



دانشگاه صنعتی اصفهان  
دانشکده مهندسی برق و کامپیوتر

عنوان: تکلیف تئوری دوم درس کامپایلر

نام و نام خانوادگی: علیرضا ابره فروش

شماره دانشجویی: ۹۸۱۶۶۰۳

نیم سال تحصیلی: بهار ۱۴۰۲/۱۴۰۱

مدرس: دکتر حسین فلسفین

۱

**a** ۱.۱

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  $\alpha$  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  $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \cdots \mid \alpha\beta_n \mid \gamma$ , where  $\gamma$  represents all alternatives that do not begin with  $\alpha$ , by

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

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

شکل ۱

$$\begin{aligned} S &\rightarrow SS' \\ S' &\rightarrow +S \mid +P \\ P &\rightarrow PP' \\ P' &\rightarrow *P \mid *I \\ I &\rightarrow -I \mid (S) \mid D \\ D &\rightarrow 0 \mid 1N \\ N &\rightarrow NN' \mid 0 \mid 1 \mid \epsilon \\ N' &\rightarrow N \end{aligned}$$

**b** ۲.۱

طبق الگوریتم زیر عمل می‌کنیم.

**Immediate left recursion can be eliminated by the following technique, which works for any number of  $A$ -productions. First, group the productions as**

$$A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_m | \beta_1 | \beta_2 | \dots | \beta_n$$

**where no  $\beta_i$  begins with an  $A$ . Then, replace the  $A$ -productions by**

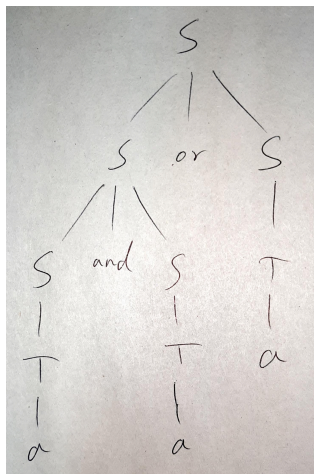
$$\begin{aligned} A &\rightarrow \beta_1 A' | \beta_2 A' | \dots | \beta_n A' \\ A' &\rightarrow \alpha_1 A' | \alpha_2 A' | \dots | \alpha_m A' | \epsilon \end{aligned}$$

**The nonterminal  $A$  generates the same strings as before but is no longer left recursive.**

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

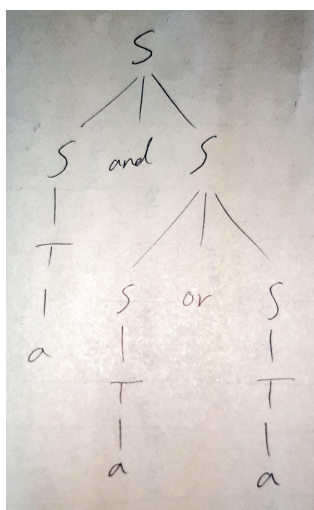
بسته به اینکه ابتدا کدام production نظیر S را پارس کنیم درخت پارس می‌تواند متفاوت باشد. پس دو درخت پارس زیر برای تجزیه این رشته موجود است.

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.

شرط لازم و کافی برای LL(1) بودن یک گرامر

A grammar  $G$  is 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  $\alpha$  and  $\beta$  derive strings beginning with  $a$ .
2. At most one of  $\alpha$  and  $\beta$  can derive the empty string.
3. If  $\beta \Rightarrow^* \varepsilon$ , then  $\alpha$  does not derive any string beginning with a terminal in  $\text{FOLLOW}(A)$ . Likewise, if  $\alpha \Rightarrow^* \varepsilon$ , then  $\beta$  does not derive any string beginning with a terminal in  $\text{FOLLOW}(A)$ .

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

### شکل ۵

با توجه به این قضیه، داریم:

$$\begin{aligned} \text{FIRST}(Z) &= \{b, \varepsilon\} \\ \text{FIRST}(Y) &= \{b, c\} \\ \text{FIRST}(bX) &= \{b\} \\ \Rightarrow \text{FIRST}(bX) \cap \text{FIRST}(Y) &\neq \emptyset \\ , \\ \text{FIRST}(bZ) &= \{b\} \\ \text{FOLLOW}(Z) &= \{c\} \\ \Rightarrow \text{FIRST}(bZ) \cap \text{FOLLOW}(Z) &= \emptyset \end{aligned}$$

از آنجایی که  $\text{FIRST}(bX) \cap \text{FIRST}(Y) \neq \emptyset$  پس گرامر LL(1) نیست.

### ۲.۳ b

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

۴

a ۱.۴

|               | 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     | $\{ +, -, * \}$                       | $\{ ), ; \}$          |

b ۲.۴

|               | :  | print   | (   | )  | ID  | = | NUM   | +  | -  | *  | \$   |
|---------------|--|---|---|--|---|---|---|--|--|--|--|
| S             |  | $S \rightarrow \text{ComponentList}$  |   |  | $S \rightarrow \text{ComponentList}$  |   |   |  |  |  | $S \rightarrow \text{ComponentList}$           |
| ComponentList |  | $\text{ComponentList} \rightarrow \text{ComponentList}, \text{ComponentList}$ |   |  | $\text{ComponentList} \rightarrow \text{ComponentList}, \text{ComponentList}$ |   |   |  |  |  | $\text{ComponentList} \rightarrow \varepsilon$ |
| Component     |  | $\text{Component} \rightarrow \text{print}(\text{Expression})$                |   |  | $\text{Component} \rightarrow \mathbf{ID} = \text{Expression}$                |   |   |  |  |  |  |
| Expression    |  |   | $\text{Expression} \rightarrow (\text{Expression})$ |  | $\text{Expression} \rightarrow \text{OperandNextStage}$                       |   | $\text{Expression} \rightarrow \text{OperandNextStage}$ |  |  |  |  |
| Operand       |  |   |   |  | $\text{Operand} \rightarrow \mathbf{ID}$                                      |   | $\text{Operand} \rightarrow \mathbf{NUM}$               |  |  |  |  |
| NextStage     | $\text{NextStage} \rightarrow \varepsilon$ |   |   | $\text{NextStage} \rightarrow \varepsilon$ |   |   |   | $\text{NextStage} \rightarrow \text{Operation}$    | $\text{NextStage} \rightarrow \text{Operation}$    | $\text{NextStage} \rightarrow \text{Operation}$    |  |
| Operation     |  |   |   |  |   |   |   | $\text{Operation} \rightarrow + \text{Expression}$ | $\text{Operation} \rightarrow - \text{Expression}$ | $\text{Operation} \rightarrow * \text{Expression}$ |  |

c ۳.۴

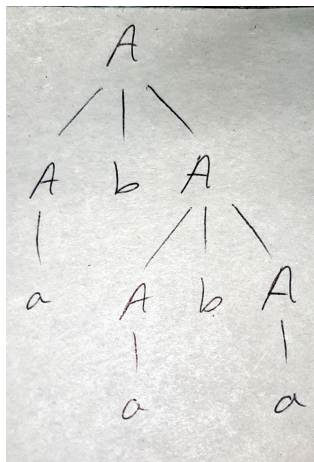
| Matched                                  | Stack                                   | Input                                       | Action   |
|--|---|---|--|
|  | \$                                      | ID = NUM + ( NUM * ID ) ; print ( ID ) ; \$ |  |
|  | ComponentList \$                        | ID = NUM + ( NUM * ID ) ; print ( ID ) ; \$ | output \$ $\rightarrow$ ComponentList                        |
|  | Component ; ComponentList \$            | ID = NUM + ( NUM * ID ) ; print ( ID ) ; \$ | output ComponentList $\rightarrow$ Component ; ComponentList |
|  | ID = Expression ; ComponentList \$      | ID = NUM + ( NUM * ID ) ; print ( ID ) ; \$ | output Component $\rightarrow$ 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 $\rightarrow$ Operand NextStage            |
| ID =                                     | NUM NextStage ; ComponentList \$        | NUM + ( NUM * ID ) ; print ( ID ) ; \$      | output Operand $\rightarrow$ NUM                             |
| ID = NUM                                 | NextStage ; ComponentList \$            | + ( NUM * ID ) ; print ( ID ) ; \$          | match NUM  |
| ID = NUM                                 | Operation ; ComponentList \$            | + ( NUM * ID ) ; print ( ID ) ; \$          | output NextStage $\rightarrow$ Operation                     |
| ID = NUM                                 | + Expression ; ComponentList \$         | + ( NUM * ID ) ; print ( ID ) ; \$          | output Operation $\rightarrow$ + Expression                  |
| ID = NUM +                               | Expression ; ComponentList \$           | ( NUM * ID ) ; print ( ID ) ; \$            | match +  |
| ID = NUM +                               | ( Expression ) ; ComponentList \$       | ( NUM * ID ) ; print ( ID ) ; \$            | output Expression $\rightarrow$ ( Expression )               |
| ID = NUM + (                             | Expression ; ComponentList \$           | NUM * ID ) ; print ( ID ) ; \$              | match (  |
| ID = NUM + (                             | Operand NextStage ; ComponentList \$    | NUM * ID ) ; print ( ID ) ; \$              | output Expression $\rightarrow$ Operand NextStage            |
| ID = NUM + (                             | NUM NextStage ; ComponentList \$        | NUM * ID ) ; print ( ID ) ; \$              | output Operand $\rightarrow$ NUM                             |
| ID = NUM + ( NUM                         | NextStage ; ComponentList \$            | * ID ) ; print ( ID ) ; \$                  | match NUM  |
| ID = NUM + ( NUM                         | Operation ; ComponentList \$            | * ID ) ; print ( ID ) ; \$                  | output NextStage $\rightarrow$ Operation                     |
| ID = NUM + ( NUM                         | * Expression ; ComponentList \$         | * ID ) ; print ( ID ) ; \$                  | output Operation $\rightarrow$ * Expression                  |
| ID = NUM + ( NUM *                       | Expression ; ComponentList \$           | ID ) ; print ( ID ) ; \$                    | match *  |
| ID = NUM + ( NUM *                       | Operand NextStage ; ComponentList \$    | ID ) ; print ( ID ) ; \$                    | output Expression $\rightarrow$ Operand NextStage            |
| ID = NUM + ( NUM *                       | ID NextStage ; ComponentList \$         | ID ) ; print ( ID ) ; \$                    | output Operand $\rightarrow$ ID                              |
| ID = NUM + ( NUM * ID                    | NextStage ; ComponentList \$            | ) ; print ( ID ) ; \$                       | match ID   |
| ID = NUM + ( NUM * ID                    | ) ; ComponentList \$                    | ) ; print ( ID ) ; \$                       | output NextStage $\rightarrow \epsilon$                      |
| 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 $\rightarrow$ Component ; ComponentList |
| ID = NUM + ( NUM * ID ) ;                | print ( Expression ) ; ComponentList \$ | print ( ID ) ; \$                           | output Component $\rightarrow$ print ( Expression )          |
| ID = NUM + ( NUM * ID ) ; print          | ( Expression ) ; ComponentList \$       | ( ID ) ; \$                                 | match print  |
| ID = NUM + ( NUM * ID ) ; print (        | Expression ; ComponentList \$           | ID ) ; \$                                   | match (  |
| ID = NUM + ( NUM * ID ) ; print (        | Operand NextStage ; ComponentList \$    | ID ) ; \$                                   | output Expression $\rightarrow$ Operand NextStage            |
| ID = NUM + ( NUM * ID ) ; print (        | ID NextStage ; ComponentList \$         | ID ) ; \$                                   | output Operand $\rightarrow$ ID                              |
| ID = NUM + ( NUM * ID ) ; print ( ID     | NextStage ; ComponentList \$            | ) ; \$                                      | match ID   |
| ID = NUM + ( NUM * ID ) ; print ( ID     | ) ; ComponentList \$                    | ) ; \$                                      | output NextStage $\rightarrow \epsilon$                      |
| ID = NUM + ( NUM * ID ) ; print ( ID )   | ; ComponentList \$                      | ; \$  | match )  |
| ID = NUM + ( NUM * ID ) ; print ( ID ) ; | ComponentList \$                        | \$  | match ;  |
| ID = NUM + ( NUM * ID ) ; print ( ID ) ; | \$                                      | \$  | output ComponentList $\rightarrow \epsilon$                  |

۵

a ۱.۵

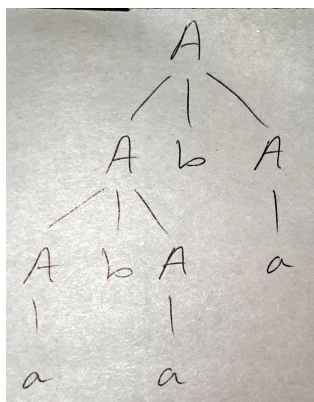
برای اینکه ثابت کنیم این گرامر مبهم است کافیست رشته‌ای بیاوریم که بیش از یک درخت پارس داشته باشد. برای مثال رشته‌ی *ababa* به دو نحو زیر پارس می‌شود:

۱.۱.۵



شکل ۶

۲.۱.۵



شکل ۷

b ۲.۵

$$A \rightarrow aA'$$

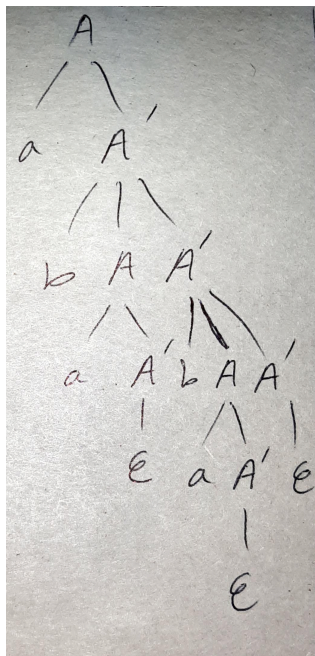
$$A' \rightarrow bAA' | \varepsilon$$

c ۳.۵

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

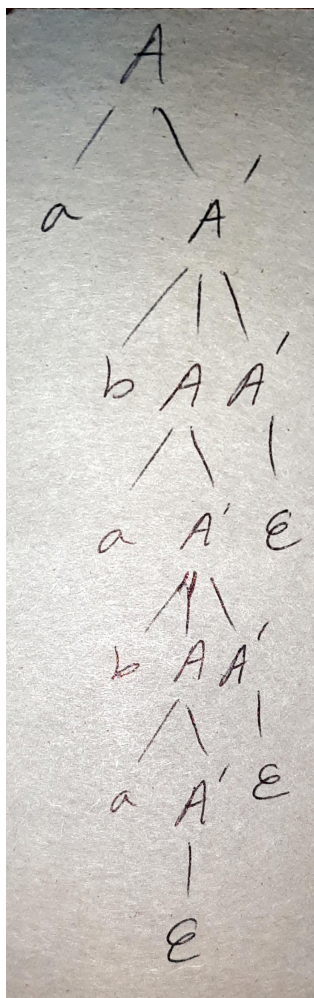


۱.۳.۵



شکل ۸

۲.۳.۵



شکل ۹

c ۴.۵

$$A \rightarrow aA'$$

$$A' \rightarrow BA' | \epsilon$$

$$B \rightarrow b$$

۶

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

$$E \rightarrow idE'$$

حال left recursion را از این گرامر حذف می‌کنیم.

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