نظریه‌ی زبان‌ها و ماشین‌ها

درس‌نامه‌ی ۶

کاظم فولادی
http://kazim.fouladi.ir

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

فهرست مطالب

۶ ساده‌سازی گرامرهای مستقل از متن و فرم‌های نرمال

SIMPLIFICATION OF CONTEXT-FREE GRAMMARS AND NORMAL FORMS

ساده‌سازی گرامر می‌تواند تعبیرهای مختلفی داشته باشد (مثلاً کاهش تعداد قواعد، کاهش تعداد ناپایانه‌ها، کاهش مجموع طول سمت راست قواعد و . . .) اما منظور ما از ساده‌سازی، حذف برخی از قواعد نامطلوب گرامر است و بنابراین همیشه به کاهش تعداد قواعد گرامر منجر نمی‌شود.

۶-۱ روش‌های تبدیل گرامرها

منظور از تبدیل گرامر، عملی است که یک گرامر را به یک گرامر معادل تبدیل می‌کند.

◄ **تذکر** **در عمل تفاوت عمده‌ای بین زبان‌های مستقل از متنی که λ را می‌پذیرند و آنهایی که λ را نمی‌پذیرند، وجود ندارد.** اگر L یک زبان مستقل از متن و $G = (V, T, S, P)$ یک گرامر مستقل از متن برای $L - \{\lambda\}$ باشد، آنگاه گرامری که با افزودن S_0 به عنوان نماد شروع و قاعده‌ی $S_0 \to S|\lambda$ به دست می‌آید، زبان L را تولید می‌کند. همچنین با داشتن گرامر مستقل از متن G می‌توان $\hat{G}$ را به گونه‌ای پیدا کرد که $L(\hat{G}) = L(G) - \{\lambda\}$ باشد.

۶-۱-۱ قاعده‌ی جایگزینی

قضیه

فرض می‌کنیم گرامر مستقل از متن $G = (V, T, S, P)$ حاوی قواعدی به شکل $A \to x_1 B x_2$ با $A, B \in V$ و $A \neq B$ و

$$B \to y_1 \mid y_2 \mid \ldots \mid y_n$$

کلیه‌ی قواعدی باشد که سمت چپ آنها B است. اگر $\hat{G} = (V, T, S, \hat{P})$ گرامری باشد که در آن $\hat{P}$ با حذف قاعده‌ی

$$A \to x_1 B x_2$$

و افزودن قاعده‌های

$$A \to x_1 y_1 x_2 \mid x_1 y_2 x_2 \mid \ldots \mid x_1 y_n x_2$$

به دست آمده باشد، آنگاه $L(\hat{G}) = L(G)$.

◄ **اثبات.**

فرض می‌کنیم $w \in L(G)$ به گونه‌ای باشد که $S \Rightarrow_G^* w$.
اگر این اشتقاق از قاعده‌ی $A \to x_1 B x_2$ استفاده نکند، آنگاه $S \Rightarrow_{\hat{G}}^* w$ و اگر استفاده کند، متغیر B در زیر نهایتاً با یکی از قواعد آن جایگزین می‌شود:

$$\exists j \; S \Rightarrow_G^* u_1 A u_2 \Rightarrow_G u_1 x_1 B x_2 u_2 \Rightarrow_G u_1 x_1 y_j x_2 u_2$$

اما با گرامر $\hat{G}$ داریم:

$$\exists j \; S \Rightarrow_{\hat{G}}^* u_1 A u_2 \Rightarrow_{\hat{G}} u_1 x_1 y_j x_2 u_2$$

بنابراین با G و $\hat{G}$ به یک فرم جمله‌ای می‌رسیم.
اگر در مراحل بعد مجدداً از قاعده‌ی $A \to x_1 B x_2$ استفاده شود، همین بحث را تکرار می‌کنیم.
با استفاده از استقرا بر روی تعداد دفعات اعمال قاعده می‌توانیم نتیجه بگیریم که:

$$S \Rightarrow_{\hat{G}}^* w$$

بنابراین اگر $w \in L(G)$ آنگاه $w \in L(\hat{G})$ است. به همین ترتیب می‌توان نشان داد که اگر $w \in L(\hat{G})$، آنگاه $w \in L(G)$ خواهد بود و در نتیجه $L(G) = L(\hat{G})$.

■

همواره می‌توان قواعد یک متغیر را در سمت راست قواعدی که این متغیر ظاهر شده است جایگزین کرد و گرامری معادل با گرامر اصلی پیدا کرد (به شرطی که قواعد بازگشتی به شکل $A \to xAy$ نداشته باشیم).

مثال

▼

گرامر $G = (\{A, B\}, \{a, b, c\}, A, P)$ با قواعد

$$A \to a \mid aaA \mid abBc \quad B \to abbA \mid b$$

را در نظر می‌گیریم.
می‌توان قواعد B را جایگزین کرد و یک گرامر معادل با گرامر فوق به دست آورد:

$$A \to a \mid aaA \mid ababbAc \mid abbc$$

▲

۶-۱-۲ حذف قواعد بی‌فایده

یک متغیر گرامر در صورتی بی‌فایده است که:

- یا به دلیلی از نماد شروع قابل دسترس نباشد،
- یا اینکه نتواند یک رشته‌ی پایانی تولید کند.

تعریف اگر $G = (V, T, S, P)$ یک گرامر مستقل از متن باشد، متغیر $A \in V$ را **مفید** *(useful)* گوییم اگر و فقط اگر

$$\exists w \in L(G)\ \ S \Rightarrow^* xAy \Rightarrow^* w$$

به عبارت دیگر، متغیری مفید است که حداقل در یک فرم جمله‌ای منتهی به یک رشته ظاهر شود.
متغیری که مفید نباشد، **بی‌فایده** *(useless)* نام دارد.
قاعده‌ای که متغیر بی‌فایده داشته باشد، **قاعده‌ی بی‌فایده** نامیده می‌شود.

تشخیص متغیرهای بی‌فایده یک متغیر مفید باید دسترس‌پذیر (reachable) و مولد (generative) باشد.
برای دسترس‌پذیری یک متغیر باید بتوانیم با آغاز از نماد شروع طی صفر یا چند مرحله به یک شکل جمله‌ای شامل آن متغیر برسیم ($S \Rightarrow^* \cdots A \cdots$).
برای مولد بودن یک متغیر باید بتوان با شروع از آن متغیر طی صفر یا چند مرحله به یک رشته‌ی پایانی رسید ($A \Rightarrow^* x,\ \ x \in T^*$).
می‌توان از الگوریتم زیر برای تشخیص متغیرهای مولد استفاده کرد:

GENERATIVE

```
1  Gen ← ∅
2  Gen′ ← {A : A → a ∈ P,  a ∈ T}
3  while Gen ≠ Gen′
4        do Gen ← Gen′
5           Gen′ ← Gen ∪ {A : A → X1X2 . . . Xk, ∀i(1 ≤ i ≤ k) → Xi ∈ Gen ∪ T}
6  return Gen′
```

هر قاعده‌ای که حاوی حداقل یک متغیر بی‌فایده باشد را می‌توان از گرامر حذف نمود.

تعریف **گراف وابستگی گرامر (grammar dependency graph).** گراف وابستگی یک گرامر مستقل از متن یک گراف جهت دار است که به صورت زیر تعریف می‌شود:
رأس‌ها: متغیرهای گرامر

یال‌ها: بین دو راس A و B یک یال وجود دارد، اگر قاعده‌ای به شکل $A \to xBy$ وجود داشته باشد. راس C یک راس پایانی است اگر قاعده‌ای به شکل $C \to x$ وجود داشته باشد که در آن x یک رشته‌ی پایانی است.

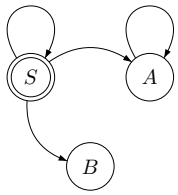
قضیه اگر $G = (V, T, S, P)$ یک گرامر مستقل از متن باشد، آن‌گاه گرامر مستقل از متن $\hat{G} = (\hat{V}, \hat{T}, \hat{S}, \hat{P})$ معادل با آن وجود دارد که هیچ متغیر یا قاعده‌ی بی‌فایده ندارد.

مثال

گرامر با قواعد

$$\begin{aligned} S &\to aSB \mid \lambda \mid A \\ A &\to aA \end{aligned}$$

را در نظر می‌گیریم. گراف وابستگی این گرامر به صورت زیر است:



مشاهده می‌شود که A و B مولد نیستند، پس بی‌فایده‌اند و می‌توان قواعد حاوی آنها را از گرامر کنار گذاشت:

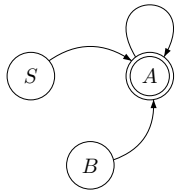
$$S \to \lambda$$

مثال

گرامر با قواعد

$$\begin{aligned} S &\to A \\ A &\to aA \mid \lambda \\ B &\to bA \end{aligned}$$

را در نظر می‌گیریم. گراف وابستگی این گرامر به صورت زیر است:



مشاهده می‌شود که B دسترس‌پذیر نیست، پس بی‌فایده است و می‌توان قواعد حاوی آن را از گرامر کنار گذاشت:

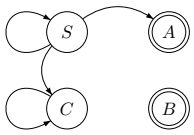
$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow aA \mid \lambda \end{aligned}$$

مثال

گرامر با قواعد

$$\begin{aligned} S &\rightarrow aS \mid A \mid C \\ A &\rightarrow a \\ B &\rightarrow aa \\ C &\rightarrow aCb \end{aligned}$$

را در نظر می‌گیریم. گراف وابستگی این گرامر به صورت زیر است:



مشاهده می‌شود که B و C متغیرهای بی‌فایده هستند و می‌توان قواعد حاوی آنها را از گرامر کنار گذاشت:

$$\begin{aligned} S &\rightarrow aS \mid A \\ A &\rightarrow a \end{aligned}$$

۶-۱-۳ حذف قواعد تهی (قواعد λ)

تعریف

در یک گرامر مستقل از متن، هر قاعده به شکل $A \rightarrow \lambda$ یک قاعده‌ی تهی نام دارد.
هر متغیر مانند A که برای آن اشتقاق $A \Rightarrow^* \lambda$ ممکن باشد، میرا *(nullable)* نامیده می‌شود.

$$Nullable(A) \Leftrightarrow$$
$$(A \rightarrow \lambda \in P) \vee (A \rightarrow \alpha_1\alpha_2 \ldots \alpha_n \in P \wedge \forall i \ 1 \leq i \leq n \Rightarrow Nullable(\alpha_i))$$

تشخیص متغیرهای میرا از الگوریتم زیر استفاده می‌کنیم:

NULLABLE

1 $Null \leftarrow \varnothing$
2 $Null' \leftarrow \{A : A \rightarrow \lambda \in P\}$
3 **while** $Null \neq Null'$
4 **do** $Null \leftarrow Null'$
5 $Null' \leftarrow Null \cup \{A : A \rightarrow X_1X_2 \dots X_k, \forall i(1 \leq i \leq k) \rightarrow X_i \in Null\}$
6 **return** $Null'$

مراحل حذف قواعد λ برای حذف قواعد λ از P و به دست آوردن قواعد P' که فاقد چنین قواعدی باشد، به صورت زیر عمل می‌کنیم:
به ازای هر قاعده به شکل $A \rightarrow X_1X_2 \dots X_k$ در P،
قواعد $A \rightarrow \alpha_1\alpha_2 \dots \alpha_k$ را به P' اضافه می‌کنیم به گونه‌ای که:

- اگر X_i میرا نباشد، آنگاه $\alpha_i = X_i$
- اگر X_i میرا باشد، آنگاه $\alpha_i = X_i|\lambda$ (به شرطی که همه‌ی α_iها λ نشوند).

◄ اگر زبان شامل λ نباشد، می‌توان تمام قواعد λ را حذف کرد، وگرنه حداقل به $S \rightarrow \lambda$ نیاز داریم.

◄ قواعد تهی در برخی تجزیه‌گرها مشکل‌آفرین هستند.

قضیه اگر G یک گرامر مستقل از متن باشد که $L(G)$ فاقد λ باشد، آنگاه گرامر $\hat{G}$ هم‌ارز با G وجود دارد که قواعد λ ندارد.

مثال

گرامر زیر را در نظر بگیرید:

$$\begin{aligned} S &\rightarrow aS_1b \\ S_1 &\rightarrow aS_1b \mid \lambda \end{aligned}$$

داریم $Nullable = \{S_1\}$. گرامر حاصل پس از حذف قواعد λ به صورت زیر است:

$$\begin{aligned} S &\rightarrow aS_1b \mid ab \\ S_1 &\rightarrow aS_1b \mid ab \end{aligned}$$

مثال

گرامر زیر را در نظر بگیرید:

$$
\begin{aligned}
S &\rightarrow ABaC \\
A &\rightarrow BC \\
B &\rightarrow b \mid \lambda \\
C &\rightarrow D \mid \lambda \\
D &\rightarrow d
\end{aligned}
$$

برای یافتن متغیرهای میرا، الگوریتم NULLABLE را اجرا می‌کنیم:

$Null$	$Null'$
$\varnothing$	$\{B, C\}$
$\{B, C\}$	$\{B, C, A\}$
$\{A, B, C\}$	$= \{A, B, C\}$

داریم $Nullable = \{A, B, C\}$. گرامر حاصل پس از حذف قواعد λ به صورت زیر است:

$$
\begin{aligned}
S &\rightarrow ABaC \mid AaC \mid ABa \mid a \mid BaC \mid Aa \mid Ba \mid aC \\
A &\rightarrow BC \mid B \mid C \\
B &\rightarrow b \\
C &\rightarrow D \\
D &\rightarrow d
\end{aligned}
$$

▲

۶-۱-۴ حذف قواعد یکه

تعریف در یک گرامر مستقل از متن، هر قاعده به شکل $A \rightarrow B$ که در آن $A, B \in V$ باشد، قاعده‌ی یکه *(unit production)* یا زنجیر *(chain)* نام دارد.

قضیه فرض کنید $G = (V, T, S, P)$ یک گرامرمستقل از متن و بدون قواعد λ باشد، در این صورت گرامر مستقل از متن $\hat{G}$ هم‌ارز با G وجود دارد که هیچ قاعده‌ی یکه‌ای ندارد.

مراحل حذف قواعد یکه

- حذف قواعد به شکل $A \rightarrow A$
- حذف قواعد به شکل $A \rightarrow B$ با جایگزینی قواعد B
- حذف قواعد در حالت وجود قواعد $A \rightarrow B$ و $B \rightarrow A$:

- برای هر $A \in V$ همه‌ی متغیرهای B را که $A \Rightarrow^* B$ می‌یابیم. با استفاده از گراف وابستگی گرامر برای قواعد یکه[۱] $A \Rightarrow^* B$ اگر و فقط اگر یک گشت بین A و B وجود داشته باشد.
- به گرامر جدید، همه‌ی قواعد **غیر یکه‌ی** گرامر اصلی را اضافه می‌کنیم.
- به ازای هر A و B که $A \Rightarrow^* B$، قواعد B را با استفاده از قواعد گرامر جدید در $A \rightarrow B$ جایگزین می‌کنیم.

◄ **نکته** حذف زنجیر موجب کاهش طول اشتقاق و افزایش سرعت تجزیه می‌شود.

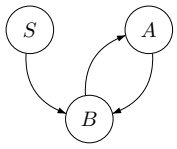
مثال

گرامر زیر را در نظر بگیرید:

$$\begin{aligned} S &\rightarrow Aa \mid B \\ B &\rightarrow A \mid bb \\ A &\rightarrow a \mid bc \mid B \end{aligned}$$

می‌خواهیم قواعد یکه را از آن حذف کنیم. در اینجا هم قاعده‌ی $A \rightarrow B$ و هم قاعده‌ی $B \rightarrow A$ وجود دارد.

گراف وابستگی این گرامر برای قواعد یکه به صورت زیر است:



با حذف قواعد یکه مطابق روال توضیح داده شده، خواهیم داشت:

$$\begin{aligned} S &\rightarrow Aa \mid bb \\ B &\rightarrow bb \\ A &\rightarrow a \mid bc \mid bb \end{aligned}$$

▲

قضیه فرض کنید که L یک زبان مستقل از متن بدون λ باشد، در این صورت گرامر مستقل از متنی وجود دارد که L را تولید می‌کند و هیچ قاعده‌ی بی‌فایده، یکه یا تهی ندارد.

[۱] در گراف وابستگی گرامر برای قواعد یکه، بین راس‌های متغیرهای X و Y یک یال وجود دارد اگر و فقط اگر قاعده‌ی یکه‌ی $X \rightarrow Y$ در آن گرامر وجود داشته باشد.

ترتیب اعمال قواعد

۱) حذف قواعد λ

۲) حذف قواعد یکه

۳) حذف قواعد بی‌فایده

زیرا:
حذف قواعد یکه، قواعد تهی λ تولید نمی‌کند.
حذف قواعد بی‌فایده،قواعد تهی یا یکه تولید نمی‌کند.

۶-۱-۵ حذف نماد شروع بازگشتی

چنانچه نماد شروع گرامر S در سمت راست برخی از قواعد ظاهر شده باشد، می‌توان با معرفی یک نماد شروع جدید S' و افزودن قاعده‌ی جدید $S' \rightarrow S$، نماد شروع بازگشتی را حذف کرد.

۶-۱-۶ حذف بازگشتی از چپ

تعریف

قاعده‌ی $A \rightarrow xAy$ که در آن متغیر سمت چپ در سمت راست قاعده نیز ظاهر شده است، یک **قاعده‌ی بازگشتی** *(recursive)* نام دارد $(A \in V,\ x, y \in (V \cup T)^*)$.
قاعده‌ی $A \rightarrow Ax$ که در آن متغیر سمت چپ در منتهی الیه چپ سمت راست قاعده نیز ظاهر شده است، یک **قاعده‌ی بازگشتی از چپ** *(left recursive)* نام دارد.
قاعده‌ی $A \rightarrow xA$ که در آن متغیر سمت چپ در منتهی الیه راست سمت راست قاعده نیز ظاهر شده است، یک **قاعده‌ی بازگشتی از راست** *(right recursive)* نام دارد.
متغیری که برای آن حداقل یک قاعده‌ی بازگشتی وجود داشته باشد، متغیر بازگشتی نام دارد.

حذف قواعد بازگشتی از چپ مستقیم برای حذف قواعد بازگشتی از چپ متغیر بازگشتی A، قواعد A را به دو بخش بازگشتی از چپ و غیر بازگشتی از چپ تقسیم می‌کنیم:

$$\begin{aligned} A &\rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \\ A &\rightarrow \beta_1 \mid \beta_2 \mid \ldots \beta_m \end{aligned}$$

حال این مجموعه قواعد را با مجموعه قواعد زیر جایگزین می‌کنیم:

$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \beta_m A' \\ A' &\rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \lambda \end{aligned}$$

◄ **نکته** در حالت ساده، قواعد $A \rightarrow A\alpha \mid \beta$ با قواعد $A \rightarrow \beta A'$ و $A' \rightarrow \alpha A' \mid \lambda$ جایگزین می‌شود.

◄ **نکته** بازگشتی از چپ، تجزیه‌گرهای بالابه پایین را دچار مشکل می‌کند.

مثال

بازگشتی از چپ را از گرامر با قواعد زیر رفع کنید:

$$\begin{aligned} A &\rightarrow Aa \mid aBc \mid \lambda \\ B &\rightarrow Bb \mid bc \end{aligned}$$

$$\begin{aligned} A &\rightarrow A\overbrace{a}^{\alpha} \mid \overbrace{aBc}^{\beta_1} \mid \overbrace{\lambda}^{\beta_2} \\ B &\rightarrow B\underbrace{b}_{\alpha} \mid \underbrace{bc}_{\beta} \end{aligned}$$

مطابق قاعده‌ی ارایه شده داریم:

$$\begin{aligned} A &\rightarrow aBcA' \mid A' \\ A' &\rightarrow aA' \mid \lambda \\ B &\rightarrow bcB' \\ B' &\rightarrow bB' \mid \lambda \end{aligned}$$

تعریف

متغیر A یک متغیر بازگشتی غیر مستقیم است اگر وفقط اگر اشتقاق $A \Rightarrow^n xAy$ در آن گرامر ممکن باشد $(n \geq ۲)$.

متغیر A یک متغیر بازگشتی از چپ غیر مستقیم است اگر وفقط اگر اشتقاق $A \Rightarrow^n Ax$ در آن گرامر ممکن باشد $(n \geq ۲)$.

متغیر A یک متغیر بازگشتی از راست غیر مستقیم است اگر وفقط اگر اشتقاق $A \Rightarrow^n xA$ در آن گرامر ممکن باشد $(n \geq ۲)$.

حذف بازگشتی از چپ غیر مستقیم برای حذف بازگشتی از چپ غیر مستقیم، به صورت زیر عمل می‌کنیم (برای گرامر مستقل از متن بدون دور و بدون قواعد یکه):

- یک ترتیب بر روی متغیرهای گرامر در نظر می‌گیریم.
- بازگشتی از چپ مستقیم را برای متغیر اول حذف می‌کنیم.
- برای متغیرهای بعدی: قواعد متغیر قبلی را جایگزین می‌کنیم و بازگشتی از چپ مستقیم را برای قواعد متغیر فعلی حذف می‌کنیم.

۶-۲ فرم‌های نرمال

۶-۲-۱ فرم نرمال چامسکی

تعریف **فرم نرمال چامسکی (CNF).** گرامری در فرم نرمال چامسکی است که تمام قواعد آن به صورت

$$A \to BC \qquad \text{یا} \qquad A \to a$$

باشد که در آن $A, B, C \in V$ و $a \in T$ است.
به عبارت دیگر سمت راست قواعد دو ناپایانه یا یک پایانه است.

قضیه هر گرامر مستقل از متن $G = (V, T, S, P)$ که در آن $\lambda \notin L(G)$، دارای گرامر معادل $\hat{G}$ در فرم نرمال چامسکی است.

- تولید رشته‌ای به طول n با یک گرامر در فرم نرمال چامسکی در $(۲n - ۱)$ مرحله اشتقاق انجام می‌شود که در آن $(n - ۱)$ مرحله اشتقاق با قواعد به شکل $A \to BC$ و n مرحله اشتقاق با قواعد به شکل $A \to a$ انجام می‌شود.
- درخت اشتقاق برای یک گرامر در شکل نرمال چامسکی، یک درخت دودویی است.

مراحل تبدیل به فرم نرمال چامسکی به صورت زیر عمل می‌کنیم:

- ابتدا گرامر را ساده می‌کنیم (حذف قواعد تهی، حذف قواعد یکه، حذف قواعد بی‌فایده)
- تبدیل نهایی به فرم نرمال چامسکی را با معرفی متغیرها و قواعد جدید انجام می‌دهیم.

مثال

گرامر زیر را به فرم نرمال چامسکی تبدیل کنید:

$$\begin{aligned} S &\to ABa \\ A &\to aab \\ B &\to Ac \end{aligned}$$

با استفاده از روال فوق داریم:

$$
\begin{aligned}
S &\rightarrow AT_1 \\
T_1 &\rightarrow BT_a \\
A &\rightarrow T_aT_2 \\
T_2 &\rightarrow T_aT_b \\
B &\rightarrow AT_c \\
T_a &\rightarrow a \\
T_b &\rightarrow b \\
T_c &\rightarrow c
\end{aligned}
$$

دقت کنید که برای هر پایانه یک ناپایانه‌ی جدید تعریف کرده‌ایم و هر قاعده‌ای که سمت راست آن بیشتر از دو نماد داشته است، به دو یا چند قاعده شکسته شده است.

▲

۶-۲-۲ فرم نرمال گریباخ

تعریف

فرم نرمال گریباخ (GNF). گرامری در فرم نرمال گریباخ است که تمام قواعد آن به صورت

$$A \rightarrow ax$$

باشد که در آن $x \in V^*$ و $a \in T$ است.
به عبارت دیگر سمت راست قواعد یک پایانه و به دنبال آن صفر یا چند ناپایانه است.

قضیه

هر گرامر مستقل از متن $G = (V, T, S, P)$ که در آن $\lambda \notin L(G)$، دارای گرامر معادل $\hat{G}$ در فرم نرمال گریباخ است.

◄ تولید رشته‌ای به طول n با یک گرامر در فرم نرمال گریباخ در n مرحله اشتقاق انجام می‌شود.

مراحل تبدیل به فرم نرمال گریباخ به صورت زیر عمل می‌کنیم:

- ابتدا گرامر را ساده می‌کنیم (حذف قواعد تهی، حذف قواعد یکه، حذف قواعد بی‌فایده)
- بازگشتی چپ (مستقیم و غیر مستقیم) را از گرامر حذف می‌کنیم.
- تبدیل نهایی به فرم نرمال گریباخ را انجام می‌دهیم.

مثال ▼

گرامر زیر را به فرم نرمال گریباخ تبدیل کنید:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aA \mid bB \mid b \\ B &\rightarrow b \end{aligned}$$

با استفاده از روال ارایه شده داریم:

$$\begin{aligned} S &\rightarrow aAB \mid bBB \mid bB \\ A &\rightarrow aA \mid bB \mid b \\ B &\rightarrow b \end{aligned}$$

مثال

گرامر زیر را به فرم نرمال گریباخ تبدیل کنید:

$$S \rightarrow abSd \mid aa$$

با استفاده از روال ارایه شده داریم:

$$\begin{aligned} S &\rightarrow aT_bST_d \\ S &\rightarrow aT_a \\ T_a &\rightarrow a \\ T_b &\rightarrow b \\ T_c &\rightarrow c \end{aligned}$$

۳-۶ الگوریتم عضویت CYK برای گرامرهای مستقل از متن

CYK یک الگوریتم عضویت برای گرامرهای مستقل از متن در فرم نرمال چامسکی است که توسط Cocke، Younger و Kasami ابداع شد.
این الگوریتم بر پایه‌ی روش برنامه‌ریزی پویا عمل می‌کند.
فرض کنید گرامر $G = (V, T, S, P)$ را در فرم نرمال چامسکی داریم و می‌خواهیم عضویت رشته‌ی w را در $L(G)$ بررسی کنیم. اگر

$$w = a_1 a_2 \dots a_n$$

باشد، زیررشته‌ی

$$w[i, j] = a_i a_{i+1} \dots a_j$$

و زیرمجموعه‌ی زیر از V را به صورت

$$V[i, j] = \{A \in V : A \Rightarrow_G^* w[i, j]\}$$

تعریف می‌کنیم. مشخص است که $w \in L(G)$ است اگر و فقط اگر $S \in V[1, n]$ باشد.

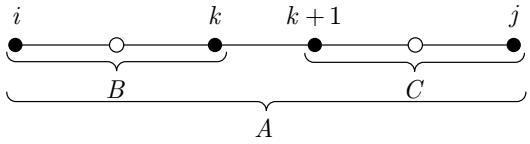
برای محاسبه‌ی $V[i,j]$:

$A \in V[i,i]$ خواهد بود، اگر و فقط اگر G دارای قاعده‌ای به شکل $A \to a_i$ باشد.

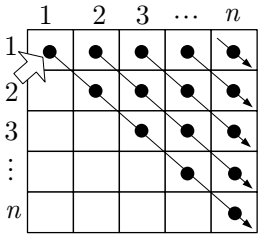
بنابراین می‌توان برای هر $1 \le i \le n$، $V[i,i]$ را محاسبه نمود.

برای حالت $i < j$، متغیر A زیررشته‌ی $w[i,j]$ را تولید می‌کند، اگر و فقط اگر قاعده‌ای مانند $A \to BC$ موجود باشد که برای یک k، $i \le k < j$، داشته باشیم ؛ یعنی $B \Rightarrow^* w[i,k]$ و $C \Rightarrow^* w[k+1,j]$

$$V[i,j] = \bigcup_{i \le k < j} \{A : A \to BC, B \in V[i,k] \wedge C \in V[k+1,j]\}$$



اگر $V[i,j]$ را به صورت یک ماتریس نشان دهیم، ابتدا مقادیر قطر اصلی آن را محاسبه می‌کنیم و به ترتیب به قطرهای بالای قطر اصلی می‌رویم و مقادیر آنها را محاسبه می‌نماییم.



◄ **نکته** به سادگی می‌توان نشان داد که تعداد قدم‌های این الگوریتم تقریباً $n^{\frac{n(n+1)}{2}}$ است که در آن $n = |w|$.

مثال

آیا رشته‌ی $w = aabbb$ در زبان تولید شده توسط گرامر زیر وجود دارد یا خیر؟

$$\begin{aligned} S &\to AB \\ A &\to BB \mid a \\ B &\to AB \mid b \end{aligned}$$

جدول CYK را تشکیل می‌دهیم و مطابق رابطه‌ی بازگشتی ارایه شده، آن را پر می‌کنیم:

	۱	۲	۳	۴	۵
۱	$\{A\}$	$\{\}$	$\{S,B\}$	$\{A\}$	$\{S,B\}$
۲		$\{A\}$	$\{S,B\}$	$\{A\}$	$\{S,B\}$
۳			$\{B\}$	$\{A\}$	$\{S,B\}$
۴				$\{B\}$	$\{A\}$
۵					$\{B\}$

چون $S \in V[1,5]$، پس w در زبان تولید شده با این گرامر وجود دارد.

مراجع

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

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