اصول طراحی کامپایلر

درس ۶

تحلیل نحوی (۱)

Syntax Analysis (1)

کاظم فولادی قلعه
دانشکده مهندسی، پردیس فارابی
دانشگاه تهران

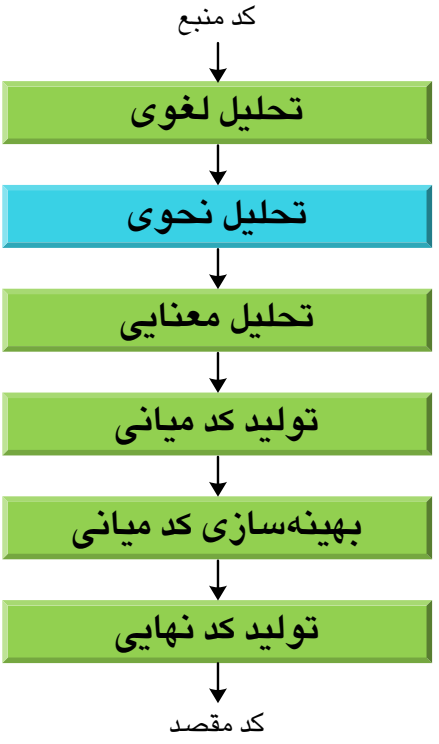
http://courses.fouladi.ir/compiler

تحلیل نحوی

۱

مقدمه

مراحل فرآیند کامپایل در یک نگاه



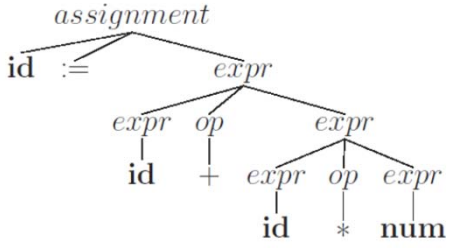
تحلیل نحوی


دنباله‌ی توکن‌ها
sequence of tokens
تحلیل‌گر نحوی
Syntax Analyzer
درخت ساختارها
tree of structures

تحلیل نحوی

مثال

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



گرامر

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

بازنمایی نحو زبان با گرامرهای مستقل از متن

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

گرامرهای مستقل از متن
برای بازنمایی نحو زبان‌های برنامه‌سازی مناسب هستند.

عبارت‌های منظم برای بازنمایی نحو محدودیت دارند
(مانند بازنمایی پرانتزگذاری صحیح).

اصول طراحی کامپایلر

تحلیل نحوی

۲

گرامرهای مستقل از متن و بازنمایی نحو

گرامر

GRAMMAR

گرامر: ابزاری برای تولید رشته‌های یک زبان

Non-terminals Terminals Start symbol **Production** rules

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

مجموعه‌ی ناپایانه‌ها (متغیرها)
(مجموعه‌ی متناهی)

$N \cap T = \varnothing$

مجموعه‌ی پایانه‌ها (الفبای گرامر / توکن‌ها)

$S \in N$ نماد شروع (یک ناپایانه‌ی خاص)

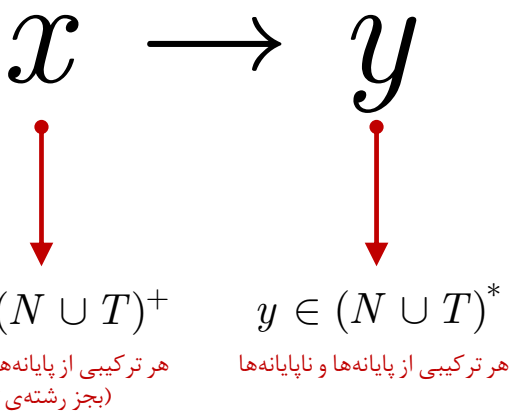
قواعد تولید (مجموعه‌ی متناهی)

قواعد تولید در گرامر

PRODUCTION RULES

قواعد تولید: اساس تعریف یک گرامر

قواعد تولید: به صورت یک زوج مرتب



از قواعد تولید در عمل اشتقاق (derivation) استفاده می‌شود.

گرامر

مثال

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

$$T = \{0, 1\}, \qquad N = \{S\}, \qquad P = \{S \to 0S1, S \to 01\}$$

$$S \Rightarrow 01$$

$$S \Rightarrow 0S1 \Rightarrow 0011$$

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$$

$$\dots$$

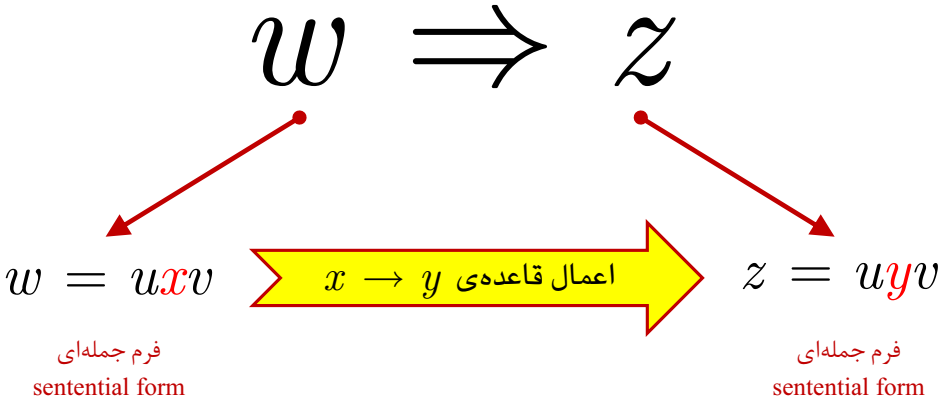
$$S \Rightarrow^* 0^n1^n$$

$$L(G) = \{0^n1^n : n \geq 1\}$$

اشتقاق

DERIVATION

اشتقاق: اعمال یک قاعده‌ی تولید بر روی یک رشته



w، z را مشتق می‌کند.
z از w مشتق می‌شود.

اشتقاق

اشتقاق در چند مرحله

می‌توان قواعد تولید را پی‌درپی و به تعداد دلخواه بر روی رشته‌های متوالی اعمال کرد تا رشته‌ی مورد نظر به دست آید.

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$$

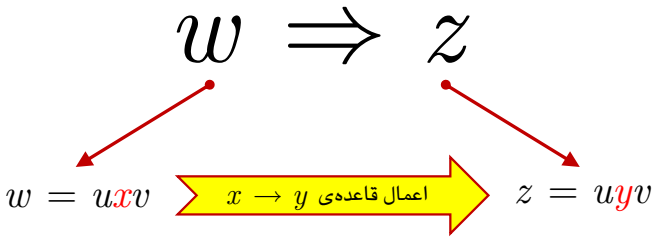
$$w_1 \Rightarrow^* w_n$$

w_n طی صفر یا چند مرحله از w_1 تولید می‌شود.

اشتقاق

قاعده‌ی اشتقاق

هرگاه سمت چپ یک قاعده با یک زیررشته از یک رشته تطابق پیدا کند،
می‌توانیم سمت راست آن قاعده را با آن زیررشته جایگزین کنیم.



$$w \Rightarrow z \quad \text{iff} \quad w = w_1 u w_2,\ z = w_1 v w_2,\ w_1,\ w_2 \in (N \cup T)^*,\ u \rightarrow v \in P$$

اشتقاق

نمادگذاری‌ها

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

زبان تولید شده توسط یک گرامر

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

زبان تولید شده توسط گرامر G

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

زبان تولید شده توسط یک گرامر

مثال ۱

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

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

$$\begin{aligned} S &\rightarrow AB && (1)\\ A &\rightarrow aA && (2)\\ A &\rightarrow \epsilon && (3)\\ B &\rightarrow bB && (4)\\ B &\rightarrow \epsilon && (5) \end{aligned}$$

$$S \overset{(1)}{\Rightarrow} AB \overset{(2)}{\Rightarrow} aAB \overset{(2)}{\Rightarrow} aaAB \overset{(2)}{\Rightarrow} aaaAB \overset{(3)}{\Rightarrow} aaaB \overset{(4)}{\Rightarrow} aaaaB \overset{(4)}{\Rightarrow} aaabbB \overset{(5)}{\Rightarrow} aaabb$$

$$L(G) = \{a^m b^n : m, n \geq 0\}$$

زبان تولید شده توسط یک گرامر

مثال ۲

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

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

$$\begin{aligned} S &\rightarrow QNQ && (1)\\ QN &\rightarrow QR && (2)\\ RN &\rightarrow NNR && (3)\\ RQ &\rightarrow NNQ && (4)\\ N &\rightarrow a && (5)\\ Q &\rightarrow \epsilon && (6) \end{aligned}$$

$$S \overset{(1)}{\Rightarrow} QNQ \overset{(2)}{\Rightarrow} QRQ \overset{(4)}{\Rightarrow} QNNQ \overset{(2)}{\Rightarrow} QRNQ \overset{(3)}{\Rightarrow} QNNRQ \overset{(4)}{\Rightarrow} QNNNNQ \Rightarrow^{*} aaaa$$

$$L(G) = \{a^{(2^n)} : n \geq 0\}$$

طبقه‌بندی چامسکی

سلسله‌مراتب زبان‌ها

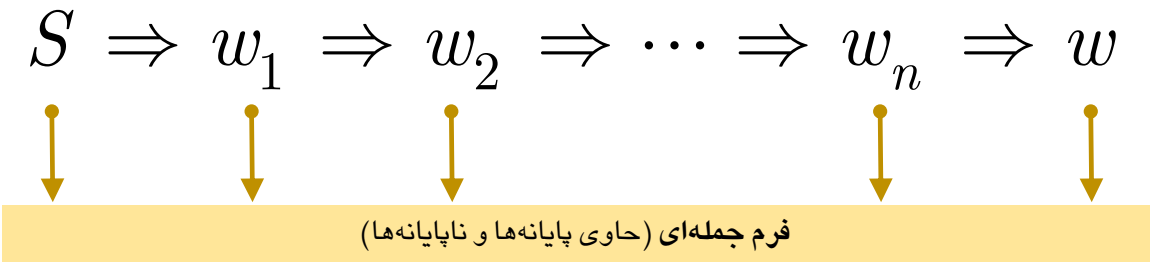
	زبان	گرامر	قالب قواعد گرامر	پذیرنده
نوع صفر	زبان‌های شمارش‌پذیر بازگشتی Recursively Enumerable $\mathcal{L}_0$	**گرامرهای بدون محدودیت** **Unrestricted**	$\alpha \rightarrow \beta$ $\alpha \neq \epsilon$	ماشین‌های تورینگ Turing Machine (TM)
نوع یک	زبان‌های حساس به متن Context-Sensitive $\mathcal{L}_1$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

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

فرم‌های جمله‌ای یک گرامر

SENTENTIAL FORMS

در یک اشتقاق از جمله‌ی w که $w \in L$:



مجموعه‌ی همه‌ی فرم‌های جمله‌ای قابل اشتقاق از گرامر G :

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

بدیهی است که:

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

هم‌ارزی گرامرها

دو گرامر G_1 و G_2 معادل هستند
اگر و فقط اگر
هر دو یک زبان واحد را تولید کنند.

$$G_1 \equiv G_2 \quad \textbf{iff} \quad L(G_1) = L(G_2)$$

گرامر مستقل از متن

CONTEXT-FREE GRAMMAR

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

$A \longrightarrow x$ همه‌ی قواعد تولید به صورت

$A \in N$
یک ناپایانه

$x \in (N \cup T)^*$
هر ترکیبی از پایانه‌ها و ناپایانه‌ها

سمت چپ هر قاعده **فقط یک ناپایانه** وجود دارد.

قراردادهای نمادگذاری برای گرامرها

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 BNF

Backus-Naur Form

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          |  /
```

اشتقاق‌های قانون‌مند

اشتقاق‌های قانون‌مند *Canonic Derivations*	
اشتقاق راست‌ترین *Right-most Derivation*	**اشتقاق چپ‌ترین** *Left-most Derivation*
جایگزینی **سمت راست‌ترین متغیر** در فرم جمله‌ای در **همه‌ی** گام‌های اشتقاق	جایگزینی **سمت چپ‌ترین متغیر** در فرم جمله‌ای در **همه‌ی** گام‌های اشتقاق
$\Rightarrow_{RM}$	$\Rightarrow_{LM}$

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

اشتقاق‌های قانون‌مند

مثال

گرامر :

$$S \rightarrow aAB$$
$$A \rightarrow bBb$$
$$B \rightarrow A \mid \epsilon$$

جمله (رشته): $abbbb$

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

اشتقاق چپ‌ترین
Left-most Derivation

اشتقاق راست‌ترین
Right-most Derivation

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

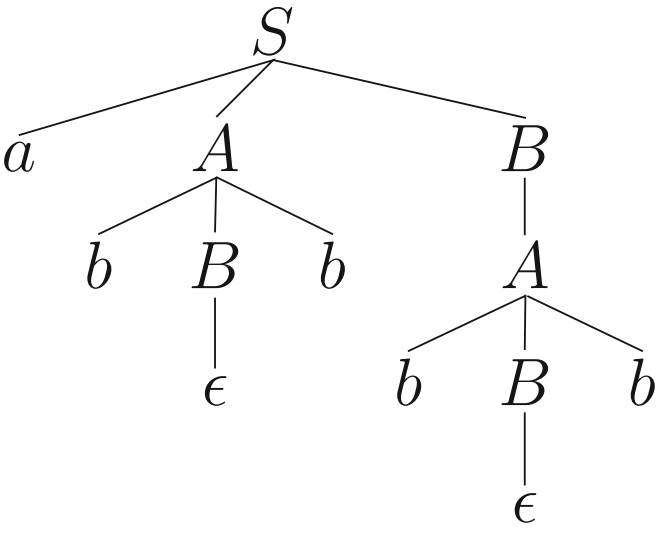
درخت اشتقاق

(درخت تجزیه)

DERIVATION TREE

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

$$S \rightarrow aAB$$
$$A \rightarrow bBb$$
$$B \rightarrow A \mid \epsilon$$



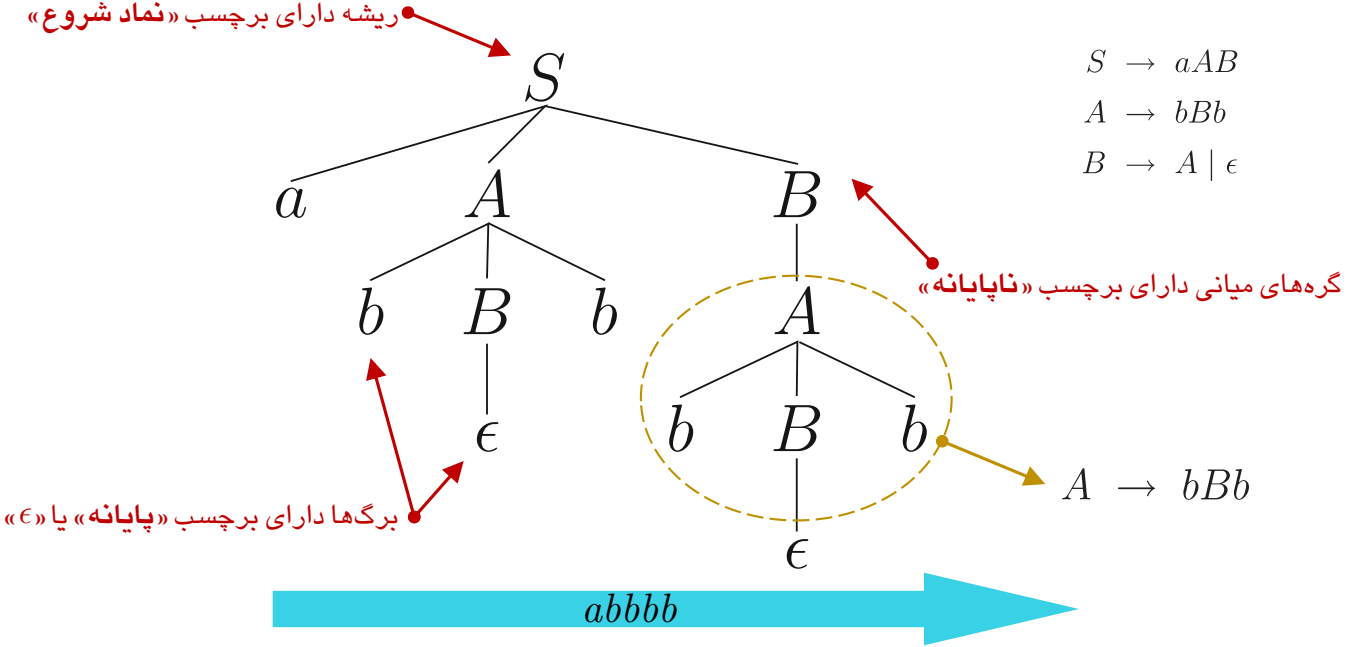
درخت اشتقاق

(درخت تجزیه)

DERIVATION TREE

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

$$S \rightarrow aAB$$
$$A \rightarrow bBb$$
$$B \rightarrow A \mid \epsilon$$



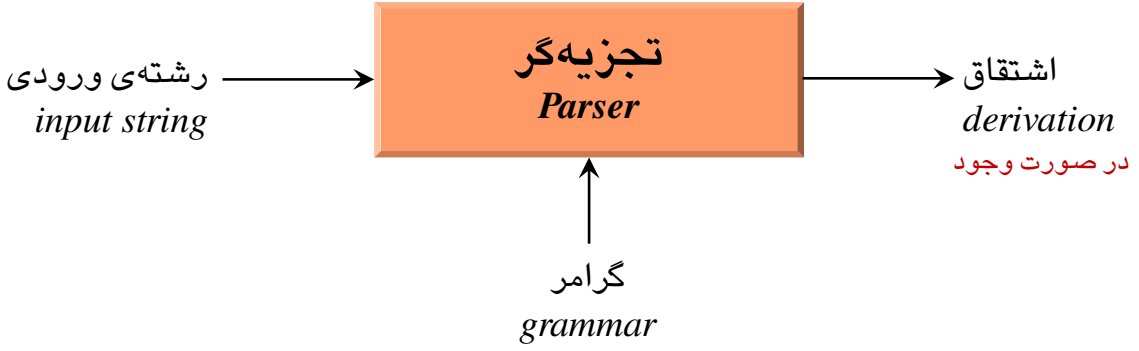
حاصل درخت اشتقاق با نوشتن برگ‌ها از سمت چپ به راست یک جمله از گرامر است.

- درخت اشتقاق، فقط قواعد استفاده شده برای تولید جمله را نشان می‌دهد، اما ترتیب استفاده از آن قواعد را نشان نمی‌دهد.
- ترتیب به‌کارگیری قواعد در هر گام، در نتیجه‌ی نهایی تأثیری ندارد.

تجزیه‌گر (پارسر)

PARSER

تجزیه‌گر: الگوریتمی است که
برای رشته‌ی دلخواه w
یک اشتقاق می‌یابد
یا می‌گوید اشتقاق ممکن نیست.



برای هر گرامر مستقل از متن الگوریتمی وجود دارد که
هر رشته‌ی دلخواه w را در تعداد مراحل متناسب با $|w|^3$ تجزیه می‌کند.

ابهام در گرامر

AMBIGUITY

گرامر مستقل از متن G **مبهم** نام دارد
اگر و فقط اگر
حداقل یک جمله $w \in L(G)$ وجود داشته باشد که
حداقل **دو درخت اشتقاق متفاوت** داشته باشد.

حداقل دو اشتقاق راست‌ترین متفاوت داشته باشد.

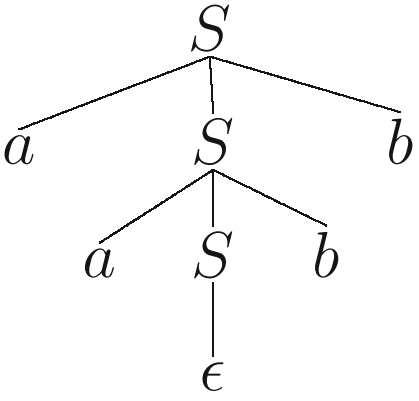
حداقل دو اشتقاق چپ‌ترین متقاوت داشته باشد.

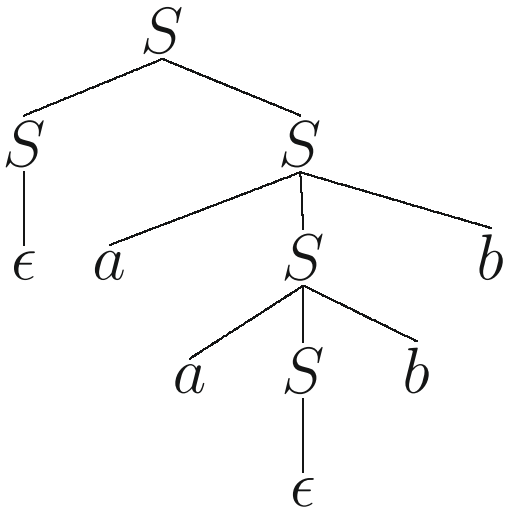
ابهام در گرامر

مثال

$$S \rightarrow aSb \mid SS \mid \epsilon$$

$$aabb$$





ابهام در گرامر

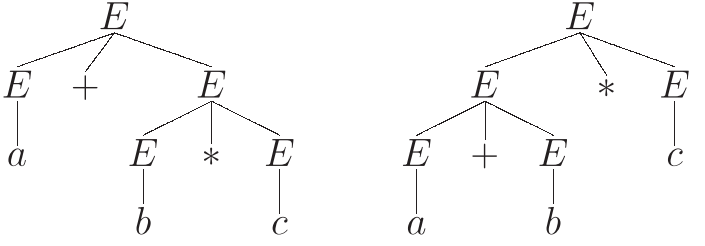
مثال

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

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

برای عبارت‌های حسابی، مبهم است:

برای رشته‌ی $a + b * c$ دو درخت اشتقاق وجود دارد:



رفع ابهام از گرامر

مثال

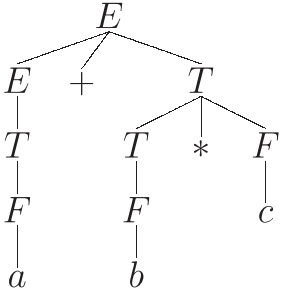
گرامر مبهم:

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

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



رفع ابهام از گرامر

استفاده از تقدم عملگرها برای رفع ابهام از گرامر عبارت‌های ریاضی

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

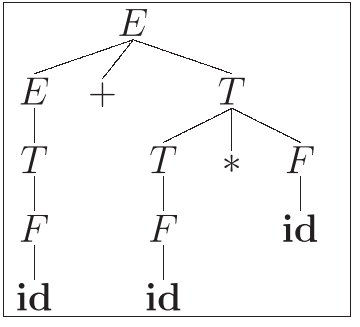
برای پیاده‌سازی قاعده‌ی فوق، **ناپایانه‌های جدیدی** را وارد گرامر می‌کنیم و به کمک آنها قواعد گرامر را بازنویسی می‌کنیم.

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

$$\Downarrow$$

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

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



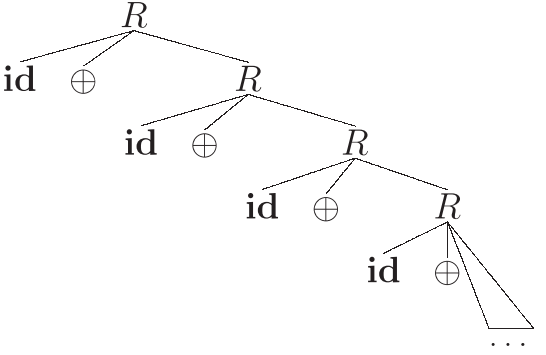
رفع ابهام از گرامر

شرکت‌پذیری عملگرها: شرکت‌پذیری از راست

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

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

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



$$\mathbf{id} \oplus \mathbf{id} \oplus \mathbf{id} \oplus \mathbf{id} \oplus \ldots$$

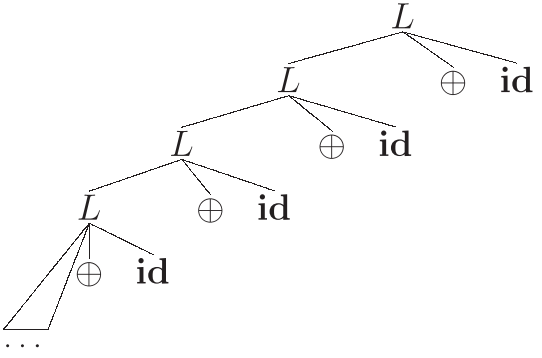
رفع ابهام از گرامر

شرکت‌پذیری عملگرها: شرکت‌پذیری از راست

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

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

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



$$\ldots \oplus \mathrm{id} \oplus \mathrm{id} \oplus \mathrm{id} \oplus \mathrm{id}$$

رفع ابهام از گرامر

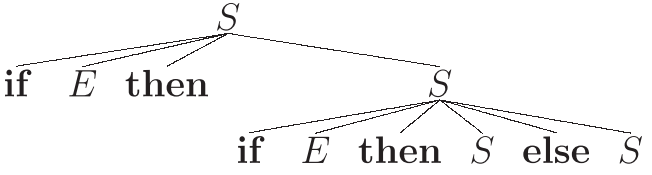
ابهام در گرامر دستور شرطی

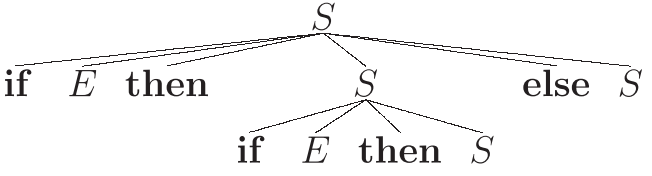
$$
\begin{array}{rcl}
S & \rightarrow & \textbf{if } E \textbf{ then } S \\
 & | & \textbf{if } E \textbf{ then } S \textbf{ else } S \\
 & | & \textbf{other}
\end{array}
$$

این گرامر مبهم است: زیرا فرم جمله‌ای منتهی به رشته‌ی نهایی

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

دارای دو درخت تجزیه‌ی متفاوت است:





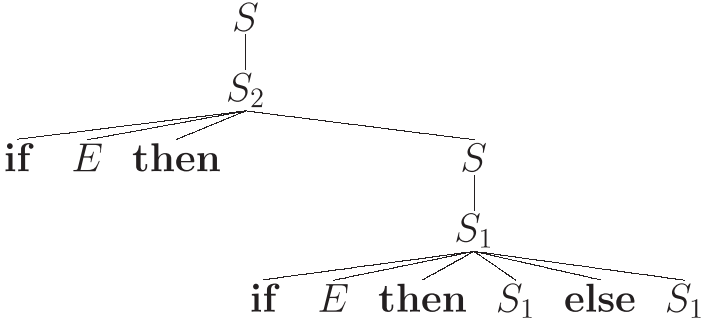
رفع ابهام از گرامر

رفع ابهام از گرامر دستور شرطی

هر else باید با نزدیک‌ترین then تطابق‌نیافته، تطابق پیدا کند.

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

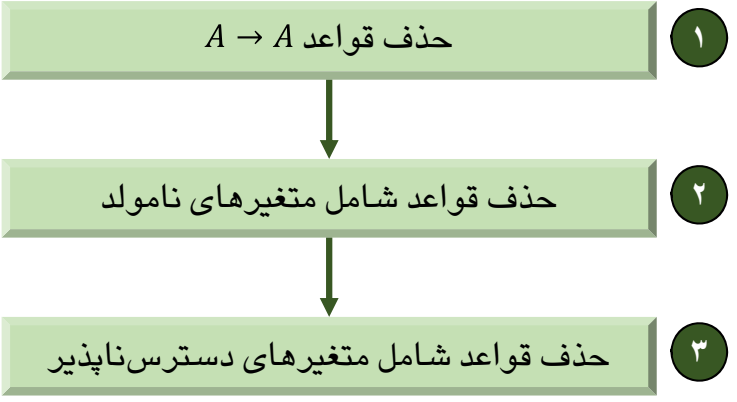
$\underline{\textbf{if } E \textbf{ then if } E \textbf{ then } S \textbf{ else } S}$



گرامرهای کاهش‌یافته، بدون دور، خالی از تهی

گرامر کاهش‌یافته *Reduced Grammar*	• عدم وجود قواعدی به صورت $A \rightarrow A$ • مولد بودن تمام ناپایانه‌ها • دسترس‌پذیر بودن تمام ناپایانه‌ها
گرامر بدون دور *Cycle-free Grammar*	عدم وجود اشتقاقی مانند $A \Rightarrow^* A$
گرامر خالی از تهی *ϵ - free Grammar*	عدم وجود قاعده‌ای مانند $A \rightarrow \epsilon$

تبدیل گرامر به گرامر کاهش‌یافته



ناپایانه‌ها/متغیرهای نامولد

مثال

NON-GENERATIVE NON-TERMINALS / VARIABLES

$$S \rightarrow X \mid Y$$
$$X \rightarrow ()$$
$$Y \rightarrow (YY)$$

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

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

ناپایانه‌ها/متغیرهای دسترس‌ناپذیر

مثال

NON-REACHABLE NON-TERMINALS / VARIABLES

$$S \rightarrow AB$$
$$A \rightarrow + \mid - \mid \epsilon$$
$$B \rightarrow \textbf{digit} \mid B\,\textbf{digit}$$
$$C \rightarrow .B$$

متغیر C دسترس‌ناپذیر است،
در نتیجه قواعد استفاده کننده از آن بی‌استفاده (useless) هستند.

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

قاعده، متغیر و گرامر بازگشتی

قاعده‌ی بازگشتی
Recursive Production-rule

متغیر سمت چپ، در سمت راست قاعده نیز ظاهر می‌شود.

$$A \rightarrow \alpha A \beta$$

قاعده‌ی بازگشتی از راست
Right-recursive Production-rule

متغیر سمت چپ، در انتهای راست سمت راست قاعده نیز ظاهر می‌شود.

$$A \rightarrow \alpha A$$

قاعده‌ی بازگشتی از چپ
Left-recursive Production-rule

متغیر سمت چپ، در ابتدای چپ سمت راست قاعده نیز ظاهر می‌شود.

$$A \rightarrow A \alpha$$

متغیر/ناپایانه بازگشتی
Recursive Variable/Non-terminal

متغیری که برای آن حداقل یک قاعده‌ی بازگشتی وجود داشته باشد.

گرامر بازگشتی
Recursive Gramamr

گرامری که در آن حداقل یک قاعده‌ی بازگشتی وجود داشته باشد.

رفع بازگشتی از چپ (مستقیم)

یافتن گرامر معادل فاقد قواعد بازگشتی از چپ

قواعد متغیر بازگشتی A را به دو قسمت تقسیم می‌کنیم:

قواعد بازگشتی از چپ A $\quad A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n$

قواعد غیر بازگشتی از چپ A $\quad A \rightarrow \beta_1 \mid \beta_2 \mid \ldots \beta_m$

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

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \beta_m A'$$
$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \epsilon$$

حاصل: گرامری معادل با گرامر اولیه بدون بازگشتی از چپ برای متغیر A

$A \rightarrow A\alpha \mid \beta$ ⟹ رفع بازگشتی از چپ ⟹ $A \rightarrow \beta A'$, $A' \rightarrow \alpha A' \mid \epsilon$

رفع بازگشتی از چپ (مستقیم)

مثال

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

رفع بازگشتی از چپ

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

بازگشتی غیرمستقیم

متغیر بازگشتی غیرمستقیم
Indirect Recursive Variable

متغیر A یک متغیر بازگشتی غیرمستقیم است اگر و فقط اگر اشتقاق زیر در گرامر ممکن باشد:

$$A \Rightarrow^{n} \alpha A \beta \quad n > 1$$

متغیر بازگشتی از راست غیرمستقیم
Indirect Right-recursive Variavle

در صورت وجود اشتقاق

$$A \Rightarrow^{n} \alpha A \quad n > 1$$

متغیر بازگشتی از چپ غیرمستقیم
Indirect Left-recursive Variavle

در صورت وجود اشتقاق

$$A \Rightarrow^{n} A \alpha \quad n > 1$$

رفع بازگشتی از چپ (مستقیم و غیر مستقیم)

یافتن گرامر معادل فاقد قواعد بازگشتی از چپ

۱. در نظر گرفتن یک ترتیب بر روی متغیرهای گرامر

۲. رفع بازگشتی از چپ مستقیم برای متغیر اول

۳. برای متغیرهای بعدی

جایگزینی قواعد متغیرهای قبلی و رفع بازگشتی از چپ مستقیم برای متغیر فعلی

- **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}

ورودی: گرامر بدون دور و خالی از تهی

رفع بازگشتی از چپ (مستقیم و غیر مستقیم)

مثال

- **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:**
    - $A \rightarrow bdA' \mid eA'$
    - $A' \rightarrow cA' \mid adA' \mid \epsilon$

پیشوند مشترک

پیشوند مشترک
Common Prefix

در مجموعه قواعد زیر، α پیشوند مشترک نام دارد:

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

رفع پیشوند مشترک

با فاکتورگیری از چپ

برای رفع پیشوند مشترک

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

با **فاکتورگیری از چپ**، این مجموعه قواعد را با مجموعه قواعد زیر جایگزین می‌کنیم:

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

حاصل: گرامری معادل با گرامر اولیه بدون پیشوند مشترک برای متغیر A

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

فاکتورگیری از چپ

$A \rightarrow \alpha B$
$B \rightarrow \beta_1 \mid \beta_2$

الگوریتم فاکتورگیری از چپ

برای رفع پیشوند مشترک

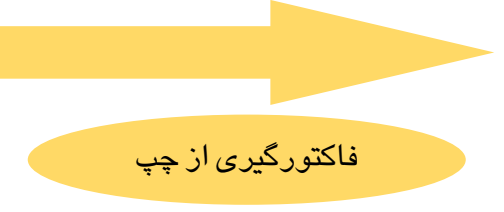
LEFT-FACTORING ALGORITHM

- **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
  - $A \longrightarrow \alpha\beta_1 \mid \cdots \mid \alpha\beta_n \mid \gamma_1 \mid \cdots \mid \gamma_m$ with
    - $A \longrightarrow \alpha A' \mid \gamma_1 \mid \cdots \mid \gamma_m$
    - $A' \longrightarrow \beta_1 \mid \cdots \mid \beta_n$
  - repeat the above process until A has no two productions with a common prefix.

رفع پیشوند مشترک

با فاکتورگیری از چپ: مثال

$$S \rightarrow (S) \mid ()$$



$$S \rightarrow (B$$
$$B \rightarrow S) \mid)$$

رفع بازگشتی از چپ و پیشوند مشترک

مثال

$$S \rightarrow (S) \mid SS \mid ()$$

رفع بازگشتی از چپ مستقیم

$$S \rightarrow (S)S' \mid ()S'$$
$$S' \rightarrow SS') \mid \epsilon$$

فاکتورگیری از چپ

$$S \rightarrow (S''$$
$$S'' \rightarrow S)S' \mid)S'$$
$$S' \rightarrow SS') \mid \epsilon$$

اصول طراحی کامپایلر

تحلیل نحوی

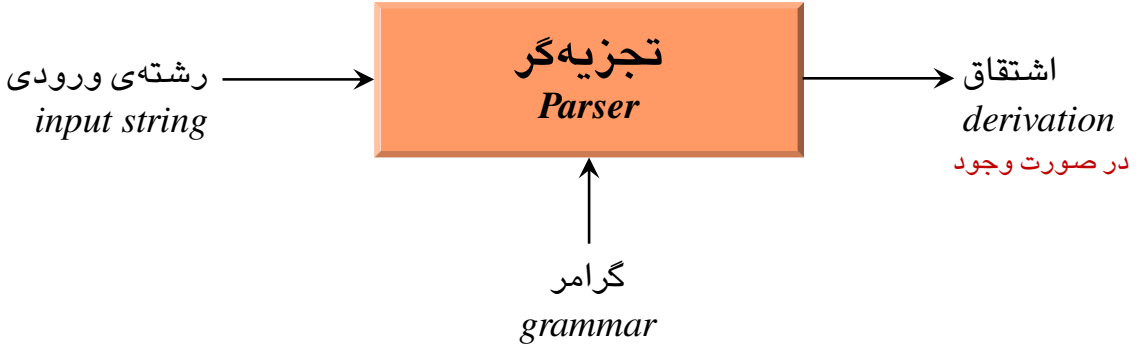
۳

روش‌های تجزیه (Parsing)

تجزیه‌گر (پارسر)

PARSER

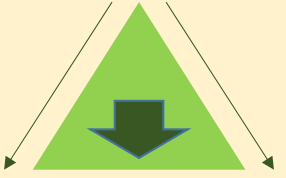
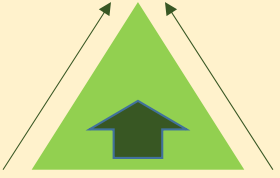
تجزیه‌گر: الگوریتمی است که
برای رشته‌ی دلخواه w
یک اشتقاق می‌یابد
یا می‌گوید اشتقاق ممکن نیست.



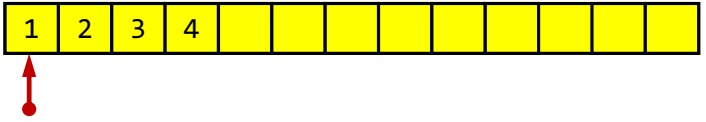
خروجی تجزیه‌گر، نوعی بازنمایی از درخت تجزیه (درخت اشتقاق) است:
مشخص می‌کند که رشته‌ی مربوطه با به‌کارگیری چه قواعدی از گرامر تولید شده است.

روش‌های تجزیه

روش‌های تجزیه
Parsing Methods

روش‌های بالا به پایین *Top-Down Parsing Methods*	روش‌های پایین به بالا *Bottom-Up Parsing Methods*
ساخت درخت تجزیه از بالا به پایین (از ریشه به سمت برگ‌ها)	ساخت درخت تجزیه از پایین به بالا (از برگ‌ها به سمت ریشه)
	
روش نزولی بازگشتی با عقب‌گرد — Recursive Descent with BT	روش‌های تقدم — Precedence
روش نزولی بازگشتی — Recursive Descent	روش LR ساده — Simple LR (SLR)
روش‌های LL(k) — LL(k)	روش LALR — Lookahead LR (LALR)
	روش CLR — Canonical LR (CLR)
	روش‌های عمومی — Early, CYK, …

Lookahead



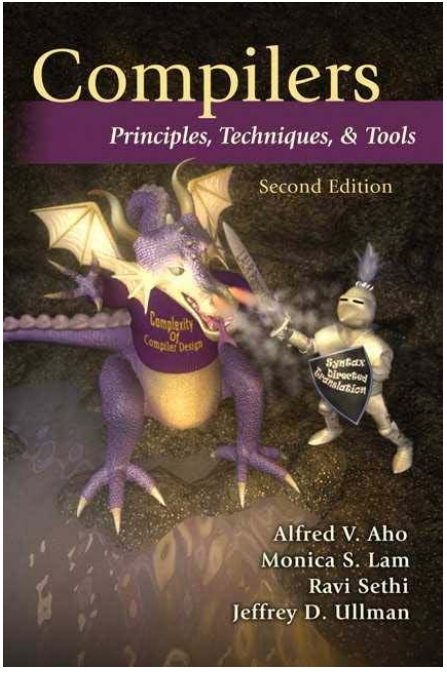
برای تشخیص قاعده‌ی مناسب، باید به چند توکن با شروع از توکن جاری نگاه کنیم؟

تحلیل نحوی

۴

منابع

منبع اصلی



A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman,
Compilers: Principles, Techniques and Tools,
Second Edition, Addison-Wesley, 2007.

Chapter 2 (2.2), Chapter 4 (4.2, 4.3)