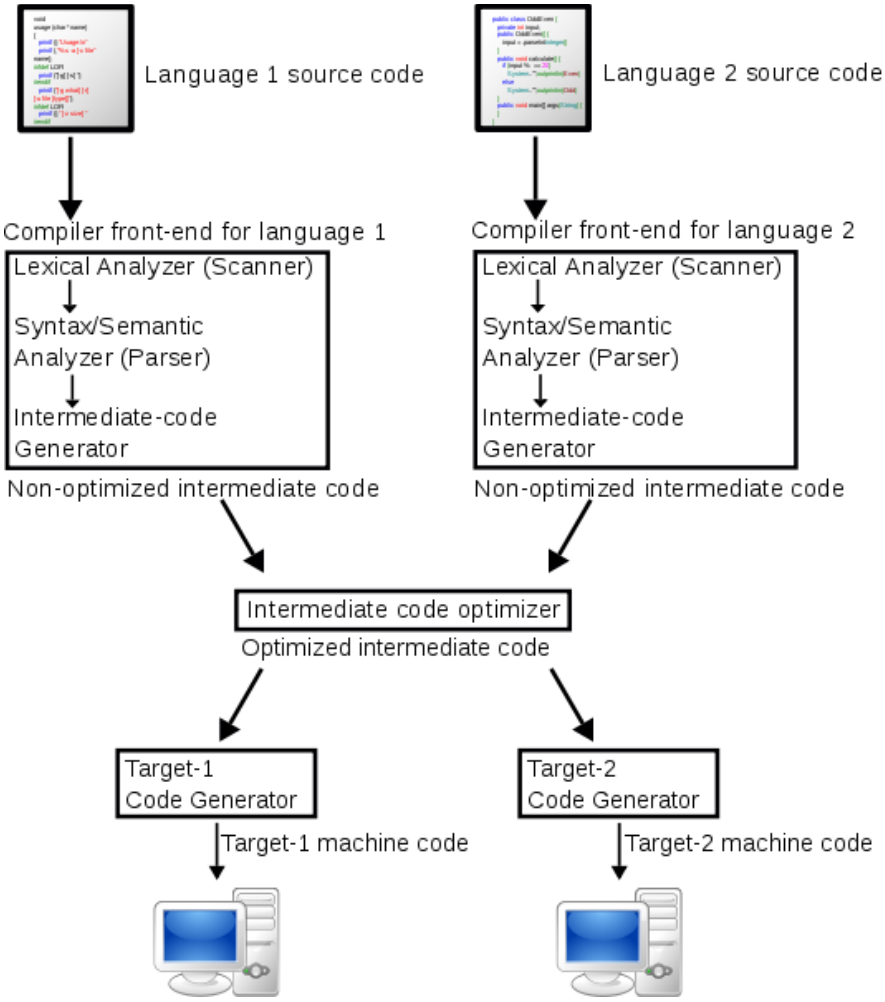
اصول طراحی کامپایلر



جواد شاهپریان

گروه مهندسی کامپیوتر

دانشکده فنی دانشگاه آزاد اسلامی واحد تهران جنوب

مهرماه ۱۳۹۱

فهرست مطالب

۱ فصل اول : مقدمات

۱,۱ انواع مترجم

۱. کامپایلر (Compiler)
۲. مفسر (Interpreter)
۳. اسمبلر (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 اسمبلر

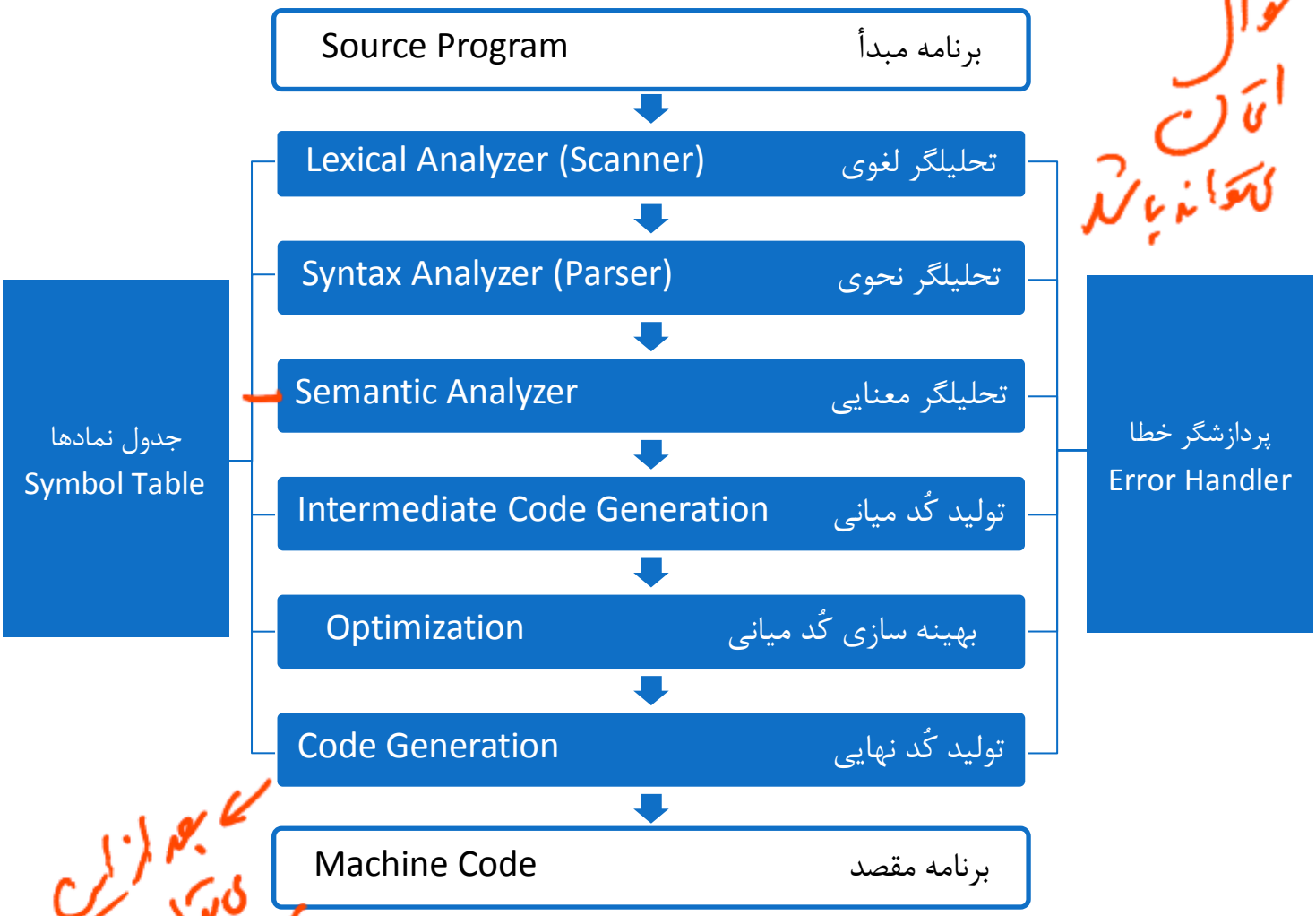
مترجمی است که زبان اسمبلی را به زبان ماشین تبدیل میکند.

۱,۴ تفاوت مفسر و کامپایلر

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

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

۱,۵ بخشهای مختلف یک کامپایلر

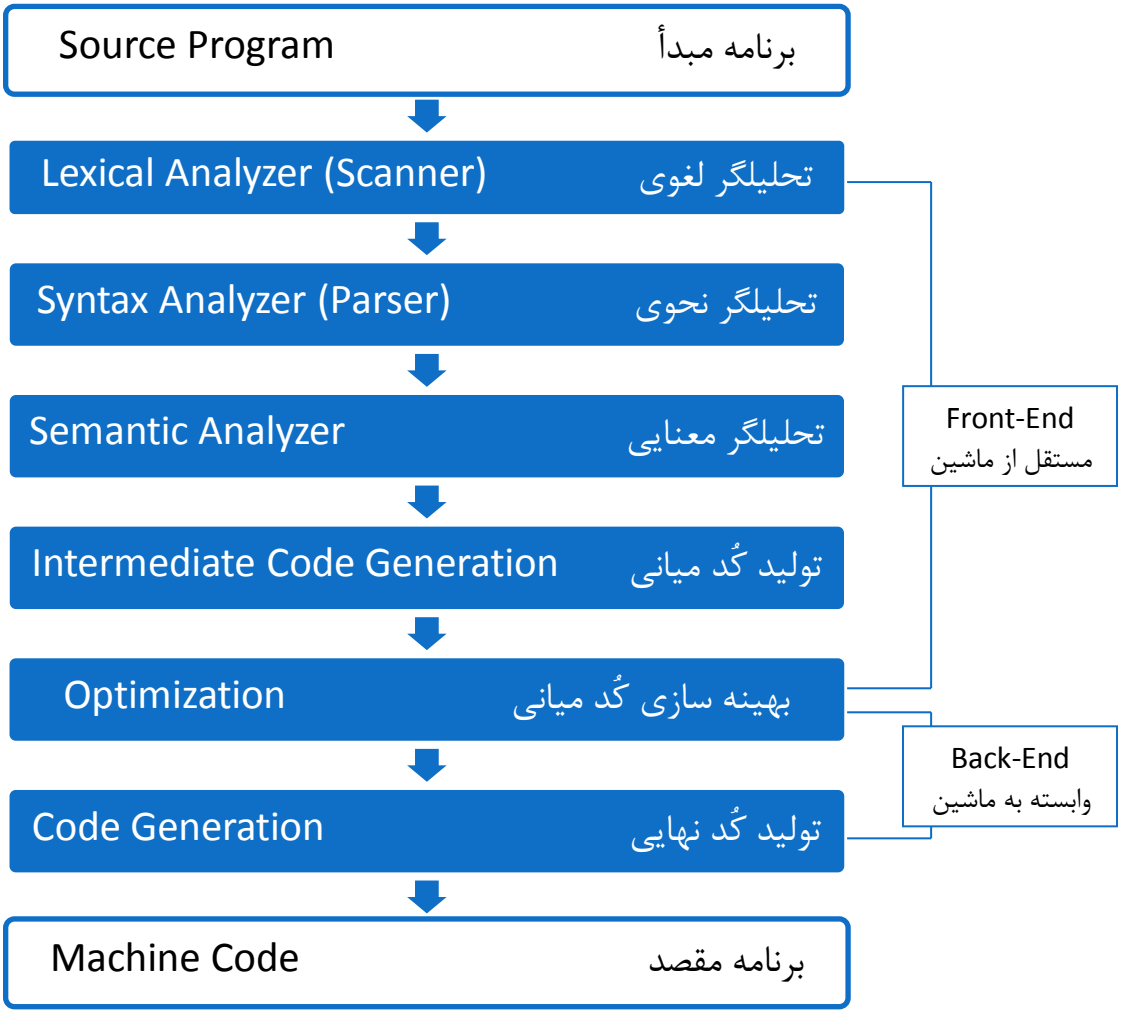


۱,۶ پردازش‌گر خطا (Error Handler)

قسمت دیگری که در همه مراحل درگیر است، پردازش‌گر خطا 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 خطا توسط کامپایلر.

خطای لغوی (بین int و a فاصله وجود ندارد)	inta;
خطای نحوی (در آخر عبارت، ' ; ' وجود ندارد)	int a
خطای معنایی (قرار دادن مقدار اعشاری در متغیر نوع صحیح)	int a = 5.5;
خطای معنایی (دستیابی به اندیس آرایه خارج از محدوده)	int a[10]; a[12] = 3;

» 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 خطا میشوند:

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

» مثال: بررسی مراحل کامپایل.

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

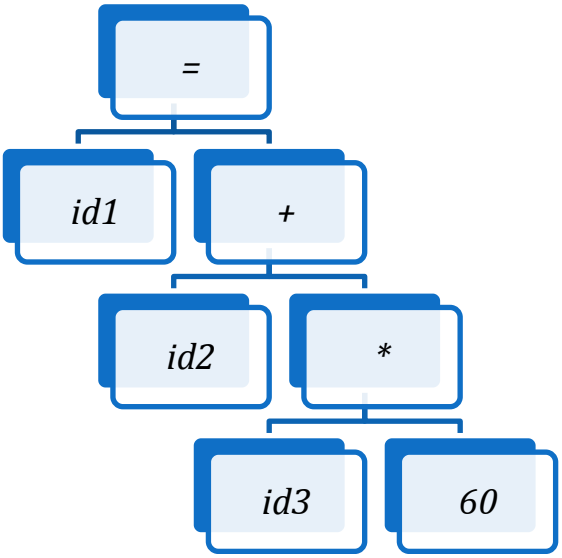
ابتدا تحلیلگر لغوی وارد عمل شده و تک تک کلمات را از هم جدا کرده (Tokenize) و بررسی میکند. سپس در جدول نمادها قرار میدهد:

Symbol Table

شناسه	اسم	نوع
id1	p	float
:::	:::	:::

$p = i + k * 60\ ; \rightarrow\ id1 = id2 + id3 * 60\ ;$

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 (صحیح) به 60.0 (اعشاری) تبدیل شود:

$$int\ to\ float\ (60) \rightarrow 60.0$$

سپس کُد میانی زیر ایجاد میشود:

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

حال کُد به دست آمده بهینه سازی میشود:

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

در انتها نیز کُد 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 = < N,T,S,P >

N: Non-Terminals	(A,B,C,…)	غیر پایانه ها (حروف بزرگ)
T: Terminals	(a,b,c,…)	پایانه ها (حروف کوچک)
S: Start Symbol	$\{S \mid S \in N\}$	نماد شروع
P: Production	$\{\alpha \rightarrow \beta\}$	تولیدات، قواعد

۲,۲ Term

به هر کدام از اجزای گرامر یک Term گفته میشود.

۲,۳ پایانه ها (Terminal)

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

۲,۴ غیر پایانه ها (Non Terminal)

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

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 (Uninon)

مجموعه رشته های متعلق به دو زبان که یا متعلق به زبان اوّل یا متعلق به زبان دوّم هستند.

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

۲,۱۲ الحاق (Concat)

$$M.N = \{ST \mid S \epsilon M \ \wedge T \epsilon N\}$$

تعریف L^* : رشته هایی به طول صفر یا بیشتر از زبان L
تعریف L^+ : رشته هایی با طول یک یا بیشتر از زبان L
تعریف L^n : رشته هایی با طول n از زبان L

$$L^0 = \{\varepsilon\}$$
$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots$$
$$L^+ = L^1 \cup L^2 \cup \dots$$

۲,۱۳ گرامرهای معادل

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

G1:

$E \rightarrow T + E \mid T - E \mid T$
$T \rightarrow F * T \mid F/T \mid F$
$F \rightarrow ID \mid D \mid (E)$
$ID \rightarrow a \mid b$
$D \rightarrow 0 \mid 1$

G2:

$E \rightarrow E + T \mid T - E \mid T$
$T \rightarrow T * F \mid F/T \mid F$
$F \rightarrow ID \mid D \mid (E)$
$ID \rightarrow a \mid b$
$D \rightarrow 0 \mid 1$

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 α ، β ، γ و ... مجموعه ای از پایانه ها و غیر پایانه ها هستند.

$$\underbrace{\alpha}_{LHS} \rightarrow \underbrace{\beta}_{RHS}$$

» مثال:

$$\begin{cases} 1: A \rightarrow aB \\ 2: A \rightarrow ef \end{cases} \} \; 1,2: A \rightarrow aB \mid ef$$

۲,۱۵ انواع گرامر

۱. گرامر نوع صفر (نامحدود) «» قوی ترین
۲. گرامر نوع یک (حساس به متن)
۳. گرامر نوع دو (مستقل از متن)
۴. گرامر نوع سه (منظم) «» ضعیف ترین

- قوانین گرامر نوع صفر:

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

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

- قوانین گرامر نوع یک:

سمت چپ باید کوچکتر یا مساوی سمت راست باشد.

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

- قوانین گرامر نوع دو:

اندازه سمت چپ باید یک باشد.

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

- قوانین گرامر نوع سه:

منظم 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 توجّه به قواعد داده شده، abc را تولید کنید.

$S \to AB$
$A \to ab$
$B \to c$

» پاسخ:

LMD: $S \to \boldsymbol{A}B \to ab\boldsymbol{B} \to abc$
RMD: $S \to A\boldsymbol{B} \to \boldsymbol{A}c \to abc$

۲,۱۷ فرم جمله ای

به هر کدام از فرم‌هایی که در مراحل اشتقاق ایجاد میشود، یک فرم جمله ای گویند. به عبارت دیگر رشته‌ای از پایانه‌ها و غیرپایانه ها که در مراحل اشتقاق ایجاد می‌شود.

$\{\alpha \dashv \mid S \to \alpha, \alpha \in (N \cup T)$

۲,۱۸ استنتاج

تبدیل یک فرم جمله ای با استفاده از قواعد زبان به یک فرم جمله ای دیگر را استنتاج گویند.

۲,۱۹ انواع استنتاج

۱. استنتاج مستقیم ($\Rightarrow$)

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

۲. استنتاج نهایی ($\overset{+}{\Rightarrow}$)

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

۳. استنتاج کلّی ($\overset{*}{\Rightarrow}$)

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

» مثال: نمونه استفاده از استنتاج در مثال قبل

$S \Rightarrow AB$ استنتاج مستقیم

$S \overset{+}{\Rightarrow} abB$ استنتاج نهایی

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

1, 2 $< Number > \rightarrow < Number > < Digit > | < Digit >$

3,...,12 $< Digit > \rightarrow 0 \mid 1 \mid 2 \mid ... \mid 9$

» پاسخ:

LMD:

$$
\begin{aligned}
< Number > &\overset{1}{\Rightarrow} < \boldsymbol{Number} > < Digit > \\
&\overset{1}{\Rightarrow} < \boldsymbol{Number} > < Digit > < Digit > \\
&\overset{2}{\Rightarrow} < \boldsymbol{Digit} > < Digit > < Digit > \\
&\overset{5}{\Rightarrow} \quad 2 \quad < \boldsymbol{Digit} > < Digit > \\
&\overset{6}{\Rightarrow} \quad 2 \quad 3 \quad < \boldsymbol{Digit} > \\
&\overset{4}{\Rightarrow} \quad 2 \quad 3 \quad 1
\end{aligned}
$$

RMD:

$$
\begin{aligned}
< Number > &\overset{1}{\Rightarrow} < Number > < \boldsymbol{Digit} > \\
&\overset{4}{\Rightarrow} < \boldsymbol{Number} > \quad 1 \\
&\overset{1}{\Rightarrow} < Number > < \boldsymbol{Digit} > \quad 1 \\
&\overset{6}{\Rightarrow} < \boldsymbol{Number} > \quad 3 \quad 1 \\
&\overset{2}{\Rightarrow} < \boldsymbol{Digit} > \quad 3 \quad 1 \\
&\overset{5}{\Rightarrow} \quad 2 \quad 3 \quad 1
\end{aligned}
$$

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؟

1 $E \rightarrow E + E$

2 $E \rightarrow E * E$

3 $E \rightarrow (E)$

4 $E \rightarrow id$

1,...,4 $E \rightarrow E + E \mid E * E \mid (E) \mid id$

حالا میخواهیم $id + id * id$ را تولید کنیم.

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

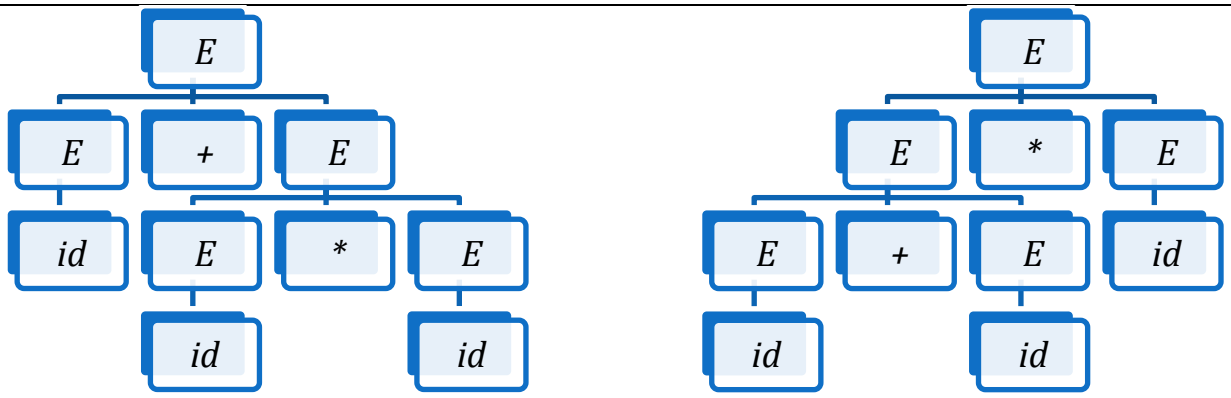
LMD 1:

$$E \overset{1}{\Rightarrow} E + E \overset{4}{\Rightarrow} id + E \overset{2}{\Rightarrow} id + E * E \overset{4}{\Rightarrow} id + id * E \overset{4}{\Rightarrow} id + id * id$$

LMD 2:

$$E \overset{2}{\Rightarrow} E * E \overset{1}{\Rightarrow} E + E * E \overset{4}{\Rightarrow} id + E * E \overset{4}{\Rightarrow} id + id * E \overset{4}{\Rightarrow} id + id * id$$

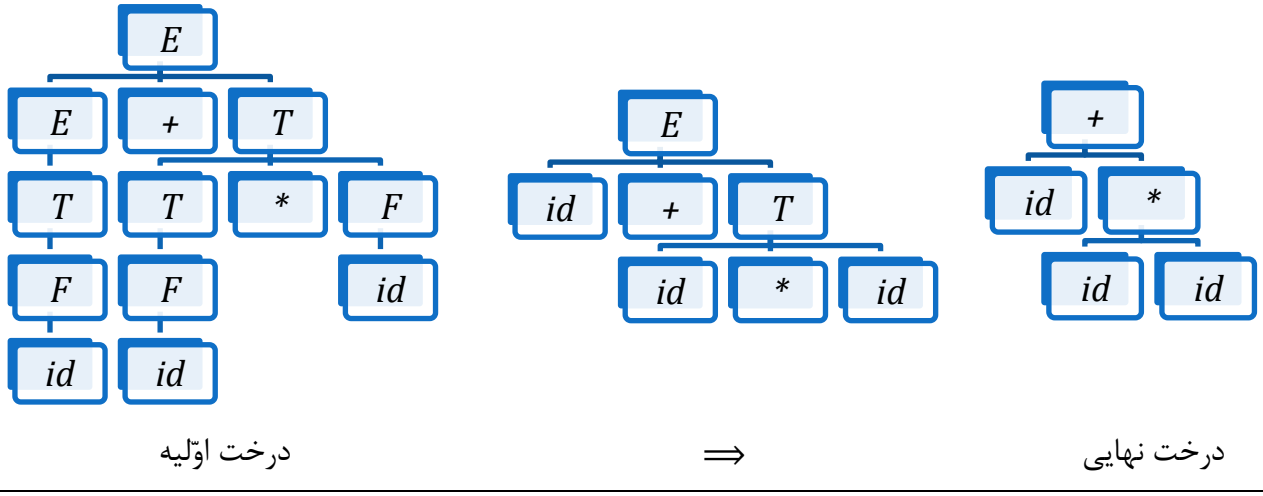
درخت نحوی گرامر فوق به صورت زیر است:



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

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

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



۲,۲۱ گرامرهای مختصر و مفید

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

۱. در آن قواعدی به فرم $A \rightarrow A$ وجود نداشته باشد.
۲. در آنها تمام غیر پایانه ها فعال (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 برسیم.

** نکته: ترتیب بررسی شرط‌ها مهم است.

» مثال: گرامر معادل مختصر و مفید را برای عبارت زیر بنویسید.

$A \rightarrow Ba$
$B \rightarrow b$
$B \rightarrow Cb$
$C \rightarrow Cb$
$C \rightarrow DdC$
$D \rightarrow \varepsilon$

» پاسخ:

$A \rightarrow Ba$
$B \rightarrow b$

» مثال:

$S \rightarrow aaa$
$S \rightarrow Abaaa$
$A \rightarrow Ab$
$A \rightarrow aBa$
$B \rightarrow aBa$
$B \rightarrow AC$
$C \rightarrow eb$
$C \rightarrow b$

» پاسخ:

$S \rightarrow aaa$

۲,۲۲ عبارتهای منظم

هر عبارتی با صفر و یک به غیر از تُهی	$(0\|1)^+ = 0, 1, 01, \dots$
هر عبارتی با صفر و یک حتی تُهی	$(0\|1)^* = \varepsilon, 0, 1, 01, \dots$
هر عبارتی که با ۱۱ شروع شود و با ۰۱۱ خاتمه یابد و بین آن دو هر ترکیبی از صفر و یک باشد	$11(0\|1)^*011$
	$(ab\|c)^+ = \cdots, abc, cab, \dots, abcabab, \dots$
نمایش ارقام	$d = 0, 1, 2, \dots$
نمایش عدد صحیح	d^+
نمایش عدد اعشاری	$d^+.d^+$
نمایش عدد صحیح یا اعشاری	$d^+(.d^+\|\varepsilon) \Leftrightarrow d^+ (.d^+)?$
نمایش عدد علامتدار (مثبت یا منفی)	$(+\|-)d^+$
نمایش عدد علامتدار یا بدون علامت (مثبت یا منفی)	$(+\|-\|\varepsilon)d^+ \Leftrightarrow (+\|-)?\, d^+$
نمایش اعداد اعشاری با نماد علمی	$(+\|-)?\, d^+(.d^+)?\,((+\|-)?\, ed^+)?$

** 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:

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

$L(L|D)^*$

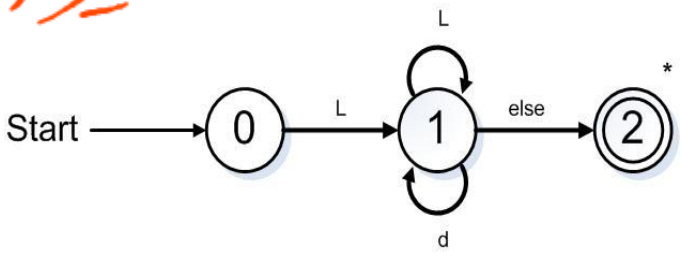
$1 \quad < ID > \rightarrow < L > |< ID >< L >| < ID >< D >$

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

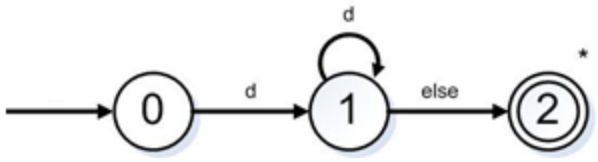
$28..38 < D > \rightarrow \ 0 \mid 1 \mid \dots \mid 9$

دیاگرام گذر شناسه فوق به صورت زیر است:

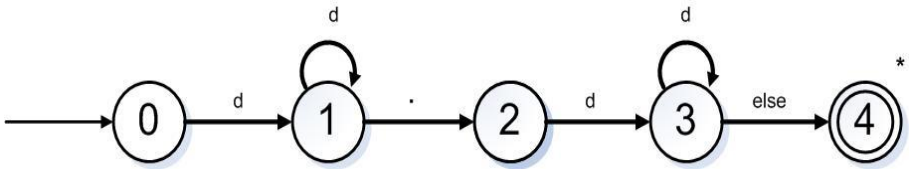
$L(L|D)^*$



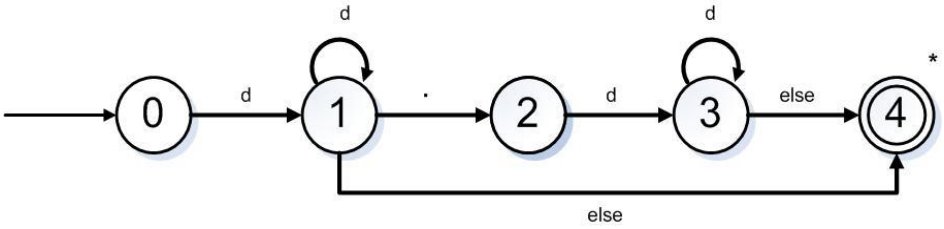
» مثال: دیاگرام گذر d^+



» مثال: دیاگرام گذر $d^+.d^+$



» مثال: دیاگرام گذر $d^+(.d^+)?$



۳ فصل سوم : تحلیلگر لغوی

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 جداکننده‌ها و تعیین نوع آنها. {مهمترین وظیفه}

۳,۲ جدا کننده ها (Separator)

جداکننده‌ها دو دسته هستند:

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

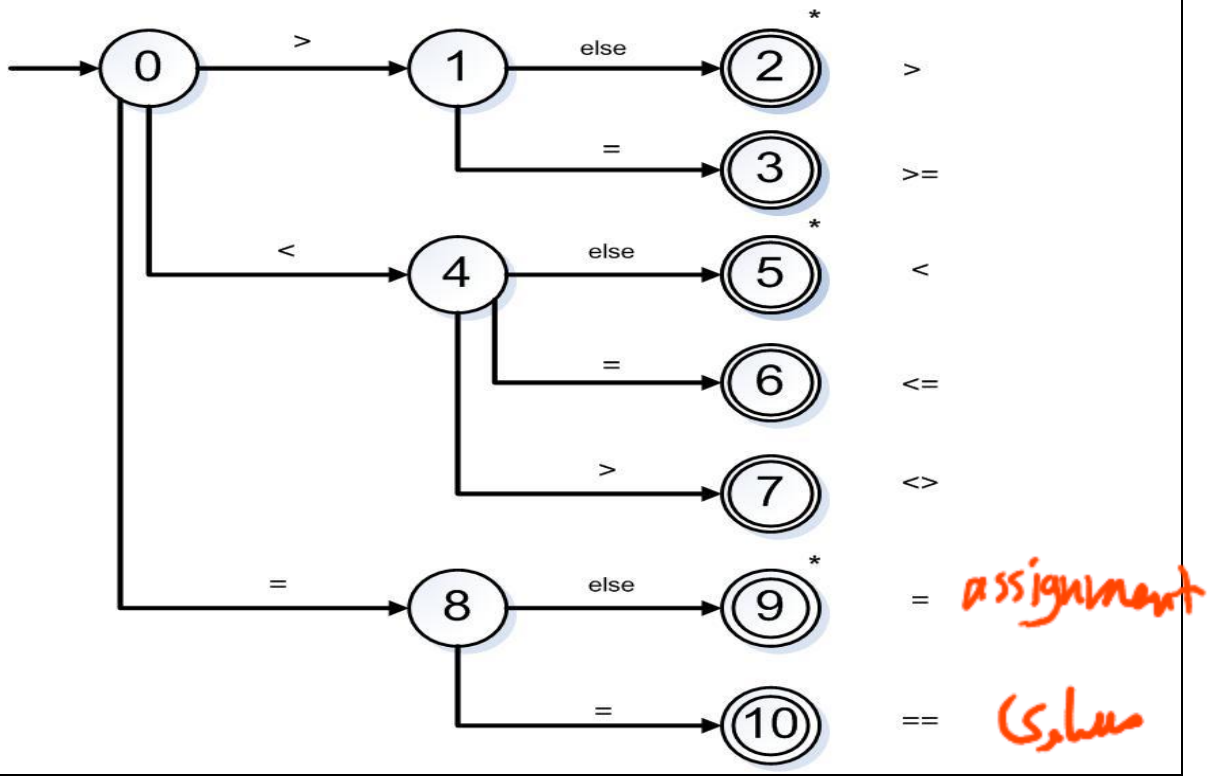
۳,۳ انواع موجود در زبان (انواع کلمات)

۱. شناسه‌ها
۲. جداکننده‌ها
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

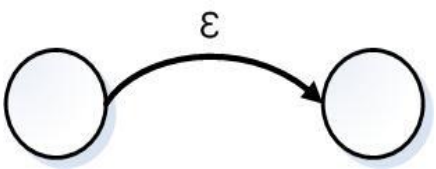


۳,۵ ماشینهای خودکار (Automata)

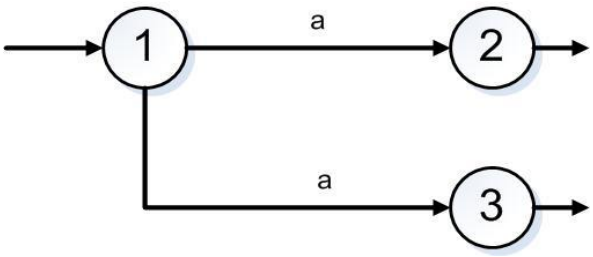
گراف‌های جهت داری هستند که برای پیاده سازی قواعد لغوی زبانها بسیار مناسب هستند. این ماشین ها دو نوع هستند: DFA (قطعی یا معین) و NFA (غیرقطعی یا نامعین).

دو ویژگی که DFA را از NFA جدا میکند:

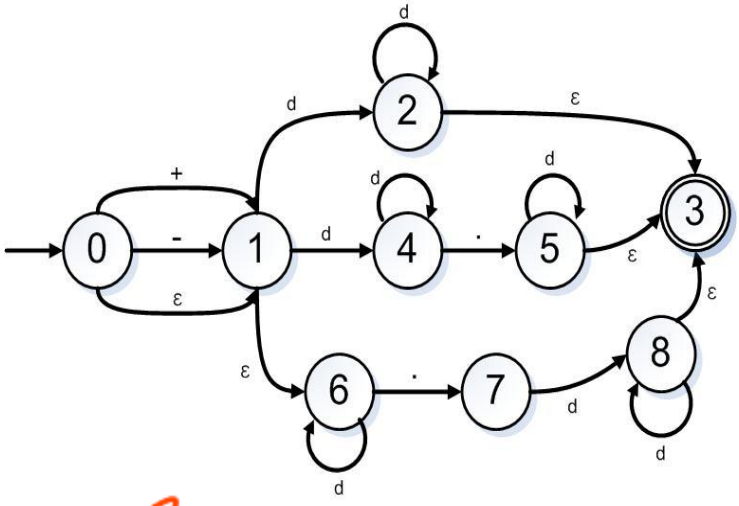
۱. در DFA برچسب تهی نداریم.



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



شکل زیر یک NFA است:



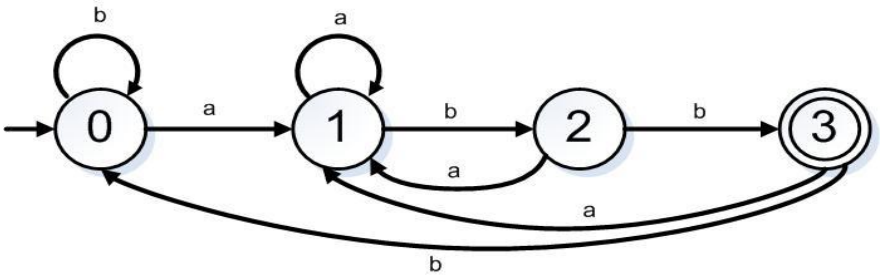
$$(+|-|\varepsilon)\begin{cases} d^+ \\ d^+.d^* \\ d^*.d^+ \end{cases}$$

تمرین انجام شود → NFA, DFA

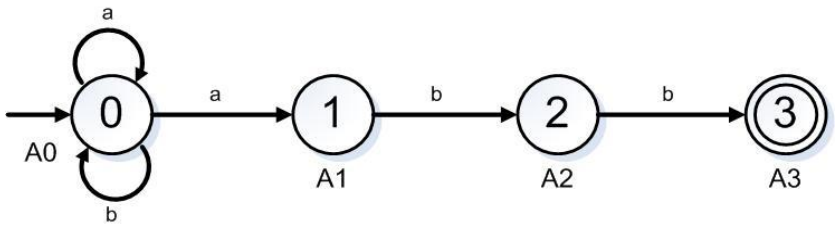
کامپایلرها برای کامپایل از DFA استفاده میکنند. چون NFA به چند حالت میرود و کامپایلر گیج میشود.

» مثال: میخواهیم عبارتی تولید کنیم که به abb ختم شود.

» پاسخ: عبارت منظم مثال فوق برابر است با $(a|b)^*abb$

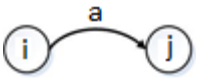


شکل فوق یک DFA برای گرامر بالا است. اگر بخواهیم یک NFA برای عبارت بالا داشته باشیم، به صورت زیر عمل میکنیم:



۳,۶ الگوریتم تبدیل یک دیاگرام به گرامر معادلش

۱. به هرکدام 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، ε بود، آنگاه $A_i \rightarrow A_j$ به گرامر اضافه میکنیم.

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

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

» مثال: گرامر معادل دیاگرام مثال قبل:

$A_0 \rightarrow aA_0$
$A_0 \rightarrow bA_0$
$A_0 \rightarrow aA_1$
$A_1 \rightarrow bA_2$
$A_2 \rightarrow bA_3$
$A_3 \rightarrow \varepsilon$

$A_0 \rightarrow aabb \mid babb \mid abb$

۴ فصل چهارم : روشهای تحلیل نحوی

۱۵ نمره از این فصل سوال می آید

۴,۱ 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. اشتقاق (Derivation)
۲. درخت نحو (Syntax Tree)

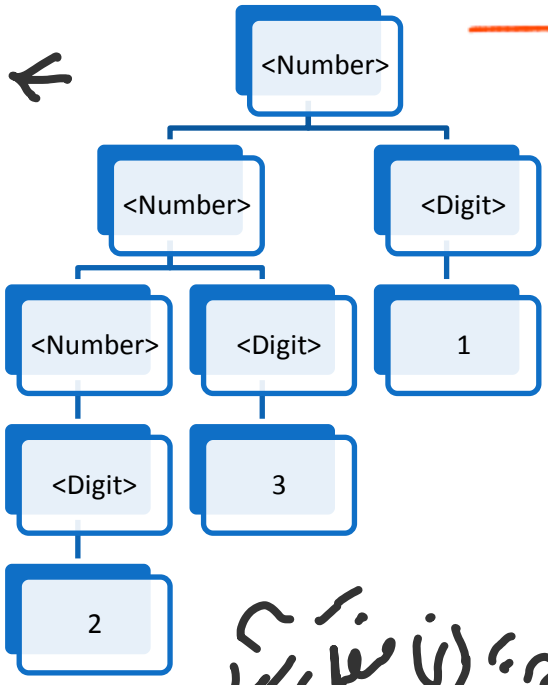
۴,۲ درخت نحوی (Syntax Tree)

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.

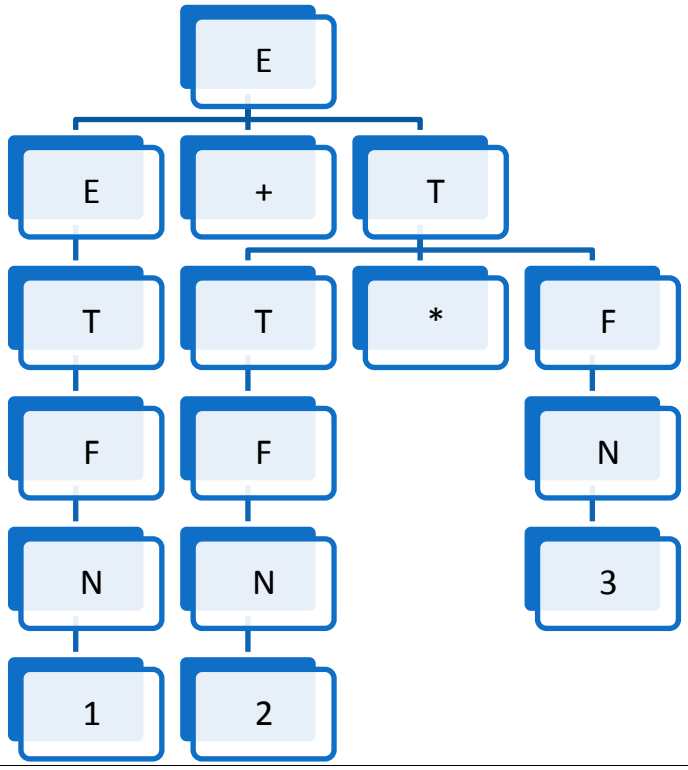
** نکته: ترتیب استفاده از قواعد هم از روی درخت مشخص نیست. درختی که در این مرحله تولید میشود برای تحلیلگر معنایی و بقیه مراحل نیز کاربرد دارد.

» مثال: بررسی کنید که عبارت $3 * 2 + 1$ جزو قواعد و گرامر زبان هست یا نه؟



گرامر رفع ابهام شده:

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow N \mid (E)$
$N \rightarrow 1 \mid 2 \mid 3$



۴,۳ 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.

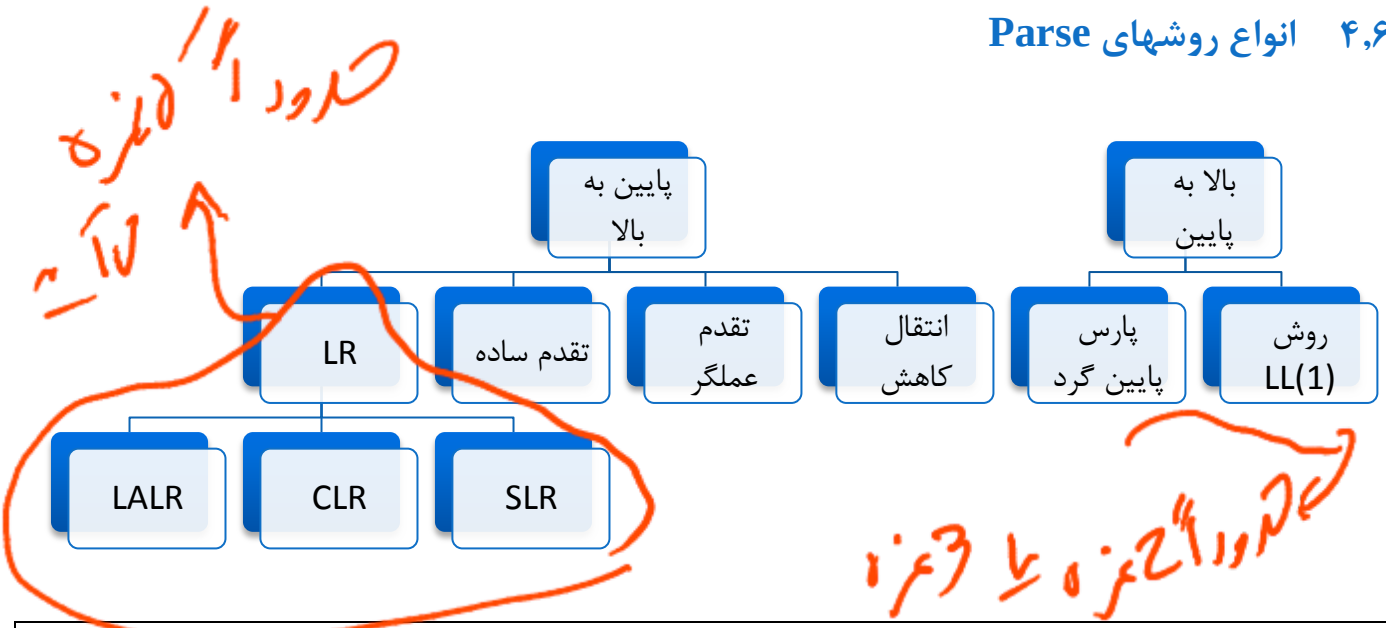
۴,۴ وظیفه پارسر

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

۴,۵ گذر (Pass)

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

۴,۶ انواع روشهای Parse



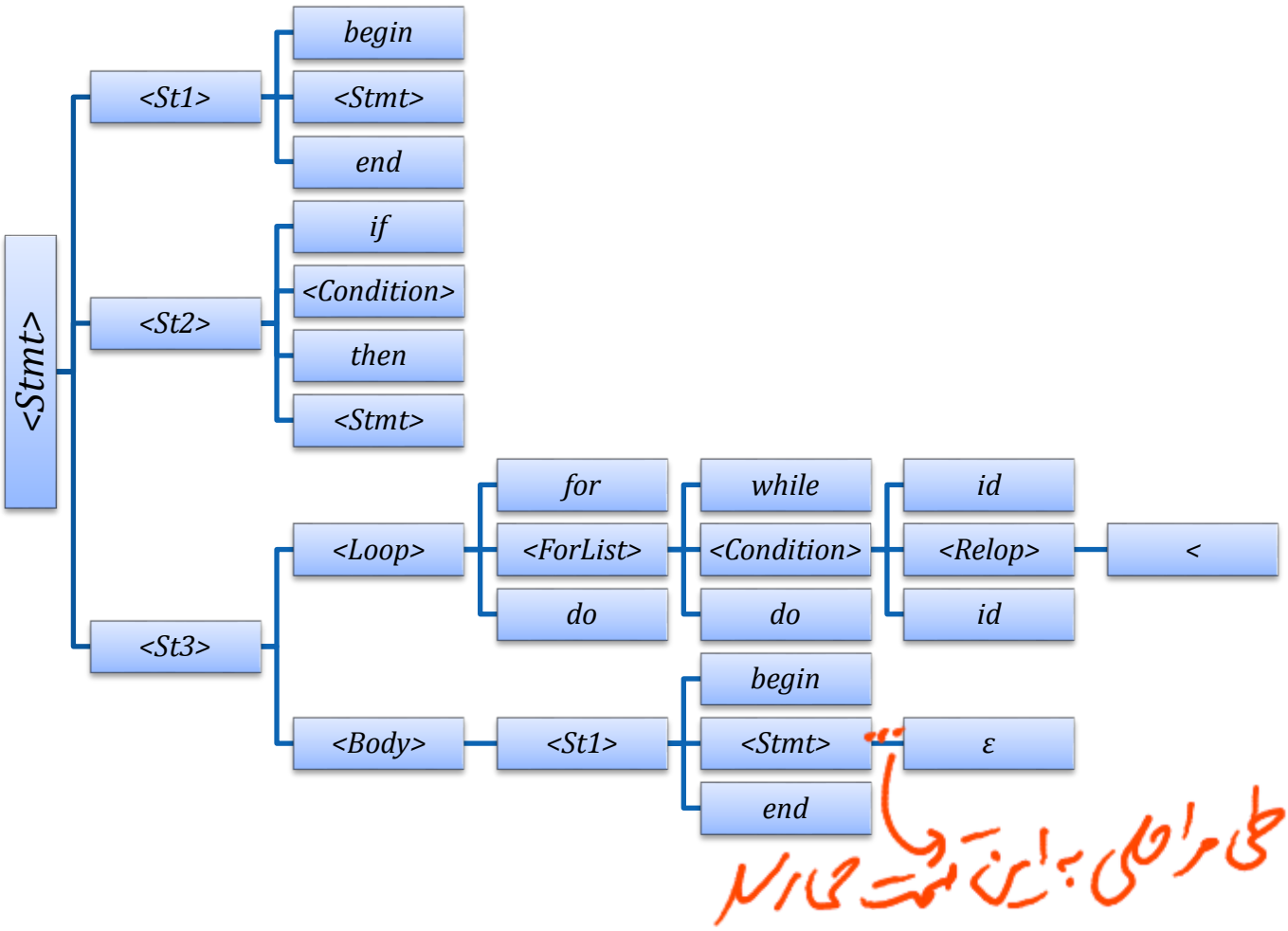
» مثال: با توجّه به گرامر داده شده، بررسی کنید که آیا عبارت $While\ i < j\ Do\ Begin\ End$ از لحاظ نحوی درست است یا خیر؟

1..4 $< Stmt > \rightarrow < St1 > | < St2 > | < St3 > | \varepsilon$
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 $< ForList > \rightarrow id := num\ to\ num$
12 $< Condition > \rightarrow id < Relop > id$
13..15 $< Reloop > \rightarrow > | < | =$

» پاسخ: می‌خواهیم بررسی کنیم با توجّه 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 توجّه 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 برخورد، 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 آن ساخته می شود **نماد پیشگویی** (Look Ahead) می‌گویند و به صورت مخفف با L.A. نشان میدهند.

۴,۸ تابع First

First (N) اولین پایانه ای است که غیر پایانه N میتواند ببیند.

رشته α: $First(\alpha) = \{a | \alpha \overset{*}{\Rightarrow} a\beta\}$

** نکته: خروجی تابع First مجموعه ای از پایانه‌هاست.

** نکته: در تابع First هیچگاه $ نداریم.

۴,۹ مراحل به دست آوردن First (x)

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

» مثال: با توجّه به گرامر زیر First(A) را بیابید:

$A \rightarrow BCDa$
$B \rightarrow b \mid \varepsilon$
$C \rightarrow e \mid \varepsilon$
$D \rightarrow f \mid \varepsilon$

» پاسخ:

$First(A) = First(BCDa) = First(B) - \{\varepsilon\} + First(C) - \{\varepsilon\} + First(D) - \{\varepsilon\} + \{a\}$
$First(B) = \{b, \varepsilon\}$
$First(C) = \{e, \varepsilon\}$
$First(D) = \{f, \varepsilon\}$
➔ $First(A) = \{b, e, f, a\}$

** تذکر: اگر در $A \rightarrow BCDa$ ، a نبود، به رشته حاصل ε اضافه میکردیم.

۴,۱۰ تابع Follow

پایانه‌ای که بعد از غیرپایانه می‌تواند دیده شود.

$Follow(A) = \{b | S \overset{*}{\Rightarrow} \alpha Ab\beta\}$

** نکته: خروجی تابع Follow مجموعه ای از پایانه‌هاست.
** نکته: در تابع Follow هیچگاه ε نداریم.

۴,۱۱ مراحل یافتن Follow (A)

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

» مثال: Follow تمام غیر پایانه های گرامر زیر را بیابید.

0 $\acute{A} \to A\$$
1 $A \to BCDa$
2,3,4 $B \to CD \mid b \mid \varepsilon$
5,6 $C \to e \mid \varepsilon$
7,8 $D \to f \mid \varepsilon$

» پاسخ:

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

۴,۱۲ پارس به روش LL(1)

- L اول (Left): یعنی عبارت ورودی از چپ به راست بررسی میشود.
- L دوم (LMD): اشتقاق چپ ترین
- (۱): یعنی نماد پیشگویی ما یک عنصر است (تک عضوی).

البته روش LL(1) قابل تعمیم به LL(2)، LL(3) تا LL(k) نیز هست، ولی کاربرد کمتری دارد.

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

$A \to abc$
$B \to ade$

۴,۱۳ جدول پارس LL(1)

در ستونهای جدول، پایانه ها قرار میگیرند به اضافه $ و در سطرهای جدول، غیر پایانه ها قرار میگیرند.

اندازه جدول : تعداد غیر پایانه ها * (تعداد پایانه ها + ۱)

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

	پایانه ها + $				
غیر پایانه ها					

۴,۱۴ نحوه تشکیل جدول پارس LL (1)

۱. برای کلیه قواعد غیر تهی به فرم $A \rightarrow \alpha$ ، شماره این قاعده را مقابل غیرپایانه A و زیر عناصر مجموعه First(α) مینویسیم.

۲. اگر α، طی صفر مرحله یا بیشتر به ε 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 $< ST > \rightarrow < Stmt > \$$

خط ۰ را به ابتدای مثال اضافه میکنیم.

» پاسخ: First را برای همه غیر پایانه ها و Follow را برای غیر پایانه‌هایی که به تهی ختم میشوند، محاسبه میکنیم:

$First(Stmt) = First(St1) + First(St2) + First(St3) + \{\varepsilon\}$ ◄

$First(St1) = \{begin\} = First(Body)$

$First(St2) = \{if\}$

$First(St3) = First(Loop) = \{for, while\}$

$First(ForList) = \{id\}$

$First(Condition) = \{id\}$

$First(Relop) = \{>, <, =\}$

$Follow(Stmt) = \{\$, end\}$

1..4 $< Stmt > \rightarrow < St1 > | < St2 > | < St3 > | \varepsilon$
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 $< ForList > \rightarrow id := num\ to\ num$
12 $< Condition > \rightarrow id < Relop > id$
13..15 $< Reloop > \rightarrow > | < | =$

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

« مثال: برای گرامر زیر جدول پارس تشکیل دهید.

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

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

$First(E) = First(T) = First(F) = \{\, (\,, id \,\}$
$First(E') = \{\, + \,, \boldsymbol{\varepsilon} \,\}$ ◄
$First(T') = \{\, * \;, \boldsymbol{\varepsilon} \,\}$ ◄

برای کشیدن جدول پارس باید Follow غیر پایانه‌هایی که First آنها به ε ختم میشوند، را هم بدست بیاوریم:

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

	id	**+**	*****	***(***	***)***	***$***
E	1			1		
E'		2			**3**	**3**
T	4			4		
T'		**6**	5		**6**	**6**
F	8			7		

۴,۱۵ پارس با استفاده از روش LL (1)

** نکته: پارسر در هر مرحله به Token بالای انباره (Stack) و عنصر اوّل 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 میابد.

- **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 اشتقاق 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 می کند که دو حالت ممکن است پیش بیاید:

۱. حالت اول: اگر $PT_{X,L.A.} = X \rightarrow \alpha$ باشد، X را از بالای انباره حذف و به جای آن α را به صورت معکوس وارد انباره میکنیم.
۲. حالت دوم: اگر $PT_{X,L.A.}$ خالی باشد، یک خطای نحوی رخ داده است.

» مثال: آیا عبارت $id + id * id$ با پارسر LL(1) از لحاظ نحوی درست است؟

» 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.

Stack

	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	$\$ E'T'\sout{id}$	$\sout{id} + id * id \$$	نظیر شدن id
5	$\$ E'T'$	$+id * id \$$	بسط با قاعده ۶
6	$\$ E'$	$+id * id \$$	بسط با قاعده ۲
7	$\$ E' T\sout{+}$	$\sout{+}id * id \$$	نظیر شدن +
8	$\$ E' T$	$id * id \$$	بسط با قاعده ۴
9	$\$ E' T'F$	$id * id \$$	بسط با قاعده ۸
10	$\$ E' T'\sout{id}$	$\sout{id} * id \$$	نظیر شدن id
11	$\$ E' T'$	$* id \$$	بسط با قاعده 5
12	$\$ E' T' F\sout{*}$	$\sout{*}id \$$	نظیر شدن *
13	$\$ E' T' F$	$id \$$	بسط با قاعده ۸
14	$\$ E' T'\sout{id}$	$\sout{id} \$$	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: LL(1) بودن گرامر زیر را بررسی کنید.

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

$First(E) = \{a, e\}$
$First(F) = \{f, \varepsilon\}$ ◄
$First(A) = \{g\}$

$Follow(E) = \{\$\} + First(F) + Follow(E) + Follow(F)$
$Follow(F) = Follow(E)$
$Follow(A) = \{b\}$
$\rightarrow Follow(E) = \{\$, f\} = Follow(F)$

	a	*b*	*e*	*f*	*g*	*$*
E	1		2			
F				3 **4**		**4**
A					5	

با توجّه به جدول، این گرامر LL(1) نیست (یعنی با روش LL(1) قابل پارس کردن نیست). چون در یک خانه دو حالت وجود دارد.

۴,۱۶ شرایط LL (1) بودن یک گرامر

دو روش وجود دارد:

۱. روش اول: کشیدن جدول پارس LL(1)

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

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

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

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

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

» مثال: آیا گرامر زیر LL(1) است؟

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

» پاسخ:

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

پس LL(1) هم نیست. تداخل دارد.

		a
A		1 2

» مثال: آیا گرامر زیر LL(2) است؟

$A \to abA \mid adC$

» 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؟

$A \to B \mid C$
$B \to ab$
$C \to ad$

» پاسخ: LL(1) نیست، چون First مشترک دارند.

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

$A \to \alpha \mid \beta \mid \gamma \mid ... \mid \varepsilon$
$\begin{cases} First(\alpha) \cap First(\beta) \cap ... = \emptyset \\ First(A) \cap Follow(A) = \emptyset \end{cases}$

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

$A \to \alpha \mid \beta \mid \gamma \mid ... \mid \varepsilon$
$\alpha \to \varepsilon$
$\beta \to \varepsilon$

** یادآوری: حروف یونانی رشته‌اند، یعنی میتوانند ترکیبی از پایانه ها و غیر پایانه ها باشند.

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

0 $S'' \to S\$$
1,2 $S \to iEtSS' \mid a$
3,4 $S' \to eS \mid \varepsilon$
5 $E \to b$

	First	Follow
S	$\{i, a\}$	$\{\$, e\} = \{\$\} + First(s') + Follow(S) + Follow(S')$
S'	$\{e, \varepsilon\}$	$\{\$, e\}$
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 First مشترک 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:

$A \rightarrow a\boldsymbol{E} \mid d$
$\boldsymbol{E} \rightarrow B \mid C$
$B \rightarrow g \mid l$
$C \rightarrow e$

به طور کلی در روش فاکتورگیری برای رشته‌های دلخواه α و β داریم:

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

و پس از فاکتورگیری داریم:

$A \rightarrow aB$
$B \rightarrow \alpha \mid \beta$

» مثال: پارسر LL(1) را برای گرامر زیر تشکیل دهید.

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

» پاسخ:

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

همچنین قاعده ۱۵ و ۱۶ (<Parameters-List>) هم دارای First مشترک id هستند.

9 $< St4 > \rightarrow id < B >$
10 $< B > \rightarrow := < Stmt > \mid (< Parameters - List >)$

15 $< Parameters - List > \rightarrow id < E >$
16 $< E > \rightarrow , (< Parameters - List >) \mid \varepsilon$

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:

$A \rightarrow Ab$
$A \rightarrow Ab \rightarrow Abb$

» مثال:

$E \rightarrow Ea \mid b$
$E \rightarrow Ea \rightarrow Eaa \rightarrow Eaaa \dots a \mid \boldsymbol{b}aaa \dots a$
$First(Ea) \cap First(b) = \{b\}$

۴,۱۹ حذف چپ‌گردی

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

با حذف چپ‌گردی داریم:

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

» مثال: چپ گردی گرامر زیر را در صورت وجود حذف کنید.

$$E \rightarrow Ed \mid EA \mid g$$
$$A \rightarrow b \mid c$$

» پاسخ:

$$E \rightarrow gE'$$
$$E' \rightarrow dE' \mid AE' \mid \varepsilon$$
$$A \rightarrow b \mid c$$

» مثال:

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

» پاسخ:

$$E \rightarrow (E)\, E' \mid id\, E'$$
$$E' \rightarrow +EE' \mid *\, EE' \mid \varepsilon$$

۴,۲۰ چپ‌گردی غیر صریح

سوال نمره بیار ایمان

گرامر زیر را در نظر بگیرید:

$$E \rightarrow Ab \mid c$$
$$A \rightarrow Ed \mid g$$

ابتدا آن را به چپ‌گردی صریح تبدیل میکنیم:

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

پس از حذف چپ‌گردی:

$$E \rightarrow gbE' \mid cE'$$
$$E' \rightarrow dbE' \mid \varepsilon$$

۵ فصل پنجم : 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 کُد را کامپایل کند و در پایان، لیست خطاها 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 توجّه به محلی که خطا رخ داده و چیزی که در ورودی دیده می شود و چیزی که انتظار دارد ببیند، یک حدس میزند و بر اساس آن حدس، تغییری در برنامه ایجاد میکند تا خطا را رفع کند. اگر حدس درست باشد، پیغام مناسب صادر میشود. پس خطا رفع شده و عبور میکند.

» مثال:

$x = a + b * c$	عبارت صحیح مورد نظر
$x = a \quad b * c$	عبارت ورودی اشتباه
$x = ab * c$	حدس اشتباه

** 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 خطا، خطای آبشاری گفته میشود.

۵,۴ روش Error Production

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

مزیّت این روش:

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

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

$E \to F + T$
:::
$\boxtimes\rangle\ E \to FT$

۵,۵ روش Global Correction

یک روش تصحیح عمومی است. روشهای قبلی به صورت محلی به خطا نگاه میکردند. 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.

این روش باید ۳ خصوصیت مهّم داشته باشد:

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

** نکته: معمولاً پارسرهای بالا به پایین از دو روش اوّل استفاده میشود.

۵,۶ اصلاح خطا به روش Panic Mode در پارسرهای 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. امّا 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 نباشد. اگر 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 پارسر LL(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 $E' \rightarrow \varepsilon$
4 $T \rightarrow FT'$
5 $T' \rightarrow * FT'$
6 $T' \rightarrow \varepsilon$
7 $F \rightarrow (E)$
8 $F \rightarrow id$

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

$First(E) = First(T) = First(F) = \{\ (, id\ \}$
$First(E') = \{\ +, \boldsymbol{\varepsilon}\ \}$
$First(T') = \{\ *\ , \boldsymbol{\varepsilon}\ \}$

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

سپس جدول پارس LL(1) را تشکیل میدهیم:
** توجّه: Synch عناصر مجموعه هماهنگ ساز است.

	id	**+**	*****	***(***	***)***	***$***
E	1			1	Synch	Synch
E'		2			**3**	**3**
T	4	Synch		4	Synch	Synch
T'		**6**	5		**6**	**6**
F	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 – استثنا 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	$\$ E' T' ~~id~~$	$~~id~~ * + id \$$	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	$\$ E' T' ~~id~~$	$~~id~~ \$$	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

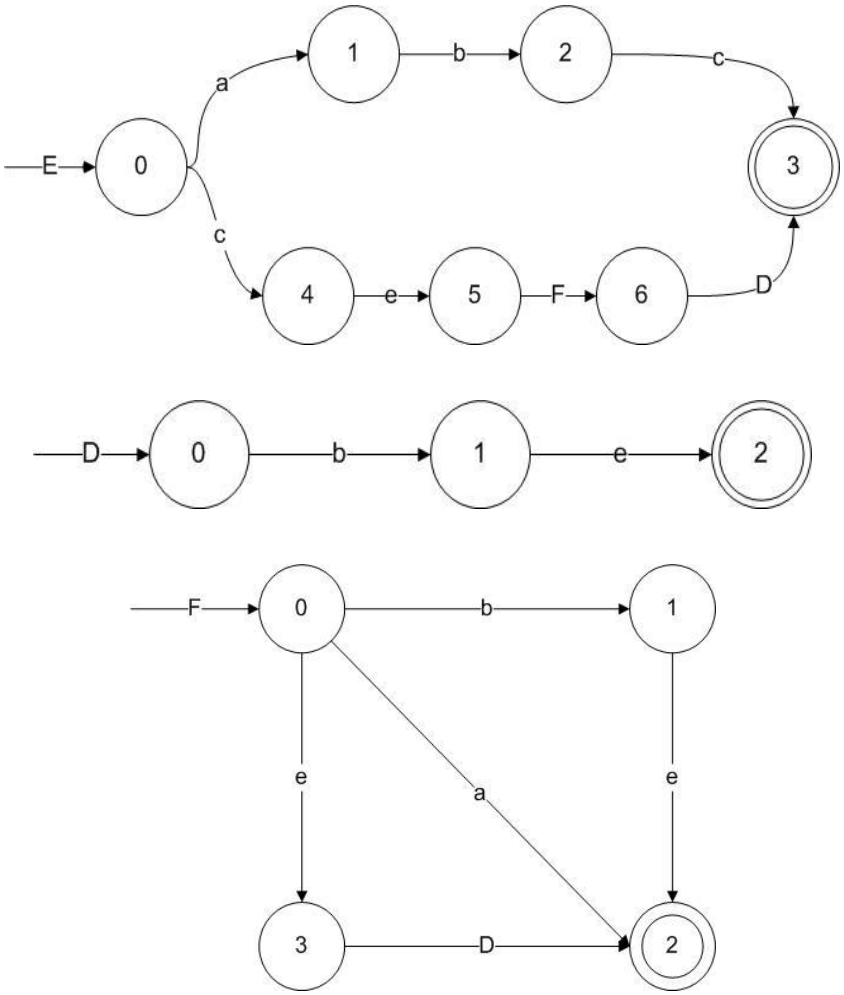
در این روش به ازای هر کدام از غیر پایانه ها یک تابع داریم.

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 قاعده ای که برای بسط انتخاب شده در ورودی پیدا کنیم (توسط پارسر انجام میشود). اگر سمت راست شامل پایانه و غیرپایانه باشد، به ازای پایانه ها 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:

1,2 $E \rightarrow abc \mid ceFD$
3 $D \rightarrow be$
4,5,6 $F \rightarrow a \mid be \mid eD$

دیاگرام گذر گرامر پیشین، به صورت زیر است:



شبه 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 $< Type > \rightarrow < Simple > | < Array\_Def >$

3,4,5 $< Simple > \rightarrow int | real | char$

6 $< Array\_Def > \rightarrow array\ [num \ldots num]\ of\ < Type >$

» 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( ] );
        Match( of );
        Type( );
}
```

** نکته: روش پارس پایین گرد به‌سادگی قابل پیاده‌سازی است، امّا 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 کُند خواهد شد.

۵,۸ پارسرهای پایین به بالا

عکس پارسرهای بالا به پایین عمل میکنند. فرض میکنیم جمله ای که میخواهیم بدانیم متعلق به زبان است، وجود دارد و هرکدام 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 پارسرها 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 پایین به بالا	پارسرهای بالا به پایین
$	$S
:	:
:	:
$S	$

۵,۹ روش انتقال-کاهش (Shift-Reduce)

- **تعریف انتقال**

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

- **تعریف کاهش**

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

- **تعریف دستگیره**

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

- **تعریف عبارت**

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

$$\alpha A\beta \overset{+}{\Rightarrow} \alpha\delta\beta$$
$$A \overset{+}{\Rightarrow} \delta$$

- **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 ساده ای است که در جهت عکس بسط سمت راست ترین (عکس عمل RMD) به وجود آمده باشد.

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

1 $S \rightarrow aABe$
2,3 $A \rightarrow Abc \mid b$
4 $B \rightarrow d$

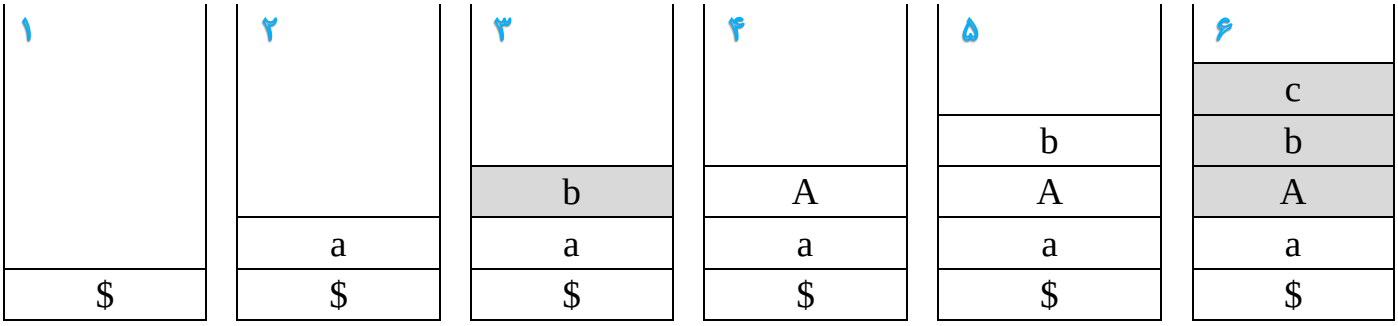
» پاسخ:

RMD:

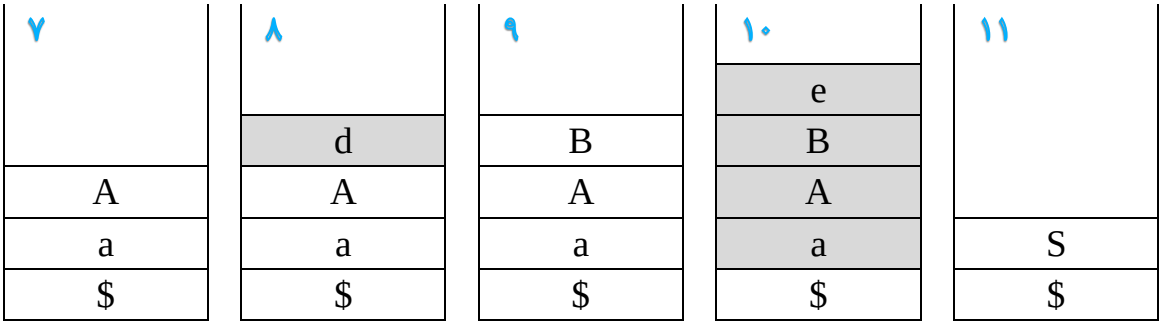
$S \overset{1}{\Rightarrow} aABe \overset{4}{\Rightarrow} aAde \overset{2}{\Rightarrow} aAbcde \overset{3}{\Rightarrow} abbcde$

RMD عکس عمل:

$S \overset{1}{\Leftarrow} aABe \overset{4}{\Leftarrow} aAde \overset{2}{\Leftarrow} aAbcde \overset{3}{\Leftarrow} abbcde$



انتقال a | انتقال b | کاهش با قاعده ۳ | انتقال b | کاهش با قاعده ۳ ✓ انتقال c | کاهش با قاعده ۲



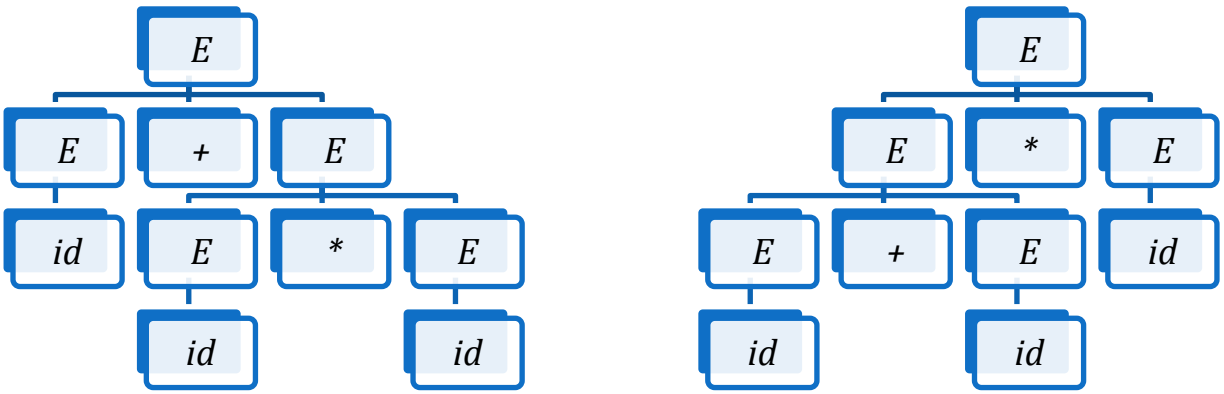
انتقال d | کاهش با قاعده ۴ | انتقال e | کاهش با قاعده ۱ | پذیرش

** 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	$ **id**	$+id * id$ $	کاهش با قاعده ۴ (R_4)
3	$ E	$+id * id$ $	انتقال +
4	$ $E +$	$id * id$ $	انتقال id
5	$ $E + id$	$* id$ $	کاهش با قاعده ۴ (R_4)
6	$ $E + E$	$* id$ $	تداخل: (**انتقال ***) یا (R_1)
7	$ $E + E *$	id $	انتقال id
8	$ $E + E *$ **id**	$	کاهش با قاعده ۴ (R_4)
9	$ $E +$ **E** $*$ **E**	$	کاهش با قاعده ۲ (R_2)
10	$ **E** $+$ **E**	$	کاهش با قاعده ۱ (R_1)
11	$ E	$	پذیرش

۵,۱۰ انواع تداخل

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

» مثال: تداخل انتقال - کاهش

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

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

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

» مثال: تداخل کاهش-کاهش

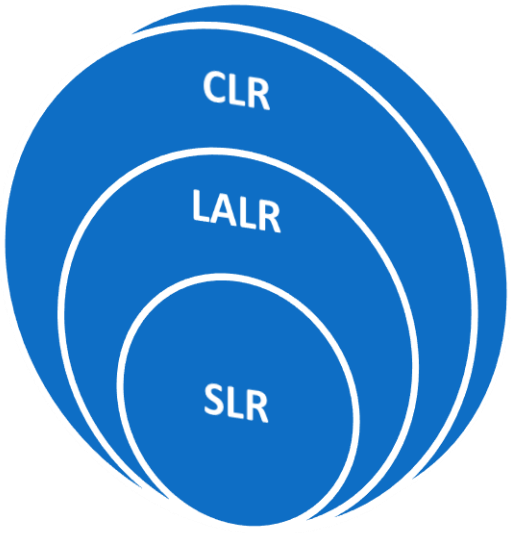
1 $A \rightarrow abc$
2 $B \rightarrow abc$

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

۵,۱۱ روشهای LR

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

۱. SLR (از همه ساده تر)
۲. CLR (از همه قوی تر)
۳. LALR (بین دو مورد بالا)



۵,۱۲ چهار مزیت اصلی روشهای LR

۱. کلّی 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 گرامرهایی که توسط روشهای LR پارس میشوند، یک مجموعه کامل از گرامرهایی هستند که توسط انواع پارسرهای دیگر پوشش داده میشود.
۴. خطاها را از همه پارسرهای دیگر سریعتر و صحیح‌تر پیدا می‌کنند.

۵,۱۳ LR(0).item

سمت راست یک قاعده یک نقطه (.) می‌افزاییم.

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

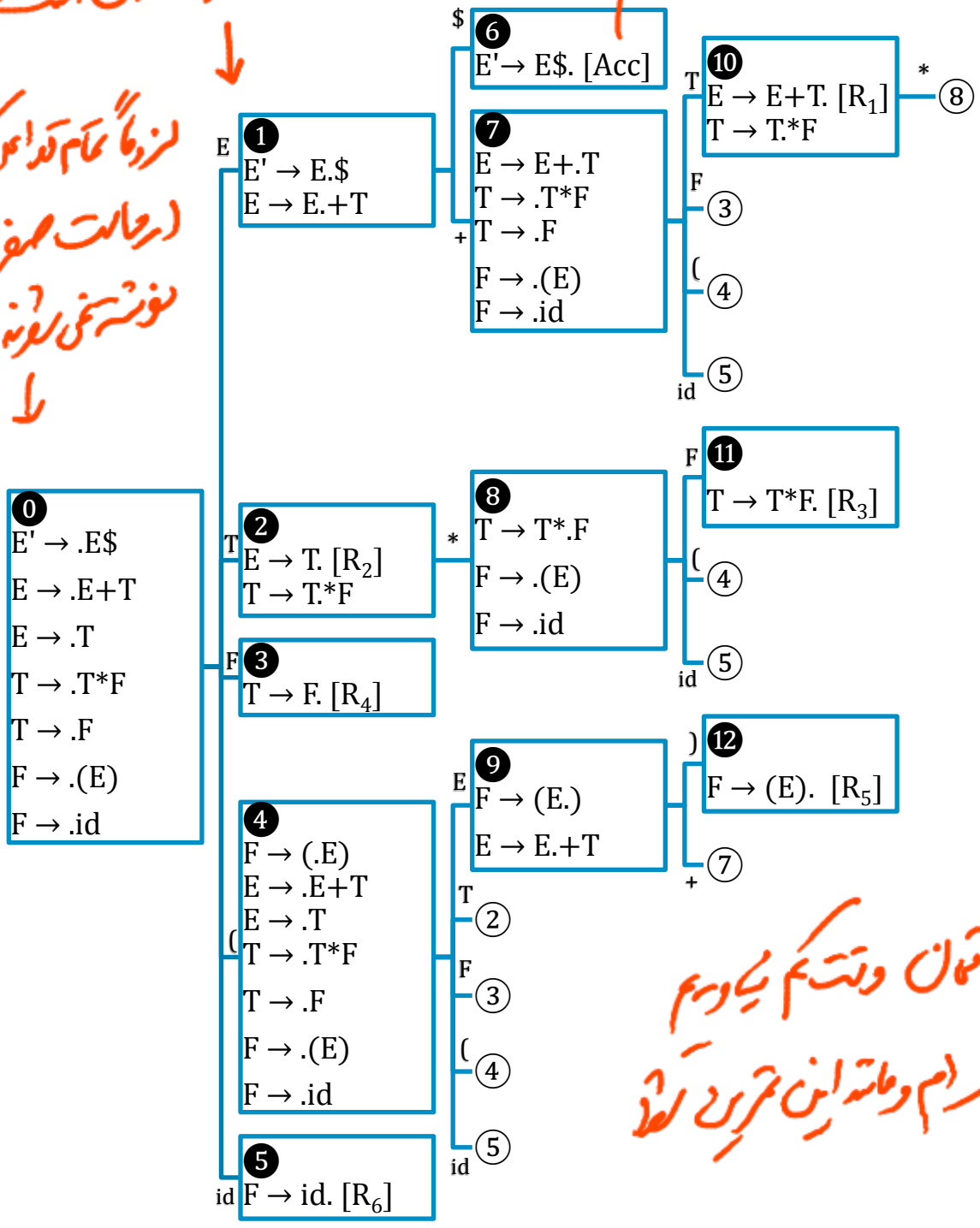
وقتی نقطه به انتهای عبارتی برسد، یعنی هنگام کاهش است.

$A \to xyz$

$A \to .xyz \overset{x}{\Rightarrow} A \to x.yz \overset{y}{\Rightarrow} A \to xy.z \overset{z}{\Rightarrow} A \to xyz.$ (زمان کاهش است)

» مثال: دیاگرام و جدول SLR گرامر زیر را رسم 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	$F \rightarrow id$



** نکته: به ازای عبارات روی فلش، شیفت داریم و به ازای R_1 تا R_n، کاهش داریم.

** نکته: حتماً باید از R_1 تا R_n را ببینیم.

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

حال با توجّه به دیاگرام SLR کشیده شده، جدول SLR را رسم میکنیم:

جدول پارس SLR:

راهنمای جدول: R=Reduce، S=Shift، Acc=Accept و Empty Cell=Error

	Terminals						*Non-Terminals*		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	S_5			S_4			1	2	3
1		S_7				Acc			
2		R_2	S_8		R_2	R_2			
3		R_4	R_4		R_4	R_4			
4	S_5			S_4			9	2	3
5		R_6	R_6		R_6	R_6			
6									
7	S_5			S_4				10	3
8	S_5			S_4					11
9		S_7			S_{12}				
10		R_1	S_8		R_1	R_1			
11		R_3	R_3		R_3	R_3			
12		R_5	R_5		R_5	R_5			
	Action						*Go To*		

** نکته: به ازای Followهای سمت چپ، عملیات کاهش را انجام میدهیم.

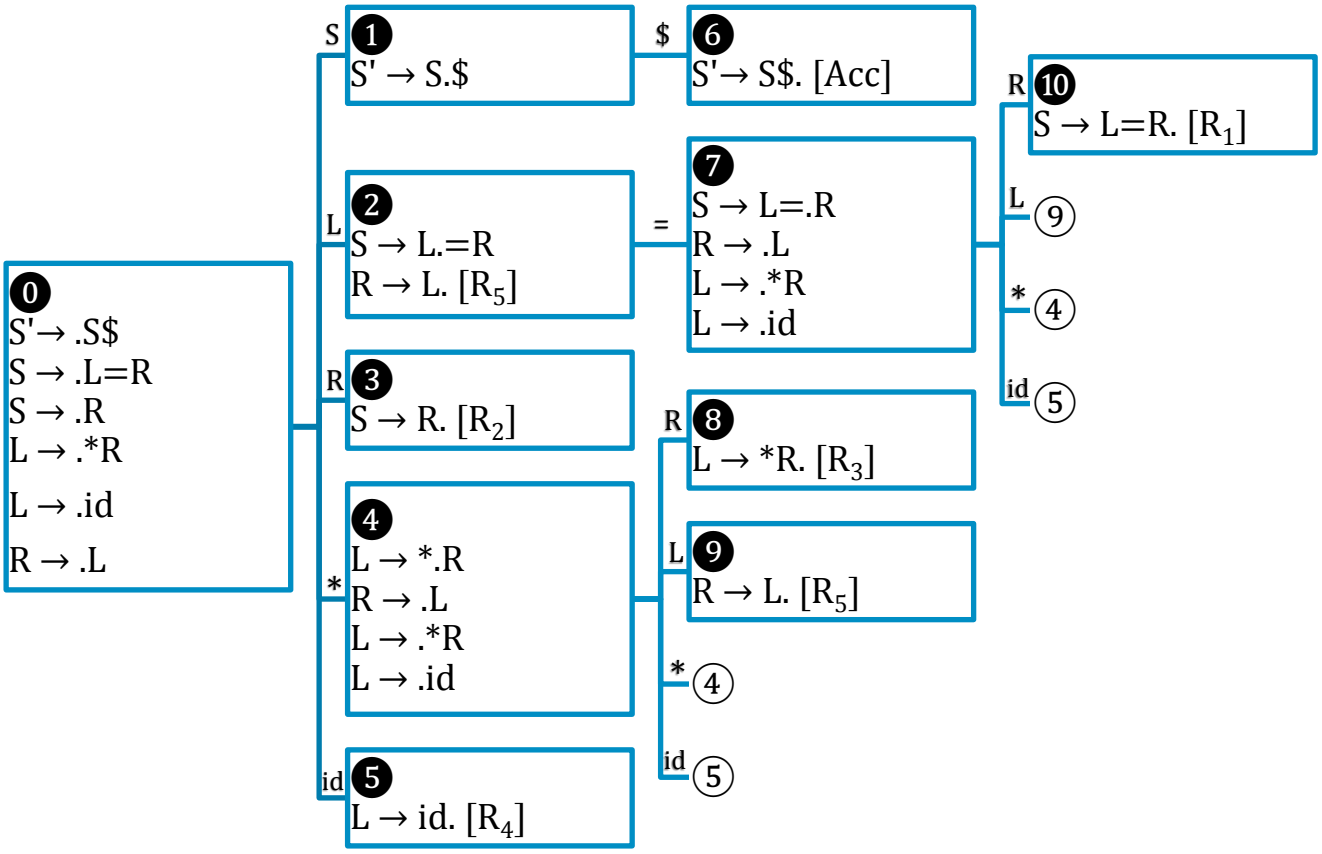
حال عبارت $id + id * id$ را با توجّه به جدول قبل به روش SLR پارس میکنیم:

	انباره	ورودی	عمل انجام شده
1	0	$id * id + id \$$	S_5 : انتقال id و رفتن به مرحله ۵
2	0 id 5	$* id + id \$$	R_6 : کاهش با قاعده ۶
3	0 F 3	$* id + id \$$	R_4 : کاهش با قاعده ۴
4	0 T 2	$* id + id \$$	S_8 : انتقال * و رفتن به مرحله ۸
5	0 T 2 $*$ 8	$id + id \$$	S_5 : انتقال id و رفتن به مرحله ۵
6	0 T 2 $*$ 8 id 5	$+ id \$$	R_6 : کاهش با قاعده ۶
7	0 T 2 $*$ 8 F 11	$+ id \$$	R_3 : کاهش با قاعده ۳
8	0 T 2	$+ id \$$	R_2 : کاهش با قاعده ۲
9	0 E 1	$+ id \$$	S_7 : انتقال + و رفتن به مرحله ۷
10	0 E 1 $+$ 7	$id \$$	S_5 : انتقال id و رفتن به مرحله ۵
11	0 E 1 $+$ 7 id 5	$\$$	R_6 : کاهش با قاعده ۶
12	0 E 1 $+$ 7 F 3	$\$$	R_4 : کاهش با قاعده ۴
13	0 E 1 $+$ 7 T 10	$\$$	R_1 : کاهش با قاعده ۱
14	0 E 1	$\$$	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 $R \rightarrow L$

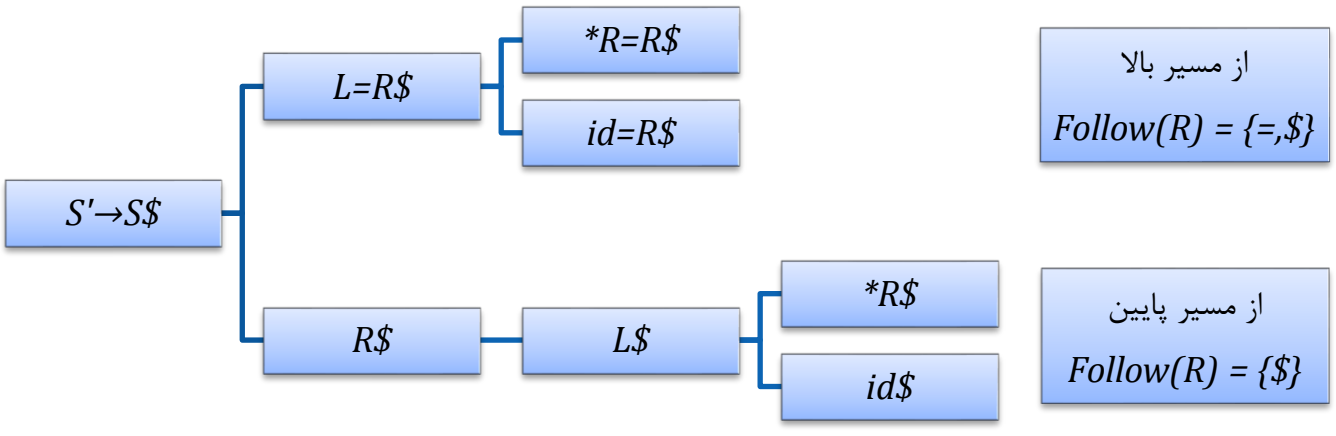
$Follow(S) = \{\$\}$
$Follow(L) = \{=\} + Follow(R) = \{\$, =\}$
$Follow(R) = Follow(S) + Follow(L) = \{\$, =\}$


0
S'→ .S$
S → .L=R
S → .R
L → .*R
L → .id
R → .L
S
1
S' → S.$
$
6
S'→ S$. [Acc]
L
2
S → L.=R
R → L. [R5]
=
7
S → L=.R
R → .L
L → .*R
L → .id
R
3
S → R. [R2]
*
4
L → *.R
R → .L
L → .*R
L → .id
id
5
L → id. [R4]
R
8
L → *R. [R3]
L
9
R → L. [R5]
*
4
id
5
R
10
S → L=R. [R1]
L
9
*
4
id
5

حال مطابق دیاگرام فوق، جدول SLR میکشیم.

	id	*	=	$	S	L	R
0	S_5	S_4			1	2	3
1				Acc			
2			S_7 / 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			

** توجّه: 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 CLR آشنا میشویم.



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

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

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

۵,۱۴ روش CLR

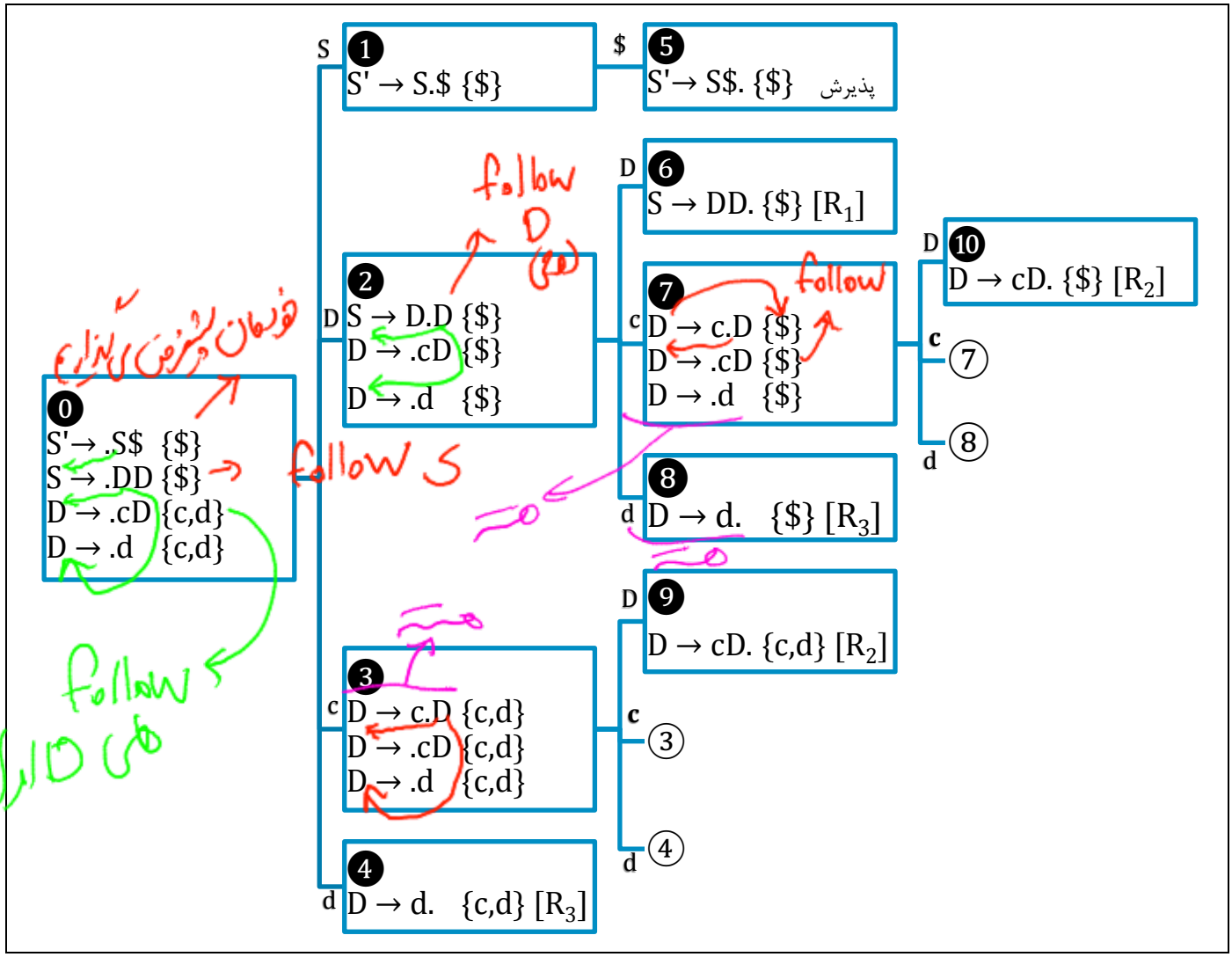
در این روش سعی میشود برای حل مشکل تداخل-کاهش، از همه Follow ها استفاده نشود، بلکه بسته به مسیر از Followها استفاده کنیم.

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.

	c	d	$	S	D
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		

۵,۱۵ روش LALR

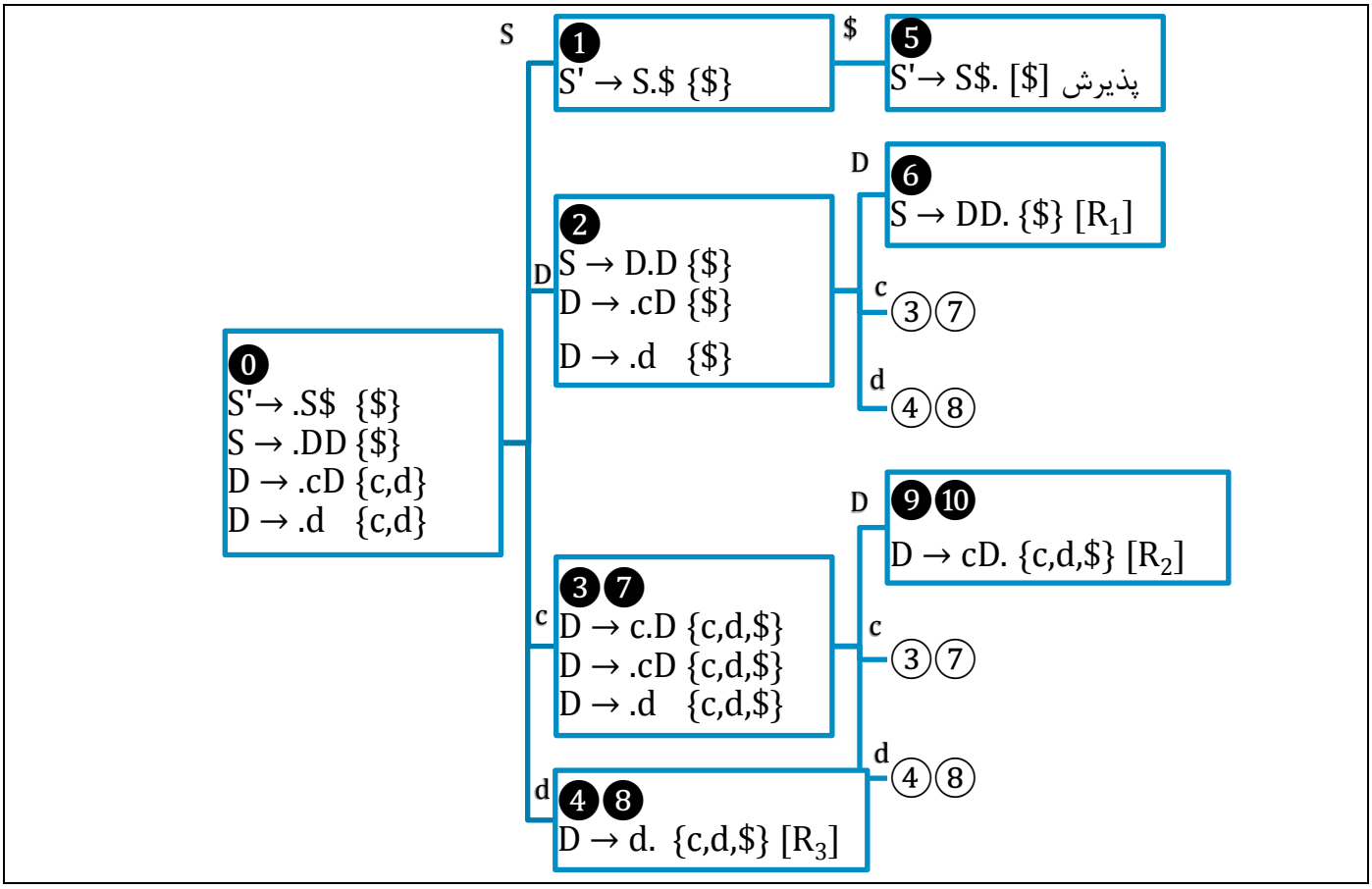
از روی CLR به دست میاید. در روش LALR وضعیت‌هایی که هسته یکسانی دارند را ادغام کرده و L.A. یا Followها را اجتماع میگیریم.

هسته: در یک وضعیت تمام قسمتها به جزء L.A. یا Follow 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 وضعیت ۳۷ میشود {c,d,$}. وضعیتهای ۴و ۸ و وضعیتهای ۹ و ۱۰ هم همین‌طور 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: CLR از SLR آمده 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.

** تذکر: مقایسه روشهای LR از لحاظ قدرت و حافظه: (SLR < LALR < CLR)

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

- پارس به روش LALR

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

- پارس به روش CLR

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

** نکته: همانطور که مشاهده میشود روش CLR زودتر خطا را 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 خطا 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 می توان به سادگی و خلاصه تر بودن آنها اشاره کرد.

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

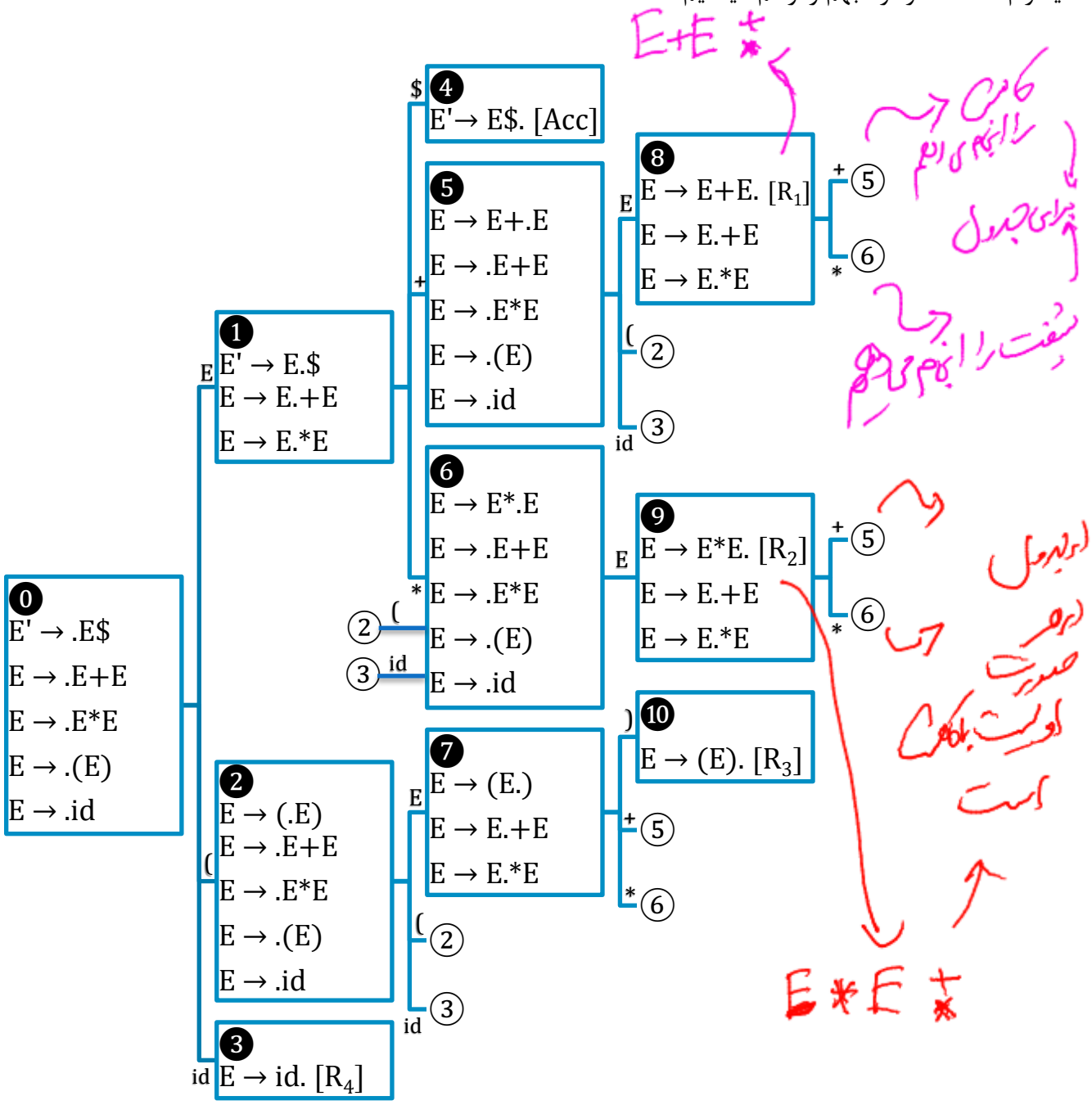
معادل غیر مبهم گرامر فوق به صورت زیر است:

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:

$E \rightarrow T$
$T \rightarrow F$
$F \rightarrow id$

دیاگرام SLR گرامر مبهم را رسم میکنیم:



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

	id	+	*	(	)	$	E
0	S_3	e_1	e_1	S_2	e_2	e_1	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	7
3		R_4	R_4		R_4	R_4	
4							
5	S_3	e_1	e_1	S_2	e_2	e_1	8
6	S_3	e_1	e_1	S_2	e_2	e_1	9
7	e_3	S_5	S_6	e_3	S_{10}	e_4	
8		~~S_5~~ / R_1	S_6 / ~~R_1~~		R_1	R_1	
9		~~S_5~~ / R_2	~~S_6~~ / 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: $E + E.+E$

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

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

- در تقاطع خانه ۸ و * داریم: $E + E.* E$

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.

$$E + \underbrace{\boldsymbol{E} * \boldsymbol{E}}_{E} \rightarrow 1 + \underbrace{2 * 3}_{6}$$

- در تقاطع خانه ۹ و + داریم: $E * E.+E$

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.

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

- در تقاطع خانه ۹ و * داریم: $E * E.* E$

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

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

پس جدول صحیح به صورت زیر است:

	id	**+**	*****	**(**	**)**	**$**	**E**
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.

تمرین اختیاری است

۵,۱۶ اصلاح خطا به روش Phrase Level در پارسرهای LR

خانه های خالی در جدول پارسر که نشانه خطا هستند را با استفاده از توابع پر میکنیم. این توابع خطاها را درست 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	پرانتز بسته ")" را از ورودی حذف کن	پرانتز بسته ")" اضافی
e_3	"+" را روی انباره بگذار و به حالت ۵ برو	عملگر گم شده است.
e_4	پرانتز بسته ")" را روی انباره بگذار و به وضعیت ۱۰ برو	پرانتز 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
e_2 : خطا	$) \$$	0 *E* 1 + 5
e_1 : خطا	$\$$	0 *E* 1 + 5
R_4	$\$$	0 *E* 1 + 5 *id* 3
R_1	$\$$	0 *E* 1 + 5 *E* 8
پذیرش با دو خطا	$\$$	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:

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

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

$E \to E + E \mid E * E \mid (E) \mid id$ عملگر هست

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

** توجّه: تمام پایانه ها عملگر هستند.

$a < b \nRightarrow b > a$ تقدّم a کمتر از b است.
$a = b \nRightarrow b = a$ تقدّم a مساوی b است.
$a > b \nRightarrow b < a$ تقدّم a بیشتر از b است.

دو تابع زیر روی غیرپایانه ها تعریف می‌شوند و حاصل آنها مجموعه‌ای از پایانه ها است:

$FirstTerm(A) = \{a \mid A \overset{+}{\Rightarrow} \boldsymbol{a}\alpha \ or \ A \overset{+}{\Rightarrow} B\boldsymbol{a}\alpha\}$ اوّلین چیز یا یکی بعد از آن

$LastTerm(A) = \{a \mid A \overset{+}{\Rightarrow} \alpha\boldsymbol{a} \ or \ A \overset{+}{\Rightarrow} \alpha\boldsymbol{a}B\}$ آخرین چیز یا یکی قبل از آن

$a = b \ \ iff \ \ \exists \ U \to \cdots ab \cdots \ or \ \ U \to \cdots aEb \cdots$
$a < b \ \ iff \ \ \exists \ U \to \cdots aB \cdots \ and \ \ b \in FirstTerm(B) \qquad a < FirstTerm(B)$
$a > b \ \ iff \ \ \exists \ U \to \cdots Ab \cdots \ and \ \ a \in LastTerm(A) \qquad LastTerm(A) > b$

جدول پارس تقدّم عملگر:

ها Terminal + $					
ها 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 $N \rightarrow \$_{①}E_{②}\$$
1,2 $E \rightarrow E_{③}+_{④}T \mid T$
3,4 $T \rightarrow T_{⑤}*_{⑥}F \mid F$
5,6 $F \rightarrow (_{⑦}E_{⑧}) \mid id$

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

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

$$
\begin{aligned}
&① : \$ < FirstTerm(E)\\
&② : LastTerm(E) > \$\\
&③ : LastTerm(E) > +\\
&④ : + < FirstTerm(T)\\
&⑤ : LastTerm(T) > *\\
&⑥ : * < FirstTerm(F)\\
&⑦ : (< FirstTerm(E)\\
&⑧ : LastTerm(E) >)
\end{aligned}
$$

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

** توجّه: هنگام جایگذاری در جدول، علامت کوچکتر (<) به صورت سطری و علامت بزرگتر (>) به صورت ستونی پر میشود.

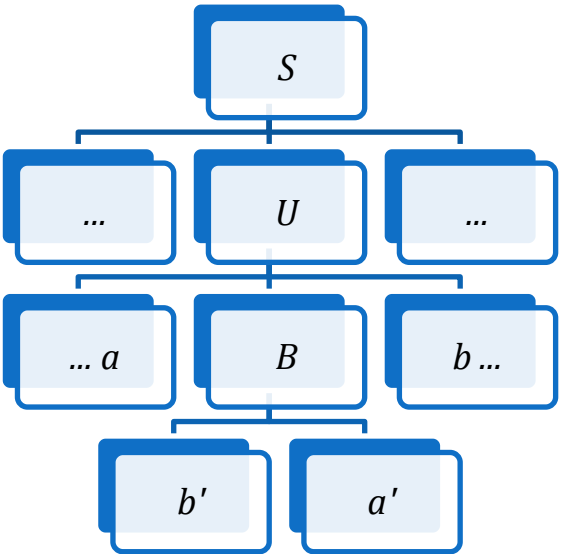
	+	*	(	)	id	$
+	>	<	<	>	<	>
*	>	>	<	>	<	>
(	<	<	<	=	<	e_2
)	>	>	e_1	>	e_1	>
id	>	>	e_1	>	e_1	>
$	<	<	<	e_3	<	=

شرح خطاها:

- e_1 : عملگر گم شده است.
- e_2 : پرانتز بسته ")" گم شده است.
- e_3 : 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

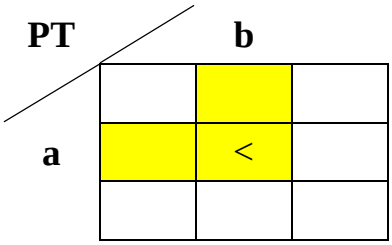


$$a = b$$
$$a' < FirstTerm(B) \rightarrow a < b'$$
$$LastTerm(B) > b \rightarrow a' > b$$

۵,۱۸ پارس به روش تقدّم عملگر

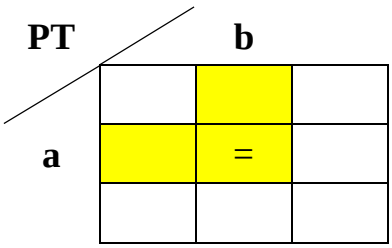
در ابتدا یک $ به انتهای رشته ورودی و یک $ داخل انباره میگذاریم. با مراجعه به بالاترین پایانه روی انباره (مثلاً a) و Token جاری (مثلاً b) و رفتن به سراغ خانه PT(a,b) در جدول پارس مراحل زیر را دنبال میکنیم:

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



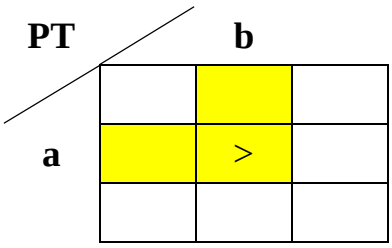
عمل انجام شده	ورودی	انباره
انتقال	$b \dots \$$	$\$ \dots a$
	$\dots \$$	$\$ \dots a < b$

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

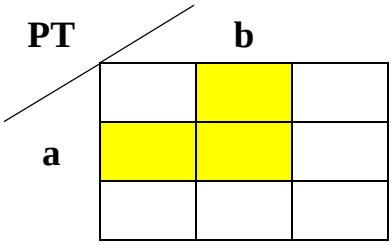


عمل انجام شده	ورودی	انباره
انتقال	$b \dots \$$	$\$ \dots a$
	$\dots \$$	$\$ \dots ab$

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 x-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.

$A \rightarrow abc$
$B \rightarrow abc$

دو تابع Head(A) و Tail(A) داریم که بر روی غیرپایانه ها تعریف میشوند و حاصل آنها مجموعه ای از علائم که همان پایانه ها و غیرپایانه ها میباشند، هستند.

$Head(A) = \{X \mid A \overset{+}{\Rightarrow} X\alpha\}$ چیزهایی که اول میایند

$Tail(A) = \{X \mid A \overset{+}{\Rightarrow} \alpha X\}$ چیزهایی که آخر میایند

» مثال:

$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid id$

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

$x \ominus y \; iff \; \exists \; U \to \cdots xy \cdots$
$x \lessdot y \; iff \; \exists \; U \to \cdots xB \cdots \; and \; y \in Head(B)$
$x \gtrdot y \; iff \; \exists \; U \to \cdots Ay \cdots \; and \; x \in Tail(A)$
$\quad or \; \exists \; U \to \cdots AB \cdots \; and \; x \in Tail(A) \; and \; y \in Head(B)$

یعنی:

$Tail(A) \gtrdot y$
$Tail(A) \gtrdot Head(B)$

** توجّه: x و y هم پایانه (T) و هم غیرپایانه (N) هستند.

** توجّه: 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 تقدّم ساده بررسی کنید.

0 $N \to \$ \; S \; \$$
1 $S \to (SS)$
2 $S \to c$

پاسخ:

0 $N \to \$ \; ① S \; ② \$$
1 $S \to (③ S \; ④ S \; ⑤)$
2 $S \to c$

① : $\$ \lessdot Head(S)$
② : $Tail(S) \gtrdot \$$
③ : $(\lessdot Head(S)$

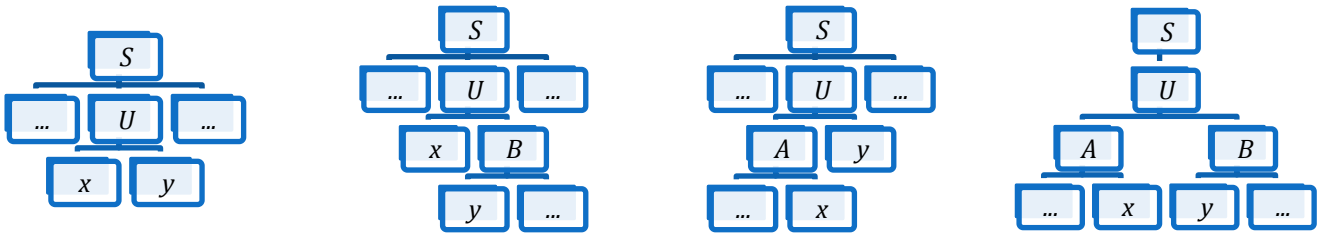
④ : $Tail(S) \circledcirc Head(S)\ , S \circledcirc Head(S)$
⑤ : $Tail(S) \circledcirc)$

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

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

	S	(	)	c	$
S	⊜	⋖	⊜	⋖	⊜
(	⊜	⋖		⋖	
)		⋗	⋗	⋗	⋗
c		⋗	⋗	⋗	⋗
$	⊜	⋖		⋖	

** توجّه: علامت‌های ⋖ (کوچکتر) به صورت سطری پُر میشوند.
** توجّه: علامت‌های ⋗ (بزرگتر) به صورت ستونی پُر میشوند.



۵,۲۰ پارس به روش تقدّم ساده

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

۱. گام اوّل: اگر PT(x,b) معادل علامت مساوی (=) باشد، Token جاری (یعنی b) را وارد انباره میکنیم. (عمل انتقال)
۲. گام دوّم: اگر PT(x,b) معادل علامت کوچکتر (<) باشد، ابتدا علامت کوچکتر (<) و سپس b را وارد انباره میکنیم. (عمل انتقال)
۳. گام سوّم: 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 Top بنامیم و اگر سمت چپ قاعده ای که سمت راست آن با دستگیره تطبیق داشته را LHS بنامیم، رابطه Top با LHS را از روی جدول پیدا کرده و به صورت زیر عمل میکنیم:

۳,۱. حالت اوّل: اگر $Top \ominus LHS$ باشد، LHS را وارد انباره میکنیم.

۳,۲. حالت دوّم: اگر $Top \oslash LHS$ باشد، ابتدا علامت کوچکتر (<) و سپس LHS را وارد انباره میکنیم.

۳,۳. حالت سوّم: اگر LHS و Top رابطه ای نداشته باشند، یک خطای نحوی رخ داده است.

** تذکر: هیچ گاه $Top \ominus LHS$ نمیشود.

۴. گام چهارم: اگر 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	$ ⋖ (	$(cc) c) $	انتقال
3	$ ⋖ (⋖ (	$cc) c) $	انتقال
4	$ ⋖ (⋖ (⋖ c	$c) c) $	⋗ کاهش $Top = (\ , \ S \to c$ (انتقال LHS) $LHS = S \ , Top \ominus LHS$
5	$ ⋖ (⋖ (S	$c) c) $	انتقال
6	$ ⋖ (⋖ (S ⋖ c	$) c) $	⋗ کاهش $Top = S \ , \ S \to c$ (انتقال LHS) $LHS = S \ , Top \ominus LHS$
7	$ ⋖ (⋖ (SS	$) c) $	انتقال
8	$ ⋖ (⋖ (SS)	$c) $	⋗ کاهش $Top = (\ , \ S \to (SS)$ (انتقال LHS) $LHS = S \ , Top \ominus LHS$
9	$ ⋖ (S	$c) $	انتقال
10	$ ⋖ (S ⋖ c	$) $	⋗ کاهش $Top = S \ , \ S \to c$ (انتقال LHS) $LHS = S \ , Top \ominus LHS$
11	$ ⋖ (SS	$) $	انتقال
12	$ ⋖ (SS)	$	⋗ کاهش $Top = \$ \ , \ S \to (SS)$ (انتقال LHS) $LHS = S \ , Top \ominus LHS$
13	$ S	$	پذیرش

۵,۲۱ چپ‌گردی

$U \to U \cdots$
$A \to \cdots xU \cdots$
$\boldsymbol{x} \ominus \boldsymbol{U}\ , x \lessdot Head(U) \Rightarrow \boldsymbol{x} \lessdot \boldsymbol{U}$ تداخل

۵,۲۲ رفع چپ‌گردی

$U \to U \cdots$
$A \to \cdots xN \cdots$
$N \to U$
$x \ominus N\ , x \lessdot Head(N) \Rightarrow x \lessdot U$

۵,۲۳ راست‌گردی

$U \to \cdots U$
$A \to \cdots Ux \cdots$
$\boldsymbol{U} \ominus \boldsymbol{x}\ , Tail(U) \gtrdot x \Rightarrow \boldsymbol{U} \gtrdot \boldsymbol{x}$ تداخل

۵,۲۴ رفع راست‌گردی

$U \to \cdots U$
$A \to \cdots Nx \cdots$
$N \to U$
$N \ominus x\ , Tail(N) \gtrdot x \Rightarrow U \gtrdot x$

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 (مانند Main)
- چک کردن کنترل جریان (Control Follow Checking)
- چک کردن پارامترها (Parameter Checking)

پویا:

- چک کردن محدوده (Range Checking)

گروه آموزشی : مهندسی کامپیوتر - نرم افزار

نام درس: اصول طراحی کامپایلر کد درس: ۷۲۳۱ مقطع تدریس: کارشناسی
تعداد واحد: ۳ نوع درس از لحاظ آکادمیک: نوع درس از لحاظ شهریه:

ساعات تشکیل کلاس در هفته:

هفته‌های تدریس	طرح درس طبق سرفصل
هفته اول	مقدمات، مفاهیم ترجمه و مترجم، انواع مترجم‌ها شامل کامپایلر، مفسر، اسمبلر و پیش‌پردازنده. مقایسه مفسر و کامپایلر. معرفی کلی ساختار و اجزاء کامپایلر شامل: تحلیل‌گر لغوی، تحلیل‌گر نحوی، تحلیل‌گر معنایی، تولید کد میانی، بهینه سازی کد، تولید کد نهایی، مدیریت جدول نمادها، مدیریت خطاها.
هفته دوم	بررسی انواع گرامرها و خواص عمومی زبان‌ها، طبقه بندی چامسکی، مرور کلی بر درس نظریه زبان‌ها و ماشین‌ها، مفاهیم درخت اشتقاق، بسط چپ‌ترین(LMD)، بسط راست‌ترین(RMD)، مفاهیم ابهام، زبان‌های مبهم، گرامر مبهم، روش‌های رفع ابهام، گرامرهای مختصر و مفید، گرامرهای معادل، فرم جمله‌ای
هفته سوم	تحلیل لغوی، تعریف و مفهوم توکن(Token)، عبارات منظم برای نمایش الگوی توکن‌ها، دیاگرام گذر، وظایف تحلیل‌گر لغوی، گرامرهای تفسیر حالت قطعی و غیرقطعی، روش‌های برخورد با خطاهای لغوی، اصلاح خطاهای لغوی
هفته چهارم	تحلیل نحوی، روش‌های تحلیل نحوی شامل تحلیل یا تجزیه بالا به پایین و تحلیل پایین به بالا، معرفی روش‌های اصلاح خطا در تحلیل‌گر نحوی، معرفی و تعریف روش‌های تحلیل بالا به پایین، معرفی پارسرهای پیشگو، معرفی روش Recursive Descent،
هفته پنجم	تحلیل به روش LL(1)، مفاهیم First و Follow، پارسر LL(1)،
هفته ششم	نحوه تشکیل جدول پارس LL(1)، شرایط LL(1) بودن یک گرامر، اصلاح خطا 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،

نام درس: **اصول طراحی کامپایلر** **کد درس:** **۷۲۳۱**

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

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.