گرامر G1 را در نظر بگیرید؛ آیا رشته 000111 توسط این گرامر تولید میشود ؛

G1 : $S \rightarrow A$

$A \rightarrow 0A1 | 01$

$S \rightarrow A \rightarrow 0A1 \rightarrow 00A11 \rightarrow 000111$

زبان گرامر G1 ،

$L(G1) = \{0^n1^n \mid n>=1\}$

عملیات انجام شده به ترتیب بر اساس قواعد تولید را اشتقاق می گویند.

بطور مثال بر اساس گرامر G1 مثال قبل ؛

$S \rightarrow A \rightarrow 01$

اگر نتوان برای رشته ای اشتقاق نوشت می توان گفت آن رشته جزو زبان گرامر نیست.

کدامیک از رشته ای زیر عضو زبان L(G2) می باشد

زبان گرامر G2 را بنویسید.

G2 :

S → A

A → 0A| 0 | 1B

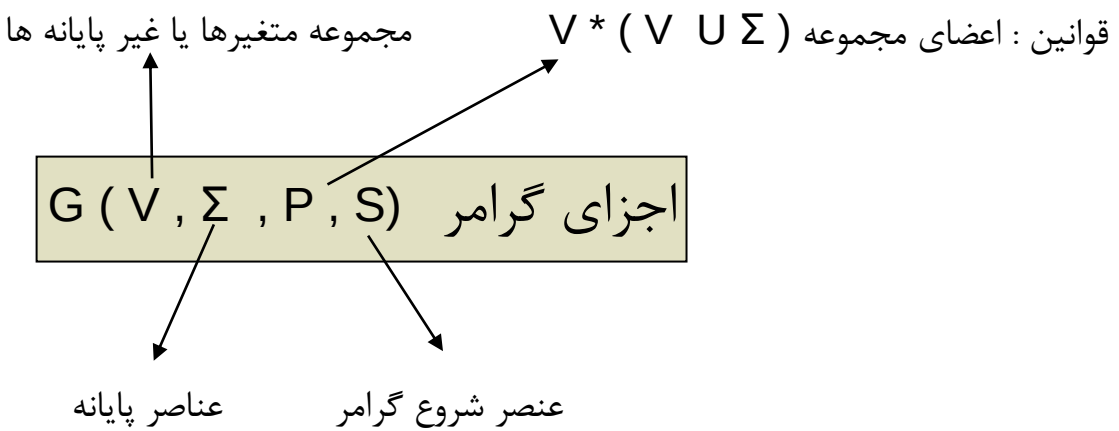
B → 1B| 1

- **0011** 
- **101** 
- **0110** 

زبان گرامر G2 ،

$$L(G2) = \{0^n1^m \mid m>=2 \cup (m=0 \cap n>=1\}$$

گرامر مستقل از متن



گرامر مستقل از متن گرامری است که قواعد آن به فرم زیر باشد:

$$A \rightarrow \alpha \ , A \varepsilon V, \alpha \varepsilon (\Sigma + V)^*$$

- ✓ در گرامرهای مستقل از متن می توان جایگزینی متغیرهای سمت چپ یک قانون را در هر زمانی که این متغیر در یک شکل جمله ای دیده می شود، انجام داد، و این بستگی به بقیه شکل جمله ندارد. (مجاز به انتخاب فقط یک متغیر در سمت چپ قانون هستیم)

- ✓ یکی از کاربردهای این زبان، بررسی صحت پرانتز گذاری در عبارات ریاضی و یا آکولادها در زبان برنامه نویسی C میباشد.
- ✓ با استفاده از این نوع گرامرها، زبانهای پیچیده تری را می توان تعریف کرد.
- ✓ گرامر های مستقل از متن در طرف چپ همان محدودیت گرامرهای منظم را دارند ولی در طرف راست آزادتر هستند.
- ✓ توجه کنید که زبانهای منظم زیرمجموعه زبانهای مستقل از متن اند.
- ✓ هر زبان متناهی منظم (و در نتیجه) مستقل از متن است.

زبانهای مستقل از متن $\subset$ زبانهای منظم

مثال گرامرهای مستقل از متن

$S \rightarrow aSb \mid aSbb$

$S \rightarrow abS \mid cB$
$B \rightarrow bB \mid b$

$S \rightarrow aSdd \mid A$
$A \rightarrow bAc \mid bc$

$S \rightarrow aSa \mid aBa$
$B \rightarrow bB \mid b$

گرامر $G = (\{S\},\{a,b\},S,P)$ ، با قوانین:

$S \rightarrow abB$
$B \rightarrow bbAa$
$A \rightarrow aaBb$
$A \rightarrow \lambda$

مستقل از متن است ولی منظم نیست.

گرامر مستقل از متن نمونه اشتقاق‌های یک رشته

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

رشته ababaa

$S \rightarrow AA$	$S \rightarrow AA$	$S \rightarrow AA$	$S \rightarrow AA$
$\rightarrow aA$	$\rightarrow AAAA$	$\rightarrow Aa$	$\rightarrow aA$
$\rightarrow aAAA$	$\rightarrow aAAA$	$\rightarrow AAAa$	$\rightarrow aAAA$
$\rightarrow abAAA$	$\rightarrow abAAA$	$\rightarrow AAbAa$	$\rightarrow aAAa$
$\rightarrow abaAA$	$\rightarrow abaAA$	$\rightarrow AAbaa$	$\rightarrow abAAa$
$\rightarrow ababAA$	$\rightarrow ababAA$	$\rightarrow AbAbaa$	$\rightarrow abAbAa$
$\rightarrow ababaA$	$\rightarrow ababaA$	$\rightarrow Ababaa$	$\rightarrow ababAa$
$\rightarrow ababaa$	$\rightarrow ababaa$	$\rightarrow ababaa$	$\rightarrow ababaa$

اشتقاق Derivation

فرآیند تولید رشته از گرامر با شروع از عنصر ابتدای گرامر و استفاده از قوانین.

- ✓ گرامرهای مستقل از متن غیر خطی، یک اشتقاق ممکن است دارای فرم های جمله ای با بیش از یک متغیر باشد.
- ✓ در این موارد در ترتیب جایگزینی متغیرها آزاد هستیم.

اشتقاق مکانیزمی است جهت؛

- تولید رشته های زبان
- ارزیابی تعلق یا عدم تعلق رشته ها به یک زبان

انواع اشتقاق

از چپ: در هر قدم انجام جایگزینی روی سمت چپ ترین غیرپایانه

اگر صورت جمله ورودی اسکن شده و از چپ به راست تعویض شود، به نام اشتقاق چپ‌ترین نامیده می‌شود.

از راست: در هر قدم انجام جایگزینی روی سمت راست ترین غیرپایانه

اگر یک ورودی را از سمت راست به چپ اسکن کرده و با قواعد ترکیبی جایگزین کنیم، به نام اشتقاق راست‌ترین نامیده می‌شود.

درخت اشتقاق

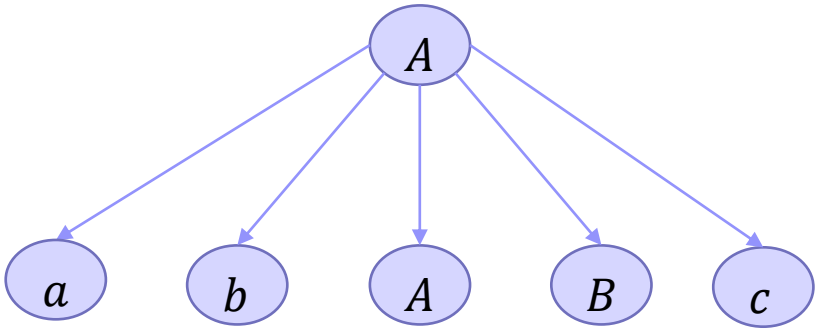
روش دوم برای نمایش اشتقاق ها، استفاده از **درخت تجزیه یا اشتقاق یا نحوی** است

در این روش ترتیب بکارگیری قوانین اهمیت ندارد.

درخت اشتقاق در واقع درخت مرتبی است که در آن گره ها با سمت چپ قوانین نامگذاری میشوند.

فرزندان یک گره به معرفی سمت راست آن گره میپردازند و به ترتیب از چپ به راست، سمبل ها و متغیرهای سمت راست می آیند.

برای مثال درخت اشتقاق برای قانون $A \rightarrow abABc$ در تمامی درخت های اشتقاق، با شروع از ریشه با متغیر شروع گرامر نامگذاری میشود و خاتمه یافتن به برگ هایی که پایانی ها و یا λ هستند، درخت تکمیل میشود.



فرض کنید $G = (V,T,S,P)$ یک گرامر مستقل از متن باشد. یک درخت مرتب، **درخت اشتقاقی** برای G خواهد بود اگر و تنها اگر خواص زیر را داشته باشد:

۱- در درختهای اشتقاق، ریشه نماد آغازگرگرامر است. (ریشه دارای نام S است.)

۲- برگهای درخت اشتقاق، پایانه های گرامر هستند. (هر یک از برگها دارای نام از $T \cup \{\lambda\}$ باشد.)

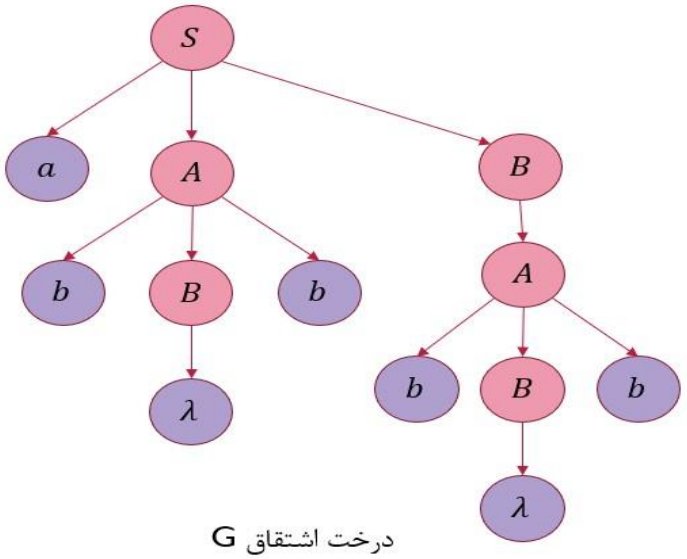
۳- غیر پایانه ها نیز گره های میانی درخت هستند.(هر یک از گره های میانی (گره ای که برگ نباشد) دارای نامی از V است.)

۴- فرزندان هر گره میانی (غیر پایانه ها) بر اساس یکی از قواعدش مشخص می شود.

۵- از کنارهم قرار دادن برگهای درخت از سمت چپ به راست، رشته مشتق شده حاصل می شود. (برگ های دارای نام λ هیچ خواهر و برادری ندارند؛ بدین معنا که گره ای که فرزند آن λ نامگذاری شده باشد، فاقد فرزند دیگری است.

گرامر با قوانین زیر را درنظر بگیرید:

$$S \rightarrow aAB$$
$$A \rightarrow bBb$$
$$B \rightarrow A|\lambda$$



درخت اشتقاق G

abbbb

- ✓ یک درخت تجزیه، **شرکت‌پذیری و تقدم عملگرها** را نشان می‌دهد و **عمیق‌ترین زیردرخت**، در وهله اول پیمایش می‌شود. از این رو عملگر موجود در آن زیردرخت نسبت به عملگری که در گره‌های والدش قرار دارد، تقدم بالاتری دارد.
- ✓ **درخت تجزیه** Parse trees نشان دهنده **چگونگی اشتقاق رشته ای** از زبان از نماد شروع گرامر
- ✓ رشته ای از سمبل ها که با خواندن برگهای درخت از چپ به راست و حذف تمامی λ های مسیر ایجاد شود، اصطلاحاً تولید یا تولید درخت نامیده میشود.
- ✓ تولید رشته ای از پایانی ها که با پیمایش اول عمق ارائه میشود را میتوان معادل عبارت توصیفی چپ به راست در نظر گرفت.

ابهام (گنگی)

- تجزیه (پویش) [parsing] یعنی یافتن یک سری قانون که با استفاده از آن ها $w \in L(G)$ مشتق می شود.
- گرامر مستقل از متن $G = (V,T,S,P)$ را در صورتی **گنگ** میگویند که یک $w \in L(G)$ باشد ، که حداقل دو درخت اشتقاق متفاوت داشته باشد. (وجود دو اشتقاق راست یا دو اشتقاق چپ رای یک رشته در گرامر)
- تعریف: اگر بتوان برای یک رشته، بر اساس یک گرامر، بیش از یک درخت اشتقاق رسم نمود، آن گرامر را مبهم می‌گویند.

✓ آگاهی از خصوصیات گرامرهای مبهم در طراحی و پیاده سازی کامپایلرها نقش مهمی دارد.

✓ در حالت کلی، الگوریتمی برای تشخیص اینکه گرامری مبهم است یا خیر وجود ندارد اما در شرایط خاصی می توان ابهام را مستقیماً تشخیص داد.

✓ امکان استفاده از گرامرهای مبهم در تحلیل نحوی (به طور مستقیم) وجود ندارد.

✓ چنانچه بخشی از گرامر یک زبان برنامه سازی مبهم باشد، باید قواعد رفع ابهام آن مشخص شده باشد.

ابهام یا گنگی درگرامرها و زبانها

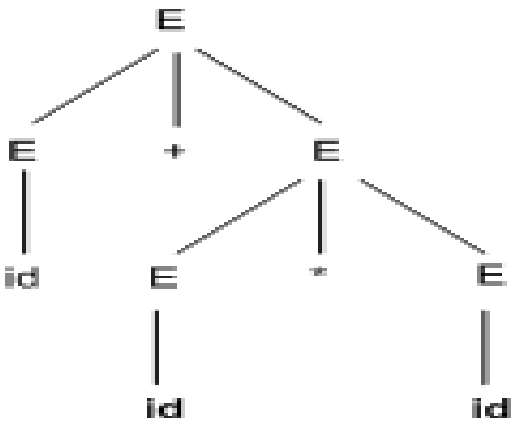
این زبان توسط یک گرامر مبهم تولید شده است و گفته می‌شود که دارای ابهام ذاتی inherently ambiguousاست. ابهام در گرامر برای ساخت کامپایلر خوب نیست. هیچ روشی برای کشف و حذف ابهام به طور خودکار وجود ندارد؛ اما با بازنویسی کل گرامر به صورت بی‌ابهام یا با تعیین و پیروی از قواعد خاص تقدم و شرکت‌پذیری می‌توان آن را از بین برد.

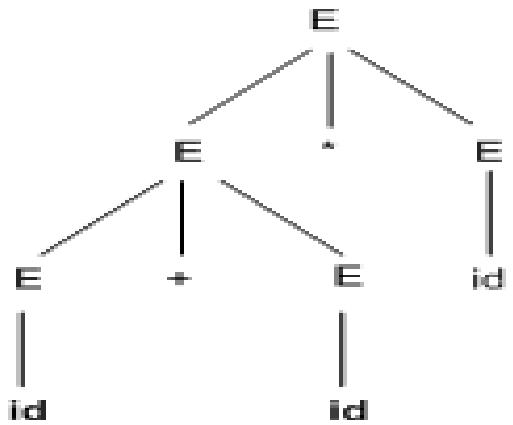
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$

مثال id + id * id

$E \rightarrow E + E \rightarrow id + E \rightarrow id + E * E \rightarrow id + id * E \rightarrow \mathbf{id + id * id}$

$E \rightarrow E * E \rightarrow id + id * E \rightarrow \mathbf{id + id * id}$





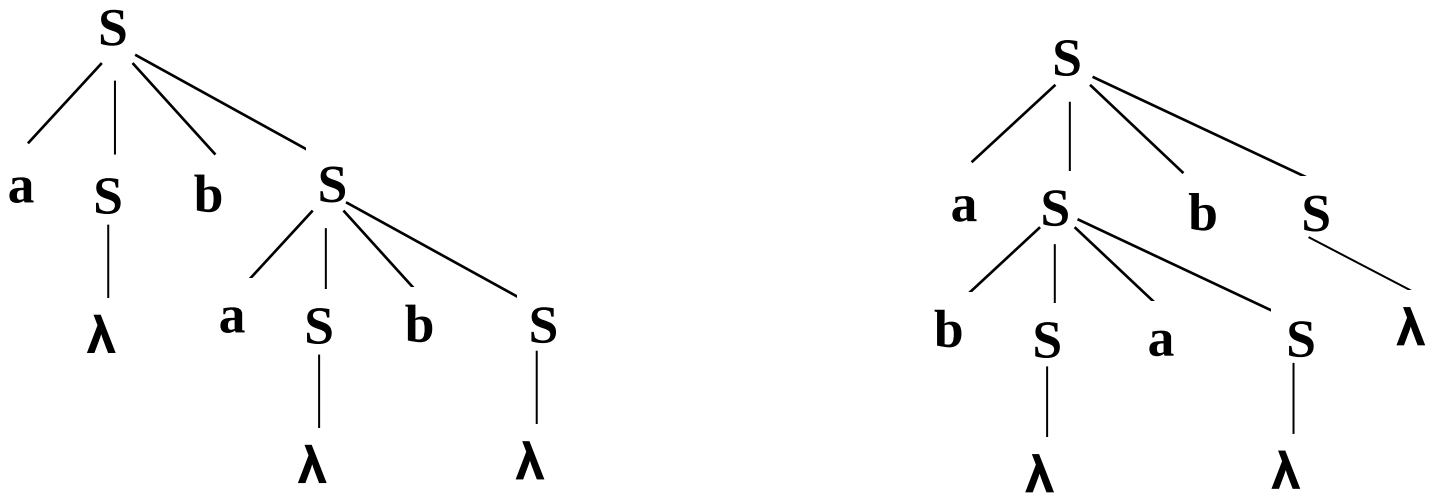
مثال: گرامر $S \rightarrow aSbS|\ bSaS|\ \lambda$ مبهم است، زیرا برای رشته ای مانند abab دو درخت اشتقاق زیر وجود دارد:

$S \rightarrow a\mathbf{S}bS \rightarrow ab\mathbf{S} \rightarrow aba\mathbf{S}bS \rightarrow abab\mathbf{S} \rightarrow abab$ اشتقاق از چپ

$S \rightarrow aSb\mathbf{S} \rightarrow aSbaSb\mathbf{S} \rightarrow aSba\mathbf{S}b \rightarrow a\mathbf{S}bab \rightarrow abab$ اشتقاق از راست

مشاهده میشود که هر دو اشتقاق رشته یکسان تولید میکنند.

برای یکسان سازی فرایند اشتقاق، بهتر است متغیرها با ترتیب خاصی جایگزین شوند



دو درخت اشتقاق متفاوت برای رشته abab

۱-شرکت‌پذیری

اگر یک عملوند در هر دو سوی خود عملگرهایی داشته باشد، این که این عملوند از عملگر کدام سمت استفاده کند به شرکت‌پذیری آن عملگرها بستگی دارد. اگر عملیات به صورت شرکت‌پذیر از چپ باشد در این صوت عملوند از سوی عملگر سمت چپ برداشته می‌شود و اگر شرکت‌پذیر از راست باشد، عملگر راست، عملوند را انتخاب می‌کند.

نمونه

عملیات‌هایی مانند جمع، ضرب، تفریق و تقسیم، **شرکت‌پذیر از چپ** هستند. اگر عبارتی شامل موارد زیر باشد:

id op id op id

به صورت زیر ارزیابی می‌شود:

(id op id) op id

(id + id) + id

عملیات‌هایی مانند توان شرکت‌پذیر از راست هستند، یعنی ترتیب ارزیابی در همان عبارت به صورت زیر خواهد بود:

id op (id op id)

id ^ (id ^ id)

۲-تقدم

اگر دو عملگر دارای یک عملوند مشترک باشند، تقدم عملگرها تعیین می‌کند که کدام یک عملوند را بر می‌دارند. یعنی ۴*۳+۲ می‌تواند دو درخت تجزیه داشته باشد، یکی متناظر با ۴*(۳+۲) و دیگری که متناظر با (۴*۳)+۲ است. با تعیین تقدم میان عملگرها، این مسئله را به راحتی می‌توان حل کرد. همانند مثال قبلی ازنظر ریاضیاتی * (ضرب) نسبت به +(جمع) تقدم دارد و از این رو عبارت ۴*۳+۲ همواره به صورت زیر تفسیر می‌شود:

(۳ * ۴) + ۲

این روش‌ها احتمال ابهام را در زبان و گرامر آن کاهش می‌دهند.

رفع ابهام

- مثال : گرامر $G = (V,T,E,P)$ را با $V = \{ E,I \}$ و $T = \{ a,b,c, +, *, (,) \}$ و قوانین زیر

$$E \to I \mid E + E \mid E * E \mid (E)$$
$$I \to a \mid b \mid c$$

گنگ است. در پیمایش عبارتهای $a + b * c$ و $a * b + c$ گنگ میباشد.

یکی از روشهای رفع ابهام برای عملگرها، استفاده از **قوانین تقدم عملگرهاست.**

برای رفع ابهام مثال قبل به شکل زیر عمل میکنیم.

- تعریف دو متغیر Term و Factor
- یک Termتشکیل شده از حاصل ضرب یا حاصل تقسیم یک ترم با یک فاکتور
- یک Factor میتواند یک متغیر ریاضی و یا یک عبارت داخل پرانتز باشد.
- یک عبارت expr هم تشکیل شده از حاصل جمع یا تفریق ترم ها

expr → expr + term | expr – term | term

term→term* factor | term/ factor| factor

factor→(expr) | Identifier

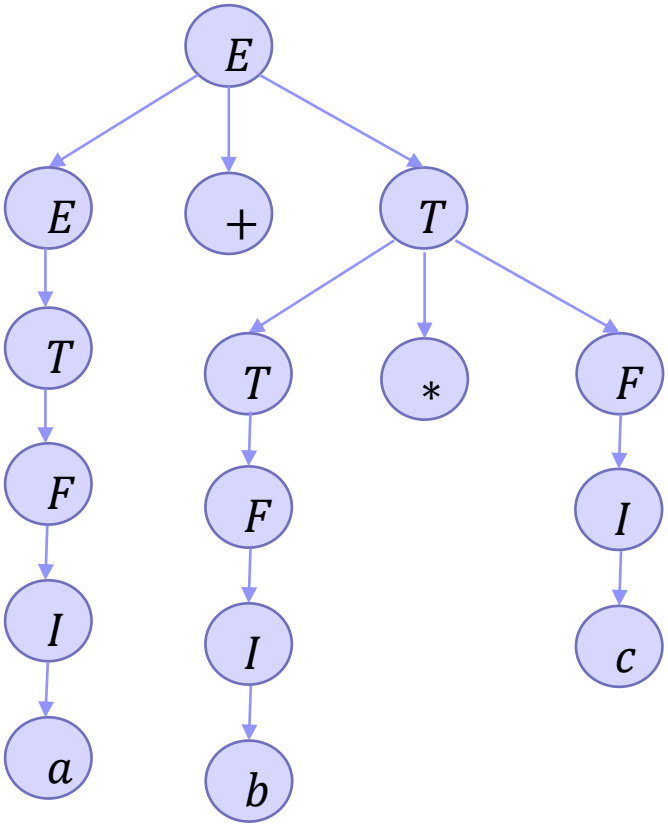
Identifier→a|b|…|z

$$E \to I \mid E + E \mid E * E \mid (E)$$
$$I \to a \mid b \mid c$$

$$E \to T \mid E + T \mid E - T$$
$$T \to F \mid T * F \mid T / F$$
$$F \to (E) \mid I$$
$$I \to a \mid b \mid c$$

- درخت اشتقاق برای عبارت a+b*c

$$E \rightarrow T|E+T\ |E-T$$
$$T \rightarrow F|T*F\ |T/F$$
$$F\ \rightarrow\ (E)|I$$
$$I\ \rightarrow a|b|c$$



- هیچ درخت اشتقاق دیگری برای این عبارت نمیتوان ترسیم کرد لذا ابهامی وجود ندارد.

زبان های مستقل ازمتن وزبان های برنامه سازی

- یکی از کاربردهای مهم نظریه زبانهای صوری در تعریف زبانهای برنامه سازی و همچنین ساخت مفسرها و کامپایلرها برای آنهاست.
- باید تعریف دقیق و روشنی از زبان برنامه سازی وجود داشته باشد تا نقطه آغازین برای نوشتن برنامه مترجم گردد.
- زبانهای منظم و مستقل از متن هر دو طراحی یک زبان برنامه سازی نقش کلیدی دارند.
- زبانهای برنامه سازی، اغلب بوسیله گرامر توصیف میشوند.
- استفاده از یک قرارداد برای تعریف گرامرها به منظور نوشتن زبانهای برنامه نویسی مرسوم است.
- این قرارداد فرم باکوس نائور (Backus-Naur Form) یا BNF گفته میشود.
- این فرم بطور خلاصه همان مجموعه نشانه هایی است که قبلاً استفاده میکردیم ولی با کمی تفاوت ظاهری.