دانشگاه صنعتی اصفهان
دانشکده مهندسی برق و کامپیوتر

عنوان: تکلیف تئوری دوم درس کامپایلر

نام و نام خانوادگی: علیرضا ابره فروش
شماره دانشجویی: ۹۸۱۶۶۰۳

نیم‌سال تحصیلی: بهار ۱۴۰۱/۱۴۰۲

مدرّس: دکتر حسین فلسفین

۱

۱.۱ a

طبق الگوریتم زیر عمل می‌کنیم.

Algorithm 4.21 : Left factoring a grammar.

INPUT: Grammar G.

OUTPUT: An equivalent left-factored grammar.

METHOD: For each nonterminal A, find the longest prefix α common to two or more of its alternatives. If $\alpha \neq \epsilon$ — i.e., there is a nontrivial common prefix — replace all of the A-productions $A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \cdots \mid \alpha\beta_n \mid \gamma$, where γ represents all alternatives that do not begin with α, by

$$\begin{aligned} A &\to \alpha A' \mid \gamma \\ A' &\to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n \end{aligned}$$

Here A' is a new nonterminal. Repeatedly apply this transformation until no two alternatives for a nonterminal have a common prefix. □

شکل ۱

$$S \longrightarrow SS'$$

$$S' \longrightarrow +S| + P$$

$$P \longrightarrow PP'$$

$$P' \longrightarrow *P| * I$$

$$I \longrightarrow -I|(S)|D$$

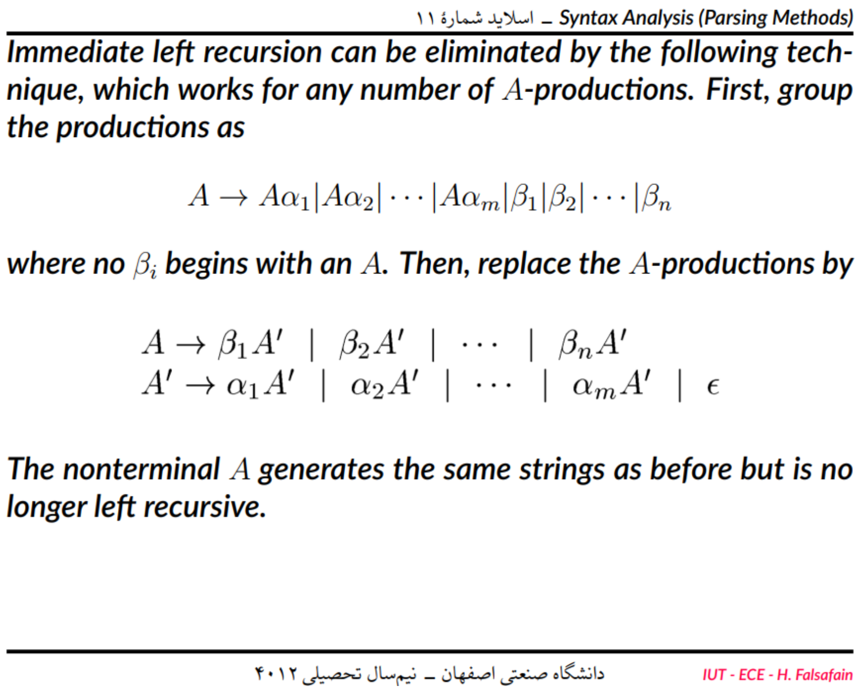
$$D \longrightarrow 0|1N$$

$$N \longrightarrow NN'|0|1|\varepsilon$$

$$N' \longrightarrow N$$

۲.۱ b

طبق الگوریتم زیر عمل می‌کنیم.



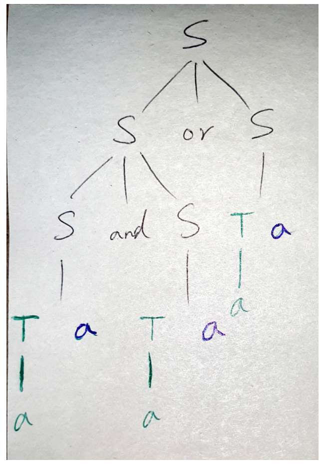
شکل ۲

$$
\begin{aligned}
S &\longrightarrow US' \\
S' &\longrightarrow aSS'|\varepsilon \\
U &\longrightarrow TU' \\
U' &\longrightarrow uUU'|\varepsilon \\
T &\longrightarrow tT'|fT'|(S)T' \\
T' &\longrightarrow |nT'|\varepsilon
\end{aligned}
$$

۲

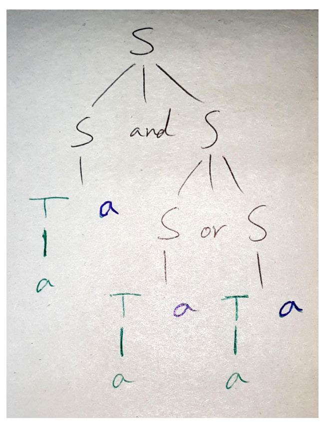
بسته به اینکه ابتدا کدام production نظیر S را پارس کنیم درخت پارس می‌تواند متفاوت باشد. پس دو درخت پارس نظیر اینکه اول and را تجزیه کنیم یا or مشابه تصاویر پایین موجود است. سپس به ازای هر یک از ۳ S ظاهر شده دو تا راه برای رسیدن به a می‌باشد. پس در کل تعداد حالات برابر است با $2 \times 2^3 = 16$ خواهد بود.

۱.۲



شکل ۳

۲.۲

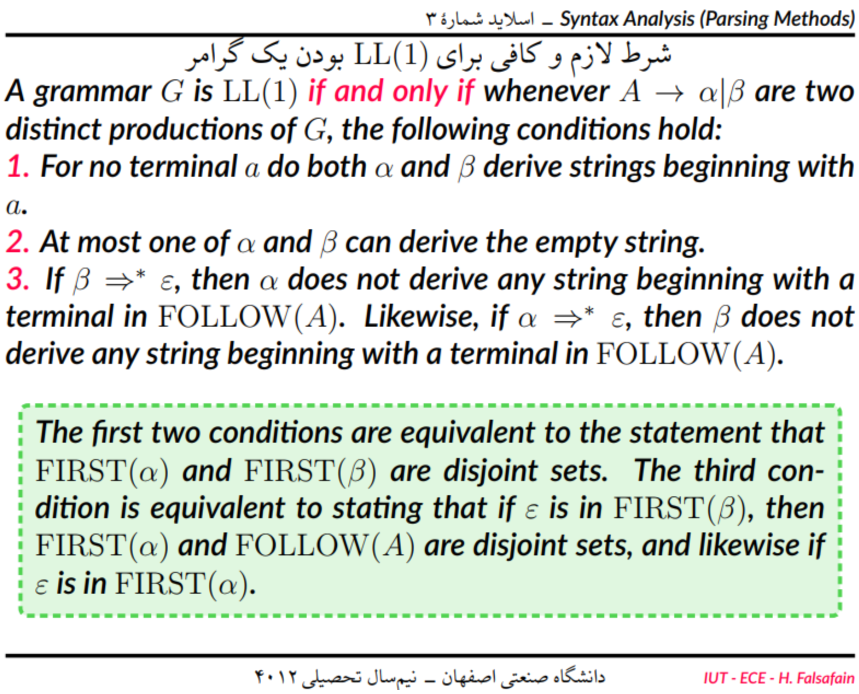


شکل ۴

۳

۱.۳ a

تصویر زیر بیان شرط لازم و کافی برای LL(1) بودن یک گرامر را شرح می‌دهد.



شکل ۵

با توجه به این قضیه، داریم:

$$FIRST(Z) = \{b, \varepsilon\}$$
$$FIRST(Y) = \{b, c\}$$
$$FIRST(bX) = \{b\}$$
$$\Rightarrow FIRST(bX) \bigcap FIRST(Y) \neq \emptyset$$

,

$$FIRST(bZ) = \{b\}$$
$$FOLLOW(Z) = \{c\}$$
$$\Rightarrow FIRST(bZ) \bigcap FOLLOW(Z) = \emptyset$$

از آنجایی که $FIRST(bX) \bigcap FIRST(Y) \neq \emptyset$ پس گرامر LL(1) نیست.

۲.۳ b

با حذف production $X \longrightarrow bX$ گرامر LL(1) می‌شود. چون بین دو شرط بالا اولی که با حذف bX ارضا می‌شود و دومی هم برقرار است.

۴

۱.۴ a

	FIRST	FOLLOW
S	$\{print, \mathbf{ID}, \varepsilon\}$	{ \$ }
ComponentList	$\{print, \mathbf{ID}, \varepsilon\}$	{ \$ }
Component	$\{print, \mathbf{ID}\}$	$\{;\}$
Expression	$\{(, \mathbf{ID}, \mathbf{NUM}\}$	$\{), ;\}$
Operator	$\{\mathbf{ID}, \mathbf{NUM}\}$	$\{+, -, *,), ;\}$
NextStage	$\{+, -, *, \varepsilon\}$	$\{), ;\}$
Operation	$\{+, -, *\}$	$\{), ;\}$

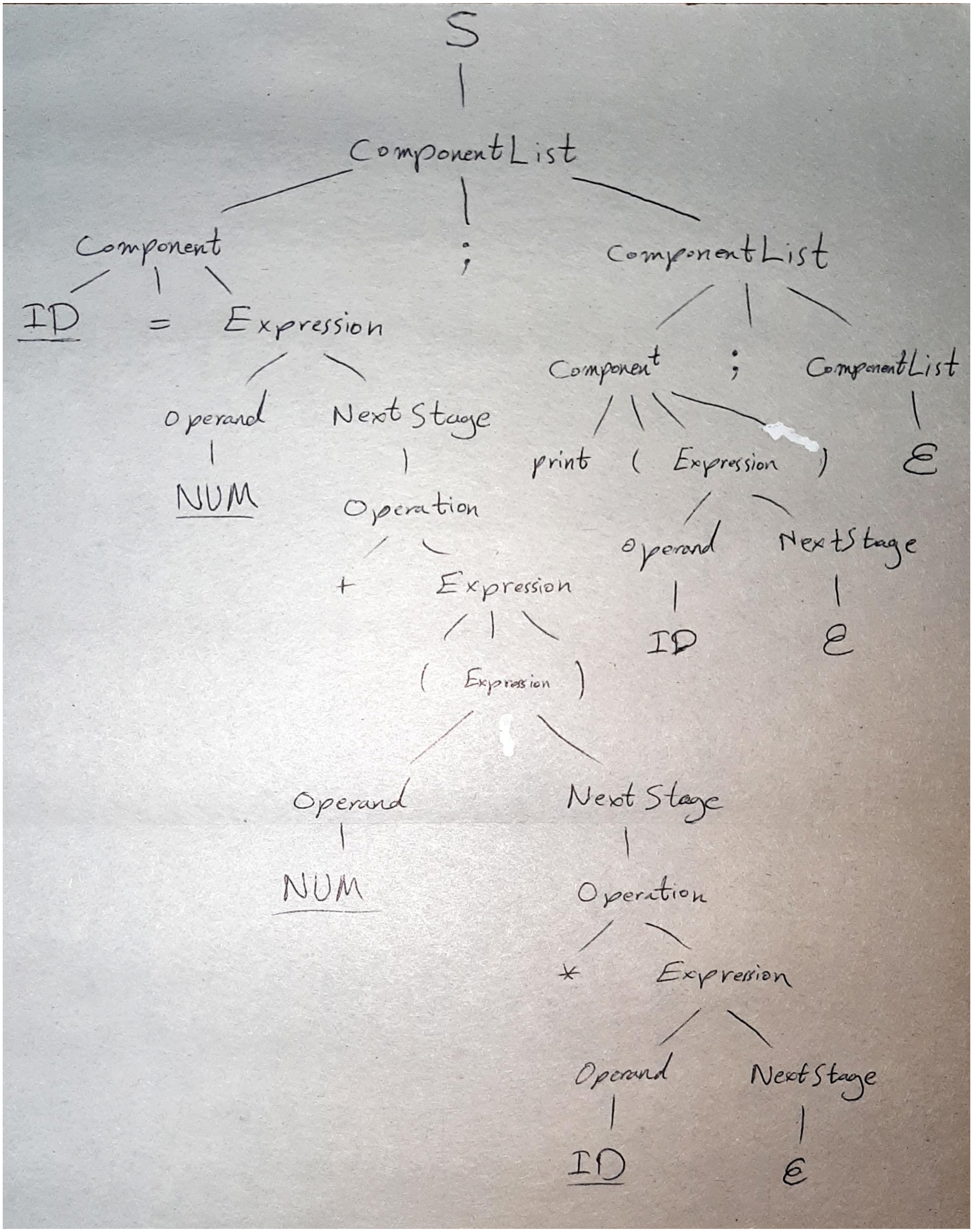
۲.۴ b

	;	print	(	)	**ID**	=	**NUM**	+	-	*	$
S		$S \longrightarrow ComponentList$			$S \longrightarrow ComponentList$						$S \longrightarrow ComponentList$
ComponentList		$ComponentList \longrightarrow Component; ComponentList$			$ComponentList \longrightarrow Component; ComponentList$						$ComponentList \longrightarrow \varepsilon$
Component		$Component \longrightarrow print(Expression)$			$Component \longrightarrow \mathbf{ID} = Expression$						
Expression			$Expression \longrightarrow (Expression)$		$Expression \longrightarrow OperandNextStage$		$Expression \longrightarrow OperandNextStage$				
Operand					$Operand \longrightarrow \mathbf{ID}$		$Operand \longrightarrow \mathbf{NUM}$				
NextStage	$NextStage \longrightarrow \varepsilon$			$NextStage \longrightarrow \varepsilon$				$NextStage \longrightarrow Operation$	$NextStage \longrightarrow Operation$	$NextStage \longrightarrow Operation$	
Operation								$Operation \longrightarrow +Expression$	$Operation \longrightarrow -Expression$	$Operation \longrightarrow *Expression$	

۳.۴ c

Matched	Stack	Input	Action
	S $	**ID** = **NUM** + (**NUM** * **ID**) ; print (**ID**) ; $	
	ComponentList $	**ID** = **NUM** + (**NUM** * **ID**) ; print (**ID**) ; $	output S ⟶ ComponentList
	Component ; ComponentList $	**ID** = **NUM** + (**NUM** * **ID**) ; print (**ID**) ; $	output ComponentList ⟶ Component ; ComponentList
	ID = Expression ; ComponentList $	**ID** = **NUM** + (**NUM** * **ID**) ; print (**ID**) ; $	output Component ⟶ **ID** = Expression
ID	= Expression ; ComponentList $	= **NUM** + (**NUM** * **ID**) ; print (**ID**) ; $	match **ID**
ID =	Expression ; ComponentList $	**NUM** + (**NUM** * **ID**) ; print (**ID**) ; $	match =
ID =	Operand NextStage ; ComponentList $	**NUM** + (**NUM** * **ID**) ; print (**ID**) ; $	output Expression ⟶ Operand NextStage
ID =	**NUM** NextStage ; ComponentList $	**NUM** + (**NUM** * **ID**) ; print (**ID**) ; $	output Operand ⟶ **NUM**
ID = **NUM**	NextStage ; ComponentList $	+ (**NUM** * **ID**) ; print (**ID**) ; $	match **NUM**
ID = **NUM**	Operation ; ComponentList $	+ (**NUM** * **ID**) ; print (**ID**) ; $	output NextStage ⟶ Operation
ID = **NUM**	+ Expression ; ComponentList $	+ (**NUM** * **ID**) ; print (**ID**) ; $	output Operation ⟶ + Expression
ID = **NUM** +	Expression ; ComponentList $	(**NUM** * **ID**) ; print (**ID**) ; $	match +
ID = **NUM** +	(Expression) ; ComponentList $	(**NUM** * **ID**) ; print (**ID**) ; $	output Expression ⟶ (Expression)
ID = **NUM** + (	Expression) ; ComponentList $	**NUM** * **ID**) ; print (**ID**) ; $	match (
ID = **NUM** + (	Operand NextStage) ; ComponentList $	**NUM** * **ID**) ; print (**ID**) ; $	output Expression ⟶ Operand NextStage
ID = **NUM** + (	**NUM** NextStage) ; ComponentList $	**NUM** * **ID**) ; print (**ID**) ; $	output Operand ⟶ **NUM**
ID = **NUM** + (**NUM**	NextStage) ; ComponentList $	* **ID**) ; print (**ID**) ; $	match **NUM**
ID = **NUM** + (**NUM**	Operation) ; ComponentList $	* **ID**) ; print (**ID**) ; $	output NextStage ⟶ Operation
ID = **NUM** + (**NUM**	* Expression) ; ComponentList $	* **ID**) ; print (**ID**) ; $	output Operation ⟶ * Expression
ID = **NUM** + (**NUM** *	Expression) ; ComponentList $	**ID**) ; print (**ID**) ; $	match *
ID = **NUM** + (**NUM** *	Operand NextStage) ; ComponentList $	**ID**) ; print (**ID**) ; $	output Expression ⟶ Operand NextStage
ID = **NUM** + (**NUM** *	**ID** NextStage) ; ComponentList $	**ID**) ; print (**ID**) ; $	output Operand ⟶ **ID**
ID = **NUM** + (**NUM** * **ID**	NextStage) ; ComponentList $	) ; print (**ID**) ; $	match **ID**
ID = **NUM** + (**NUM** * **ID**	) ; ComponentList $	) ; print (**ID**) ; $	output NextStage ⟶ ε
ID = **NUM** + (**NUM** * **ID**)	; ComponentList $	; print (**ID**) ; $	match)
ID = **NUM** + (**NUM** * **ID**) ;	ComponentList $	print (**ID**) ; $	match ;
ID = **NUM** + (**NUM** * **ID**) ;	Component ; ComponentList $	print (**ID**) ; $	output ComponentList ⟶ Component ; ComponentList
ID = **NUM** + (**NUM** * **ID**) ;	print (Experssion) ; ComponentList $	print (**ID**) ; $	output Component ⟶ print (Expression)
ID = **NUM** + (**NUM** * **ID**) ; print	(Experssion) ; ComponentList $	(**ID**) ; $	match print
ID = **NUM** + (**NUM** * **ID**) ; print (	Experssion) ; ComponentList $	**ID**) ; $	match (
ID = **NUM** + (**NUM** * **ID**) ; print (	Operand NextStage) ; ComponentList $	**ID**) ; $	output Expression ⟶ Operand NextStage
ID = **NUM** + (**NUM** * **ID**) ; print (	**ID** NextStage) ; ComponentList $	**ID**) ; $	output Operand ⟶ **ID**
ID = **NUM** + (**NUM** * **ID**) ; print (**ID**	NextStage) ; ComponentList $	) ; $	match **ID**
ID = **NUM** + (**NUM** * **ID**) ; print (**ID**	) ; ComponentList $	) ; $	output NextStage ⟶ ε
ID = **NUM** + (**NUM** * **ID**) ; print (**ID**)	; ComponentList $	; $	match)
ID = **NUM** + (**NUM** * **ID**) ; print (**ID**) ;	ComponentList $	$	match ;
ID = **NUM** + (**NUM** * **ID**) ; print (**ID**) ;	$	$	output ComponentList ⟶ ε

۴.۴ d


S
ComponentList
Component
;
ComponentList
ID
=
Expression
Operand
NextStage
NUM
Operation
+
Expression
(
Expression
)
Operand
NextStage
NUM
Operation
*
Expression
Operand
NextStage
ID
ε
Component
;
ComponentList
print
(
Expression
)
ε
Operand
NextStage
ID
ε

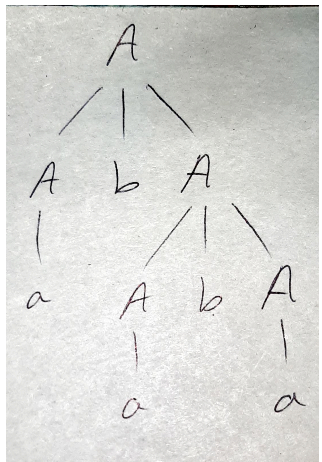
شکل ۶

۵

۱.۵ a

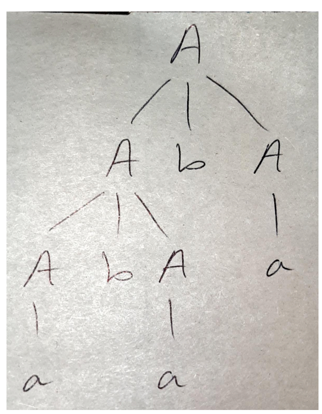
برای اینکه ثابت کنیم این گرامر مبهم است کافی‌ست رشته‌ای بیاوریم که بیش از یک درخت پارس داشته باشد. برای مثال رشته‌ی $ababa$ به دو نحو زیر پارس می‌شود:

۱.۱.۵



شکل ۷

۲.۱.۵



شکل ۸

۲.۵ b

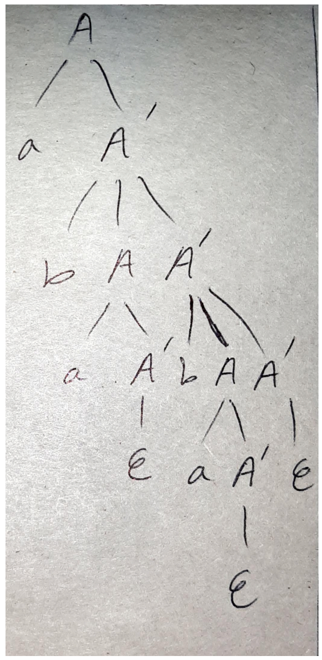
$A \longrightarrow aA'$

$A' \longrightarrow bAA'|\varepsilon$

۳.۵ c

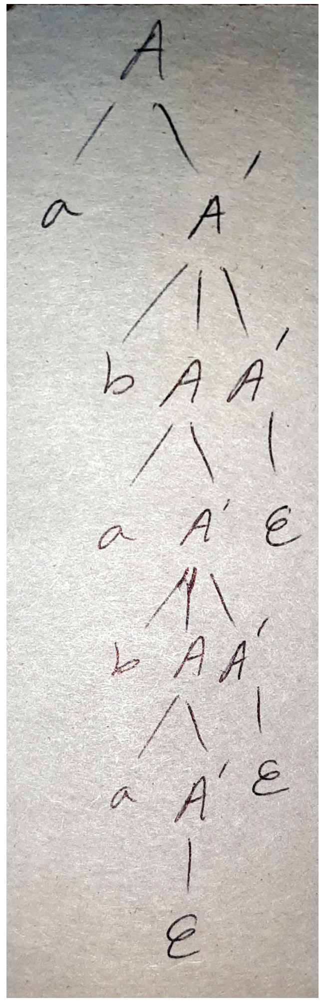
مبهم است. برای مثال رشته‌ی $ababa$ به دو نحو زیر پارس می‌شود:

۱.۳.۵



شکل ۹

۲.۳.۵



شکل ۱۰

۴.۵ c

$A \longrightarrow aA'$

$A' \longrightarrow BA'|\varepsilon$

$B \longrightarrow b$

۶

$E \longrightarrow EE+|EE*|id$

حال left recursion را از این گرامر حذف می‌کنیم.

$E \longrightarrow idE'$

$E' \longrightarrow E + E'|E * E'|\varepsilon$

سپس left factoring را انجام می‌دهیم.

$$E \longrightarrow idE'$$
$$E' \longrightarrow EE''|\varepsilon$$
$$E'' \longrightarrow +E'|*E'$$

تصویر زیر بیان شرط لازم و کافی برای LL(1) بودن یک گرامر را شرح می‌دهد.

Syntax Analysis (Parsing Methods) ــ اسلاید شمارهٔ ۳

شرط لازم و کافی برای $\mathrm{LL}(1)$ بودن یک گرامر

A grammar G is $\mathrm{LL}(1)$ if and only if whenever $A \rightarrow \alpha|\beta$ are two distinct productions of G, the following conditions hold:

1. For no terminal a do both α and β derive strings beginning with a.

2. At most one of α and β can derive the empty string.

3. If $\beta \Rightarrow^* \varepsilon$, then α does not derive any string beginning with a terminal in $\mathrm{FOLLOW}(A)$. Likewise, if $\alpha \Rightarrow^* \varepsilon$, then β does not derive any string beginning with a terminal in $\mathrm{FOLLOW}(A)$.

The first two conditions are equivalent to the statement that $\mathrm{FIRST}(\alpha)$ and $\mathrm{FIRST}(\beta)$ are disjoint sets. The third condition is equivalent to stating that if ε is in $\mathrm{FIRST}(\beta)$, then $\mathrm{FIRST}(\alpha)$ and $\mathrm{FOLLOW}(A)$ are disjoint sets, and likewise if ε is in $\mathrm{FIRST}(\alpha)$.

IUT - ECE - H. Falsafain دانشگاه صنعتی اصفهان ــ نیم‌سال تحصیلی ۴۰۱۲

شکل ۱۱

حال مجموعه‌های FIRST و FOLLOW را به دست می‌آوریم.

	FIRST	FOLLOW
E	$\{id\}$	{ +, *, $ }
E'	$\{id, \varepsilon\}$	{ +, *, $ }
E''	$\{+, *\}$	{ +, *, $ }

از آنجایی که FOLLOW(E') با FIRST(EE'') اشتراک ندارد، پس گرامر LL(1) است.