

به نام خدا

طراحی کامپایلر

آرش شفیعی



تحليل لغوی

- در این فصل نشان می‌دهیم چگونه یک تحلیل‌گر لغوی ساخته می‌شود.
- یک روش ساده برای ساختن تحلیل‌گر لغوی این است که برنامه‌ای بنویسیم که لغات را یک به یک خوانده و شناسایی کند و اطلاعات توکن‌ها را استخراج کند.
- یک روش دیگر برای ساختن تحلیل‌گر لغوی استفاده از ابزاری است که الگوهای مورد نظر برای استخراج واژه‌ها را دریافت کرده، یک تحلیل‌گر لغوی بسازد. چنین ابزاری تولیدکننده تحلیل‌گر لغوی¹ نامیده می‌شود.
- استفاده از یک ابزار تولیدکننده تحلیل‌گر لغوی مزیت‌هایی دارد، از جمله اینکه تغییر تحلیل‌گر لغوی را آسان می‌کند و فرایند توسعه تحلیل‌گر لغوی را تسریع می‌کند، چراکه برنامه‌نویس تنها نیاز دارد یک توصیف سطح بالا از تحلیل‌گر لغوی ارائه کند. و جزئیات پیاده‌سازی بر عهده تولیدکننده تحلیل‌گر لغوی خواهد بود.

¹ lexical analyzer generator

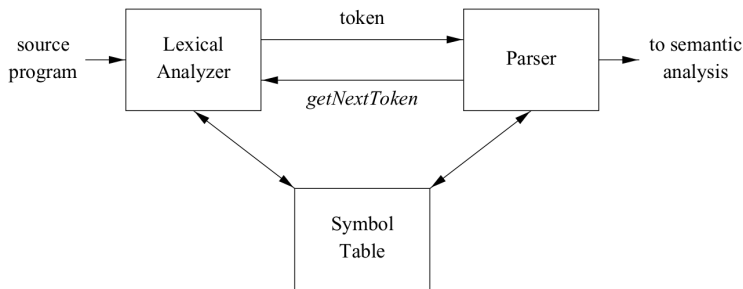
- یکی از مهم‌ترین ابزارهای تولید کننده تحلیل‌گر لغوی فلکس¹ نام دارد.
- برای توصیف تحلیل‌گر لغوی می‌توانیم از زبان‌های منظم² استفاده کنیم. برای هر زبان منظم می‌توان یک ماشین متناهی قطعی یا غیر قطعی ساخت و برای یک ماشین متناهی می‌توان کد تولید کرد.

¹ Flex / Lex

² regular languages

تحلیل‌گر لغوی

- تحلیل‌گر لغوی کاراکترهای ورودی را یک‌به‌یک دریافت کرده، آنها را دسته‌بندی و به واژه¹ تبدیل می‌کند. خروجی یک تحلیل‌گر لغوی دنباله‌ای است از واژه‌های برنامه‌مبداء. این دنباله از واژه‌ها برای تحلیل نحوی به تحلیل‌گر نحوی یا پارسر ارسال می‌شود. تحلیل‌گر لغوی همچنین هنگام تشخیص یک شناسه جدید آن را در جدول علائم وارد می‌کند.
- ارتباط بین تحلیل‌گر لغوی و تحلیل‌گر نحوی در زیر نشان داده شده است.



¹ lexeme

- تحلیل‌گر لغوی همچنین وظیفه دارد توضیحات¹ و فاصله‌های خالی² را از برنامه جدا کند.
- تحلیل‌گر لغوی همچنین شماره خطوط مربوط به برنامه ورودی را نگهداری می‌کند، چرا که کامپایلر برای ارسال پیام خطا نیازمند چاپ شماره خطوط برنامه است.
- در صورتی که در یک برنامه از ماکروها استفاده شده باشد، وظیفه جایگزین کردن مقادیر ماکروها در کد برنامه برعهده تحلیل‌گر لغوی است.
- جداسازی تحلیل‌گر لغوی و تحلیل‌گر نحوی بدین دلیل است که طراحی کامپایلر ساده‌تر شود و پیچیدگی آن کاهش یابد. با کاهش پیچیدگی می‌توان روش‌هایی برای افزایش کارایی تحلیل‌گر لغوی و تحلیل‌گر نحوی به طور جداگانه به کار برد.

¹ comment

² white space

- یک توکن¹ یک دوتایی است شامل نام توکن و ویژگی‌های آن. نام توکن در واقع نوع آن را نشان می‌دهد.
- یک الگو² توصیفی است از شکلی که واژه‌های مربوط به توکن‌ها می‌توانند داشته باشند.
- یک واژه³ دنباله‌ای است از کاراکترهای یک برنامه. یک واژه دارای یک نوع است که بر اساس آن نوع یک توکن برای واژه ساخته می‌شود و یک واژه از یک نوع معین بر اساس یک الگو ساخته می‌شود و کاراکترهای آن بر یک الگو منطبق⁴ می‌شوند.

¹ token

² pattern

³ lexeme

⁴ match

- در شکل زیر تعدادی توکن رایج در زبان‌های برنامه‌نویسی نشان داده شده‌اند.

TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters i, f	if
else	characters e, l, s, e	else
comparison	< or > or <= or >= or == or !=	<=, !=
id	letter followed by letters and digits	pi, score, D2
number	any numeric constant	3.14159, 0, 6.02e23
literal	anything but ", surrounded by "'s	"core dumped"

- در بسیاری از زبان‌های برنامه‌نویسی می‌توان توکن‌ها را به دسته‌های زیر تقسیم کرد.

۱. یک توکن به ازای هر یک از کلمات کلیدی زبان

۲. یک توکن به ازای یک گروه از عملگرها

۳. یک توکن برای نمایش شناسه‌ها

۴. یک یا چند توکن برای نمایش اعداد و رشته‌ها

۵. چند توکن برای نمادهای علامت گذاری مانند ویرگول و پرانتز و نقطه ویرگول.

- یکی از ویژگی‌هایی که برای همه توکن‌ها می‌توان ذخیره کرد، شماره خط توکن در برنامه مبدأ است که برای صدور خطا در تحلیل‌گر نحوی می‌تواند مورد استفاده قرار بگیرد. برای شناسه‌ها معمولاً توکن id ساخته می‌شود که ویژگی‌های آن شامل نام شناسه، نوع شناسه، اولین خطی که شناسه در آن تعریف شده است می‌شود. ویژگی‌های شناسه‌ها معمولاً در جدول علائم نگهداری می‌شود، بنابراین ویژگی توکن id می‌تواند اشاره‌گری باشد به آن شناسه در جدول علائم.

- در عبارت $E = M * C ** 2$ در زبان نورتن توکن ها به صورت زیر استخراج می شوند.

<id, pointer to symbol-table entry for E>

<assign_op>

<id, pointer to symbol-table entry for M>

<mult_op>

<id, pointer to symbol-table entry for C>

<exp_op>

<number, integer value 2>

- در اینجا توکن های متناظر با عملگرها هیچ ویژگی ندارند.

- تحلیل‌گر لغوی به تنهایی قادر نیست خطاهای برنامه ورودی را تشخیص دهد. برای مثال فرض کنید در قسمتی از برنامه مبدأ داشته باشیم $f(x) = a$ یا $f(x) = a$. در اینجا تحلیل‌گر لغوی نمی‌تواند تشخیص دهد آیا $f(x)$ همان $f(x)$ است که به اشتباه نوشته شده است و یا $f(x)$ یک شناسه است که در واقع نام تابع است. تحلیل‌گر لغوی تنها توکن‌ها را به تحلیل‌گر نحوی تحویل می‌دهد و صدور خطا برعهده تحلیل‌گر نحوی خواهد بود.
- وقتی یک تحلیل‌گر لغوی نمی‌تواند یک واژه را تشخیص دهد، می‌تواند ادامه رشته ورودی را تا جایی که به یک واژه درست می‌رسد حذف کند. همچنین برای صدور خطای مناسب می‌تواند سعی کند کلمه را با یک کلمه دیگر جایگزین کند. از چنین روش‌هایی می‌توان برای ارائه پیشنهادات توسط کامپایلر استفاده کرد.

- برای تشخیص الگوهای توکن‌ها از عبارت‌ها منظم استفاده می‌کنیم. عبارت‌های منظم معمولاً در تولید کننده‌های تحلیل‌گر لغوی استفاده می‌شوند.
- یک الفبا¹ مجموعه‌ای است محدود از نمادها و یا سمبول‌ها². برای مثال الفبای 0, 1 از دو نماد صفر و یک تشکیل شده است که می‌توان با این الفبا همه اعداد دودویی را تشکیل داد.
- مجموعه حروف اسکی³ یا مجموعه حروف یونیکد⁴ در مثال دیگر از الفبا هستند.

¹ alphabet

² symbols

³ ASCII

⁴ Unicode

- یک رشته¹ بر روی یک الفبا دنباله‌ای است محدود از نمادهای الفبا. در نظریه زبان‌ها به یک رشته از الفبا یک جمله² نیز گفته می‌شود. طول رشته s را با $|s|$ نمایش می‌دهیم که درواقع تعداد نمادهای رشته است. یک رشته تهی را با نماد ϵ نمایش می‌دهیم که طول آن صفر است.
- یک زبان³ مجموعه‌ای است شما را از رشته‌هایی بر روی یک الفبای معین. برای مثال مجموعه همه برنامه‌های درست در زبان سی، زبان سی را تشکیل می‌دهند. همچنین مجموعه همه جملات درست در زبان فارسی، زبان فارسی را تشکیل می‌دهند. در تعریف یک زبان فقط صورت نحوی جملات آن را در نظر می‌گیریم و مورد معانی جملات صحبت نمی‌کنیم.

¹ string

² sentence

³ language

- اگر دو رشته x و y را به یکدیگر الحاق¹ کنیم، رشته به دست می‌آوریم که از اضافه کردن y به انتهای رشته x تشکیل می‌شود و آن را با xy نمایش می‌دهیم. از الحاق رشته تهی به یک رشته خود رشته به دست می‌آید
بنابراین $s\epsilon = \epsilon s = s$
- اگر الحاق دو رشته را همانند ضرب دو عدد در نظر بگیریم، آنگاه رشته s به توان i رشته‌ای است که از الحاق s به تعداد i بار با خودش به دست می‌آید. می‌توانیم تعریف کنیم $s^i = s^{i-1}s$ و $s^0 = \epsilon$

¹ concatenation

- یک پیشوند¹ از رشته s رشته‌ای است که از حذف صفر یا تعدادی نمادهای s از انتهای آن به دست می‌آید.
- یک پسوند² از رشته s رشته‌ای است که از حذف صفر یا تعداد بیشتری نمادهای s را ابتدای آن به دست می‌آید.
- یک زیر رشته³ از رشته s با حذف یک پیشوند و یک پسوند از s به دست می‌آید.
- یک پیشوند، پسوند، یا زیر رشته کامل⁴، پیشوند، پسوند یا زیررشته‌ای است که تهی نباشد.
- یک زیر دنباله⁵ از رشته s رشته‌ای است که با حذف صفر یا تعدادی بیشتر نمادها از رشته به دست آید.

¹ prefix

² suffix

³ substring

⁴ proper prefix, suffix, substring

⁵ subsequence

- مهم‌ترین عملیات بر روی زبان‌ها عملیات اجتماع¹، الحاق² و بستار³ هستند.
- اجتماع دو زبان مجموعه‌ای است که از اجتماع جمله‌های دو زبان به دست می‌آید.
- الحاق دو زبان مجموعه‌ای است از همهٔ رشته‌هایی که از الحاق یک رشته از زبان اول و یک رشته از زبان دوم به دست می‌آیند.

¹ union

² concatenation

³ closure

- بستار ستاره یا بستار کلینی¹ بر روی زبان L که به صورت L^* نشان داده می‌شود، مجموعه‌ای است از همه رشته‌هایی که از الحاق جملات زبان L صفر بار یا بیشتر با خودشان به دست می‌آید. عبارت L^0 به معنای الحاق رشته‌های L صفر بار با خودشان است که برابر با زبانی است که تنها از رشته تهی تشکیل شده است. بنابراین $L^0 = \epsilon$. همچنین L^i از الحاق L و L^{i-1} به دست می‌آید بنابراین $L^i = L^{i-1}L$.
- بستار مثبت² یک زبان که با L^+ نشان داده می‌شود مجموعه‌ای از همه رشته‌هایی که از الحاق جملات زبان L یک بار یا بیشتر با خودشان به دست می‌آید.

¹ Kleene closure

² positive closure

- جدول زیر عملگرهای رایج بر روی زبان‌ها را نشان می‌دهد.

OPERATION	DEFINITION AND NOTATION
<i>Union of L and M</i>	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
<i>Concatenation of L and M</i>	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
<i>Kleene closure of L</i>	$L^* = \bigcup_{i=0}^{\infty} L^i$
<i>Positive closure of L</i>	$L^+ = \bigcup_{i=1}^{\infty} L^i$

- جملات یک زبان منظم را می‌توان توسط یک عبارت منظم¹ توصیف کرد. این توصیف با استفاده از عملگرهای اجتماع، الحاق و بستار به دست می‌آید.
- برای مثال فرض کنید `letter_` مجموعه‌ای از همه حروف زبان انگلیسی و علامت زیرخط² باشد و `digit` مجموعه‌ای از همه ارقام ریاضی باشد. آنگاه یک شناسه در زبان سی را می‌توانیم توسط عبارت منظم `letter_(letter_|digit)*` توصیف کنیم.
- علامت خط عمودی به معنای اجتماع است و علامت ستاره به معنای بستار ستاره است. پس یک شناسه تشکیل شده است از یک حرف یا علامت زیرخط که به صفر یا تعدادی حروف و ارقام الحاق شده است.
- یک عبارت منظم r زبان $L(r)$ را توصیف می‌کند. قوانینی که توسط آن به صورت بازگشتی عبارت منظم r به زبان $L(r)$ را توصیف می‌کند به شرح زیر است.

¹ regular expression

² underline

۰. عبارت منظم ϵ زبان $L(\epsilon)$ یا $\{\epsilon\}$ را توصیف می‌کند. همچنین اگر a یک نماد در الفبای Σ باشد آنگاه a زبان $L(a)$ یا $\{a\}$ را توصیف می‌کند.

۱. عبارت $s \mid r$ عبارت منظمی است که زبان $L(r) \cup L(s)$ را توصیف می‌کند.

۲. عبارت rs عبارت منظمی است که زبان $L(r)L(s)$ را توصیف می‌کند.

۳. عبارت $(r)^*$ عبارت منظمی است که زبان $(L(r))^*$ را توصیف می‌کند.

۴. عبارت (r) عبارت منظمی است که زبان $L(r)$ را توصیف می‌کند. بنابراین می‌توان به تعداد دلخواه یک عبارت را پرانتزگذاری کرد.

- عملگر بستار بالاترین اولویت را دارد. عملگر الحاق در مرتبه دوم اولویت قرار دارد و عملگر اجتماع پایین‌ترین اولویت را داراست.
- طبق این تعریف $((b)(c)) \mid (a)$ برابر است با $a \mid bc$ و این عبارت زبانی است که یکی از جملات آن a است و بقیه جملات آن از صفر یا تعداد بیشتری b الحاق شده به نماد c تشکیل شده‌اند.
- زبانی که توسط یک عبارت منظم تعریف می‌شود یک مجموعه منظم¹ نیز نامیده می‌شود. اگر دو عبارت منظم r و s یک مجموعه را توصیف کنند می‌گوییم r و s معادل یکدیگرند و می‌نویسیم $s = r \cdot$ برای مثال $(a \mid b) = (b \mid a)$.

¹ regular set

- تعدادی قوانین جبری برای عبارات منظم وجود دارند که در جدول زیر نشان داده شده‌اند.

LAW	DESCRIPTION
$r s = s r$	$ $ is commutative
$r (s t) = (r s) t$	$ $ is associative
$r(st) = (rs)t$	Concatenation is associative
$r(s t) = rs rt; (s t)r = sr tr$	Concatenation distributes over $ $
$\epsilon r = r\epsilon = r$	ϵ is the identity for concatenation
$r^* = (r \epsilon)^*$	ϵ is guaranteed in a closure
$r^{**} = r^*$	$*$ is idempotent

- گاهی برای سادگی در تعاریف، به برخی یا قسمتی از عبارات منظم یک نام منتسب می‌کنیم و از آن نام‌ها مانند نمادهای الفبای استفاده می‌کنیم.
- اگر \sum یک الفبا از نمادها باشد، آنگاه یک تعریف منظم دنباله‌ای است از تعاریف به صورت $d_1 \rightarrow r_1, d_2 \rightarrow r_2, \dots, d_n \rightarrow r_n$
- به طوری که هر یک از d_i ها یک نماد است که در \sum نیست و با بقیه d ها متفاوت است و هر یک از r_i ها یک عبارت منظم بر روی الفبای $\sum \cup \{d_1, d_2, \dots, d_n\}$ است.
- برای جلوگیری از تعاریف بازگشتی، تعاریف را به گونه‌ای محدود می‌کنیم که r_i بتواند تنها از d های از قبل تعریف شده استفاده کند.

- یک تعریف منظم برای شناسه‌ها در زبان سی به صورت زیر است.

<i>letter_</i>	→	A B ... Z a b ... z _
<i>digit</i>	→	0 1 ... 9
<i>id</i>	→	<i>letter_</i> (<i>letter_</i> <i>digit</i>)*

- اعداد اعشاری را می‌توانیم با استفاده از تعاریف منظم زیر توصیف کنیم.

<i>digit</i>	→	$0 \mid 1 \mid \dots \mid 9$
<i>digits</i>	→	$digit \, digit^*$
<i>optionalFraction</i>	→	$\cdot \, digits \mid \epsilon$
<i>optionalExponent</i>	→	$(E (+ \mid - \mid \epsilon) \, digits) \mid \epsilon$
<i>number</i>	→	$digits \, optionalFraction \, optionalExponent$

تعمیم عبارات منظم

- عبارات منظم در دهه ۱۹۵۰ توسط کلین^۱ با سه عملگر اجتماع، الحاق و بستار معرفی شدند.
- پس از آن عبارات منظم برای افزایش توانایی بیان عبارات، تعمیم داده شدند.
- بستار مثبت^۲ با استفاده از عملگر + نشان داده می‌شود. اگر r یک عبارت منظم باشد، آنگاه $(r)^+$ زبان $(L(r))^+$ را توصیف می‌کند. داریم $r^+ = rr^* = r^*r$ و همچنین $r^* = r^+|\epsilon$
- عملگر صفر یا یک^۳ با استفاده از ? نشان داده می‌شود. داریم $r? = r|\epsilon$ یا به عبارت دیگر $L(r?) = L(r) \cup \{\epsilon\}$. تقدم عملگر صفر یا یک همانند بستار ستاره و بستار مثبت است.

^۱ kleen

^۲ positive closure

^۳ zero or one

- گروه یا کلاس نمادها¹ یک مخفف است که دسته‌بندی نمادها به کار می‌رود. به جای عبارت $[a_1|a_2|\dots|a_n]$ می‌نویسیم $[a_1\ a_2\ \dots\ a_n]$. همچنین در صورتی که یک ترتیب منطقی در نمادها وجود داشته باشد می‌نویسیم $[a_1 - a_n]$. برای مثال $[a - z]$ معادل است با $a|b|c|\dots|z$.

¹ character class

- شناسه‌ها در زبان سی را می‌توانیم به صورت زیر نشان دهیم.

<i>letter_</i>	→	[A-Za-z_]
<i>digit</i>	→	[0-9]
<i>id</i>	→	<i>letter_</i> (<i>letter_</i> <i>digit</i>)*

- اعداد را می‌توانیم به صورت زیر نمایش دهیم.

<i>digit</i>	→	$[0-9]$
<i>digits</i>	→	$digit^+$
<i>number</i>	→	$digits (. digits) ? (E [+ -] ? digits) ?$

- آموختیم چگونه عبارات منظم را توسط الگوهای منظم توصیف کنیم. حال می‌خواهیم با استفاده از این الگوها برنامه‌ای بنویسیم که توکن‌ها را از رشته ورودی استخراج کند.
- گرامر زیر را در نظر بگیرید.

<i>stmt</i>	→	if <i>expr</i> then <i>stmt</i>
		if <i>expr</i> then <i>stmt</i> else <i>stmt</i>
		ε
<i>expr</i>	→	<i>term</i> relop <i>term</i>
		<i>term</i>
<i>term</i>	→	id
		number

شناسایی توکن‌ها

- در این مثال `relop` یک عملگر مقایسه‌ای است مانند علامت کوچکتر `<` ، یا بزرگتر `>` ، مساوی `=` و یا نامساوی `<>` :
- در این این گرامر ترمینال‌ها عبارتند از `id` ، `relop` ، `else` ، `if` و `number` می‌خواهیم با استفاده از تعاریف منظم این ترمینال‌ها را توصیف کنیم.
- الگوهای ترمینال‌ها در این الگوریتم به صورت زیر هستند.

<i>digit</i>	→	[0-9]
<i>digits</i>	→	<i>digit</i> ⁺
<i>number</i>	→	<i>digits</i> (. <i>digits</i>) ? (E [+ -] ? <i>digits</i>) ?
<i>letter</i>	→	[A-Za-z]
<i>id</i>	→	<i>letter</i> (<i>letter</i> <i>digit</i>) *
<i>if</i>	→	<i>if</i>
<i>then</i>	→	<i>then</i>
<i>else</i>	→	<i>else</i>
<i>relop</i>	→	< > <= >= = <>

- فرض می‌کنیم شناسه‌ها در این زبان نمی‌توانند کلمات کلیدی باشند. چنین فرضی کار تحلیل لغوی را ساده‌تر می‌کند. همچنین فاصله‌های خالی را با عنوان توکن `ws` به صورت زیر تعریف می‌کنیم.

$$ws \rightarrow (\text{blank} \mid \text{tab} \mid \text{newline})^+$$

- در اینجا `blank` ، `tab` ، `newline` درواقع کلماتی هستند که به جای کاراکترهای اسکی فاصله خالی، کاراکتر ستون جدید و کاراکتر خط جدید به کار برده شده‌اند.
- توکن `ws` به تحلیل‌گر نحوی یا پارسر تحویل داده نمی‌شود بلکه تحلیل‌گر لغوی از آن چشم‌پوشی می‌کند و کار شناسی توکن بعدی را آغاز می‌کند.

- در تحلیل‌گر لغوی می‌خواهیم توکن‌ها را شناسایی کرده و طبق جدول زیر آنها را به پارسر تحویل دهیم.

LEXEMES	TOKEN NAME	ATTRIBUTE VALUE
Any <i>ws</i>	–	–
if	if	–
then	then	–
else	else	–
Any <i>id</i>	id	Pointer to table entry
Any <i>number</i>	number	Pointer to table entry
<	relop	LT
<=	relop	LE
=	relop	EQ
<>	relop	NE
>	relop	GT
>=	relop	GE

دیاگرام‌های گذار

- قبل از تولید برنامه برای ساخت تحلیل‌گر لغوی، ابتدا الگوهای تهیه شده توسط زبان منظم را به دیاگرام‌های گذار¹ تبدیل می‌کنیم.
- ابتدا به صورت دستی عبارت منظم را به دیاگرام‌های گذار تبدیل می‌کنیم و سپس روشی خودکار برای ساخت دیاگرام‌های گذار با استفاده از عبارات منظم ارائه می‌دهیم.
- دیاگرام گذار به صورت گرافی نمایش داده می‌شود که رئوس آن حالت‌ها² دیاگرام نامیده می‌شوند.
- هر حالت نشان دهنده یک موقعیت در فرایند خواندن رشته ورودی در تحلیل‌گر لغوی است. یال‌های جهت‌دار دیاگرام گذار برای گذار از یک حالت به حالت دیگر استفاده می‌شوند. هریک از یال‌ها دارای یک برچسب³ است و آن برچسب یک یا مجموعه‌ای از نمادهای الفباست. در فرایند تحلیل رشته ورودی، دیاگرام گذار در یکی از حالت‌ها قرار می‌گیرد. فرض کنید دیاگرام در حالت s قرار داشته باشد و نماد بعدی در رشته ورودی a باشد. در این صورت اگر یالی از حالت ورودی s یا برچسب a خارج شده باشد، رأس مقصد آن یال حالت بعدی در دیاگرام گذار را تعیین خواهد کرد.

¹ transition diagram

² states

دیاگرام‌های گذار

- در ابتدا فرض می‌کنیم دیاگرام حالت قطعی¹ است، بدین معنا که به ازای هر حالت فعلی و هر نماد الفبا تنها یک حالت بعدی در دیاگرام حالت وجود دارد. در مورد دیاگرام حالت غیرقطعی بعدها صحبت خواهیم کرد.
- تعدادی از حالت‌های دیاگرام حالت را حالت نهایی² یا حالت پذیرنده³ می‌نامیم. اگر یک رشته خوانده شود و دیاگرام در یک حالت نهایی قرار بگیرد، رشته مورد نظر پذیرفته می‌شود و یک توکن به دست می‌آید. حالت‌های غیرنهایی را با یک دایره و حالت‌های نهایی را با دو دایره تودرتو نمایش می‌دهیم.
- وقتی به یک حالت نهایی می‌رسیم، یک زیر رشته از رشته ورودی پذیرفته می‌شود و یک توکن بازگردانده می‌شود. در صورتی که نیاز داشته باشیم پس از پذیرفتن یک زیررشته یک کاراکتر در رشته ورودی به عقب بازگردیم، در کنار حالت پایانی یک علامت ستاره قرار می‌دهیم.

¹ deterministic

² final

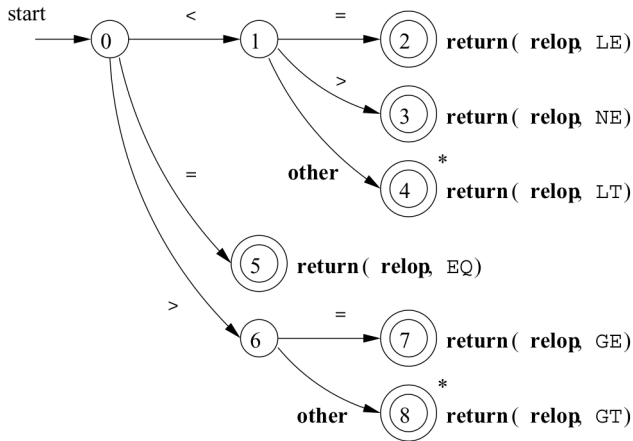
³ accepting

- یکی از حالت‌ها، حالت اولیه¹ نامیده می‌شود. این حالت را با یک یال ورودی بدون مبدأ به نام start مشخص می‌کنیم. یک دیاگرام حالت همیشه با حالت ابتدایی آغاز می‌شود.
- در هر لحظه دیاگرام حالت در یک حالت قرار دارد. به ازای دریافت یک نماد از ورودی حالت تغییر می‌کند. اگر دیاگرام در یک حالت نهایی قرار گرفت یک زیر رشته از رشته ورودی پذیرفته می‌شود و یک توکن استخراج می‌شود.

¹ initial state

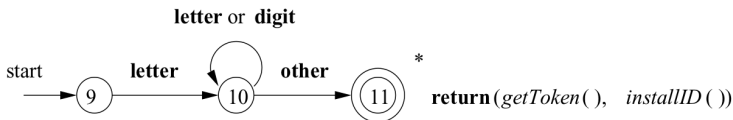
دیاگرام‌های گذار

- دیاگرام گذار زیر برای استخراج عملگرهای مقایسه‌ای طراحی شده است.



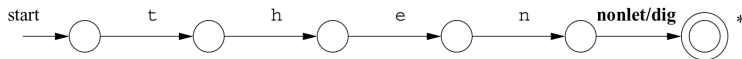
کلمات کلیدی و شناسه‌ها

- برای شناسایی و استخراج شناسه‌ها می‌توانیم از یک دیاگرام گذار مانند دیاگرام زیر استفاده کنیم.

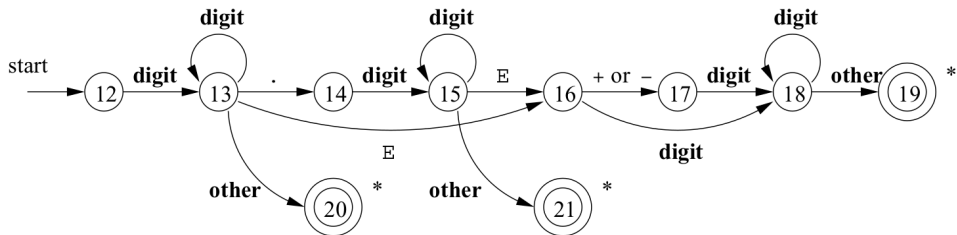


- مشکلی که در این دیاگرام وجود دارد این است که کلمات کلیدی `if` ، `then` و `else` و غیره نیز به عنوان شناسه‌ها استخراج می‌شوند.
- یک راه حل برای حل این مشکل این است که کلمات کلیدی را در ابتدا در جدول علائم قرار دهیم. وقتی یک شناسه تشخیص داده شده، بررسی می‌شود آیا شناسه در جدول علائم قرار دارد یا خیر. اگر شناسه در جدول علائم قرار داشته باشد اشاره‌گری به آن در توکن ذخیره می‌شود. همچنین شناسه‌ها و کلمات کلیدی در جدول علائم تمیز داده می‌شوند.

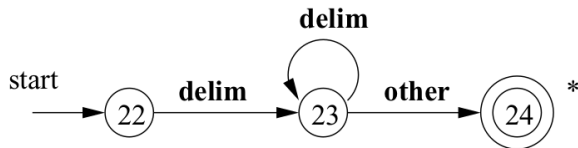
- یک راه حل دیگر برای تشخیص کلمات کلیدی این است که در دیاگرام گذار به ازای هر کلمه کلیدی یک مسیر جداگانه وجود داشته باشد. برای مثال در شکل زیر کلمه then به طور جداگانه تشخیص داده می شود. البته این راه حل برای پیاده سازی پیچیده تر است.



- حال می‌خواهیم دیاگرام گذار را با شناسایی اعداد تکمیل کنیم. دیاگرام زیر برای شناسایی اعداد می‌تواند مورد استفاده قرار بگیرد.



- همچنین قسمتی از دیاگرام گذار برای تشخیص فاصله‌های خالی به صورت زیر می‌تواند طراحی شود.



تحلیل گر لغوی بر مبنای دیاگرام گذار

- برای پیاده‌سازی یک دیاگرام گذار می‌توانیم یک متغیر به نام state تعریف کنیم شماره حالت فعلی را نگهداری می‌کند. سپس بسته به حالت فعلی و کاراکتر خوانده شده از ورودی شماره حالت را تغییر می‌دهیم و در صورتی که به حالت نهایی رسیدیم، یک توکن استخراج می‌کنیم.

تحلیل‌گر لغوی بر مبنای دیاگرام گذار

- در برنامه زیر دیاگرام گذار برای تشخیص عملگرهای رابطه‌ای پیاده‌سازی شده است.

```
TOKEN getRelop()
{
    TOKEN retToken = new(RELOP);
    while(1) { /* repeat character processing until a return
                or failure occurs */
        switch(state) {
            case 0: c = nextChar();
                    if ( c == '<' ) state = 1;
                    else if ( c == '=' ) state = 5;
                    else if ( c == '>' ) state = 6;
                    else fail(); /* lexeme is not a relop */
                    break;
            case 1: ...
            ...
            case 8: retract();
                    retToken.attribute = GT;
                    return(retToken);
        }
    }
}
```

تحلیل گر لغوی بر مبنای دیاگرام گذار

- از این تابع در تحلیل گر معنایی به چند روش می توانیم استفاده کنیم.
- یک روش این است که به ازای هر نوع توکن تابع مربوط فراخوانی شود تا این که رشته ورودی بر یک نوع توکن منطبق شود.
- روش دوم این است که یک دیاگرام گذار کلی برای همه توکن ها طراحی کرده و این دیاگرام گذار را پیاده سازی کنیم. مشکلی که در این روش وجود دارد این است که با ترکیب دیاگرام گذار همه توکن ها در یک دیاگرام، یک دیاگرام غیر قطعی تولید شود. دیاگرام غیر قطعی دیاگرامی است که از یک حالت و یک نماد امکان تغییر حالت به بیش از یک حالت وجود داشته باشد. در چنین مواردی دیاگرام غیر قطعی را باید به دیاگرام قطعی تبدیل کنیم.

- یکی از ابزارهایی که برای تولید تحلیل‌گر لغوی استفاده می‌شود، ابزار لکس¹ است که نسخه جدیدتر آن فلکس² نامیده می‌شود. این ابزار با دریافت توصیف الگوهای توکن‌ها در زبان منظم تحلیل‌گر لغوی تولید می‌کند.
- در اینجا می‌خواهیم بررسی کنیم ابزارهای تولید تحلیل‌گر لغوی چگونه طراحی شده‌اند.
- در طراحی تحلیل‌گرهای لغوی از ماشین‌های متناهی³ استفاده می‌شود.
- ماشین‌های متناهی با اندکی تفاوت شبیه به دیاگرام گذار هستند.

¹ Lex

² Flex

³ Finite Automata

- ماشین‌های متناهی ابزارهایی برای تشخیص¹ رشته‌ها هستند. به عبارت دیگر یک ماشین متناهی یک زبان را توصیف می‌کند. در صورتی که رشته‌ای به ماشین داده شود، ماشین رشته را می‌پذیرد اگر رشته متعلق به زبان ماشین باشد و در صورتی که رشته متعلق به زبان نباشد، ماشین رشته را نمی‌پذیرد. پس پاسخ ماشین به ازای هر رشته ورودی بله و خیر است.
- دو نوع ماشین متناهی وجود دارد : ماشین متناهی غیر قطعی² و ماشین متناهی قطعی³
- ماشین متناهی به صورت گرافی نمایش داده می‌شود که رئوس آن حالت‌های ماشین هستند و یال‌های آن گذارها از یک حالت به حالت دیگر.

¹ recognize

² Nondeterministic finite automata (NFA)

³ Deterministic finite automata (DFA)

- یکی از حالت‌ها، حالت آغازین ماشین است و بقیه حالت‌ها می‌توانند پایانی یا غیرپایانی باشند. ماشین در حالت آغازین عملیات خود را آغاز می‌کند و به ازای خواندن نماد a از رشته ورودی در صورتی که ماشین در حالت p باشد و یال (p, q) با نماد a برچسب‌گذاری شده باشد، ماشین در حالت q قرار می‌گیرد. در صورتی که رشته به اتمام رسیده باشد و ماشین در یک حالت پایانی باشد، ماشین رشته را می‌پذیرد¹، در غیراینصورت ماشین رشته را نمی‌پذیرد یا رد² می‌کند.

¹ accept

² reject

- ماشین متناهی غیر طبیعی بر روی یال‌های خود می‌تواند هر برجسبی داشته باشد. یال‌ها همچنین می‌تواند با نماد ϵ برجسب‌گذاری شده باشند.
- ماشین متناهی قطعی ماشینی است که در آن به ازای هر نماد a از الفبا و هر رأس q_i تنها یک یال خارج شونده از q_i با برجسب a وجود داشته باشد. همچنین هیچ یالی با برجسب ϵ در ماشین متناهی قطعی وجود ندارد.
- ماشین‌های متناهی قطعی و غیرقطعی قادر به شناسایی زبان‌های منظم هستند.

ماشین‌های متناهی غیرقطعی

- یک ماشین غیرقطعی متناهی تشکیل شده است از :

۱. یک مجموعه متناهی S از حالات ¹ ماشین
۲. یک مجموعه \sum از نمادهای ورودی به نام الفبای ورودی ² . فرض می‌کنیم ϵ در الفبای ورودی قرار ندارد.
۳. یک تابع گذار ³ که به ازای هر حالت و هر نماد در $\sum \cup \{\epsilon\}$ حالت بعدی ⁴ را باز می‌گرداند.
۴. یک حالت شروع ⁵ یا حالت آغازین ⁶ S_0 که در مجموعه S قرار دارد.
۵. مجموعه‌ای از حالت‌های پذیرش ⁷ یا حالت‌های نهایی ⁸ به نام F که زیر مجموعه‌ای از S است.

¹ state

² input alphabet

³ transition function

⁴ next state

⁵ start state

⁶ initial state

⁷ accepting state

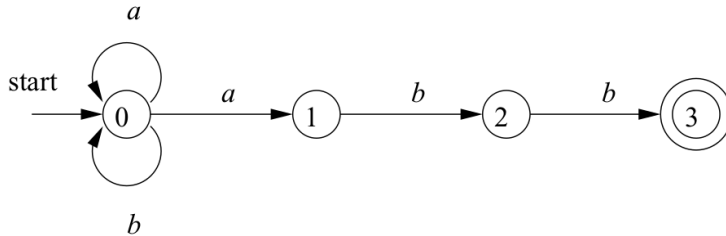
⁸ final state

- ماشین‌های متناهی قطعی و غیرقطعی را می‌توانیم با یک گراف گذار¹ نشان دهیم به طوری که رئوس گراف حالت‌های ماشین و یال‌های برچسب زده شده گراف توابع گذار هستند.
- اگر حالت بعدی s با رؤیت نماد a حالت t باشد آنگاه یالی از s به t با برچسب a در گراف گذار وجود خواهد داشت.
- در ماشین متناهی غیرقطعی چند یال خارج شونده از یک رأس می‌توانند برچسب یکسان داشته باشند و همچنین برچسب یک یال علاوه بر نمادهای الفبای می‌تواند ϵ نیز باشد.

¹ transition graph

ماشین‌های متناهی غیرقطعی

- ماشین متناهی غیرقطعی زیر عبارت منظم $(a|b)^*abb$ را تشخیص می‌دهد.



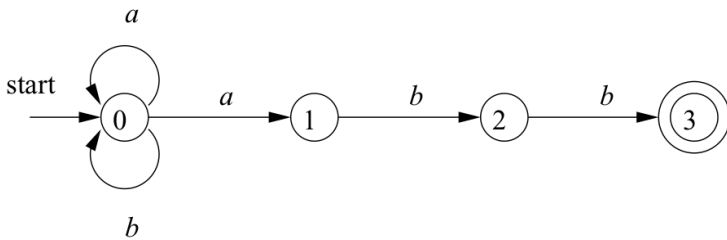
- یک ماشین متناهی غیرقطعی را می‌توانیم توسط جدول گذار¹ نیز نمایش دهیم که در آن هر سطر متعلق به یک حالت و هر ستون متعلق به یک نماد از الفبا یا نماد ϵ است. یک سلول در این جدول در سطر p و ستون a مجموعه حالت‌های بعدی p را با رؤیت نماد a مشخص می‌کند.

¹ transition table

جداول گذار

- یک جدول گذار زیر یک ماشین متناهی غیرقطعی را توصیف می‌کند.

STATE	a	b	ϵ
0	$\{0, 1\}$	$\{0\}$	\emptyset
1	\emptyset	$\{2\}$	\emptyset
2	\emptyset	$\{3\}$	\emptyset
3	\emptyset	\emptyset	\emptyset

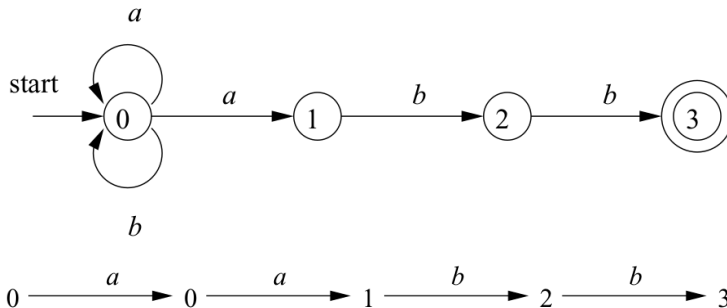


- یک ماشین متناهی غیرطبیعی رشته x را می‌پذیرد¹ اگر و تنها اگر یک مسیر در گراف گذار ماشین با شروع از حالت آغازین و پایان در یک حالت پایانی وجود داشته باشد به طوری که یال‌های مسیر با نمادهای رشته x برچسب گذاری شده باشند.

¹ accepts

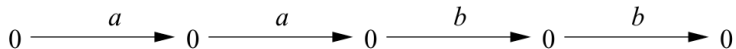
پذیرش یک رشته ورودی

- رشته aabb توسط ماشین متناهی غیرقطعی زیر پذیرفته می‌شود زیرا مسیری با برچسب aabb وجود دارد.



پذیرش یک رشته ورودی

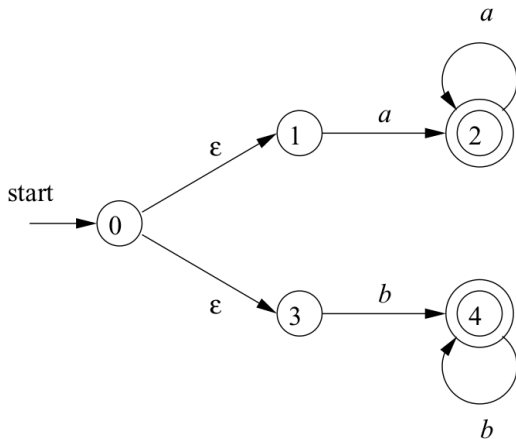
– همچنین مسیر دیگری برای پذیرفتن رشته aabb وجود دارد.



- یک زبان توصیف شده توسط یک ماشین متناهی غیرطبیعی مجموعه رشته‌هایی است که توسط آن ماشین پذیرفته می‌شوند. زبان ماشین A را $L(A)$ می‌نامیم.

پذیرش یک رشته ورودی

- ماشین متناهی قطعی زیر زبان $L(aa^* | bb^*)$ را می‌پذیرد.

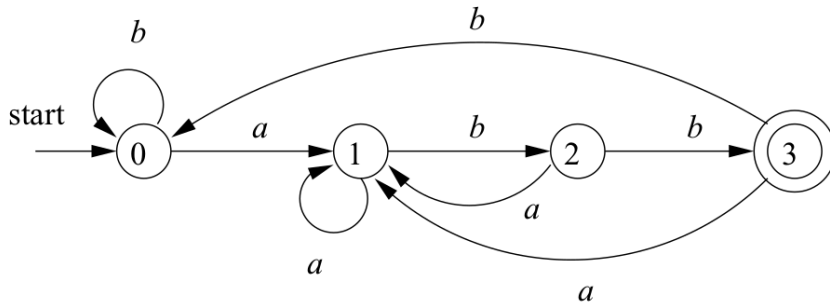


- یک ماشین متناهی قطعی، یک حالت خاص از ماشین متناهی غیرقطعی است که در آن هیچ حرکت با ورودی ϵ وجود ندارد و همچنین به ازای هر حالت s و هر نماد ورودی a تنها یک حالت بعدی برای s وجود دارد.
- به عبارت دیگر در گراف گذار ماشین متناهی قطعی یالی با نماد ϵ وجود ندارد و به ازای هر رأس s و هر نماد ورودی a تنها یک یال خارج شونده از s با برچسب a وجود دارد.
- جدول گذار یک ماشین متناهی قطعی یک جدول کامل است که همه سلول‌های آن مقدار دارند و ستونی با نماد ϵ وجود ندارد و همچنین هر سلول از این جدول تنها یک حالت را در برمی‌گیرد.

- الگوریتم زیر نحوه اجرای یک ماشین متناهی قطعی را نشان می‌دهد. به ازای هریک از کاراکترهای رشته ورودی ماشین به حالت بعدی حرکت می‌کند. اگر با تمام رشته ماشین در حالت پایانی قرار گرفت رشته پذیرفته می‌شود.

```
 $s = s_0;$   
 $c = nextChar();$   
while (  $c \neq eof$  ) {  
     $s = move(s, c);$   
     $c = nextChar();$   
}  
if (  $s$  is in  $F$  ) return "yes";  
else return "no";
```

- گراف گذار زیر یک ماشین متناهی قطعی را نمایش می‌دهد که زبان $abb(a|b)^*$ را می‌پذیرد.



تبدیل عبارت منظم به ماشین

- تولیدکننده‌های تحلیل‌گر لغوی عباراتی به زبان منظم دریافت کرده، به صورت خودکار یک تحلیل‌گر لغوی تولید می‌کنند. برای انجام این کار باید الگوریتمی وجود داشته باشد که عبارات منظم را به یک برنامه تبدیل کند.
- تا اینجا روندی برای تبدیل یک ماشین متناهی قطعی به کد ارائه کردیم. به عبارت دیگر الگوریتمی وجود دارد که یک ماشین متناهی قطعی را دریافت کرده برنامه متناظر آن را تولید می‌کند.
- برای تکمیل تولید کننده تحلیل‌گر لغوی باید الگوریتمی ارائه کنیم که یک عبارت منظم را به یک ماشین متناهی تبدیل کند. یک الگوریتم ساده برای تبدیل عبارت منظم به ماشین متناهی غیرقطعی وجود دارد. بنابراین ابتدا عبارت منظم را به ماشین متناهی غیر قطعی تبدیل می‌کنیم و سپس الگوریتمی برای تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی ارائه می‌کنیم. در نهایت ماشین متناهی قطعی به روشی که ارائه کردیم به کد تبدیل می‌شود. در صورتی که تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی زمان بر باشد، می‌توانیم ماشین متناهی غیر قطعی را مستقیماً به کد تبدیل کنیم.

تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- ایده اصلی تبدیل ماشین متناهی غیر قطعی به ماشین متناهی قطعی این است که هر حالت در ماشین قطعی ساخته شده متناظر است با یک مجموعه از حالات در ماشین غیر قطعی.
- پس از خواندن رشته $a_1 a_2 \dots a_n$ از ورودی ماشین قطعی در حالتی قرار می‌گیرد که متناظر با مجموعه‌ای از حالات ماشین غیرقطعی که با شروع از حالت اولیه و خواندن رشته $a_1 a_2 \dots a_n$ در ماشین غیرقطعی قابل دسترسی هستند.
- این مکان وجود دارد که تعداد حالات ماشین قطعی تولید شده، از مرتبه‌نمایی نسبت به تعداد حالات ماشین غیر قطعی باشد که باعث می‌شود تبدیل ماشین غیرقطعی به ماشین قطعی بسیار زمان‌بر باشد.
- با این حال برای زبان‌های رایج ماشین‌های غیرقطعی و ماشین‌های قطعی معادل آنها تعداد حالاتشان تقریباً برابر است و بنابراین در عمل بدترین حالت اتفاق نمی‌افتد و در زمان معقول می‌توان یک ماشین متناهی غیرطبیعی را به ماشین معادل قطعی تبدیل کرد.

تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- فرض کنید می‌خواهیم ماشین متناهی غیرطبیعی N را به ماشین متناهی قطعی D که همان زبان ماشین N را می‌پذیرد تبدیل می‌کنیم.
- الگوریتمی برای انجام این کار ارائه می‌کنیم.
- یک جدول گذار $Dtran$ برای ماشین D تشکیل می‌دهیم. هریک از حالات ماشین D مجموعه‌ای است از حالات ماشین N . حال $Dtran$ را به گونه‌ای تشکیل می‌دهیم که ماشین N را شبیه‌سازی کند.

تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- برای توصیف الگوریتم تبدیل ابتدا چند تابع را به صورت زیر تعریف می‌کنیم. توجه کنید در جدول زیر s یک حالت از ماشین N است در حالی که T مجموعه‌ای از حالات N را نشان می‌دهد

OPERATION	DESCRIPTION
$\epsilon\text{-closure}(s)$	Set of NFA states reachable from NFA state s on ϵ -transitions alone.
$\epsilon\text{-closure}(T)$	Set of NFA states reachable from some NFA state s in set T on ϵ -transitions alone; $= \cup_{s \text{ in } T} \epsilon\text{-closure}(s)$.
$\text{move}(T, a)$	Set of NFA states to which there is a transition on input symbol a from some state s in T .

تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- الگوریتم تابع ϵ -closer را می‌توانیم به صورت زیر توصیف کنیم.

```
push all states of  $T$  onto  $stack$ ;  
initialize  $\epsilon$ -closure( $T$ ) to  $T$ ;  
while (  $stack$  is not empty ) {  
    pop  $t$ , the top element, off  $stack$ ;  
    for ( each state  $u$  with an edge from  $t$  to  $u$  labeled  $\epsilon$  )  
        if (  $u$  is not in  $\epsilon$ -closure( $T$ ) ) {  
            add  $u$  to  $\epsilon$ -closure( $T$ );  
            push  $u$  onto  $stack$ ;  
        }  
}
```

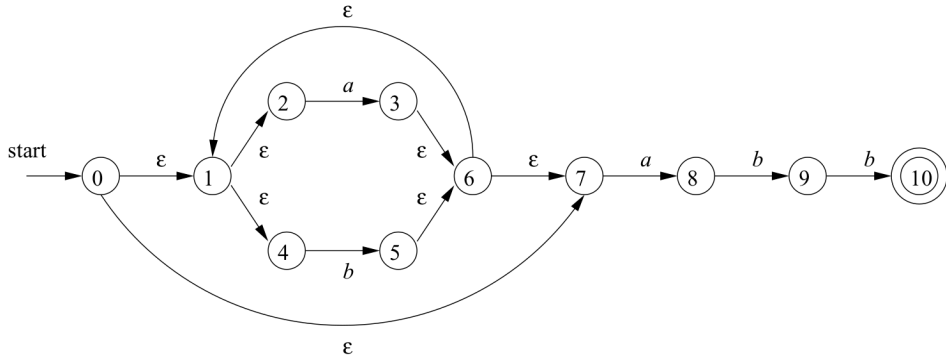
تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- سپس با استفاده این تعاریف الگوریتمی برای تبدیل ماشین غیرطبیعی با ماشین قطعی به صورت زیر طراحی می‌کنیم.

```
initially,  $\epsilon$ -closure( $s_0$ ) is the only state in  $Dstates$ , and it is unmarked;  
while ( there is an unmarked state  $T$  in  $Dstates$  ) {  
    mark  $T$ ;  
    for ( each input symbol  $a$  ) {  
         $U = \epsilon$ -closure( $move(T, a)$ );  
        if (  $U$  is not in  $Dstates$  )  
            add  $U$  as an unmarked state to  $Dstates$ ;  
         $Dtran[T, a] = U$ ;  
    }  
}
```

تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- فرض کنید می‌خواهیم ماشین غیرقطعی زیر را به ماشین معادل قطعی آن تبدیل کنیم.



تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- حالت اولیه ماشین قطعی که از $\epsilon - \text{closure}(0)$ به دست می‌آید برابر است با $A = \{0, 1, 2, 4, 7\}$
- همچنین داریم :

$$\text{Dtans}[A, a] = \epsilon - \text{closure}(\text{move}(A, a)) = \epsilon - \text{closure}(\{3, 8\}) = \{0, 1, 2, 4, 6, 7, 8\} = B$$

$$\text{Dtans}[A, a] = \epsilon - \text{closure}(\{5\}) = \{0, 1, 2, 4, 5, 6, 7\} = C$$

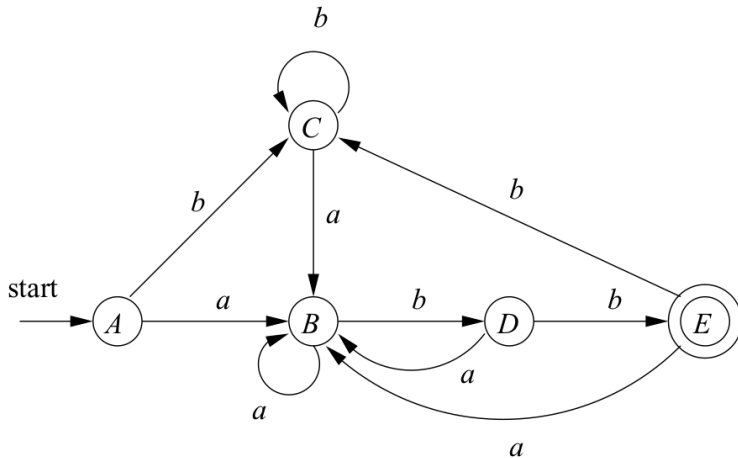
تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- بدین ترتیب می‌توانیم جدول زیر را تشکیل دهیم.

NFA STATE	DFA STATE	<i>a</i>	<i>b</i>
$\{0, 1, 2, 4, 7\}$	<i>A</i>	<i>B</i>	<i>C</i>
$\{1, 2, 3, 4, 6, 7, 8\}$	<i>B</i>	<i>B</i>	<i>D</i>
$\{1, 2, 4, 5, 6, 7\}$	<i>C</i>	<i>B</i>	<i>C</i>
$\{1, 2, 4, 5, 6, 7, 9\}$	<i>D</i>	<i>B</i>	<i>E</i>
$\{1, 2, 4, 5, 6, 7, 10\}$	<i>E</i>	<i>B</i>	<i>C</i>

تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- ماشین متناهی قطعی به دست آمده به صورت زیر خواهد بود.



شبیه سازی ماشین های متناهی قطعی

- یک روش برای تبدیل عبارات منظم به تحلیل گر لغوی این است که عبارت منظم را به ماشین متناهی غیرقطعی تبدیل کرده، سپس ماشین متناهی غیرقطعی را بدون تبدیل به ماشین قطعی شبیه سازی کنیم.

شبیه سازی ماشین های متناهی قطعی

- الگوریتم بدین صورت عمل می کند که مجموعه ای از حالات فعلی به نام S را نگهداری می کند که با شروع از S_0 و خواندن رشته تا به حال خوانده شده قابل دسترسی هستند. با خواندن کاراکتر ورودی c مجموعه همه حالاتی که از حالات فعلی و خواندن کاراکتر c قابل دسترسی هستند محاسبه می شوند و مجموعه حالات فعلی تغییر می کند.

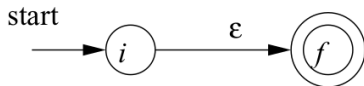
```
1)  $S = \epsilon\text{-closure}(s_0);$   
2)  $c = \text{nextChar}();$   
3) while (  $c \neq \text{eof}$  ) {  
4)      $S = \epsilon\text{-closure}(\text{move}(S, c));$   
5)      $c = \text{nextChar}();$   
6) }  
7) if (  $S \cap F \neq \emptyset$  ) return "yes";  
8) else return "no";
```

تبدیل عبارت منظم به ماشین غیرقطعی

- حال می‌خواهیم الگوریتمی طراحی کنیم که یک عبارت منظم دلخواه را به یک ماشین متناهی غیر قطعی تبدیل کند. فرض کنید عبارت منظم r بر روی الفبای Σ تعریف شده است. می‌خواهیم ماشین غیر قطعی N را طراحی کنیم که ماشین $L(r)$ را بپذیرد.
- در این الگوریتم ابتدا به ازای هر زیر عبارت یک ماشین تولید می‌شود و سپس بر اساس عملگرهای عبارت منظم، زیر عبارت‌ها به یکدیگر متصل می‌شوند.

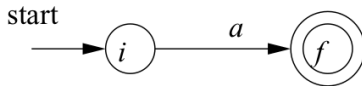
تبدیل عبارت منظم به ماشین غیرقطعی

- ماشین غیرقطعی متناظر عبارت ϵ به صورت زیر است.



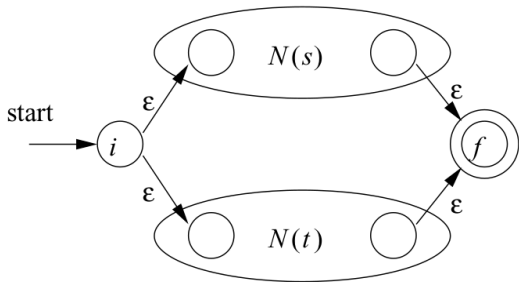
- در اینجا i یک حالت غیرپایانی جدید و f یک حالت پایانی جدید است.

- ماشین غیرقطعی متناظر هر زیر عبارت a در Σ به صورت زیر است.



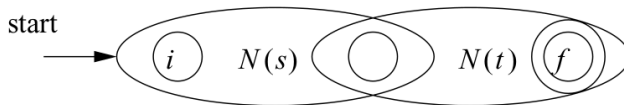
تبدیل عبارت منظم به ماشین غیرقطعی

- فرض کنید $N(s)$ و $N(t)$ دو ماشین غیرقطعی برای دو عبارت منظم s و t باشند، آنگاه ماشین غیرقطعی $N(r)$ برای عبارت منظم $r = s|t$ به صورت زیر است.



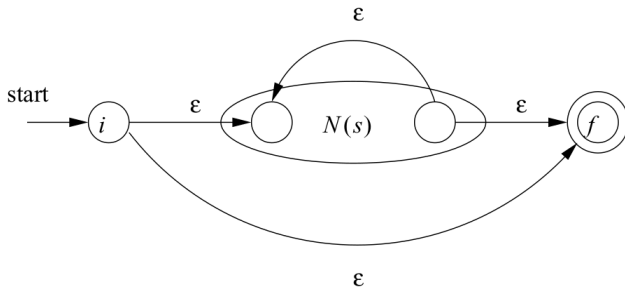
تبدیل عبارت منظم به ماشین غیرقطعی

- همچنین ماشین غیرقطعی $N(r)$ برای عبارت منظم $r = st$ به صورت زیر است.



تبدیل عبارت منظم به ماشین غیرقطعی

- ماشین غیرقطعی $N(r)$ برای عبارت $r = s^*$ به صورت زیر است.

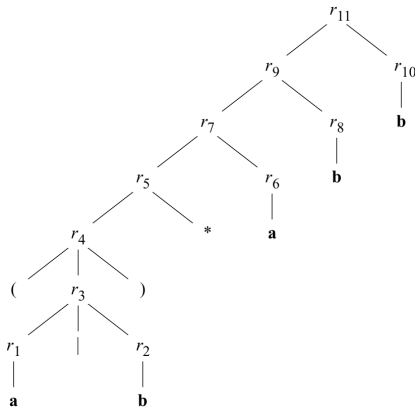


تبدیل عبارت منظم به ماشین غیرقطعی

- همچنین اگر داشته باشیم $r = (s)$ آنگاه $L(r) = L(s)$.
- بدین ترتیب برای هر عبارت منظم داده شده می‌توانیم یک ماشین غیرقطعی بسازیم.

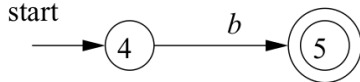
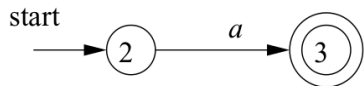
تبدیل عبارت منظم به ماشین غیرقطعی

- فرض کنید عبارت $r = (a|b)^* abb$ داده شده است. ابتدا یک درخت تجزیه برای این عبارت به صورت زیر می‌سازیم و سپس با استفاده از قوانین ذکر شده در الگوریتم قبل یک ماشین غیرقطعی معادل این عبارت منظم می‌سازیم.

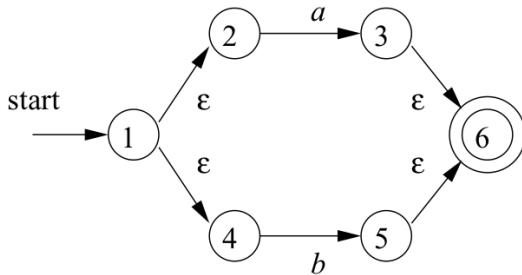


تبدیل عبارت منظم به ماشین غیرقطعی

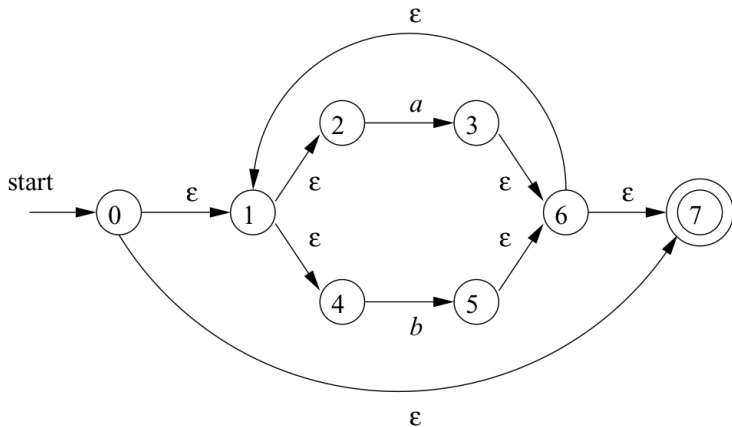
- حال ماشین غیرقطعی را به صورت زیر تولید می‌کنیم.



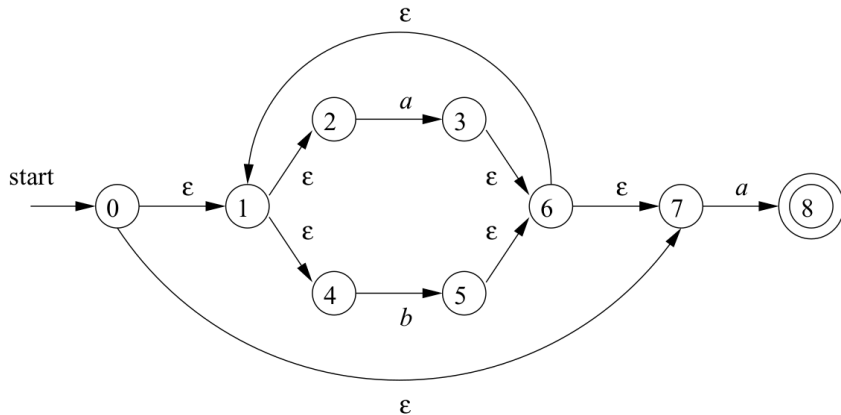
تبدیل عبارت منظم به ماشین غیرقطعی



تبدیل عبارت منظم به ماشین غیرقطعی



تبدیل عبارت منظم به ماشین غیرقطعی



کارایی الگوریتم‌های پردازش رشته

- برای شبیه‌سازی یک ماشین متناهی قطعی برای رشته x به زمان $O(|x|)$ نیاز داریم، درحالی که شبیه‌سازی یک ماشین متناهی غیرقطعی با تعداد حالت‌های s در زمان $O(|x|s)$ انجام می‌شود.
- بنابراین شبیه‌سازی، ماشین قطعی بسیار سریع‌تر انجام می‌گیرد. مشکلی که وجود دارد این است که تبدیل ماشین غیرقطعی به ماشین قطعی در بدترین حالت به زمان نهایی نیاز دارد.

کارایی الگوریتم‌های پردازش رشته

- معمولاً تحلیل‌گرهای لغوی با این انتخاب روبرو می‌شوند که عبارت منظم را به ماشین قطعی تبدیل کنند و یا از ماشین غیرقطعی برای تحلیل لغوی استفاده کنند.
- این انتخاب بستگی به کاربرد تحلیل‌گر لغوی دارد. اگر می‌خواهیم یک عبارت منظم را به یک تحلیل‌گر لغوی تبدیل کنیم و از تحلیل‌گر لغوی به کرات استفاده کنیم، هزینه‌ای که برای تبدیل ماشین غیرقطعی به ماشین قطعی صرف می‌کنیم تنها یک بار صرف می‌شود. اما گاهی از تحلیل‌گر لغوی تنها یک بار استفاده می‌کنیم. برای مثال در برنامه `grep` در لینوکس یک عبارت منظم برای جستجوی فایل‌ها استفاده می‌شود. در چنین مواردی ماشین غیرقطعی مستقیماً اجرا می‌شود.
- همچنین الگوریتمی برای کاهش تعداد حالات ماشین قطعی وجود دارد که در مبحث نظریه زبان‌ها و ماشین‌ها به آن پرداخته شده است.