اصول طراحی کامپایلر

درس‌نامه‌ی ۴

کاظم فولادی
http://kazim.fouladi.ir

ویراست اول: ۱۳۸۸
ویراست دوم: ۱۳۹۳

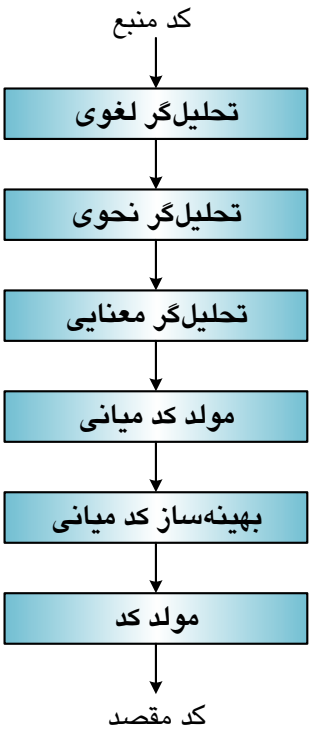
فهرست مطالب

۴ تحلیل نحوی

SYNTAX ANALYSIS

۴-۱ مقدمه

تحلیل نحوی دومین مرحله‌ی فرایند کامپایل است.



شکل ۴-۱: تحلیل نحوی به عنوان دومین مرحله‌ی فرایند کامپایل

- در تحلیل نحوی، توکن‌ها در ساختار گرامری زبان گروه‌بندی می‌شوند.
- در این مرحله ساختار سلسله‌مراتبی کد منبع به دست می‌آید.

شکل ۴-۲: مرحله‌ی تحلیل نحوی

۴-۲ بازنمایی نحو زبان با گرامرهای مستقل از متن

هر زبان برنامه‌سازی، قواعدی دارد که ساختار نحوی برنامه‌های درست را مشخص می‌کند.

- عبارت‌های منظم برای بازنمایی نحو محدودیت دارند (مانند بازنمایی پرانتزگذاری صحیح).
- گرامرهای مستقل از متن برای بازنمایی نحو زبان‌های برنامه‌سازی مناسب هستند.

۴-۲-۱ گرامر

گرامر، ابزاری برای تولید رشته‌های یک زبان است

گرامر G یک چهارتایی مرتب به صورت

$$G = (N, T, S, P)$$

است که در آن

N مجموعه‌ای متناهی از **ناپایانه‌ها** *(non-terminal)* یا **متغیرها** *(variable)*

T مجموعه‌ای از **پایانه‌ها** (terminal) (الفبای گرامر، مجموعه‌ی توکن‌ها)

S یک عنصر خاص از N با نام **نماد شروع** *(start symbol)*

P مجموعه‌ای متناهی از **قواعد تولید** *(production rule)*

با فرض اینکه N و T ناتهی و مجزا هستند $(N \cap T = \varnothing)$

قواعد تولید

قواعد تولید اساس تعریف یک گرامر است.

هر **قاعده‌ی تولید** (قاعده) یک زوج مرتب به صورت $(x, y) \in P$ است که به شکل

$$x \to y$$

نمایش داده می‌شود که در آن

$x \in (N \cup T)^+$ (هر ترکیبی از پایانه‌ها و ناپایانه‌های گرامر بجز رشته‌ی تهی)

$y \in (N \cup T)^*$ (هر ترکیبی از پایانه‌ها و ناپایانه‌های گرامر)

از قواعد تولید در عمل اشتقاق (Derivation) برای تولید رشته استفاده می‌شود.

مثال

$$G = (N, T, S, P)$$

$$T = \{۰, ۱\}, \quad N = \{S\}, \quad P = \{S \to ۰S۱, S \to ۰۱\}$$

$$S \Rightarrow ۰۱$$

$$S \Rightarrow ۰S۱ \Rightarrow ۰۰۱۱$$

$$S \Rightarrow ۰S۱ \Rightarrow ۰۰S۱۱ \Rightarrow ۰۰۰۱۱۱$$

$$\dots$$

$$S \Rightarrow^* ۰^n۱^n$$

(* اشتقاق طی صفر یا چند مرحله را نشان می‌دهد.)

$$L(G) = \{۰^n۱^n : n \geq ۱\}$$

اشتقاق

اشتقاق (Derivation): اعمال یک قاعده‌ی تولید بر روی یک فرم جمله‌ای

اگر رشته‌ی w به صورت $w = uxv$ باشد و قاعده‌ی تولید $x \to y$ را داشته باشیم، می‌توانیم آن را بر روی این رشته اعمال کنیم و رشته‌ی z به صورت $z = uyv$ را به دست آوریم. به این عمل اشتقاق می‌گوییم و آن را به صورت

$$w \Rightarrow z$$

نشان می‌دهیم و می‌گوییم w، z را مشتق می‌کند یا z از w مشتق می‌شود.

می‌توان قواعد تولید را به طور پی در پی و به تعداد دلخواه استفاده کرد تا رشته‌ی مورد نظر به دست آید:

$$w_۱ \Rightarrow w_۲ \Rightarrow \dots \Rightarrow w_n$$

و گفته می‌شود که w_n از $w_۱$ مشتق می‌شود. می‌نویسیم

$$w_۱ \Rightarrow^* w_n$$

که به معنی آن است که w_n طی صفر یا چند مرحله از $w_۱$ تولید می‌شود.

قاعده‌ی اشتقاق هرگاه سمت چپ یک قاعده با یک زیررشته از یک فرم جمله‌ای تطابق پیدا کند، می‌توانیم سمت راست آن قاعده را با آن زیررشته جایگزین کنیم.

$$w \Rightarrow z \quad \textbf{iff} \quad w = w_۱uw_۲,\ z = w_۱vw_۲,\ w_۱,\ w_۲ \in (N \cup T)^*,\ u \to v \in P$$

نمادهای عمل اشتقاق برای بیان جزئیات عمل اشتقاق از نمادگذاری‌های زیر استفاده می‌شود:

$\Rightarrow_G$	اشتقاق توسط گرامر G	
$\Rightarrow$	اشتقاق در یک مرحله G	
$\Rightarrow^*$	اشتقاق در صفر مرحله یا بیشتر G	(دارای خاصیت بازتابی و تراگذری)
$\Rightarrow^+$	اشتقاق در یک مرحله یا بیشتر G	(دارای خاصیت تراگذری)
$\Rightarrow^n$	اشتقاق در n مرحله	
$\overset{(r)}{\Rightarrow}$	اشتقاق با قاعده‌ی r	

زبان تولید شده توسط یک گرامر

اگر $G = (N, T, S, P)$ یک گرامر باشد، آن‌گاه مجموعه‌ی

$$L(G) = \{w \in T^* : S \Rightarrow^* w\}$$

زبان تولید شده توسط G نام دارد.

مثال

$$G = (N, T, S, P)$$

$$T = \{a, b\}, \quad N = \{S, A, B\},$$

$$\begin{aligned} S &\rightarrow AB && (۱)\\ A &\rightarrow aA && (۲)\\ A &\rightarrow \epsilon && (۳)\\ B &\rightarrow bB && (۴)\\ B &\rightarrow \epsilon && (۵) \end{aligned}$$

$$S \overset{(۱)}{\Rightarrow} AB \overset{(۲)}{\Rightarrow} aAB \overset{(۲)}{\Rightarrow} aaAB \overset{(۲)}{\Rightarrow} aaaAB \overset{(۳)}{\Rightarrow} aaaB \overset{(۴)}{\Rightarrow} aaabB \overset{(۴)}{\Rightarrow} aaabbB \overset{(۵)}{\Rightarrow} aaabb$$

$$L(G) = \{a^m b^n : m, n \geq ۰\}$$

مثال

$$G = (N, T, S, P)$$

$$T = \{a\}, \quad N = \{S, N, Q, R\}$$

$$\begin{aligned} S &\rightarrow QNQ && (۱)\\ QN &\rightarrow QR && (۲)\\ RN &\rightarrow NNR && (۳)\\ RQ &\rightarrow NNQ && (۴)\\ N &\rightarrow a && (۵)\\ Q &\rightarrow \epsilon && (۶) \end{aligned}$$

$$S \overset{(۱)}{\Rightarrow} QNQ \overset{(۲)}{\Rightarrow} QRQ \overset{(۴)}{\Rightarrow} QNNQ \overset{(۲)}{\Rightarrow} QRNQ \overset{(۳)}{\Rightarrow} QNNRQ \overset{(۴)}{\Rightarrow} QNNNNQ \Rightarrow^* aaaa$$

$$L(G) = \{a^{(۲^n)} : n \geq ۰\}$$

▲

فرم‌های جمله‌ای یک گرامر

اگر $w \in L$ باشد، آن‌گاه

$$S \Rightarrow w_۱ \Rightarrow w_۲ \Rightarrow \ldots \Rightarrow w_n \Rightarrow w$$

را یک اشتقاق از جمله‌ی w می‌گویند.

رشته‌های S، $w_۱$، $w_۲$، ...، w_n و w که حاوی پایانه‌ها و ناپایانه‌ها هستند به **فرم جمله‌ای** (sentential form) موسوم هستند.

مجموعه‌ی همه‌ی فرم‌های جمله‌ای قابل اشتقاق از یک گرامر را با $SF(G)$ نمایش می‌دهیم و به صورت

$$SF(G) = \{W \in (T \cup N)^* : S \Rightarrow^* W\}$$

تعریف می‌کنیم. بدیهی است که

$$L(G) \subset SF(G)$$

۴-۲-۲ هم‌ارزی گرامرها

دو گرامر $G_۱$ و $G_۲$ معادل هستند، اگر و فقط اگر هر دو یک زبان واحد را تولید کنند.

$$G_۱ \equiv G_۲ \quad \textbf{iff} \quad L(G_۱) = L(G_۲)$$

۴-۲-۳ گرامر مستقل از متن

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

$$A \rightarrow x \quad , x \in (N \cup T)^*,\ A \in N$$

باشد.

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

قراردادهای نمادگذاری

از قراردادهای زیر برای خلاصه‌نویسی گرامرها استفاده می‌کنیم:

پایانه‌ها	حروف کوچک ابتدای الفبای انگلیسی	$a, b, c, \ldots$
	رشته‌های سیاه (bold)	$\mathbf{id}, \ldots$
	ارقام	$1, 2, 3, \ldots$
	عملگرها و نمادهای خاص	$+, -, (,), \ldots$
ناپایانه‌ها	حروف بزرگ ابتدای الفبای انگلیسی	$A, B, C, \ldots$
	حرف بزرگ S	
	رشته‌های ایتالیک	$stmt, expr, \ldots$
نمادهای گرامر (پایانه یا ناپایانه)	حروف بزرگ انتهای الفبای انگلیسی	$\ldots, X, Y, Z$
رشته‌ای از نمادهای گرامر	حروف کوچک یونانی	$\alpha, \beta, \gamma, \ldots$
رشته‌ای از پایانه‌ها	حروف کوچک انتهای الفبای انگلیسی	$\ldots, w, x, y, z$
نماد شروع	ناپایانه‌ی سمت چپ اولین قاعده	

نمادگذاری بی. ان. اف (BNF)

Backus-Naur Form

این فرم نمایشی از گرامرهای مستقل از متن، در توصیف گرامرهای شامل تعداد زیادی پایانه و ناپایانه به کار می‌رود.

ناپایانه‌ها	درون براکت گوشه‌دار	`<expr>`
پایانه‌ها	رشته‌های کاراکتری	`number`
نماد جایگزینی	::=	

```
1  <goal>  ::=  <expr>
2  <expr>  ::=  <expr> <op> <expr>
3           |   number
4           |   id
5    <op>  ::=  +
6           |   -
7           |   *
8           |   /
```

در فرم گسترش یافته (EBNF-Extended BNF)، سمت راست قواعد می‌تواند با یک عبارت منظم متشکل از نمادهای گرامر (پایانه یا ناپایانه) نمایش داده شود.

اشتقاق‌های قانونمند

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

- **اشتقاق راست‌ترین: (Right-most Derivation):**

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

در صورت وجود اشتقاق، هم اشتقاق راست‌ترین و هم اشتقاق چپ‌ترین وجود خواهد داشت.

مثال

گرامر با قواعد

$$\begin{aligned} S &\rightarrow aAB \\ A &\rightarrow bBb \\ B &\rightarrow A \mid \epsilon \end{aligned}$$

را در نظر بگیرید. برای رشته‌ی $abbbb$:
یک اشتقاق چپ‌ترین:

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

یک اشتقاق راست‌ترین:

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

درخت اشتقاق

اگر $G = (N, T, S, P)$ 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 $T \cup \{\epsilon\}$ باشد.

۳) هر یک از گره‌های داخلی دارای برچسبی از N باشد.

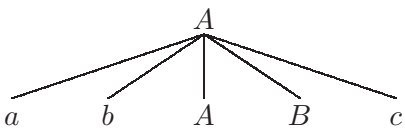
۴) اگر گره‌ای دارای برچسب $A \in N$ باشد و فرزندان آن از چپ به راست به صورت $a_1, a_2, \ldots, a_n$ برچسب‌گذاری شده باشد، آن‌گاه P حاوی قاعده‌ای به شکل $A \rightarrow a_1 a_2 \ldots a_n$ باشد.

۵) گره‌ای که دارای فرزندی با برچسب ϵ 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 $A \rightarrow abABc$:



درخت اشتقاق جزیی

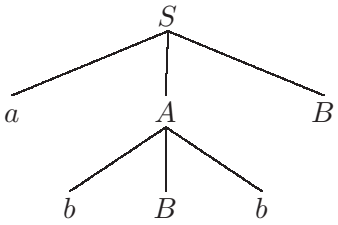
درخت اشتقاق جزیی همانند درخت اشتقاق است با این تفاوت که «هر برگ دارای برچسبی از $N \cup T \cup \{\epsilon\}$ است»

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

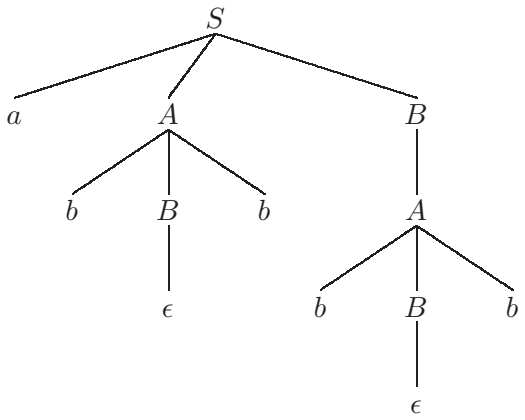
- اگر $G = (N, T, S, P)$ یک گرامر مستقل از متن باشد، در این صورت به ازای هر $w \in L(G)$ یک درخت اشتقاق برای G داریم که حاصل آن w است و برعکس حاصل درخت اشتقاق در $L(G)$ است.
- اگر t_G یک درخت اشتقاق جزیی برای G با ریشه‌ی S باشد، آنگاه حاصل t_G یک فرم جمله‌ای از G است.
- درخت اشتقاق نشان می‌دهد که چه 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.

مثال

گرامر G با قواعد $S \to aAB,\ A \to bBb,\ B \to A|\epsilon$ را در نظر بگیرید:
یک درخت اشتقاق جزیی برای فرم جمله‌ای $abBbB$ از G:



یک درخت اشتقاق برای جمله‌ی $abbbb$:



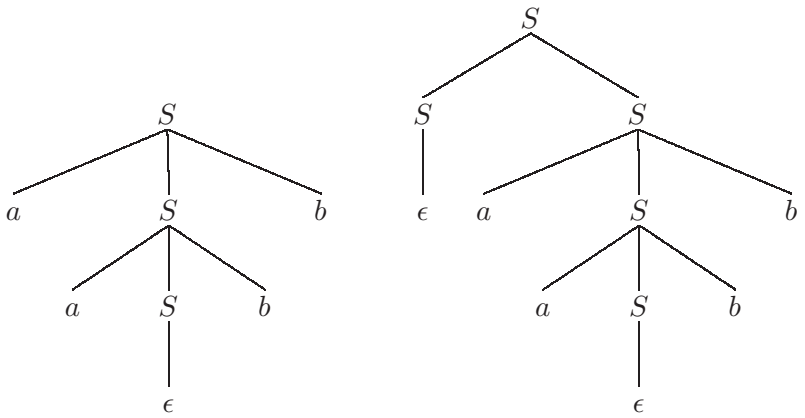
۴-۳ ابهام در گرامر

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.

ابهام به معنی وجود دو یا چند اشتقاق چپ‌ترین متفاوت یا راست‌ترین متفاوت برای G است.

مثال

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



۱-۳-۴ زبان ذاتاً مبهم

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

مثال

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

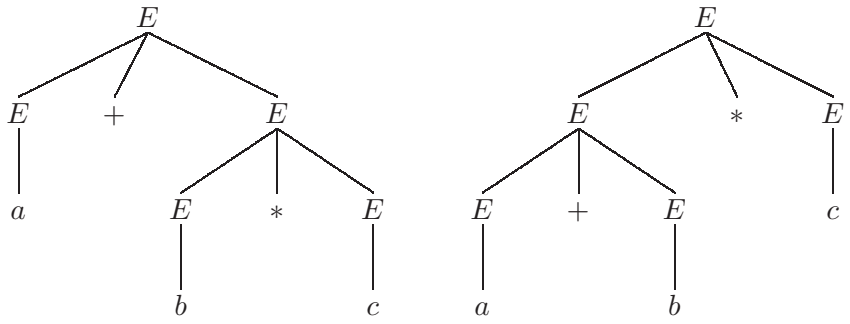
- زبان L را غیرمبهم می‌گوییم اگر و فقط اگر حداقل یک گرامر غیرمبهم برای آن موجود باشد.
- مساله‌ی تشخیص ابهام در یک گرامر مستقل از متن، تصمیم‌ناپذیر است، یعنی هیچ الگوریتمی برای تشخیص یا رفع ابهام وجود ندارد.

مثال

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

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

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



مثال

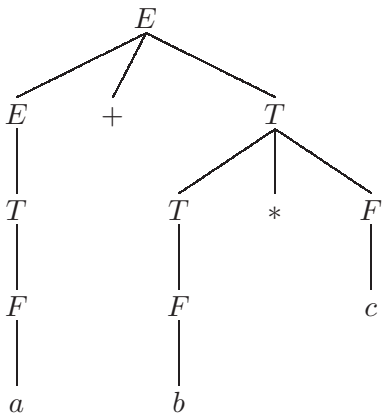
(رفع ابهام از گرامر) برای گرامر مبهم

$$E \rightarrow E + E \mid E * E \mid (E) \mid a \mid b \mid c$$

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

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid a \mid b \mid c \end{aligned}$$

در این گرامر رشته‌ی $a + b * c$ تنها یک درخت اشتقاق دارد:



۲-۳-۴ تقدم عملگرها برای رفع ابهام از گرامر عبارات ریاضی

از دو عملگر که در یک سطح پرانتزگذاری ظاهر شده‌اند، آنکه تقدم بالاتری دارد، باید در عمق بیشتری نسبت به دیگری در درخت تجزیه ظاهر شود.

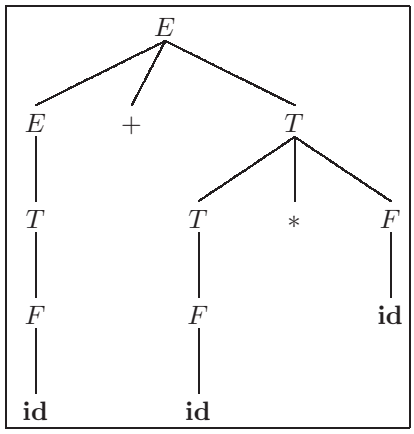
برای این منظور باید **ناپایانه‌های جدیدی** را وارد گرامر کنیم و به کمک آنها قواعد گرامر را طوری بازنویسی کنیم که قاعده‌ی فوق پیاده شود.

مثال

$$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$$

$$\Downarrow$$

$$\begin{array}{lcl} E & \rightarrow & E + T \mid T \\ T & \rightarrow & T * F \mid F \\ F & \rightarrow & (E) \mid \mathbf{id} \end{array}$$



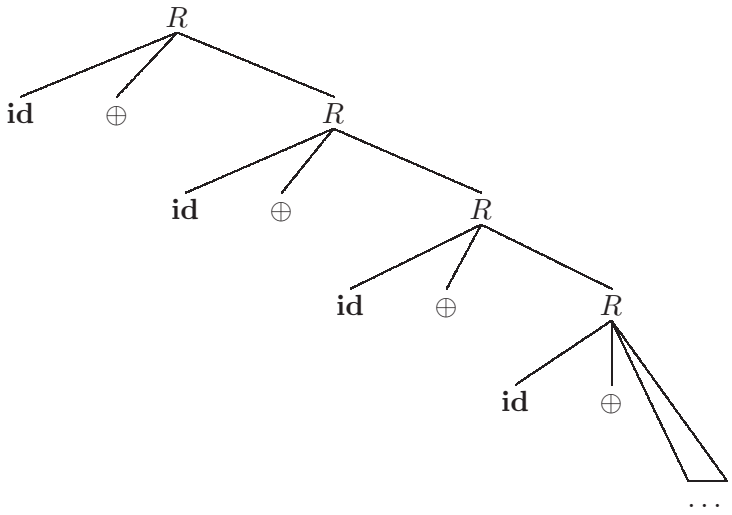
$$\mathbf{id} \ + \mathbf{id} \ * \mathbf{id}$$

۴-۳-۳ شرکت‌پذیری عملگرها: شرکت‌پذیری از راست

عملگرهایی که شرکت‌پذیر از راست (right-associative) هستند،
با یک قاعده‌ی بازگشتی از راست (right-recursive) تولید می‌شوند.

$$R \rightarrow \mathbf{id} \oplus R$$

درخت تجزیه برای قاعده‌ی بازگشتی از راست، از راست رشد می‌کند.



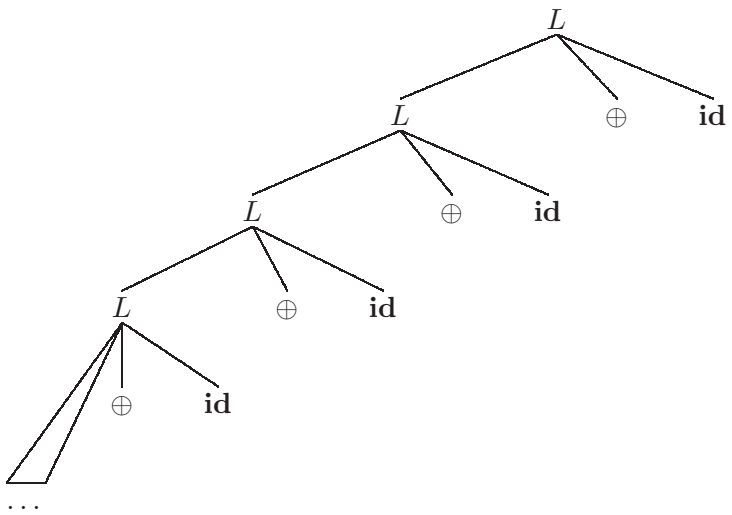
$$\mathbf{id} \oplus \mathbf{id} \oplus \mathbf{id} \oplus \mathbf{id} \oplus \dots$$

۴-۳-۴ شرکت‌پذیری عملگرها: شرکت‌پذیری از چپ

عملگرهایی که شرکت‌پذیر از چپ (left-associative) هستند،
با یک قاعده‌ی بازگشتی از چپ (left-recursive) تولید می‌شوند.

$$L \rightarrow L \oplus \mathbf{id}$$

درخت تجزیه برای قاعده‌ی بازگشتی از چپ، از چپ رشد می‌کند.



$$\dots \oplus \mathbf{id} \oplus \mathbf{id} \oplus \mathbf{id} \oplus \mathbf{id}$$

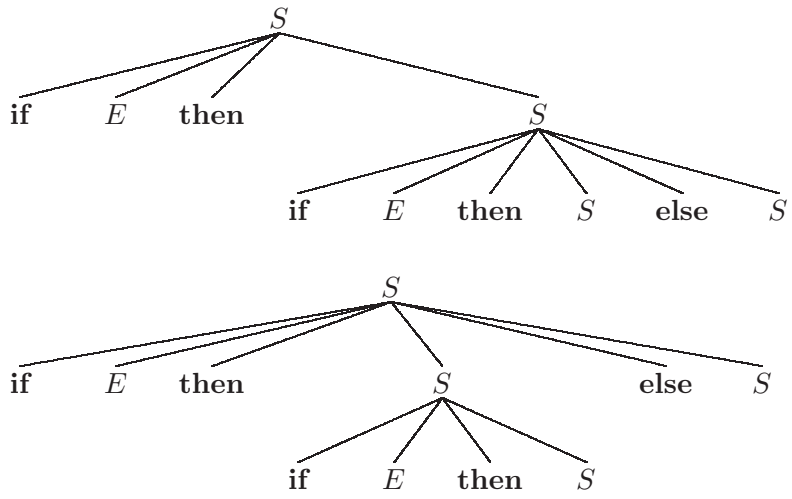
۴-۳-۵ گرامر دستورهای شرطی

$$
\begin{array}{rcl}
S & \rightarrow & \textbf{if } E \textbf{ then } S \\
 & | & \textbf{if } E \textbf{ then } S \textbf{ else } S \\
 & | & \textbf{other}
\end{array}
$$

این گرامر مبهم است، زیرا فرم جمله‌ای متنهی به رشته‌ی نهایی

$$\textbf{if } E \textbf{ then if } E \textbf{ then } S \textbf{ else } S$$

دارای دو درخت اشتقاق متفاوت است:

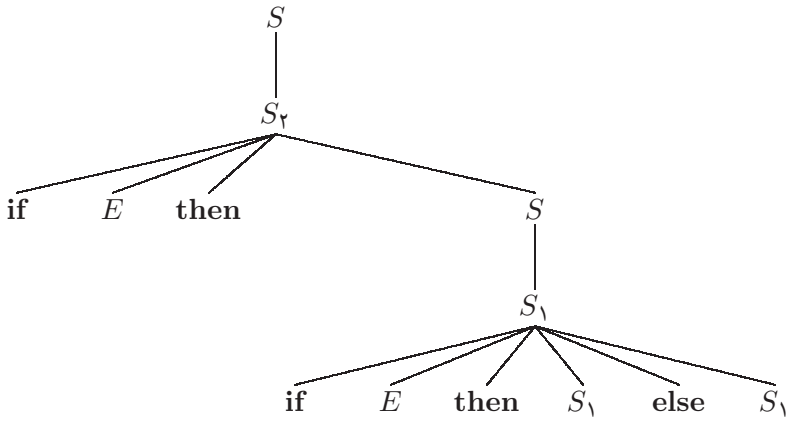


رفع ابهام از گرامر دستورهای شرطی برای این منظور، از قاعده‌ی معنایی زیر استفاده می‌کنیم که در بیشتر زبان‌های برنامه‌سازی در نظر گرفته می‌شود:

هر **else** باید با نزدیک‌ترین **then** تطابق‌نیافته، تطابق یابد.

$$
\begin{array}{rcl}
S & \rightarrow & S_1 \mid S_2 \\
S_1 & \rightarrow & \textbf{if } E \textbf{ then } S_1 \textbf{ else } S_1 \\
 & | & \textbf{other} \\
S_2 & \rightarrow & \textbf{if } E \textbf{ then } S \\
 & | & \textbf{if } E \textbf{ then } S_1 \textbf{ else } S_2
\end{array}
$$

$$\underline{\textbf{if } E \textbf{ then if } E \textbf{ then } S \textbf{ else } S}$$



۴-۴ تبدیل‌های گرامرها

۴-۴-۱ گرامرهای کاهش‌یافته، بدون دور و خالی از تهی

- گرامر **کاهش یافته** (Reduced) گرامری است که در آن:
  ۱) قواعدی به صورت $A \to A$ وجود ندارد،
  ۲) تمام ناپایانه‌ها مولد هستند $\forall A \in N \exists x \in T^* \; A \Rightarrow^* x$،
  ۳) همه‌ی ناپایانه‌ها دسترس‌پذیر هستند $\forall A \in N \exists \alpha, \beta \in (N \cup T)^* \; S \Rightarrow^* \alpha A \beta$.
- گرامر **بدون دور** (Cycle-free) گرامری است که در آن اشتقاقی مانند $A \Rightarrow^* A$ موجود نباشد.
- گرامر **خالی از تهی** (ϵ-free) گرامری است که قاعده‌ای مانند $A \to \epsilon$ ندارد.

ایجاد گرامرهای کاهش‌یافته

به ترتیب:

- حذف قواعد $A \to A$
- حذف قواعد شامل متغیرهای نامولد
- حذف قواعد شامل متغیرهای دسترس‌ناپذیر

مثال

متغیرهای نامولد

$$\begin{aligned} S &\to X \mid Y \\ X &\to () \\ Y &\to (YY) \end{aligned}$$

متغیر Y نامولد است و در نتیجه قواعد استفاده کننده از آن بی‌استفاده (useless) هستند.

هر متغیر بازگشتی که آلترناتیوی مختوم به یک رشته‌ی پایانی نداشته باشد، نامولد است.

مثال

متغیرهای دسترس‌ناپذیر

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow + \mid - \mid \epsilon \\ B &\rightarrow \textbf{digit} \mid B\ \textbf{digit} \\ C &\rightarrow .B \end{aligned}$$

متغیر C دسترس‌ناپذیر است و در نتیجه قاعده‌ی آخر بی‌استفاده (useless) است.

بجز S هر ناپایانه‌ای که در سمت راست هیچ قاعده‌ی تولیدی قرار نداشته باشد، دسترس‌ناپذیر است.

۲-۴-۴ ناپایانه (متغیر)، قاعده و گرامر بازگشتی

- قاعده‌ی $A \rightarrow \alpha A \beta$ که در آن متغیر سمت چپ در سمت راست قاعده نیز ظاهر شده است، یک **قاعده‌ی بازگشتی** *(recursive)* نام دارد.
- قاعده‌ی $A \rightarrow A\alpha$ 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 $A \rightarrow \alpha A$ 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.
- گرامری که در آن حداقل یک قاعده‌ی بازگشتی وجود داشته باشد، گرامر بازگشتی نام دارد.

حذف قواعد بازگشتی از چپ مستقیم

گاهی اوقات وجود قواعد بازگشتی از چپ در گرامر برای ما نامطلوب است. برای مثال در تجزیه‌گرهای بالا به پایین وجود این قواعد باعث بروز حلقه‌ی بی‌نهایت می‌شود. در چنین مواردی لازم است این قواعد با قواعد مناسبی که بازگشتی از چپ نیستند، جایگزین شوند. به این فرآیند، حذف بازگشتی از چپ گفته می‌شود.

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:

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

گرامر حاصل معادل با گرامر اولیه است.

در حالت ساده، قواعد $A \rightarrow A\alpha \mid \beta$ با قواعد $A \rightarrow \beta A'$ و $A' \rightarrow \alpha A' \mid \epsilon$ جایگزین می‌شود.

مثال

حذف قواعد بازگشتی از چپ مستقیم:

$$\begin{array}{lcl} E & \rightarrow & E + T \mid T \\ \hline T & \rightarrow & T * F \mid F \\ \hline F & \rightarrow & (E) \mid \mathbf{id} \end{array}$$

جایگزینی با:

$$\begin{array}{lcl} E & \rightarrow & TE' \\ E' & \rightarrow & +TE' \mid \epsilon \\ \hline T & \rightarrow & FT' \\ T' & \rightarrow & *FT' \mid \epsilon \\ \hline F & \rightarrow & (E) \mid \mathbf{id} \end{array}$$

- متغیر A یک متغیر بازگشتی غیر مستقیم است اگر وفقط اگر اشتقاق $A \Rightarrow^n \alpha A \beta$ در آن گرامر ممکن باشد $(n \geq 2)$.
- متغیر A یک متغیر بازگشتی از چپ غیر مستقیم است اگر وفقط اگر اشتقاق $A \Rightarrow^n A\alpha$ در آن گرامر ممکن باشد $(n \geq 2)$.
- متغیر A یک متغیر بازگشتی از راست غیر مستقیم است اگر وفقط اگر اشتقاق $A \Rightarrow^n \alpha A$ در آن گرامر ممکن باشد $(n \geq 2)$.

حذف بازگشتی از چپ غیر مستقیم

برای حذف بازگشتی از چپ غیر مستقیم، به صورت زیر عمل می‌کنیم:

- 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.

- **Input: grammar G without cycles and ϵ-productions.**
- **Output: An equivalent grammar without left recursion.**
- **Number the nonterminals in some order $A_1, A_2, \ldots, A_n$**
- **for $i = 1$ to n do**
  - **for $j = 1$ to $i - 1$ do**
    - ***replace*** $A_i \rightarrow A_j\gamma$
    - ***with*** $A_i \rightarrow \delta_1\gamma \mid \cdots \mid \delta_k\gamma$
    - ***where*** $A_j \rightarrow \delta_1 \mid \cdots \mid \delta_k$ ***are all the current*** A_j***-productions.***
  - **Eliminate immediate left-recursion for A_i**
    - ***New nonterminals generated above are numbered*** A_{i+n}

مثال

حذف بازگشتی از چپ غیر مستقیم:

- **Original Grammar:**
  - **(1)** $S \rightarrow Aa \mid b$
  - **(2)** $A \rightarrow Ac \mid Sd \mid e$
- **Ordering of nonterminals: $S \equiv A_1$ and $A \equiv A_2$.**
- $i = 1$
  - **do nothing as there is no immediate left-recursion for S**
- $i = 2$
  - **replace $A \rightarrow Sd$ by $A \rightarrow Aad \mid bd$**
  - **hence (2) becomes $A \rightarrow Ac \mid Aad \mid bd \mid e$**
  - **after removing immediate left-recursion:**
    - $A \rightarrow bdA' \mid eA'$
    - $A' \rightarrow cA' \mid adA' \mid \epsilon$

۳-۴-۴ پیشوند مشترک

در مجموعه قواعد

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_m$$

α پیشوند مشترک نام دارد.

فاکتورگیری از چپ برای حذف پیشوند مشترک

برای حذف پیشوند مشترک،

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_m$$

این مجموعه قواعد را با فاکتورگیری از چپ (left-factoring)، با مجموعه قواعد زیر جایگزین می‌کنیم:

$$\begin{aligned} A &\rightarrow \alpha B \\ B &\rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m \end{aligned}$$

گرامر حاصل معادل با گرامر اولیه است.

در حالت ساده، قواعد $A \to \alpha\beta_1 \mid \alpha\beta_2$ با قواعد $A \to \alpha B$ و $B \to \beta_1 \mid \beta_2$ جایگزین می‌شود.

الگوریتم رفع پیشوندهای مشترک با فاکتورگیری از چپ

برای رفع پیشوندهای مشترک، آن قدر روال فوق را تکرار می‌کنیم تا دیگر هیچ دو آلترناتیو یک متغیر دارای پیشوند مشترک نباشند.

- **Input: context free grammar G**
- **Output: equivalent left-factored context-free grammar G'**
- **for each nonterminal A do**
  - **find the longest non-ϵ prefix α that is common to right-hand sides of two or more productions**
  - **replace**
  - $A \to \alpha\beta_1 \mid \cdots \mid \alpha\beta_n \mid \gamma_1 \mid \cdots \mid \gamma_m$ **with**
    - $A \to \alpha A' \mid \gamma_1 \mid \cdots \mid \gamma_m$
    - $A' \to \beta_1 \mid \cdots \mid \beta_n$
  - **repeat the above process until A has no two productions with a common prefix.**

مثال

با فاکتورگیری از چپ از گرامر:

$$S \to (S) \mid ()$$

و جایگزینی با:

$$\begin{aligned} S &\to (B \\ B &\to S) \mid) \end{aligned}$$

در یک مرحله هیچ دو آلترناتیوی با پیشوند مشترک باقی نمی‌ماند.

مثال

فاکتورگیری از چپ و حذف بازگشتی از چپ

- **Original grammar:**
  $S \to (S) \mid SS \mid ()$
- **To remove immediate left-recursion, we have**
  - $S \to (S)S' \mid ()S'$
  - $S' \to SS' \mid \epsilon$
- **To do left-factoring, we have**
  - $S \to (S''$
  - $S'' \to S)S' \mid)S'$
  - $S' \to SS' \mid \epsilon$

۴-۵ تجزیه‌گر (Parser)

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

خروجی تجزیه‌گر، نوعی بازنمایی از درخت تجزیه (درخت اشتقاق) است.

به ازای هرگرامر مستقل از متن، الگوریتمی وجود دارد که هر رشته‌ی $w \in L(G)$ را در تعداد مراحلی که متناسب با $|w|^3$ است، تجزیه می‌کند.

مثال

نمونه‌ای ساده از تحلیل نحوی:

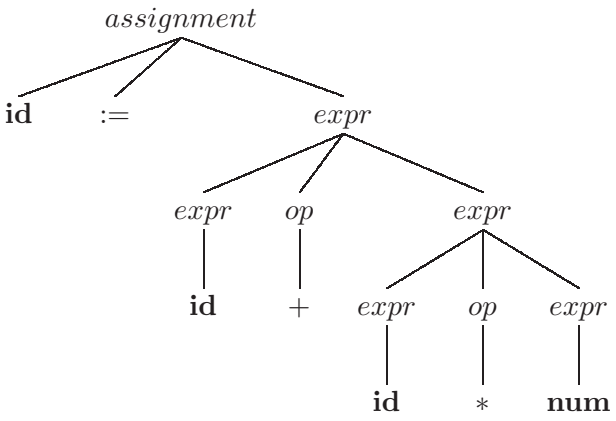
$$\begin{aligned} assignment &\rightarrow \mathbf{id} := expr \\ expr &\rightarrow \mathbf{id} \mid \mathbf{num} \mid expr\ op\ expr \mid (expr) \\ op &\rightarrow + \mid - \mid * \mid / \end{aligned}$$

دنباله‌ی توکن‌ها

$$\mathbf{id} := \mathbf{id} + \mathbf{id} * \mathbf{num}$$

$\downarrow$

سلسله‌مراتب ساختارها



۱-۵-۴ روش‌های تجزیه

- **روش‌های بالا به پایین (Top-Down Parsing)**

  درخت تجزیه از بالا به پایین (از ریشه به سمت برگ‌ها) ساخته می‌شود.

- **روش‌های پایین به بالا (Bottom-Up Parsing)**
  درخت تجزیه از پایین به بالا (از برگ‌ها به سمت ریشه) ساخته می‌شود.

طبقه‌بندی روش‌های تجزیه

روش‌های بالا به پایین	
ـ روش نزولی بازگشتی با عقبگرد	Recursive Descent with Backtracking
ـ روش نزولی بازگشتی	Recursive Descent
ـ روش‌های LL(k)	
روش‌های پایین به بالا	
ـ روش تقدم عملگر	Operator Precedence
ـ روش تقدم ساده	Simple Precedence
ـ روش LR ساده (SLR)	Simple LR
ـ روش LALR	Lookahead LR
ـ روش CLR	Canonical LR
ـ روش‌های عمومی	Earley, CYK, ...

نماد پیش‌بینی (Lookahead) نماد پیش‌بینی، نمادی است که باید برای تشخیص قاعده‌ی مناسب، از پیش آن را ببینیم. پرسش این است که به چند توکن بعد با شروع از توکن جاری باید نگاه کنیم؟

1	2	3	4	5	...
↑					

۶-۴ طبقه‌بندی چامسکی

	زبان	گرامر	قالب قواعد گرامر	پذیرنده
نوع صفر	زبان‌های شمارش‌پذیر بازگشتی Recursively Enumerable $\mathcal{L}_0$	**گرامرهای بدون محدودیت** **Unrestricted**	$\alpha \to \beta$ $\alpha \neq \epsilon$	ماشین‌های تورینگ Turing Machine (TM)
نوع یک	زبان‌های حساس به متن Context-Sensitive $\mathcal{L}_1$	**گرامرهای حساس به متن** **Context-Sensitive**	$\alpha \to \beta$ $\lvert\alpha\rvert \leq \lvert\beta\rvert$	آتوماتای کراندار خطی Linear Bounded Automata (LBA)
نوع دو	زبان‌های مستقل از متن Context-Free $\mathcal{L}_2$	**گرامرهای مستقل از متن** **Context-Free**	$\alpha \to \beta$ $\alpha \in N,\ \lvert\alpha\rvert = 1$	آتوماتای پشته‌ای Push-down Automata (PDA)
نوع سه	زبان‌های منظم Regular $\mathcal{L}_3$	**گرامرهای منظم** **Regular**	$\alpha \to \beta$ $\beta = aB$ یا $\beta = a$ $\beta = Ba$ یا $\beta = a$	آتوماتای متناهی Finite Automata (FA)

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$$

◄ تمرین

۱. گرامر مستقل از متن زیر را در نظر بگیرید:

$$S \rightarrow SS+ \mid SS* \mid a$$

آ) نشان دهید که رشته‌ی $aa+a*$ را می‌توان توسط این گرامر تولید نمود.

ب) درخت تجزیه (parse tree) را برای این رشته بسازید.

پ) این گرامر چه زبانی را تولید می‌کند؟

۲. چه زبانی توسط هر یک از گرامرهای زیر تولید می‌شود؟

آ) $S \rightarrow \circ S \text{۱} \mid \circ \text{۱}$

ب) $S \rightarrow +SS \mid -SS \mid a$

پ) $S \rightarrow S(S)S \mid \epsilon$

ت) $S \rightarrow aSbS \mid bSaS \mid \epsilon$

ث) $S \rightarrow a \mid S+S \mid SS \mid S* \mid (S)$

۳. کدام یک از گرامرهای تمرین فوق مبهم است؟ چرا؟

۴. برای هر یک از زبان‌های زیر، یک گرامر مستقل از متن بسازید:

آ) عبارت‌های ریاضی با عملگرهای $+$، $*$ و توان به شکل پیشوندی (prefix)

ب) عبارت‌های ریاضی با عملگرهای $+$، $*$ و توان به شکل پسوندی (postfix)

پ) لیست‌های «شرکت‌پذیر از چپ» (left-associative) از شناسه‌ها که با کاما جدا شده‌اند.

ت) لیست‌های «شرکت‌پذیر از راست» (right-associative) از شناسه‌ها که با کاما جدا شده‌اند.

ث) عبارت‌های ریاضی از اعداد و شناسه‌ها با چهار عملگر دوعملوندی $+ - * /$

ج) به عملگرهای ریاضی (۴ث) جمع و تفریق تک‌عملوندی (unary) را اضافه نمایید.

۵. گرامرهای تمرین فوق را به گونه‌ای بازنویسی کنید که مبهم نباشند.

۶. گرامرهای تمرین فوق را به گونه‌ای بازنویسی کنید که پیشوند مشترک یا بازگشتی از چپ نداشته باشند.

۷. یک گرامر مستقل از متن برای اعداد رومی بنویسید. برای توضیحات بیشتر در مورد ساختار اعداد رومی، به آدرس زیر مراجعه کنید:

`http://fa.wikipedia.org/wiki/عددنویسی_رومی`

۸. در ارتباط با گرامر زیر، پرسش‌های زیر را پاسخ دهید:

$$\begin{aligned} S &\rightarrow Aa \mid aBb \mid aBbDc \\ A &\rightarrow Sa \mid \epsilon \\ B &\rightarrow CD \\ C &\rightarrow Ba \mid \epsilon \\ D &\rightarrow Db \mid E \mid \epsilon \\ E &\rightarrow Ec \mid D \mid \epsilon \end{aligned}$$

آ) صورت کاهش‌یافته‌ی گرامر فوق را بنویسید.

ب) بازگشتی از چپ مستقیم و غیرمستقیم را از گرامر فوق رفع کنید و پس از فاکتورگیری از چپ، گرامر حاصل را بنویسید.

۹. بازگشتی از چپ را از گرامر زیر برطرف کنید.

$$\begin{aligned} S &\rightarrow Sab \mid SaS \mid X \\ X &\rightarrow Xc \mid a \mid b \end{aligned}$$

۱۰. گرامر زیر را در نظر بگیرید:

$$S \rightarrow S۰۰ \mid ۱۱S \mid ۰S۱ \mid ۱۰S \mid \epsilon$$

آ) نشان دهید که رشته‌ی ۱۱۱۰۰۰ به وسیله‌ی گرامر فوق قابل تولید است.

ب) چند درخت تجزیه‌ی (parse tree) متفاوت برای این رشته وجود دارد؟ همه‌ی آنها را بسازید.

پ) با اضافه کردن قواعد لازم، گرامر فوق را به گرامر مستقل از متنی تبدیل کنید که تمام رشته‌ها به طول زوج را تولید می‌کند.

۱۱. گرامر روبرو دو عملگر $ و # را برای رشته‌هایی متشکل از حروف الفبای انگلیسی تعریف می‌کند. عملگر # دو عملوند خود را به هم می‌چسباند (concatenation) و عملگر $ عملوند اول خود را معکوس (reverse) کرده و آن را به انتهای عملوند دوم خود می‌چسباند. مثال برای عملگر $:

$$abc\$def = defcba$$

عملگر $ نسبت به عملگر # اولویت بالاتری دارد. با توجه به گرامر زیر:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow T\$S \mid T\#S \mid T \\ T &\rightarrow string \end{aligned}$$

آ) حاصل عبارت $xyz\$hij\#cs$ چیست؟

ب) یک اشتقاق راست‌ترین، درخت تجزیه (parse tree) و درخت نحو (syntax tree) را برای رشته $a\#b\$c$ نشان دهید.

پ) با توجه به اینکه عملگر $ نسبت به عملگر # اولویت بالاتری دارد، گرامر معادلی را بیابید که عاری از ابهام باشد.

۱۲. برای هر یک از زبان‌های زیر یک گرامر طراحی کنید:

آ) مجموعه‌ی تمامی رشته‌های متشکل از ۰ و ۱ که هر ۰ بلافاصله با حداقل یک ۱ دنبال شود.

ب) مجموعه‌ی تمامی رشته‌های متشکل از ۰ و ۱ به طوری که تعداد صفرها و یک‌ها مساوی باشد.

پ) مجموعه‌ی تمامی رشته‌های متشکل از ۰ و ۱ به طوری که زیر رشته ۰۱۱ را شامل نشوند.

۱۳. برای هر یک از زبان‌های زیر یک گرامر غیر مبهم طراحی کنید:

آ) رشته‌های به شکل *۱*۰ به طوری که تعداد صفرها از تعداد یک‌ها بیشتر باشد.

ب) رشته‌های آینه‌ای (palindrome) روی الفبای ۰ و ۱ (به رشته‌ای آینه‌ای گفته می‌شود که خواندش از چپ به راست با خواندش از راست به چپ یکسان باشد).

پ) رشته‌های متشکل از پرانتزها و براکت‌های متوازن، مانند: ([[][()][()]])

۱۴. گرامر زیر را در نظر بگیرید که برای عملیات منطقی نوشته شده است .در رابطه با اولویت عملگرها نسبت به یکدیگر چه نتیجه‌ای می‌توان گرفت؟

$$
\begin{aligned}
X &\rightarrow X \textbf{ or } Y \mid Y \\
Y &\rightarrow Y \textbf{ and } Z \mid Z \\
Z &\rightarrow \textbf{not } Z \mid (X) \mid \textbf{true} \mid \textbf{false}
\end{aligned}
$$