۱- در مورد گرامر و رشته‌ی ورودی زیر به سوالاتی که در ادامه آمده است پاسخ دهید :

$S \rightarrow S + S \mid S\,S \mid (\,S\,) \mid S^* \mid a$ with string $(a + a) * a$

الف ) یک اشتقاق چپ (left-most derivation ) برای رشته‌ی ورودی بنویسید.

ب ) یک اشتقاق راست (right-most derivation ) برای رشته‌ی ورودی بنویسید.

ج ) یک درخت تجزیه برای این رشته بنویسید.

د ) آیا این گرامر مبهم است یا غیرمبهم ؟ توضیح دهید.

ه ) زبانی که این گرامر تولید میکند را شرح دهید.

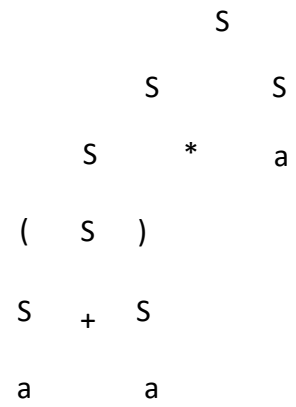
الف )

$$S \rightarrow S\,S \rightarrow S * S \rightarrow (\,S\,) * S \rightarrow (\,S + S\,) * S \rightarrow (\,a + S\,) * S \rightarrow (\,a + a\,) * S \rightarrow (\,a + a\,) * a$$

ب )

$$S \rightarrow S\,S \rightarrow S\,a \rightarrow S * a \rightarrow (\,S\,) * a \rightarrow (\,S + S\,) * a \rightarrow (\,S + a\,) * a \rightarrow (\,a + a\,) * a$$

ج )




د )

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

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

این گرامر مبهم است ، همانطور که در بالا دیدیم ، دو اشتقاق چپ مختلف برای یک عبارت ورودی وجود داشت که این نشانگر ابهام گرامر است.

جهت رفع ابهام ، میتوانیم اولویت گذاری کنیم .

ه ) این گرامر سعی دارد به نوعی RE ها را تولید کند ، به این شکل که + بیانگر or است و عدم وجود هیچ اوپراتوری ( منظور قانون SS است ) به معنای concat می‌باشد و * همان Kleene Star است .

۲- گرامر را به گرامری تبدیل کنید که بازگشتی چپ مستقیم یا ضمنی نداشته باشد.

$A \rightarrow Ba \mid Aa \mid c$

$B \rightarrow Bb \mid Ab \mid d$

ابتدا بازگشتی های مستقیم B را حذف میکنم :

$'B \rightarrow AbB' \mid dB$

$B' \rightarrow bB' \mid \varepsilon$

حالا با درنظر گرفتن قانون جدیدB ، آنها را در قانون A جایگزین میکنم :

$A \rightarrow AbB'a \mid dB'a \mid Aa \mid c$

حالا بازگشتی های مستقیم A را نیز حذف میکنم :

$'A \rightarrow dB'aA' \mid cA$

$A' \rightarrow bB'aA' \mid aA' \mid \varepsilon$

در نهایت ، گرامر اصلاح شده به صورت زیر است :

$'A \rightarrow dB'aA' \mid cA$

$A' \rightarrow bB'aA' \mid aA' \mid \varepsilon$

$'B \rightarrow AbB' \mid dB$

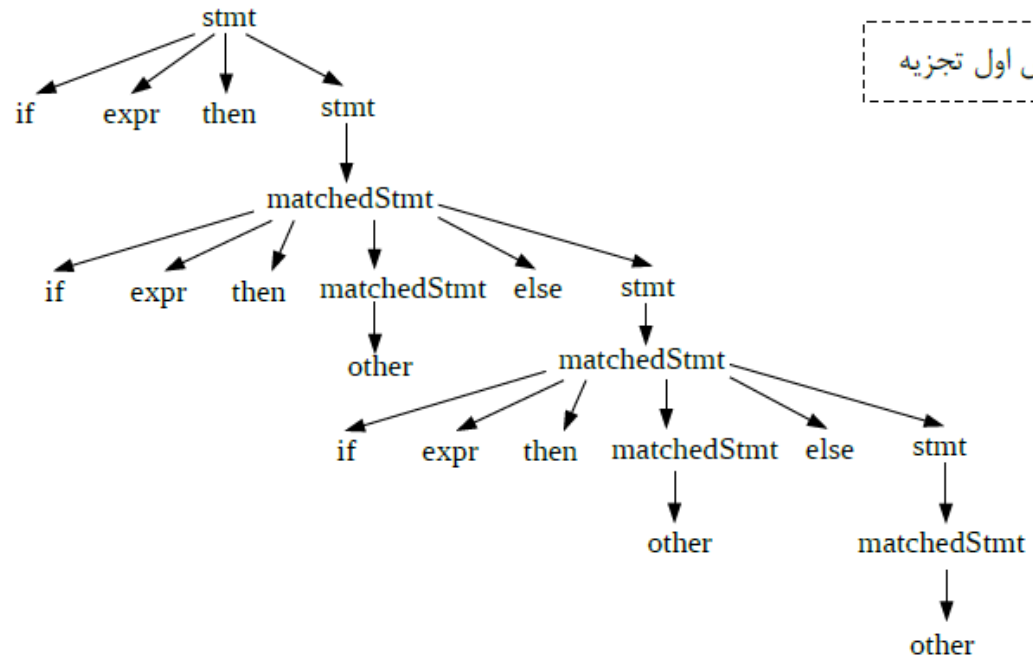
$B' \rightarrow bB' \mid \varepsilon$

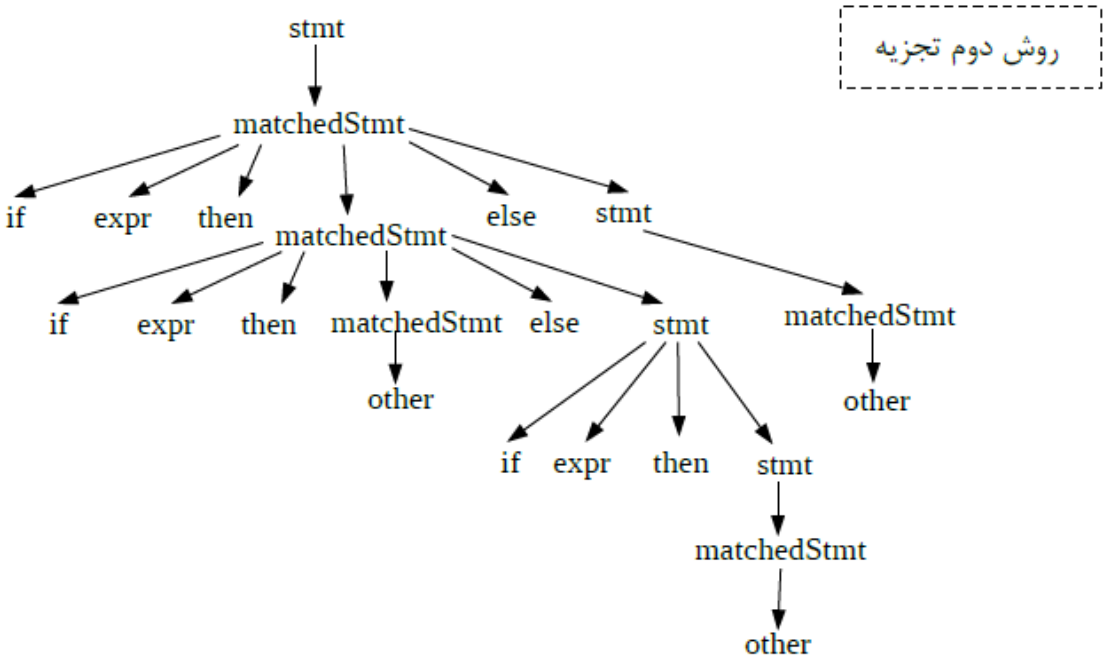
---

**۳- گرامر زیر تلاشی جهت نوشتن گرامر عبارات شرطی بدون ابهام است . آیا ابهام برطرف شده است ؟ اگر هنوز گرامر مبهم است ، نشان دهید که ابهام وجود دارد ؟**

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & matchedStmt \\
matchedStmt & \rightarrow & \textbf{if } expr \textbf{ then } matchedStmt \textbf{ else } stmt \\
 & | & \textbf{other}
\end{array}
$$

مطابق زیر دو درخت تجزیه متفاوت برای یک رشته یکتا وجود دارد پس گرامر مبهم است.



۴- برای گرامر زیر یک sudocode برای اجرای الگوریتم Recursive Descent به صورت بازگشتی بنویسید .

$$
\begin{aligned}
S &\rightarrow S \text{ and } S \\
&\mid S \text{ or } S \\
&\mid T \\
&\mid \text{a} \\
T &\rightarrow \text{a}
\end{aligned}
$$

پاسخ :

در این سؤال مشخص نشده است که RD به صورت پیشگو یا backtracked باشد. اما از آنجا که گرامر فوق ابهام دارد، دارای بازگشتی چپ است و نیاز به فاکتورگیری هم دارد، هیچ یک از دو روش backtracked و پیشگو برای آن صحیح کار نمی‌کند. برای اینکه الگوریتم RD به درستی کار کند باید گرامر ابهام نداشته باشد، دارای بازگشتی از چپ نباشد و نیاز به فاکتورگیری هم نداشته باشد.

برای رفع ابهام می توانیم ابتدا اپراتورهای قواعد گرامر را اولویت بندی کنیم و گرامر جدیدی برای زبان گرامر اولیه بنویسیم سپس در گرامر جدید بازگشتی از چپ وجود دارد که آن را حذف میکنیم. پس در اینصورت مطابق زیر خواهیم داشت:

$$
\begin{aligned}
S &\rightarrow S \text{ or } R \mid R \\
R &\rightarrow R \text{ and } T \mid T \\
T &\rightarrow \text{a}
\end{aligned}
$$

$$
\begin{aligned}
S &\rightarrow RS' \\
S' &\rightarrow \text{or } RS' \mid \varepsilon \\
R &\rightarrow TR' \\
R' &\rightarrow \text{and} TR' \mid \varepsilon \\
T &\rightarrow \text{a}
\end{aligned}
$$

```
Match(token x) {if (*next == x) {next++; return true;}
                else return false;}
main(){
  S();
}
```

اکنون اگر RD را به صورت backtracked بنویسیم به صورت زیر خواهد بود:

```
 bool S() { return R() & S'() }
bool S'(){ save=next;
        return {match('or') & R() & S'(); } || {next=save; return}
}
bool R(){ return T() & R'();}
bool R'(){ save=next;
        return {match('and') & T() & R'(); } || {next=save; return}
}
bool T(){ return match('a') }
```

و اگر RD را به صورت predictive بنویسیم به صورت زیر خواهد بود:

```
bool S() { if *next=='a' { R(); S'(); } else Error}
bool S'(){ if *next=='or'){ match('or'); R(); S'(); }
        else if *next==$ return
          else Error}
bool R(){ if *next=='a' {T(); R'();} else Error}
bool R'(){ if *next == 'and'){ match('and'); T(); R'(); }
            else if *next=$ return
          else Error}
bool T(){ return match('a') }
```

5- الف) خیر زیرا گرامر هم دارای تداخل first ، first و هم دارای بازگشتی چپ است پس LL1نیست.

ب) شرط لازم و کافی برای LL(k) بودن گرامر مطابق زیر است :

یعنی برای هر قانون $A \rightarrow \alpha \mid \beta$ داریم :

1. $first_k(\alpha) \cap first_k(\beta) = \emptyset$
2. if $\beta \rightarrow^* \varepsilon$ then : $follow_k(A) \cap first_k(\alpha) = \emptyset$

گرامر اولیه LL(k) نیست زیرا به ازای هر k می توان گفت :

aa .. a (k number )∈ Firstk(Ax)

aa .. a (k number )∈Firstk(Ay)

aa .. a (k number )∈Firstk(aA)

در حالی که اگر بازگشتی چپ را حذف کنیم خواهیم داشت :

A → aabA' | aAA'

A' → xA' | yA' | epsilon

گرامر فوق یک گرامر LL(3) است زیرا :

First1(aabA') = {a} , First2(aabA') = {aa}, First3(aabA') = {aab} ,

First1(aAA') = {a} , First2(aAA') = {aa}, First3(aAA') = {aaa }

Fallow3 (A') = {$} ∩ first3(xA') = ø

Fallow3 (A') = {$} ∩ first3(yA') = ø

اولین K ای که موجب تداخل نمی شود k=3 است پس این گرامر یک گرامر LL(4) و به طور کلی LL(K) برای k>=3 است.

ج)

گرامر LL(1) معادل مطابق زیر است.

A → aaTS

T → b | aT

S → xS | yS | epsilon

د)برای هر گرامر LL(K) با فاکتور گیری K-1 توکن اول می توان گرامر معادل و LL(1) طراحی کرد به عبارتی قدرت تجزیه گر LL(K) و LL(1) مشابه است.

**۶- (سوال اختیاری) گرامر زیر را در نظر بگیرید و فرض کنید e , s ترمینال هستند. جهت حل ابهامی که در بسط اختیاری else ( در غیرپایانی stmtTail ) پیش می‌آید ، هر جا از ورودی ، else دیدیم آن را مصرف می‌کنیم. بدین ترتیب می‌توانیم یک تجزیه‌گر پیشگو ( predictive parser ) بسازیم. با استفاده از ایده سمبل‌های هماهنگ ساز (synchronizing ) به سوالات زیر پاسخ دهید.**

**الف ) برای این گرامر یک جدول تجزیه بسازید که خطاها را نیز اصلاح کند.**

| | | |
|---|---|---|
| *stmt* | → | **if** *e* **then** *stmt stmtTail* |
| | \| | **while** *e* **do** *stmt* |
| | \| | **begin** *list* **end** |
| | \| | *s* |
| *stmtTail* | → | **else** *stmt* |
| | \| | $\epsilon$ |
| *list* | → | *stmt listTail* |
| *listTail* | → | ; *list* |
| | \| | $\epsilon$ |

قواعد از 1 تا 9 به ترتیب نامگذاری شده اند.

First(if e then stmt stmtTail) = {if}

First(while e do stmt) = {while}

First(begin list end) = {begin}

First(s) = {s}

First(else stmt) = {else}

First(stmt listTail) = {if , while, begin, s}

First(; list) = {;}

Fallow(stmt) = {$, else, ; , end }

Fallow(stmtTail) = {$, else, ; , end}

Fallow(list) = {end}

Fallow(listTail) = {end}

| | if | then | while | begin | s | do | else | ; | end | e | $ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Stmt | 1 | | 2 | 3 | 4 | | s | s | s | | s |
| stmtTail | | | | | | | 5<br>6 حذف | 6 | 6 | | 6 |
| List | 7 | | 7 | 7 | 7 | | | | s | | |
| listTail | | | | | | | | 8 | 9 | | |