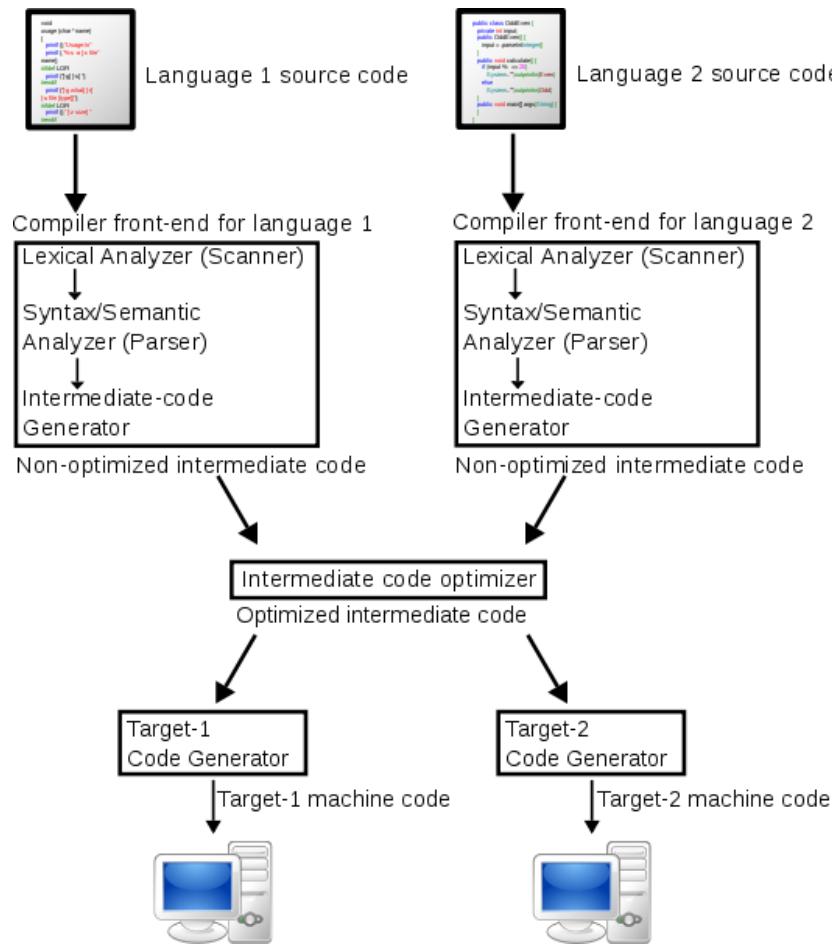


# اصول طراحی کامپایلر



جواد شاهپریان

گروه مهندسی کامپیوتر

دانشکده فنی دانشگاه آزاد اسلامی واحد تهران جنوب

مهرماه ۱۳۹۱

## فهرست مطالب

۵	۱ فصل اول : مقدمات
۵	۱,۱ انواع مترجم
۵	۱,۲ تعریف کامپایلر
۵	۱,۳ تعریف اسمبلر
۵	۱,۴ تفاوت مفسر و کامپایلر
۶	۱,۵ بخش‌های مختلف یک کامپایلر
۶	۱,۶ پردازشگر خطا (Error Handler)
۶	۱,۷ جدول نمادها (Symbol Table)
۱۰	۲ فصل دوم : بررسی انواع گرامرها و خواص عمومی زبانها
۱۰	۲,۱ گرامر
۱۰	Term ۲,۲
۱۰	۲,۳ پایانه ها (Terminal)
۱۰	۲,۴ غیر پایانه ها (Non Terminal)
۱۰	۲,۵ نماد شروع (Start Symbol)
۱۰	۲,۶ الفبا (Alphabet)
۱۰	۲,۷ رشته (String)
۱۱	۲,۸ پیشوند (Prefix)
۱۱	۲,۹ پسوند (Postfix)
۱۱	۲,۱۰ زیر رشته (SubString)
۱۱	۲,۱۱ اجتمع (Union)
۱۱	۲,۱۲ الحاق (Concat)
۱۱	۲,۱۳ گرامرهای معادل
۱۲	۲,۱۴ زبان (Language)
۱۲	۲,۱۵ انواع گرامر
۱۳	۲,۱۶ اشتقاق (Derivation)
۱۳	۲,۱۷ فرم جمله‌ای
۱۳	۲,۱۸ استنتاج

۱۳	۲,۱۹ انواع استنتاج
۱۴	۲,۲۰ گرامر مبهم
۱۶	۲,۲۱ گرامرهای مختصر و مفید
۱۷	۲,۲۲ عبارتهای منظم
۱۹	۳ فصل سوم : تحلیلگر لغوی
۱۹	۳,۱ وظایف تحلیلگر لغوی (Scanner)
۱۹	۳,۲ جدا کننده ها (Separator)
۱۹	۳,۳ انواع موجود در زبان (انواع کلمات)
۲۰	۳,۴ شناسه (Identifier)
۲۰	۳,۵ ماشینهای خودکار (Automata)
۲۲	۳,۶ الگوریتم تبدیل یک دیاگرام به گرامر معادلش
۲۳	۴ فصل چهارم : روشاهی تحلیل نحوی
۲۳	۴,۱ تحلیلگر نحوی (Parser)
۲۳	۴,۲ درخت نحوی (Syntax Tree)
۲۵	۴,۳ نحوه ایجاد درخت خلاصه شده
۲۵	۴,۴ وظیفه پارسرا (Pass)
۲۵	۴,۵ گذر (Pass)
۲۵	۴,۶ انواع روشاهی Parse
۲۶	۴,۷ پارسراهی پیشگو
۲۷	۴,۸ تابع First
۲۷	۴,۹ مراحل به دست آوردن (x) First
۲۷	۴,۱۰ تابع Follow
۲۸	۴,۱۱ مراحل یافتن (A) Follow
۲۸	۴,۱۲ پارس به روش LL(1)
۲۹	۴,۱۳ جدول پارس (LL(1))
۲۹	۴,۱۴ نحوه تشکیل جدول پارس (LL(1))
۳۱	۴,۱۵ پارس با استفاده از روش (1) LL
۳۳	۴,۱۶ شرایط (1) LL بودن یک گرامر
۳۵	۴,۱۷ فاکتور گیری

۳۶	چپگردی ۴,۱۸
۳۷	حذف چپگردی ۴,۱۹
۳۷	چپگردی غیر صریح ۴,۲۰
۳۸	۵ فصل پنجم : روشهای اصلاح خطای نحوی
۳۸	۵,۱ انواع خطای
۳۸	روش Panic Mode ۵,۲
۳۸	روش Phrase Level ۵,۳
۳۹	روش Error Production ۵,۴
۳۹	روش Global Correction ۵,۵
۴۰	۵,۶ اصلاح خطای به روش Panic Mode در پارسرهای LL(1)
۴۲	۵,۷ روش پارس پایین گرد
۴۵	۵,۸ پارسرهای پایین به بالا
۴۵	۵,۹ روش انتقال-کاهش (Shift-Reduce)
۴۷	۵,۱۰ انواع تداخل
۴۸	۵,۱۱ روشهای LR
۴۹	۵,۱۲ چهار مزیت اصلی روشهای LR
۴۹	LR(0).item ۵,۱۳
۵۵	۵,۱۴ روش CLR
۵۷	۵,۱۵ روش LALR
۶۴	۵,۱۶ اصلاح خطای به روش Phrase Level در پارسرهای LR
۶۵	۵,۱۷ روش تقدم عملگر
۶۸	۵,۱۸ پارس به روش تقدم عملگر
۷۰	۵,۱۹ روش تقدم ساده
۷۲	۵,۲۰ پارس به روش تقدم ساده
۷۴	۵,۲۱ چپگردی
۷۴	۵,۲۲ رفع چپگردی
۷۴	۵,۲۳ راستگردی
۷۴	۵,۲۴ رفع راستگردی
۷۴	۵,۲۵ تحلیلگر معنایی



## ۱ فصل اول : مقدمات

### ۱,۱ انواع مترجم

۱. کامپایلر (Compiler)
۲. مفسر (Interpreter)
۳. اسembler (Assembler)

### ۱,۲ تعریف کامپایلر

عمل ترجمه از یک زبان سطح بالا به زبان سطح پایین را کامپایل گویند که در واقع زبان مبدأ را به زبان مقصد ترجمه مینماید.



### ۱,۳ تعریف اسembler

مترجمی است که زبان اسembler را به زبان ماشین تبدیل میکند.

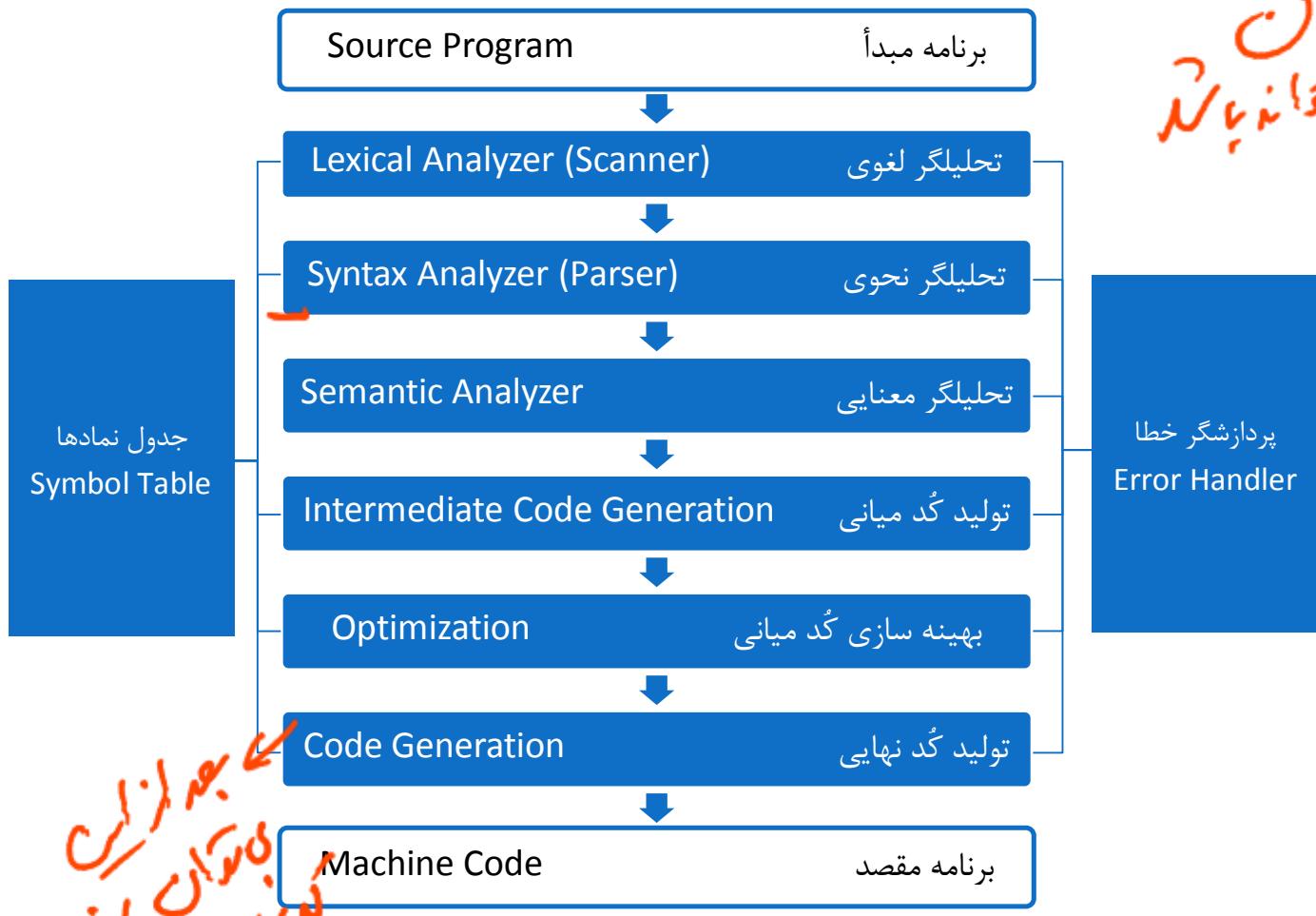
### ۱,۴ تفاوت مفسر و کامپایلر

کامپایلر کل برنامه را یکجا و یکباره کامپایل میکند و بارها آن را اجرا میکند، اما مفسر خط به خط برنامه را ترجمه و اجرا میکند.

« تمرین: تفاوتها و شباهت‌های کامپایلر و مفسر را بیان کنید.

**کوال اکھن ہی کفار نہ پار** ✓

## ۱.۵ بخش‌های مختلف یک کامپایلر

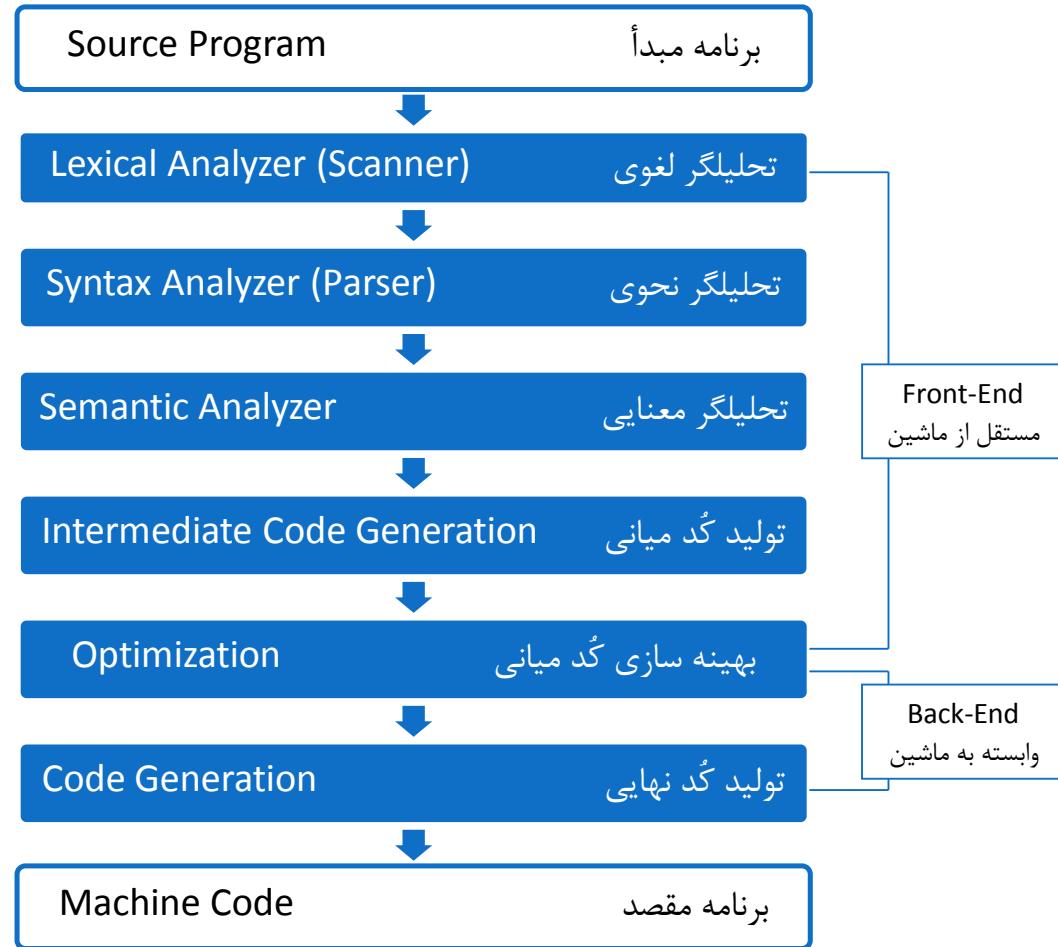


## ۱.۶ پردازشگر خطای (ErrorHandler)

قسمت دیگری که در همه مراحل درگیر است، پردازشگر خطای است که به صورت مفصل به آن خواهیم پرداخت.

## ۱.۷ جدول نمادها (Symbol Table)

قسمت دیگر کامپایلر جدولی است که در آن نمادها و علایم کامپایلر قرار دارند.



« مثال: نمونه هایی از تشخیص خطای توسط کامپایلر.

inta;

خطای لغوی (بین int و a فاصله وجود ندارد)

int a

خطای نحوی (در آخر عبارت، ';' وجود ندارد)

int a = 5.5;

خطای معنایی (قرار دادن مقدار اعشاری در متغیر نوع صحیح)

int a[10]; a[12] = 3;

خطای معنایی (دستیابی به اندیس آرایه خارج از محدوده)

«مثال: گرامر زبان فارسی:

```

<جمله> ← <فاعل> <مفعول> < فعل > .
<فعل> ← خرید | خورد
<مفعول> ← سیب | موز
<فاعل> ← علی | حسن | تمام کلمات ۴ حرفی که با حرف "د" شروع میشوند

```

عباراتی که در هنگام کامپایل باعث بروز خطای میشوند:

- |  |                |
|--|----------------|
| < خطای در تحلیلگر لغوی: احمد به عنوان فاعل تعریف نشده است. | احمد سیب خورد. |
| < خطای در تحلیلگر نحوی: قواعد جمله رعایت نشده است.         | حسن سیب موز.   |
| < خطای در تحلیلگر معنایی: جمله بی معنی است.                | درخت موز خورد. |
| < خطای در تحلیلگر نحوی: جمله با نقطه به پایان نرسیده است.  | حسن سیب خرید   |

«مثال: بررسی مراحل کامپایل.

```

float p,i,k ;
p = i + k * 60 ;

```

ابتدا تحلیلگر لغوی وارد عمل شده و تک تک کلمات را از هم جدا کرده (Tokenize) و بررسی میکند. سپس در جدول نمادها قرار میدهد:

Symbol Table

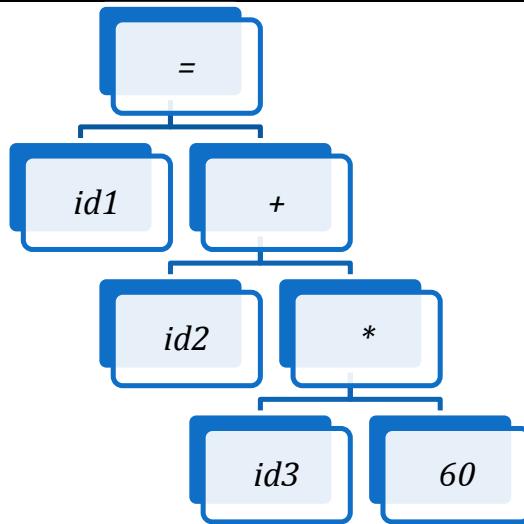
شناسه	اسم	نوع
id1	p	float
...	...	...

```

p = i + k * 60 ; → id1 = id2 + id3 * 60 ;

```

سپس تحلیلگر نحوی باید تشخیص دهد که این جمله متعلق به زبان است یا نه؟ برای این منظور درخت جمله را تشکیل میدهد:



در تحلیلگر معنایی باید عدد 60 (اعشاری) به 60.0 (صحیح) تبدیل شود:

*int to float (60) → 60.0*

سپس گُدد میانی زیر ایجاد میشود:

$T1 = \text{int to float} (60)$   
 $T2 = id3 * T1$   
 $T3 = id2 + T2$   
 $id1 = T3$

حال گُدد به دست آمده بهینه سازی میشود:

$T2 = id3 * 60.0$   
 $id1 = id2 + T2$

در انتها نیز گُدد به دست آمده به زبان ماشین تبدیل میشود:

*Movf id3 ,R2*  
*Mulf #60.0 ,R2*  
*Movf id2 ,R1*  
*Addf R2 ,R1*  
*Movf R1 ,id1*

## ۲ فصل دوم : بررسی انواع گرامرها و خواص عمومی زبانها

### ۲,۱ گرامر

گرامر شامل فرهنگ لغات زبان و قواعدی راجع به ساخته شدن جملات از این لغات است.

$$G = \langle N, T, S, P \rangle$$

N: Non-Terminals	(A,B,C,...)	غیر پایانه ها (حروف بزرگ)
T: Terminals	(a,b,c,...)	پایانه ها (حروف کوچک)
S: Start Symbol	{S   S ∈ N}	نماد شروع
P: Production	{α → β}	تولیدات، قواعد

### ۲,۲ Term

به هر کدام از اجزای گرامر یک Term گفته میشود.

### ۲,۳ پایانه ها (Terminal)

هایی از گرامر هستند که عیناً در جملات نهایی زبان وجود دارند و خودشان قابل مشتق شدن نیستند. Term

### ۲,۴ غیر پایانه ها (Non Terminal)

هایی از گرامر هستند که در جملات نهایی ظاهر نمیشوند و قابل مشتق شدن هستند. Term

### ۲,۵ نماد شروع (Start Symbol)

تمام جملات زبان از نماد شروع آغاز میشوند. نماد شروع جزو غیر پایانه هاست. معمولاً نماد شروع را با S نمایش میدهند.

### ۲,۶ الفبا (Alphabet)

به مجموعه حروف و علائم، الفبا میگویند. مانند زیر:

$$L = \{a, b, s \dots z\}$$

$$D = \{0,1, \dots 9\}$$

### ۲,۷ رشته (String)

دنباله محدودی از علائم مربوط به الفبا

Abc , 012 , ali , ...

## ۲,۸ پیشوند (Prefix)

اگر رشته ای داشته باشیم که از انتهای آن، تعداد صفر یا بیشتر علامت را حذف کنیم، باقیمانده پیشوند خواهد بود. تهی پیشوند تمام رشته هاست.

## ۲,۹ پسوند (Postfix)

اگر از ابتدای رشته تعداد صفر یا بیشتر علامت را حذف کنیم، آنچه بماند پسوند است.

## ۲,۱۰ زیر رشته (SubString)

اگر از یک رشته، یک پیشوند یا یک پسوند یا هر دو را حذف کنیم، آنچه بماند زیر رشته است.

## ۲,۱۱ اجتماع (Union)

مجموعه رشته های متعلق به دو زبان که یا متعلق به زبان اول یا متعلق به زبان دوم هستند.

$$M \cup N = \{S \mid S \in M \vee S \in N\}$$

## ۲,۱۲ الحاق (Concat)

$$M \cdot N = \{ST \mid S \in M \wedge T \in N\}$$

تعريف  $L^*$ : رشته هایی به طول صفر یا بیشتر از زبان  $L$

تعريف  $L^+$ : رشته هایی با طول یک یا بیشتر از زبان  $L$

تعريف  $L^n$ : رشته هایی با طول  $n$  از زبان  $L$

$$\begin{aligned} L^0 &= \{\epsilon\} \\ L^* &= L^0 \cup L^1 \cup L^2 \cup \dots \\ L^+ &= L^1 \cup L^2 \cup \dots \end{aligned}$$

## ۲,۱۳ گرامرهای معادل

اگر زبانی دو گرامر تولید کند که یکسان باشد، آن دو گرامر معادلند:

**G1:**

$$\begin{aligned} E &\rightarrow T + E \mid T - E \mid T \\ T &\rightarrow F * T \mid F / T \mid F \\ F &\rightarrow ID \mid D \mid (E) \\ ID &\rightarrow a \mid b \\ D &\rightarrow 0 \mid 1 \end{aligned}$$

**G2:**

$$\begin{aligned} E &\rightarrow E + T \mid T - E \mid T \\ T &\rightarrow T * F \mid F / T \mid F \\ F &\rightarrow ID \mid D \mid (E) \\ ID &\rightarrow a \mid b \\ D &\rightarrow 0 \mid 1 \end{aligned}$$

## ۲.۱۴ زبان (Language)

به مجموعه ای از رشته های تعریف شده بر روی یک الفبا زبان میگویند. مثلا زبان  $A$  روی الفبای  $L$  تعریف شده:

$$A = \{ali, abc, \dots\}$$

$$B = \{0,1,123, \dots\}$$

$$L(G): \{\alpha \mid S \rightarrow \alpha, \alpha \in T\}$$

زبان ترکیبی از پایانه ها و غیر پایانه هاست (ترکیبی از  $N$  و  $T$ ).

$$V = T \cup N$$

حروف یونانی مانند  $\alpha, \beta, \gamma, \dots$  مجموعه ای از پایانه ها و غیر پایانه هاست.

$$\begin{array}{ccc} \alpha & \rightarrow & \beta \\ LHS & & RHS \end{array}$$

«مثال:

$$\left\{ \begin{array}{l} 1: A \rightarrow aB \\ 2: A \rightarrow ef \end{array} \right\} \quad 1,2: A \rightarrow aB \mid ef$$

## ۲.۱۵ انواع گرامر

### قوی ترین

۱. گرامر نوع صفر (نامحدود)

۲. گرامر نوع یک (حساس به متن)

۳. گرامر نوع دو (مستقل از متن)

۴. گرامر نوع سه (منظمه)

### ضعیف ترین

#### قوانين گرامر نوع صفر:

اگر گرامر به صورت  $\alpha \rightarrow \beta$  باشد، تنها شرط ما  $\alpha \neq \lambda$  است.

$$\alpha \rightarrow \beta, \alpha \neq \lambda$$

#### قوانين گرامر نوع یک:

سمت چپ باید کوچکتر یا مساوی سمت راست باشد.

$$|\alpha| \leq |\beta|, \alpha \neq \lambda$$

#### قوانين گرامر نوع دو:

اندازه سمت چپ باید یک باشد.

$$|\alpha| = 1, \alpha \in N$$

#### قوانين گرامر نوع سه:

منظمه از چپ یا راست باشد.

$$|\beta| = 1, \beta \in T$$

$$|\beta| = 2, \begin{cases} \beta = aA \\ \beta = Aa \end{cases}$$

## ۲.۱۶ اشتقاق (Derivation)

دو نوع اشتقاق داریم:

۱. اشتقاق از چپ (LMD - Left Most Derivation) همیشه سمت چپ ترین غیر پایانه اشتقاق داده می شود.
۲. اشتقاق از راست (RMD - Right Most Derivation) همیشه سمت راست ترین غیر پایانه اشتقاق داده می شود.

» مثال: با توجه به قواعد داده شده، abc را تولید کنید.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow ab \\ B &\rightarrow c \end{aligned}$$

» پاسخ:

$$\text{LMD: } S \rightarrow AB \rightarrow abB \rightarrow abc$$

$$\text{RMD: } S \rightarrow AB \rightarrow Ac \rightarrow abc$$

## ۲.۱۷ فرم جمله ای

به هر کدام از فرم‌هایی که در مراحل اشتقاق ایجاد می‌شود، یک فرم جمله‌ای گویند. به عبارت دیگر رشته‌ای از پایانه‌ها و غیرپایانه‌ها که در مراحل اشتقاق ایجاد می‌شود.

$$\{\alpha \vdash \mid S \rightarrow \alpha, \alpha \in (N \cup T)\}$$

## ۲.۱۸ استنتاج

تبديل یک فرم جمله‌ای با استفاده از قواعد زبان به یک فرم جمله‌ای دیگر را استنتاج گویند.

## ۲.۱۹ انواع استنتاج

### ۱. استنتاج مستقیم ( $\Rightarrow$ )

اگر با استفاده مستقیم از یکی از قواعد زبان از یک فرم جمله‌ای به یک فرم جمله‌ای دیگر برسیم، به آن استنتاج مستقیم گویند.

### ۲. استنتاج نهایی ( $\Rightarrow^+$ )

اگر طی استفاده یکبار یا بیشتر از قواعد زبان از یک فرم جمله‌ای به فرم جمله‌ای دیگر برسیم، به آن استنتاج نهایی گویند.

### ۳. استنتاج کلی ( $\Rightarrow^*$ )

اگر طی استفاده صفر بار یا بیشتر از قواعد زبان از یک فرم جمله‌ای به یک فرم جمله‌ای دیگر برسیم، به آن استنتاج کلی گویند.

« مثال: نمونه استفاده از استنتاج در مثال قبل »

$$\begin{array}{ll} S \Rightarrow AB & \text{استنتاج مستقیم} \\ S \stackrel{+}{\Rightarrow} abB & \text{استنتاجنهایی} \end{array}$$

« مثال: آیا عدد ۲۳۱ متعلق به زبان زیر هست یا خیر؟ (به روش اشتقاد) »

$$\begin{array}{l} 1, 2 \quad <\text{Number}> \rightarrow <\text{Number}> <\text{Digit}> \mid <\text{Digit}> \\ 3, \dots, 12 \quad <\text{Digit}> \rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9 \end{array}$$

« پاسخ: »

LMD:

$$\begin{array}{l} <\text{Number}> \stackrel{1}{\Rightarrow} <\text{Number}> <\text{Digit}> \\ \quad \stackrel{1}{\Rightarrow} <\text{Number}> <\text{Digit}> <\text{Digit}> \\ \quad \stackrel{2}{\Rightarrow} <\text{Digit}> <\text{Digit}> <\text{Digit}> \\ \quad \stackrel{5}{\Rightarrow} \quad 2 \quad <\text{Digit}> <\text{Digit}> \\ \quad \stackrel{6}{\Rightarrow} \quad 2 \quad \quad 3 \quad <\text{Digit}> \\ \quad \stackrel{4}{\Rightarrow} \quad 2 \quad \quad 3 \quad \quad 1 \end{array}$$

RMD:

$$\begin{array}{l} <\text{Number}> \stackrel{1}{\Rightarrow} <\text{Number}> <\text{Digit}> \\ \quad \stackrel{4}{\Rightarrow} <\text{Number}> \quad 1 \\ \quad \stackrel{1}{\Rightarrow} <\text{Number}> <\text{Digit}> \quad 1 \\ \quad \stackrel{6}{\Rightarrow} <\text{Number}> \quad 3 \quad 1 \\ \quad \stackrel{2}{\Rightarrow} <\text{Digit}> \quad 3 \quad 1 \\ \quad \stackrel{5}{\Rightarrow} \quad 2 \quad 3 \quad 1 \end{array}$$

## ۲.۲۰ گرامر مبهم

اگر به ازای یک جمله واحد در یک گرامر بتوانیم دو بسط سمت چپ (LMD) متفاوت یا دو بسط سمت راست (RMD) متفاوت یا دو درخت نحوی (Pars Tree) متفاوت پیدا کنیم، آنگاه گرامر مبهم یا گنگ است.

« مثال: آیا عبارت  $id + id * id$  که توسط گرامر زیر تولید میشود، مبهم است یا خیر؟ »

$$\begin{array}{ll} 1 \quad E \rightarrow E + E & \\ 2 \quad E \rightarrow E * E & \\ 3 \quad E \rightarrow (E) & \quad 1, \dots, 4 \quad E \rightarrow E + E \mid E * E \mid (E) \mid id \\ 4 \quad E \rightarrow id & \end{array}$$

حالا میخواهیم  $id * id$  را تولید کنیم.

« پاسخ: گرامر بالا مبهم است، زیرا میتوان دو بسط LMD برای آن نوشت:

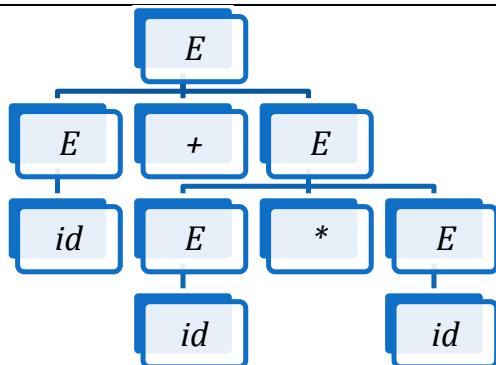
### LMD 1:

$$E \xrightarrow{1} E + E \xrightarrow{4} id + E \xrightarrow{2} id + E * E \xrightarrow{4} id + id * E \xrightarrow{4} id + id * id$$

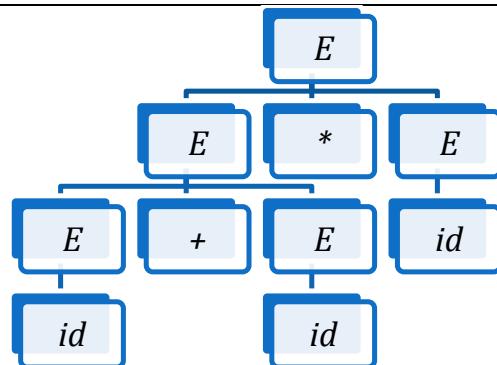
### LMD 2:

$$E \xrightarrow{2} E * E \xrightarrow{1} E + E * E \xrightarrow{4} id + E * E \xrightarrow{4} id + id * E \xrightarrow{4} id + id * id$$

درخت نحوی گرامر فوق به صورت زیر است:



**LMD 1**



**LMD 2**

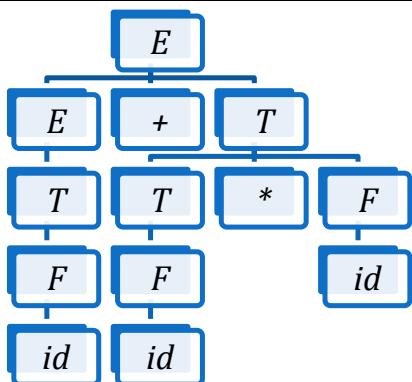
گرامر رفع ابهام شده (در این گرامر اولویت به ضرب داده شده است):

$$1,2 \quad E \rightarrow E + T \mid T$$

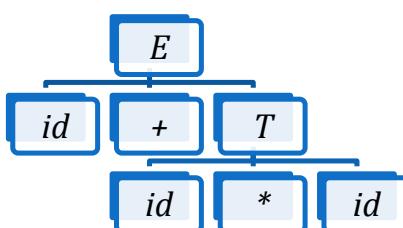
$$3,4 \quad T \rightarrow T * F \mid F$$

$$5,6 \quad F \rightarrow (E) \mid id$$

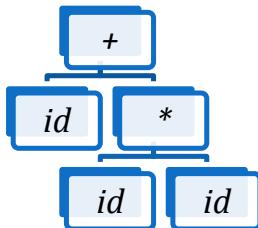
درخت خلاصه شده: (نحوه ایجاد درخت خلاصه شده در صفحه ۲۵)



درخت اوّلیه



$\Rightarrow$



درخت نهایی

## ۲.۲۱ گرامرهای مختصر و مفید

گرامری که سه شرط زیر را داشته باشد، مختصر و مفید خواهد بود:

- در آن قواعدی به فرم  $A \rightarrow A$  وجود نداشته باشد.

**نمایه**: تمام غیر پایانه ها فعل (ACTIVE) باشند.

۳. تمام غیر پایانه ها دسترس پذیر (Reachable) باشند.

- فعال بودن: غیر پایانه ها با استفاده از قواعد به دنباله ای از پایانه ها تبدیل شوند.

- دسترس پذیر بودن: اگر از نماد شروع آغاز کردیم، بتوانیم به غیر پایانه مورد نظر بررسیم.

\*\* نکته: ترتیب بررسی شرطها مهم است.

« مثال: گرامر معادل مختصر و مفید را برای **کلمه زیر** بفرمایید.

$$\begin{aligned} A &\rightarrow Ba \\ B &\rightarrow b \\ B &\rightarrow Cb \\ C &\rightarrow Cb \\ C &\rightarrow DdC \\ D &\rightarrow \epsilon \end{aligned}$$

**کلمه آن نهف را در**  $\rightarrow$  **D (سرمه نورشیه هف در کسرد.**

» پاسخ:

$$\begin{aligned} A &\rightarrow Ba \\ B &\rightarrow b \end{aligned}$$

$$\begin{aligned} S &\rightarrow aaa \\ S &\rightarrow Abaaa \\ A &\rightarrow Ab \\ A &\rightarrow aBa \\ B &\rightarrow aBa \\ B &\rightarrow AC \\ C &\rightarrow eb \\ C &\rightarrow b \end{aligned}$$

$$S \rightarrow aaa$$

« مثال: **A و B اگرتر نسخه**

» پاسخ:

## ۲.۲۲ عبارتهای منظم

$(0 1)^+ = 0, 1, 01, \dots$	هر عبارتی با صفر و یک به غیر از تهی
$(0 1)^* = \epsilon, 0, 1, 01, \dots$	هر عبارتی با صفر و یک حتی تهی
$11(0 1)^*011$	هر عبارتی که با شروع بود و پایان آن دو نفر ترکیبی از صفر و یک باشد
$(ab c)^+ = \dots, abc, cab, \dots, abcabab, \dots$	نمایش ارقام
$d = 0, 1, 2, \dots$	نمایش عدد صحیح
$d^+$	نمایش عدد اعشاری
$d^+.d^+$	نمایش عدد صحیح یا اعشاری
$d^+(\cdot.d^+ \epsilon) \Leftrightarrow d^+(\cdot.d^+)?$	نمایش عدد علامتدار (مثبت یا منفی)
$(+ -)d^+$	نمایش اعداد اعشاری با نماد علمی
$(+ - \epsilon)d^+ \Leftrightarrow (+ -)?d^+$	نمایش عدد علامتدار یا بدون علامت (مثبت یا منفی)
$(+ -)?d^+(\cdot.d^+)?(\cdot.d^+)?(\cdot.d^+)?$	

کم توان از حرف را کم می سرد

\*\* نکته: "?" به معنی وجود یا عدم وجود پرانتز است.

«مثال: فرض کنید بخواهیم برای یک شناسه گرامر بنویسیم و سپس برای آن دیاگرام گذر رسم کنیم. داریم:

$$d^+ = 0, 1, 2, \dots, 9$$

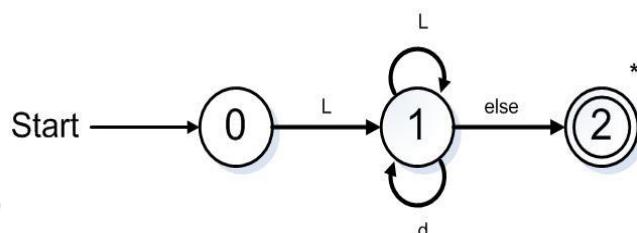
$$L(L|D)^*$$

$$1 \quad < ID > \rightarrow < L > | < ID > < L > | < ID > < D >$$

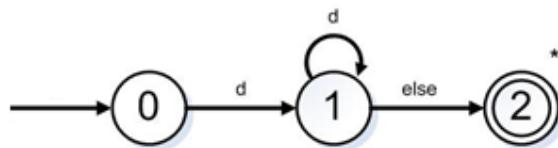
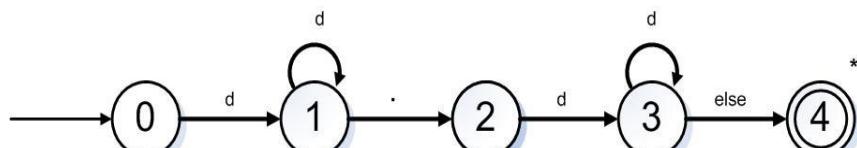
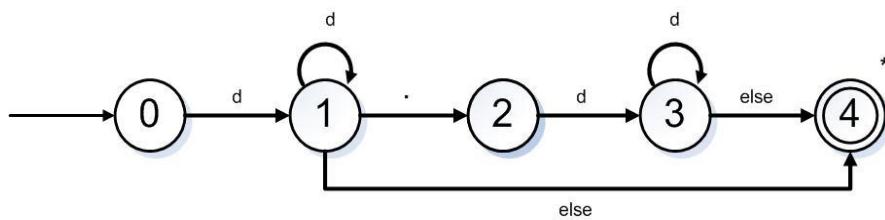
$$2..27 \quad < L > \rightarrow a | b | \dots | z$$

$$28..38 \quad < D > \rightarrow 0 | 1 | \dots | 9$$

دیاگرام گذر شناسه فوق به صورت زیر است:



کم توان از حرف را کم می سرد  
که های سه که  
با حرف را کم می سرد

« مثال: دیاگرام گذرهای  $d^+$  »« مثال: دیاگرام گذرهای  $d^+.d^+$  »« مثال: دیاگرام گذرهای  $d^+(.d^+)?$  »

## ۳ فصل سوم : تحلیلگر لغوی

مقدمة

۴

### ۳,۱ وظایف تحلیلگر لغوی (Scanner)

۱. تشخیص توضیحات (Comment) و حذف آنها.
۲. ساختن جدول نمادها (Symbol Table).
۳. مشخص کردن کلمات سازنده (Token) توسط جداکننده ها و تعیین نوع آنها. { مهمترین وظیفه }

### ۳,۲ جداکننده ها (Separator)

جداکننده ها دو دسته هستند:

۱. فضاهای خالی مانند Tab و Space وغیره.
۲. نمادهای خاص وابسته به زبان مانند پرانتز باز و بسته، عملگرهای منطقی و عملگرهای محاسباتی وغیره.

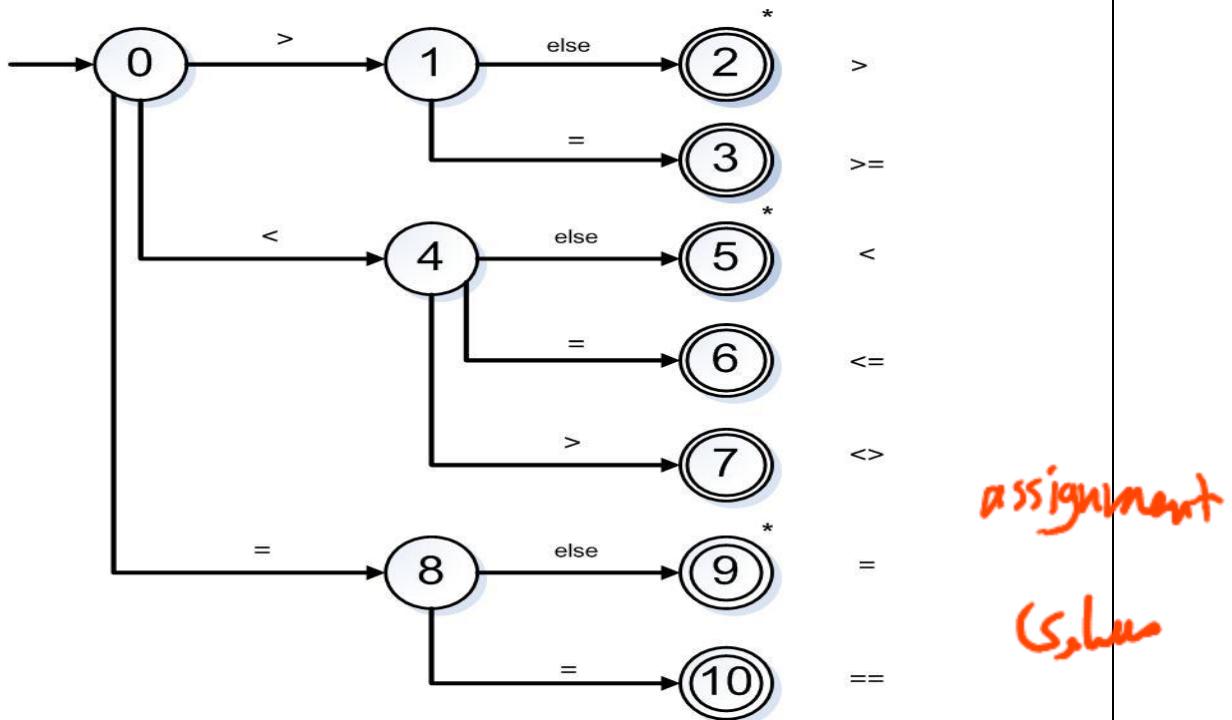
### ۳,۳ انواع موجود در زبان (انواع کلمات)

۱. شناسه ها
۲. جداکننده ها
۳. اعداد
۴. رشته ها
۵. کلمات کلیدی
۶. ثابت ها
۷. عملگرهای منطقی
۸. عملگرهای محاسباتی

### ۳.۴ شناسه (Identifier)

به هر جزء که بوسیله برنامه‌نویس تعریف می‌شود، شناسه می‌گویند. به ازاء هر شناسه در برنامه باید یک سری اطلاعات در هنگام ترجمه (کامپایل) نگهداری شود چون به آن اطلاعات در این فاز و فازهای بعدی نیاز داریم. به ازاء هر شناسه یک سطر در نظر گرفته می‌شود و اطلاعات آن شناسه از قبیل نام، نوع طول، اولین محل دیده شدن و غیره در آن ذخیره می‌شود.

« مثال: دیاگرام عملگرهای مقایسه ای »

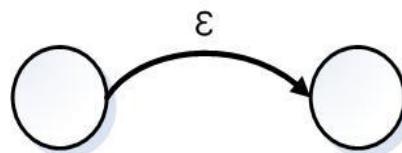


### ۳.۵ ماشینهای خودکار (Automata)

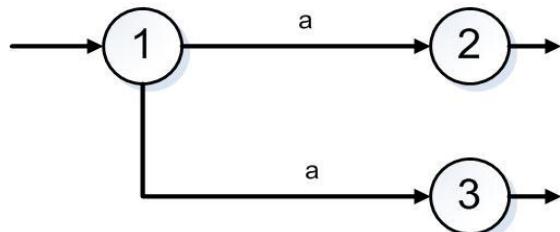
گراف‌های جهت داری هستند که برای پیاده سازی قواعد لغوی زبانها بسیار مناسب هستند. این ماشین‌ها دو نوع هستند: DFA (قطعی یا معین) و NFA (غیرقطعی یا نامعین).

دو ویژگی که DFA را از NFA جدا می‌کند:

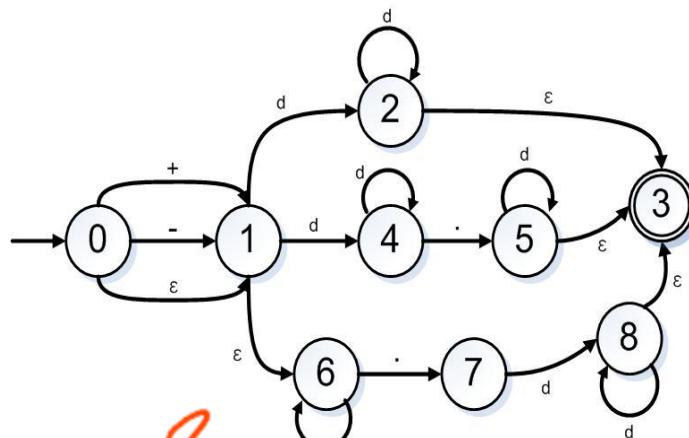
- در DFA برچسب تهی نداریم.



۲. به ازای یک وضعیت یا State دو لبه خارجی با برچسب یکسان نداریم.



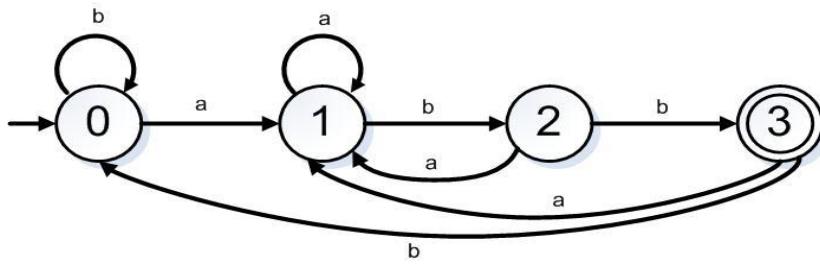
شکل زیر یک NFA است:



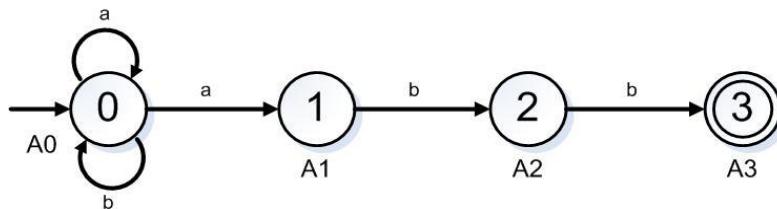
$(+|-|\epsilon) \left\{ \begin{array}{l} d^+ \\ d^+.d^* \\ d^*.d^+ \end{array} \right.$  *ترین اینها را کجا می‌خواهیم*  $\rightarrow NFA, DFA$  معرفی شود

کامپایلرها برای کامپایل از DFA استفاده می‌کنند. چون NFA به چند حالت می‌رود و کامپایلر گیج می‌شود.

« مثال: میخواهیم عبارتی تولید کنیم که به  $abb$  ختم شود.  
 « پاسخ: عبارت منظم مثال فوق برابر است با  $(a|b)^*abb$



شکل فوق یک DFA برای گرامر بالا است. اگر بخواهیم یک NFA برای عبارت بالا داشته باشیم، به صورت زیر عمل میکنیم:



### ۳.۶ الگوریتم تبدیل یک دیاگرام به گرامر معادلش

۱. به هر کدام از وضعیتهای داخل ماشین یک غیر پایانه نسبت میدهیم.

۲. اگر آنگاه  $A_i \rightarrow aA_j$  به گرامر اضافه میکنیم.

۳. اگر در شکل فوق به جای  $a$ ,  $\epsilon$  بود، آنگاه  $A_i \rightarrow A_j$  به گرامر اضافه میکنیم.

۴. به ازای وضعیتهای پایانی، غیر پایانه مربوطه را برابر تهی قرار میدهیم.  $A_n \rightarrow \epsilon$

۵. غیر پایانه مربوط به وضعیت شروع را به عنوان نماد شروع گرامر در نظر می‌گیریم.

« مثال: گرامر معادل دیاگرام مثال قبل:

$$\begin{array}{l}
 A_0 \rightarrow aA_0 \\
 A_0 \rightarrow bA_0 \\
 A_0 \rightarrow aA_1 \\
 A_1 \rightarrow bA_2 \\
 A_2 \rightarrow bA_3 \\
 A_3 \rightarrow \epsilon
 \end{array} \qquad 
 A_0 \rightarrow aabb \mid babbb \mid abbb$$

# ٤ فصل چهارم : روش‌های تحلیل نحوی مفرد (Parser)

## ٤.١ تحلیلگر نحوی (Parser)

اطلاعات مربوط به زبان و قواعد زبان به تحلیلگر نحوی داده می‌شود. ورودی آن Token های متعلق به زبان است. تحلیلگر نحوی جملات را با گرامر زبان مطابقت میدهد و بررسی می‌کند که قواعد زبان رعایت شده یا نه. برای این کار دو روش وجود دارد:

۱. اشتاقاق (Derivation)
۲. درخت نحوی (Syntax Tree)

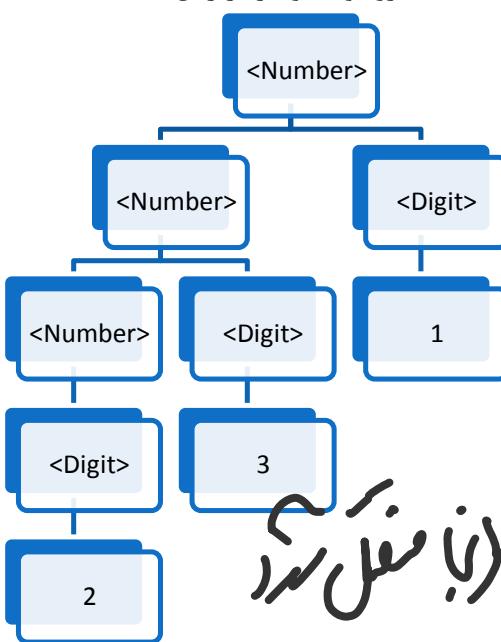
## ٤.٢ درخت نحوی (Syntax Tree)

ریشه رخت به عنوان نماد شروع است و با استفاده از قواعد زبان، سطرهای درخت را می‌سازیم. اگر بتوانیم درختی بسازیم که جملات نهایی در آن وجود داشته باشد، پس آن درخت به آن زبان تعلق دارد.

\*\* نکته: از روی درخت نمی‌توان تشخیص داد که کدام طرف را بسط دادیم. پس از روی درخت نمی‌توانیم بگوییم LMD است یا RMD.

\*\* نکته: ترتیب استفاده از قواعد هم از روی درخت مشخص نیست. درختی که در این مرحله تولید می‌شود برای تحلیلگر معنایی و بقیه مراحل نیز کاربرد دارد.

# ۱۴) مرعوط بصر



کارن فها با هم (با مغلوب)

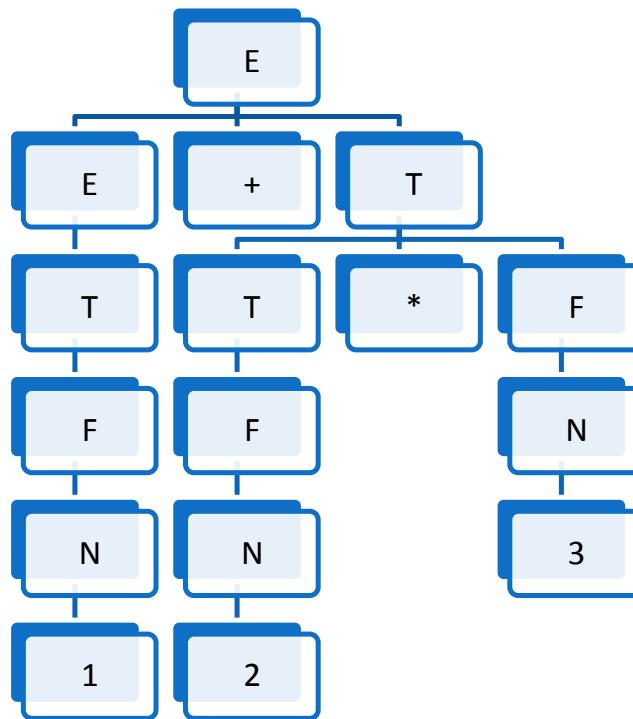
گرامر رفع ابهام شده:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow N \mid (E)$$

$$N \rightarrow 1 \mid 2 \mid 3$$



### ٤.٣ نحوه ایجاد درخت خلاصه شده

۱. اگر گره ای داشته باشیم که تنها یک فرزند داشت، آن گره را با فرزندش جایگزین میکنیم.
۲. در فازهای بعدی به غیر پایانه هم نیازی نداریم. پس غیرپایانه ها را با عملگرهای مناسب جایگزین میکنیم.

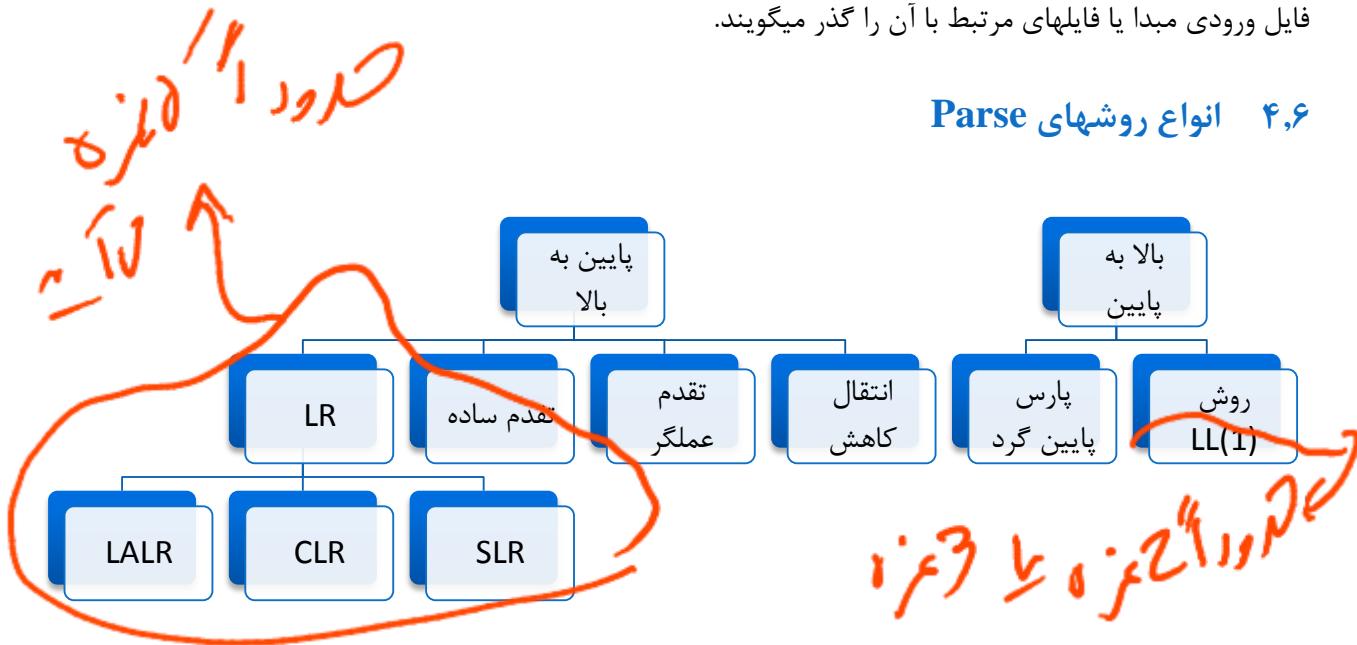
### ٤.٤ وظیفه پارس

وظیفه تحلیلگر نحوی این است که جملات را با قواعد مربوط به زبان مطابقت دهد.

### ٤.٥ گذر (Pass)

عبارت است از تعداد دفعاتی که فایل ورودی مبدأ یا فایلهای مرتبط با آن از اول تا آخر خوانده میشود. در واقع هر بار مرور فایل ورودی مبدأ یا فایلهای مرتبط با آن را گذر میگویند.

### ٤.٦ انواع روشهای Parse



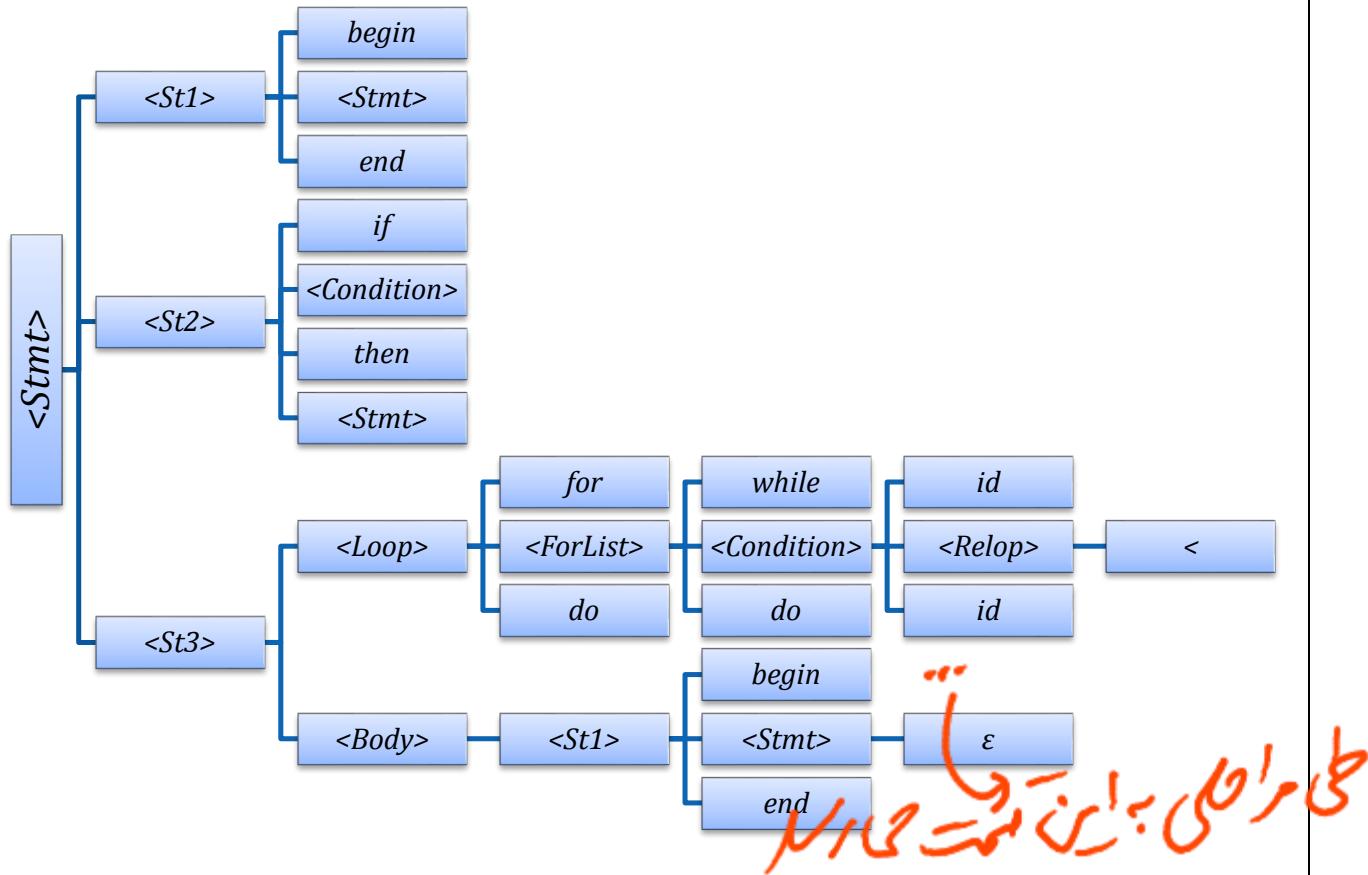
«مثال: با توجه به گرامر داده شده، بررسی کنید که آیا عبارت *While i < j Do Begin End* از لحاظ نحوی درست است یا خیر؟

- 1..4  $< Stmt > \rightarrow < St1 > | < St2 > | < St3 > | \epsilon$
- 5  $< St1 > \rightarrow begin < Stmt > end$
- 6  $< St2 > \rightarrow if < Condition > then < Stmt >$
- 7  $< St3 > \rightarrow < Loop > < Body >$
- 8  $< Loop > \rightarrow for < ForList > do$
- 9  $< Loop > \rightarrow while < Condition > do$
- 10  $< Body > \rightarrow < St1 >$
- 11  $< ForList > \rightarrow id := num \text{ to } num$
- 12  $< Condition > \rightarrow id < Relop > id$
- 13..15  $< Reloop > \rightarrow > | < | =$

برعطاً،  
دست آخدا  
دست کمال

« پاسخ: می خواهیم بررسی کنیم با توجه به گرامر فوق عبارت زیر از نظر نحوی صحیح است یا خیر.  
 $While\ i < j\ Do\ Begin\ End$

اگر پارسر موفق به ساختن درخت نحوی این عبارت شود، نتیجه می‌گیریم عبارت مورد نظر صحیح است.



با توجه به این که درخت ساخته شد، عبارت فوق از نظر نحوی صحیح است.

برای بررسی صحت این عبارت تمام شاخه ها را بسط دادیم و در صورت عدم مطابقت، Back Track انجام دادیم به عبارت دیگر از روش آزمون و خطا استفاده کردیم. این روش بسیار زمانگیر است. برای حل این مشکل از پارسرهای پیشگو استفاده می‌شود.

## ۴.۷ پارسرهای پیشگو

در پارسرهای پیشگو درخت به صورتی ساخته می‌شود که نیازی به Back Track نداشته باشد و اگر جایی به مشکل برمی‌گردد، نتیجه می‌گیرد که جمله ورودی از لحاظ نحوی نادرست بوده است. پس درخت را به صورتی می‌سازد که حتماً درست باشد. در روش قبل (آزمون و خطا) اول درخت را می‌ساخت و بعد به ورودی نگاه می‌کرد. در این روش ابتدا به

ورودی نگاه می‌کند و بر اساس آن درخت را می‌سازد. به نمادی که درخت از روی **نماد پیشگویی مخفت مجموعه زیرگرام (Look Ahead)** می‌گویند و به صورت مخفف با **L.A.** نشان میدهد.



## تابع ۴.۸ First

اولین پایانه‌ای است که غیر پایانه  $N$  می‌تواند ببیند.

$$\text{First}(\alpha) = \left\{ a \mid \alpha \xrightarrow{*} a\beta \right\}, \quad \alpha:$$

\*\* نکته: خروجی تابع First مجموعه‌ای از پایانه‌هاست.

\*\* نکته: در تابع First هیچگاه  $\$$  نداریم.

## مراحل به دست آوردن (x) ۴.۹

۱. اگر  $x$  یک پایانه باشد، آنگاه  $\{\}$ .
۲. اگر قاعده‌ای به فرم  $\epsilon \rightarrow x$  داشته باشیم، آنگاه  $\epsilon$  را به مجموعه  $\text{First}(x)$  اضافه می‌کنیم.
۳. اگر قاعده‌ای به فرم  $y_1 y_2 \dots y_k \rightarrow x$  داشته باشیم، آنگاه  $\{\epsilon\}$  را به  $\text{First}(y_1) - \{\epsilon\}$  اضافه می‌کنیم.
۴. اگر  $y_1$  طی صفر مرحله یا بیشتر به  $\epsilon$  برسد، آنگاه  $\{\epsilon\}$  را نیز به  $\text{First}(y_2) - \{\epsilon\}$  اضافه می‌کنیم و همین روند را برای بقیه ادامه می‌دهیم.
۵. اگر  $y_{k-1}$  طی صفر مرحله یا بیشتر به  $\epsilon$  برسد، آنگاه  $\text{First}(y_k)$  را به  $\text{First}(x)$  اضافه می‌کنیم.

« مثال: با توجه به گرامر زیر  $\text{First}(A)$  را بباید:

$$A \rightarrow BC Da$$

$$B \rightarrow b \mid \epsilon$$

$$C \rightarrow e \mid \epsilon$$

$$D \rightarrow f \mid \epsilon$$

» پاسخ:

$$\text{First}(A) = \text{First}(BC Da) = \text{First}(B) - \{\epsilon\} + \text{First}(C) - \{\epsilon\} + \text{First}(D) - \{\epsilon\} + \{a\}$$

$$\text{First}(B) = \{b, \epsilon\}$$

$$\text{First}(C) = \{e, \epsilon\}$$

$$\text{First}(D) = \{f, \epsilon\}$$

$$\Rightarrow \text{First}(A) = \{b, e, f, a\}$$

\*\* تذکر: اگر در  $a, A \rightarrow BC Da$  نبود، به رشتہ حاصل  $\epsilon$  اضافه می‌کردیم.

## تابع ۴.۱۰ Follow

پایانه‌ای که بعد از غیرپایانه می‌تواند دیده شود.

$$\text{Follow}(A) = \left\{ b \mid S \xrightarrow{*} \alpha Ab\beta \right\}$$

\*\* نکته: خروجی تابع Follow مجموعه ای از پایانه هاست.

\*\* نکته: در تابع Follow هیچگاه  $\epsilon$  نداریم.

#### ٤.١١ مراحل یافتن (A) Follow

۱. اگر A علامت شروع گرامر باشد، آنگاه علامت  $\{\$\}$  را به Follow(A) اضافه میکنیم.
۲. اگر قاعده ای به فرم  $\alpha A \beta \rightarrow X$  داشته باشیم، در آن صورت  $\{\epsilon\}$  را به Follow(A) اضافه میکنیم.
۳. اگر قاعده ای به فرم  $\alpha A \rightarrow X$  داشته باشیم یا قاعده ای به فرم  $\alpha A \beta \rightarrow X$  داشته باشیم که در آن  $\epsilon \Rightarrow^* \beta$  آنگاه  $\{\epsilon\}$  را به مجموعه Follow(X) اضافه میکنیم.

« مثال: تمام غیر پایانه های گرامر زیر را بیابید.

0	$\hat{A} \rightarrow A\$$
1	$A \rightarrow BCDA$
2,3,4	$B \rightarrow CD \mid b \mid \epsilon$
5,6	$C \rightarrow e \mid \epsilon$
7,8	$D \rightarrow f \mid \epsilon$



» پاسخ:

$$Follow(A) = \{\$\}$$

$$Follow(B) = First(C) + First(D) + \{a\}$$

$$Follow(C) = First(D) + \{a\} + Follow(B) = \{f\} + \{a\} + \{e, f, a\} = \{f, a, e\}$$

$$Follow(D) = \{a\} + Follow(B) = \{a, e, f\}$$

گرامر این مرتضی

ارائه دهنده آزادی

وی سود (B)

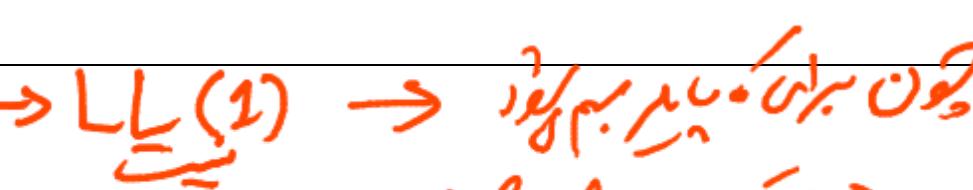
#### ٤.١٢ پارس به روشن (LL(1))

- اول (L<sub>Left</sub>): یعنی عبارت ورودی از چپ به راست بررسی میشود.
- دوم (L<sub>Middle</sub>): اشتاقاق چپ ترین
- (1): یعنی نماد پیشگویی ما یک عنصر است (تک عضوی).

البته روش LL(1) قابل تعمیم به LL(k) تا LL(3) نیز هست، ولی کاربرد کمتری دارد.

« مثال: گرامر زیر LL(1) نیست ولی LL(2) هست.

$$\begin{aligned} A &\rightarrow abc \\ B &\rightarrow ade \end{aligned}$$



## ۴.۱۳ جدول پارس (LL(1))

در ستونهای جدول، پایانه‌ها قرار می‌گیرند به اضافه \$ و در سطرهای جدول، غیر پایانه‌ها قرار می‌گیرند.

اندازه جدول : تعداد غیر پایانه‌ها \* (تعداد پایانه‌ها + 1)

$$\text{Table Size} = |N| * (|T| + 1)$$

	\$ + پایانه‌ها				
.					
بازگشتن					
.					
بازگشتن					
.					

## ۴.۱۴ نحوه تشکیل جدول پارس (LL(1))

- برای کلیه قواعد غیر تهی به فرم  $A \rightarrow \alpha$ ، شماره این قاعده را مقابل غیرپایانه A و زیر عناصر مجموعه First( $\alpha$ ) مینویسیم.
- اگر  $\alpha$ ، طی صفر مرحله یا بیشتر به ۶ برود، شماره قاعده مربوطه را مقابل غیرپایانه A و زیر عناصر مجموعه Follow(A) مینویسیم.

« مثال: جدول پارس (LL(1)) برای مثال Stmt بکشید. »

$$0 \quad < ST > \rightarrow < Stmt > \$$$

خط . را به ابتدای مثال اضافه می‌کنیم.

« پاسخ: First را برای همه غیر پایانه‌ها و Follow را برای غیر پایانه‌هایی که به تهی ختم می‌شوند، محاسبه می‌کنیم:

$$\begin{aligned} First(Stmt) &= First(St1) + First(St2) + First(St3) + \{\epsilon\} \\ First(St1) &= \{begin\} = First(Body) \\ First(St2) &= \{if\} \\ First(St3) &= First(Loop) = \{for, while\} \\ First(ForList) &= \{id\} \\ First(Condition) &= \{id\} \\ First(Relop) &= \{>, <, =\} \\ \\ Follow(Stmt) &= \{\$, end\} \end{aligned}$$

```

1..4 < Stmt > → < St1 > | < St2 > | < St3 > | ε
5   < St1 > → begin < Stmt > end
6   < St2 > → if < Condition > then < Stmt >
7   < St3 > → < Loop >< Body >
8   < Loop > → for < ForList > do
9   < Loop > → while < Condition > do
10  < Body > → < St1 >
11  < ForList > → id := num to num
12  < Condition > → id < Relop > id
13..15 < Reloop > → > | < | =

```

	<i>begin</i>	<i>end</i>	<i>if</i>	<i>then</i>	<i>for</i>	<i>do</i>	<i>while</i>	<i>id</i>	<i>num</i>	<i>to</i>	$:=$	$>$	$<$	$=$	$\$$
<i>Stmt</i>	1	<u>4</u>	2		3		3								<u>4</u>
<i>St1</i>		5													
<i>St2</i>			6												
<i>St3</i>					7		7								
<i>Loop</i>					8		9								
<i>Body</i>	10														
<i>ForList</i>									11						
<i>Condition</i>									12						
<i>Relop</i>												13	14	15	

« مثال: برای گرامر زیر جدول پارس تشکیل دهید.

- 0  $E'' \rightarrow E\$$
- 1  $E \rightarrow TE'$
- 2  $E' \rightarrow +TE'$
- 3  $E' \rightarrow \epsilon$
- 4  $T \rightarrow FT'$
- 5  $T' \rightarrow *FT'$
- 6  $T' \rightarrow \epsilon$
- 7  $F \rightarrow (E)$
- 8  $F \rightarrow id$

« پاسخ: ابتدا First همه غیر پایانه ها را مشخص میکنیم:

$$First(E) = First(T) = First(F) = \{( , id\}$$

$$First(E') = \{ +, \epsilon \} \blacktriangleleft$$

$$First(T') = \{ *, \epsilon \} \blacktriangleleft$$

اِنْ تَوَكِّدْ بِعْرَرَنْ سَادَةَ الْفَخْرِم

برای کشیدن جدول پارس باید Follow غیر پایانه هایی که آنها به  $\epsilon$  ختم میشوند، را هم بدست بیاوریم:

$$Follow(E') = Follow(E) = \{ \$, ) \}$$

$$Follow(T') = Follow(T) = First(E') + Follow(E) + Follow(E') = \{ +, \$, ) \}$$

	<i>id</i>	+	*	(	)	\$
<b><i>E</i></b>	1			1		
<b><i>E'</i></b>		2			<b>3</b>	<b>3</b>
<b><i>T</i></b>	4			4		
<b><i>T'</i></b>		<b>6</b>	5		<b>6</b>	<b>6</b>
<b><i>F</i></b>	8			7		

## ۴.۱۵ پارس با استفاده از روش (LL(1))

نکته: پارسر در هر مرحله به Token بالای انباره (Stack) و عنصر اول عبارت یا Token جاری نگاه می‌کند.

- مرحله صفر (شرایط آغازین)

یک \$ به انتهای رشته ورودی اضافه می‌کنیم و یک S\$ به صورت معکوس وارد انباره می‌کنیم. (S یعنی نماد شروع)

- مرحله اول

اگر  $X = L.A. = \$$  بود، عمل پارس خاتمه می‌باید.

- مرحله دوم

اگر  $X = L.A. \neq \$$  بود، آنگاه Scanner فراخوانی می‌شود که Token بعد را پیدا کند و X از بالای انباره حذف می‌شود. اگر X پایانه باشد، ولی مخالف  $L.A.$  باشد، یک خطای نحوی رخ داده است. همیشه یک اشتتقاق به صورت برعکس در انباره LL(1) قرار می‌گیرد که عنصر روی انباره که عنصر سمت چپ عبارت است، بالای انباره قرار خواهد گرفت.

- مرحله سوم

اگر X غیر پایانه باشد، پارسر به خانه  $PT_{X,L.A.}$  مراجعه می‌کند که دو حالت ممکن است پیش بیايد:

1. حالت اول: اگر  $\alpha \rightarrow \alpha$  باشد، X را از بالای انباره حذف و به جای آن  $\alpha$  را به صورت معکوس وارد انباره می‌کنیم.

2. حالت دوم: اگر  $PT_{X,L.A.}$  خالی باشد، یک خطای نحوی رخ داده است.

« مثال: آیا عبارت  $id + id * id$  با پارسرا (1) از لحاظ نحوی درست است؟

**Stack**

« پاسخ: اگر پذیرش رخ دهد، یعنی عبارت فوق با این گرامر همخوانی دارد و از نظر نحوی درست است. اگر به طور مثال T خانه خالی ببیند، یعنی این عبارت جزء گرامر نبوده است.

	انباره	ورودی	عمل انجام شده
1	\$ E	$id + id * id $$	بسط با قاعده ۱
2	\$ E'T	$id + id * id $$	بسط با قاعده ۴
3	\$ E'T'F	$id + id * id $$	بسط با قاعده ۸
4	\$ E'T'id	$id + id * id $$	نظیر شدن id
5	\$ E'T'	$+id * id $$	بسط با قاعده ۶
6	\$ E'	$+id * id $$	بسط با قاعده ۲
7	\$ E' T +	$+id * id $$	نظیر شدن +
8	\$ E' T	$id * id $$	بسط با قاعده ۴
9	\$ E' T'F	$id * id $$	بسط با قاعده ۸
10	\$ E' T'id	$id * id $$	نظیر شدن id
11	\$ E' T'	$* id $$	بسط با قاعده ۵
12	\$ E' T' F *	$* id $$	نظیر شدن *
13	\$ E' T' F	$id $$	بسط با قاعده ۸
14	\$ E' T'id	$id $$	نظیر شدن id
15	\$ E' T'	$$$	بسط با قاعده ۶
16	\$ E'	$$$	بسط با قاعده ۳
17	\$	$$$	پذیرش

شرط قبول جمله ورودی این است که در نهایت در ورودی فقط علامت \$ و در انباره هم فقط علامت \$ وجود داشته باشد.

« مثال: (1) LL بودن گرامر زیر را بررسی کنید.

- 0       $E' \rightarrow E\$$
- 1,2     $E \rightarrow aAbEF \mid e$
- 3,4     $F \rightarrow fE \mid \epsilon$
- 5       $A \rightarrow g$

$$First(E) = \{a, e\}$$

$$First(F) = \{f, \epsilon\} \blacktriangleleft$$

$$First(A) = \{g\}$$

$$\text{Follow}(E) = \{\$\} + \text{First}(F) + \text{Follow}(E) + \text{Follow}(F)$$

$$\text{Follow}(F) = \text{Follow}(E)$$

$$\text{Follow}(A) = \{b\}$$

$$\rightarrow \text{Follow}(E) = \{\$, f\} = \text{Follow}(F)$$

نحوی  $\Rightarrow$  کاپلدر لج کا گور

لذ آڑ اصر منو  
لکا نسے

	<i>a</i>	<i>b</i>	<i>e</i>	<i>f</i>	<i>g</i>	\$
<i>E</i>	1			2		
<i>F</i>					3 4	4
<i>A</i>					5	

با توجه به جدول این گرامر (1) LL نیست (یعنی بار وس ای (1) LL قابل بار وس کردن نیست). چون در یک خانه دو حالت وجود دارد.

بررسی کردن

#### ۴.۱۶ شرایط (1) LL بودن یک گرامر

دو روش وجود دارد:

۱. روش اول: کشیدن جدول پارس (1) LL

اگر در خانه های جدول هیچ تداخلی وجود نداشت، گرامر (1) LL است و در غیر این صورت (1) LL نیست.

\*\* نکته: تداخل یعنی در یک خانه بیش از یک عدد باشد.

۲. روش دوم: گرامری (1) LL است که سه شرط زیر را داشته باشد:

- شرط اول: اگر سمت چپ قواعد یکسان بود (مثلا  $A \rightarrow \alpha | \beta | \gamma | \dots$ ) First هیچ کدام از آنها نباید اشتراک داشته باشند، یا به عبارت دیگر اشتراک آنها باید تهی باشد:

$$\text{First}(\alpha) \cap \text{First}(\beta) \cap \dots = \emptyset$$

« مثال: آیا گرامر زیر (1) LL است؟ »

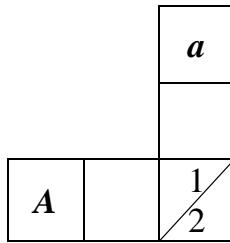
$$A \rightarrow \underbrace{aB}_{\alpha} \mid \underbrace{aC}_{\beta}$$

« پاسخ:

$$\text{First}(\alpha) \cap \text{First}(\beta) = \{a\}$$

تهی نیست

پس (1) LL هم نیست. تداخل دارد.



« مثال: آیا گرامر زیر LL(2) است؟

$$A \rightarrow abA \mid adC$$

« پاسخ: LL(2) هست ولی LL(1) نیست.

\*\* نکته: گاهی اوقات ممکن است قوانین طوری طراحی شده باشند که به صورت غیر مستقیم First مشترک داشته باشند.

« مثال: آیا گرامر زیر LL(1) است؟

$$A \rightarrow B \mid C$$

$$B \rightarrow ab$$

$$C \rightarrow ad$$

« پاسخ: LL(1) نیست، چون First مشترک دارد.

- شرط دوم: قواعدی که سمت چپ یکسانی دارند، اگر غیرپایانه مورد نظر سمت چپ به تهی برود، علاوه بر نداشتن اشتراک در هایشان، اشتراک First و Follow آنها هم باید تهی باشد.

$$A \rightarrow \alpha \mid \beta \mid \gamma \mid \dots \mid \varepsilon$$

$$\{First(\alpha) \cap First(\beta) \cap \dots = \emptyset$$

$$\} First(A) \cap Follow(A) = \emptyset$$

- شرط سوم: قواعدی که سمت چپ یکسانی دارند، باید دو قاعده به صورت همزمان به تهی بروند.

$$A \rightarrow \alpha \mid \beta \mid \gamma \mid \dots \mid \varepsilon$$

$$\alpha \rightarrow \varepsilon$$

$$\beta \rightarrow \varepsilon$$

\*\* یادآوری: حروف یونانی رشته‌اند، یعنی میتوانند ترکیبی از پایانه‌ها و غیر پایانه‌ها باشند.

« مثال: آیا گرامر زیر LL(1) است؟ (به هر دو روش بررسی کنید)

$$0 \quad S'' \rightarrow S\$$$

$$1,2 \quad S \rightarrow iEtSS' \mid a$$

$$3,4 \quad S' \rightarrow eS \mid \varepsilon$$

$$5 \quad E \rightarrow b$$

	First	Follow
$S$	$\{i, a\}$	$\{\$\}, e\} = \{\$\} + First(s') + Follow(S) + Follow(S')$
$S'$	$\{e, \epsilon\}$	$\{\$\}, e\}$
$E$	$\{b\}$	$\{t\}$

	$i$	$t$	$a$	$e$	$b$	\$
$S$	1		2			
$S'$				3 4		4
$E$					5	

LL(1) نیست زیرا بین First و Follow اشتراک وجود دارد.

#### ۴.۱۷ فاکتورگیری

با استفاده از روش فاکتورگیری میتوان گرامر غیر LL(1) که مشترک First دارند را به گرامر LL(1) تبدیل کرد.

» مثال: با استفاده از روش فاکتورگیری، گرامر زیر را به LL(1) تبدیل کنید.

$$A \rightarrow aB \mid aC \mid d$$

$$B \rightarrow g \mid l$$

$$C \rightarrow e$$

» پاسخ:

$$A \rightarrow aE \mid d$$

$$E \rightarrow B \mid C$$

$$B \rightarrow g \mid l$$

$$C \rightarrow e$$

به طور کلی در روش فاکتورگیری برای رشته‌های دلخواه  $\alpha$  و  $\beta$  داریم:

$$A \rightarrow a\alpha \mid a\beta$$

و پس از فاکتورگیری داریم:

$$A \rightarrow aB$$

$$B \rightarrow \alpha \mid \beta$$

« مثال: پارسرا برای گرامر زیر تشکیل دهید.

```

1..5 < Stmt > → < St1 > | < St2 > | < St3 > | < St4 > | < St5 >
6   < St1 > → begin < Stmt > end
7   < St2 > → if < Condition > then
8   < St3 > → < Loop > < Body >
9   < St4 > → id ::= < Stmt >
10  < St5 > → id(< Parameters – List >)
11  < Condition > → id < Relop > id
12  < Loop > → for < ForList > do
13  < Loop > → while < Condition > do
14  < Body > → < St1 >
15, 16 < Parameters – List > → id, (< Parameters – List >) | id
17..19 < Relop > → > | < | =
20   < ForList > → id ::= num to num

```

» پاسخ:

قاعده ۹ ( $< St4 >$ ) و قاعده ۱۰ ( $< St5 >$ ) دارای First مشترک id هستند.

همچنین قاعده ۱۵ و ۱۶ هم دارای First مشترک id هستند.

```

9   < St4 > → id < B >
10  < B > → ::= < Stmt > | (< Parameters – List >)

15  < Parameters – List > → id < E >
16  < E > → , (< Parameters – List >) | ε

```

#### ۴.۱۸ چپگردی

اگر در گرامر قاعده‌ای داشته باشیم که در آن غیرپایانه سمت چپ به عنوان اولین Term سمت راست ظاهر شده باشد، گرامر را چپگرد می‌گوییم.

\*\* نکته: اگر در گرامر چپگردی داشته باشیم، گرامر LL(1) نیست.

» مثال:

$$\begin{aligned} A &\rightarrow Ab \\ A &\rightarrow Ab \rightarrow Abb \end{aligned}$$

» مثال:

$$\begin{aligned} E &\rightarrow Ea \mid b \\ E &\rightarrow Ea \rightarrow Eaa \rightarrow Eaaa \dots a \mid baaa \dots a \\ First(Ea) \cap First(b) &= \{b\} \end{aligned}$$

## ۴.۱۹ حذف چپ‌گردی

$$A \rightarrow \underbrace{A\alpha_1 | A\alpha_2 | \dots | A\alpha_n}_{\text{چپگرد}} \mid \underbrace{\beta_1 | \beta_2 | \dots | \beta_m}_{\text{معمولی}}$$

با حذف چپ‌گردی داریم:

$$\begin{cases} A \rightarrow \beta_1 A' | \beta_2 A' | \dots | \beta_m A' \\ A' \rightarrow \alpha_1 A' | \alpha_2 A' | \dots | \alpha_n A' | \varepsilon \end{cases}$$

« مثال: چپ‌گردی گرامر زیر را در صورت وجود حذف کنید.

$$\begin{aligned} E &\rightarrow Ed \mid EA \mid g \\ A &\rightarrow b \mid c \end{aligned}$$

« پاسخ:

$$\begin{aligned} E &\rightarrow gE' \\ E' &\rightarrow dE' \mid AE' \mid \varepsilon \\ A &\rightarrow b \mid c \end{aligned}$$

« مثال:

$$E \rightarrow E + E \mid E * E \mid (E) \mid id$$

« پاسخ:

$$\begin{aligned} E &\rightarrow (E) E' \mid id E' \\ E' &\rightarrow +EE' \mid *EE' \mid \varepsilon \end{aligned}$$

## کوآل غزه برآهان

## ۴.۲۰ چپ‌گردی غیر صریح

گرامر زیر را در نظر بگیرید:

$$\begin{aligned} E &\rightarrow Ab \mid c \\ A &\rightarrow Ed \mid g \end{aligned}$$

ابتدا آن را به چپ‌گردی صریح تبدیل می‌کنیم:

$$E \rightarrow Edb \mid gb \mid c$$

پس از حذف چپ‌گردی:

$$\begin{aligned} E &\rightarrow gbE' \mid cE' \\ E' &\rightarrow dbE' \mid \varepsilon \end{aligned}$$

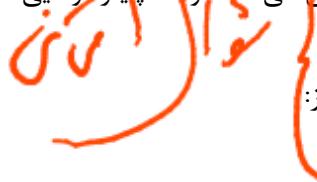
# ← کهفیل فصل ۴: روش‌های اصلاح خطای نحوی

## ۵.۱ انواع خطا

چهار نوع خطا داریم:

۱. خطای لغوی: یعنی در جایی قوانین لغوی زبان را رعایت نکنیم.
۲. خطای نحوی: به دلیل رعایت نکردن قواعد نحوی (دستور زبان) به وجود می‌آید. (Syntax Error)
۳. خطای معنایی: مانند نسبت دادن یک عدد اعشاری به یک متغیر صحیح یا درخواست دسترسی به خانه پانزدهم یک آرایه دهتایی در صورتی که قواعد دستوری رعایت شده است.
۴. خطای منطقی: یعنی برنامه از لحاظ منطقی اشکال داشته باشد. به عبارت دیگر الگوریتم اشتباه پیاده‌سازی شده باشد. کامپایلر نمی‌تواند این نوع خطا را تشخیص دهد و تشخیص و اصلاح آن به عهده برنامه‌نویس است.

سه خطای اول را خطاهای اصلی می‌نامند و کامپایلر توانایی تشخیص آنها را دارد.



روش‌های اصلاح خطا عبارتند از:

- (۱) Panic Mode
- (۲) Phrase Level
- (۳) Error Production
- (۴) Global Correction

دلیل اصلاح خطا:

به خاطر اینکه کامپایلر در هنگام کامپایل کردن برنامه به محض برخورد با خطا متوقف نشود و تا انتهای گُد را کامپایل کند و در پایان، لیست خطاهای را نمایش دهد.

## ۵.۲ روش Panic Mode

پایه این روش بر اساس حذف است. یعنی هرجا به خطا برخورد کرد، از ورودی حذف می‌کند. آنقدر حذف می‌کند تا به عنصری از مجموعه هماهنگ ساز برسد. عناصر مجموعه هماهنگ ساز به نحوی نشان دهنده مقاطعی از برنامه هستند. سادگی این روش در پیاده سازی آن است و هیچ وقت در حلقه نامحدود نمی‌افتد، چون حداکثر این است که همه را حذف می‌کند.

## ۵.۳ روش Phrase Level

با توجه به محلی که خطا رخ داده و چیزی که در ورودی دیده می‌شود و چیزی که انتظار دارد ببیند، یک حدس میزند و بر اساس آن حدس، تغییری در برنامه ایجاد می‌کنند تا خطا را رفع کند. اگر حدس درست باشد، پیغام مناسب صادر می‌شود. پس خطا رفع شده و عبور می‌کند.

$x = a + b * c$	عبارة صحيح مورد نظر
$x = a - b * c$	عبارة ورودی اشتباه
$x = ab * c$	حدس اشتباه

## «مثال:

\*\*\* نکته: حدس اشتباه باعث میشود که نتیجه محاسبه اشتباه در عبارات دیگر هم تاثیر بگذارد و نتیجه آنها نیز اشتباه حساب شود. به این نوع خطأ، خطای آبشاری گفته میشود.

## روش ۵،۴ Error Production

یک بررسی آماری روی خطاهایی که در یک زبان خاص، برنامه‌نویسان بیشتر مرتكب می‌شوند، انجام میدهد. سپس خطاهای را به شکل قواعد همراه با علامتی که از بقیه قواعد تفکیک شود، به زبان اضافه می‌کنند. اگر برنامه‌ای از یکی از این قواعد استفاده کند، یعنی برنامه از خطا استفاده کرده و پیغامی میدهد که مناسب آن خطا باشد.

۱. پیغامی که میدهد مناسب است، چون خطرا میشناسد.
  ۲. به سادگی میتواند از روی خطاهای عبور کند، چون قاعدهاش در گرامر وجود دارد.

» مثال: مثلاً عبارت  $a+b$  به صورت  $ab$  وارد شده و این خطأ از قبل پیش بینی شده است.

$$E \rightarrow F + T$$

•

$\boxtimes E \rightarrow FT$

## روش ۵,۵ Global Correction

این روش تصحیح عمومی است. روش‌های قبلی به صورت محلی به خطا نگاه میکرند. در این روش، همه برنامه را به صورت کلی بررسی میکنیم. سپس حداقل تعداد تغییرات و اصلاحات که در برنامه باید انجام شود را اعمال میکنیم تا کل برنامه درست شود. این روش یک روش تئوری و آزمایشگاهی است و جنبه عملی پیدا نکرده است.

۱. پیغامی که به کاربر میدهد باید مناسب و درست باشد و به کاربر در اصلاح خطأ کمک کند.
  ۲. باید سرعت بالایی داشته باشد.
  ۳. حجم پردازش‌ها نباید زیاد باشد تا برنامه‌های درست زیاد معطل و گُند نشوند.

\*\*\* نکته: معمولاً پارسراهی بالا به پایین از دو روش اول استفاده میشود.

## ۵.۶ اصلاح خطای نحوی با روش Panic Mode در پارس‌های LL(1)

در روش Panic Mode به ازای هر غیر پایانه مجموعه هماهنگ ساز را مشخص می‌کنیم که همان Follow غیرپایانه مورد نظر است. البته در صورتیکه خانه مورد نظر خالی باشد.

\* اگر پایانه روی انباره با ورودی تطبیق نداشت، پایانه روی انباره را حذف می‌کنیم. اما اگر عنصر روی انباره غیرپایانه بود، سه حالت دارد:

۱. یا با یک شماره قاعده پر شده که با همان شماره قاعده بسط میدهیم.

۲. یا آن خانه خالی است که از ورودی حذف می‌کنیم.

۳. یا خانه مورد نظر عنصر هماهنگ ساز (Synch) است که از انباره حذف می‌کنیم.

\*\* نکته: (حالت استثناء) به شرطی حذف از انباره صورت می‌گیرد که نماد شروع در موقعیت حذف قرار گیرد، آنوقت این نماد حذف نمی‌شود و به جای آن از ورودی حذف می‌کنیم.

«مثال: گرامر زیر را در نظر بگیرید. با استفاده از پارس (1) و روش Panic Mode عبارت " + id \* + id " را پارس و خطاهای موجود را اعلام کنید.

0	$E'' \rightarrow E\$$
1	$E \rightarrow TE'$
2	$E' \rightarrow +TE'$
3	$E' \rightarrow \epsilon$
4	$T \rightarrow FT'$
5	$T' \rightarrow *FT'$
6	$T' \rightarrow \epsilon$
7	$F \rightarrow (E)$
8	$F \rightarrow id$

» پاسخ: ابتدا تمامی First ها و Follow ها را می‌یابیم:

$$First(E) = First(T) = First(F) = \{ (, id \}$$

$$First(E') = \{ +, \epsilon \}$$

$$First(T') = \{ *, \epsilon \}$$

$$\blacktriangleright Follow(E) = \{ \$, ) \}$$

$$Follow(E') = Follow(E) = \{ \$, ) \}$$

$$\blacktriangleright Follow(T) = First(E') + Follow(E) + Follow(E') = \{ +, \$, ) \}$$

$$Follow(T') = Follow(T) = \{ +, \$, ) \}$$

$$\blacktriangleright Follow(F) = First(T') + Follow(T) + Follow(T') = \{ *, +, \$, ) \}$$

سپس جدول پارس (1) را تشکیل میدهیم:

\*\* توجه: Synch عناصر مجموعه هماهنگ ساز است.

	<i>id</i>	+	*	(	)	\$
<i>E</i>	1			1	Synch	Synch
<i>E'</i>		2			<u>3</u>	<u>3</u>
<i>T</i>	4	Synch		4	Synch	Synch
<i>T'</i>		<u>6</u>	5		<u>6</u>	<u>6</u>
<i>F</i>	8	Synch	Synch	7	Synch	Synch

عبارت مورد نظر:  $+ ) id * + id$

	انباره	ورودی	عمل انجام شده
1	$\$ E$	$+ ) id * + id \$$	خطا - حذف "+"
2	$\$ E$	$) id * + id \$$	خطا - استثنای حذف "("
3	$\$ E$	$id * + id \$$	بسط با قاعده ۱
4	$\$ E' T$	$id * + id \$$	بسط با قاعده ۴
5	$\$ E' T' F$	$id * + id \$$	بسط با قاعده ۸
6	$\$ E' T' \cancel{id}$	$\cancel{id} * + id \$$	نظیر شدن id
7	$\$ E' T'$	$* + id \$$	بسط با قاعده ۵
8	$\$ E' T' F *$	$* + id \$$	نظیر شدن *
9	$\$ E' T' F$	$+ id \$$	خطا - حذف F
10	$\$ E' T'$	$+ id \$$	بسط با قاعده ۶
11	$\$ E'$	$+ id \$$	بسط با قاعده ۲
12	$\$ E' T +$	$+ id \$$	نظیر شدن +
13	$\$ E' T$	$id \$$	بسط با قاعده ۴
14	$\$ E' T' F$	$id \$$	بسط با قاعده ۸
15	$\$ E' T' \cancel{id}$	$\cancel{id} \$$	نظیر شدن id
16	$\$ E' T'$	$\$$	بسط با قاعده ۶
17	$\$ E'$	$\$$	بسط با قاعده ۳
18	$\$$	$\$$	پذیرش با ۳ خطا

## ۵.۷ روش پارس پایین گرد

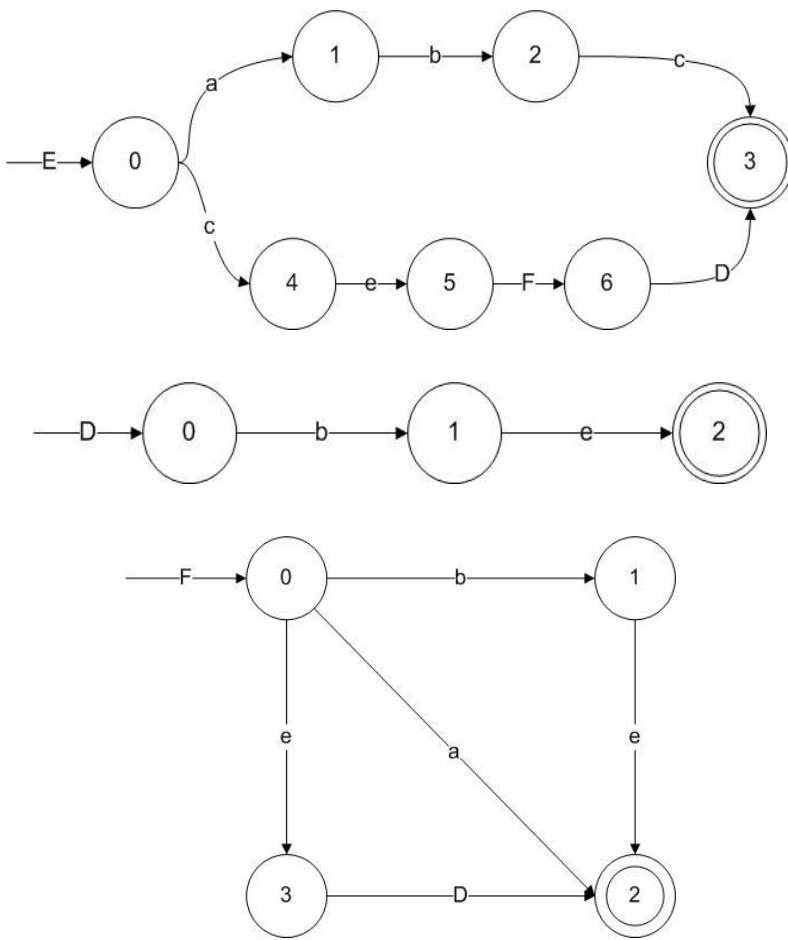
در این روش به ازای هر کدام از غیر پایانه‌ها یک تابع داریم.

۱. اگر غیر پایانه چند بسط مختلف داشته باشد، با مقایسه نماد پیشگویی با مجموعه First آن بسط، یکی از بسط‌های آن را انتخاب می‌کنیم.
۲. سعی می‌کنیم سمت راست آن قاعده‌ای که برای بسط انتخاب شده در ورودی پیدا کنیم (توسط پارس انجام می‌شود). اگر سمت راست شامل پایانه و غیرپایانه باشد، به ازای پایانه‌ها باید آنها را عیناً در ورودی داشته باشیم و اگر غیر پایانه باشد باید تابع مربوطه را صدا کنیم.

برای توضیح این قسمت گرامر زیر را به عنوان مثال در نظر بگیرید:

$$\begin{array}{l} 1,2 \quad E \rightarrow abc \mid cefD \\ 3 \quad D \rightarrow be \\ 4,5,6 \quad F \rightarrow a \mid be \mid eD \end{array}$$

دیاگرام گذر گرامر پیشین، به صورت زیر است:



شیه کد گرامری که در بالا گفته شد، به صورت زیر است:

**Proc E ( )**

```
{
    if L.A. = a then 1
    else if L.A. = c then 2
    else error;
}
```

**Proc D ( )**

```
{
    if L.A. = b then 3
    else error;
}
```

**Proc F ( )**

```
{
    if L.A. = a then 4
    else if L.A. = b then 5
    else if L.A. = e then 6
    else error;
}
```

**Proc Match ( i : token)**

```
{
    if L.A. = i then L.A. = next_token( )
    else error;
}
```

\*\* نکته: به جای پایانه ها، Match میگذاریم و به جای غیرپایانه ها، توابع آنها را فراخوانی میکنیم. پس اگر بخواهیم توابع قبل را کاملتر بنویسیم داریم:

**Proc E ( )**

```
{
    If L.A. = a then Match(a); Match(b); Match(c);
    else if L.A. = c then Match(c); Match(e); Proc F( ); Proc D( );
    else error;
}
```

**Proc D ( )**

```
{
    if L.A. = b then Match(b); Match(e);
    else error;
}
```

**Proc F ( )**

```
{
    if L.A. = a then Match(a);
    else if L.A. = b then Match(b); Match(e);
    else if L.A. = e then Match(e); Proc D( );
    else error;
}
```

« مثال: به روش پارس پایین‌گرد گرامر زیر را پارس کنید.

```
1,2   < Type > → < Simple > | < Array_Def >
3,4,5 < Simple > → int | real | char
6     < Array_Def > → array [num ... num] of < Type >
```

« پاسخ:

**Proc Type ( )**

```
{
    if L.A. = { int, real, char } then Simple( );
    else if L.A. = array then Array_Def( );
    else error;
}
```

**Proc Simple ( )**

```
{
    if L.A. = int then Match(int);
    else if L.A. = real then Match(real);
    else Match(char);
}
```

**Proc Array\_Def ( )**

```
{
    Match( array );
    Match( [ ] );
    Match( num );
    Match( ... );
    Match( num );
    Match( J );
    Match( of );
    Type( );
}
```

\*\* نکته: روش پارس پایین گرد به سادگی قابل پیاده‌سازی است، اما به دلیل فراخوانی‌های بازگشتی متعددی که دارد، روش کندی است و معمولاً در کامپایلرها مورد استفاده قرار نمی‌گیرد. اگر حجم برنامه زیاد باشد، عملیات کامپایلر گند خواهد شد.

### ۵.۸ پارسرهای پایین به بالا

عکس پارسرهای بالا به پایین عمل می‌کنند. فرض میکنیم جمله‌ای که میخواهیم بدانیم متعلق به زبان است، وجود دارد و هر کدام از اجزای آن برگهای یک درخت هستند. از پایین به بالای درخت حرکت میکنیم تا به نماد شروع برسیم. اگر به نماد شروع رسیدیم، یعنی جمله متعلق به زبان است. چند روش گوناگون برای این نوع پارسرهای وجود دارد.

\*\* نکته: وضعیت شروع در پارسرهای پایین به بالا معادل وضعیت پایان در پارسرهای بالا به پایین است.

\*\* نکته: وضعیت پایان در پارسرهای پایین به بالا معادل وضعیت شروع در پارسرهای بالا به پایین است.

پارسرهای بالا به پایین	پارسرهای پایین به بالا
\$S	\$
:	:
:	:
\$	\$S

### ۵.۹ روش انتقال-کاهش (Shift-Reduce)

- **تعريف انتقال**

انتقال یعنی پایانه‌ای را از ورودی بخوانیم و آن را در انباره قرار دهیم.

- **تعريف کاهش**

کاهش یعنی اگر در بالای انباره یک دستگیره ظاهر شد، آن را با معادل سمت چپش در گرامر جایگزین کنیم.

- **تعريف دستگیره**

دستگیره به جزئی از ورودی می‌گویند که برای جایگذاری انتخاب می‌شود.

- **تعريف عبارت**

به قسمتی از یک فرم جمله‌ای که در یک مرحله یا بیشتر، از یک غیرپایانه بدست آمده باشد، عبارت می‌گویند.

$$\alpha A \beta \xrightarrow{+} \alpha \delta \beta$$

$$A \xrightarrow{+} \delta$$

### • تعریف عبارت ساده

عبارتی که در یک مرحله از غیرپایانه مربوطه بدست آمده باشد.

$$\alpha A\beta \Rightarrow \alpha\delta\beta$$

$$A \Rightarrow \delta$$

### • تعریف دیگری از دستگیره

دستگیره عبارت ساده‌ای است که در جهت عکس بسط سمت راست ترین (عکس عمل RMD) به وجود آمده باشد.

«مثال: آیا عبارت abcdē متعلق به زبان زیر است یا خیر؟»

$$1 \quad S \rightarrow aABe$$

$$2,3 \quad A \rightarrow Abc \mid b$$

$$4 \quad B \rightarrow d$$

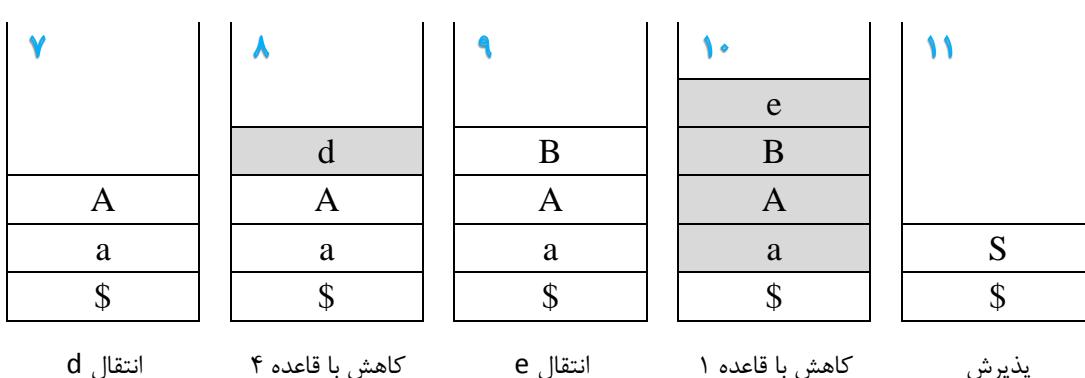
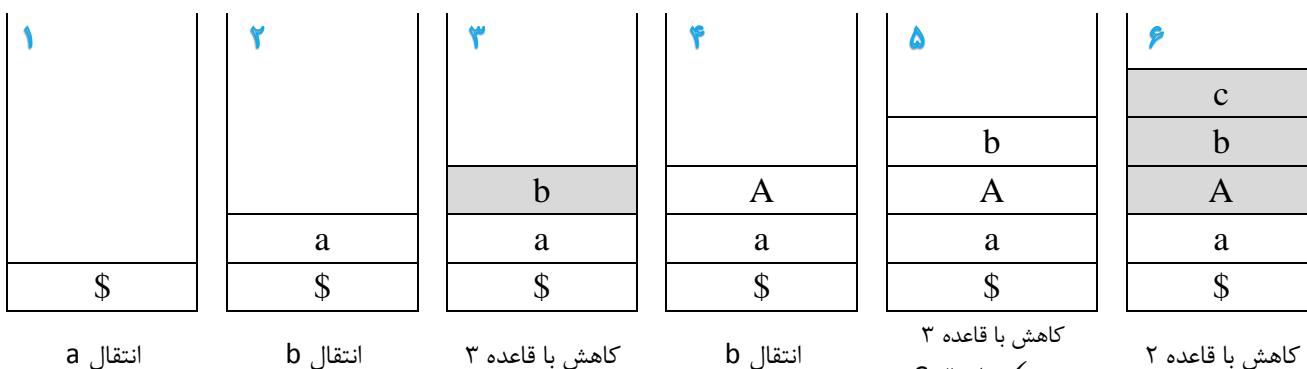
«پاسخ:

**RMD:**

$$S \xrightarrow{1} aABe \xrightarrow{4} aAde \xrightarrow{2} aAbcde \xrightarrow{3} abcdē$$

**عکس عمل:**

$$S \Leftarrow aABe \Leftarrow aAde \Leftarrow aAbcde \Leftarrow abcdē$$

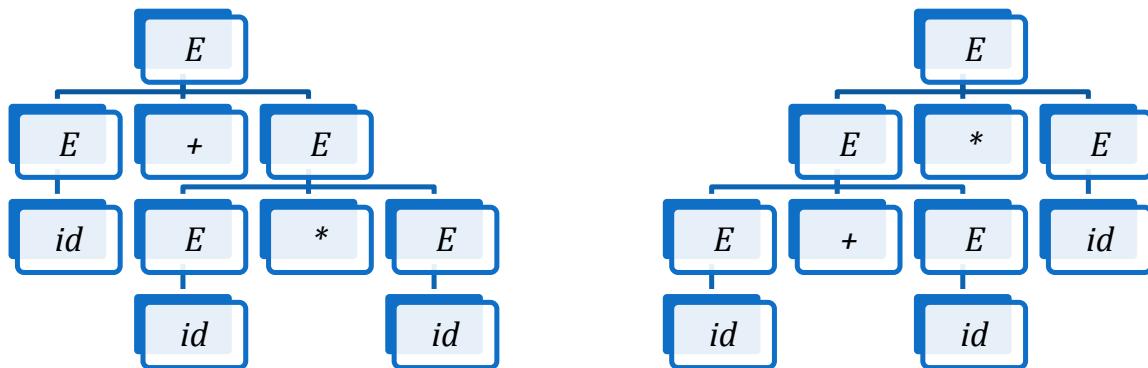


\*\* نکته: مشکل روش انتقال-کاهش این است که انتخاب دستگیره صحیح دشوار است و بین دو قاعده که سمت راست یکسان دارند، یافتن دستگیره مناسب‌تر دشوار است.

«مثال: عبارت  $id + id * id$  را با روش انتقال-کاهش بررسی کنید.»

- 1  $E \rightarrow E + E$
- 2  $E \rightarrow E * E$
- 3  $E \rightarrow (E)$
- 4  $E \rightarrow id$

«پاسخ:



	انباره	ورودی	عمل انجام شده
1	\$	$id + id * id \$$	انتقال id
2	\$ <b><i>id</i></b>	$+id * id \$$	کاهش با قاعده ۴ ( $R_4$ )
3	\$ <b><i>E</i></b>	$+id * id \$$	انتقال +
4	\$ <b><i>E +</i></b>	$id * id \$$	انتقال id
5	\$ <b><i>E + id</i></b>	$* id \$$	کاهش با قاعده ۴ ( $R_4$ )
6	\$ <b><i>E + E</i></b>	$* id \$$	تداخل: <u>(انتقال *)</u> یا <u>(<math>R_1</math>)</u>
7	\$ <b><i>E + E *</i></b>	$id \$$	انتقال id
8	\$ <b><i>E + E * id</i></b>	$\$$	کاهش با قاعده ۴ ( $R_4$ )
9	\$ <b><i>E + E * E</i></b>	$\$$	کاهش با قاعده ۲ ( $R_2$ )
10	\$ <b><i>E + E</i></b>	$\$$	کاهش با قاعده ۱ ( $R_1$ )
11	\$ <b><i>E</i></b>	$\$$	پذیرش

## ۵.۱۰ انواع تداخل

۱. تداخل انتقال - کاهش (Shift-Reduce): زمانیکه پارسرا نمیداند باید انتقال دهد یا کاهش.
۲. تداخل کاهش - کاهش (Reduce-Reduce): زمانیکه پارسرا نمیداند با کدام قاعده باید کاهش دهد.

« مثال: تداخل انتقال - کاهش »

1     $< Stmt > \rightarrow if < Condition > then < Stmt >$   
 2     $< Stmt > \rightarrow if < Condition > then < Stmt > else < Stmt >$

انباره	ورودی	عمل انجام شده
$\dots if < Condition > then < Stmt >$	$else < Stmt >$	تداخل انتقال-کاهش (S-R)

در اینجا انتقال فقط درست است. چون  $else$  بدون  $if$  معنی ندارد.

« مثال: تداخل کاهش-کاهش »

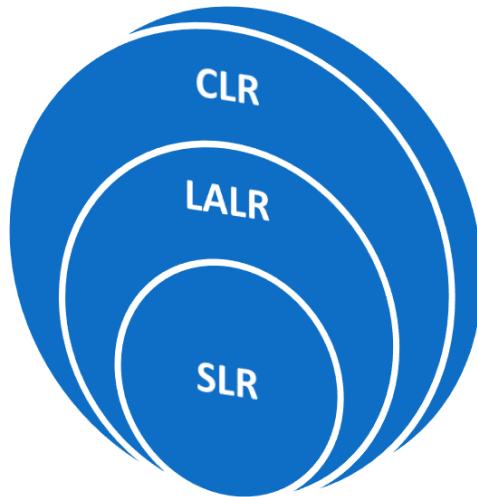
1     $A \rightarrow abc$   
 2     $B \rightarrow abc$

انباره	ورودی	عمل انجام شده
$\dots abc$	$efg$	تداخل کاهش-کاهش ( $R_2$ یا $R_1$ )

## ۵.۱۱ روش‌های LR

این روشها جزء قویترین روش‌های پارس پایین به بالا هستند که به سه دسته تقسیم می‌شود:

۱. SLR (از همه ساده‌تر)
۲. CLR (از همه قوی‌تر)
۳. LALR (بین دو مورد بالا)



## ۵.۱۲ چهار مزیت اصلی روش‌های LR

۱. کلی ترین روش پارس پایین به بالا هستند و معمولاً آنها را میتوان به کارایی روش‌های دیگر پیاده سازی کرد.
۲. بیشتر ساختارهای زبانهای برنامه‌نویسی را پشتیبانی میکند.
۳. مجموعه گرامرهايی که توسط روش‌های LR پارس میشوند، یک مجموعه کامل از گرامرهايی هستند که توسط انواع پارسرهای دیگر پوشش داده میشود.
۴. خطاهای را از همه پارسرهای دیگر سریعتر و صحیح‌تر پیدا می‌کنند.

## LR(0).item ۵.۱۳

سمت راست یک قاعده یک نقطه (.) می‌افزاییم.

وقتی نقطه قبل از یک عبارت است، یعنی پارس اگر آن عبارت را ببیند چه میکند.

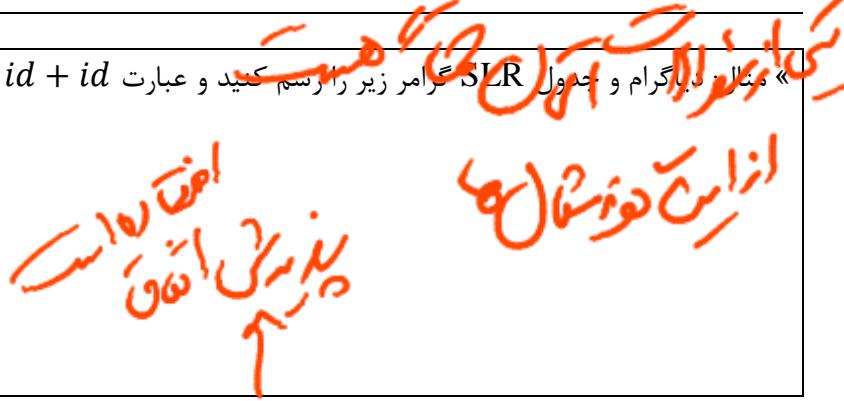
وقتی نقطه به انتهای عبارتی برسد، یعنی هنگام کاهش است.

$$\begin{aligned} A &\rightarrow xyz \\ A &\rightarrow .xyz \xrightarrow{x} A \rightarrow x.yz \xrightarrow{y} A \rightarrow xy.z \xrightarrow{z} A \rightarrow xyz. \quad (\text{زمان کاهش است}) \end{aligned}$$

SLR گرامر زیر رارسم کنید و عبارت  $id * id + id$  را توسط این دو پارس کنید.

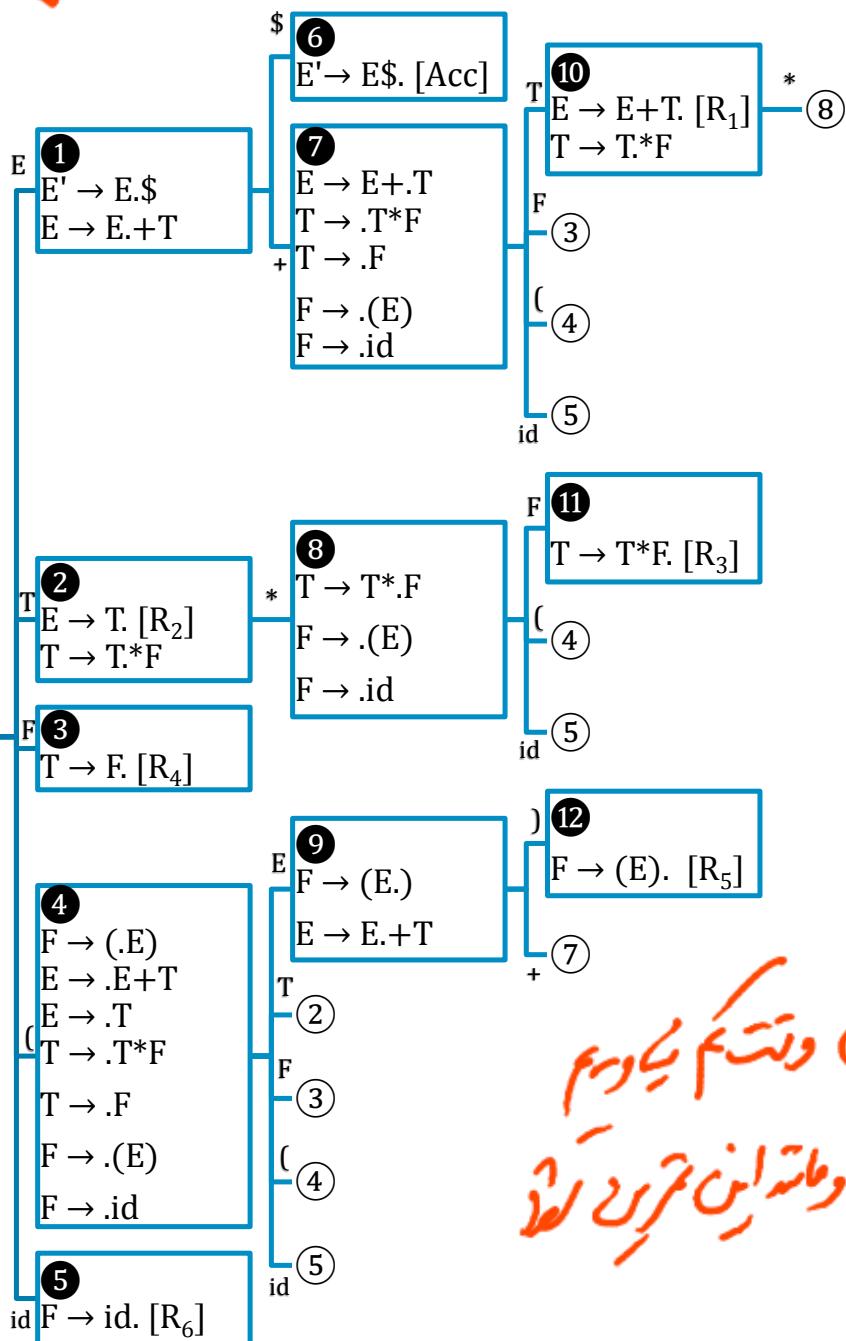
- 0  $E' \rightarrow E\$$
- 1  $E \rightarrow E + T$
- 2  $E \rightarrow T$
- 3  $T \rightarrow T * F$
- 4  $T \rightarrow F$
- 5  $F \rightarrow (E)$
- 6  $F \rightarrow id$

به عنوان مثال این



لزذا تمام تابع گرامر  
(روالنت صفر)  
مغایر شدنی نیست

- 0  $E' \rightarrow .E\$$
- 1  $E \rightarrow .E + T$
- 2  $E \rightarrow .T$
- 3  $T \rightarrow .T * F$
- 4  $T \rightarrow .F$
- 5  $F \rightarrow .(E)$
- 6  $F \rightarrow .id$



\*\* نکته: به ازای عبارات روی فلش، شیفت داریم و به ازای  $R_n$  تا  $R_1$  کاهش داریم.  
 \*\* نکته: حتماً باید از  $R_1$  تا  $R_n$  را بینیم.

$$Follow(E) = \{\$, +, )\}$$

$$Follow(T) = \{\$, +, ), *\}$$

$$Follow(F) = \{\$, +, ). *\}$$



حال با توجه به دیاگرام SLR کاشیده شده، جدول SLR را بینید.

راهنمای جدول: SLR جدول پارس را در متریک تحریر لغزشی درخواست کنید.  
 Empty Cell=Error و Acc=Accept .S=Shift .R=Reduce

برای این عکس کوچکتر  
 (non نون نفرمی را  
 بزرگتر نمایند)

شروع اس

Terminals									Non-Terminals		
id	+	*	(	)	\$	E	T	F			
0	S <sub>5</sub>			S <sub>4</sub>		1	2	3	4	5	6
1		S <sub>7</sub>			Acc	1	2	3	4	5	6
2	R <sub>2</sub>	S <sub>8</sub>	follow T	R <sub>2</sub>	R <sub>2</sub>	1	2	3	4	5	6
3	R <sub>4</sub>	R <sub>4</sub>		R <sub>4</sub>	R <sub>4</sub>						
4	S <sub>5</sub>			S <sub>4</sub>		9	2	3			
5	R <sub>6</sub>	R <sub>6</sub>		R <sub>6</sub>	R <sub>6</sub>						
6											
7	S <sub>5</sub>			S <sub>4</sub>			10	3			
8	S <sub>5</sub>			S <sub>4</sub>				11			
9	S <sub>7</sub>				S <sub>12</sub>						
10	R <sub>1</sub>	S <sub>8</sub>		R <sub>1</sub>	R <sub>1</sub>						
11	R <sub>3</sub>	R <sub>3</sub>		R <sub>3</sub>	R <sub>3</sub>						
12	R <sub>5</sub>	R <sub>5</sub>		R <sub>5</sub>	R <sub>5</sub>						
Action									Go To		

مکانیزم عد  
کارکرده آنها  
ردیابی

\*\* نکته: به ازای Follow های سمت چپ، عملیات کاهش را انجام میدهیم.

حال عبارت  $id * id + id$  را با توجه به جدول قبل به روش SLR پارس میکنیم:

	عمل انجام شده
1	: انتقال $id$ و رفتن به مرحله ۵ $S_5$
2	: کاهش با قاعده ۶ $R_6$
3	: کاهش با قاعده ۴ $R_4$
4	: انتقال $*$ و رفتن به مرحله ۸ $S_8$
5	: انتقال $id$ و رفتن به مرحله ۵ $S_5$
6	: کاهش با قاعده ۶ $R_6$
7	: کاهش با قاعده ۳ $R_3$
8	: کاهش با قاعده ۲ $R_2$
9	: انتقال $+$ و رفتن به مرحله ۷ $S_7$
10	: انتقال $id$ و رفتن به مرحله ۵ $S_5$
11	: کاهش با قاعده ۶ $R_6$
12	: کاهش با قاعده ۴ $R_4$
13	: کاهش با قاعده ۱ $R_1$
14	پذیرش

« مثال: جدول پارس SLR را برای گرامر زیر رسم کنید.

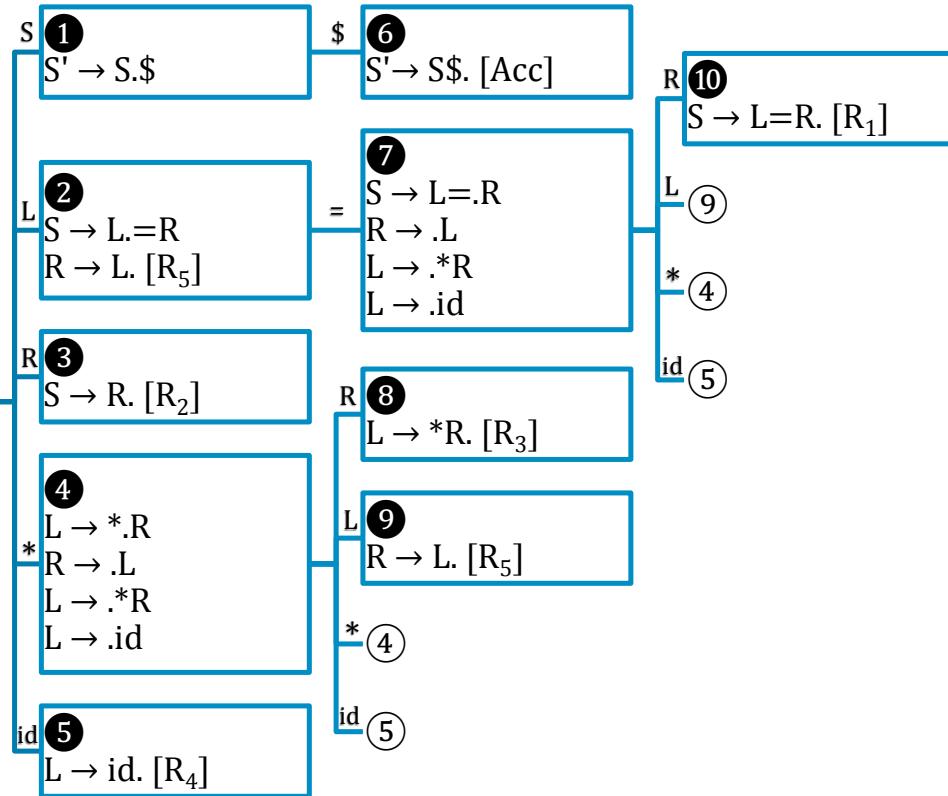
- 0  $S' \rightarrow S\$$
- 1  $S \rightarrow L = R$
- 2  $S \rightarrow R$
- 3  $L \rightarrow * R$
- 4  $L \rightarrow id$
- 5  $R \rightarrow L$

$$Follow(S) = \{\$\}$$

$$Follow(L) = \{=\} + Follow(R) = \{ \$, = \}$$

$$Follow(R) = Follow(S) + Follow(L) = \{ \$, = \}$$

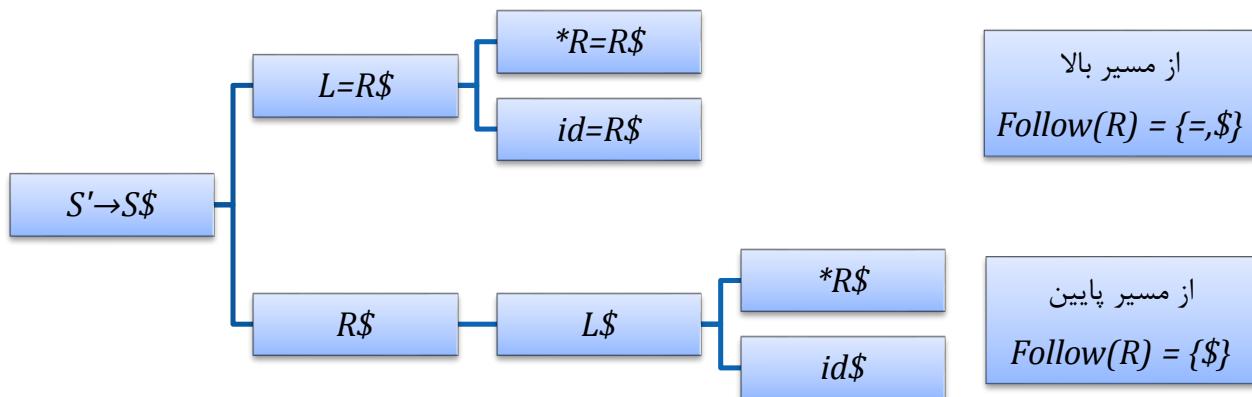
۰  
 $S' \rightarrow .S\$$   
 $S \rightarrow .L=R$   
 $S \rightarrow .R$   
 $L \rightarrow .*R$   
 $L \rightarrow .id$   
 $R \rightarrow .L$



حال مطابق دیاگرام فوق، جدول SLR میکشیم.

<b>id</b>	*	=	\$	S	L	R
0	$S_5$	$S_4$		1	2	3
1			Acc			
2			$S_7$ $\diagup$ $R_5$	$R_5$		
3				$R_2$		
4	$S_5$	$S_4$			9	8
5			$R_4$	$R_4$		
6						
7	$S_5$	$S_4$			9	10
8			$R_3$	$R_3$		
9			$R_5$	$R_5$		
10				$R_1$		

\*\* توجه: همانطور که شاهد هستیم در State 2 تداخل انتقال-کاهش داریم. پس گرامر فوق SLR نیست. این تداخل به ازای Followهای R به وجود آمده است. برای حل این مشکل از روش‌های دیگری استفاده میکنیم. در ادامه با روش آشنا میشویم CLR



روش بالا که با توجه به مسیر Follow را تعیین می‌کند، روش CLR می‌گویند. با توجه به این روش اگر بخواهیم تداخل از بین برود، در 2 State به ازای مساوی (=) فقط  $S_7$  می‌ماند.

« تمرين: برای گرامر زیر جدول پارس SLR بکشید و عبارت cacbdb را به وسیله آن پارس کنید.

- 1       $S \rightarrow AB$
- 2,3     $A \rightarrow aBd \mid \epsilon$
- 4       $B \rightarrow cacbdb$

## CLR روش ۵,۱۴

در این روش سعی می‌شود برای حل مشکل تداخل-کاهش، از همه Follow ها استفاده نشود، بلکه بسته به مسیر از Follow ها استفاده کنیم.

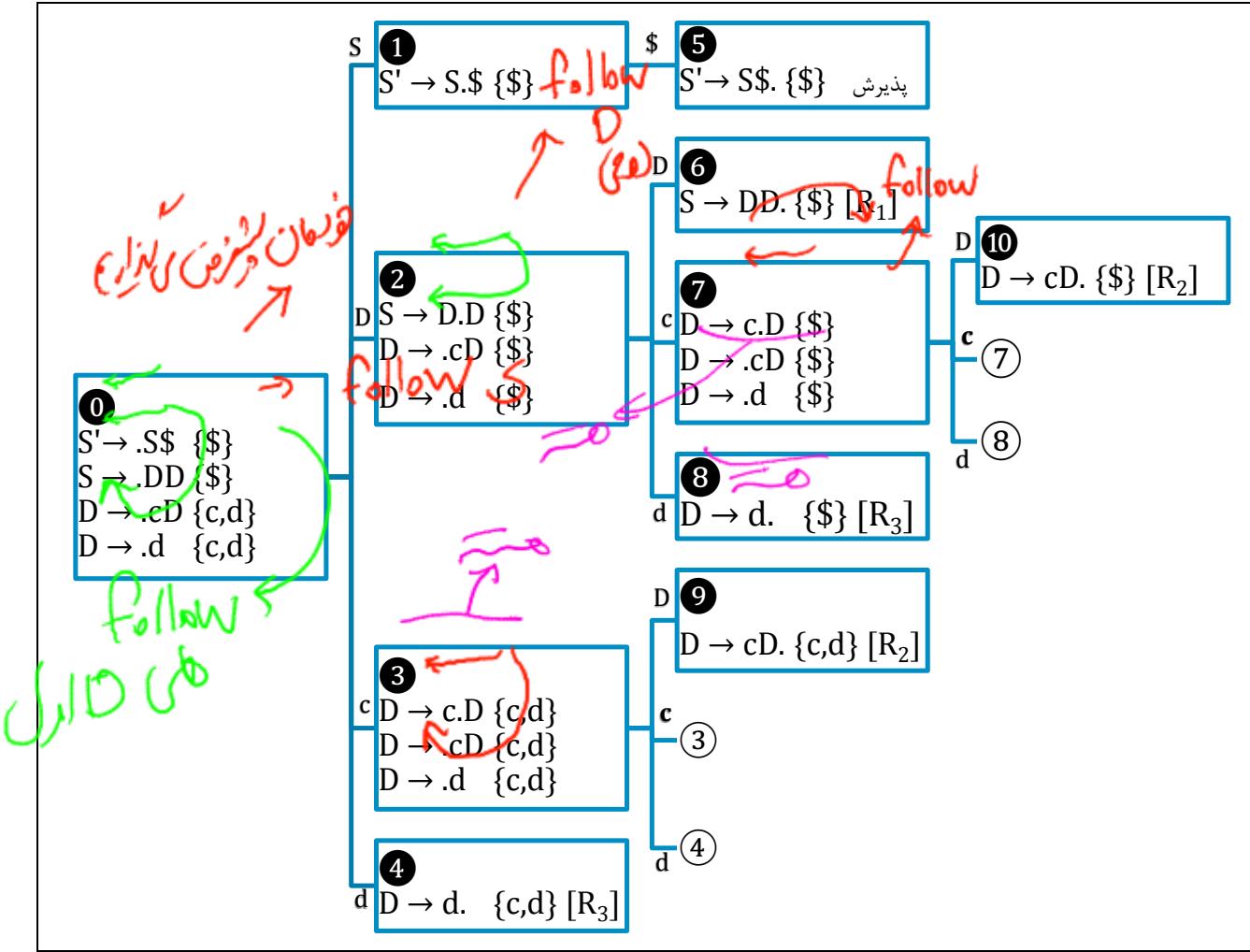
LR(0) + L.A.

« مثال: دیاگرام گرامر زیر را به روش CLR رسم کنید.

- 0       $S' \rightarrow S\$$
- 1       $S \rightarrow DD$
- 2,3     $D \rightarrow cD \mid d$

$$Follow(S) = \{\$\}$$

$$Follow(D) = \{c, d, \$\}$$



\*\* تذکر: تفاوت دیاگرام CLR در این است که Follow چون کاهش را وقتی نقطه به آخر رسید به ازای Follow سمت چپ انجام می‌دهیم، پس Follow سمت چپ هر گرامر را در جلوی آن می‌نویسیم.

\*\* تذکر: CLR هم از لحاظ حافظه و هم از لحاظ قدرت از SLR بیشتر است. ( $CLR > SLR$ )

\*\* نکته: اگر گرامری CLR باشد، لزوماً SLR نیست.

\*\* نکته: اگر گرامری CLR نباشد، حتماً SLR هم نیست.

\*\* نکته: اگر گرامری SLR باشد، حتماً CLR هم هست.

\*\* نکته: اگر گرامری SLR نباشد، نمیتوان گفت که CLR هم نیست.

	<b>c</b>	<b>d</b>	\$	<b>S</b>	<b>D</b>
0	$S_3$	$S_4$		1	2
1			Acc		
2	$S_7$	$S_8$			6
3	$S_3$	$S_4$			9
4	$R_3$	$R_3$			
5					
6			$R_1$		
7	$S_7$	$S_8$			10
8			$R_3$		
9	$R_2$	$R_2$			
10			$R_2$		

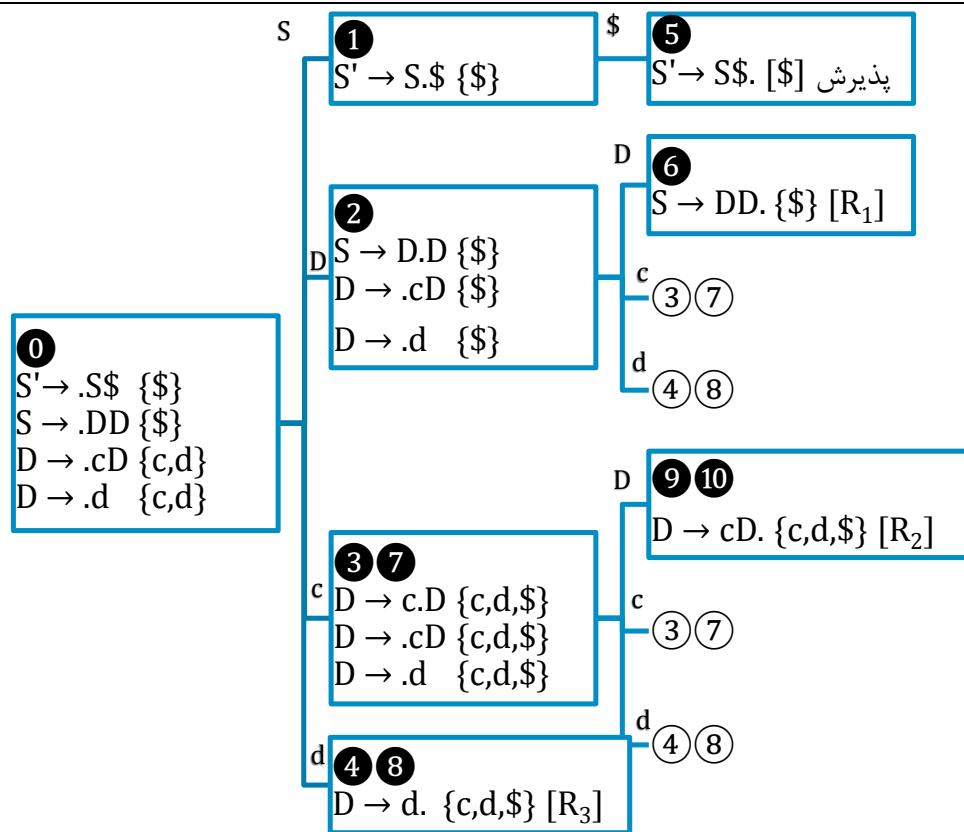
### LALR ۵,۱۵ روش

از روی CLR به دست می‌اید. در روش LALR وضعیت‌هایی که هسته یکسانی دارند را ادغام کرده و L.A. یا Follow ها را اجتماع می‌گیریم.

هسته: در یک وضعیت تمام قسمتها به جزء Follow یا L.A. ها را هسته می‌گوییم.

« مثال: دیاگرام گرامر زیر را به روش LALR رسم کنید. »

1       $S \rightarrow DD$   
 2,3     $D \rightarrow cD \mid d$



وضعیتهای ۳ و ۷ چون هسته یکسان دارند با هم ادغام می‌شوند و وضعیت جدید به صورت ۳۷ نشان داده می‌شود. (وضعیت ۷ را نیز حذف می‌کنیم) و Follow ۳۷ می‌شود  $\{c,d,\$ \}$ . وضعیتهای ۴ و ۸ و وضعیتهای ۹ و ۱۰ هم همین طور هستند.

	c	d	\$	S	D
0	$S_{37}$	$S_{48}$		1	2
1			Acc		
2	$S_{37}$	$S_{48}$			6
37	$S_{37}$	$S_{48}$			910
48	$R_3$	$R_3$	$R_3$		
5					
6			$R_1$		

910

$R_2$	$R_2$	$R_2$		
-------	-------	-------	--	--

\*\* نکته: از SLR امده بود و تداخل نداشت. در LALR چون Follow ها ادغام می‌شوند و به ازای Follow ها کاهش داریم، پس امکان تداخل کاهش-کاهش وجود دارد.

(SLR < LALR < CLR) از لحاظ قدرت و حافظه:

« مثال: عبارت  $ccd$  را با LALR و CLR پارس کنید. »

• پارس به روش LALR

	انباره	ورودی	عمل انجام شده
1	0	$ccd \$$	c : انتقال $S_{37}$
2	0 c 37	$cd \$$	c : انتقال $S_{37}$
3	0 c 37 c 37	$d \$$	d : انتقال $S_{48}$
4	0 c 37 c 37 d 48	\$	کاهش با قاعده ۳ $R_3$
5	0 c 37 c 37 D 910	\$	کاهش با قاعده ۲ $R_2$
6	0 c 37 D 910	\$	کاهش با قاعده ۲ $R_2$
7	0 D 2	\$	خطا

• پارس به روش CLR

	انباره	ورودی	عمل انجام شده
1	0	$ccd \$$	c : انتقال $S_3$
2	0 c 3	$cd \$$	c : انتقال $S_3$
3	0 c 3 c 3	$d \$$	d : انتقال $S_4$
4	0 c 3 c 3 d 4	\$	خطا

\*\* نکته: همانطور که مشاهده می‌شود روش CLR زودتر خطرا شناسایی می‌کند.

\*\* تذکر: در اینجا ممکن است این سوال مطرح شود که اگر گرامری CLR باشد، بعد از اینکه به LALR تبدیل شود، آیا در آن احتمال بروز خطرا وجود دارد یا نه؟

جواب این است که اصلاً احتمال بروز تداخل انتقال-کاهش وجود ندارد. تنها ممکن است تداخل کاهش-کاهش به وجود بیاید، زیرا Follow ها با هم ترکیب می‌شوند.

« تمرین: گرامر زیر را با روش LALR بررسی کنید. »

- 0  $S' \rightarrow S\$$
- 1  $S \rightarrow L = R$
- 2  $S \rightarrow R$
- 3  $L \rightarrow * R$
- 4  $L \rightarrow id$
- 5  $R \rightarrow L$

نکته: از مزایای گرامرهای مبهم می‌توان به سادگی و خلاصه تر بودن آنها اشاره کرد.

- 1  $E \rightarrow E + E$
- 2  $E \rightarrow E * E$
- 3  $E \rightarrow (E)$
- 4  $E \rightarrow id$

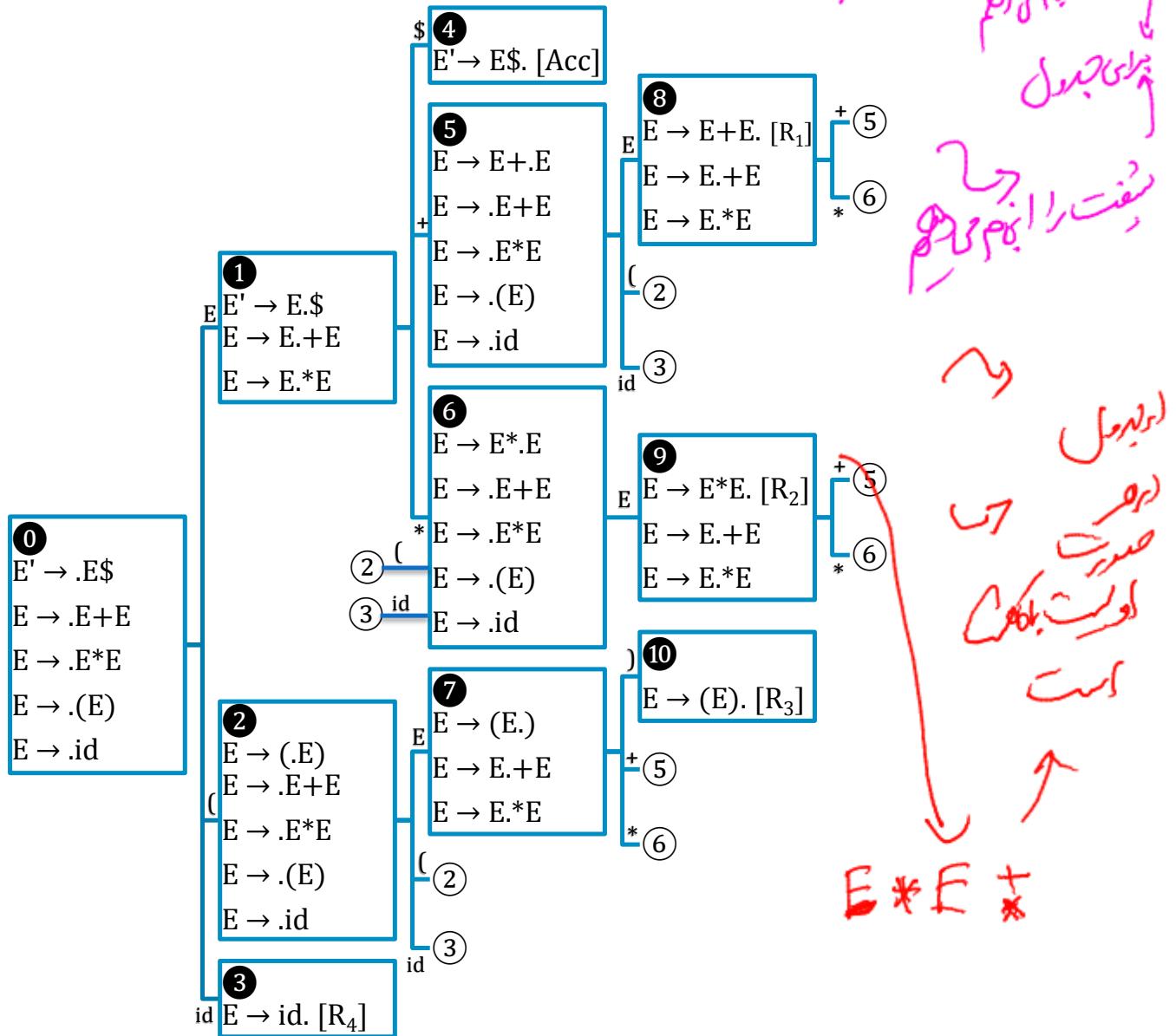
معادل غیر مبهم گرامر فوق به صورت زیر است:

- 1  $E \rightarrow E + T$
- 2  $E \rightarrow T$
- 3  $T \rightarrow T * F$
- 4  $T \rightarrow F$
- 5  $F \rightarrow (E)$
- 6  $F \rightarrow id$

اگر بتوانیم مشکلات گرامرهای مبهم را برطرف کنیم و از آنها استفاده کنیم بهتر است. برای مثال در گرامر مبهم فوق از قاعده ۴ مستقیماً نتیجه می‌گیریم  $E \rightarrow id$  ولی در گرامر غیر مبهم داریم:

- $$\begin{aligned} E &\rightarrow T \\ T &\rightarrow F \\ F &\rightarrow id \end{aligned}$$

دیگر گرام SLR گرامر می‌بهم را رسم می‌کنیم:



$$\text{Follow}(E) = \{\$, +, *, ()\}$$

<b>id</b>	<b>+</b>	<b>*</b>	<b>(</b>	<b>)</b>	<b>\$</b>	<b>E</b>
0	$S_3$	$e_1$	$e_1$	$S_2$	$e_2$	$e_1$
1	$e_2$	$S_5$	$S_6$	$e_3$	$e_2$	Acc
2	$S_3$	$e_1$	$e_1$	$S_2$	$e_2$	$e_1$
3		$R_4$	$R_4$		$R_4$	$R_4$
4						
5	$S_3$	$e_1$	$e_1$	$S_2$	$e_2$	$e_1$
6	$S_3$	$e_1$	$e_1$	$S_2$	$e_2$	$e_1$
7	$e_3$	$S_5$	$S_6$	$e_3$	$S_{10}$	$e_4$
8		$\cancel{S_5^*}$ $\cancel{R_1}$	$\cancel{S_6^*}$ $\cancel{R_1}$		$R_1$	$R_1$
9		$\cancel{S_5^*}$ $\cancel{R_2}$	$\cancel{S_6^*}$ $\cancel{R_2}$		$R_2$	$R_2$
10		$R_3$	$R_3$		$R_3$	$R_3$

خانه‌هایی را که در آنها تداخل داریم را بررسی کرده و سعی میکنیم تداخل آنها را رفع کنیم.

- در تقاطع خانه ۸ و + داریم:

با توجه به این که عبارت از چپ به راست بررسی میشود، ابتدا باید (+) اوّل اعمال شود، سپس (+) دوم؛ به عبارت دیگر اولویت با عمل کاهش است.

$$\underbrace{E + E}_E + E \rightarrow \underbrace{1 + 2}_3 + 3$$

- در تقاطع خانه ۸ و \* داریم:

در اینجا ابتدا باید عمل ضرب (\*) اعمال شود، چرا که عمل ضرب (\*) نسبت به عمل جمع (+) تقدّم دارد؛ به عبارت دیگر اولویت با عمل انتقال است.

$$E + \underbrace{E * E}_E \rightarrow 1 + \underbrace{2 * 3}_6$$

- در تقاطع خانه ۹ و  $+E$  داریم:

در این حالت مشخص است که اولویت با عمل ضرب (\*) است و سپس عمل جمع (+) اعمال میشود؛ به عبارت دیگر اولویت با عمل کاهش است.

$$\underbrace{E * E}_{E} + E \rightarrow \underbrace{1 * 2}_{2} + 3$$

- در تقاطع خانه ۹ و  $*E$  داریم:

همانطور که میبینیم هر دو عمل ضرب (\*) دارای اولویت یکسان هستند، پس از چپ شروع میکنیم؛ به عبارت دیگر اولویت با عمل کاهش است.

$$\underbrace{E * E}_{E} * E \rightarrow \underbrace{1 * 2}_{2} * 3$$

پس جدول صحیح به صورت زیر است:

	<b>id</b>	+	*	(	)	\$	<b>E</b>
0	$S_3$			$S_2$			1
1		$S_5$	$S_6$			Acc	
2	$S_3$			$S_2$			7
3		$R_4$	$R_4$		$R_4$	$R_4$	
4							
5	$S_3$			$S_2$			8
6	$S_3$			$S_2$			9
7		$S_5$	$S_6$		$S_{10}$		
8		$R_1$	$S_6$		$R_1$	$R_1$	
9		$R_2$	$R_2$		$R_2$	$R_2$	
10		$R_3$	$R_3$		$R_3$	$R_3$	

» تمرین: مثال قبل را با روش LALR بررسی کنید.

لگزین اثیر را نمایست

## ۵.۱۶ اصلاح خطای روش LR در پارسرهای Phrase Level

خانه های حالي در جدول پارسر که نشانه خطای هستند را با استفاده از توابع بر میکنیم. این توابع خطاهای را بررسی کرده و پیام مناسب چاپ میکنند.

خطا	عملیات	پیغام
$e_1$	یک id روی انباره بگذار و به وضعیت ۳ برو	عملوند گم شده است.
$e_2$	پرانتر بسته "(" را از ورودی حذف کن	پرانتر بسته "(" اضافی
$e_3$	"+" را روی انباره بگذار و به حالت ۵ برو	عملگر گم شده است.
$e_4$	پرانتر بسته ")" را روی انباره بگذار و به وضعیت ۱۰ برو	پرانتر بسته ")" گم شده است.

« مثال: عبارت  $\$ + id$  را پارس کنید. »

عمل انجام شده	ورودی	انباره
$S_3$	$id + \$$	0
$R_4$	$+ \$$	$0 id 3$
$S_5$	$+ \$$	$0 E 1$
خطا : $e_2$	$) \$$	$0 E 1 + 5$
خطا : $e_1$	$\$$	$0 E 1 + 5$
$R_4$	$\$$	$0 E 1 + 5 id 3$
$R_1$	$\$$	$0 E 1 + 5 E 8$
پذیرش با دو خطأ	$\$$	$0 E 1$

## ۵.۱۷ روش تقدّم عملگر

- برای اینکه بتوانیم از این روش استفاده کنیم، باید گرامر عملگر باشد. شرایط عملگر بودن یک گرامر به شرح زیر است:
۱. فاقد قواعد تهی باشد.
  ۲. دو تا غیر پایانه در سمت راست قواعد، مجاور یکدیگر نباشند.

«مثال:

$$\begin{cases} E \rightarrow EAE \mid (E) \mid id \\ A \rightarrow + \mid * \end{cases}$$

عملگر نیست

حال باید گرامر غیر عملگر فوق را به گرامر عملگر تبدیل کنیم:

$$E \rightarrow E + E \mid E * E \mid (E) \mid id$$

عملگر هست

در کل ما در این روش میخواهیم به صورتی بین عملگرهای موجود، تقدّم قائل شویم.

\*\* توجه: تمام پایانه‌ها عملگر هستند.

تقدّم  $a$  کمتر از  $b$  است.  $a < b \not\Rightarrow b > a$

تقدّم  $a$  مساوی  $b$  است.  $a = b \not\Rightarrow b = a$

تقدّم  $a$  بیشتر از  $b$  است.  $a > b \not\Rightarrow b < a$

دوتابع زیر روی غیرپایانه‌ها تعریف می‌شوند و حاصل آنها مجموعه‌ای از پایانه‌ها است:

$$FirstTerm(A) = \{a \mid A \xrightarrow{+} a\alpha \text{ or } A \xrightarrow{+} B\alpha\} \quad \text{اولین چیز یا یکی بعد از آن}$$

$$LastTerm(A) = \{a \mid A \xrightarrow{+} \alpha a \text{ or } A \xrightarrow{+} \alpha aB\} \quad \text{آخرین چیز یا یکی قبل از آن}$$

$$a = b \text{ iff } \exists U \rightarrow \dots ab \dots \text{ or } U \rightarrow \dots aEb \dots$$

$$a < b \text{ iff } \exists U \rightarrow \dots aB \dots \text{ and } b \in FirstTerm(B) \quad a < FirstTerm(B)$$

$$a > b \text{ iff } \exists U \rightarrow \dots Ab \dots \text{ and } a \in LastTerm(A) \quad LastTerm(A) > b$$

## جدول پارس تقدّم عملگر:

$\text{Terminal} + \$$				
$\text{Terminal} + \$$				

ابعاد جدول پارس به روش تقدّم عملگر:  $(|T| + 1)^2$

« مثال: گرامر زیر را با روش تقدّم عملگر بررسی کنید.

- 0  $N \rightarrow \$E\$$
- 1,2  $E \rightarrow E + T \mid T$
- 3,4  $T \rightarrow T * F \mid F$
- 5,6  $F \rightarrow (E) \mid id$

پاسخ:

- 0  $N \rightarrow \$\textcircled{1}E\textcircled{2}\$$
- 1,2  $E \rightarrow E\textcircled{3} + \textcircled{4}T \mid T$
- 3,4  $T \rightarrow T\textcircled{5} * \textcircled{6}F \mid F$
- 5,6  $F \rightarrow (\textcircled{7}E\textcircled{8}) \mid id$

$$\text{FirstTerm}(E) = \{+, *, (), id\}$$

$$\text{FirstTerm}(T) = \{*, (), id\}$$

$$\text{FirstTerm}(F) = \{(), id\}$$

$$\text{LastTerm}(E) = \{+, *, (), id\}$$

$$\text{LastTerm}(T) = \{*, (), id\}$$

$$\text{LastTerm}(F) = \{(), id\}$$

- (1) :  $\$ < FirstTerm(E)$
- (2) :  $LastTerm(E) > \$$
- (3) :  $LastTerm(E) > +$
- (4) :  $+ < FirstTerm(T)$
- (5) :  $LastTerm(T) > *$
- (6) :  $* < FirstTerm(F)$
- (7) :  $(< FirstTerm(E)$
- (8) :  $LastTerm(E) > )$

$\{ \begin{matrix} \$ = \$ \\ (=) \end{matrix} \}$  روابط تساوی

\*\* توجه: هنگام جایگذاری در جدول، علامت کوچکتر ( $<$ ) به صورت سط्रی و علامت بزرگتر ( $>$ ) به صورت ستونی پر میشود.

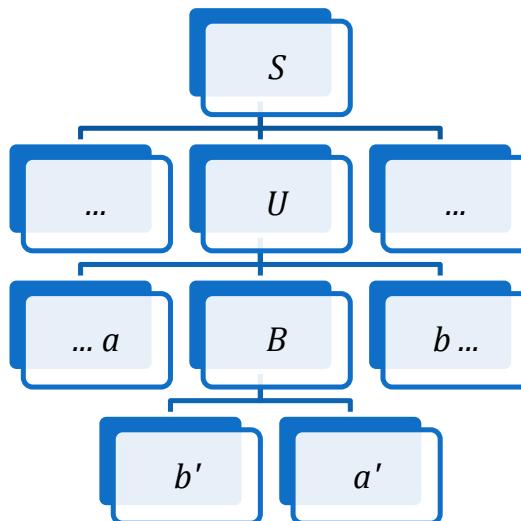
	+	*	(	)	<b>id</b>	\$
+	>	<	<	>	<	>
*	>	>	<	>	<	>
(	<	<	<	=	<	e <sub>2</sub>
)	>	>	e <sub>1</sub>	>	e <sub>1</sub>	>
<b>id</b>	>	>	e <sub>1</sub>	>	e <sub>1</sub>	>
\$	<	<	<	e <sub>3</sub>	<	=

شرح خطاهای:

- e<sub>1</sub> : عملگر گم شده است.
- e<sub>2</sub> : پرانتز بسته ")" گم شده است.
- e<sub>3</sub> : پرانتز بسته "(" اضافی.

توضیح: برای پارس به روش تقدّم عملگر، شرط لازم این است که گرامر عملگر باشد و شرط کافی این است که بین هیچ دو عملگری بیش از یک رابطه تقدّم نباشد. به عبارت دیگر یعنی در جدول تداخل وجود نداشته باشد. منظور از عملگرها همان پایانه ها هستند.

## ساختار درختی

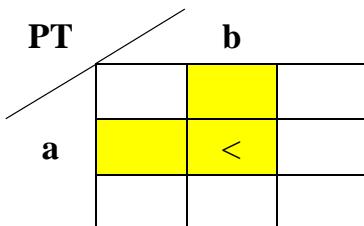


$$\begin{aligned}
 & a = b \\
 & a' < \text{FirstTerm}(B) \rightarrow a < b' \\
 & \text{LastTerm}(B) > b \rightarrow a' > b
 \end{aligned}$$

### ۵.۱۸ پارس به روش تقدّم عملگر

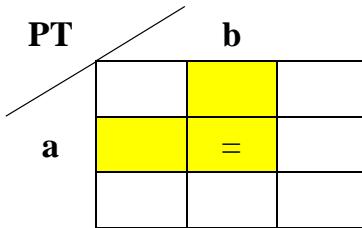
در ابتدا یک  $\$$  به انتهای رشته ورودی و یک  $\$$  داخل انباره می‌گذاریم. با مراجعه به بالاترین پایانه روی انباره (مثال  $a$ ) و Token جاری (مثال  $b$ ) و رفتن به سراغ خانه  $PT(a,b)$  در جدول پارس مراحل زیر را دنبال می‌کنیم:

۱. اگر  $PT(a,b)$  معادل علامت کوچکتر ( $<$ ) باشد، ابتدا علامت کوچکتر ( $<$ ) و سپس Token جاری (یعنی  $b$ ) را وارد انباره می‌کنیم. (عمل انتقال)



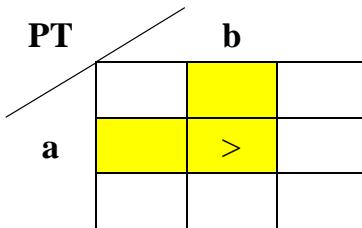
انباره	ورودی	عمل انجام شده
$\$ \dots a$	$b \dots \$$	انتقال
$\$ \dots a < b$	$\dots \$$	

۲. اگر  $PT(a,b)$  معادل علامت مساوی ( $=$ ) باشد،  $a$  وارد انباره  $b$  را جاری Token (یعنی  $b$ ) می‌کنیم. (عمل انتقال)

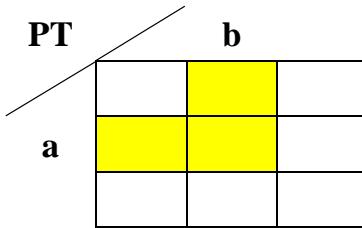


انباره	ورودی	عمل انجام شده
$\$ \dots a$	$b \dots \$$	انتقال
$\$ \dots ab$	$\dots \$$	

۳. اگر  $PT(a,b)$  معادل علامت بزرگتر ( $>$ ) باشد، داخل انباره تا رسیدن به اولین علامت کوچکتر پیش می‌رویم. رشته بالای این علامت و غیرپایانه آن در صورت وجود، Handle یا دستگیره است و از انباره حذف می‌شود و به جای آن یک غیرپایانه نمادین وارد انباره می‌کنیم. (عمل کاهش)



۴. اگر  $PT(a,b)$  خانه خالی باشد، یک خطای نحوی رخ داده است و رویه پردازش خطا فراخوانی می‌شود.



« مثال: عبارت  $id + id * id$  را پارس کنید.

	انباره	ورودی	عمل انجام شده
1	\$	$id + id * id $$	انتقال
2	$$ < id$	$+id * id $$	کاهش
3	$$ P$	$+id * id $$	انتقال
4	$$ P < +$	$id * id $$	انتقال
5	$$ P < + < id$	$* id $$	کاهش
6	$$ P < + P$	$* id $$	انتقال
7	$$ P < + P < *$	$id $$	انتقال
8	$$ P < + P < * < id$	\$	کاهش
9	$$ P < + P < * P$	\$	کاهش
10	$$ P < + P$	\$	کاهش
11	\$ P	\$	پذیرش

### معایب روش تقدّم عملگر:

۱. به دلیل محدودیتهایی که دارد، گرامرهای کمی وجود دارد که بتوانند از این روش استفاده کنند.
۲. عملگرهایی مثل "- " که دارای دو تقدّم متفاوتند، با این روش کار نمی‌کنند. مثل X- و Y-.
۳. روش چندان دقیقی نیست، چون ممکن است بعضی از خطاهای نحوی را کشف نکند.

### ۵.۱۹ روش تقدّم ساده

این روش بهبود یافته روش تقدّم عملگر است. در این روش بین تمامی علائم اعم از پایانه ها و غیرپایانه ها تقدّم تعریف می‌شود. شرط استفاده از این روش این است که قاعده تهی نداشته باشیم و سمت راست هیچ دو قاعده ای یکسان نباشد. ولی برخلاف روش تقدّم عملگر میتوانیم دو غیرپایانه در کنار هم داشته باشیم.

« مثال: در گرامر زیر به دلیل وجود عبارات یکسان در سمت راست قواعد تداخل کاهش-کاهش پیش می‌اید.

$$\begin{aligned} A &\rightarrow abc \\ B &\rightarrow abc \end{aligned}$$

دوتابع  $\text{Head}(A)$  و  $\text{Tail}(A)$  داریم که بر روی غیرپایانه‌ها تعریف می‌شوند و حاصل آنها مجموعه‌ای از علامت که همان پایانه‌ها و غیرپایانه‌ها می‌باشند، هستند.

$$\text{Head}(A) = \{X \mid A \xrightarrow{+} X\alpha\} \quad \text{چیزهایی که اول می‌باشد}$$

$$\text{Tail}(A) = \{X \mid A \xrightarrow{+} \alpha X\} \quad \text{چیزهایی که آخر می‌باشد}$$

«مثال:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid id$$

$$\text{Head}(E) = \{E, T, F, (, id\}$$

$$\text{Tail}(E) = \{T, F, (, id\}$$

$$\text{Head}(T) = \{T, F, (, id\}$$

$$\text{Tail}(T) = \{F, (, id\}$$

$$\text{Head}(F) = \{(, id\}$$

$$\text{Tail}(F) = \{ (, id\}$$

$$x \ominus y \text{ iff } \exists U \rightarrow \dots xy \dots$$

$$x \otimes y \text{ iff } \exists U \rightarrow \dots xB \dots \text{ and } y \in \text{Head}(B)$$

$$x \otimes y \text{ iff } \exists U \rightarrow \dots Ay \dots \text{ and } x \in \text{Tail}(A)$$

$$\text{or } \exists U \rightarrow \dots AB \dots \text{ and } x \in \text{Tail}(A) \text{ and } y \in \text{Head}(B)$$

يعنى:

$$\text{Tail}(A) \otimes y$$

$$\text{Tail}(A) \otimes \text{Head}(B)$$

\*\*\* توجه:  $x$  و  $y$  هم پایانه  $(\Gamma)$  و هم غیرپایانه  $(N)$  هستند.

\*\*\* توجه: به طور کلی هر دو علامتی که کنار هم باشند، مساوی هستند، چه  $T$  و چه  $N$

«مثال: گرامر زیر را با استفاده از روش تقدم ساده بررسی کنید.

$$0 \quad N \rightarrow \$ S \$$$

$$1 \quad S \rightarrow (SS)$$

$$2 \quad S \rightarrow c$$

پاسخ:

$$0 \quad N \rightarrow \$ \textcircled{1} S \textcircled{2} \$$$

$$1 \quad S \rightarrow (\textcircled{3} S \textcircled{4} S \textcircled{5})$$

$$2 \quad S \rightarrow c$$

$$\textcircled{1} : \$ \otimes \text{Head}(S)$$

$$\textcircled{2} : \text{Tail}(S) \otimes \$$$

$$\textcircled{3} : ( \otimes \text{Head}(S)$$

(4) :  $Tail(S) \otimes Head(S), S \otimes Head(S)$

(5) :  $Tail(S) \otimes )$

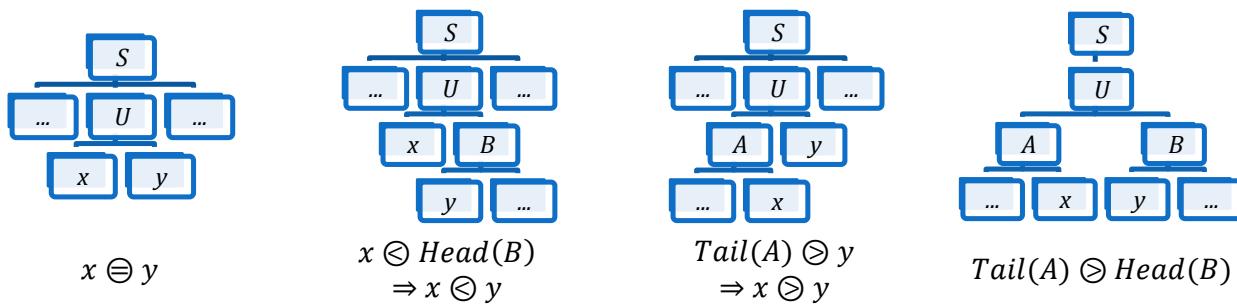
$$Tail(S) = \{ \ , c \} \quad Head(S) = \{ ( , c \ }$$

ابعاد جدول پارس به روش تقدّم ساده:  $(|T| + |N| + 1)^2$

	S	(	)	c	\$
S	$\ominus$	$\otimes$	$\ominus$	$\otimes$	$\ominus$
(	$\ominus$	$\otimes$		$\otimes$	
)		$\otimes$	$\otimes$	$\otimes$	$\otimes$
c		$\otimes$	$\otimes$	$\otimes$	$\otimes$
\$	$\ominus$	$\otimes$		$\otimes$	

\*\*\* توجه: علامت‌های  $\otimes$  (کوچکتر) به صورت سطحی پُر می‌شوند.

\*\*\* توجه: علامت‌های  $\ominus$  (بزرگتر) به صورت ستونی پُر می‌شوند.



## ۵.۲۰ پارس به روش تقدّم ساده

در این روش در هر قدم به بالاترین علامت انباره (مثالاً x) و Token جاری (ورودی) (مثالاً b) نگاه کرده و سراغ (PT(x,b) می‌رویم.

۱. گام اول: اگر PT(x,b) معادل علامت مساوی (=) باشد، Token جاری (یعنی b) را وارد انباره می‌کنیم. (عمل انتقال)
۲. گام دوم: اگر PT(x,b) معادل علامت کوچکتر (<) باشد، ابتدا علامت کوچکتر (<) و سپس b را وارد انباره می‌کنیم. (عمل انتقال)

۳. گام سوم: اگر PT(x,b) معادل علامت بزرگتر (>) باشد، در این صورت داخل انباره تا رسیدن به اولین علامت کوچکتر پیش می‌رویم. Handle یا دستگیره رشته بالای این علامت است. دستگیره را از بالای انباره حذف کرده و در صورت صحت از لحاظ نحوی باید با سمت راست یکی از قواعد مطابقت داشته باشد. اگر علامتی را که پس از حذف

دستگیره بالای انباره باقی می‌ماند را  $\text{Top}$  بنامیم و اگر سمت چپ قاعده‌ای که سمت راست آن با دستگیره تطبیق داشته را  $\text{LHS}$  بنامیم، رابطه  $\text{Top}$  با  $\text{LHS}$  را از روی جدول پیدا کرده و به صورت زیر عمل می‌کنیم:

۱.۳. حالت اول: اگر  $\text{Top} \ominus \text{LHS}$  باشد،  $\text{LHS}$  را وارد انباره می‌کنیم.

۲.۳. حالت دوم: اگر  $\text{Top} \otimes \text{LHS}$  باشد، ابتدا علامت کوچکتر ( $<$ ) و سپس  $\text{LHS}$  را وارد انباره می‌کنیم.

۳.۳. حالت سوم: اگر  $\text{Top}$  و  $\text{LHS}$  رابطه‌ای نداشته باشند، یک خطای نحوی رخ داده است.

\*\* تذکر: هیچ گاه  $\text{Top} \otimes \text{LHS}$  نمی‌شود.

۴. گام چهارم: اگر  $\text{PT}(x, b)$  خانه خالی باشد، یک خطای نحوی رخ داده است.

« مثال: عبارت  $((cc)c)$  را پارس کنید. »

	انباره	ورودی	عمل انجام شده
1	\$	$((cc)c)$ \$	انتقال
2	\$ $\otimes$ (	$(cc)c$ \$	انتقال
3	\$ $\otimes$ ( $\otimes$ (	$cc)c$ ) \$	انتقال
4	\$ $\otimes$ ( $\otimes$ ( $\otimes$ c	c ) c ) \$	$\text{Top} = ( , S \rightarrow c \otimes$ $LHS = S, Top \ominus LHS$ (LHS) (انتقال)
5	\$ $\otimes$ ( $\otimes$ ( S	c ) c ) \$	انتقال
6	\$ $\otimes$ ( $\otimes$ ( S $\otimes$ c	) c ) \$	$\text{Top} = S, S \rightarrow c \otimes$ $LHS = S, Top \ominus LHS$ (LHS) (انتقال)
7	\$ $\otimes$ ( $\otimes$ ( S S	) c ) \$	انتقال
8	\$ $\otimes$ ( $\otimes$ ( S S )	c ) \$	$\text{Top} = ( , S \rightarrow (SS) \otimes$ $LHS = S, Top \ominus LHS$ (LHS) (انتقال)
9	\$ $\otimes$ ( S	c ) \$	انتقال
10	\$ $\otimes$ ( S $\otimes$ c	) \$	$\text{Top} = S, S \rightarrow c \otimes$ $LHS = S, Top \ominus LHS$ (LHS) (انتقال)
11	\$ $\otimes$ ( S S	) \$	انتقال
12	\$ $\otimes$ ( S S )	\$	$\text{Top} = \$, S \rightarrow (SS) \otimes$ $LHS = S, Top \ominus LHS$ (LHS) (انتقال)
13	\$ S	\$	پذیرش

**۵.۲۱ چپ‌گردی**

$U \rightarrow U \dots$   
 $A \rightarrow \dots xU \dots$   
 $x \ominus U, x \otimes Head(U) \Rightarrow x \otimes U$  تداخل

**۵.۲۲ رفع چپ‌گردی**

$U \rightarrow U \dots$   
 $A \rightarrow \dots xN \dots$   
 $N \rightarrow U$   
 $x \ominus N, x \otimes Head(N) \Rightarrow x \otimes U$

**۵.۲۳ راست‌گردی**

$U \rightarrow \dots U$   
 $A \rightarrow \dots Ux \dots$   
 $U \ominus x, Tail(U) \otimes x \Rightarrow U \otimes x$  تداخل

**۵.۲۴ رفع راست‌گردی**

$U \rightarrow \dots U$   
 $A \rightarrow \dots Nx \dots$   
 $N \rightarrow U$   
 $N \ominus x, Tail(N) \otimes x \Rightarrow U \otimes x$

**۵.۲۵ تحلیل‌گر معنایی**

ایستا: چک کردن (تحلیل) در زمان ترجمه (Compile) صورت می‌گیرد.  
پویا: چک کردن (تحلیل) در زمان اجرا (Run) صورت می‌گیرد.

انواع چک کردن:

ایستا:

- چک کردن نوع (Type Checking)
- چک کردن ساختارهای تو در تو
- چک کردن یکتایی یا یکی بودن (Main)
- چک کردن کنترل جریان (Control Follow Checking)
- چک کردن پارامترها (Parameter Checking)

پویا:

- چک کردن محدوده (Range Checking)

## گروه آموزشی : مهندسی کامپیویلر - نرم افزار

مقطع تدریس: کارشناسی

کد درس: ۷۲۳۱

نام درس: اصول طراحی کامپیویلر

نوع درس از لحاظ شهریه:

نوع درس از لحاظ آکادمیک:

تعداد واحد: ۳

ساعات تشکیل کلاس در هفته:

هر هفتاهای تدریس	طرح درس طبق سرفصل
هفته اول	مقدمات، مفاهیم ترجمه و مترجم، انواع مترجم‌ها شامل کامپیویلر، مفسر، اسپلیتر و پیش‌پردازنده. مقایسه مفسر و کامپیویلر. معرفی کلی ساختار و اجزاء کامپیویلر شامل: تحلیل گر لغوى، تحلیل گر معنای، تولید گر معنای، تولید کد میانی، بهینه سازی کد، تولید کد نهایی، مدیریت جدول نمادها، مدیریت خطاهای
هفته دوم	بررسی انواع گرامرها و خواص عمومی زبان‌ها، طبقه‌بندی چامسکی، مرور کلی بر درس نظریه زبان‌ها و ماشین‌ها، مفاهیم درخت اشتراق، بسط چپ‌ترین (LMD)، بسط راست‌ترین (RMD)، مفاهیم ابهام، زبان‌های مبهم، گرامر مبهم، روش‌های رفع ابهام، گرامرها مختص و مفید، گرامرها متعادل، فرم جمله‌ای
هفته سوم	تحلیل لغوى، تعریف و مفهوم توکن (Token)، عبارات منظم برای نمایش الگوی توکن‌ها، دیاگرام گذر، وظایف تحلیل گر لغوى، گرامرها نفسیز حالت قطعی و غیرقطعی، روش‌های برخورد با خطاهای لغوى، اصلاح خطاهای لغوى
هفته چهارم	تحلیل نحوی، روش‌های تحلیل نحوی شامل تحلیل یا تجزیه بالا به پایین و تحلیل پایین به بالا، معرفی روش‌های اصلاح خطاء در تحلیل گر نحوی، معرفی و تعریف روش‌های تحلیل بالا به پایین، معرفی پارسرهای پیشگوی، معرفی روش Recursive Descent
هفته پنجم	تحلیل به روش LL(1)، مفاهیم First و Follow، پارسرا (LR(1))
هفته ششم	نحوه تشکیل جدول پارس (SLR)، شرایط LL(1) بودن یک گرامر، اصلاح خطاء در روش LL(1)
هفته هفتم	مفاهیم تحلیل پایین به بالا، معرفی و بیان ویژگی‌های روش انتقال کاهش، انواع تداخل‌ها در روش انتقال کاهش، تداخل انتقال-کاهش و تداخل کاهش-کاهش
هفته هشتم	روش‌های LR، مفهوم آیتم (LR(0)، تعریف پیکربندی، بررسی روش SLR، نحوه تشکیل جدول SLR
هفته نهم	شرایط SLR بودن، انجام عملیات تجزیه به روش SLR، بررسی تداخل‌ها در روش SLR
هفته دهم	تجزیه به روش CLR، نحوه تشکیل جدول و رسم دیاگرام در روش CLR.

۷۲۳۱

کد درس:

نام درس: اصول طراحی کامپایلر

طرح درس طبق سرفصل	هفته‌های تدریس
تجزیه به کمک روش LALR، نحوه تشکیل جدول در روش LALR، مقایسه روش‌های LR، نحوه برخورد با خطأ و اصلاح خطأ در روش‌های LR	هفته یازدهم
روش تقدم عملگر، نحوه تشکیل جدول در این روش، تعریف First Term و Last Term. تحلیل به روش تقدم عملگر	هفته دوازدهم
تشریح روش تقدم ساده، نحوه تعریف تقدم، تجزیه به روش تقدم ساده، جداول توابع (تتابع تقدم)	هفته سیزدهم
تحلیل معنایی، بررسی ایستا و پویا، انواع نسبت ایستا و پویا، وظایف تحلیل‌گر معنایی، مدیریت جدول نمادها، روش‌های تخصیص حافظه	هفته چهاردهم
تولید کد میانی، انواع کد میانی، مختصه در مورد کد نهایی	هفته پانزدهم
بهینه سازی کد میانی، تحلیل جریان داده و تحلیل جریان کنترل	هفته شانزدهم

منابع اطلاعاتی:

1. Aho, Lam, Sethi and Ullman, "Compilers: Principles, Techniques, and Tools", 2nd Edition, 2007.
2. Tremblay and Sorenson, "The Theory and Practice of Compiler Writing", McGraw-Hill, 1985.
3. Pittman and Peters, " The Art Of Compiler Design: Theory And Practice", Prentice Hall, 1992.
4. R. Mak, "Writing Compilers and Interpreters: An Applied Approach Using C++", 2nd Ed., John Wiley, 1996.