش نمی دهد، بلکه عناصر پایانی اضافی نیز تولید می کند. در واقع، این قانون یک متغیر را مجددا نامگذاری می کند. قوانینی به این شکل به قوانین زنجیره ای موسومند.

قوانین زیر را در نظر بگیرید:

$A \rightarrow aA \mid a \mid B$

$B \rightarrow bB \mid b \mid C$

قانون زنجیره ای $A \rightarrow B$ مشخص می کند که هر رشته قابل اشتقاق از B، از A نیز قابل اشتقاق است. بکارگیری یک قانون زنجیره ای، یک مرحله اضافی است که می تواند با افزودن قوانین B حذف شود.

$A \rightarrow aA \mid a \mid bB \mid b \mid C$

$B \rightarrow bB \mid b \mid C$

تعریف: اشتقاقی که فقط در آن از قوانین زنجیره استفاده شده باشد زنجیر نامیده می شود. $A \overset{*}{\Rightarrow} C$

فرمهای نرمال Normal forms

حذف قوانین زنجیره

قضیه: فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن غیر کاهشی باشد. الگوریتمی وجود دارد که مجموعه متغیرهای قابل اشتقاق از متغیر A را (که فقط با استفاده از قوانین زنجیره ای تولید می شود) بدست آوریم.

ورودی: یک گرامر مستقل از متن غیر کاهشی $G=(V,\sum,P,S)$

```
CHAIN(A) : = {A}
PREV : = ø
repeat
    NEW : = CHAIN(A)-PREV
    PREV : = CHAIN(A)
    for each variable B ϵ NEW do
        for each rule B→C do
            CHAIN(A) : = CHAIN(A) U {C}
until CHAIN(A) = PREV
```

قضیه: فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن غیرکاهشی باشد. الگوریتمی برای ایجاد یک گرامر مستقل از متن G_C وجود دارد به گونه ای که:

۱) $L(G_C)=L(G)$ ۲) G_C فاقد قوانین زنجیره است.

فرمهای نرمال Normal forms

حذف قوانین زنجیره

G_L: S → ACA | CA | AA| AC | A | C | λ

A → aAa | aa | B | C

B → bB | b

C → cC | c

CHAIN(S)={S,A,C,B}
CHAIN(A)={A,B,C}
CHAIN(B)={B}
CHAIN(C)={C}

G_C: S → ACA | CA | AA| AC | aAa | aa | bB | b | cC | c | λ

A → aAa | aa | bB | b | cC | c

B → bB | b

C → cC | c

نکته: هر یک از قوانین گرامر G_C به یکی از سه صورت زیر می باشد.

S→ λ ($\lambda \in L(G_C)$)

A→ a ($A \in V, a \in \sum^*$)

A→ w ($A \in V, w \in (V \cup \sum^*)$, Length(w)≥2)

فرمهای نرمال Normal forms

حذف نمادهای زائد

فرض کنید G یک گرامر مستقل از متن است. در این صورت $x \in (V \cup \sum)^*$ مفید (useful) است اگر یک اشتقاق $S \overset{*}{\underset{G}{\Rightarrow}} uxv \overset{*}{\underset{G}{\Rightarrow}} w$ وجود داشته باشد که در آن $u,v \in (V \cup \sum)^*$ و $w \in \sum^*$ باشد. در غیراینصورت نماد x زائد است.

یک متغیر مفید است اگر:

۱- بتواند در نهایت به یک رشته(ترم) پایانی برسد.

۲- از سرترم قابل دسترسی باشد.

قضیه: فرض کنید $G=(V,\sum,P,S)$ یک گرامر مستقل از متن است. الگوریتمی برای ایجاد یک گرامر مستقل از متن $G_T=(V_T,\sum_T,P_T,S)$ وجود دارد به گونه ای که:

۱) $L(G_T)=L(G)$

۲) هر متغیر در G_T یک رشته پایانی در $L(G_T)$ را تولید می کند.

حذف نمادهای زائد

الگوریتم تولید مجموعه متغیرهایی که قادر به تولید ترم پایانی هستند.

ورودی: یک گرامر مستقل از متن $G=(V,\sum,P,S)$

```
TERM : = {A l there is a rule A→ w ϵP, w ϵ ∑*}
 repeat
        PREV := TERM
        for each variable A ϵ V do
          If there is an A rule A → w and  w ϵ (PREV U ∑ )* then
                TERM: = TERM U {A}
until PREV = TERM
```


حذف نمادهای زائد

الگوریتم زیر متغیرهایی را تعیین می کند که از سرترم گرامر قابل دسترسی هستند.

ورودی: یک گرامر مستقل از متن $G=(V,\sum,P,S)$

```
REACH := {S}
PREV : = ø
repeat
     NEW := REACH – PREV
     PREV := REACH
     for each variable A ϵ NEW do
          for each rule A → W do
               add all variables in w to REACH
until REACH = PREV
```

قضیه: فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن باشد. الگوریتمی برای ایجاد یک گرامر مستقل از متن G_U وجود دارد به گونه ای که:

۱) $L(G_U)=L(G)$ ۲) G_U فاقد نمادهای زائد است.

فرمهای نرمال Normal forms

حذف نمادهای زائد

ترتیب به کارگیری دو الگوریتم فوق بسیار مهم است. ۱) TERM ۲) Reach

مثال:

گرامر زیر را در نظر بگیرید. $G : S \rightarrow a \mid AB$

$A \rightarrow b$

روش اول: ۱) حذف عناصری که از سرترم قابل دسترسی نیستند.
۲) حذف متغیرهایی که رشته (ترم) پایانی تولید نمی کنند. روش غلط

$Reach=\{S,A,B\}$ $TERM=\{S,A\}$

$G_U : S \rightarrow a \mid AB$ $A \rightarrow b$ $G_T : S \rightarrow a$ $A \rightarrow b$

روش دوم: ۱) حذف متغیرهایی که رشته (ترم) پایانی تولید نمی کنند.
۲) حذف عناصری که از سرترم قابل دسترسی نیستند.

$TERM=\{S,A\}$ $G_T : S \rightarrow a$ $A \rightarrow b$ $Reach=\{S\}$ $G_U : S \rightarrow a$

فرمهای نرمال Normal forms

حذف نمادهای زائد

مثال:

$G: S \rightarrow AC \mid BS \mid B$
$A \rightarrow aA \mid aF$
$B \rightarrow CF \mid b$
$C \rightarrow cC \mid D$
$D \rightarrow aD \mid BD \mid C$
$E \rightarrow aA \mid BSA$
$F \rightarrow bB \mid b$

1 ⇒

$V_T = \{S,A,B,E,F\}$
$\sum_T = \{a,b\}$
$G_T : S \rightarrow BS \mid B$
$A \rightarrow aA \mid aF$
$B \rightarrow b$
$E \rightarrow aA \mid BSA$
$F \rightarrow bB \mid b$ $TERM=\{B,F,S,A,E\}$

$G_U : S \rightarrow BS \mid B$
$B \rightarrow b$ $REACH=\{S,B\}$

⇐ 2

فرمهای نرمال Normal forms

فرم نرمال شومسکی

یک گرامر مستقل از متن $G=(V,\sum,P,S)$ در فرم نرمال شومسکی است اگر هر قانون دارای یکی از حالتهای زیر باشد:

$A \rightarrow BC$	$B,C \in V-\{S\}$
$A \rightarrow a$	$A \in V$
$A \rightarrow \lambda$	$a \in \sum$

الگوریتم تبدیل:

۱) حذف خاصیت بازگشتی سرترم گرامر

۲) حذف قوانین لامبدا

۳) حذف قوانین زنجیره

۴) حذف نمادهای زائد

۵) تبدیل شکل قوانین به فرم نرمال شومسکی

فرمهای نرمال Normal forms

مثال: گرامر زیر را به فرم نرمال شومسکی تبدیل کنید.

G: $S \rightarrow aABC \mid a$
$A \rightarrow aA \mid a$
$B \rightarrow bcB \mid bc$
$C \rightarrow cC \mid c$

چهار شرط اول برقرار است.

G' : $S \rightarrow A'T_1 \mid a$
$T_1 \rightarrow AT_2$
$T_2 \rightarrow BC$
$A \rightarrow A'A \mid a$
$B \rightarrow B'T_3 \mid B'C'$
$T_3 \rightarrow C'B$
$C \rightarrow C'C \mid c$
$A' \rightarrow a$
$B' \rightarrow b$
$C' \rightarrow c$

فرمهای نرمال Normal forms

فرم نرمال شومسکی

مزیت: درخت اشتقاق در یک گرامر به فرم شومسکی یک درخت دودویی است. تمامی عملیاتهای روی درختهای دودویی در عملیات parsing قابل استفاده است.

مثال: گرامر G به فرم نرمال شومسکی را درنظر بگیرید و فرض کنید $w \in L(G)$ و طول رشته w، n باشد.

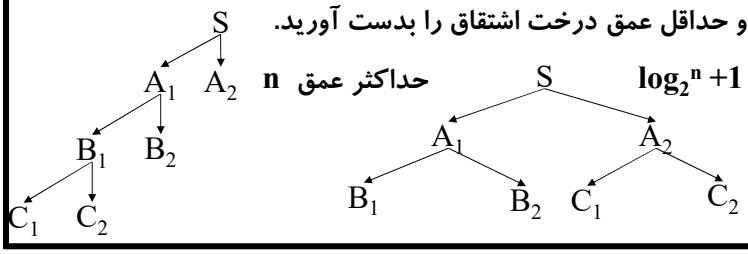
الف) تعداد مراحل اشتقاق را بدست آورید. $n-1+n=2n-1$

رسیدن از یک متغیر به n متغیر، n-1 بار استفاده از قانون $S \rightarrow AB$: n-1

n بار استفاده از قانون $A \rightarrow a$: n

ب) حداکثر و حداقل عمق درخت اشتقاق را بدست آورید.

حداقل عمق $\log_2^n + 1$ **حداکثر عمق n**



فرمهای نرمال Normal forms

حذف بازگشتی چپ مستقیم

قانونی به صورت $A \rightarrow A\alpha$ که در آن $\alpha \in (\Sigma UV)^+$ باشد را قانونی با بازگشت مستقیم چپ می‌نامیم. از آنجا که تمامی الگوریتمهای parsing به دلیل وجود بازگشتی چپ در حلقه نامتناهی قرار می گیرند، بررسی این مساله ضروری است.

مثال:

$A \rightarrow Aa \mid b \quad \Rightarrow \quad A \rightarrow bZ \mid b$

$Z \rightarrow aZ \mid a$ $\quad ba^*$

مثال:

$A \rightarrow Aa \mid Ab \mid b \mid c \quad \Rightarrow \quad A \rightarrow bZ \mid cZ \mid b \mid c$

$Z \rightarrow aZ \mid bZ \mid a \mid b$ $\quad (b \cup c)(a \cup b)^*$

مثال:

$A \rightarrow AB \mid BA \mid a \quad \Rightarrow \quad A \rightarrow BAZ \mid aZ \mid BA \mid a$

$B \rightarrow b \mid c \qquad\qquad Z \rightarrow BZ \mid B$

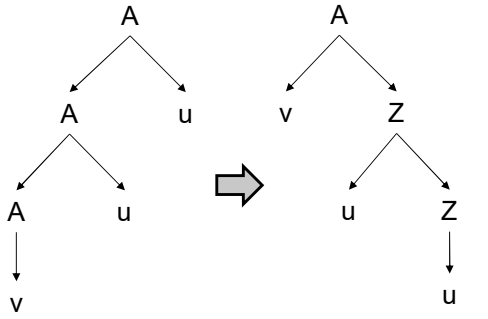
$B \rightarrow b \mid c$ $\quad (b \cup c)^*a(b \cup c)^*$

فرمهای نرمال Normal forms

قانون کلی حذف بازگشتی چپ مستقیم

قضیه :فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن بوده و $A \epsilon V$ یک متغیر بازگشتی چپ مستقیم در G باشد. الگوریتمی وجود دارد که یک گرامر مستقل از متن معادل $G'=(V',\sum,P',S')$ ایجاد نماید بطوریکه A خاصیت بازگشتی چپ مستقیم نداشته باشد.

برای حذف بازگشت مستقیم چپ، به ازای هر قانونی که بازگشت مستقیم چپ دارد به صورت زیر عمل می شود:



$A \rightarrow Au_1|Au_2|\ldots|Au_n$
$A \rightarrow v_1|v_2|\ldots|v_3$

$A \rightarrow v_1|v_2|\ldots|v_3$
$A \rightarrow v_1Z|v_2Z|\ldots|v_3Z$
$Z \rightarrow u_1Z|u_2Z|\ldots|u_nZ$
$Z \rightarrow u_1|u_2|\ldots|u_n$

فرمهای نرمال Normal forms

متغیرهایی که طی چند مرحله اشتقاق یکدیگر را در سمت چپ قانون مربوط به خود تولید می‌کنند، متغیرهایی با بازگشت چپ غیر مستقیم هستند. قوانین این متغیرها به فرم زیر است: $A \rightarrow B\alpha_1, B \rightarrow C\alpha_1, C \rightarrow A\alpha_2$ و $\alpha_i \epsilon (\Sigma UV)^+$

روش اول: برای شناسایی متغیرهایی که بازگشت غیر مستقیم دارند، تمام متغیرها را بر طبق قانون زیر شماره‌گذاری می‌کنیم. اگر قانونی به صورت $A \rightarrow B\alpha$ در گرامر وجود دارد باید شماره متغیر A کمتر از B باشد.

- ✓ **اگر در شماره‌گذاری بر طبق قانون فوق، تناقضی به صورت $n_A<n_A$ به وجود آید، گرامر بازگشت مستقیم دارد.**
- ✓ **اگر در شماره‌گذاری، تناقضی به صورت $n_A<n_B<\ldots<n_A$ به وجود آید، گرامر بازگشت غیر مستقیم دارد.**

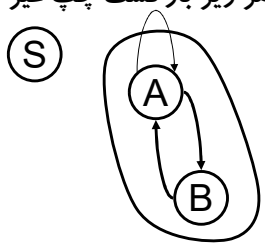
روش دوم: برای شناسایی متغیرهایی که بازگشت غیر مستقیم دارند، گرافی به صورت زیر رسم می‌کنیم. به ازای هر متغیر، یک گره به گراف اضافه می‌شود. اگر قانونی به صورت $A \rightarrow B\alpha$ در گرامر وجود دارد، یالی از گره A به گره B اضافه شود.

- ✓ **اگر گراف حلقه‌ای به طول صفر داشته باشد، گرامر بازگشت مستقیم دارد.**
- ✓ **اگر گراف حلقه‌ای با طول بیشتر داشته باشد، گرامر بازگشت غیر مستقیم دارد.**

فرمهای نرمال Normal forms

مثال: در گرامر زیر بازگشت چپ غیر مستقیم را شناسایی کنید.

S→ aA |cB
A→ Aa |Ba |c
B→ Ab |b



B<A

برای حذف بازگشت غیر مستقیم، ابتدا باید آن را به بازگشت مستقیم تبدیل کرد و سپس با استفاده از روشی که قبلا توضیح داده شده، آن را حذف نمود.

برای تبدیل بازگشت غیر مستقیم به بازگشت مستقیم باید در قوانینی که در حلقه بازگشت قرار دارند، متغیرهای بازگشتی را جایگزین کرد.

در گرامر زیر بازگشت چپ را حذف کنید.

S→ aA |cB	S→ aA |cB	S→ aA |cB
A→ aA |Ba |c	A→ aA |c|Aba |ba	A→ aAZ |cZ |baZ |aA |c |ba
B→ Ab |b	B→ Ab |b	B→ Ab |b
		Z→ baZ |ba

فرمهای نرمال Normal forms

فرم نرمال گریباش

یک گرامر مستقل از متن $G=(V,\sum,P,S)$ در فرم نرمال گریباش است اگر قوانین آن به یکی از صورتهای زیر باشد.

$A \rightarrow aA_1A_2\ldots A_n$	$A_i \in V-\{S\}\ \ 1\leq i \leq n$
$A \rightarrow a$	$A \in V,\ a \in \sum$
$S \rightarrow \lambda$	$\lambda \in L(G)$

الگوریتم تبدیل:

۱) تبدیل به فرم نرمال شومسکی

۲) حذف بازگشتی چپ مستقیم و غیرمستقیم

۳) تبدیل به فرم نرمال گریباش

حذف بازگشت چپ می‌تواند سبب ایجاد زنجیر یا متغیر غیر مفید گردد. به این دلیل دو مرحله زیر را مجددا انجام می‌دهیم.

حذف زنجیره/حذف متغیرهای غیر مفید

فرمهای نرمال Normal forms

فرم نرمال گریباش

پس از تبدیل به فرم نرمال شومسکی و حذف بازگشتی، در این مرحله باید قوانین را بر طبق فرم نرمال گریباش به نحوی تغییر دهیم که سمت راست همه آنها با حرف کوچک شروع شود.

متغیرها را به ترتیبی که قبلا توضیح داده شد، شماره‌گذاری می‌کنیم. آخرین متغیر را در تمام قوانینی که این متغیر در سمت راست آنها ظاهر شده است، جایگزین می‌کنیم، و این کار تا جایگزین شدن همه متغیرها ادامه می‌یابد. این کار معادل حذف تمام یالها در گراف توضیح داده شده است.

به ازای هر حرف الفبا که در سمت راست قوانین (به جز حرف اول آنها) ظاهر شده است، یک متغیر جدید در نظر می‌گیریم و حرف الفبا را با آن جایگزین می‌کنیم. متغیرهای جدید شماره کوچکتری نسبت به متغیرهای قبلی دارند.

اگر در یک گرامر هم بازگشتی چپ و هم راست داشتیم، آن گرامر مبهم است.

مزیت فرم نرمال گریباش: تعداد مراحل اشتقاق برای هر جمله به طول n دقیقاً n+1 است.

فرمهای نرمال Normal forms

مثال: گرامر زیر را به فرم نرمال گریباش تبدیل کنید.

G: S → SaB | aB
B → bB | λ

حذف بازگشتی سرترم
S' →S
S → SaB | aB
B → bB | λ

حذف قوانین لامبدا NULL={B}
S' →S
S → SaB | aB | Sa | a
B → bB | b

حذف قوانین زنجیره CHAIN(S')={S,S'}, CHAIN(S)={S}, CHAIN(B)={B}
S'→ SaB | aB | Sa | a
S → SaB | aB | Sa | a
B → bB | b

حذف نمادهای زائد TERM={S',S,B}, REACH={S',S,B}
ندارد

تبدیل به فرم نرمال شومسکی
S'→ ST | SA | AB | a
S → ST | SA | AB | a
B → B'B | b
T → AB
A → a
B'→ b

فرمهای نرمال Normal forms

مثال: گرامر زیر را به فرم نرمال گریباش تبدیل کنید.

تبدیل به فرم نرمال گریباش

S' →aBZT l aZT l aBT l aT l aBZA
l aZA l aBA l aA l aB l a
S →aBZ l aZ l aB l a
B →bB l b
T → aB
A → a
B' → b
Z → aBZ l aZ l aB l a

فرم نرمال شومسکی

S'→ ST | SA | AB | a
S → ST | SA | AB | a
B → B'B | b
T → AB
A → a
B'→ b

حذف بازگشتی چپ

S'→ ST | SA | AB | a
S → ABZ | aZ | AB | a
B → B'B | b
T → AB
A → a
B'→ b
Z →TZ | AZ | T | A

فرمهای نرمال Normal forms

مثال: رشته w=abaaba را توسط گرامرهای فوق اشتقاق می دهیم.

فرم نرمال گریباش	فرم نرمال شومسکی	G
S'⇒aBZA	S'⇒SA	S⇒SaB
⇒abZA	⇒STA	⇒SaBaB
⇒abaZA	⇒SATA	⇒SaBaBaB
⇒abaaBA	⇒ABATA	⇒aBaBaBaB
⇒abaabA	⇒aBATA	⇒abBaBaBaB
⇒abaaba	⇒abATA	⇒abaBaBaB
	⇒abaTA	⇒abaaBaB
	⇒abaABA	⇒abaabBaB
	⇒abaaBA	⇒abaabaB
	⇒abaabA	⇒abaaba
	⇒abaaba	

فرمهای نرمال Normal forms

الگویتم CYK

الگوریتم تجزیه و عضویت یک رشته برای گرامر مستقل از متن است که تقریبا $|w|^3$ مرحله برای تجزیه رشته w نیاز دارد.

این الگوریتم تعیین می کند که آیا رشته ای متعلق به زبان گرامر می باشد یا خیر.

مبتکر این الگوریتم J. Cocke، D. H. Younger و T. Kasami بودند.

الگوریتم در صورتی کار می کند که گرامر به فرم نرمال Chomsky باشد.

الگوریتم CYK از پردازش پایین به بالا استفاده می کند.

با استفاده از برنامه نویسی پویا dynamic programming پیاده سازی می شود.

در این الگوریتم ابتدا برای هر حرف یک جمله ورودی، تمام متغیرهایی که منجر به تولید آن حرف می‌شوند را در مجموعه ای مجزا قرار می‌دهد. بعد همین کار را برای زیرجمله‌های دوحرفی تکرار می‌کند و بعد زیرجمله‌های سه حرفی تا به کل جمله برسد. در نهایت اگر متغیر ابتدایی (سرترم) را در مجموعه متغیرهای جمله پایانی یافت، نتیجه می‌گیرد که جمله توسط گرامر تولید شده‌است.

فرمهای نرمال Normal forms

الگویتم CYK

فرض کنید گرامر $G=(V,\sum,P,S)$ به فرم نرمال شومسکی و $w=a_1a_2\dots a_n$ یک رشته باشد. یک زیررشته $w_{ij}=a_i\dots a_j$ مشخص می کنیم. و زیر مجموعه $V_{ij}=\{A\in V:A\overset{*}{\Rightarrow}w_{ij}\}$ را تعیین می کنیم.

$W\in L(G)$ اگر و فقط اگر $S\in V_{1n}$.

برای محاسبه V_{ij} در نظر بگیرید که $A\in V_{ii}$ اگر و فقط اگر G قانون $A\rightarrow a_i$ را داشته باشد. بنابراین V_{ii} برای همه $1\le i\le n$ با بررسی رشته و قوانین گرامر می تواند محاسبه شود. برای $j>i$

$$V_{ij}=\cup_{k\in\{i,i+1,\dots,j-1\}}\{A: A\rightarrow BC, with\ B\in V_{ik}, C\in V_{k+1,j}\}$$

از فرمول بالا همه V_{ij}ها به ترتیب محاسبه می شوند.

$V_{11}V_{22}\dots V_{nn}$
$V_{12}V_{23}\dots V_{n-1,n}$
$V_{13}V_{24}\dots V_{n-2,n}$ ....

فرمهای نرمال Normal forms

الگویتم CYK

مثال: فرض کنید w=aabbb و گرامر به صورت زیر باشد: $S \rightarrow AB$

$A \rightarrow BB \mid a$

$B \rightarrow AB \mid b$

$W_{11}=a$ $V_{11}=\{A\}$ $V_{12}=\{A: A \rightarrow BC, B \in V_{11}, C \in V_{22}\}$ AA
$W_{22}=a$ $V_{22}=\{A\}$ $V_{23}=\{A: A \rightarrow BC, B \in V_{22}, C \in V_{33}\}$ AB
$W_{33}=b$ $V_{33}=\{B\}$ $V_{34}=\{A: A \rightarrow BC, B \in V_{33}, C \in V_{44}\}$ BB
$W_{44}=b$ $V_{44}=\{B\}$ $V_{45}=\{A: A \rightarrow BC, B \in V_{44}, C \in V_{55}\}$ BB
$W_{55}=b$ $V_{55}=\{B\}$

$V_{12}=\emptyset$ $V_{23}=\{S, B\}$ $V_{34}=\{A\}$ $V_{45}=\{A\}$
$V_{13}=\{S, B\}$ $V_{24}=\{A\}$ $V_{35}=\{S, B\}$
$V_{14}=\{A\}$ $V_{25}=\{S, B\}$
$V_{15}=\{S, B\}$

بنابراین $W \in L(G)$

ماشین خودکار محدود قطعی DFA

ماشین خودکار محدود (Finite Automata)

به یک روال موثر برای تعیین اینکه آیا یک رشته ورودی متعلق به یک زبان است یا خیر یک پذیرنده زبان گفته می شود.

گرامر مولد زبان، عبارات مجموعه ای و عبارات باقاعده توصیفگر زبان و ماشین پذیرنده زبان است.

ویژگی مشترک تمامی ماشینها پردازش یک رشته و تولید خروجی است:

۱- ماشین های خودکار محدود (Finite Automata)

قطعی (Deterministic Finite Automata :DFA)

غیرقطعی (Non-deterministic Finite Automata :NFA)

غیرقطعی λ (Non-deterministic Finite Automata :NFA- λ)

۲- ماشینهای پشته ای(PDA: Push-Down Automata)

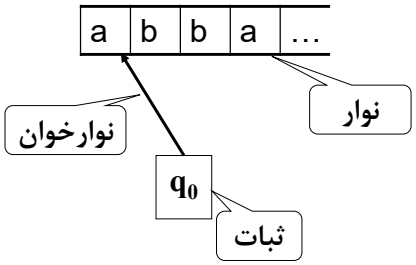
۳- ماشینهای تورینگ(Turing Machines)

ماشین خودکار محدود قطعی DFA

ماشین خودکار محدود قطعی (DFA: Deterministic Finite Automata)

تعریف: یک DFA به صورت پنج تایی $M=(Q,\sum,\delta,q_0,F)$ بیان می شود که در آن Q مجموعه متناهی از حالات(state) است که به صورت $(q_0,q_1,\ldots,q_n)$ نمایش می دهیم،$\sum$ مجموعه متناهی الفبا، $q_0 \in Q$ حالت شروع ماشین، F زیرمجموعه ای از Q است شامل حالات نهایی یا حالات پذیرش (accept final)، و δ یک تابع گذر از $Q \rightarrow Q\times\sum$ می باشد.

از آنجا که DFA مدل یک سخت افزار انتزاعی است می توان آن را شامل قسمتهای سخت افزاری زیر دانست.



۱- ثبات (register)

۲- نوار (حکم حافظه را دارد) tape

۳- نوارخوان tape reader

۴- مجموعه مقادیر ثبات $(q_0,q_1,\ldots,q_n)$

۵- مجموعه مقادیر نوار (عناصر الفبا)

۶-مجوعه دستورالعملها (توابع گذر)

ماشین خودکار محدود قطعی DFA

- ✓ هر نوار از سمت چپ محدود و از سمت راست نامحدود است. ورودی از چپ به راست روی آن نوشته می‌شود. نوار ورودی به خانه‌هایی تقسیم می‌شود که در هر قسمت تنها یک عنصر الفبا ذخیره می‌شود. قبل از شروع به کار ماشین تمام ورودی روی نوار قرار داده می‌شود.
- ✓ نوارخوان تنها قادر است اطلاعات را مرور کند و قادر به انتقال اطلاعات به/از نوار نیست. فقط روی نوار به سمت راست حرکت می‌کند. اجرای هر دستورالعمل، نوارخوان را یک واحد به سمت راست انتقال می‌دهد. هنگام شروع به کار ماشین، نوارخوان روی اولین خانه نوار قرار دارد.
- ✓ ثبات: تنها حافظه ماشین است. مقدار ذخیره شده در آن نشانگر وضعیت ماشین است. تعداد حالتهای یک ماشین متناهی، متناهی است.
- ✓ هر دستور العمل ماشین به صورت $\delta(q_i, a)=q_j$ تعریف می‌شودبه این معنی که اگر ماشین در حالت q_i و نوارخوان مقابل عنصر a باشد، نوارخوان یک واحد به راست انتقال داده می‌شود و حالت ماشین به q_j تبدیل می‌گردد.
- ✓ یک محاسبه عبارت است از اجرای دنباله ای از دستورالعمل ها است.

ماشین خودکار محدود قطعی DFA

- ✓ برای اینکه جمله w توسط ماشین M پذیرفته شود باید هر دو شرط زیر برقرار باشند: خاتمه ورودی- توقف در حالت نهایی
- ✓ اگر مقدار نهایی ثبات درمجموعه F باشد رشته موجود روی نوار پذیرفته می شود (accept).

فرض کنید که $M=(Q,\sum,\delta,q_0,F)$ یک DFA باشد. زبان M، که با L(M) نشان داده می شود، مجموعه ای از رشته ها در $\sum^*$ است که توسط M پذیرفته می شود.

عناصر سمت چپ نوارخوان در ادامه محاسبه تأثیری ندارند. در هر لحظه وضعیت ماشین به حالت فعلی، عنصر مقابل نوارخوان و عناصر سمت راست آن بستگی دارد.

اگر وضعیت ماشین q_i و باقیمانده ورودی wباشد (حرف اول w مقابل نوارخوان است)، $[q_i, w]$ را وضعیت لحظه‌ای ماشین می‌گویند.

دو ماشین معادلند اگر هر دو یک زبان را بپذیرند.

تابع guild به صورت $\vdash_M : Q\times\sum^+ \rightarrow Q\times\sum^*$ تعریف می شود بیانگر هر مرحله از محاسبه ماشین است.

$[q_i,aw] \vdash [\delta(q_i,a),w]$ $\quad w \in \sum^*$ $\quad a \in \sum$ $\quad q_i \in Q$

ماشین خودکار محدود قطعی DFA

مثال: محاسبه توسط ماشین

w=aba

$Q=\{q_0,q_1\}$

$\sum=\{a,b\}$

$F=\{q_1\}$

δ: $\delta(q_0,a)=q_1$

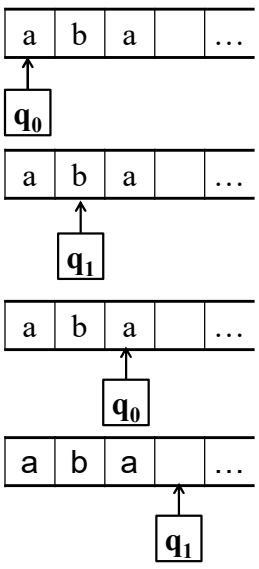
$\delta(q_0,b)=q_0$

$\delta(q_1,a)=q_1$

$\delta(q_1,b)=q_0$

L(M)= strings end with a

(a U b)* a



ماشین خودکار محدود قطعی DFA

مثال: ماشین متناهی M با $\Sigma=\{a, b\}$، $Q=\{q_0, q_1, q_2\}$ و $F=\{q_2\}$ را در نظر بگیرید. تابع گذر حالات آن به صورت جدول مقابل که جدول گذر حالات نامیده می‌شود، نشان داده شده است. محاسبه ماشین برای جملات **abba** و **abab** را بنویسید.

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2

$L(M)=(aUb)^*bb(aUb)^*$

مجموعه رشته هایی که حداقل یک **bb** دارند.

$[q_0, abab] \vdash [q_0, bab] \vdash [q_1, ab] \vdash [q_0, b] \vdash [q_1, \lambda] \vdash [q_0, abab] \vdash^* [q_1, \lambda]$

جمله پذیرفته نمی‌شود.

$[q_0, abba] \vdash [q_0, bba] \vdash [q_1, ba] \vdash [q_2, a] \vdash [q_2, \lambda] \vdash [q_0, abba] \vdash^* [q_2, \lambda]$

جمله پذیرفته می‌شود.

ماشین خودکار محدود قطعی DFA

قضیه: تابع گذر توسعه یافته **extended transition** که آن را با $\hat{\delta}$ نمایش می دهند تابعی به صورت $\hat{\delta}: Q\times\sum^* \rightarrow Q$ می باشد که به صورت زیر تعریف می شود:

۱) حالت پایه: $length(w)=0$. در این صورت $w=\lambda$ و $\hat{\delta}(q_i, \lambda)= q_i$ است.
$length(w)=1$. در این صورت $w=a$ (برای $a\epsilon\sum$) و $\hat{\delta}(q_i,a)= \delta(q_i,a)$ است.

۲) قانون بازگشتی: اگر $length(w)>1$ باشد. آنگاه $w=ua$ (برای $u\epsilon\sum^+$ و $a\epsilon\sum$) و $\hat{\delta}(q_i,ua)= \delta(\hat{\delta}(q_i,u),a)$ خواهد بود.

محاسبه یک ماشین در حالت q_i با رشته ورودی w در حالت $\hat{\delta}(q_i,w)$ متوقف می شود رشته w پذیرفته می شود اگر $\hat{\delta}(q_i,w)\ \epsilon\ F$ و زبان ماشین M به صورت زیر تعریف می شود:

$L(M)=\{w|\ \hat{\delta}(q_i,w)\ \epsilon\ F\}$

ماشین خودکار محدود قطعی DFA

مثال: ماشین متناهی M با $\Sigma=\{a, b\}$، $Q=\{q_0, q_1, q_2\}$ و $F=\{q_2\}$ را در نظر بگیرید. تابع گذر حالات آن به صورت جدول مقابل که جدول گذر حالات نامیده می‌شود، نشان داده شده است. محاسبه ماشین برای جملات abba را با تابع گذر توسعه یافته بنویسید.

$L(M)=(aUb)^*bb(aUb)^*$

مجموعه رشته هایی که حداقل یک bb دارند.

$\hat{\delta}(q_0,abba)$

$=\delta(\hat{\delta}(q_0,abb), a)$

$=\delta(\delta(\hat{\delta}(q_0,ab),b), a)$

$=\delta(\delta(\delta(\hat{\delta}(q_0,a),b),b), a)$

$=\delta(\delta(\delta(\delta(q_0,a),b),b), a)$

q_1 q_0

q_2 q_2

q_2

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2

ماشین خودکار محدود قطعی DFA

نمودار حالت (state diagram)

می توان ماشین DFA را با استفاده از نمودار نیز بیان کرد.

نمودار حالت یک DFA $(Q,\sum, \delta,q_0,F)$ یک گراف جهت دار بر چسب دار G با شرایط زیر است:

۱) گره های گراف عناصر مجموعه Q هستند.

۲) لبه های گراف عناصر مجموعه $\sum$ هستند.

۳) q_0 گره شروع با علامت است.

۴) F مجموعه گره های پذیرش است که هرگره پذیرش با  نمایش داده می شود.

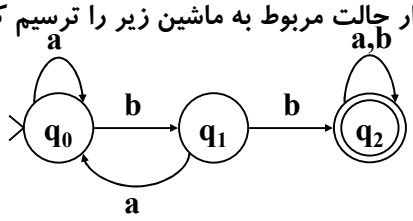
۵) یک لبه از گره q_i به q_j با برچسب a وجود دارد اگر $\delta(q_i,a)= q_j$ باشد.

۶) برای هر گره q_i و نماد a دقیقاً و فقط یک لبه با برچسب a وجود دارد که از q_i خارج می شود.

ماشین خودکار محدود قطعی DFA

مثال: نمودار حالت مربوط به ماشین زیر را ترسیم کنید.

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2



قضیه: فرض کنید که $M=(Q,\sum,\delta,q_0,F)$ یک DFA و $w \in \sum^*$ باشد. در آنصورت برای رشته w مسیرمنحصربه فرد p_w در نمودار حالت ماشین وجود دارد و $\hat{\delta}(q_0,w)= q_w$

مثال: محاسبه DFA با ورودی w=ababb و مسیر متناظر آن در نمودار حالت:

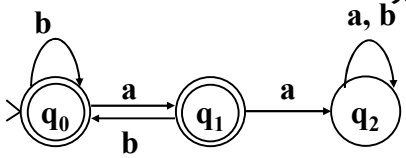
مسیر: $q_0q_0q_1q_0q_1q_2$

رشته ababb قابل پذیرش است زیرا در حالت پایانی q_2 متوقف شده است.

محاسبه	مسیر
$[q_0,ababb]$	q_0
├ $[q_0,babb]$	q_0
├ $[q_1,abb]$	q_1
├ $[q_0,bb]$	q_0
├ $[q_1,b]$	q_1
├ $[q_2,\lambda]$	q_2

ماشین خودکار محدود قطعی DFA

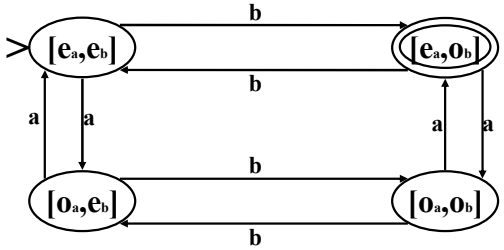
مثال: ماشین DFA زیر چه زبانی را می پذیرد.



$L(M)=(b \cup ab)^* (a \cup \lambda)$

مثال: برای پذیرش زبان زیر یک DFA طراحی کنید.

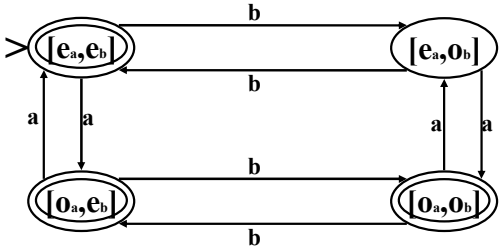
مجموعه رشته هایی که تعداد a زوج و تعداد b فرد است.



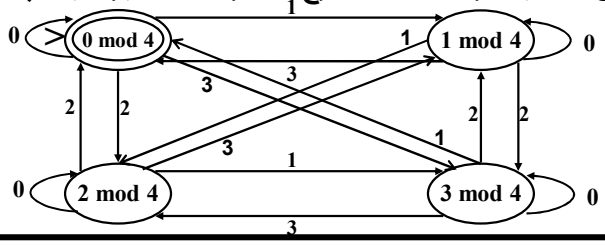
قضیه: فرض کنید $M=(Q,\sum,\delta,q_0,F)$ یک DFA باشد درآنصورت $M'=(Q,\sum,\delta,q_0,Q-F)$ یک DFA با زیر است. $L(M')= \sum^* - L(M)$

ماشین خودکار محدود قطعی DFA

مثال: فرض کنید زبان مثال قبل M باشد. ماشین DFA طراحی کنید به گونه ای که زبان آن M' به صورت زیر باشد. $L(M')= \sum^* - L(M)$

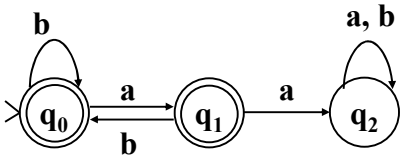


مثال: فرض کنید $\sum =\{0,1,2,3\}$ و $w \in \sum^*$ دنباله ای از اعداد صحیح باشد. یک DFA طراحی کنید به گونه ای که مجموع عناصر رشته w بر ۴ بخشپذیر باشد.



ماشین خودکار محدود قطعی DFA

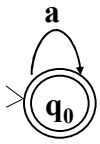
مثال: ماشین DFA زیر چه زبانی را می پذیرد.



$L(M)=(b \cup ab)^* (a \cup \lambda)$

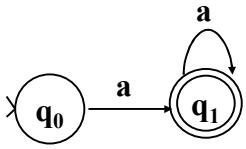
مثال: نمودار حالتی برای زبان زیر طراحی کنید.

$L=a^*$



مثال: نمودار حالتی برای زبان زیر طراحی کنید.

$L=a^+$



ماشین خودکار محدود قطعی DFA

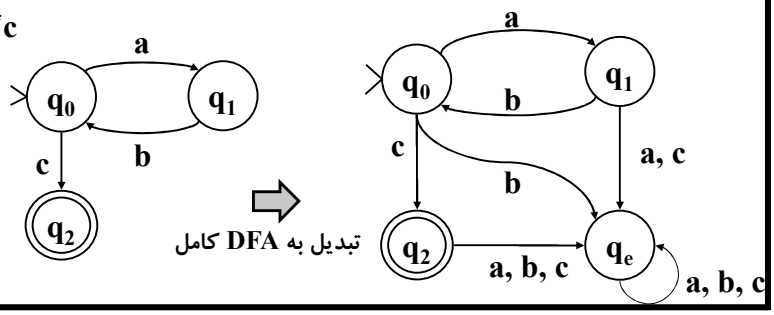
اگر به ازای هر زوج مرتب (q_i,a) یک تابع گذر در DFA موجود باشد به آن DFA کامل یا DFA complete گفته می شود. تمامی مثالهای قبل از این نوعند.

اگر چنین وضعیتی برقرار نباشد ماشین DFA حاصل ناقص یا DFA incomplete گویند.

هر DFA ناقص را با افزودن یک حالت جدید به نام حالت خطا می توان به DFA کامل تبدیل نمود.

مثال: نمودار حالتی برای زبان زیر طراحی کنید.

$L=(ab)^*c$



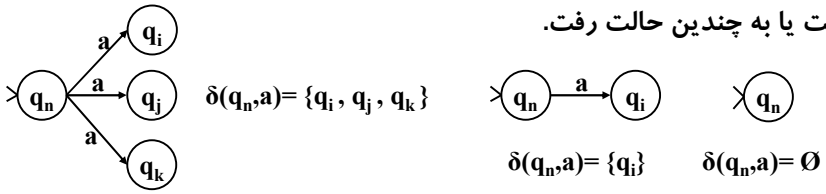
ماشین خودکار محدود غیرقطعی NFA

ماشین خودکار محدود غیرقطعی (NFA: Non-deterministic Finite Automata)

تعریف: یک NFA به صورت پنج تایی $M=(Q,\sum,\delta,q_0,F)$ بیان می شود که در آن Q مجموعه متناهی از حالات(state) است که به صورت $(q_0,q_1,\ldots,q_n)$ نمایش می دهیم، $\sum$ مجموعه متناهی الفبا، $q_0 \in Q$ حالت شروع ماشین، F زیرمجموعه ای از Q است شامل حالات نهایی یا حالات پذیرش (accept final)، و δ یک تابع گذر از$Q\times\sum \rightarrow P(Q)$ می باشد.

P(Q) (مجموعه توانی): مجموعه های تمام زیرمجموعه Q را مجموعه توانی گویند.

این امکان وجود دارد که در ماشین های خودکار از یک وضعیت ماشین بتوان چند دستور را برای اجرا به عنوان کاندید انتخاب کرد. تنها تفاوت ماشین قطعی با غیرقطعی در این است که در ماشین غیرقطعی می توان با دیدن کمان به حالتی نرفت یا به چندین حالت رفت.



ماشین خودکار محدود غیرقطعی NFA

هر DFA یک NFA است.

مثال: ماشین متناهی M با $\Sigma=\{a, b\}$، $Q=\{q_0, q_1, q_2\}$ و $F=\{q_2\}$ را در نظر بگیرید. تابع گذر حالات آن به صورت جدول مقابل که جدول گذر حالات (Transition Table) نامیده می‌شود، نشان داده شده است. محاسبه ماشین برای رشته ababb را بنویسید.

δ	a	b
q_0	$\{q_0\}$	$\{q_0,q_1\}$
q_1	$\varnothing$	$\{q_2\}$
q_2	$\varnothing$	$\varnothing$

$L(M)=(a \cup b)^* bb$ و طبق محاسبه ۳، $w \in L(M)$

محاسبه۱	محاسبه۲	محاسبه۳
$[q_0, ababb]$	$[q_0, ababb]$	$[q_0, ababb]$
$\vdash [q_0, babb]$	$\vdash [q_0, babb]$	$\vdash [q_0, babb]$
$\vdash [q_0, abb]$	$\vdash [q_1, abb]$	$\vdash [q_0, abb]$
$\vdash [q_0, bb]$	$q_1 \notin F$	$\vdash [q_0, bb]$
$\vdash [q_0, b]$		$\vdash [q_1, b]$
$\vdash [q_0, \lambda]$		$\vdash [q_2, \lambda]$
$q_0 \notin F$		$q_2 \in F$

ماشین خودکار محدود غیرقطعی NFA

تعریف: زبان ماشین NFA که با L(M) نشان داده می شود، مجموعه ای از رشته های پذیرفته شده به وسیله ماشین است. به گونه ای که

$$L(M)=\{w \mid [q_0,w] \overset{*}{\vdash} [q_i,\lambda];\ q_i \in F\}$$

رشته ورودی مورد پذیرش است اگر محاسبه ای موجود باشد به گونه ای که تمامی رشته ورودی را خوانده و در حالت پذیرش متوقف گردد.

نمودار حالت یک NFA $(Q,\sum, \delta,q_0,F)$ یک گراف جهت دار بر چسب دار G با شرایط زیر است:

۱) گره های گراف عناصر مجموعه Q هستند.

۲) لبه های گراف عناصر مجموعه $\sum$ هستند.

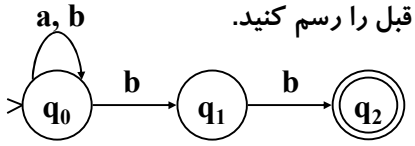
۳) q_0 گره شروع است.

۴) F مجموعه گره های پذیرش است.

۵) یک لبه از گره q_i به q_j با برچسب a وجود دارد اگر $\delta(q_i,a)= q_j$ باشد.

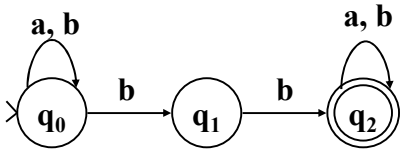
ماشین خودکار محدود غیرقطعی NFA

مثال: دیاگرام حالت ماشین مثال قبل را رسم کنید.

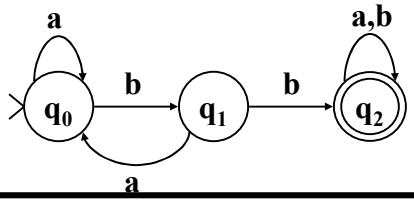


مثال: برای زبان زیر یک NFA و یک DFA طراحی کنید.

L={w| w contains bb}, Σ={a,b} (a U b)* bb (a U b)*



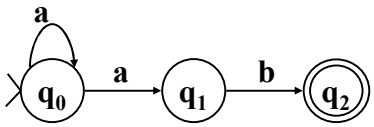
NFA



DFA

ماشین خودکار محدود غیرقطعی NFA

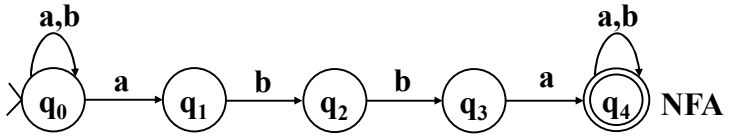
مثال: برای زبان زیر یک NFA طراحی کنید. $L=a^+b$



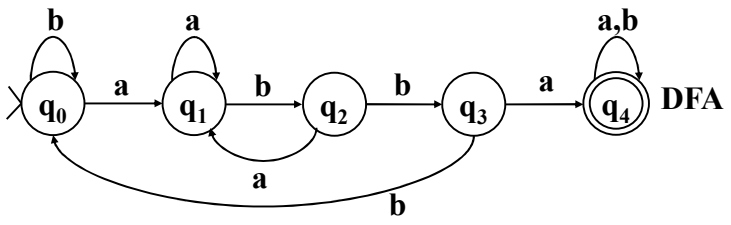
NFA

مثال: برای پذیرش زبان زیر یک NFA و یک DFA طراحی کنید.

L={w| w contains abba}, Σ={a,b}



NFA



DFA

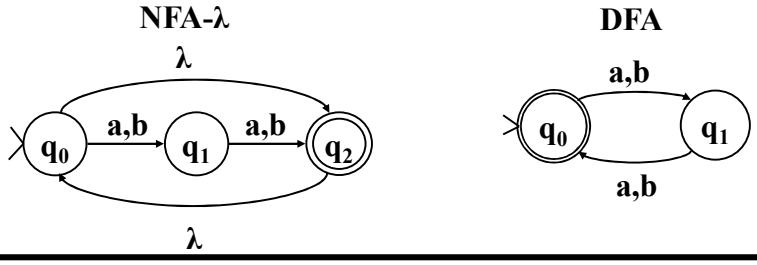
ماشین خودکار محدود غیرقطعی NFA-λ

گذر لامبدا گذری است که باعث تغییر حالت یک ماشین از q_i به q_j شده بدون آنکه عنصری از ورودی پردازش شود. ماشینی که دارای این نوع گذرها باشد NFA-λ نامیده می شود.

تعریف: یک NFA-λ به صورت پنج تایی $M=(Q,\sum,\delta,q_0,F)$ بیان می شود که چهار پارامتر Q، $\sum$، q_0 و F مانند NFA هستند و δ به صورت زیر تعریف می شود: $Q\times(\sum \cup \{\lambda\}) \rightarrow P(Q)$

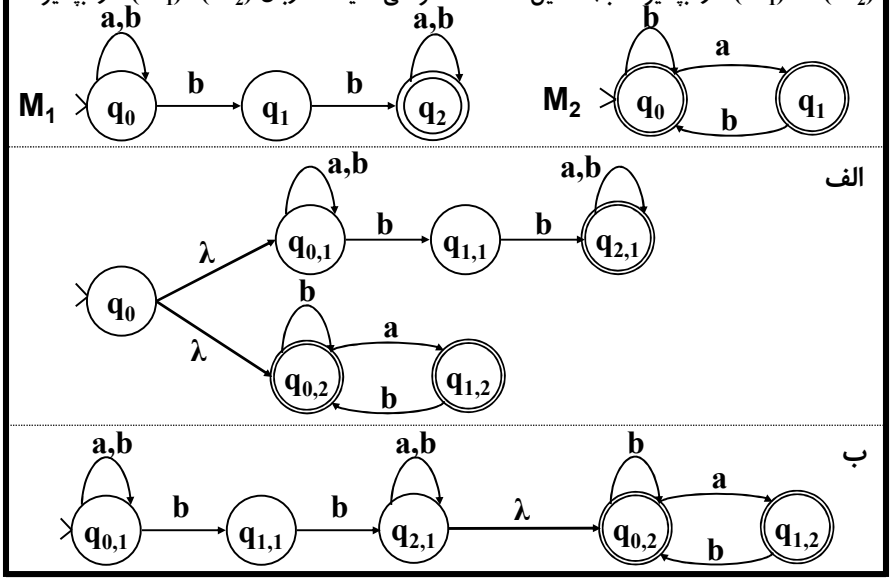
مثال: یک NFA-λ برای زبان زیر طراحی کنید.

مجموعه رشته هایی به طول زوج روی {a,b}



ماشین خودکار محدود غیرقطعی NFA-λ

مثال: دو ماشین M_1 و M_2 را درنظر بگیرید. الف) ماشین NFA-λ طراحی کنید که زبان $L(M_1)UL(M_2)$ را بپذیرد. ب) ماشین NFA-λ طراحی کنید که زبان $L(M_1)L(M_2)$ را بپذیرد.



ماشین خودکار محدود غیرقطعی NFA-λ

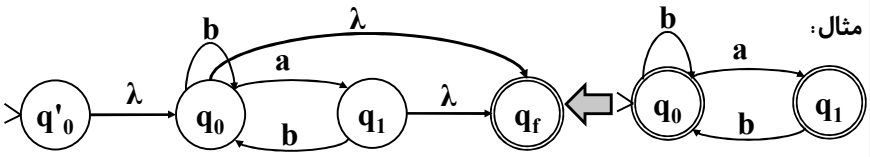
قضیه: فرض کنید $M=(Q,\sum,\delta,q_0,F)$ یک NFA-λ باشد در آنصورت NFA-λ معادل $M'=(QU\{q'_0,q_f\},\sum,\delta',q'_0,\{q_f\})$ وجود دارد که دارای شرایط زیر است.

الف) درجه ورودی q'_0 صفر است.

ب)تنها حالت پذیرش ماشین M'، q_f است.

ج)درجه خروجی q_f صفر است.

مثال:



کافیست توابع گذر زیر را به δ اضافه کنیم:

$\delta'(q_i,a)=\delta(q_i,a)$ $\quad a \in \sum, q_i \in Q$

$\delta'(q'_0,\lambda)=\{q_0\}$

$\delta'(q_i,\lambda)=\{q_f\}$ $\quad q_i \in F$

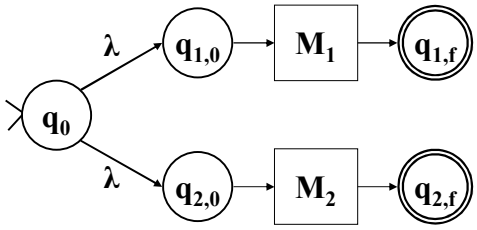
ماشین خودکار محدود غیرقطعی NFA-λ

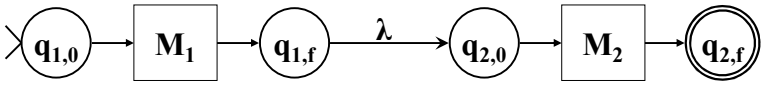
قضیه: فرض کنید M_1 و M_2 دو NFA-λ باشند در آنصورت NFA-λ وجود دارد که زبانهای زیر را بپذیرد.

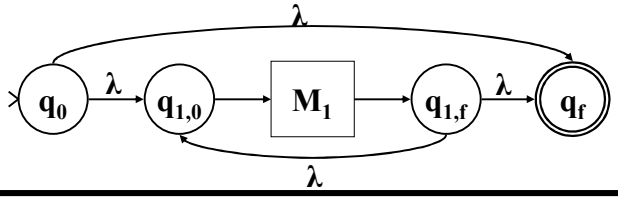
$L(M_1)UL(M_2)$

$L(M_1)L(M_2)$

$L(M_1)^*$







ماشین خودکار محدود غیرقطعی λ-NFA

رفع عدم قطعیت:

قدرت پذیرش هر سه نوع ماشین DFA، NFA و λ-NFA یکی است. الگوریتمی برای تبدیل هرλ-NFA به DFA معادل با آن بیان می کنیم.

تابع بستار لامبدا (lambda-closure)

تعریف: بستار لامبدا حالت q_i که با نماد λ-closure(q_i) بیان می کنیم به صورت زیر تعریف می شود:

۱- حالت پایه: $q_i \in \lambda$-closure(q_i)

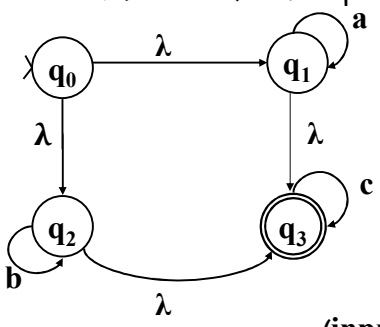
۲- قانون بازگشتی: اگر $q_j \in \lambda$-closure(q_i) و $q_k \in \delta(q_j,\lambda)$ آنگاه $q_k \in \lambda$-closure(q_i)

۳- بسته بودن: $q_j \in \lambda$-closure(q_i) اگر و فقط اگر با انجام مرحله دوم به تعداد متناهی حاصل شود.

هدف از این تابع بیان این مطلب است که از حالت q_i فقط با دیدن λ به چه حالتهایی می توان رفت.

ماشین خودکار محدود غیرقطعی λ-NFA

مثال: در ماشین زیر λ-closure(q_i) را برای تمام حالتها را به دست آورید.



λ-closure(q_0)= $\{q_0, q_1, q_2, q_3\}$

λ-closure(q_1)= $\{q_1, q_3\}$

λ-closure(q_2)= $\{q_2, q_3\}$

λ-closure(q_3)= $\{q_3\}$

تابع گذر ورودی (input transition function)

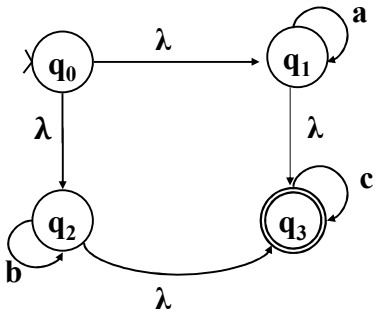
تابع گذر ورودی را با نماد t نشان می دهند تابع $t: Q\times\sum \rightarrow P(Q)$ است و تعریف آن به صورت زیر است.

$$t(q_i,a)=\bigcup_{q_j \in \lambda\text{-closure}(q_i)} \lambda\text{-closure}(\delta(q_j,a))$$

مفهوم این تابع این است که از حالت q_i با دیدن فقط یک a به چه حالتهایی می توان رفت.

ماشین خودکار محدود غیرقطعی NFA-λ

مثال: تابع t را برای ماشین زیربه دست آورید.



$t(q_0,a)= \{q_1, q_3\}$ $t(q_0,b)= \{q_2, q_3\}$

$t(q_1,a)= \{q_1, q_3\}$ $t(q_1,b)= \emptyset$

$t(q_2,a)= \emptyset$ $t(q_2,b)= \{q_2, q_3\}$

$t(q_3,a)= \emptyset$ $t(q_3,b)= \emptyset$

$t(q_0,c)= \{q_3\}$

$t(q_1,c)= \{q_3\}$

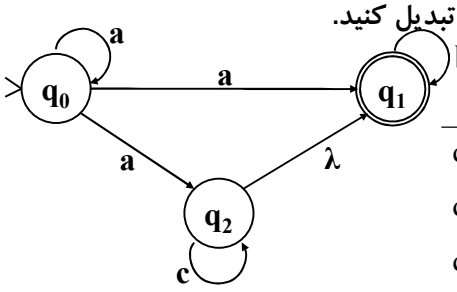
$t(q_2,c)= \{q_3\}$

$t(q_3,c)= \{q_3\}$

t	q_0	q_1	q_2	q_3
a	$\{q_1, q_3\}$	$\{q_1, q_3\}$	$\emptyset$	$\emptyset$
b	$\{q_2, q_3\}$	$\emptyset$	$\{q_2, q_3\}$	$\emptyset$
c	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$	$\{q_3\}$

ماشین خودکار محدود غیرقطعی NFA-λ

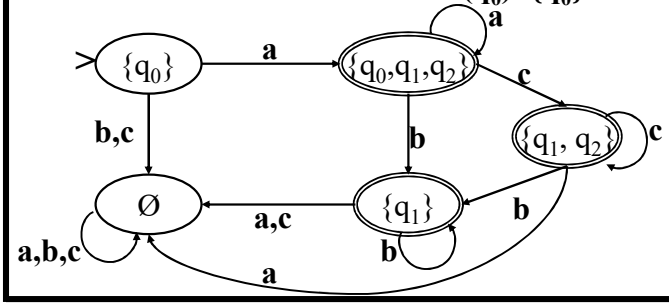
مثال: ماشین NFA-λ زیر را به DFA تبدیل کنید.



$F=\{q_1\}$

t	a	b	c
q_0	$\{q_0, q_1, q_2\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_1\}$	$\{q_1, q_2\}$

حالت شروع ماشین: DFA، $\lambda\text{-closure}(q_0)=\{q_0\}$



$L(M)=a^+c^*b^*$

ارتباط عبارت باقاعده و ماشین های خودکارمحدود

ماشین های خودکار محدود فقط درباره عبارت باقاعده است.

الف) ایجاد عبارت باقاعده از FA (ماشینهای خودکار محدود)

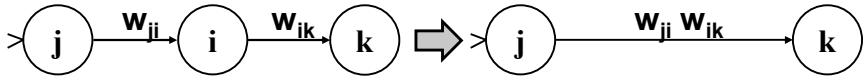
الگوریتم:

۱- به تعداد حالات پذیرش FA از آن کپی می سازیم و در هرکپی یکی از حالات پذیرش را در نظر می گیریم.

۲- برای هرکپی عملیات زیر را انجام می دهیم:

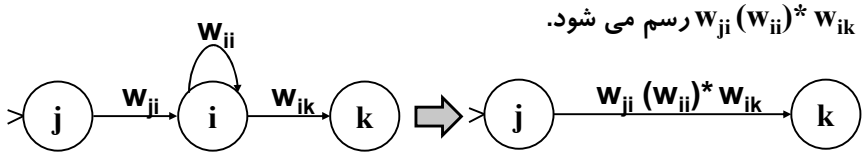
الف) سه متغیر i، j و k را بدین صورت مقداردهی می کنیم که i شماره حالتی است که نه شروع و نه حالت پذیرش است و k و j شماره حالاتی خواهند بود که با i یکی نیست.

ب) اگر $w_{ji}\neq Ø$، $w_{ik}\neq Ø$، $w_{ii}=Ø$ در آنصورت از j به k لبه ای با برچسب $w_{ji}\, w_{ik}$ رسم می شود.

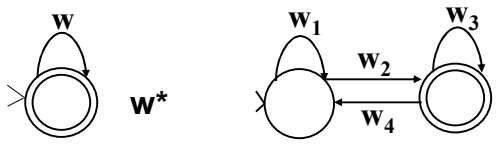


ارتباط عبارت باقاعده و ماشین های خودکارمحدود

ج) اگر $w_{ji}\neq Ø$، $w_{ik}\neq Ø$، $w_{ii} \neq Ø$ در آنصورت از j به k لبه ای با برچسب $w_{ji}\,(w_{ii})^*\, w_{ik}$ رسم می شود.



د) پس از انجام مراحل فوق گره شماره i را حذف می کنیم و عملیات فوق آنقدر تکرار می شودتا به یکی از دو حالت زیر می رسیم. (بعضی w ها می توانند تهی باشند)



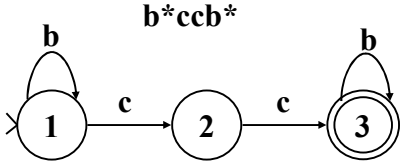
۳- زبانهای به دست آمده از هر کپی طبق روش فوق را با عملگر اجتماع ترکیب کرده و زبان نهایی ماشین به دست می آید.

ارتباط عبارت باقاعده و ماشین های خودکارمحدود

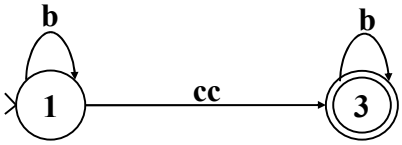
دیاگرام حالت یک ماشین با الفبای $\sum$ را می توان به عنوان یک گراف عبارت در نظر گرفت. برچسب ها شامل لامبدا و عبارات متناظر با عناصر $\sum$ می باشند. کمانهای یک عبارت می توانند با ø نیز برچسب دار شوند. هر مسیر در یک گراف عبارت، یک عبارت با قاعده تولید می کند. زبان یک گراف عبارت اجتماعی از مجموعه های ارائه شده توسط عبارات با قاعده پذیرفته شده است.

نمودار حالتی که روی لبه های (کمان) آن عبارات باقاعده نوشته شوند، گراف عبارت گفته می شود.

b*ccb*



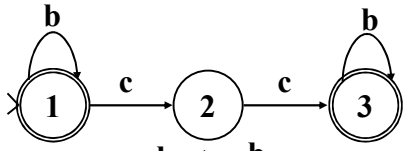
نمودار حالت

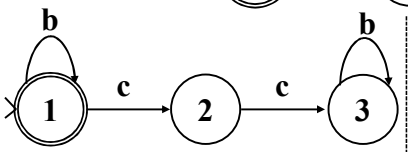


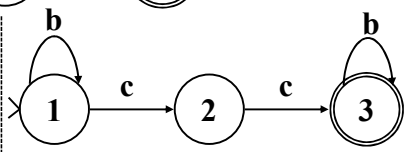
گراف عبارت

ارتباط عبارت باقاعده و ماشین های خودکارمحدود

مثال:

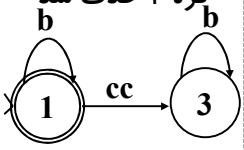






j	i	k	
1	2	1	✖
1	2	3	cc
3	2	1	✖
3	2	3	✖

گره ۲ حذف شد



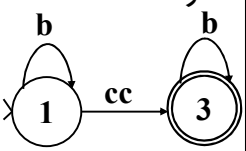
گره ۳حذف شد

j	i	k	
1	3	1	✖

b*

j	i	k	
1	2	1	✖
1	2	3	cc
3	2	1	✖
3	2	3	✖

گره ۲ حذف شد



b*ccb*

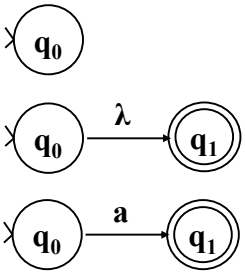
L(M)=b* U b*ccb*

ارتباط عبارت باقاعده و ماشین های خودکارمحدود

ب) ایجاد ماشین خودکار متناهی (محدود) از عبارت باقاعده

الگوریتم:

۱- حالت پایه



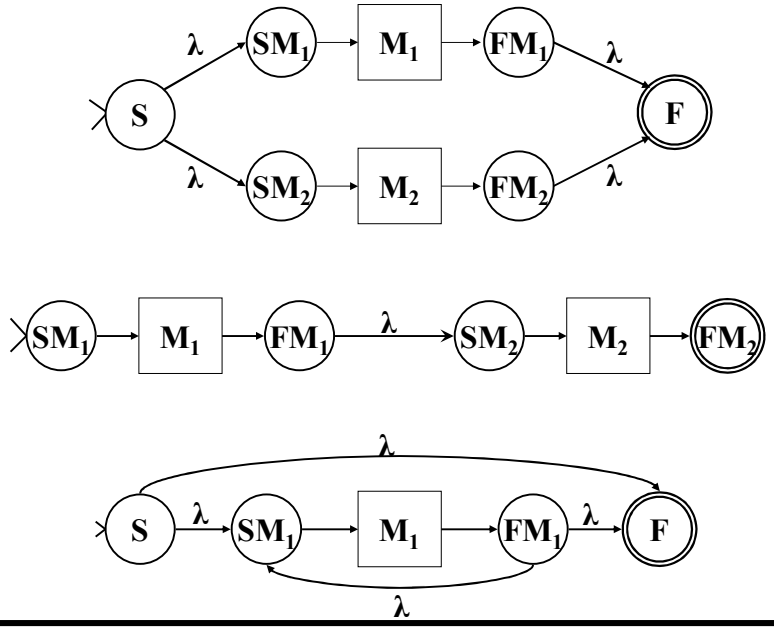
۲- قانون بازگشتی: فرض کنید FM_1 و SM_1 و FM_2 و SM_2 حالتهای پذیرش و شروع ماشین های M_1 و M_2 باشند. F و S حالت ماشین ترکیبی حاصل است.

$L(M_1)UL(M_2)$

$L(M_1)L(M_2)$

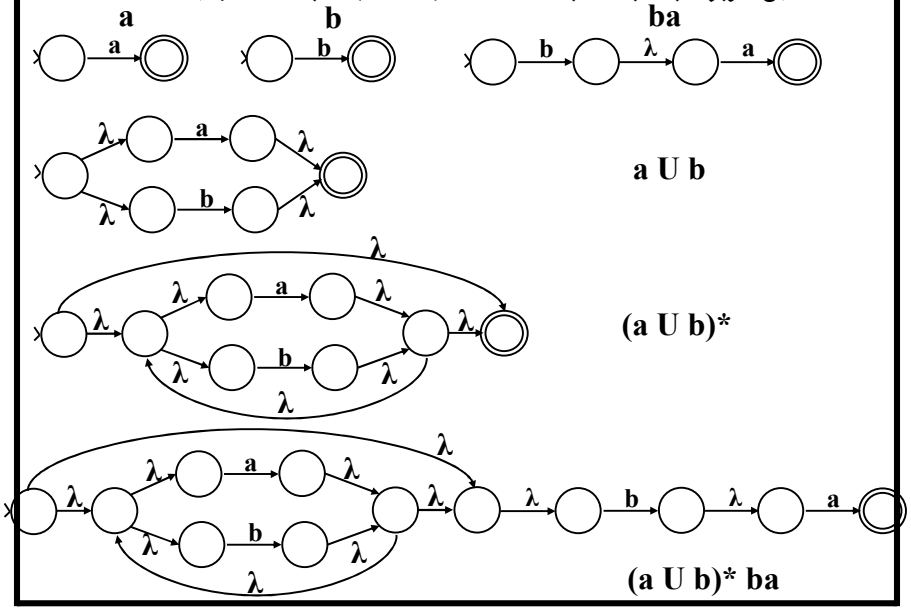
$L(M_1)^*$

ارتباط عبارت باقاعده و ماشین های خودکارمحدود



ارتباط عبارت باقاعده و ماشین های خودکارمحدود

مثال: ماشین مربوط به عبارت باقاعده ba*(a U b) را بدست آورید.



ارتباط عبارت باقاعده و ماشین های خودکارمحدود

قضیه kleen: زبان L توسط یک DFA با الفبای∑ پذیرفته می شود اگر و تنها اگر L یک عبارت با قاعده روی∑ باشد.

قضیه: فرض کنید G=(V,∑,P,S) یک گرامر باقاعده باشد درآنصورت ماشین NFA، M=(Q,∑,δ,S,F) را به صورت زیر تعریف می کنیم. (Z در صورت A → b اضافه می شود و Z همیشه حالت پذیرش است)

Q= V U {Z} if A → a ϵ P

Q= V else

δ(A,a)=B if A → aB ϵ P

δ(A,a)=Z if A → a ϵ P

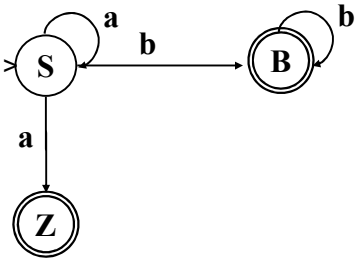
F={A|A → λ ϵ P} U {Z} if Z ϵ Q

F={A|A → λ ϵ P} else

ارتباط عبارت باقاعده و ماشین های خودکارمحدود

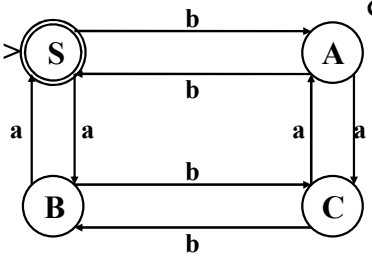
مثال: گرامر زیر را در نظر بگیرید. NFA معادل آن را ترسیم کنید.

G: S → aS | bB | a
B → bB | λ



مثال: گرامر زیر را در نظر بگیرید. NFA معادل آن را ترسیم کنید. تعداد a و تعداد b زوج

G: S → aB | bA | λ
A → bS | aC
B → bC | aS
C → aA | bB



زبان بی قاعده و باقاعده

مثال: زبان $L=\{i\geq 0, a^ib^i\}$ با قاعده نیست(بی قاعده است).

مثال: زبان $L=\{i\leq n, a^ib^i\}$ با قاعده است.

مثال: اگر L شامل تمام رشته های متقارن روی {a,b} باشد، L زبان بی‌قاعده است.

تعریف: تمامی زبانهای محدود باقاعده هستند.

مثال: زبان $L=\{ a^i \mid i$ عدد اول است $\}$ با قاعده نیست(بی قاعده است).

مثال: زبان $L=\{w|$length(w) مربع کامل است$\}$ با قاعده نیست.

تعریف: فرض کنید L_1 و L_2 زبانهای باقاعده پذیرفته شده توسط M_1 و M_2 باشند درآنصورت $L=(L_1\cap\overline{L_2})\cup(\overline{L_1}\cap L_2)$ زبان نیز باقاعده است. L تهی است اگر و تنها اگر L_1 و L_2 یکسان باشند.

مثال: نشان دهید زبان L باقاعده است: رشته هایی روی {a,b} که دارای زیررشته aa بوده ولی زیررشته bb در آنها وجود ندارد. زبان باقاعده تحت ∩ و ¯ بسته است.

$L_1=(a \cup b)^*$ aa $(a \cup b)^*$ $\quad L=L_1\cap\overline{L_2}$
$L_2=(a \cup b)^*$ bb $(a \cup b)^*$

زبان بی قاعده و باقاعده

فرض کنید L زبانی شامل رشته های u_i و v_i $(i\geq 0)$ با شرایط $u_iv_i\epsilon L$ ولی $u_iv_j\notin L$ $i,j\geq 0$, $i\neq j$ باشد در آنصورت L یک زبان باقاعده نیست.

مثال: فرض کنید L شامل تمامی رشته های متقارن روی $\Sigma=\{a, b\}$ باشد، ثابت کنید این زبان بی قاعده است.

اگر u_i و v_i $(i\geq 0)$ با شرایط $u_iv_i\epsilon L$ ولی $i\neq j$, $i,j\geq 0$ $u_iv_j\notin L$ پیدا کنیم در آنصورت L باقاعده نیست. اگر $u_i=a^i$ و $v_i=ba^i$ باشد، L باقاعده نیست.

مثال: اثبات کنید زبانهای برنامه نویسی که در آنها می توان عبارات منطقی و حسابی را به فرم infix نوشت، باقاعده نیستند.

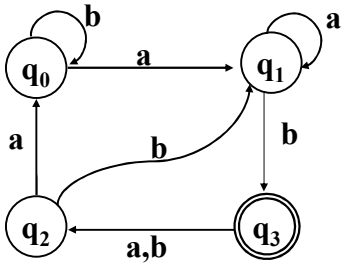
اگر u_i و v_i $(i\geq 0)$ با شرایط $u_iv_i\epsilon L$ ولی $i\neq j$, $i,j\geq 0$ $u_iv_j\notin L$ پیدا کنیم در آنصورت L باقاعده نیست. اگر $u_i=(^i\ b$ و $v_i=)^i$ باشد، L باقاعده نیست.

مثال: اثبات کنید که رشته های روی $\Sigma=\{(,)\}$ که در آن پرانتزها جفت هستند()، (()) و ... باقاعده نیستند.

اصل pumping برای زبانهای باقاعده

pumping: عمل pumping به معنای ایجاد رشته های جدید با تکرار زیر رشته هایی در داخل رشته اصلی است.

مثال: DFA زیر را در نظر بگیرید.



فرض کنید $z=ababbaaab \in L(M)$

باشد و $u=a$، $v=bab$، $w=baaab$ در نتیجه

$z=uvw$

رشته های $a(bab)^ibaaab$ با pumping

زیررشته bab در داخل رشته z بدست می آید.

مسیرتولید شده توسط v حلقه است بنابراین تکرار زیر رشته v باعث می گردد رشته های جدید حاصل نیز به $L(M)$ متعلق باشد.

قضیه: فرض کنید G نمودار حالت یک DFA با k حالت باشد در آنصورت هر مسیر به طول k دارای حداقل یک حلقه است.

اصل pumping برای زبانهای باقاعده

قضیه: فرض کنید که G دیاگرام حالت یک DFA با k حالت بوده و p یک مسیر به طول k یا بیشتر باشد. در این صورت مسیر p را می توان به زیر مسیرهای q، r وs تجزیه نمود به گونه‌ای که: p=qrs و length(qr)≤k بوده و r یک حلقه باشد.

قضیه pumping: فرض کنید که L یک زبان با قاعده است که توسط یک DFA M با k حالت پذیرفته می شود و فرض کنید که z هر رشته موجود در L با length(z)≥k باشد. در این صورت z می تواند به صورت uvw نوشته شود بطوریکه length(uv)≤k، length(v)>0 و به ازای هر i≥0 ، $uv^iw \in L$ باشد.

تمامی زبانهای محدود باقاعده هستند.

قضیه pumping ابزار قدرتمندی برای اثبات باقاعده نبودن زبانها است. رشته دلخواهی مانند $z \in L$ را انتخاب کرده و نشان می دهیم در شرایط قضیه pumping صدق نمی کند.

اصل pumping برای زبانهای باقاعده

مثال: ثابت کنید $L=\{z\in\{a,b\}^* | length(z) \text{ is perfect square}\}$ باقاعده نیست.

فرض می شود L با قاعده است بنابراین توسط یک DFA با k حالت پذیرفته می شود. بنابر قضیه pumping هر رشته $z\epsilon L$ با length(z)≥k به سه زیر رشته صورت uvw نوشته شود بطوریکه length(uv)≤k، length(v)>0 و به ازای هر $uv^iw \in L$ ، i≥0 باشد.

رشته z به طول k^2 در نظر گرفته می شود. بنابر قضیه pumping این رشته باید بتواند به سه زیر رشته u، w، v که در آن v دارای شرط 0< length(v)≤k است تقسیم گردد.

رشته uv^2w (i=2) را در نظر گرفته و نشان می دهیم که طول این رشته نمی تواند مربع کامل باشد. $k^2<length(uv^2w)= length(uvw)+length(v)$

$$= k^2+length(v)\leq k^2+k$$
$$<(k^2+k+1)=(k+1)^2$$

روابط فوق نشان می دهد که طول رشته uv^2w مربع کامل نیست. در نتیجه z را نمی توان به سه زیررشته تقسیم نمود. به تناقض رسیده، در نتیجه L باقاعده نیست.

اصل pumping برای زبانهای باقاعده

مثال: ثابت کنید $L=\{a^i | i \text{ is prime}\}$ باقاعده نیست.

فرض می شود L با قاعده است بنابراین توسط یک DFA با k حالت پذیرفته می شود. n را عدد اولی بزرگتر از k در نظر می گیریم. بنابر قضیه pumping هر رشته $z \epsilon L$ با $length(z) \geq k$ به سه زیر رشته صورت uvw نوشته شود بطوریکه $length(uv) \leq k$، $length(v) > 0$ و به ازای هر $i \geq 0$ ، $uv^i w \epsilon L$ باشد.

رشته $z=a^n$ در نظر گرفته می شود. طول رشته $uv^{n+1}w$ باید اول باشد اگر $z \epsilon L$.

$$length(uv^{n+1}w) = length(uvw) + length(v^n)$$
$$= n + n\, length(v) = n(1 + length(v))$$
$$= n \times m \quad (m \geq 2)$$

روابط فوق نشان می دهد که طول رشته $uv^{n+1}w$ اول نیست. در نتیجه a^n را نمی توان به سه زیررشته تقسیم نمود. به تناقض رسیده، در نتیجه L باقاعده نیست.

اصل pumping برای زبانهای باقاعده

قضیه: فرض کنید که Mیک DFA با k حالت باشد.

الف) L(M) تهی نیست اگر و فقط اگر M رشته z با $length(z) < k$ را بپذیرد.

ب) L(M) دارای تعداد نامحدودی عضو است اگر و فقط اگر ماشین M رشته دلخواهی با شرط $k \leq length(z) < 2k$ را بپذیرد.

نتیجه: فرض کنید Mیک DFA باشد، الگوریتمی برای تعیین هر یک از موارد زیر وجود دارد.

الف) L(M) تهی است.

ب) L(M) محدود است.

ج) L(M) نامحدود است.

اگر k تعداد حالات و j تعداد الفبای automata باشد. $j^{k-1}+1$ رشته به طول کمتر از k وجود دارد.

خواص بستارهای زبانهای باقاعده (closure property)

یک زبان روی الفبای $\sum$ با قاعده است، اگر یکی از موارد زیر را بتوان انجام داد:
الف) مجموعه (عبارت) باقاعده ای روی$\sum$ برای توصیف زبان به دست آورد.
ب) این زبان توسط NFA,DFA یا NFA-λ (FA) بتواند پذیرفته شود.
ج) این زبان بتواند توسط یک گرامر با قاعده تولید شود.

مجموعه زبانهای باقاعده تحت چه عملگرهایی بسته هستند.

قضیه ۱: اگر L_1 و L_2 دو زبان با قاعده باشند، آنگاه زبانهای L_1L_2 , L_1UL_2 وL_1^* نیز باقاعده خواهند بود.

قضیه ۲: زبانهای باقاعده نسبت به عمل متمم بسته هستند. اگر L روی الفبای$\sum$ با قاعده باشد، آنگاه $\overline{L}=\sum^*-L$ نیز با قاعده خواهند بود.

قضیه ۳: اگر L_1 و L_2 دو زبان با قاعده باشند، آنگاه زبان $L_1\cap L_2$ باقاعده است زیرا$L_1\cap L_2= \overline{\overline{L_1}U\overline{L_2}}$.

قضیه ۴: فرض کنید که L_1 یک زبان با قاعده و L_2 یک زبان مستقل از متن باشد. زبان $L_1\cap L_2$ لزوماً باقاعده نیست.

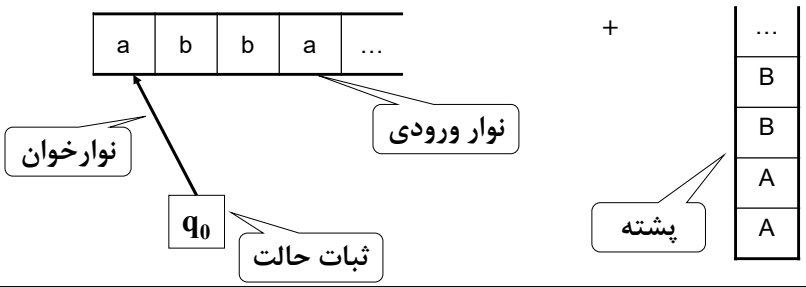
ماشینهای پشته ای PDA

ماشین پشته ای متناهی (Push Down Automata)

تعریف: PDA یک شش تایی $(Q,\sum,\Gamma,\delta,q_0,F)$ است که Q یک مجموعه متناهی از حالات، $\sum$ یک مجموعه متناهی موسوم به الفبای ورودی، Γ یک مجموعه متناهی موسوم به الفبای پشته، q_0 حالت ابتدایی $q_0 \in Q$ ، F یک مجموعه متناهی از حالات پایانی، δ یک تابع گذر به صورت زیر است:

$$\delta: Q\times(\Sigma U\{\lambda\})\times(\Gamma U\{\lambda\})\rightarrow P(Q\times(\Gamma U\{\lambda\}))$$

یک ماشین PDA از چهار قسمت تشکیل می‌شود:



ماشینهای پشته ای PDA

یک ماشین خودکار **pushdown**، یک ماشین با حالات متناهی همراه با یک حافظه اضافی با ساختار پشته است. افزودن حافظه اضافی باعث می گردد قدرت پذیرش زبانها در این ماشین افزایش پیدا کند.

نوار ورودی: از سمت چپ محدود و از سمت راست نامحدود است. ورودی از چپ به راست روی آن نوشته می‌شود. نوار ورودی به خانه‌هایی تقسیم می‌شود که در هر قسمت تنها یک حرف ذخیره می‌شود. قبل از شروع به کار ماشین تمام ورودی روی نوار قرار داده می‌شود.

نوارخوان: تنها قادر است اطلاعات را مرور کند و قادر به انتقال اطلاعات به/از نوار نیست. فقط روی نوار به سمت راست حرکت می‌کند. اجرای هر دستورالعمل، نوارخوان را یک واحد به سمت راست انتقال می‌دهد. هنگام شروع به کار ماشین، نوارخوان روی اولین خانه نوار قرار دارد.

ثبات حالت: مقدار ذخیره شده در آن نشانگر وضعیت ماشین است. تعداد حالتهای یک ماشین پشته‌ای، متناهی است. بر خلاف ماشینهای متناهی، در ماشینهای پشته‌ای ثبات حالت تنها حافظه ماشین نیست.

عناصر الفبای پشته ای را با حروف بزرگ نشان می دهیم.

ماشینهای پشته ای PDA

تابع گذر زیر را درنظر بگیرید:

$[q_j,B] \in \delta(q_i,a,A)$ یا $\delta(q_i, a, A)=\{[q_j, B]\}$

الف) حالت ماشین را از q_i به q_j تغییر می دهد.

ب) عنصر **a** را پردازش می کند(حرکت نوار خوان به سمت راست).

ج) **A** را از بالای پشته حذف می کند(عمل **pop**).

د) **B** را در پشته درج می کند(عمل **push**).

برخی توابع گذر خاص که در این ماشین قابل استفاده است.

1) $[q_i , \lambda] \in \delta(q_j , \lambda, A)$

2) $[q_i , A] \in \delta(q_j , \lambda, \lambda)$

3) $[q_i , \lambda] \in \delta(q_j , a, \lambda)$

در هر لحظه وضعیت ماشین به حالت فعلی، عنصر مقابل نوارخوان و عناصر سمت راست آن و محتویات پشته بستگی دارد. اگر وضعیت ماشین q_i، عناصر پشته α و باقیمانده ورودی (رشته پردازش نشده) **w** باشد (حرف اول **w** مقابل نوارخوان است) وضعیت هر لحظه از محاسبات ماشین را می توان به صورت سه تایی $[q_i,w,\alpha]$ بیان کرد.

ماشینهای پشته ای PDA

نمودار حالت PDA:

۱) گره های گراف عناصر مجموعه Q هستند.

۲) لبه های گراف عناصر مجموعه $\sum$ هستند.

۳) q_0 گره شروع است.

۴) F مجموعه گره های پذیرش است.

۵) به ازای هر دو حالت q_i و q_j لبه‌ای با برچسب a,A|B بین دو گره q_i و q_j وجود دارد اگر و فقط اگر $[q_j,B] \in \delta(q_i, a, A)$.

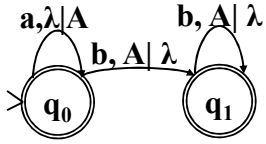
مثال: یک PDA برای پذیرش زبان $L=\{a^i b^j | i \geq 0\}$ طراحی کنید.

$Q=\{q_0,q_1\}$ $\sum=\{a,b\}$ $\Gamma=\{A\}$ $F=\{q_0,q_1\}$

$\delta(q_0,a,\lambda)=\{[q_0,A]\}$

$\delta(q_0,b,A)=\{[q_1,\lambda]\}$

$\delta(q_1,b,A)=\{[q_1,\lambda]\}$



ماشینهای پشته ای PDA

تعریف: فرض کنید که $(Q,\sum,\Gamma,\delta,q_0,F)$ یک PDA باشد. رشته $w \in \sum^*$ توسط ماشین قابل پذیرش است اگر یک محاسبه $[q_0,w,\lambda] \vdash^* [q_i,\lambda,\lambda]$ وجود داشته باشد بطوریکه $q_i \in F$ باشد. زبان ماشین M مجموعه تمامی رشته های پذیرفته شده توسط ماشین است.

۱- همه رشته خوانده شود.

۲- پشته خالی شود.

۳- در حالت پذیرش متوقف شود.

مثال: رشته w=aabb را توسط PDA مثال قبل مورد پردازش قرار دهید.

$[q_0,aabb,\lambda]$

$\vdash [q_0,abb,A]$

$\vdash [q_0,bb,AA]$

$\vdash [q_1,b,A]$

$\vdash [q_1,\lambda,\lambda]$ $q_1 \in F$

$\delta(q_0,a,\lambda)=\{[q_0,A]\}$

$\delta(q_0,b,A)=\{[q_1,\lambda]\}$

$\delta(q_1,b,A)=\{[q_1,\lambda]\}$

ماشینهای پشته ای PDA

مثال: یک PDA برای پذیرش زبان $\{wcw^R | w \epsilon \{a,b\}^*\}$ طراحی کنید.

$Q=\{q_0,q_1\}$ $\sum=\{a,b,c\}$ $\Gamma=\{A,B\}$ $F=\{q_1\}$

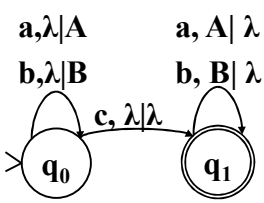
$\delta(q_0,a,\lambda)=\{[q_0,A]\}$

$\delta(q_0,b,\lambda)=\{[q_0,B]\}$

$\delta(q_0,c,\lambda)=\{[q_1,\lambda]\}$

$\delta(q_1,a,A)=\{[q_1,\lambda]\}$

$\delta(q_1,b,B)=\{[q_1,\lambda]\}$



مثال: رشته w=abcba را توسط PDA فوق مورد پردازش قرار دهید.

$[q_0,abcba,\lambda]$

$\vdash [q_0,bcba,A]$

$\vdash [q_0,cba,BA]$

$\vdash [q_1,ba,BA]$

$\vdash [q_1,a,A]$

$\vdash [q_1,\lambda,\lambda]$ $q_1 \epsilon F$

ماشینهای پشته ای PDA

تعریف: یک PDA قطعی (deterministic) است اگر حداکثر یک گذر برای هر ترکیبی از حالت، عنصر ورودی و عنصر بالای پشته وجود داشته باشد. (برای هر ترکیبی به صورت $\delta(q_i, a, A)$ حداکثر یک تابع گذر وجود داشته باشد)

دو تابع گذر $[q_j,C] \epsilon \delta(q_i,U,A)$ و $[q_k,D] \epsilon \delta(q_i,V,B)$ سازگار (compatible) هستند اگر یکی از شرایط زیر وجود داشته باشند:

1) $U=V$, $A=B$ $\delta(q_1,a,A)$, $\delta(q_1,a,A)$

2) $U=V$, ($A=\lambda$ یا $B=\lambda$) $\delta(q_1,a,\lambda)$, $\delta(q_1,a,A)$

3) ($U=\lambda$ یا $V=\lambda$), $A=B$ $\delta(q_1,a,A)$, $\delta(q_1,\lambda,A)$

4) ($U=\lambda$ یا $V=\lambda$), ($A=\lambda$ یا $B=\lambda$) $\delta(q_1,a,\lambda)$, $\delta(q_1,\lambda,A)$

یک PDA قطعی است اگر دارای هیچ دو تابع گذر سازگاری نباشد.

برخلاف ماشین های FA، PDA قطعی و غیرقطعی بایکدیگر معادل نیستند. به عبارت دیگر برای پردازش برخی از زبانها فقط می توان PDAغیرقطعی طراحی کرد. قدرت پردازش PDA غیرقطعی بیشتر از PDA قطعی است.

ماشینهای پشته ای PDA

مثال: یک PDA برای پذیرش زبان $L=\{a^i| i\geq 0\}U\{a^ib^i| i\geq 0\}$ طراحی کنید.

$Q=\{q_0,q_1,q_2\}$ $\sum=\{a,b\}$ $\Gamma=\{A\}$ $F=\{q_1,q_2\}$

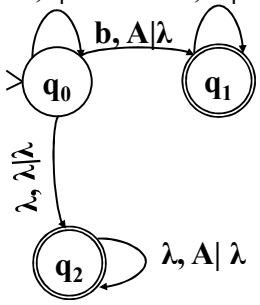
$\delta(q_0,a, \lambda)=\{[q_0,A]\}$

$\delta(q_0,\lambda, \lambda)=\{[q_2, \lambda]\}$

$\delta(q_0,b, A)=\{[q_1, \lambda]\}$

$\delta(q_1,b, A)=\{[q_1, \lambda]\}$

$\delta(q_2,\lambda, A)=\{[q_2, \lambda]\}$



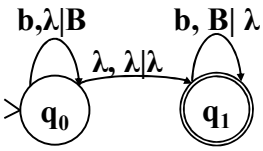
مثال: یک PDA برای پذیرش زبان $L=\{ww^R| w \in \{a,b\}^*\}$ طراحی کنید.

$Q=\{q_0,q_1\}$ $\sum=\{a,b\}$ $\Gamma=\{A,B\}$ $F=\{q_1\}$

$\delta(q_0,a, \lambda)=\{[q_0,A]\}$ $\delta(q_1,a, A)=\{[q_1, \lambda]\}$

$\delta(q_0,b, \lambda)=\{[q_0,B]\}$ $\delta(q_1,b, B)=\{[q_1, \lambda]\}$

$\delta(q_0,\lambda, \lambda)=\{[q_1, \lambda]\}$



ماشینهای پشته ای PDA

انواع PDA

الف) از دیدگاه توابع گذر

در هر تابع گذر در PDA سه عمل زیر انجام می گیرد:

۱-pop ۲-push ۳-پردازش یک عنصر ورودی

تعریف: یک PDA ساده (atomic) است اگر در هر تابع گذر تنها یکی از این سه عمل انجام گیرد.

قضیه: فرض کنید M یک PDA باشد در آنصورت PDA ساده 'M وجود دارد به گونه ای که $L(M')=L(M)$

اثبات: می توان هر تابع گذر به صورت $[q_j,B] \in \delta(q_i,a,A)$ را با در نظر گرفتن حالات جدید به صورت زیر تبدیل کرد.

$\delta(q_i,a, \lambda)=\{[p_1, \lambda]\}$

$\delta(p_1,\lambda, A)=\{[p_2, \lambda]\}$

$\delta(p_2,\lambda, \lambda)=\{[q_j, B]\}$

ماشینهای پشته ای PDA

تعریف: یک PDA توسعه یافته (extended) است اگر در هر تابع گذر بتوان بیش از یک عنصر را درون پشته درج کرد مانند $[q_j,BCD] \in \delta(q_i,a,A)$.

قضیه: فرض کنید M یک PDA توسعه یافته باشد در آنصورت یک M' PDA وجود دارد به گونه ای که $L(M')=L(M)$

اثبات: می توان برای تابع گذری که k عنصر را در پشته درج می کند، با در نظر گرفتن k-1 حالت جدید تابع گذر توسعه یافته را به تابع گذر معمولی تبدیل کرد.

هر تابع گذر به صورت $[q_j,BCD] \in \delta(q_i,a,A)$ را با در نظر گرفتن حالات جدید به صورت زیر تبدیل کرد.

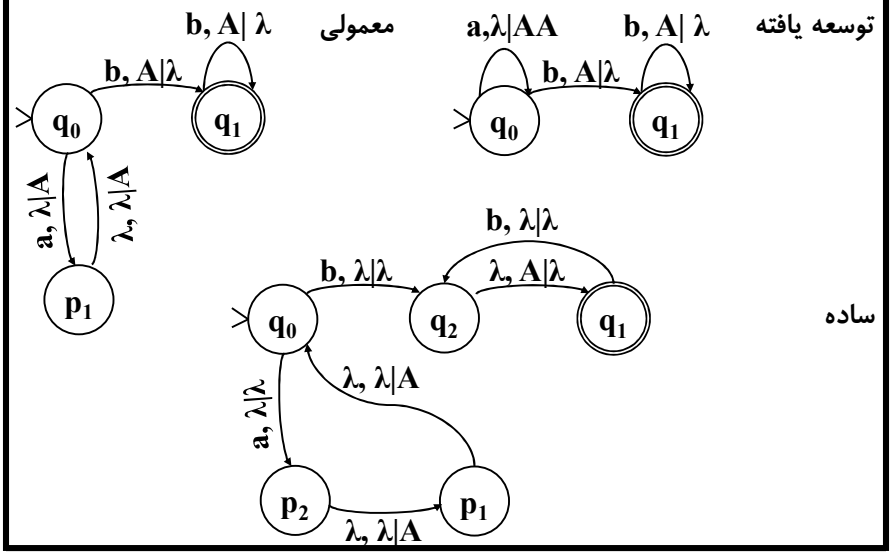
$\delta(q_i,a, A)=\{[p_1, D]\}$

$\delta(p_1,\lambda, \lambda)=\{[p_2, C]\}$

$\delta(p_2,\lambda, \lambda)=\{[q_j, B]\}$

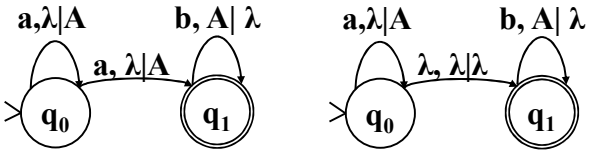
ماشینهای پشته ای PDA

مثال: سه نوع PDA معمولی, ساده و توسعه یافته برای پذیرش زبان $L=\{a^ib^{2i} | i\geq 1\}$ طراحی کنید.



ماشینهای پشته ای PDA

مثال: برای پذیرش زبان های $L=\{a^nb^n \mid n\geq 0\}$ و $L=\{a^nb^n \mid n>0\}$ PDA طراحی کنید.



انواع PDA

ب) از دیدگاه پذیرش رشته

برای پذیرش رشته، به سه شکل می توان PDA را طراحی کرد.

دیدگاه اول: تمامی رشته روی نوار خوانده شده، پشته خالی شده و در حالت پذیرش متوقف گردد.

دیدگاه دوم: تمامی رشته روی نوار خوانده شده و در حالت پذیرش متوقف گردد.

دیدگاه سوم: تمامی رشته روی نوار خوانده شده و، پشته خالی شود.

ماشینهای پشته ای PDA

توضیح در مورد دیدگاه دوم:

پردازشی به صورت $[q_0,w,\lambda] \vdash^* [q_i,\lambda,\alpha]$ که $q_i \in F$ موجود است. زبان پذیرفته شده توسط چنین PDA با نماد $L_F(M)$ نمایش داده می شود.

توضیح د رمورد دیدگاه سوم:

پردازشی به صورت $[q_0,w,\lambda] \vdash^* [q_i,\lambda,\lambda]$ که $q_i \in F$ موجود است. زبان پذیرفته شده توسط چنین PDA با نماد $L_E(M)$ نمایش داده می شود.

قضیه: فرض کنید L زبان پذیرفته شده به وسیله ماشین پشته‌ای $(Q,\sum,\Gamma,\delta,q_0,F)$ باشدکه مطابق دیدگاه دوم رشته ها را می پذیرد. در آنصورت یک ماشین پشته ای معادل با M که بر اساس دیدگاه اول رشته ها را پردازش می کند وجود دارد.

قضیه: فرض کنید L زبان پذیرفته شده به وسیله ماشین پشته‌ای $(Q,\sum,\Gamma,\delta,q_0,F)$ باشدکه مطابق دیدگاه سوم رشته ها را می پذیرد. در آنصورت یک ماشین پشته ای معادل با M که بر اساس دیدگاه اول رشته ها را پردازش می کند وجود دارد.

محدوده PDA زبانهای مستقل از متن است.

ارتباط زبانهای مستقل از متن و ماشین های پشته ای

قضیه: فرض کنید L یک زبان مستقل از متن باشد، یک PDA برای پذیرش زبان L وجود دارد.

از آنجا که L یک زبان مستقل از است توسط یک گرامر مستقل از متن قابل تولید است. این گرامر مستقل از متن به فرم نرمال گریباش تبدیل می شود. سپس از روی فرم نرمال گریباش تولید ماشین PDA شروع می شود.

فرض کنید که $G=(V,\sum,P,S)$ یک گرامر مستقل از متن به فرم نرمال گریباش برای تولید زبان L باشد در آنصورت ماشین پشته ای توسعه یافته طبق روابط زیر قابل ایجاد است.

$\delta(q_0,a, \lambda)=\{[q_1, w]\}$ $\quad S \rightarrow aw \in P$

$\delta(q_1,a, A)=\{[q_1, w]\}$ $\quad A \rightarrow aw \in P$

$\delta(q_0,\lambda, \lambda)=\{[q_1, \lambda]\}$ $\quad S \rightarrow \lambda \in P$

مثال: با استفاده از روش فوق ماشین پشته ای برای پذیرش زبان $L=\{a^ib^i \mid i>0\}$ طراحی کنید.

ارتباط زبانهای مستقل از متن و ماشین های پشته ای

مثال: یک گرامر به فرم نرمال گریباش نوشته، به ازای هر قانون یک δ نوشته شود.

$S \rightarrow aAB \mid aB$

$A \rightarrow aAB \mid aB$

$B \rightarrow b$

	توابع گذر	پردازش رشته w=aaabbb
$S \rightarrow aAB$	$\delta(q_0,a, \lambda)=\{[q_1, AB]\}$	$\vdash [q_0,aaabbb, \lambda]$
$S \rightarrow aB$	$\delta(q_0,a, \lambda)=\{[q_1, B]\}$	$\vdash [q_1,aabbb, AB]$
$A \rightarrow aAB$	$\delta(q_1,a, A)=\{[q_1, AB]\}$	$\vdash [q_1,abbb, ABB]$
$A \rightarrow aB$	$\delta(q_1,a, A)=\{[q_1, B]\}$	$\vdash [q_1,bbb, BBB]$
$B \rightarrow b$	$\delta(q_1,b, B)=\{[q_1, \lambda]\}$	$\vdash [q_1,bb, BB]$
		$\vdash [q_1,b, B]$
		$\vdash [q_1, \lambda, \lambda]$

اصل pumping برای زبانهای مستقل از متن

قضیه: فرض کنید که G یک گرامر مستقل از متن در فرم نرمال شومسکی باشد و $S \overset{*}{\Rightarrow} w$ $w \in \sum^*$ یک اشتقاق از با درخت اشتقاق T باشد. اگر عمق T برابر n باشد، آنگاه $length(w) \le 2^{n-1}$ خواهد بود.

اگر $length(w) \le 2^n$ باشد، آنگاه عمق درخت اشتقاق حداقل برابر $n+1$ خواهد بود.

قضیه Pumping:

فرض کنید که L یک زبان مستقل از متن باشد. در آنصورت یک عدد مانند k (وابسته به زبان L) وجود دارد به گونه ای که هر رشته $z \in L$ با $length(z) \ge k$ می توان رشته z را به صورت $z=uvwxy$ نوشت که دارای شرایط زیر باشد.

۱) $length(vwx) \le k$

۲) $length(v)+length(x)>0$

۳) برای $i \ge 0$، $uv^iwx^iy \in L$

این قضیه ابزار قدرتمندی برای اثبات مستقل از متن نبودن یک زبان است.

اصل pumping برای زبانهای مستقل از متن

مثال: نشان دهید $L=\{w \in a^* | length(w) \text{ is prime}\}$ مستقل از متن نیست.

فرض می شود L مستقل از متن و n عدد اولی بزرگتر از k باشد. در آنصورت باید رشته $z=a^n$ بنابر شرایط قضیه pumping قابل تقسیم باشد. $z= uvwxy$

$length(uv^iwx^iy)=?$

$length(u)+length(w)+length(y)=m$

$length(v^ix^i)=i\ length(vx)=i(length(z)-length(u+w+y))=i(n-m)$

$length(uv^iwx^iy)=m+i(n-m)$

if $i=m$

$length(uv^mwx^my)=m+m(n-m)$

$=m(1+n-m)$

≥ 2

بنابراین طول رشته اول نبوده و به L تعلق ندارد. در نتیجه L مستقل از متن نیست.

خواص بستاری زبانهای مستقل از متن (closure property)

قضیه: مجموعه زبانهای مستقل از متن تحت عملگرهای U (اجتماع)، الحاق و * (kleen star) بسته هستند.

قضیه: مجموعه زبانهای مستقل از متن تحت عملگرهای $\cap$ و $^-$ بسته نیستند.

مثال: دو زبان $L_1=\{a^ib^ic^j|i,j\geq 0\}$ و $L_2=\{a^ib^jc^j|i,j\geq 0\}$ مستقل از متن هستند زیرا توسط گرامرهای زیر تولید می شوند.

G_1: $S\rightarrow AC$
$A\rightarrow aAb|\lambda$
$C\rightarrow cC|\lambda$

G_2: $S\rightarrow BC$
$B\rightarrow aB|\lambda$
$C\rightarrow bCc|\lambda$

ولی اشتراک این دو زبان مستقل از متن نیست.

$L_1\cap L_2=\overline{\overline{L_1}\cup\overline{L_2}}$

قضیه: فرض کنید که R یک زبان با قاعده و L یک زبان مستقل از متن باشد. در این صورت $R\cap L$ مستقل از متن است.

ماشینهای پشته ای PDA

ماشین های دو پشته ای ماشین (Two stack Push Down Automata)

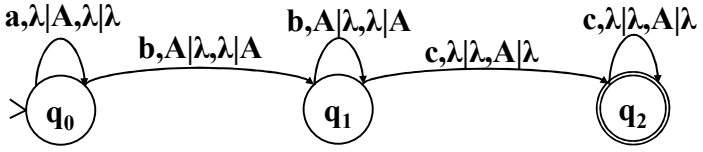
تعریف: ماشین دو پشته ای یک شش تایی $(Q,\sum,\Gamma,\delta,q_0,F)$ است Q، $\sum$، Γ، q_0 و F مانند ماشین های تک پشته ای است و δ به صورت زیر است:

$\delta: Q\times(\Sigma\cup\{\lambda\})\times(\Gamma\cup\{\lambda\})\times(\Gamma\cup\{\lambda\})\rightarrow P(Q\times(\Gamma\cup\{\lambda\})\times(\Gamma\cup\{\lambda\}))$

دو پشته مستقل از هم عمل می کنند.

ماشین دو پشته ای محدود به زبان مستقل از متن نیست.

مثال: یک ماشین دوپشته ای برای پذیرش زبان $L=\{a^ib^ic^i \mid i>0\}$ طراحی کنید.



مثال: یک ماشین دوپشته ای برای پذیرش زبان $L=\{a^ib^ic^id^i \mid i>0\}$ طراحی کنید.

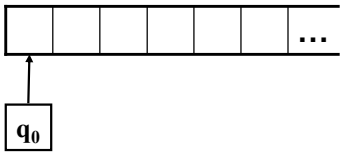
ماشین های تورینگ Turing Machines

ماشین تورینگ مدلی برای طراحی و توسعه کامپیوترهای مدرن امروزی است تنها تفاوت آن در این است که از لحاظ زمان و حافظه بدون محدودیت می باشد. ماشین تورینگ یک ماشین با حالات متناهی است که هر گذر آن عنصری را روی نوار چاپ می کند.

ماشین تورینگ استاندارد:

ماشین تورینگ یک پنج تایی $(Q,\sum,\Gamma,\delta,q_0)$ است که در آن Q مجموعه متناهی از حالات، Γ مجموعه متناهی موسوم به الفبای نوار شامل یک عنصر ویژه B که معرف blank است، $\sum$ یک مجموعه از $\Gamma-\{B\}$ به نام الفبای ورودی، δ توابع گذر $Q\times\Gamma \rightarrow Q\times\Gamma\times(L,R)$ و $q_0 \epsilon Q$ حالت شروع ماشین می باشد.

نوار ماشین تورینگ از یک سمت نامحدود است. خانه های نوار با اعداد طبیعی از چپ به راست شماره گذاری می شوند.



ماشین های تورینگ Turing Machines

زبان DFA: باقاعده، زبان PDA مستقل از متن، زبان ماشین تورینگ شمارش پذیر بازگشتی REL است.

یک تابع گذر شامل سه عمل است:

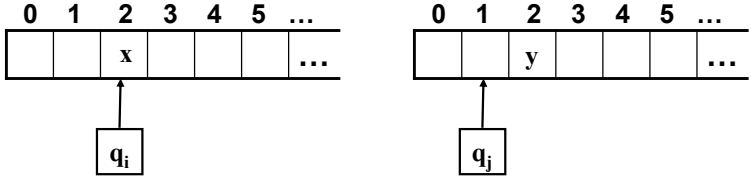
۱- تغییر حالت

۲- نوشتن یک عنصر روی نوار توسط نوارخوان

۳- حرکت نوارخوان

جهت حرکت توسط آخرین جزء گذر تعیین می شود.

این گذر حالت ماشین را از q_i به q_j تغییر داده، عنصر x نوار را با y جایگزین نموده و هد نوارخوان را یک مکان به چپ حرکت می دهد. $\delta(q_i,x)=\{[q_j,y,L]\}$



ماشین های تورینگ Turing Machines

یک ماشین تورینگ هنگامی متوقف (halt) می شود که برای زوج مرتب حالت جاری و عنصر ورودی، هیچ گذری تعریف نشده باشد. یک گذر ممکن است بخواهد از مکان صفر نوار یک حرکت به چپ محدوده نوار انجام دهد که در این صورت متوقف می شود و به این نوع توقف، توقف غیر عادی گویند. هرگاه گفته شود یک محاسبه متوقف می شود، منظور توقف در شرایط عادی است.

در ماشین تورینگ استاندارد باید توقف به گونه ای طراحی شود که در مکان صفر نوار متوقف گردد.

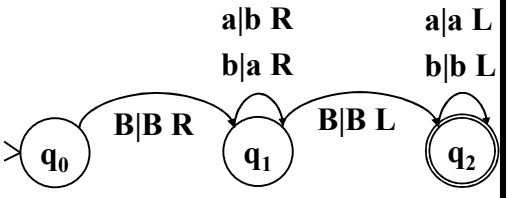
مثال: ماشین تورینگ استانداردی را روی الفبای $\sum = \{a,b\}$ طراحی کنید که در این ماشین تورینگ نمادهای a و b با یکدیگر تعویض شوند.

δ	B	a	b
q_0	q_1,B,R		
q_1	q_2,B,L	q_1,b,R	q_1,a,R
q_2		q_2,a,L	q_2,b,L

ماشین های تورینگ Turing Machines

نمودار حالت ماشین تورینگ فوق به صورت زیر است.

δ	B	a	b
q_0	q_1,B,R		
q_1	q_2,B,L	q_1,b,R	q_1,a,R
q_2		q_2,a,L	q_2,b,L



پردازش رشته w=abab توسط ماشین فوق

- q_0BababB
- B$q_1$ababB
- Bb$q_1$babB
- Bba$q_1$abB
- Bbabq_1bB
- Bbabaq_1B
- Bbabq_2aB
- Bba$q_2$baB
- Bb$q_2$abaB
- B$q_2$babaB
- q_2BbabaB

ماشین های تورینگ Turing Machines

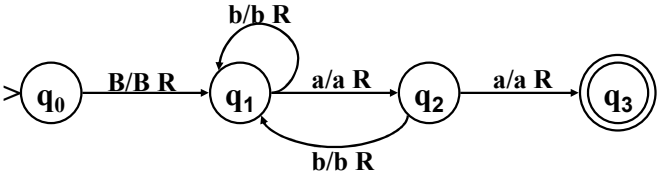
تعریف: فرض کنید که $(Q,\sum,\Gamma,\delta,q_0,F)$ یک ماشین تورینگ باشد. یک رشته $w \in \sum^*$ توسط حالت پذیرش (final state) پذیرفته می شود اگر محاسبه M با ورودی w در یک حالت پذیرش متوقف شود. محاسبه ای که به طور غیر عادی متوقف می شود رشته ورودی را بدون توجه به حالت پذیرش ماشین رد می کند. زبان M، که با L(M) نشان داده می شود، مجموعه تمامی رشته های پذیرفته شده توسط M است.

یک زبان پذیرفته شده توسط یک ماشین تورینگ به زبان شمارش پذیر بازگشتی (recursively enumerable language) موسوم است.

مثال: ماشین تورینگی طراحی کنید که زبان $(aUb)^*aa(aUb)^*$ را بپذیرد.
در ماشین تورینگ برای پذیرش یک رشته لزومی به خواندن تمام رشته نیست.

$q_0BaabbB$
Bq_1aabbB
Baq_2abbB
$Baaq_3bbB$

$q_3 \in F$

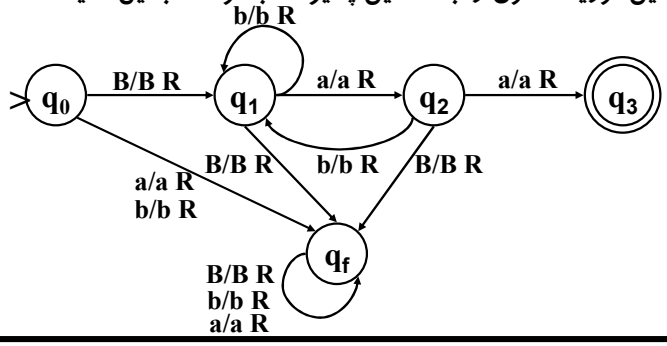


ماشین های تورینگ Turing Machines

تعریف: فرض کنید که $(Q,\sum,\Gamma,\delta,q_0,F)$ یک ماشین تورینگ با پذیرش توسط توقف باشد. یک رشته $w \in \sum^*$ توسط توقف پذیرفته می شود اگر محاسبه M با ورودی w به طور عادی متوقف شود.

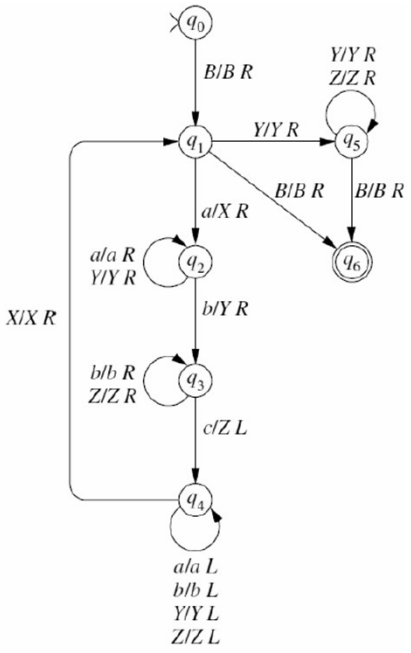
دو جمله زیر معادل یکدیگر هستند.
الف)زبان L توسط یک ماشین تورینگ با حالت پذیرش پذیرفته می شود.
ب) زبان L توسط یک ماشین تورینگ با توقف پذیرفته می شود.

مثال: ماشین تورینگ فوق را به ماشین پذیرنده با توقف تبدیل کنید.



ماشین های تورینگ Turing Machines

مثال: ماشین تورینگی برای پذیرش زبان $L=\{a^i b^i c^i \mid i \geq 0\}$ طراحی کنید.



ماشین های تورینگ Turing Machines

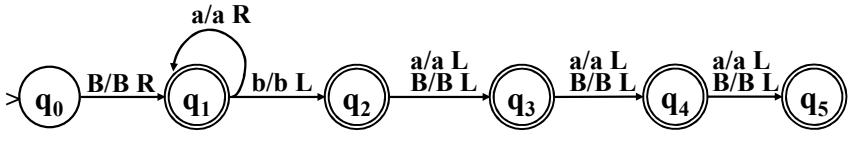
ماشین تورینگ با نوار دوطرفه (Two-Way Tape):

ماشین تورینگ با نوار دو طرفه مشابه ماشین تورینگ استاندارد است با این تفاوت که نوار آن از هر دو طرف تا بینهایت ادامه می یابد.

رشته ورودی می تواند هر جایی روی نوار قرار گیرد و برای تعیین محدوده چپ نوار از نماد (#) استفاده می شود که فرض می شود این عنصر (#) از عناصر الفبای ماشین نباشد.

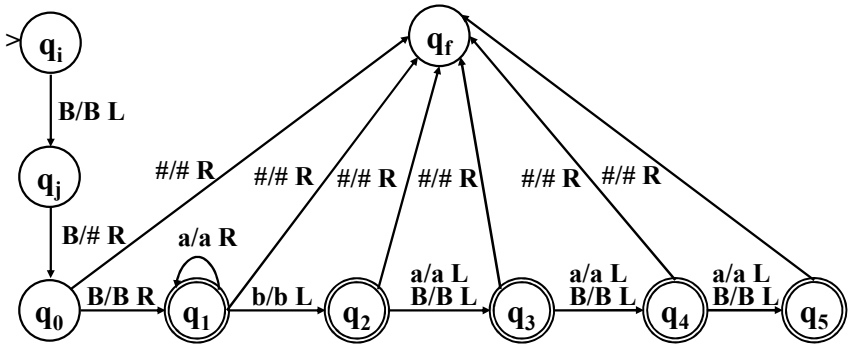
قضیه: زبان L توسط ماشین تورینگ با یک نوار دوطرفه پذیرفته می شود اگر و تنها اگر توسط یک ماشین تورینگ استاندارد پذیرفته می شود.

مثال: ماشین تورینگی طراحی کنیدکه رشته هایی روی {a,b} را بپذیرد که در آن قبل از اولین b، در صورت وجود، حداقل سه a وجود داشته باشد.



ماشین تورینگ با نوار دوطرفه (Two-Way Tape):

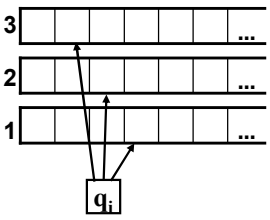
مثال: ماشین تورینگ با نوار دو طرفه طراحی کنیدکه رشته هایی روی {a,b} را بپذیرد که در آن قبل از اولین b، در صورت وجود، حداقل سه a وجود داشته باشد.



ماشین تورینگ چند نواره (Multitape Machines):

در این ماشین تورینگ چند نوار که می توانند به صورت مستقل پردازش شوند وجود دارد. هر نوار برای خود دارای یک نوارخوان مستقل است و همگی این نوارخوان ها به یک ثبات متصل هستند. برای نمونه ساختار ماشین تورینگ سه نواره به صورت زیر است.

هر تابع گذر در این ماشین عملیات زیر را انجام می دهد:



۱- تغییر وضعیت ماشین

۲- نوشتن نماد روی هر یک از نوارها

۳- تغییر مکان نوارخوان ها به صورت مستقل از یکدیگر

برای نمونه تابع گذر ماشین تورینگ دو نواره در حالی که x روی نوار اول و y روی نوار دوم است به صورت زیر می باشد:

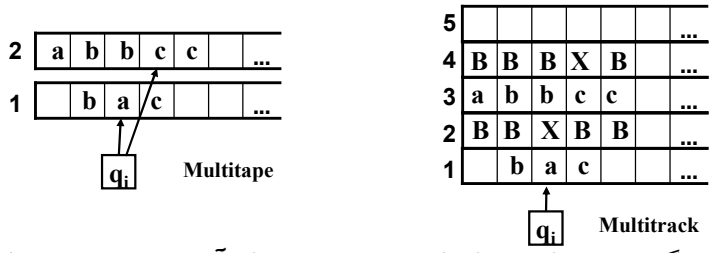
$$\delta(q_i,x,y)=\{[q_j;z,d_1;w,d_2]\} \qquad d_1,d_2 \in \{L,R,S\}$$

L−> left R −> right S −> stationary

ماشین های تورینگ Turing Machines

ماشین تورینگ چند نواره (Multitape Machines):

هر ماشین تورینگ k نواره (k-tape) را می توان با یک ماشین تورینگ 2k+1 شیار (2k+1 track) شبیه سازی کرد.



در ماشین تورینگ چند شیاره، شیارهای فرد به جز شیار آخر به ترتیب محتوای نوارها و در شیارهای زوج به ترتیب مکان اشاره نوارخوان هر نوار با در نظرگیری نمادی مانند X نشان داده می شود. شیار آخر نیز برای شبیه سازی محاسبات ماشین است.

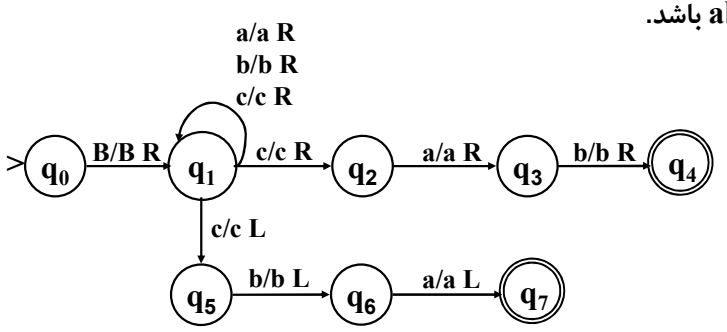
قضیه: زبان L توسط ماشین تورینگ چند نواره پذیرفته می شود اگر و فقط اگر توسط ماشین تورینگ استاندارد پذیرفته شود.

ماشین های تورینگ Turing Machines

ماشین تورینگ غیرقطعی (Nondeterministic Turing Machines):

یک ماشین تورینگ غیر قطعی ممکن است یک تعداد متناهی از گذرها را برای یک وضعیت مشخص تعیین کند. اجزا یک ماشین غیر قطعی، به جز تابع گذر، مشابه اجزا ماشین تورینگ استاندارد است. تابع گذر در یک ماشین غیر قطعی به وسیله یک تابع جزئی از $Q\times\Gamma$ به زیر مجموعه هایی از $Q\times\Gamma\times\{L,R\}$ تعریف می شود.

مثال: ماشین غیر قطعی زیر رشته هایی را می پذیرد که در آن یک c قبل یا بعد از ab باشد.



طبقه بندی شومسکی The Chomsky Hierarchy

گرامرهای بدون محدودیت بزرگترین گروه از گرامرهای عبارت می باشند. یک قانون $u \rightarrow v$ مشخص می کند که یک رخداد از زیررشته u در یک رشته ممکن است با رشته v جایگزین شود. یک اشتقاق دنباله ای از جایگزینی های مجاز است. تنها محدودیتی که روی یک قانون از یک گرامر اعمال می شود این است که سمت چپ آن نباید تهی باشد. به این سیستمهای بازنویسی (rewriting) رشته ها، گرامرهای نوع صفر نیز گفته می شود.

یک گرامر بدون محدودیت یک گرامر چهارتایی $(V,\sum,P,S)$ است که V یک مجموعه متناهی از متغیرها، $\sum$ (الفبا) یک مجموعه متناهی از عناصر پایانی، P یک مجموعه از قوانین و S یک عنصر مشخص در Vمی باشد. یک قانون از یک گرامر بدون محدودیت به شکل $u \rightarrow v$ است که در آن $u \in (VU\sum)^+$ و $v \in (VU\sum)^*$ می باشد. فرض می شود که مجموعه های Vو $\sum$ جدا از هم هستند.

گرامرهای باقاعده و مستقل از متن زیر مجموعه هایی از گرامرهای بدون محدودیت هستند. یک گرامر مستقل ازمتن گرامری است که در سمت چپ هر قانون آن تنها یک متغیر وجود دارد. قوانین یک گرامر باقاعده:

$A \rightarrow aB, A \rightarrow a, A \rightarrow \lambda$ $\quad A,B \in V, \quad a \in \sum$

طبقه بندی شومسکی The Chomsky Hierarchy

گرامر بدون محدودیت زیر با سرترم S زبان $L=\{a^i b^i c^i \mid i \geq 0\}$ را تولید می کند.

$V=\{S,A,C\}$

$\sum=\{a,b,c\}$

$S \rightarrow aAbc|\lambda$

$A \rightarrow aAbC|\lambda$

$Cb \rightarrow bC$

$Cc \rightarrow cc$

قضیه: فرض کنید که $G=(V,\sum,P,S)$ یک گرامر بدون محدودیت باشد. در این صورت $L(G)$یک زبان شمارش پذیر بازگشتی است.

قضیه: فرض کنید که Lیک زبان شمارش پذیر بازگشتی باشد. در این صورت یک گرامر بدون محدودیت G با $L(G)=L$ وجود دارد.

قضیه: مجموعه زبانهای شمارش پذیر بازگشتی نسبت به عمل اجتماع، الحاق و عملگر ستاره (kleen star) بسته است.

طبقه بندی شومسکی The Chomsky Hierarchy

گرامرهای وابسته به متن یک مرحله میانی بین گرامرهای مستقل از متن و گرامرهای بدون محدودیت می باشند. در این گرامرها، هیچ محدودیتی برای سمت چپ یک قانون وجود ندارد، اما طول سمت راست آن بایستی حداقل به اندازه طول سمت چپ باشد.

یک گرامر عبارت وابسته به متن است اگر هر قانون به شکل $u \rightarrow v$ باشد که $u \in (VU\sum)^+$ و $v \in (VU\sum)^*$ و $length(u) \leq length(v)$ است.

به قانونی که در شرایط فوق صدق کند یکنواخت (monotonic)گفته می شود.

قضیه: هر زبان وابسته به متن بازگشتی است.

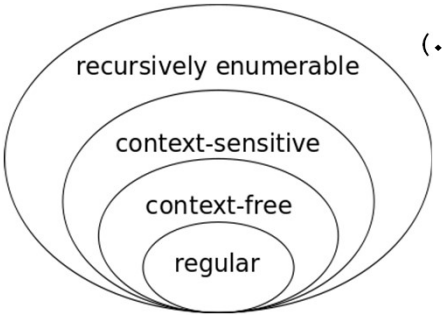
آتاماتای محدود خطی (Linear Bounded Automata)

LBA ماشین تورینگی است که مقدار نوار در دسترس با طول رشته ورودی مشخص می شود. رشته ورودی دو نماد < و > را برای مشخص کردن مرز سمت راست و چپ نوار دارد.

آتاماتای خطی محدود یک ساختار $M=(Q,\sum,\Gamma,\delta,q_0,<,>,F)$ است که مشابه ماشین تورینگ غیر قطعی می باشند و عناصر > و < عناصری مشخص از Γ هستند.

طبقه بندی شومسکی The Chomsky Hierarchy

طبقه بندی شومسکی شامل چهار گروه از گرامرها(زبانها) است:



۱- گرامرهای بدون محدودیت (نوع ۰)

۲- گرامرهای وابسته به متن (نوع ۱)

۳- گرامرهای مستقل از متن (نوع ۲)

۴- گرامرهای باقاعده (نوع ۳)

Grammars	Languages	Accepting Machines
Type 0: Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1: Context-sensitive grammar	Context-sensitive language	Linear-bounded automata
Type 2: Context-free grammar	Context-free language	Pushdown automata
Type 3: Regular grammar	Regular language	Finite state automata

تصمیم پذیری Decideability

یک مسئله تصمیم‌گیری شامل مجموعه ای از سوالاتی است که جواب آنها *بله* یا خیر باشد. یک راه حل برای یک مسئله تصمیم‌گیری یک روال کاراست که جواب را برای هر یک از سوالات مجموعه تعیین می‌کند.

یک مسئله تصمیم‌گیری تصمیم ناپذیر است، اگر هیچ الگوریتمی که مساله را حل کند وجود نداشته باشد.

قابلیت ماشین های تولیدی در انجام واکنشهای قطعی و غیر قطعی موجب می شود که بتوان آنها را به عنوان سیستم های ریاضی مناسبی جهت ایجاد راه حل هایی برای مسائل تصمیم‌گیری در نظر گرفت.

تز چرچ *تورینگ* بیان می کند که برای حل هر مسئله تصمیم گیری که توسط یک روال کارا قابل حل است، می‌توان یک ماشین تورینگ را طراحی نمود. در نتیجه برای اینکه ثابت کنیم که یک مسئله حل ناپذیر است کافیست نشان دهیم که هیچ راه حل ماشین تورینگ برای آن وجود ندارد.

تصمیم پذیری Decideability

مسایل تصمیم گیری:

سوال "آیا ۸ مربع کامل است؟" یک مثال از نمونه سوالی است که در یک مسئله تصمیم‌گیری وجود دارد.

مسئله تصمیم‌گیری معمولا شامل یک تعداد نامتناهی از سوالات مرتبط به هم است.

برای مثال، مسئله ای که تعیین می‌کند یک عدد طبیعی دلخواه مربع کامل است یا خیر شامل سوالات زیر می باشد:

۱- آیا ۰ مربع کامل است؟

۲- آیا ۱ مربع کامل است؟

۳- آیا ۲مربع کامل است؟

...

تصمیم پذیری Decideability

مسایل تصمیم گیری:

یک راه حل برای مسئله تصمیم‌گیری **P** الگوریتمی است که جواب مناسبی را برای هر سوال $p \in P$ تعیین می کند. از آنجایی که راه حل یک مسئله تصمیم‌گیری یک الگوریتم است، لذا مروری بر محاسبات الگوریتمی مفید خواهد بود. اما تا به حال تعریفی از الگوریتم ارائه ندادیم و احتمالا نمی توانیم آن را به طور دقیق تعریف کنیم. در واقع این مفهوم از جمله مفاهیمی که درباره آنها می‌توان گفت: " من قادر به توصیف آن نیستم ولی به محض مشاهده آن را می‌شناسم".

می‌توانیم به برخی از ویژگی های اصلی مفهوم الگوریتم اشاره کنیم:

کامل: برای هر سوال موجود در حیطه مسئله جواب مثبت یا منفی داده می‌شود.

مکانیکی: شامل یک دنباله متناهی از دستورات است که هر یک از آنها را می‌توان بدون نیاز به آگاهی، ابتکار یا کارهای تخمینی اجرا نمود.

قطعی: همواره برای ورودی‌های یکسان جواب های مشابهی تولید خواهد شد.

اغلب به روالی که دارای ویژگی های فوق باشد کارا (effective) گفته می‌شود.

تصمیم پذیری Decideability

مسایل تصمیم گیری:

محاسبات یک ماشین تورینگ استاندارد به وضوح مکانیکی و قطعات ماشین تورینگی که برای هر رشته ورودی متوقف می‌شود نیز کامل است. به دنبال کارایی محاسبات ماشینهای تورینگ می توان از آنها به عنوان یک چهارچوب مناسب برای ایجاد راه حل های مسائل تصمیم‌گیری استفاده کرد. به یک مسئله قطعاً پاسخ داده می شود اگر ورودی توسط یک ماشین تورینگ پذیرفته شود و یک مسئله قطعاً بی‌جواب ماند اگر ورودی توسط هیچ ماشین تورینگی پذیرفته نشود.

برای کامل بودن یک محاسبه کارا لازم است که محاسبه ماشین برای هر رشته ورودی متوقف شود. بنابراین زبان پذیرفته شده توسط ماشین تورینگی که یک مسئله تصمیم‌گیری را حل می‌کند بازگشتی است. برعکس، هر ماشین تورینگ قطعی **M** که یک زبان بازگشتی را می‌پذیرد، می‌تواند به عنوان یک راه حل برای یک مسئله تصمیم‌گیری در نظر گرفته شود. ماشین **M** مسئله شامل دنباله هایی به شکل " آیا رشته **w** در **L(M)** است؟" را برای هر رشته $w \in \sum^*$ حل می‌کند. با استفاده از دوگانگی بین مسائل تصمیم‌گیری حل پذیر و زبان های بازگشتی می توان روش های موجود برای ارائه تصمیم پذیری یک مسئله تصمیم گیری را بسط داد.

تصمیم پذیری Decideability

تز چرچ-تورینگ (Church-Turing Thesis):

ماشینهای انتزاعی برای ارائه یک فرمول از محاسبات الگوریتمی و ماشین های تورینگ، جهت پذیرش زبانها و حل مسائل تصمیم‌گیری معرفی شده بودند. این محاسبات به بازگرداندن جواب بله یا خیر محدود شده اند.

تز چرچ-تورینگ ادعا می کند که هر مسئله تصمیم‌گیری حل پذیر را می توان به یک مسئله ماشین تورینگ معادل تبدیل نمود. با تعریف نتیجه یک محاسبه توسط عناصر موجود روی نوار در لحظه توقف می توان از ماشین های تورینگ برای محاسبه توابع نیز استفاده نمود.

یک راه حل برای یک مسئله تصمیم گیری به محاسبه ای نیاز دارد که یک جواب را برای هر نمونه مسئله بازگرداند. با کمی اغماض نسبت به این محدودیت، به یک راه‌حل جزئی برای مسئله دست می‌یابیم. یک راه حل جزئی برای یک مسئله تصمیم‌گیری p لزوما یک روال کاملاً کارا نیست که یک واکنش قطعی برای هر مسئله $p \in P$ نشان دهد که جواب آن بله باشد. اگر جواب p منفی باشد، یعنی اینکه راه حل به جواب خیر رسیده و یا اینکه در تولید جواب با شکست مواجه شده است.

تصمیم پذیری Decideability

تز چرچ-تورینگ (Church-Turing Thesis):

تز چرچ تورینگ برای مسائل تصمیم‌گیری: **یک روال کارا برای یک مسئله تصمیم‌گیری وجود دارد اگر و تنها اگر ماشین تورینگی وجود داشته باشد که برای تمام رشته های ورودی متوقف شده و مسئله را حل نماید.**

تز چرچ تورینگ توسعه یافته برای مسائل تصمیم‌گیری: **یک مسئله تصمیم‌گیری P حل پذیر جزئی است اگر و تنها اگر ماشین تورینگی وجود داشته باشد که دقیقاً اعضایی از P که جواب آنها بله است را بپذیرد.**

تز چرچ تورینگ ادعایی ندارد که هیچ سیستم دیگری محاسبات الگوریتمی را انجام نمی‌دهد بلکه ادعای او این است که یک محاسبه انجام شده در چنین سیستمی را می توان توسط یک ماشین تورینگ با طراحی مناسب اجرا نمود. شاید بزرگترین حامی تز چرچ تورینگ این تفکر است که از این پس تمامی روال های کارهای شناخته شده می توانند به ماشین‌های تورینگ معادل تبدیل شوند.

تصمیم پذیری Decideability

تز چرچ-تورینگ (Church-Turing Thesis):

قدرت ماشین های تورینگ استاندارد حامی دیگری برای تز چرچ تورینگ می‌باشد. افزودن شیار های چندگانه، نوار های چندگانه و محاسبه غیرقطعی مجموعه زبان‌های قابل تشخیص را افزایش نمی دهد. در فصل ماشین های تورینگ نشان داده شد که زبانهای بازگشتی شمارش پذیر دقیقاً توسط گرامرهای بدون محدودیت تولید می‌شود.

تز چرچ تورینگ ادعا می‌کند که یک مسئله تصمیم‌گیری P یک راه حل دارد اگر و تنها اگر ماشین تورینگی وجود داشته باشد که جوابی را برای هر $p \in P$ تعیین کند. اگر چنین ماشینی وجود نداشته باشد به مسئله تصمیم ناپذیر گفته می‌شود. وجود یک ماشین‌های تورینگ برای یک مسئله تصمیم گیری کاملاً به طبیعت مسئله بستگی دارد نه به میزان حافظه قابل دسترس یا زمان پردازنده مرکزی.

جامعیت محاسبات ماشین تورینگ نیز نتایجی برای تصمیم ناپذیری به دنبال دارد. اگر یک مسئله نتواند به وسیله یک ماشین تورینگ حل شود به وضوح نمی تواند توسط یک ماشین با منابع محدود حل شود.

تصمیم پذیری Decideability

ماشین تورینگ جهانی (Universal Turing Machine):

ماشین های تورینگی که تا کنون از آنها سخن گفته ایم همگی کارهای خاصی را انجام می دهند و برنامه آنها چنان نوشته شده است که یک محاسبه معین را انجام دهند یا یک زبان خاص را شناسایی کنند. اما می توان یک ماشین تورینگ جهانی ساخت که بتواند هر ماشین تورینگ دیگری را شبیه سازی کند. این ماشین که آن را با UTM نشان می دهند. در واقع مدل نظری کامپیوترهای قابل برنامه ریزی امروزی است. در کامپیوترهای امروزی ما می توانیم با افزودن نرم افزارهای جدید به کامپیوتر کاری کنیم که کامپیوتر اولیه مان بتواند کارهای تازه تری انجام دهد. با نصب نرم افزارهای Mathematica کاری می کنیم که کامپیوترساده اولیه توانایی محاسبه ریاضی و تحلیلی پیدا کند، یا با نصب یک نرم افزاردیگر این توانایی را به آن می دهیم که نقش یک کتاب لغت یا دایره المعارف گویا را نیز بازی کند یا این که بازی شطرنج را به نحو استادانه ای بازی کند و هزاران کار متنوع دیگر. همه اینها نشان دهنده این هستند که لپ تاپ یک ماشین تورینگ جهانی است یعنی می تواند ماشین های تورینگ دیگر را شبیه سازی کند. البته یک لپ تاپ به ماشین تورینگ نزدیک است ولی یک ماشین تورینگ ایده آل نیست زیرا به هرحال حافظه محدودی در دسترس دارد و طول نوارش محدود (اگر چه بسیار بزرگ) است.

تصمیم پذیری Decideability

ماشین تورینگ جهانی (Universal Turing Machine):

از لحاظ نظری یک ماشین جهانی تورینگ چگونه ساخته می شود؟

ماشین های تورینگ یک مجموعه شمارش پذیر را تشکیل می دهند. دلیل این امر آن است که یک ماشین تورینگ با تعداد حالت هایش یعنی اندازه مجموعه Q، حروف الفبا، و تعداد دستورها مشخص می شود. هر دستور نیز چیزی نیست جز یک نگاشت از یک مجموعه محدود به یک مجموعه محدود دیگر. شمارش پذیر بودن مجموعه تمام ماشین های تورینگ این امکان را می دهد که به هرماشین یک عدد طبیعی نسبت دهیم. این عدد، عدد تورینگ (Turing Number) آن ماشین خوانده می شود. بنابراین می توانیم از ماشین های تورینگ $M_1,M_2,\cdots,M_K,\ldots$ نام ببریم. برای آنکه ماشین جهانی تورینگ را بسازیم کافی است که علاوه بر رشته ورودی $x = x_1x_2\cdots x_n$، برنامه مورد نظر(یعنی شماره ماشین تورینگ مورد نظر) را به صورت یک رشته به ماشین UTM بدهیم. بنابراین هرگاه خروجی ماشین M_i روی ورودی x را با $M_i(x)$ نشان دهیم، ماشین جهانی کار زیر را انجام می دهد:

UTM : (i b x) → (i b M_i(x))
UTM : (j b x) → (j b M_j(x))
UTM : (k b x) → (k b M_k(x))

تصمیم پذیری Decideability

مساله توقف (Halting problem) برای ماشین های تورینگ:

مشهور ترین مسائل تصمیم‌گیری مربوط به ویژگی های ماشین تورینگ است. مسئله توقف ممکن است به صورت زیر مطرح شود: آیا محاسبه ماشین تورینگ دلخواه M با الفبای ورودی $\sum$ برای رشته ورودی $w\in\sum^*$ متوقف می‌شود؟ هیچ الگوریتمی وجود ندارد که مسئله توقف را حل کند. تصمیم ناپذیری در مسئله توقف از مهمترین نتایج نظریه علوم کامپیوتر است.

برای پاسخ به مسئله هیلبرت در مورد وجود یک الگوریتم که بتواند همه مسائل ریاضیات را حل کند، تورینگ این پرسش را طرح کرد که آیا یک الگوریتم وجود دارد که بتواند بگوید یک ماشین تورینگ M_x روی یک ورودی y توقف می کند یا نه؟ برای پاسخ به این سوال تورینگ پرسش حتی ساده تری را پیش نهاد و آن اینکه آیا یک الگوریتم وجود دارد که بتواند بگوید یک ماشین تورینگ M_x روی یک ورودی x توقف می کند یا نه؟ یادآوری می کنیم که در این جا هر ماشین تورینگ را با شماره اش نشان داده ایم و از شمارش پذیربودن ماشین های تورینگ استفاده کرده ایم.

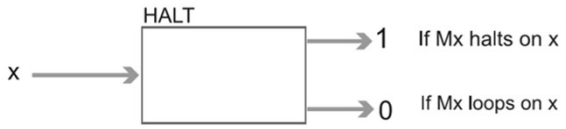
تصمیم پذیری Decideability

مساله توقف (Halting problem) برای ماشین های تورینگ:

اگر چنین الگوریتمی وجود داشته باشد معنایش این است که تابع زیر یک تابع قابل محاسبه است:

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

معنای محاسبه پذیر بودن این است که یک ماشین تورینگ (الگوریتم) وجود دارد که می تواند این تابع را حساب کند و برای هر ورودی x مقدار h(x) را که یا ۰ است یا ۱ تعیین کند. این ماشین تورینگ یا الگوریتم را HALT می نامیم.

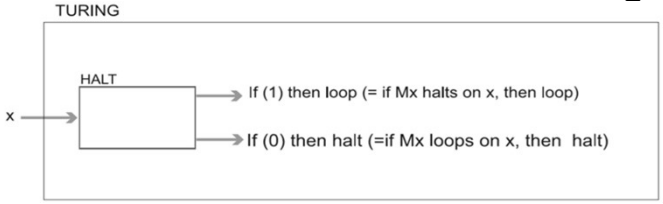


الگوریتم HALT برای محاسبه تابع h

تصمیم پذیری Decideability

مساله توقف (Halting problem) برای ماشین های تورینگ:

حال نشان می دهیم که فرض وجود چنین ماشینی به تناقض منطقی می انجامد. برای این کار به ترتیب زیر استدلال می کنیم. یک برنامه بزرگتر درست می کنیم که برنامه HALT را فراخوانی کند. شکل زیر طرز کار این برنامه را نشان می دهد. دقت کنیدکه این برنامه که آن را Turing می نامیم بازهم ورودی اش یک عدد صحیح x است. اگر برنامه Halt وجود داشته باشد، برنامه Turing هم وجود خواهد داشت. تا اینجا هیچ تناقضی وجود ندارد. برای نشان دادن تناقض، دقت می کنیم که ماشین Turing خودش می بایست یک شماره داشته باشد که با عدد صحیح t بیان شود. بنابراین می توان نوشت M_t = Turing.

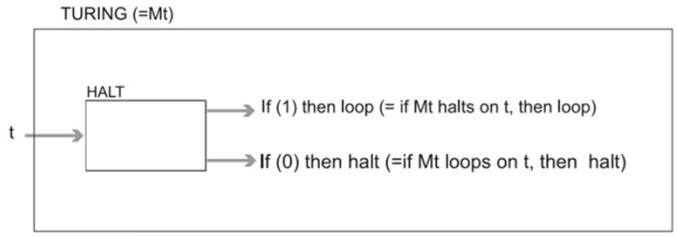


اگر الگوریتم یا ماشین تورینگHALT وجود داشته باشد، الگوریتم یا ماشین تورینگ Turing نیز وجود خواهدداشت.

تصمیم پذیری Decideability

مساله توقف (Halting problem) برای ماشین های تورینگ:

حال به این ماشین همان عدد t را می دهیم. حال همان طور که شکل زیر نشان می دهد، به یک تناقض منطقی می رسیم.



می توانیم به ماشین Turing یا همان M_t ورودی t را بدهیم و تناقض منطقی وجود ماشین Turing و درنتیجه ماشین HALT را ببینیم.

این که ماشین HALT وجود ندارد، به این معناست که تابع h(x) محاسبه پذیرنیست. به عبارت دیگر هیچ الگوریتمی وجودندارد که این تابع را حساب کند.

تصمیم پذیری Decideability

مساله توقف (Halting problem) برای ماشین های تورینگ:

اگر چنین ماشین تورینگی وجود داشته باشد به این معناست که می تواند بگوید آیا یک الگوریتم کار می کند یا نه؟ برای آنکه اهمیت این مسئله را دریابیم یادآوری می کنیم که اگر چنین ماشینی وجود داشته باشد به این معناست که می توان تمام قضایای مهم ریاضی را به طور الگوریتمی حل کرد.

به عنوان مثال یک مسئله مهم در نظریه اعداد مثل حدس گلدباخ رادر نظر بگیرید. این حدس بیان می کند که هر عدد زوج را حتما می توان به صورت مجموع دو عدد اول نوشت. کافی است که خواننده به مثال های ساده زیر توجه کند:

۴ = ۲ + ۲ ,۶ = ۳ + ۳ ,۸,۳ = ۳ + ۵ ,۱۰ = ۵ + ۵ ,۲۰ = ۷ + ۱۳, ۳۰ = ۲۳ + ۷,۰۰۰.

تا کنون کسی نتوانسته است این حدس را اثبات کند و یا مثال نقیضی برعلیه آن بیاورد و این مسئله همچنان یک مسئله باز در ریاضی است. حال فرض کنید که ما یک برنامه $M_{Goldbach}$ می نویسیم و این برنامه هر عدد زوج x را می گیرد و تست می کند که آیا این عدد مجموع دو عدد اول هست یاَ نه. اگر جواب مثبت بود ماشین می ایستد و جواب آری می دهد واگرجواب منفی بود بازهم ماشین می ایستد و جواب نه می دهد.

تصمیم پذیری Decideability

مساله توقف (Halting problem) برای ماشین های تورینگ:

مسلم است که چنین برنامه ای را می توان نوشت. این برنامه یعنی برنامه $M_{Goldbach}$ تکلیف هر عددی را روشن می کند و ما می توانیم این برنامه را برای هر عددی از ۲ تا یک عدد زوج بسیار بسیار بزرگ چند هزار رقمی اجرا کنیم و می توانیم که مثلاً بگوییم که حدس گلدباخ تا اعداد ۱۰۰۰ رقمی صحیح است. ولی این به این معنی نیست که ما حدس گلدباخ را ثابَت کرده ایم. (البته اگر یک عدد نقیض پیدا کنیم آنگاه حدس گلدباخ را نفی کرده ایم.) در واقع زبان این ماشین به این صورت است:

L = {The set of all even integers which are the sum of two prime numbers}.

تصمیم پذیری Decideability

کاهش پذیری (Reducibility):

یک مسئله تصمیم‌گیری P کاهش پذیر تورینگ به یک مسئله P' است. اگر ماشین تورینگی وجود داشته باشد که هر مسئله $p_i \in P$ را به عنوان ورودی پذیرفته و یک مسئله $p'_i \in P'$ را تولید کند به طوری که جواب مسئله اصلی P_i را بتوان از جواب P'_i به دست آورد. اگر یک مسئله تصمیم‌گیری P' تصمیم پذیر بوده و P به P' کاهش پذیر باشد آنگاه P نیز تصمیم پذیر است. کاهش روشی است که اغلب در حل مسائل به کار گرفته می‌شود. در برخورد با یک مسئله جدید اغلب سعی می‌کنیم که آن را به مسئله ای که از قبل حل شده است تبدیل کنیم. این دقیقاً استراتژی به کار رفته در کاهش مسائل تصمیم‌گیری است.

مثال: مسئله P شامل رشته های پذیرشی موجود در زبان $L=\{uu|u=a^ib^ic^i,i\geq 0\}$ را در نظر بگیرید ماشین M قبلا طراحی کرده ایم که زبان $L=\{a^ib^ic^i|i\geq 0\}$ را می پذیرد. مسئله P را به مسئله تشخیص نمونه منفرد $a^ib^ic^i$ کاهش خواهیم داد. آنگاه مسئله اصلی می تواند با استفاده از کاهش و ماشین M حل شود.

تصمیم پذیری Decideability

مسئله متناظر پست (Post Corresponding Problem):

یک مساله تصمیم‌ناپذیر بر روی رشته‌ها است که در سال ۱۹۴۶ میلادی به‌وسیله امیل پست معرفی و اثبات شد. با نگاشت این مسئله بر روی مسائل تصمیم‌ناپذیر دیگر می‌توانیم تصمیم‌ناپذیری آن‌ها را نیز اثبات کنیم (مانند تصمیم‌ناپذیری ماشین‌های تورینگ).

مثال: هر نمونه از این مسئله شامل دو لیست از رشته‌هایی است که بر روی الفبای $\sum$ تعریف شده‌اند. فرض کنید این دو لیست A و B نام دارند که برای یک مقدار k این‌گونه تعریف می‌شوند:

$$A = x_1, x_2, \ldots, x_k$$

$$B = y_1, y_2, \ldots, y_k$$

برای هر i جفت (x_i,y_i) را یک جفت تناظر می‌گویند.

تصمیم پذیری Decideability

مسئله متناظر پست (Post Corresponding Problem):

جواب مسأله متناظر پست به صورت زیر است:

برای یک نمونه از مسئله جواب وجود دارد اگر دنباله‌ای از اعداد صحیح مانند i_1، i_2، ...، i_n وجود داشته باشد که شرط زیر را ارضاء کند:

$$x_{i_1} x_{i_2} .. x_{i_n} = y_{i_1} y_{i_2} .. y_{i_n}$$

پس به‌طور کلی مسأله متناظر پست این‌گونه تعریف می‌شود:

"آیا برای یک نمونه از مسأله متناظر پست جوابی وجود دارد یا خیر؟"

مثال: فرض کنید $\sum$={a,b} و

دو لیست A و B به صورت زیر هستند:

list A	list B	
x_i	y_i	i
a	aaa	۱
abaaa	ab	۲
ab	b	۳

تصمیم پذیری Decideability

مسئله متناظر پست (Post Corresponding Problem):

یک جواب برای این نمونه با n=4، این‌گونه است: $i_1=2, i_2=1, i_3=1, i_4=3$ زیرا در این صورت:

$$x_{i_1} x_{i_2} x_{i_3} x_{i_4} = x_2 x_1 x_1 x_3 = abaaaaaab$$

$$y_{i_1} y_{i_2} y_{i_3} y_{i_4} = y_2 y_1 y_1 y_3 = abaaaaaab$$

که این دو شرط $x_{i1}x_{i2}x_{i3}x_{i4}=y_{i1}y_{i2}y_{i3}y_{i4}$

ایده کلی برای اثبات تصمیم ناپذیری مسأله متناظر پست این است که آن را بر روی یک مسأله تصمیم ناپذیر معروف دیگری نگاشت کنیم و بهترین گزینه مساله توقف است. این نگاشت را این‌گونه در نظر بگیرید: «یک مسأله متناظر پست می‌تواند یک ماشین تورینگ دلخواه را با یک ورودی خاص اجرا کند. مسأله متناظر پست در صورتی جواب دارد اگر و فقط اگر ماشین تورینگ ورودی‌اش را قبول و توقف کند.»

کلاس‌های پیچیدگی محاسباتی

یکی از اهداف نظریه محاسبه، بررسی حداقل منابعی است که برای حل کلاس های مسائل مورد نیاز است. این منابع عبارتنداز زمان (time)، حافظه (space) و انرژی(Energy).

یک مسئله یا یک الگوریتم معین را در نظر بگیرید. به عنوان مثال می خواهیم ببنیم که آیا در یک گراف یک مسیر هامیلتونی وجود دارد یا نه؟ ورودی این مسئله اطلاعات مربوط به یک گراف است. یک گراف عبارت است ازمجموعه ای از نقاط که بعضی از آنها به هم وصل شده اند. مجموعه نقاط یا راس ها را با V و مجموعه یال ها را با E نشان می‌دهیم. در واقع $E \subset V \times V$، یعنی اینکه عناصر E جفت هایی هستند به صورت (v,w) که نشان دهنده این است که راسهایv و w به هم وصل هستند. معمولاً یک گراف را به صورت G = (V,E) نشان می دهیم و تمامی اطلاعات این گراف رابه صورت یک رشته طولانی که در آن راس‌های V و یال های E با فاصله از هم جدا شده‌اند، به صورت یک رشته به ماشین تورینگ می دهیم. طول این رشته را اندازه این مسئله خاص می گیریم و با عدد صحیح n نشان می دهیم. سوالی که با آن روبرو هستیم این است که برای حل مسئله مورد نظر با اندازه n حداقل زمان لازم یا حداقل حافظه لازم چقدراست. تعیین این حداقل منابع اهمیت زیادی در نظریه محاسبه دارد.

کلاس‌های پیچیدگی محاسباتی

نکته مهم آن است که ما به رفتار مجانبی این زمان ها یا حافظه ها وقتیکه اندازه مسئله بزرگ می شود توجه داریم و نه به منابع لازم برای حل یک مسئله خاص. می خواهیم بدانیم که وقتی گرافهای بزرگ و بزرگ تر را بررسی می کنیم زمان لازم یا حافظه لازم برای حل مسئله چگونه رشد می کند؟ بنابراین ابتدا باید روی تعاریفی از رفتار مجانبی توابع توافق کنیم.

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

$$f(n) \text{ is } O(g(n)), \quad \text{if} \quad \exists\, c \text{ and } n0 \quad \forall n > n0 \quad f(n) \le cg(n)$$

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

$$f(n) \text{ is } \Omega(g(n)), \quad \text{if} \quad \exists\, c \text{ and } n0 \quad \forall n > n0 \quad f(n) \ge cg(n)$$

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

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

کلاس‌های پیچیدگی محاسباتی

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

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

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

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

کلاس P:

تعریف: به کلاس مسایلی که یک ماشین تورینگ با زمان اجرای $O(n^k)$ آنها را حل می کند، کلاس $DTime(n^k)$ می گوییم. لفظ DTime به این خاطر آمده است که از یک ماشین تورینگ معینی استفاده کردهَ ایم. معنای این حرف این است که حل دشوارترین مسائل در این کلاس نیز به زمان اجرایی کمتر از مضربی از n^k نیاز دارد. دقت کنید که $DTime(n^k)$ خانواده ای ازمسئله ها یا زبان هاست و نه یک زبان خاص. حال می توانیم کلاس مسائل P (Polynomial Time Problems or Polynomial Time Algorithms)، یعنی مسائل چند جمله ای را تعریف کنیم:

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

کلاس P:

کلاس مسایل P، مجموعه همه مسایل یا الگوریتم هایی است که در زمان چندجمله ای حل می شوند. طبیعی است که چنین مسائلی را مسائل آسان بنامیم. به عنوان مثال، مسئله ای مثل مرتب کردن یک مجموعه n تایی از اعداد، یا ضرب دو ماتریس nبعدی، یا جستجو در یک پایگاه داده ها که n عضو دارد، همه مسائلی هستند که برای حل آنها زمان چند جمله ای مورد نیازاست. بسیاری از مسائلی که نرم افزار هایی مثل Maple و Mathematica حل می کنند نیز چنین هستند. البته نمی توان گفت که هرنوع مسئله ای که این برنامه ها و نظایرآن حل می کنند از نوع مسائل آسان یا P است، زیرا آنها فقط یک نمونه از مسئله را بااندازه معین حل می کنند که به ازای حل آن هم بعضا زمان طولانی صرف می کنند. به عنوان مثال اگر وقت کافی به هرکدام از این برنامه ها بدهید قادر خواهند بود که یک عدد بزرگ را تجزیه کنند، ولی مسئله تجزیه عدد یک مسئله از نوع P نیست زیرا الگوریتمی که برحسب اندازه مسئله (تعداد رقم های عدد اولیه) چند جمله ای باشد شناخته نشده است.(دراینجا فقط درباره الگوریتم های کلاسیک بحث می کنیم).

کلاس‌های پیچیدگی محاسباتی

کلاس NP:

تعریف: به کلاس مسایلی که یک ماشین تورینگ نامعین در زمان $O(n^k)$ آنها را حل می کند، کلاس $NDTime(n^k)$ میگوییم. لفظ NDTime به این دلیل آمده است که از یک ماشین تورینگ نامعین استفاده کَرده ایم. حال می توانیم کلاس مسائل NP (Non-deterministic Polynomial Time Problems or Algorithms) را تعریف کنیم:

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

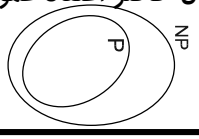
از آنجا که یک ماشین تورینگ معین نوع خاصی ماشین تورینگ نامعین است نتیجه بدیهی تعاریف بالا آن است که $DTime(n^k) \subset NDTime(n^k)$

و در نتیجه $P \subset NP$

کلاس‌های پیچیدگی محاسباتی

کلاس NP:

یک ماشین تورینگ نامعین وقتی یک مسئله را در زمان چند جمله ای حل می کند که یکی از شاخه های محاسبه به جواب برسد، یا به حالت q_{Accept} برسد. اگر بخواهیم کار یک ماشین تورینگ نامعین را با یک ماشین تورینگ معین شبیه سازی کنیم طبیعتا باید همه شاخه ها را همزمان دنبال کنیم. این کار موجب افزایش نمایی تعداد حالت ها و تعداد مراحل محاسبه می شود. بنابراین به طور شهودی وقتی که یک مسئله روی یک ماشین تورینگ نامعین زمان اجرای چندجمله ای نیاز دارد به این معناست که این مسئله روی ماشین تورینگ معین زمان اجرای نمایی از نوع $O(2^n)$ است. درنتیجه می توان گفت که این نوع مسائل، نوعاً مسائل دشوارند، زیرا زمان لازم برای حل آنها با افزایش اندازه مسئله به صورت نمایی رشد می کند. البته ممکن است که بتوان برای همان مسئله روی ماشین تورینگ معین هم یک الگوریتم چندجمله ای پیدا کرد. این سوال که آیا کلاس NP واقعاً ازکلاس P بزرگتر است، یک سوال باز و بسیار مهم در نظریه محاسبه است. در حال حاضر اعتقاد عمومی این است که $P \neq NP$.



کلاس NP:

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

کلاس NP:

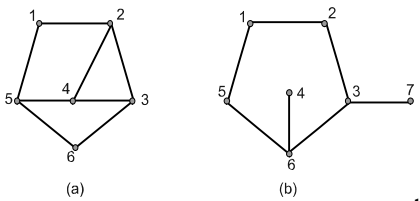
دراینجا منظور از n اندازه ورودی x است.

if $x \in L \; \exists\, y \in L \mid TM\ haltsinq_{Accept}\ ontheinput(x\ b\ y)\ after\ time(Poly(n)),$

if $x \notin L \; \forall\, y \in L \mid TM\ haltsinq_{Reject}\ ontheinput(x\ b\ y)\ after\ time(Poly(n)).$

مسئله گراف های هامیلتونی:

در یک گراف $G = (V,E)$ که دارای راس های **V** و یال های **E** است. یک مسیر، مسیر هامَیلتونی خوانده می شود، هرگاه این مسیر تمام راس های گراف را یک بار و فقط یک بار ملاقات کند (ازآنها بگذرد). یک گراف هامیلتونی، گرافی است که حداقل یک مسیر هامیلتونی داشته باشد. به عنوان مثال در زیر:



گراف **a** هامیلتونی است ولی گراف **b** هامیلتونی نیست.