# ۵ زبان‌های مستقل از متن

CONTEXT-FREE LANGUAGES

## ۵-۱ گرامرها و زبان‌های مستقل از متن

**تعریف** **گرامر مستقل از متن** گرامر $G = (V, T, S, P)$ را مستقل از متن می‌گوییم، اگر و فقط اگر همه‌ی قواعد $P$ به صورت

$$A \to x \qquad , x \in (V \cup T)^*, \;\; A \in V$$

باشد.

---

در گرامر مستقل از متن، سمت چپ هر قاعده فقط یک ناپایانه وجود دارد.

**تعریف** **زبان مستقل از متن** زبان $L$ مستقل از متن است، اگر و فقط اگر یک گرامر مستقل از متن $G$ وجود داشته باشد که $L = L(G)$ باشد.

---

- هر گرامر منظم، یک گرامر مستقل از متن است $\Leftarrow$ هر زبان منظم، یک زبان مستقل از متن است.
- خانواده‌ی زبان‌های منظم، زیرمجموعه‌ی خانواده‌ی زبان‌های مستقل از متن است.
- مستقل از متن یعنی این که جایگزینی متغیرهای سمت چپ قواعد را می‌توان در هر زمانی که آن متغیر در یک شکل جمله‌ای ظاهر می‌شود انجام داد و این به بقیه‌ی فرم جمله‌ای وابستگی ندارد (نتیجه‌ی وجود تنها یک متغیر در سمت چپ هر قاعده).

**تعریف** **گرامر خطی** گرامر $G = (V, T, S, P)$ خطی است که در سمت راست همه‌ی قواعد تولید آن حداکثر یک ناپایانه وجود داشته باشد. یعنی همه‌ی قواعد $P$ به صورت

$$A \to xBy \;\; \text{یا} \;\; A \to x \qquad , x, y \in T^*, \;\; A, B \in V$$

باشد.

---

**مثال**

گرامر $G = (S, a, b, S, P)$ با قواعد $S \to aSa,\ \ S \to bSb,\ \ S \to \lambda$ مستقل از متن است.

یک نمونه اشتقاق از این گرامر به صورت زیر است.

$$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabbaa$$

می‌توان با استقرا ثابت کرد که گرامر فوق زبان زیر را تولید می‌کند:

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

مشاهده می‌شود که مطابق تعریف، گرامر فوق یک گرامر خطی است.

**مثال**

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

ملاحظه می‌کنیم که زبان فوق می‌تواند به صورت اجتماع دو زبان نوشته شود:

$$L = \{a^n b^m : n > m\} \cup \{a^n b^m : n < m\}$$

و به این ترتیب گرامر مستقل از متن تولید کننده‌ی آن می‌شود:

$$S \to S_1 \mid S_2$$

$$\begin{array}{lll@{\qquad}lll} S_1 & \to & AS' & S_2 & \to & S''b \\ S' & \to & aS'b \mid \lambda & S'' & \to & aS''b \mid \lambda \\ A & \to & aA \mid a & B & \to & bB \mid b \end{array}$$

مشاهده می‌شود که مطابق تعریف، گرامر فوق یک گرامر خطی نیست.

**مثال**

گرامر با قواعد $S \to aSb|SS|\lambda$ مستقل از متن است، ولی خطی نیست.

زبان این گرامر عبارت است از:

$$L = \{w \in \{a,b\}^* : n_a(w) = n_b(w) \wedge n_a(v) \geq n_b(v), \text{هر زیررشته‌ای از } w \ \ v\}$$

که با در نظر گرفتن $a$ به عنوان پرانتز باز و $b$ به عنوان پرانتز بسته ساختار پرانتزهای تودرتوی صحیح را مشخص می‌کند.

## ۵-۱-۱ اشتقاق در گرامرهای مستقل از متن

**اشتقاق چپ‌ترین** در هر قدم از اشتقاق، سمت چپ‌ترین متغیر در فرم جمله‌ای جایگزین می‌شود $(\Rightarrow_{LM})$.

**اشتقاق راست‌ترین** در هر قدم از اشتقاق، سمت راست‌ترین متغیر در فرم جمله‌ای جایگزین می‌شود $(\Rightarrow_{RM})$.

◄ **نکته** در صورت وجود اشتقاق، هم اشتقاق راست و هم اشتقاق چپ وجود خواهد داشت.

**مثال**

گرامر با قواعد $S \to aAB,\ A \to bBb,\ B \to A|\lambda$ را در نظر بگیرید. برای رشته‌ی $abbbb$:

یک اشتقاق چپ‌ترین:

$$S \Rightarrow_{LM} a\underline{A}B \Rightarrow ab\underline{B}bB \Rightarrow ab\underline{A}bB \Rightarrow abb\underline{B}bbB \Rightarrow abbbb\underline{B} \Rightarrow abbbb$$

یک اشتقاق راست‌ترین:

$$S \Rightarrow_{RM} aA\underline{B} \Rightarrow a\underline{A} \Rightarrow ab\underline{B}b \Rightarrow ab\underline{A}b \Rightarrow abb\underline{B}bb \Rightarrow abbbb$$

**تعریف**

**درخت اشتقاق** اگر $G = (V, T, S, P)$ یک گرامر مستقل از متن باشد، درخت ریشه‌دار مرتب $t_G$ یک درخت اشتقاق *(derivation tree)* برای $G$ است، اگر و فقط اگر

۱) ریشه دارای برچسب $S$ باشد.

۲) هر یک از برگ‌ها دارای برچسبی از $T \cup \{\lambda\}$ باشد.

۳) هر یک از گره‌های داخلی دارای برچسبی از $V$ باشد.

۴) اگر گره‌ای دارای برچسب $A \in V$ باشد و فرزندان آن از چپ به راست به صورت

$$a_۱, a_۲, \ldots, a_n$$

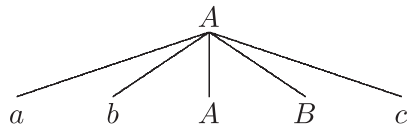
برچسب‌گذاری شده باشد، آن‌گاه $P$ حاوی قاعده‌ای به شکل $A \to a_۱a_۲\ldots a_n$ باشد.

۵) گره‌ای که دارای فرزندی با برچسب $\lambda$ باشد، هیچ فرزند دیگری نداشته باشد.

حاصل درخت اشتقاق از پیمایش عمق ـ اول برگ‌ها به دست می‌آید و یک رشته در زبان گرامر است.

**مثال**

درخت اشتقاق برای قاعده‌ی $A \to abABc$:




▲

**تعریف**

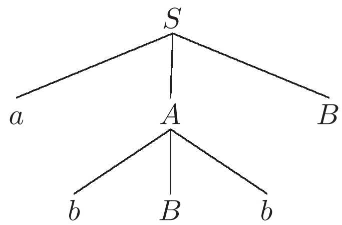
**درخت اشتقاق جزیی** درخت اشتقاق جزیی همانند درخت اشتقاق است با این تفاوت که به جای شرط (۲)، شرط «هر برگ دارای برچسبی از $V \cup T \cup \{\lambda\}$ است» را دارد.

حاصل درخت اشتقاق جزیی از پیمایش عمق ـ اول برگ‌ها به دست می‌آید و یک فرم جمله‌ای از گرامر است.

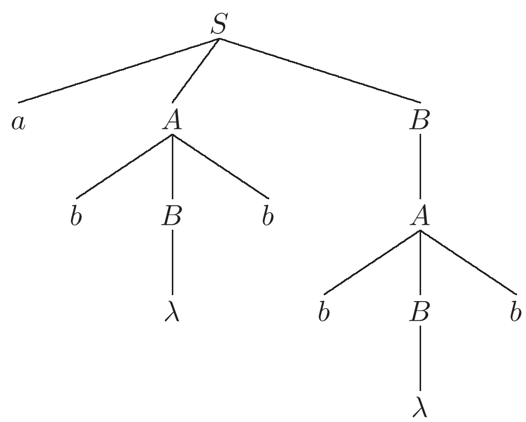
**مثال** ▼

گرامر $G$ با قواعد $S \to aAB,\ \ A \to bBb,\ \ B \to A|\lambda$ را در نظر بگیرید:

یک درخت اشتقاق جزیی برای فرم جمله‌ای $abBbB$ از $G$:




یک درخت اشتقاق برای جمله‌ی $abbbb$:




▲

**قضیه**

اگر $G = (V, T, S, P)$ یک گرامر مستقل از متن باشد، در این صورت به ازای هر $w \in L(G)$ یک درخت اشتقاق برای $G$ داریم که حاصل آن $w$ است و برعکس حاصل درخت اشتقاق در $L(G)$ است.

اگر $t_G$ یک درخت اشتقاق جزیی برای $G$ با ریشه‌ی $S$ باشد، آن‌گاه حاصل $t_G$ یک شکل جمله‌ای از $G$ است.

◄ اثبات.

برای اثبات اینکه برای هر فرم جمله‌ای از $L(G)$ یک درخت اشتقاق جزیی وجود دارد، از استقرا بر روی تعداد گام‌های اشتقاق استفاده می‌کنیم:

- **پایه:** این گزاره برای هر فرم جمله‌ای که در یک گام مشتق می‌شود، صحیح است (زیرا اگر $S \to u \in P$، آن‌گاه $S \Rightarrow^{۱} u$).
- **فرض:** برای هر فرم جمله‌ای که در $n$ گام به دست می‌آید یک درخت اشتقاق جزیی وجود دارد ($S \Rightarrow^n w$).
- **حکم:** هر رشته‌ی $w$ که در $n + ۱$ قدم مشتق می‌شود باید به گونه‌ای باشد که در $n$ گام داشته باشیم:

$$S \Rightarrow^* xAy,\ \ x, y \in (V \cup T)^*,\ \ A \in V$$

$$xAy \Rightarrow xa_۱a_۲ \dots a_m y = w,\ \ a_i \in V \cup T$$

بر اساس فرض استقرا یک درخت اشتقاق جزیی با حاصل $xAy$ وجود دارد و چون گرامر باید قاعده‌ای مانند $A \to a_۱a_۲ \dots a_m$ داشته باشد، با توسعه‌ی برگ $A$ یک درخت اشتقاق جزیی به دست می‌آوریم که حاصل آن $xa_۱a_۲ \dots a_m y$ است.

بنابراین با استفاده از استقرا ادعا می‌کنیم که نتیجه برای همه‌ی فرم‌های جمله‌ای صحیح است.

به همین ترتیب می‌توانیم نشان دهیم که هر درخت اشتقاق یک فرم جمله‌ای را نشان می‌دهد.

■

درخت اشتقاق نشان می‌دهد که چه قواعدی برای به دست آوردن یک جمله استفاده شده است، اما ترتیب استفاده از آنها را نشان نمی‌دهد. به عبارت دیگر ترتیب به کارگیری قواعد در هر مرحله در نتیجه‌ی نهایی تاثیری ندارد.

## ۵-۲ تجزیه و ابهام

**تعریف**

تجزیه‌گر (*parser*)، الگوریتمی است که برای رشته‌ی $w$ یک اشتقاق می‌یابد و یا می‌گوید اشتقاق ممکن نیست.

### ۵-۲-۱ جستجوی جامع به عنوان یک الگوریتم عضویت

برای تشخیص $w \in L(G)$ می‌توانیم

- ابتدا اشتقاق‌های یک مرحله‌ای یعنی $S \Rightarrow x$ را بررسی کنیم،
- سپس اشتقاق‌های دو مرحله‌ای،
- $\ldots$
- اشتقاق‌های $n = |w|$ مرحله‌ای و $\ldots$

این روش یک اشکال دارد و آن این است که اگر $w \notin L(G)$ این روال خاتمه نمی‌یابد.
اگر گرامر مورد نظر قاعده‌ی $\lambda$ و قاعده‌ی یکه $A \to B$ را نداشته باشد، این روال پس از $|w|$ مرحله متوقف می‌شود.

**قضیه**

اگر $G = (V, T, S, P)$ یک گرامر مستقل از متن باشد که قواعدی به شکل $A \to B$ یا $A \to \lambda$ $(A, B \in V)$ را نداشته باشد، آنگاه روش جستجوی کامل می‌تواند به صورت الگوریتمی در آید که برای هر $w \in \Sigma^*$ یا یک تجزیه برای $w$ تعیین می‌کند و یا به ما می‌گوید که تجزیه ممکن نیست.

◄ **اثبات.**

برای هر فرم جمله‌ای، **طول** و **تعداد پایانه‌های** آن را در نظر می‌گیریم.
هرگام اشتقاق حداقل یکی از این دو مورد را افزایش می‌دهد (به دلیل عدم وجود قواعد یکه و تهی).
از آنجا که نه طول فرم جمله‌ای و نه تعداد پایانه‌ها نمی‌تواند بیش از $|w|$ باشد، یک اشتقاق نمی‌تواند بیش از $2|w|$ مرحله داشته باشد
و تا آنجا یا تجزیه به طور موفق به پایان رسیده است و یا $w$ نمی‌تواند توسط آن گرامر تولید شود.

$$\text{تعداد دورها} \leq |w| + |w| = 2|w|$$

■

کاربرد عملی جستجوی جامع برای تجزیه بسیار محدود است، زیرا تعداد فرم‌های جمله‌ای بسیار زیاد است. برای محاسبه‌ی حد بالای تعداد فرم‌های جمله‌ای با فرض استفاده از اشتقاق چپ‌ترین،

در دور اول حداکثر $|P|$ فرم جمله‌ای،

در دور دوم حداکثر $|P|^{۲}$ فرم جمله‌ای،

...،

در حداکثر دور $۲|w|$ حداکثر $|P|^{۲|w|}$ فرم جمله وجود دارد.

در مجموع حداکثر تعداد کل فرم‌های جمله‌ای می‌شود:

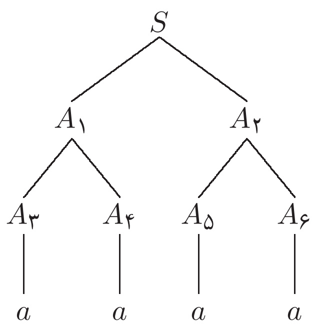
$$M = |P| + |P|^{۲} + \ldots + |P|^{۲|w|} = |P| \frac{۱ - |P|^{۲|w|}}{۱ - |P|}$$

**مثال**

مثالی از بدترین حالت تعداد فرم‌های جمله‌ای را می‌توان درگرامر زیر مشاهده کرد:

$$\begin{array}{lcl} S & \rightarrow & A_{۱}A_{۲} \\ A_{۱} & \rightarrow & A_{۳}A_{۴} \\ A_{۲} & \rightarrow & A_{۵}A_{۶} \\ A_{۳} & \rightarrow & a \\ A_{۴} & \rightarrow & a \\ A_{۵} & \rightarrow & a \\ A_{۶} & \rightarrow & a \end{array}$$

که درخت اشتقاق آن برای رشته‌ی $aaaa$ در شکل زیر رسم شده است:




**قضیه** به ازای هر گرامر مستقل از متن، الگوریتمی وجود دارد که هر رشته‌ی $w \in L(G)$ را در تعداد مراحلی که متناسب با $|w|^{۳}$ است، تجزیه می‌کند.

اثبات این قضیه بر پایه‌ی روش CYK برای تجزیه است که در فصل بعدی تشریح می‌شود.

**تعریف**

**گرامر ساده** گرامر مستقل از متن $G = (V, T, S, P)$ یک گرامر ساده *(simple grammar, s-grammar)* نام دارد اگر و فقط اگر تمامی قواعد آن به فرم

$$A \to ax \quad , A \in V, a \in T, x \in V^*$$

باشد و هر زوج $(A, a)$ حداکثر یک مرتبه در $P$ ظاهر شود.

**مثال**

گرامر $S \to aSbSSc \mid b$ یک گرامر ساده است ولی $S \to aS \mid bSS \mid aSS \mid C$ این‌گونه نیست.

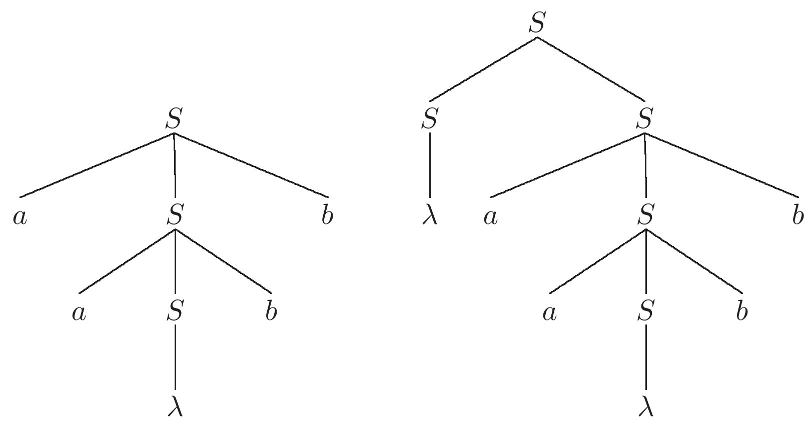
### ۵-۲-۲ ابهام در گرامر و زبان

**تعریف**

**گرامر مبهم** گرامر مستقل از متن $G$ را مبهم *(ambiguous)* می‌نامیم اگر و فقط اگر حداقل یک $w \in L(G)$ وجود داشته باشد که حداقل دو درخت اشتقاق متفاوت داشته باشد.
به عبارت دیگر، ابهام به معنی وجود دو یا چند اشتقاق چپ‌ترین یا راست‌ترین برای $G$ است.

**مثال**

گرامر $S \to aSb|SS|\lambda$ مبهم است: جمله‌ی $aabb$ دو درخت اشتقاق متفاوت دارد:




بنابراین حداقل دو اشتقاق چپ‌ترین (راست‌ترین) برای این رشته وجود خواهد داشت.

**تعریف** **زبان ذاتاً مبهم** زبان $L$ را یک زبان ذاتاً مبهم *(inherently ambiguous)* می‌گوییم، اگر همه‌ی گرامرهای تولیدکننده‌ی $L$ مبهم باشند.

---

**مثال**

زبان $\{a^n b^m c^m\} \cup \{a^n b^n c^m\} = L$ یک زبان ذاتاً مبهم است.

---

**تعریف** **زبان غیرمبهم** زبان $L$ را غیرمبهم می‌گوییم اگر و فقط اگر حداقل یک گرامر غیرمبهم برای آن موجود باشد.

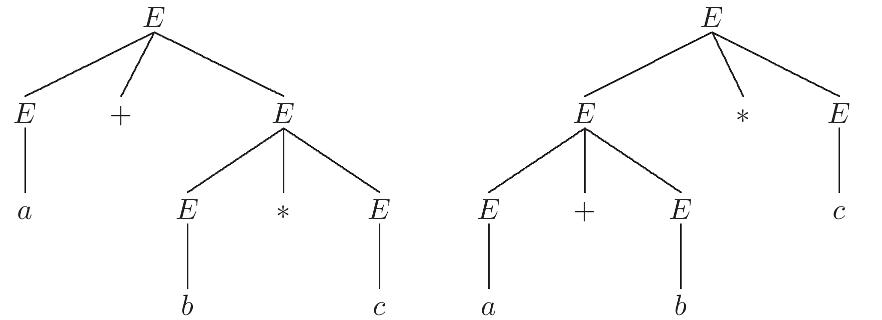
- زبان مبهم نداریم (زبان ذاتاً مبهم داریم).
- مساله‌ی تشخیص ابهام در یک گرامر مستقل از متن، تصمیم‌ناپذیر است، یعنی هیچ الگوریتمی برای تشخیص یا رفع ابهام وجود ندارد.
- هیچ زبان منظمی ذاتاً مبهم نیست.

**مثال**

گرامر $G = (E, a, b, c, +, *, (, ), E, P)$ با قواعد

$$E \to E + E | E * E | (E) | a | b | c$$

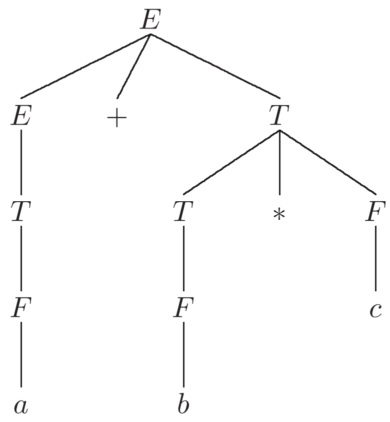
برای عبارت‌های حسابی، مبهم است: برای رشته‌ی $a + b * c$ دو درخت اشتقاق وجود دارد:



می‌توان با بازنویسی گرامر، یک گرامر معادل غیر مبهم برای این گرامر پیدا کرد:

$$E \to E + T | T,\ \ T \to T * F | F,\ \ F \to (E) | a | b | c$$

در این گرامر رشته‌ی یاد شده تنها یک درخت اشتقاق دارد:




▲

### تشخیص ابهام یک گرامر مستقل از متن به عنوان یک مساله‌ی تصمیم‌ناپذیر

**تعریف**

**گراف گرامر** اگر $G$ یک گرامر مستقل از متن باشد، گراف جهت دار $g_G = (N, E)$ که در آن $N$ مجموعه‌ی راس‌ها، مجموعه‌ی فرم‌های جمله‌ای گرامر $G$ و $E$ مجموعه‌ی یال‌های برچسب‌دار گراف به صورت

$$E = \{(x, r, y) : x \underset{G\ LM}{\overset{r}{\Rightarrow}} y, \qquad x, y \in N\}$$

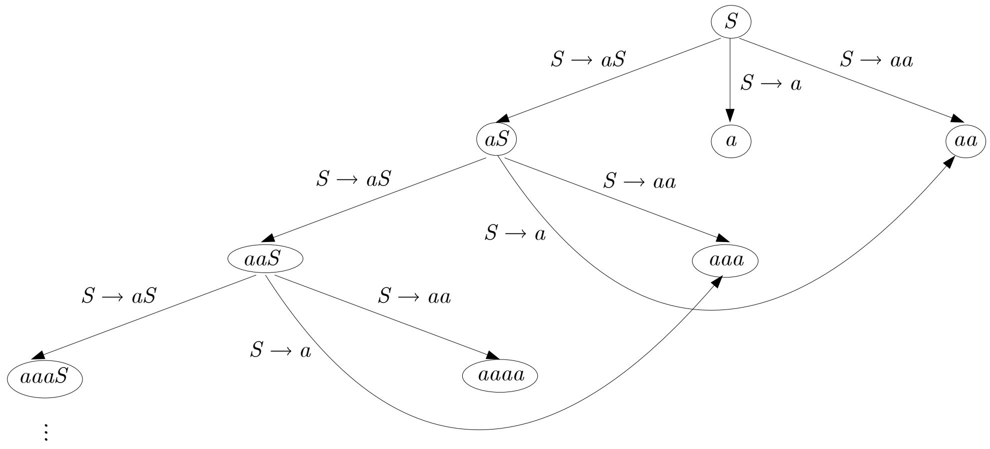
را گراف (چپ) گرامر $G$ می‌گوییم.

- گراف گرامر با درخت تجزیه متفاوت است: درخت تجزیه چگونگی ایجاد یک رشته توسط گرامر را مشخص می‌کند،
  گراف گرامر کلیه‌ی اشتقاق‌های چپ (یا راست) ممکن درون یک گرامر را مشخص می‌کند.
- در درخت تجزیه هر گره معرف یک نماد گرامر است، در صورتی که در گراف گرامر هر گره معرف یک فرم جمله‌ای است.
- تجزیه‌گر (parser) برای یافتن چگونگی اشتقاق، فرم‌های جمله‌ای در گراف گرامر را جستجو می‌کند.
  بسته به نحوه‌ی جستجو، روش‌های مختلفی برای تجزیه خواهیم داشت:
  - بالا به پایین
  - پایین به بالا
  - عمق ـ اول
  - عرض ـ اول
- گرامر مبهم، در گراف خود حداقل دارای یک حلقه است.
- اگر گرامری مبهم نباشد، گراف آن یک درخت است.

گرامر $G$ مبهم است اگر و فقط اگر گراف آن حداقل حاوی یک حلقه باشد.

**مثال**

گراف گرامر مبهم $S \to aS|a|aa$ حاوی حلقه است.




## ۵-۳ زبان‌های برنامه‌سازی و گرامرهای مستقل از متن

- تعریف زبان‌های برنامه‌سازی با گرامر متداول است.
- همه‌ی خواص زبان‌های برنامه‌سازی را نمی‌توان با گرامرهای مستقل از متن ساده توصیف کرد ($\Leftarrow$ نیاز به گرامرهای پیچیده‌تر مانند LL یا LR)
- همه‌ی خواص زبان‌های برنامه‌سازی را نمی‌توان با گرامرهای مستقل از متن توصیف کرد. برخی از این خواص با گرامرهای سطح بالاتر قابل توصیف هستند ($\Leftarrow$ تحلیل معنایی)
- گرامر زبان‌های برنامه‌سازی باید غیر مبهم باشد.

### ۵-۳-۱ نمادگذاری BNF برای گرامرهای مستقل از متن

نمادگذاری BNF (Backus-Naur Form) از نمادهای زیر استفاده می‌کند:

- `<variable>` یک متغیر (ناپایانه) گرامر
- `::=` به جای نماد $\to$
- پایانه‌ها بدون علامت اضافی به کار می‌روند.

اسامی ناپایانه‌ها به شکل معنی‌دارتری انتخاب می‌شود.

**مثال**

نمادگذاری BNF برای چند قاعده‌ی گرامری زبان:

```
<if-statement> ::= if <expression> <then-clause> <else-clause>
<expression>   :: = <term> | <expression> + <term>
<term>         :: = <factor> | <term> + <factor>
```