به نام خدا

طراحی کامپایلر

آرش شفیعی

تحلیل لغوی

تحلیل‌گر 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 لغوی [1] نامیده می‌شود.
- استفاده از یک ابزار تولید کننده 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. جزئیات پیاده‌سازی بر عهده تولیدکننده تحلیل‌گر لغوی خواهد بود.

[1] lexical analyzer generator

تحلیل‌گر لغوی

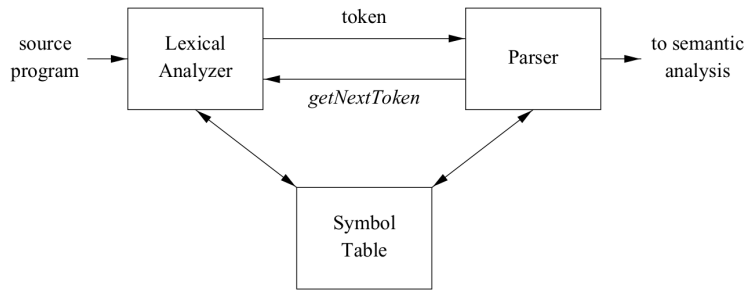
- یکی از مهم‌ترین ابزارهای تولید کننده تحلیل‌گر لغوی فلکس [1] نام دارد.
- برای توصیف تحلیل‌گر لغوی می‌توانیم از زبان‌های منظم [2] استفاده کنیم. برای هر زبان منظم می‌توان یک ماشین متناهی قطعی یا غیر قطعی ساخت و برای یک ماشین متناهی می‌توان کد تولید کرد.

[1] Flex / Lex
[2] regular languages

تحلیل‌گر لغوی

- تحلیل‌گر لغوی کاراکترهای ورودی را یک‌به‌یک دریافت کرده، آنها را دسته‌بندی و به واژه [1] تبدیل می‌کند. خروجی یک تحلیل‌گر لغوی دنباله‌ای است از واژه‌های برنامهٔ مبدأ. این دنباله از واژه‌ها برای تحلیل نحوی به تحلیل‌گر نحوی یا پارسر ارسال می‌شود. تحلیل‌گر لغوی همچنین هنگام تشخیص یک شناسهٔ 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

- تحلیل‌گر لغوی همچنین وظیفه دارد توضیحات [1] و فاصله‌های خالی [2] 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.
- 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

تحلیل‌گر لغوی

- یک توکن [1] یک دوتایی است شامل نام توکن و ویژگی‌های آن. نام توکن در واقع نوع آن را نشان می‌دهد.
- یک الگو [2] توصیفی است از شکلی که واژه‌های مربوط به توکن‌ها می‌توانند داشته باشند.
- یک واژه [3] 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 منطبق [4] می‌شوند.

[1] token
[2] pattern
[3] lexeme
[4] match

تحلیل‌گر لغوی

- در جدول زیر تعدادی توکن رایج در زبان‌های برنامه‌نویسی توصیف 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"`

تحلیل‌گر لغوی

- 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.

تحلیل‌گر لغوی

- یکی از ویژگی‌هایی که برای همه توکن‌ها می‌توان ذخیره کرد، شمارهٔ خط توکن در برنامهٔ 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، و اولین خطی که شناسه در آن رؤیت شده است می‌شود. ویژگی‌های شناسه‌ها معمولاً در جدول علائم نگهداری می‌شود، بنابراین ویژگی توکن `id` می‌تواند اشاره‌گری باشد به آن شناسه در جدول علائم.

تحلیل‌گر لغوی

- در عبارت `E = M * C ** 2` در زبان فورترن توکن‌ها به صورت زیر استخراج می‌شوند.

<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>

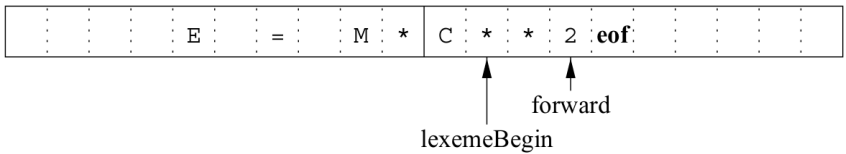
- در اینجا توکن‌های متناظر با عملگرها هیچ ویژگی ندارند.

تحلیل‌گر لغوی

- تحلیل‌گر لغوی به تنهایی قادر نیست خطاهای برنامه ورودی را تشخیص دهد. برای مثال فرض کنید در قسمتی از برنامه مبدأ داشته باشیم `((a==f(x)) fi` . در اینجا تحلیل‌گر لغوی نمی‌تواند تشخیص دهد آیا `fi` همان `if` است که به اشتباه نوشته شده است و یا `fi` 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.

تحلیل‌گر لغوی

- از آنجایی که رشته ورودی می‌تواند طولانی باشد و نمی‌توان همهٔ ورودی را در حافظه قرار داد و همچنین خواندن کاراکترها یک‌به‌یک از دیسک زمان بر است، معمولاً یک قطعه از برنامه از دیسک خوانده می‌شود و در حافظه قرار داده می‌شود. همچنین در هنگام خواندن از روی حافظه، کامپایلر معمولاً قسمت دیگری از حافظه را پر می‌کند تا به محض به اتمام رسیدن بافر اول، بافر دوم محتوای کافی برای پردازش داشته باشد.
- یک اشاره‌گر به کاراکتری از ورودی که در حال خوانده شدن است اشاره می‌کند و یک اشاره‌گر در رشته ورودی جلو می‌رود تا یک توکن تشخیص داده شود.



توصیف توکن‌ها

- برای تشخیص الگوهای توکن‌ها از عبارت‌های منظم استفاده می‌کنیم. عبارت‌های منظم معمولاً در تولید کننده‌های تحلیل‌گر لغوی استفاده می‌شوند.
- یک الفبا [1] مجموعه‌ای است محدود از نمادها و یا سمبول‌ها [2] . برای مثال الفبای $\Sigma = \{0, 1\}$ از دو نماد صفر و یک تشکیل شده است که می‌توان با این الفبا همه اعداد دودویی را تشکیل داد.
- مجموعهٔ حروف اسکی [3] یا مجموعه حروف یونیکد [4] دو مثال دیگر از الفبا هستند.

[1] alphabet
[2] symbols
[3] ASCII
[4] Unicode

توصیف توکن‌ها

- یک رشته [1] بر روی یک الفبا دنباله‌ای است محدود از نمادهای الفبا. در نظریه زبان‌ها به یک رشته از الفبا یک جمله [2] نیز گفته می‌شود. طول رشته s را با |s| نمایش می‌دهیم که درواقع تعداد نمادهای رشته است. یک رشتهٔ تهی را با نماد ϵ نمایش می‌دهیم که طول آن صفر است.
- یک زبان [3] مجموعه‌ای است شمارا از رشته‌هایی برروی یک الفبای معین. برای مثال مجموعهٔ همه برنامه‌های درست در زبان سی، زبان سی را تشکیل می‌دهند. همچنین مجموعهٔ 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 در مورد معانی جملات صحبت نمی‌کنیم.

[1] string
[2] sentence
[3] language

توصیف توکن‌ها

- اگر دو رشته x و y را به یکدیگر الحاق [1] کنیم، رشته‌ای 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

توصیف توکن‌ها

- یک پیشوند [1] از رشته s رشته‌ای است که از حذف صفر یا تعدادی نمادهای s از انتهای آن به دست می‌آید.
- یک پسوند [2] از رشته s رشته‌ای است که از حذف صفر یا تعداد بیشتری نمادهای s را ابتدای آن به دست می‌آید.
- یک زیر رشته [3] از رشته s با حذف یک پیشوند و یک پسوند از s به دست می‌آید.
- یک پیشوند، پسوند، یا زیر رشته کامل [4] ، پیشوند، پسوند یا زیررشته‌ای است که تهی نباشد.
- یک زیر دنباله [5] از رشته s رشته‌ای است که با حذف صفر یا تعداد بیشتری از نمادها از رشته به دست آید.

---

[1] prefix
[2] suffix
[3] substring
[4] proper prefix، suffix، substring
[5] subsequence

توصیف توکن‌ها

- مهم‌ترین عملیات برروی زبان‌ها عملیات اجتماع [1] ، الحاق [2] و بستار [3] 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

توصیف توکن‌ها

- بستار ستاره یا بستار کلینی [1] برروی زبان L که به صورت L^* نشان داده می‌شود، مجموعه‌ای است از همهٔ 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 مثبت [2] 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

توصیف توکن‌ها

- جدول زیر عملگرهای رایج برروی زبان‌ها را نشان می‌دهد.

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

عبارات منظم

- جملات یک زبان منظم را می‌توان توسط یک عبارت منظم [1] توصیف کرد. این توصیف با استفاده از عملگرهای اجتماع، الحاق و بستار به دست می‌آید.
- برای مثال فرض کنید letter_ مجموعه‌ای از همهٔ حروف زبان انگلیسی و علامت زیرخط [2] باشد و digit مجموعه‌ای از همهٔ ارقام 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، علامت زیرخط و ارقام الحاق شده است.

[1] regular expression
[2] underline

عبارات منظم

- یک عبارت منظم r زبان $L(r)$ را توصیف می‌کند. قوانینی که توسط آن عبارت منظم r زبان $L(r)$ را توصیف می‌کند به شرح زیر است.

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

- اگر r و s دو عبارت منظم باشند، آنگاه:

۱. عبارت $r|s$ 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.

عبارات منظم

- عملگر بستار بالاترین اولویت را دارد. عملگر الحاق در مرتبه دوم اولویت قرار دارد و عملگر اجتماع پایین‌ترین اولویت را داراست.
- عبارت منظم $(a)|((b)^*(c))$ برابراست با $a|b^*c$ 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 منظم [1] نیز نامیده می‌شود. اگر دو عبارت منظم r و s یک مجموعه را توصیف کنند می‌گوییم r و s معادل یکدیگرند و می‌نویسیم $r = s$. برای مثال $(a|b) = (b|a)$.

[1] regular set

عبارات منظم

- تعدادی قوانین جبری برای عبارات منظم وجود دارند که به شرح زیر اند:
- اجتماع جابجایی‌پذیر [1] و شرکت‌پذیر [2] است.
- الحاق شرکت‌پذیر و بر روی اجتماع توزیع‌پذیر [3] است.
- رشتهٔ تهی در الحاق و بستار ستاره عضو خنثی است.
- عملگر بستار ستاره بر روی تکرار خنثی است یا به عبارت دیگر خودتوان [4] است.

[1] commutative
[2] associative
[3] distributive
[4] idempotent

عبارات منظم

- قوانین جبری عبارات منظم 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

تعاریف منظم

- 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
$d_1 \to r_1, d_2 \to r_2, \cdots, d_n \to r_n$
به طوری که هر یک از d_i ها یک نماد است که در $\sum$ نیست و با بقیه d_i ها متفاوت است و هر یک از r_i ها یک عبارت منظم بر روی الفبای $\sum \cup \{d_1, d_2, \cdots, d_n\}$ است.

- برای جلوگیری از تعاریف بازگشتی، تعاریف را به گونه‌ای محدود می‌کنیم که r_i بتواند تنها از d_i هایی که از قبل تعریف شده‌اند، استفاده کند.

تعاریف منظم

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

$$
\begin{aligned}
letter\_ &\rightarrow \texttt{A} \mid \texttt{B} \mid \cdots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \mid \texttt{\_} \\
digit &\rightarrow \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
id &\rightarrow letter\_ \; (\; letter\_ \mid digit \;)^*
\end{aligned}
$$

تعاریف منظم

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

$$
\begin{aligned}
\mathit{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
\mathit{digits} &\rightarrow \mathit{digit}\ \mathit{digit}^* \\
\mathit{optionalFraction} &\rightarrow \texttt{.}\ \mathit{digits} \mid \epsilon \\
\mathit{optionalExponent} &\rightarrow (\ \texttt{E}\ (\ \texttt{+} \mid \texttt{-} \mid \epsilon\ )\ \mathit{digits}\ ) \mid \epsilon \\
\mathit{number} &\rightarrow \mathit{digits}\ \mathit{optionalFraction}\ \mathit{optionalExponent}
\end{aligned}
$$

تعمیم عبارات منظم

- عبارات منظم در دههٔ ۱۹۵۰ توسط کلین [1] با سه عملگر اجتماع، الحاق و بستار معرفی شدند.
- پس از آن عبارات منظم برای افزایش توانایی بیان عبارات، تعمیم داده شدند.
- بستار مثبت [2] 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 [3] 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.

[1] Kleene
[2] positive closure
[3] zero or one

تعمیم عبارات منظم

- گروه یا کلاس نمادها [1] یک مخفف است که دسته‌بندی نمادها به کار می‌رود. به جای عبارت $a_1|a_2|\cdots|a_n$ می‌نویسیم $[a_1\ a_2\ \cdots\ a_n]$. همچنین در صورتی که یک ترتیب منطقی در نمادها وجود داشته باشد می‌نویسیم $[a_1 - a_n]$. برای مثال $[a - z]$ معادل است با $a|b|c|\cdots|z$.

[1] character class

تعمیم عبارات منظم

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

$$
\begin{aligned}
letter\_ &\rightarrow [\texttt{A-Za-z\_}] \\
digit &\rightarrow [\texttt{0-9}] \\
id &\rightarrow letter\_ \ (\ letter\_ \mid digit\ )^*
\end{aligned}
$$

تعمیم عبارات منظم

- اعداد را می‌توانیم به صورت زیر نمایش دهیم.

$$
\begin{aligned}
\mathit{digit} &\rightarrow [\texttt{0-9}] \\
\mathit{digits} &\rightarrow \mathit{digit}^{+} \\
\mathit{number} &\rightarrow \mathit{digits}\ (\texttt{.}\ \mathit{digits})?\ (\ \texttt{E}\ [\texttt{+-}]?\ \mathit{digits}\ )?
\end{aligned}
$$

شناسایی توکن‌ها

- 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.

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & | & \epsilon \\
expr & \rightarrow & term \textbf{ relop } term \\
 & | & term \\
term & \rightarrow & \textbf{id} \\
 & | & \textbf{number}
\end{array}
$$

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 `if` ، `else` ، `relop` ، `id` و `number` می‌خواهیم با استفاده از تعاریف منظم این ترمینال‌ها را توصیف کنیم.
- الگوهای ترمینال‌ها در این الگوریتم به صورت زیر هستند.

$$
\begin{array}{rcl}
\mathit{digit} & \rightarrow & \texttt{[0-9]} \\
\mathit{digits} & \rightarrow & \mathit{digit}^{+} \\
\mathit{number} & \rightarrow & \mathit{digits}\ (\ \texttt{.}\ \mathit{digits})?\ (\ \texttt{E}\ \texttt{[+-]}?\ \mathit{digits}\ )? \\
\mathit{letter} & \rightarrow & \texttt{[A-Za-z]} \\
\mathit{id} & \rightarrow & \mathit{letter}\ (\ \mathit{letter} \mid \mathit{digit}\ )^{*} \\
\mathit{if} & \rightarrow & \texttt{if} \\
\mathit{then} & \rightarrow & \texttt{then} \\
\mathit{else} & \rightarrow & \texttt{else} \\
\mathit{relop} & \rightarrow & \texttt{<} \mid \texttt{>} \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{=} \mid \texttt{<>}
\end{array}
$$

شناسایی توکن‌ها

- 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.

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

- 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 چشم پوشی می‌کند و کار شناسی توکن بعدی را آغاز می‌کند.

شناسایی توکن‌ها

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

LEXEMES	TOKEN NAME	ATTRIBUTE VALUE
Any *ws*	–	–
`if`	**if**	–
`then`	**then**	–
`else`	**else**	–
Any *id*	**id**	Pointer to table entry
Any *number*	**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 گذار [1] تبدیل می‌کنیم.
- ابتدا به صورت دستی عبارات 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 حالت‌های [2] دیاگرام نامیده می‌شوند.

[1] transition diagram
[2] state

دیاگرام‌های گذار

- 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 برچسب [1] 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] label

دیاگرام‌های گذار

- در ابتدا فرض می‌کنیم دیاگرام حالت قطعی [1] 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 نهایی [2] یا حالت پذیرنده [3] 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

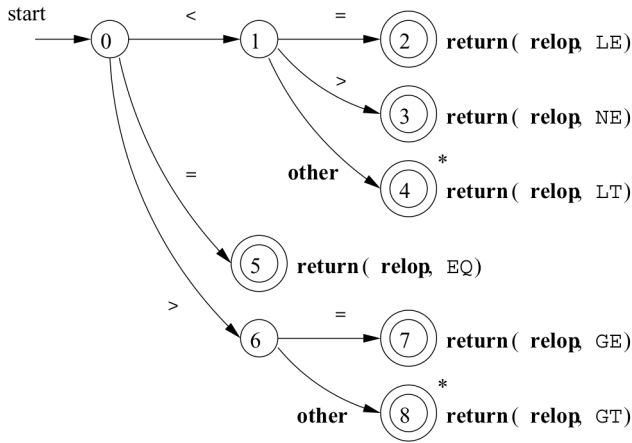
دیاگرام‌های گذار

- یکی از حالت‌ها، حالت اولیه [1] نامیده می‌شود. این حالت را با یک یال ورودی بدون مبدأ به نام `start` مشخص می‌کنیم. یک دیاگرام گذار 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

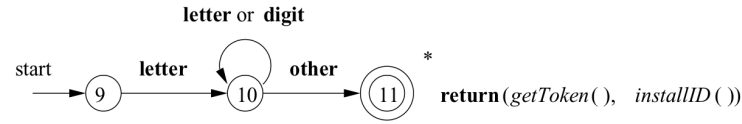
دیاگرام‌های گذار

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



کلمات کلیدی و شناسه‌ها

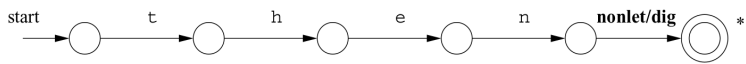
- برای شناسایی و استخراج شناسه‌ها می‌توانیم از یک دیاگرام گذار مانند دیاگرام زیر استفاده کنیم.



- 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 شد، 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.

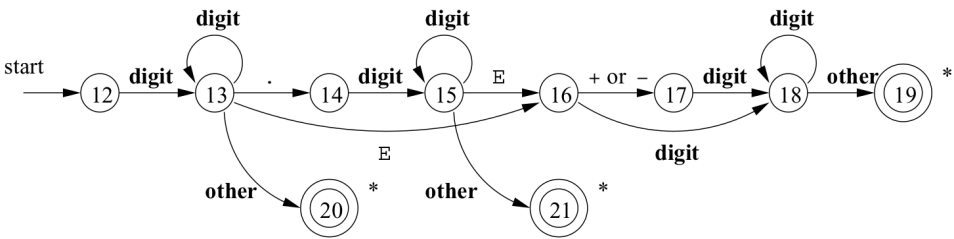
کلمات کلیدی و شناسه‌ها

- یک راه‌حل 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 می‌شود. البته این راه‌حل برای پیاده‌سازی پیچیده‌تر است.



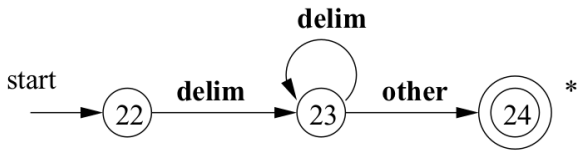
شناسایی اعداد

- دیاگرام زیر برای شناسایی اعداد می‌تواند مورد استفاده قرار بگیرد.



شناسایی فاصله‌های خالی

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



تحلیل‌گر لغوی بر مبنای دیاگرام گذار

- برای پیاده‌سازی یک دیاگرام گذار می‌توانیم یک متغیر به نام `state` تعریف کنیم که 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);
        }
    }
}
```

تحلیل‌گر لغوی بر مبنای دیاگرام گذار

- از این تابع در تحلیل‌گر معنایی به چند روش می‌توانیم استفاده کنیم.
- یک روش این است که به ازای هر نوع توکن تابع مربوطه فراخوانی شود تا این که رشته ورودی بر یک نوع توکن منطبق شود.
- روش دوم این است که یک دیاگرام گذار کلی برای همه توکن‌ها طراحی کرده و این دیاگرام گذار را پیاده‌سازی کنیم. مشکلی که در این روش وجود دارد این است که با ترکیب دیاگرام گذار همهٔ توکن‌ها در یک دیاگرام، یک دیاگرام غیر قطعی تولید شود. دیاگرام غیرقطعی دیاگرامی است که در آن از یک حالت و خواندن یک نماد امکان تغییر حالت به بیش از یک حالت وجود داشته باشد. همچنین در یک دیاگرام گذار غیرقطعی امکان تغییر حالت با خواندن کاراکتر تهی وجود دارد. در چنین مواردی دیاگرام غیرقطعی را باید به دیاگرام قطعی تبدیل کنیم.

ماشین‌های متناهی

- یکی از ابزارهایی که برای تولید تحلیل‌گر لغوی استفاده می‌شود، ابزار لکس [1] است که نسخه جدیدتر آن فلکس [2] 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 متناهی [3] استفاده می‌شود.

[1] Lex
[2] Flex
[3] Finite Automata

ماشین‌های متناهی

- ماشین‌های متناهی ابزارهایی برای تشخیص [1] 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 یا خیر است.
- دو نوع ماشین متناهی وجود دارد : ماشین متناهی غیر قطعی [2] و ماشین متناهی قطعی [3] .

[1] recognize
[2] Nondeterministic finitive automata (NFA)
[3] Deterministic finitive automata (DFA)

ماشین‌های متناهی

- ماشین متناهی را می‌توان با استفاده از یک گراف گذار نمایش داد که رئوس آن حالت‌های ماشین هستند و یال‌های آن گذارها از یک حالت به حالت دیگر.
- 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 می‌پذیرد [1] ، در غیراینصورت ماشین رشته را نمی‌پذیرد یا رد می‌کند [2].

[1] accept
[2] reject

ماشین‌های متناهی

- ماشین متناهی غیرقطعی برروی یال‌های خود می‌تواند هر برچسبی داشته باشد. یال‌ها همچنین می‌توانند با نماد ϵ برچسب‌گذاری 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 برچسب a وجود داشته باشد. همچنین هیچ یالی با برچسب ϵ در ماشین متناهی قطعی وجود ندارد.
- ماشین‌های متناهی قطعی و غیرقطعی قادر به شناسایی زبان‌های منظم هستند.

ماشین‌های متناهی غیرقطعی

- یک ماشین متناهی غیرقطعی 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 هرحالت و هرنماد در $\sum \cup \{\epsilon\}$ حالت بعدی [4] را بازمی‌گرداند.
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 غیرقطعی

- ماشین‌های متناهی قطعی و غیرقطعی را می‌توانیم با یک گراف گذار [1] نشان دهیم به طوری‌که 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.

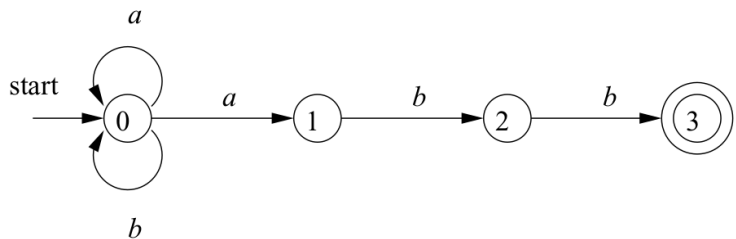
[1] transition graph

ماشین‌های متناهی غیرقطعی

- یک ماشین متناهی غیرقطعی برای تشخیص زبان منظم $L((a|b)^*abb)$ طراحی کنید.

ماشین‌های متناهی غیرقطعی

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



جداول گذار

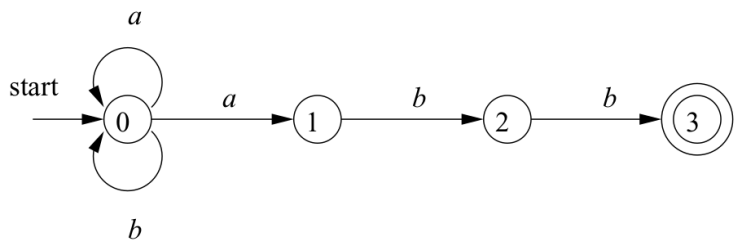
- یک ماشین متناهی غیرقطعی را می‌توانیم توسط جدول گذار [1] نیز نمایش دهیم که در آن هر سطر متعلق به یک حالت و هر ستون متعلق به یک نماد از الفبا یا نماد ϵ است. یک خانه در این جدول در سطر p و ستون a مجموعهٔ حالت‌های بعدی p را با رؤیت نماد a مشخص می‌کند.

[1] transition table

جداول گذار

- جدول گذار زیر، ماشین متناهی غیرقطعی رسم شده را توصیف می‌کند.

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



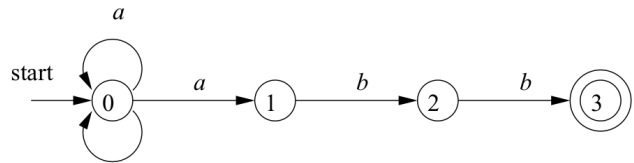
پذیرش یک رشته ورودی

- یک ماشین متناهی غیرقطعی، رشته x را می‌پذیرد [1] 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.

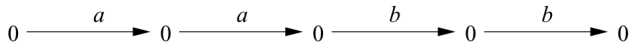
[1] accepts

پذیرش یک رشته ورودی

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



- همچنین مسیر دیگری برای رشته aabb وجود دارد که البته این مسیر به یک حالت نهایی منتهی نمی‌شود.

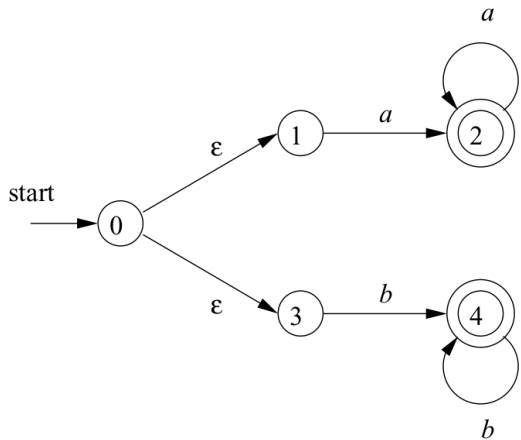


پذیرش یک رشته ورودی

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

پذیرش یک رشته ورودی

- ماشین متناهی غیرقطعی زیر زبان $L(aa^* \mid bb^*)$ را می‌پذیرد.



ماشین‌های متناهی قطعی

- 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 خانه‌های آن مقدار دارند و ستونی با نماد ϵ وجود ندارد و همچنین هر خانه از این جدول تنها یک حالت را در برمی‌گیرد.

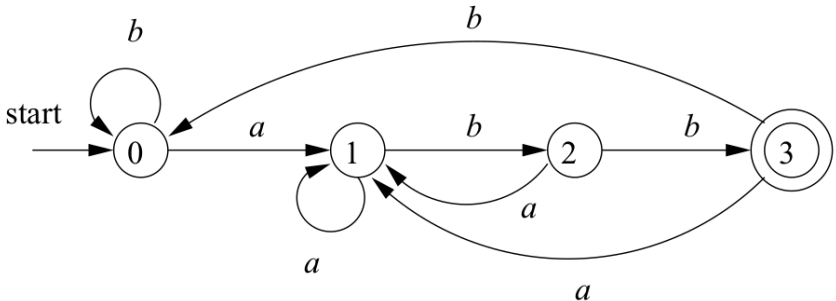
ماشین‌های متناهی قطعی

- 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 اتمام رشته، ماشین در حالت پایانی قرار گرفت، رشته پذیرفته می‌شود.

$$
\begin{array}{l}
s = s_0; \\
c = nextChar(); \\
\textbf{while } (\ c \mathrel{!}= \textbf{eof}\ ) \ \{ \\
\qquad s = move(s, c); \\
\qquad c = nextChar(); \\
\} \\
\textbf{if } (\ s \text{ is in } F\ ) \textbf{ return } \texttt{"yes"}; \\
\textbf{else return } \texttt{"no"};
\end{array}
$$

ماشین‌های متناهی قطعی

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



تبدیل عبارت منظم به ماشین

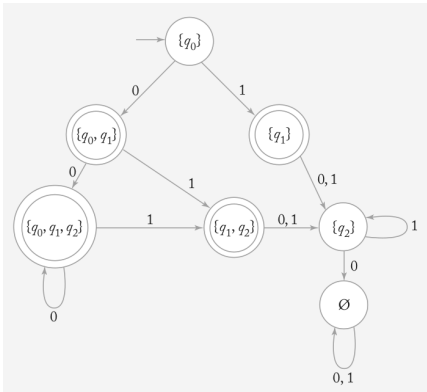
- تولیدکننده‌های تحلیل‌گر لغوی عباراتی به زبان منظم دریافت کرده، به صورت خودکار یک تحلیل‌گر لغوی تولید می‌کنند. برای انجام این کار باید الگوریتمی وجود داشته باشد که عبارات منظم را به یک برنامه تبدیل کند.
- تا اینجا روندی برای تبدیل یک ماشین متناهی قطعی به کد ارائه کردیم. به عبارت دیگر الگوریتمی وجود 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 مستقیما 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 $a_1\ a_2\ \cdots\ a_n$ از ورودی، ماشین قطعی در حالتی قرار می‌گیرد که متناظر با مجموعه‌ای از حالات ماشین غیرقطعی است که با شروع از حالت اولیه و خواندن رشته $a_1\ a_2\ \cdots\ a_n$ در ماشین غیرقطعی قابل دسترسی هستند.
- این امکان وجود دارد که تعداد حالات ماشین قطعی تولید شده، از مرتبه نمایی 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، تعداد حالات ماشین‌های غیرقطعی و ماشین‌های قطعی معادل آنها تقریبا برابر است و بنابراین در عمل بدترین حالت اتفاق نمی‌افتد و در زمان غیرنمایی می‌توان یک ماشین متناهی غیرطبیعی را به ماشین معادل قطعی تبدیل کرد.

هم‌ارزی ماشین‌های متناهی قطعی و غیرقطعی

- ماشین‌های متناهی قطعی و غیرقطعی زیر معادل یکدیگرند.



تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- فرض کنید می‌خواهیم ماشین متناهی غیرقطعی 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 می‌سازیم. هریک از حالات ماشین D مجموعه‌ای است از حالات ماشین N . حال Dtran را به گونه‌ای تکمیل می‌کنیم 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
ϵ-*closure*(s)	Set of NFA states reachable from NFA state s on ϵ-transitions alone.
ϵ-*closure*(T)	Set of NFA states reachable from some NFA state s in set T on ϵ-transitions alone; $= \cup_{s \text{ in } T}\ \epsilon$-*closure*$(s)$.
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-closure را می‌توانیم به صورت زیر توصیف کنیم.

```
push all states of T onto stack;
initialize ε-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 ε )
                if ( u is not in ε-closure(T) ) {
                        add u to ε-closure(T);
                        push u onto stack;
                }
}
```

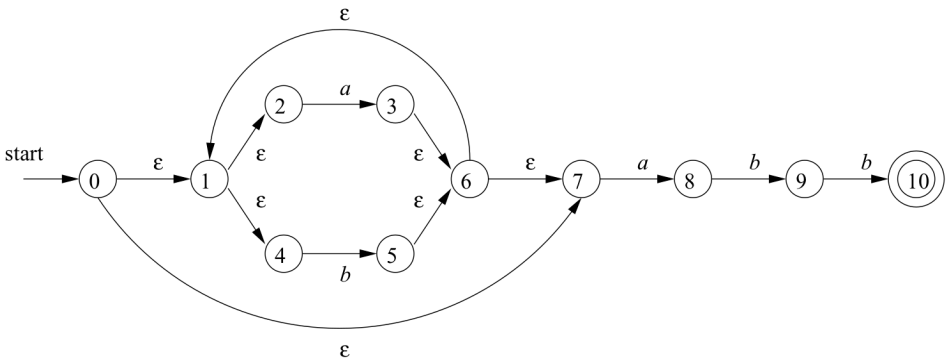
تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

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

initially, ϵ-$closure(s_0)$ 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) {
        $U = \epsilon$-$closure\big(move(T, a)\big)$;
        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 ϵ-closure(0) به دست می‌آید برابراست با $A = \{0, 1, 2, 4, 7\}$
- همچنین داریم :

$$Dtans[A،a] = \epsilon\text{-closure}(move\ (A،a)) = \epsilon\text{-closure}(\{3, 8\}) = \{0, 1, 2, 4, 6, 7, 8\} = B$$
$$Dtans[A،b] = \epsilon\text{-closure}(move\ (A،b)) = \epsilon\text{-closure}(\{5\}) = \{0, 1, 2, 4, 5, 6, 7\} = C$$

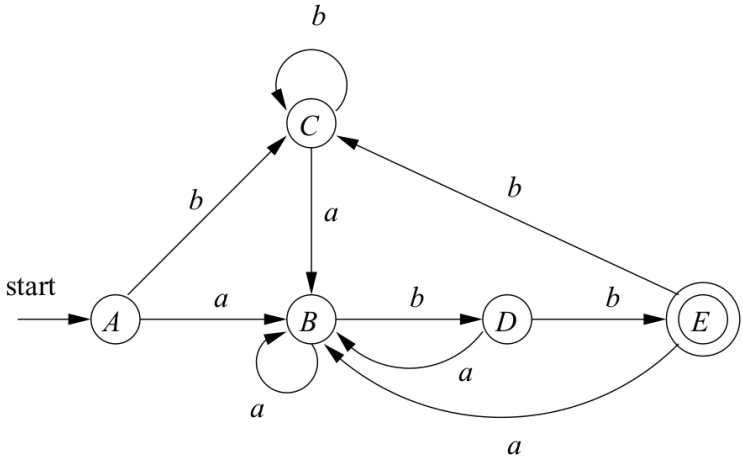
تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

- بدین ترتیب می‌توانیم جدول زیر را تشکیل دهیم.

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

تبدیل ماشین متناهی غیرقطعی به ماشین متناهی قطعی

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



شبیه سازی ماشین‌های متناهی قطعی

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

شبیه سازی ماشین‌های متناهی قطعی

- الگوریتم بدین صورت عمل می‌کند که مجموعه‌ای از حالات فعلی به نام S را نگهداری می‌کند که با شروع از s_0 و خواندن رشته تا به حال خوانده شده قابل دسترسی هستند. با خواندن کاراکتر ورودی c مجموعهٔ همهٔ حالاتی که از حالات فعلی و خواندن کاراکتر c قابل دسترسی هستند محاسبه می‌شوند و مجموعه حالات فعلی تغییر می‌کند.

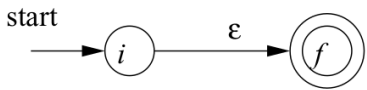
```
1)  S = ε-closure(s0);
2)  c = nextChar();
3)  while ( c != eof ) {
4)          S = ε-closure(move(S, c));
5)          c = nextChar();
6)  }
7)  if ( S ∩ F != ∅ ) return "yes";
8)  else return "no";
```

تبدیل عبارت منظم به ماشین غیرقطعی

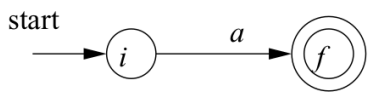
- حال می‌خواهیم الگوریتمی طراحی کنیم که یک عبارت منظم دلخواه را به یک ماشین متناهی غیر قطعی تبدیل کند. فرض کنید عبارت منظم r بر روی الفبای $\sum$ تعریف شده است. می‌خواهیم ماشین غیر قطعی N را طراحی کنیم که زبان $L(r)$ را بپذیرد.
- در این الگوریتم ابتدا به ازای هر زیر عبارت یک ماشین تولید می‌شود و سپس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

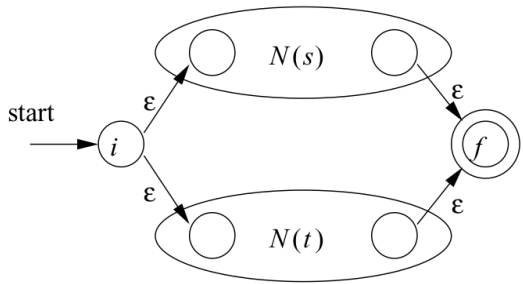


- در اینجا i یک حالت غیرپایانی جدید و f یک حالت پایانی جدید است.
- ماشین غیرقطعی متناظر هر زیر عبارت a در $\sum$ به صورت زیر است.



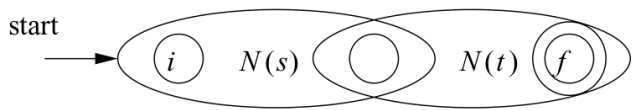
تبدیل عبارت منظم به ماشین غیرقطعی

- 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.



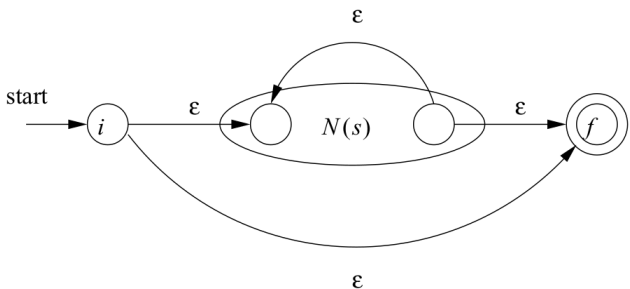
تبدیل عبارت منظم به ماشین غیرقطعی

- 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.

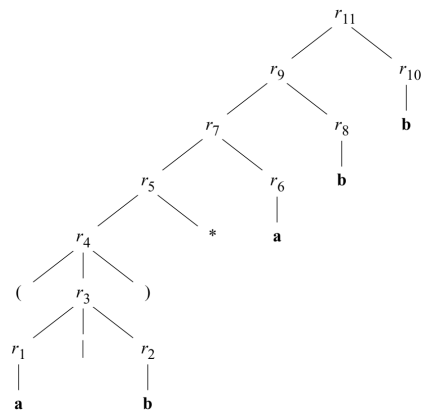


تبدیل عبارت منظم به ماشین غیرقطعی

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

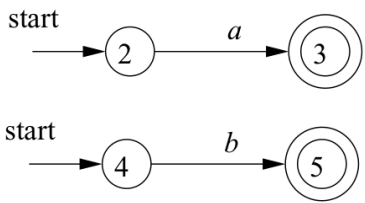
تبدیل عبارت منظم به ماشین غیرقطعی

- 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.



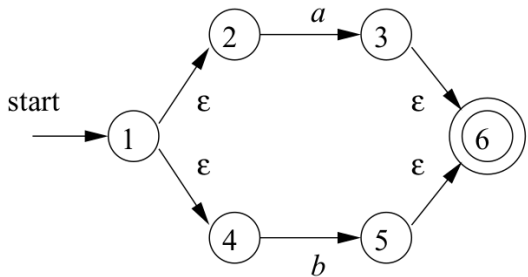
تبدیل عبارت منظم به ماشین غیرقطعی

- حال ماشین غیرقطعی را به صورت زیر تولید می‌کنیم. ابتدا دو ماشین متناهی برای عبارات منظم a و b می‌سازیم.



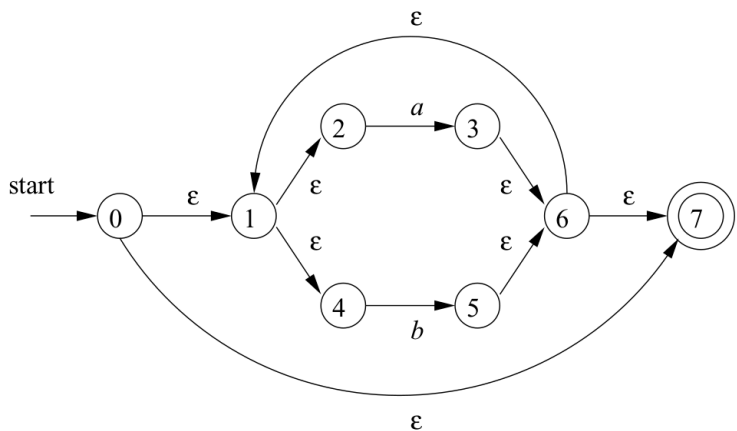
تبدیل عبارت منظم به ماشین غیرقطعی

- سپس یک ماشین برای عبارت a|b می‌سازیم. همچنین ماشین عبارات a|b و (a|b) یکسان اند.



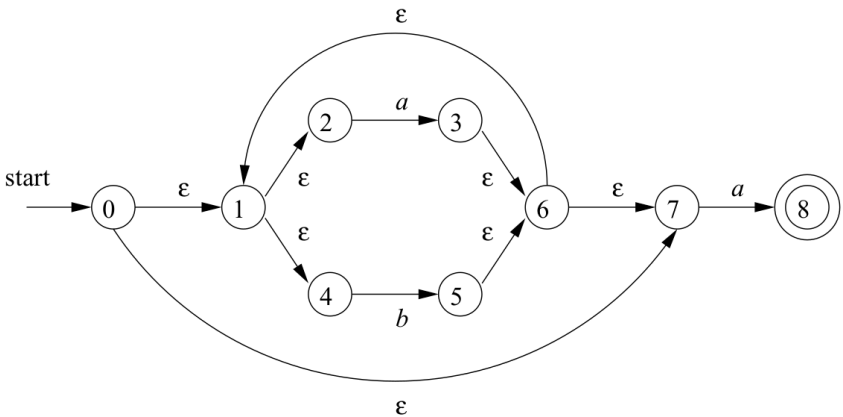
تبدیل عبارت منظم به ماشین غیرقطعی

- در گام بعد ماشینی برای عبارت $(a|b)^*$ می‌سازیم.



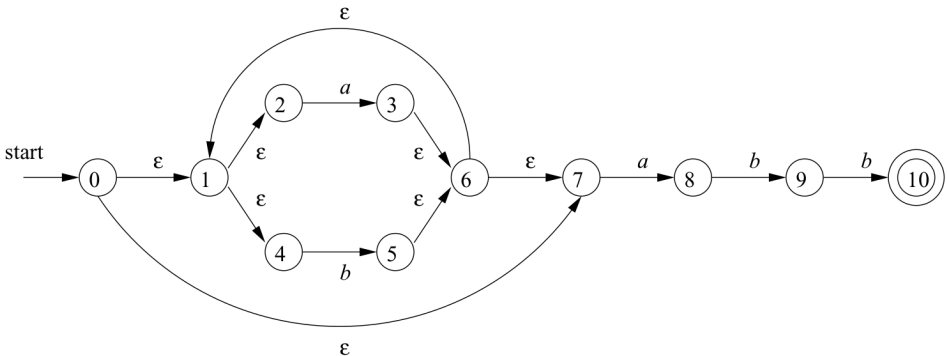
تبدیل عبارت منظم به ماشین غیرقطعی

- سپس ماشین عبارت $(a|b)^*a$ به صورت زیر ساخته می‌شود.



تبدیل عبارت منظم به ماشین غیرقطعی

- در نهایت ماشین غیرقطعی معادل عبارت $(a|b)^*abb$ به صورت زیر به دست می‌آید.



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 نمایی 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 یک‌بار استفاده می‌کنیم. برای مثال در برنامه `grep` در لینوکس یک عبارت منظم برای جستجوی فایل‌ها استفاده می‌شود. در چنین مواردی ماشین غیرقطعی مستقیما اجرا می‌شود.
- همچنین الگوریتمی برای کاهش تعداد حالات ماشین قطعی وجود دارد که در ادامه توضیح داده خواهد شد.

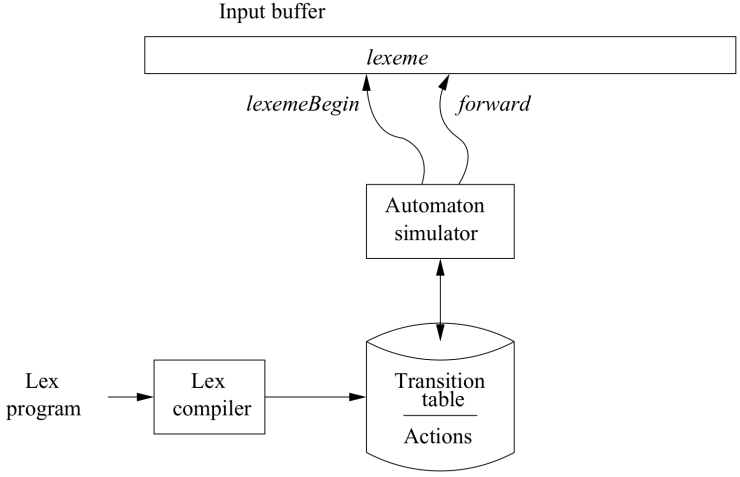
طراحی تولید کننده تحلیل‌گر لغوی

- می‌خواهیم بررسی کنیم چگونه یک تولید کننده تحلیل‌گر لغوی [1] مانند لکس [2] ساخته شده است.

[1] lexical analyzer generator
[2] Lex

طراحی تولید کننده تحلیل‌گر لغوی

- شکل زیر ساختار یک تحلیل‌گر لغوی ساخته شده توسط لکس را نشان می‌دهد.



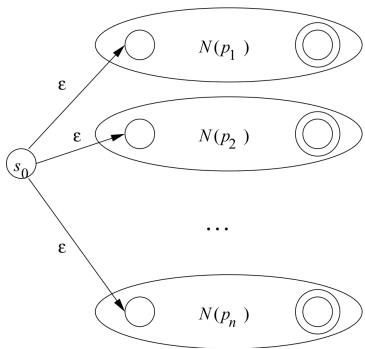
طراحی تولید کننده تحلیل‌گر لغوی

- یک برنامه در زبان لکس تشکیل شده است از تعدادی عبارت منظم. این عبارات منظم ابتدا توسط کامپایلر زبان لکس تبدیل به یک ماشین متناهی می‌شوند و ماشین متناهی به طور خودکار تبدیل به یک برنامه تحلیل‌گر لغوی می‌شود.
- برای تبدیل یک عبارت منظم به یک ماشین متناهی غیر قطعی از الگوریتمی که قبلا توصیف کردیم استفاده می‌شود.

طراحی تولید کننده تحلیل‌گر لغوی

- به ازای هر یک از الگوها در زبان منظم یک ماشین متناهی غیرقطعی ساخته می‌شود. در مرحله بعد برای ساختن یک ماشین متناهی واحد، همهٔ ماشین‌های غیرقطعی را توسط یک گذار تهی به یک حالت اولیه متصل می‌کنیم.

- فرض کنید برای الگوی p_i ماشین متناهی غیرقطعی $N(p_i)$ ساخته شده است. می‌توانیم ماشین زیر را برای شناسایی الگوها بسازیم.



طراحی تولید کننده تحلیل‌گر لغوی

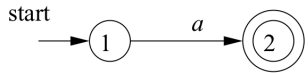
- فرض کنید سه الگوی زیر را در یک زبان داشته باشیم.

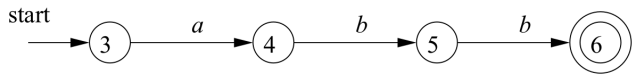
a	{عملیات A_1 برای الگوی p_1}
abb	{عملیات A_2 برای الگوی p_2}
a^*b^+	{عملیات A_3 برای الگوی p_3}

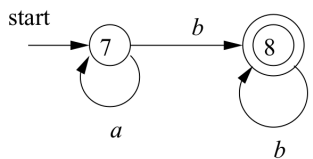
- توجه کنید که ممکن است این الگوها با یکدیگر اشتراکاتی دارند. در چنین شرایطی، عملیاتی که اولویت بالاتری دارد اجرا می‌شود. برای مثال رشته abb مطابق با الگوی دوم و سوم است اما عملیات A_2 انجام می‌شود زیرا اولویت آن از A_3 بالاتر است.

طراحی تولید کننده تحلیل‌گر لغوی

- سه ماشین غیرقطعی زیر برای این سه الگو ساخته می‌شوند.

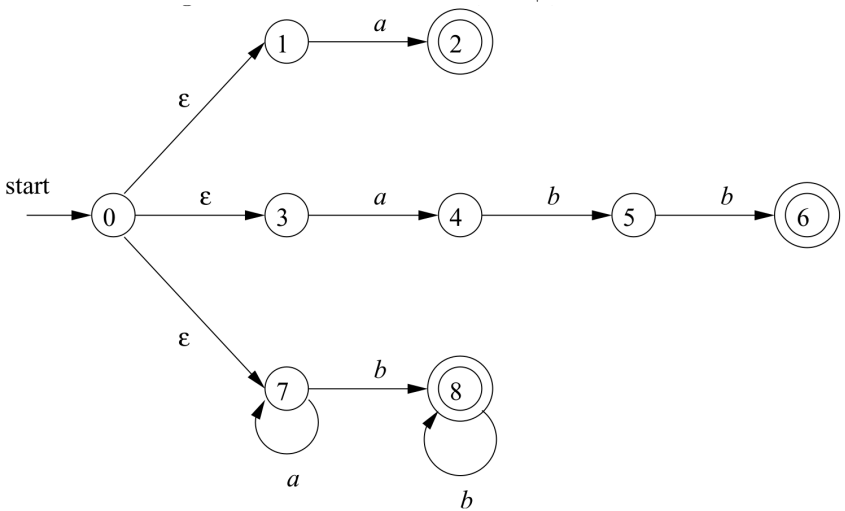






طراحی تولید کننده تحلیل‌گر لغوی

- سپس این سه ماشین در یک ماشین ادغام می‌شوند.

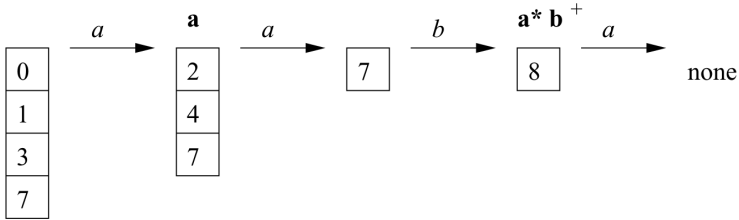


طراحی تولید کننده تحلیل‌گر لغوی

- حال تحلیل‌گر لغوی باید این ماشین غیرقطعی را شبیه‌سازی کند.
- یک اشاره‌گر به نام `lexemeBegin` به ابتدای توکن اشاره می‌کند و یک اشاره‌گر به نام `forward` در رشته ورودی به جلو حرکت می‌کند و با خواندن هر کاراکتر حالت‌های بعدی را محاسبه می‌کند. اگر با خواندن ورودی به نقطه‌ای رسیدیم که حالت‌های بعدی قابل محاسبه نبودند محاسبات متوقف می‌شود و توکنی که طول آن از توکن‌های تشخیص داده شده بیشتر است بازگردانده می‌شود. اگر چند الگو پذیرفته شوند، الگویی پذیرفته می‌شود که اولویت آن بالاتر باشد.

طراحی تولید کننده تحلیل‌گر لغوی

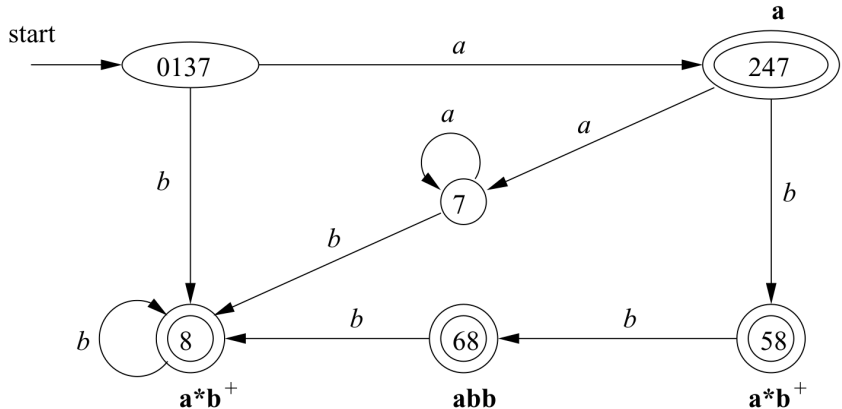
- فرض کنید سه الگوی a ، abb و a^*b^+ را داشته باشیم و ورودی با رشته $aaba$ آغاز شود.
- شکل زیر دنباله حالت‌هایی که به آنها وارد می‌شویم را نشان می‌دهد.



- بعد از خواندن چهار نماد از ورودی در مجموعه‌ای خالی از حالت‌ها قرار می‌گیریم پس هیچ گذار ممکنی بعد از آن وجود نخواهد داشت. در این شرایط باید به عقب بازگردیم و مجموعه‌ای از حالت‌ها را پیدا کنیم که یکی از آنها حالت پایانی باشد. بعد از خواندن aab در حالت ۸ قرار گرفته‌ایم که یک حالت پایانی است بنابراین aab به عنوان یک کلمه تشخیص داده می‌شود و عملیات A_3 انجام می‌شود.

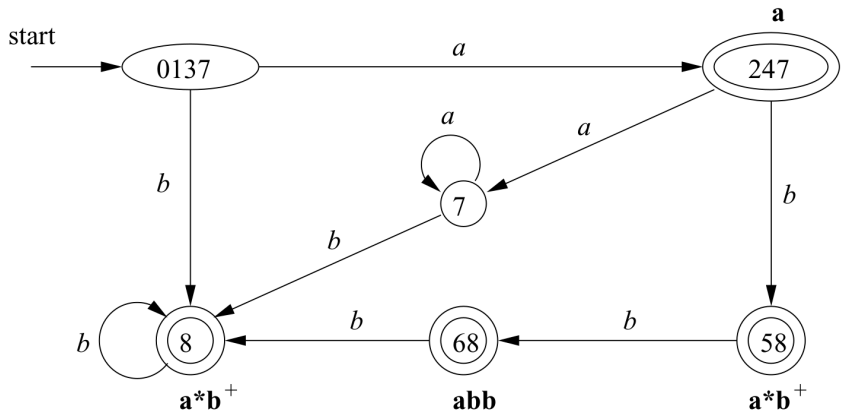
طراحی تولید کننده تحلیل‌گر لغوی

- یک روش دیگر برای استخراج توکن‌ها تبدیل ماشین غیرقطعی به ماشین قطعی است.
- شکل زیر ماشین قطعی معادل ماشین غیرقطعی قبل را نشان می‌دهد.



طراحی تولید کننده تحلیل‌گر لغوی

- با خواندن رشته abb ماشین وارد حالت {6, 8} می‌شود که متناظر دو الگوی abb و a^*b^+ است. الگوی abb اولویت بالاتری دارد بنابراین عملیات مربوط به آن اعمال می‌شود.



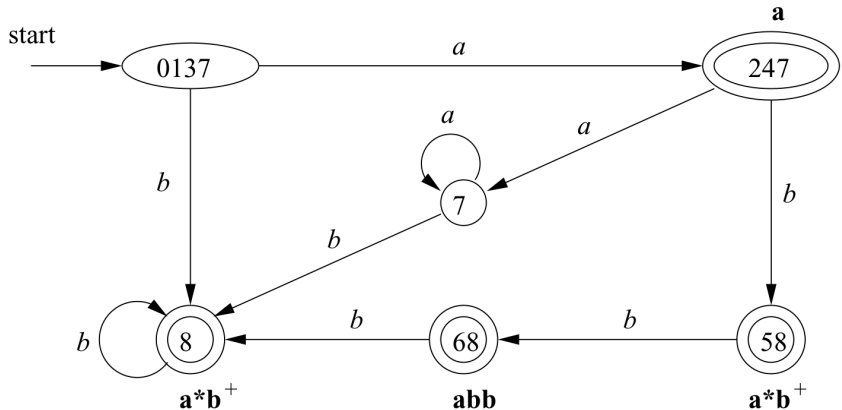
طراحی تولید کننده تحلیل‌گر لغوی

- برای استفاده از ماشین قطعی، از حالت اولیه شروع می‌کنیم و رشته ورودی را می‌خوانیم تا جایی که حالت بعدی وجود نداشته باشد یا به یک حالت بی‌استفاده [1] برسیم. سپس به عقب باز می‌گردیم تا به آخرین حالت پذیرش برسیم و رشته‌ای که در آن حالت پذیرفته شده است را به عنوان یک توکن باز می‌گردانیم.

[1] dead state

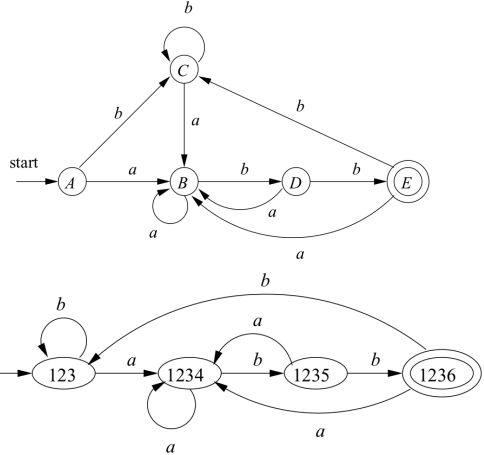
طراحی تولید کننده تحلیل‌گر لغوی

- برای مثال با خواندن رشته abba به حالت 8 می‌رسیم و پس از آن گذاری ممکن نیست. بنابراین به حالت 68 باز می‌گردیم و رشته abb را به عنوان توکن می‌پذیریم.



کاهش تعداد حالات ماشین متناهی قطعی

- برای یک زبان می‌تواند تعداد زیادی ماشین متناهی قطعی وجود داشته باشد که آن را تشخیص می‌دهند.
- برای مثال دو ماشین متناهی قطعی زیر هر دو زبان $L((a|b)^*abb)$ را تشخیص می‌دهند.



کاهش تعداد حالات ماشین متناهی قطعی

- اگر بخواهیم یک تحلیل‌گر لغوی توسط یک ماشین متناهی قطعی بسازیم، ترجیح می‌دهیم ماشین متناهی قطعی کمترین تعداد حالات ممکن را داشته باشد.
- همیشه برای یک عبارت منظم یک ماشین متناهی قطعی با کمترین تعداد حالات وجود دارد.
- الگوریتمی وجود دارد که با دسته‌بندی حالات یک ماشین متناهی قطعی، یک ماشین متناهی قطعی متناظر با آن با کمترین تعداد حالات به دست می‌آورد.

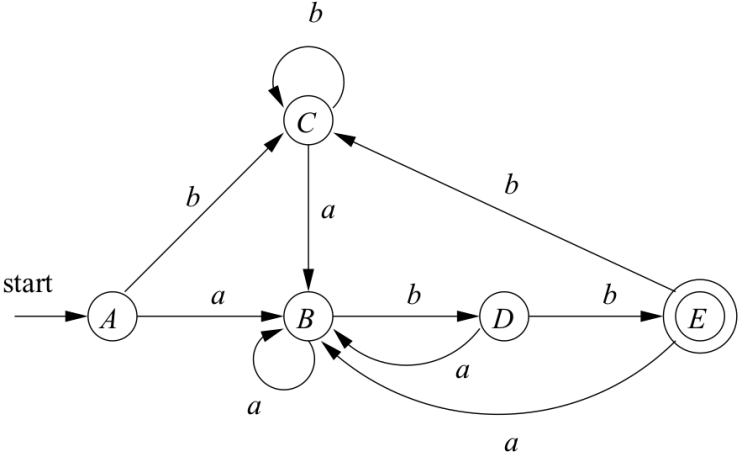
کاهش تعداد حالات ماشین متناهی قطعی

- می‌گوییم رشته x حالت s را از حالت t متمایز [1] می‌کند اگر با شروع از حالت s و t و خواندن رشته ورودی x ، یک بار به یک حالت پایانی و یک بار به یک حالت غیرپایانی برسیم.
- حالت s از حالت t متمایز [2] است اگر حداقل یک رشته وجود داشته باشد که این دو حالت را تمییز دهد.

[1] distinguish
[2] distinguishable

کاهش تعداد حالات ماشین متناهی قطعی

- در ماشین متناهی قطعی زیر، رشته bb حالت A را از حالت B متمایز می‌کند، زیرا رشته bb حالت A را به حالت غیرپایانی C می‌برد، اما حالت B را به حالت پایانی E می‌برد.



کاهش تعداد حالات ماشین متناهی قطعی

- الگوریتم کاهش تعداد حالات ماشین قطعی به این صورت عمل می‌کند که حالت‌های ماشین را به مجموعه‌هایی از حالات تقسیم می‌کند. حالاتی که در یک مجموعه قرار می‌گیرند غیر متمایزاند و هر جفت حالت از دو مجموعه متفاوت متمایزاند. سپس حالاتی که در یک مجموعه قرار می‌گیرند با یکدیگر ادغام می‌شوند و یک حالت واحد را تشکیل می‌دهند.
- در این الگوریتم چند مجموعه از حالات متمایز نگهداری می‌شوند. به محض تشخیص دو حالت متمایز در یک مجموعه، یک مجموعه جدید تشکیل می‌شود. در صورتی که نتوان هیچ یک از مجموعه‌ها را به مجموعه‌های کوچک‌تر تقسیم کرد الگوریتم به پایان می‌رسد.

کاهش تعداد حالات ماشین متناهی قطعی

- در ابتدا دو مجموعهٔ متمایز داریم : مجموعهٔ حالات نهایی و مجموعهٔ حالات غیرنهایی. در هر مرحله از دسته‌بندی حالات هریک از مجموعه‌ها مانند مجموعه $A = \{s_1, s_2, \cdots, s_k\}$ و هریک از نمادها مانند نماد a را در نظر می‌گیریم. دو حالت s_i و s_j متمایزند اگر توسط نماد a ماشین را به دو حالت متمایز ببرند یعنی دو حالتی که در دو مجموعهٔ متفاوت قرار می‌گیرند. در این صورت s_i و s_j باید در دو مجموعهٔ جدا قرار بگیرند. این روند را ادامه می‌دهیم تا جایی که مجموعه‌ها را نتوان به مجموعه‌های کوچکتر تقسیم کرد.

کاهش تعداد حالات ماشین متناهی قطعی

- فرض کنید ماشین متناهی قطعی D با مجموعه حالات S و الفبای ورودی $\sum$ ، حالت اولیه s_0 و مجموعه حالات پذیرش F را داریم. می‌خواهیم ماشین متناهی قطعی D' را به دست آوریم که زبان ماشین D را بپذیرد و تعداد حالات آن کمترین تعداد ممکن باشد.

کاهش تعداد حالات ماشین متناهی قطعی

- الگوریتم کاهش تعداد حالات ماشین متناهی قطعی به صورت زیر عمل می‌کند.

۱. با دسته‌بندی حالات Π آغاز می‌کنیم که دو مجموعه F و $S - F$ از ماشین D را تشکیل می‌دهد.

۲. یک دسته‌بندی جدید به نام $\Pi_{\texttt{new}}$ ایجاد می‌کنیم.
به ازای هر مجموعه G در دسته‌بندی Π اگر دو حالت s و t وجود داشته باشند، به طوری که حالت بعدی آنها به ازای حداقل یکی از حروف الفبا متعلق به دو دسته متمایز در Π باشد، آن دو حالت در دو دسته متفاوت در $\Pi_{\texttt{new}}$ قرار می‌گیرند.

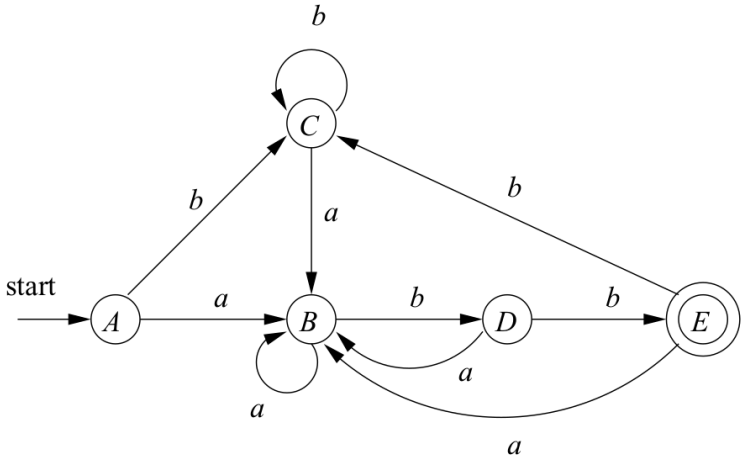
۳. اگر $\Pi_{\texttt{new}}$ برابر با Π باشد، آنگاه قرار می‌دهیم $\Pi_{\texttt{final}} = \Pi$ و با گام چهارم ادامه می‌دهیم. در غیراینصورت گام دوم را با قرار دادن $\Pi = \Pi_{\texttt{new}}$ تکرار می‌کنیم.

کاهش تعداد حالات ماشین متناهی قطعی

۴. هر مجموعه (دسته) در دسته‌بندی ایجاد شده، به عنوان یک حالت برای D' در نظر گرفته می‌شود.
اگر یک مجموعه، حاوی حالت اولیه D باشد، حالت متناظر با آن مجموعه به عنوان حالت اولیه ماشین D' در نظر گرفته می‌شود.
اگر یک مجموعه حاوی حداقل یک حالت نهایی باشد، حالت متناظر با آن مجموعه در ماشین D' یک حالت نهایی خواهد بود.
در پایان باید یال‌ها در ماشین D' را اضافه کنیم. فرض کنید s یکی از حالت‌های مجموعه G و t یکی از حالت‌های مجموعه H در $\Pi_{\texttt{final}}$ باشد و فرض کنید ماشین D در حالت s با خواندن ورودی a به حالت t رود. آنگاه در ماشین D' یالی با برچسب a از حالتی متناظر با مجموعه G به حالت متناظر با مجموعه H رسم می‌کنیم.

کاهش تعداد حالات ماشین متناهی قطعی

- ماشین متناهی قطعی زیر را در نظر بگیرید.



کاهش تعداد حالات ماشین متناهی قطعی

- دسته‌بندی اولیه شامل دو مجموعه {A, B, C, D}{E} است. این دو مجموعه و دو نماد a و b را در نظر می‌گیریم. مجموعه {E} نمی‌تواند تقسیم شود.
- مجموعه {A, B, C, D} و نماد a را در نظر می‌گیریم. با خواندن a هرکدام از حالات این مجموعه به حالت B می‌روند که در درون مجموعه قرار دارد. پس نماد a هیچ‌کدام از حالات این مجموعه را متمایز نمی‌کند. با ورودی b حالت D به E می‌رود که در یک مجموعه دیگر قرار دارد پس یک مجموعه دیگر تشکیل می‌دهیم و خواهیم داشت {A, B, C}{D}{E}.
- در مرحله بعد {A, B, C} را به دو مجموعه {A, C}{B} تقسیم می‌کنیم زیرا B با خواندن b به D می‌رود، در حالی که A و C هرکدام با خواندن b به یکی از حالات مجموعه {A, B, C} می‌روند.
- در پایان مجموعه‌ها به صورت {A, C}{B}{D}{E} افراز می‌شوند.

کاهش تعداد حالات ماشین متناهی قطعی

- اگر این تقسیم‌بندی را ادامه دهیم، مجموعه‌ها به مجموعه‌های کوچکتر تقسیم نمی‌شوند.
- حال برای هریک از حالات یک نماینده در نظر می‌گیریم و چهار حالت A ، B ، D و E را در ماشین D' تشکیل می‌دهیم.
- حالت اولیه در این ماشین A است و تنها حالت نهایی E است.
- در نهایت جدول گزار ماشین D' را به صورت زیر رسم می‌کنیم.

STATE	a	b
A	B	A
B	B	D
D	B	E
E	B	A