اصول طراحی کامپایلر

نیمسال دوم ۱۴۰۲ - ۱۴۰۱

LR	بخش Action						بخش Goto		
	id	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				Acc			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

علی سلیمانی

عضو هیئت علمی دانشگاه آزاد واحد اصفهان (خوراسگان)

فصل اول- مقدمه‌ای بر درس کامپایلر

چکیده: در این درس با کاربردهای درس کامپایلر آشنا شده و نشان میدهیم مفاهیم درس کامپایلر علاوه بر ساخت یک زبان برنامه‌نویسی جدید همچنین میتواند در دریافت داده‌های فرمتدار ، طراحی مترجمها و پردازش زبان طبیعی به کار گرفته شود.

برنامه‌نویس کامپیوتر: آشنایی با تکنیکهای کدنویسی و مسلط بودن به یک یا دو زبان برنامه‌نویسی

مهندس کامپیوتر: علاوه بر کدنویسی ، باید با روش توصیف یک زبان و تکنیکهای ساخت آن آشنایی داشته باشد.

چرا درس کامپایلر: قطعا آشنایی با تکنیکهای ساخت زبان میتواند در ساخت یک زبان برنامه‌نویسی جدید به کار گرفته شود. آیا در دنیای پر از زبانهای قدرتمند که اغلب دو یا سه دهه از طراحی و توسعه آنها میگذرد ضرورت و انگیزه‌ای برای خلق یک زبان جدید میتواند وجود داشته باشد؟

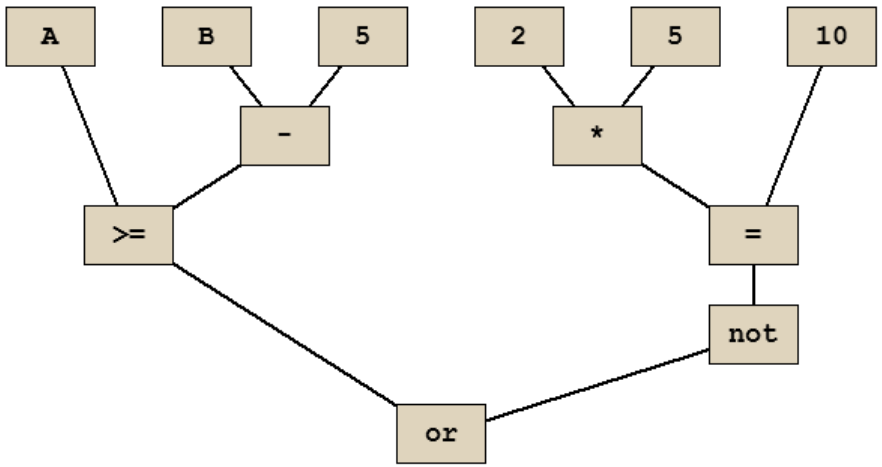
مثال۱: دریافت لیستی از نقاط

```
5
10        -2
-3        5
8         16
-4        -7
17        6
```

```
Pol = (10, -2), (-3, 5), (8, 16), (-4, -7), (17, 6);  // a comment
```

مثال۲: ترسیم شکل گرافیکی درخت نحو یک عبارت جبری

```
Exp = (A >= B - 5) or not (2 * 5 = 10)
```



مثال۳: تایید درستی عبارتهای پرانتزی و محاسبه عمق آن

```
(a,(a,a),(a),a)   →  depth = 2
((a),(a,(a)))     →  depth = 3
```

مثال٤: تبدیل عبارت میانوندی به عبارتهای پیشوندی و پسوندی

```
Regular = (a|b)*.a.b.b
Postfix = a , b , | , * , a , . , b , . , b , .
```

مثال٥: طراحی ماشین حساب متنی

```
Exp = 10 * (-5 + 8) / ((4 + - 7) * - (10.2 - 2.2))
Result = 1.25
```

مثال٦: ساده کردن چندجمله‌ای

```
Polynomial = ((n - 1) ^ 3 - 3n + 1) ^ 2 * (n ^ 3 + 5n - 1)
Simple = n9 - 6n8 + 14n7 - 31n6 + 51n5 - 9n
```

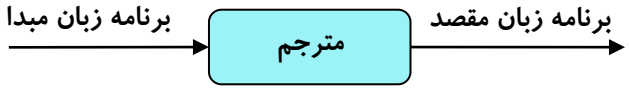
کاربردهای درس کامپایلر

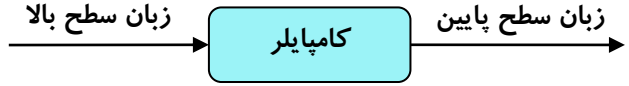
- **ساخت یک زبان برنامه‌نویسی جدید:** اصلیترین کاربرد این درس است.
- **دریافت داده‌های فرمتدار:** مثل دریافت لیستی از نقاط
- **تولید مترجمها:** مثل تبدیل عبارت میانوندی به پسوندی و طراحی ماشین حساب متنی
- **پردازش زبان طبیعی:** درس کامپایلر مقدمه‌ای بر پردازش نحوی زبان طبیعی است.

فصل اول- بخش پردازنده‌های زبان

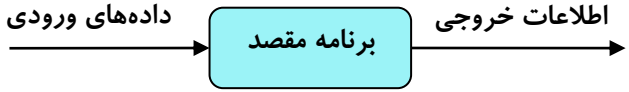
چکیده: در این درس مهمترین پردازنده‌های زبان (کامپایلر و مفسر) را تعریف کرده و مزایای استفاده از هر کدام را نسبت به دیگری بیان میکنیم. سپس فاز را تعریف کرده و فازهای مربوط به یک کامپایلر را معرفی میکنیم.

مترجم: برنامه‌ای است که به عنوان ورودی برنامه نوشته شده در یک زبان (زبان مبدا) را دریافت میکند و به عنوان خروجی برنامه‌ای به زبان دیگر (زبان مقصد) را تولید میکند.

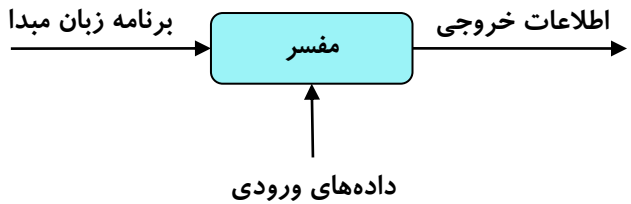




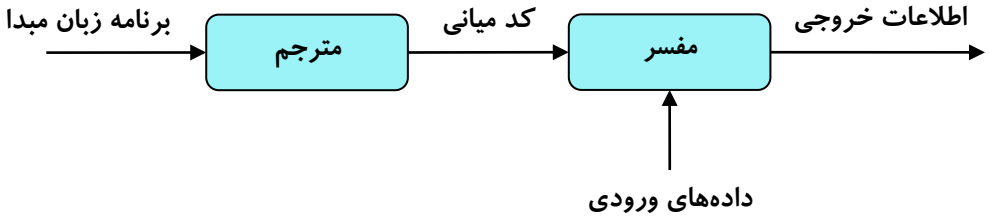
مرحله اجرا: برنامه‌هایی که توسط کامپایلر به زبان ماشین ترجمه میشوند بلافاصله میتوانند اجرا شوند.



مفسر: نوع دیگری از پردازنده زبان است که به جای ترجمه و تولید برنامه مقصد مستقیما دستورهای نوشته شده در برنامه مبدا را روی داده‌های ورودی اجرا و خروجیها را تولید میکند.



پردازنده‌های ترکیبی: نمونه دیگری از پردازنده‌های زبان عمل کامپایل و تفسیر را با هم ترکیب میکنند.



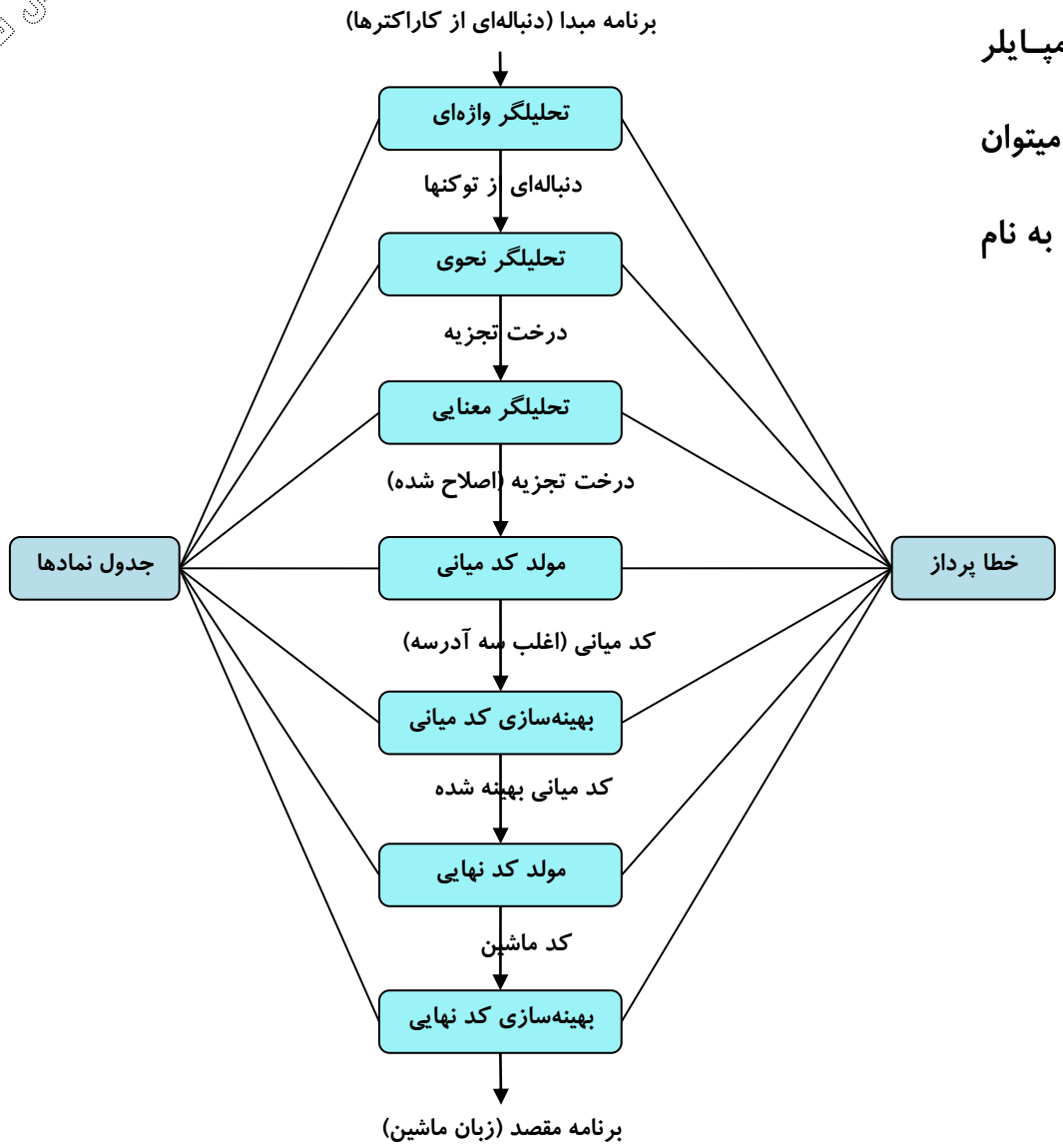
مزایای کامپایلر

- **سرعت اجرای بالا:** برنامه فقط یکبار ترجمه میشود.
- **اجرای برنامه مستقل از کامپایلر:** برای اجرای برنامه‌های کامپایل شده نیاز به کامپایلر نیست.
- **حفاظت از کد منبع:** تنها کد اجرایی نرم‌افزار در اختیار مشتریان قرار میگیرد.

مزایای مفسر

- **پیاده‌سازی آسان:** نوشتن یک مفسر از نوشتن یک کامپایلر ساده‌تر است.
- **سرعت توسعه نرم‌افزار:** بعد از هر تغییر در برنامه برای مشاهده عملکرد آن نیاز نیست همه برنامه ترجمه شود چون اجرای برنامه با رسیدن به اولین خط شروع میشود.
- **خطایابی بهتر:** مفسر از اطلاعاتی که در زمان اجرا دارد میتواند برای خطایابی بهتر استفاده کند. با فرض `A[1..10]` و این فرض که `i` مقدار 11 داشته باشد، جمله `A[i]:= 120` را اجرا نمیکند و به جای آن یک خطای خارج از بازه را اعلام میکند.

ساختار کامپایلر: پـردازش یـک کامپـایلر آنقدر پیچیده است که از نظر منطقی میتوان آن را به چندین زیرپردازش کوچکتر به نام فاز تقسیم کرد.



فصل اول- بخش تحلیلگر واژه‌ای (اسکنر)

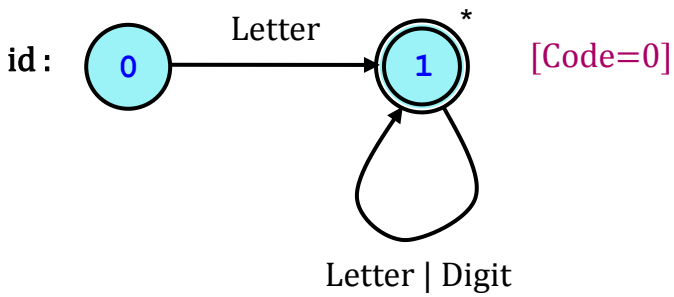
هدف: در این درس با تحلیلگر واژه‌ای (اسکنر) که اولین فاز کامپایلر است آشنا میشویم. این آشنایی در حد مقدماتی و به صورت مفهومی است. روش طراحی اسکنر و جزئیات الگوریتمی مربوط به آن در فصل ۲ بررسی میشوند.

تعریف توکن: هر چند کاراکتر کنار هم که یک نشانه در زبان برنامه‌نویسی باشند را توکن مینامند.

انواع توکنها

- بی قاعده: کلمه‌های رزرو شده (if) ، نمادهای عملگر (=>) ، نمادهای جداساز (;)
- با قاعده: شناسه‌ها (Letter1) ، اعداد (15) ، رشته‌ها (‘Hello’)
- توکن‌نماها: فضاهای خالی ، کامنتها

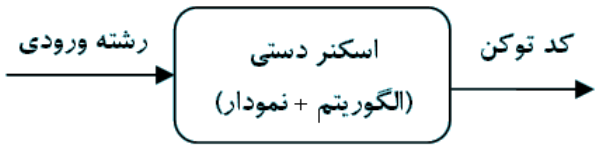
نمودار انتقالی: روشی برای تشخیص توکن است و با افزودن یک الگوریتم ساده، به راحتی به برنامه اسکنر تبدیل میشود.



روش دستی: کدنویسی و ساختن آن آسانتر است.

$$\left.\begin{array}{l} Token_1 \xrightarrow{\text{دستی}} diag_1 \\ Token_2 \xrightarrow{\text{دستی}} diag_2 \\ \dots \\ Token_n \xrightarrow{\text{دستی}} diag_n \end{array}\right\} \xrightarrow{\text{ادغام (دستی)}} Diag \Big\} = \text{اسکنر}$$

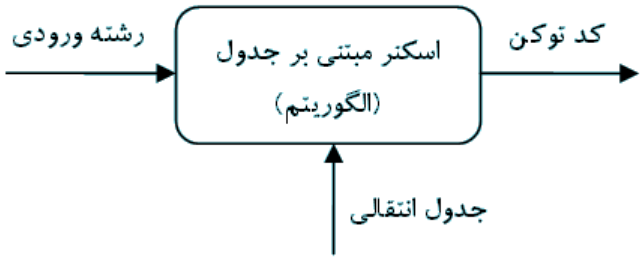
الگوریتم تشخیص توکن



روش خودکار: برای انجام تغییر در عبارتهای منظم نیازی نیست برنامه اسکنر بازنویسی شود.

id = Letter (Letter | Digit)*

$$\left.\begin{array}{l} Re_1 \xrightarrow{\text{خودکار}} diag_1 \\ Re_2 \xrightarrow{\text{خودکار}} diag_2 \\ \dots \\ Re_n \xrightarrow{\text{خودکار}} diag_n \end{array}\right\} \xrightarrow{\text{ادغام (خودکار)}} Diag$$



فصل اول- بخش تحلیلگر نحوی (تجزیه‌گر)

هدف: در این درس با تحلیلگر نحوی (تجزیه‌گر) که دومین فاز کامپایلر است آشنا میشویم. این آشنایی در حد مقدماتی و به صورت مفهومی است. روش طراحی تجزیه‌گر و جزئیات الگوریتمی مربوط به آن در فصلهای بعدی بررسی میشوند.

تعریف ساختار: هر چند توکن کنار هم که یک مفهوم در زبان برنامه‌نویسی ایجاد میکنند یک ساختار (مثل جمله شرطی) و هر چند ساختار کنار هم که مفهوم پیچیده‌تری در زبان ایجاد میکنند (مثل زیربرنامه) یک ساختار دیگر تشکیل میدهند.

درخت تجزیه: تحلیلگر نحوی توکنهای زبان را خوانده و همه آنها را در یک ساختار سلسله مراتبی به هم مرتبط میکند.

برنامه اصلی → زیربرنامه‌ها → جملات شرطی و حلقه‌ها → جملات انتساب → عبارتها → توکنها

گرامر مستقل از متن: روشی برای توصیف ساختارهای زبان است و با افزودن یک الگوریتم خاص ، به یک برنامه تجزیه‌گر تبدیل میشود.

$S \rightarrow id{:=}E$

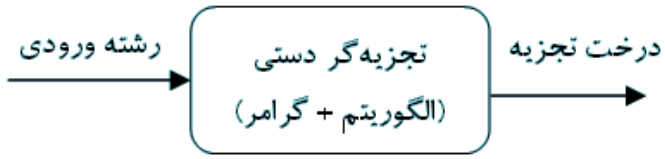
$E \rightarrow E{+}T \mid E{-}T \mid T$

$T \rightarrow T{*}F \mid T/F \mid F$

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

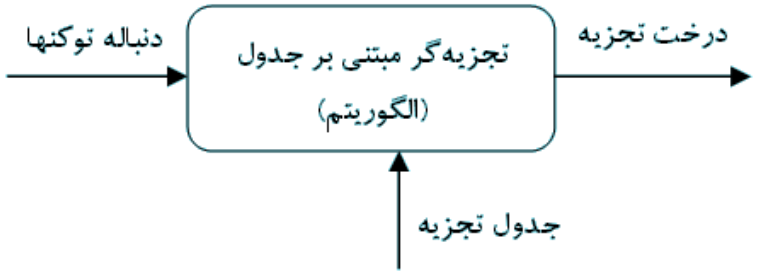
روش دستی: کدنویسی و ساختن آن آسانتر است.

$$\left.\begin{array}{l} N_1 + \text{الگوریتم تجزیه} \rightarrow proc_1 \\ N_2 + \text{الگوریتم تجزیه} \rightarrow proc_2 \\ \dots \\ N_m + \text{الگوریتم تجزیه} \rightarrow proc_n \end{array}\right\} = \text{تجزیه‌گر}$$



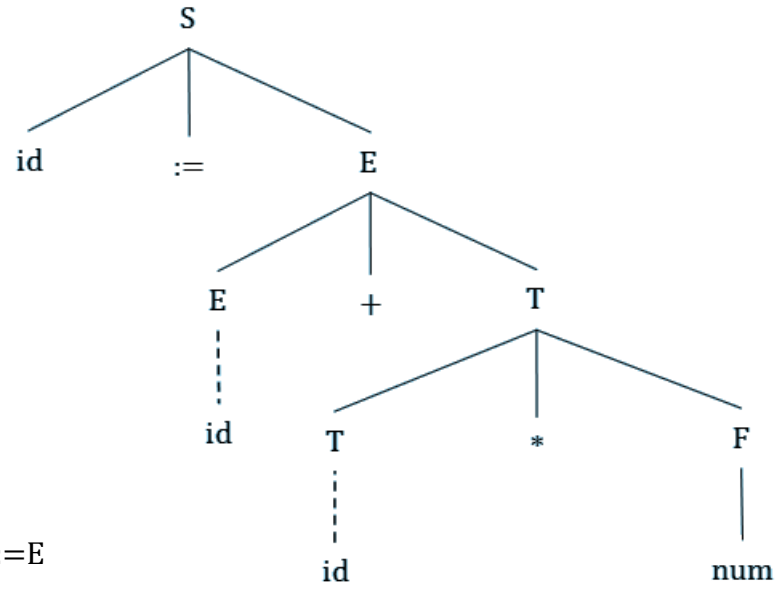
روش مبتنی بر جدول: برای انجام تغییر در گرامر نیازی نیست برنامه تجزیه‌گر بازنویسی شود.

$$Gram \xrightarrow{\text{خودکار}} \text{جدول تجزیه}$$



خروجی تجزیه‌گر

- درخت نحو (Syntax Tree) : آدرسها در برگ و عملگرها در گره‌های میانی
- درخت تجزیه (Parse Tree) : پایانه‌ها در برگ و غیرپایانه در گره‌های میانی



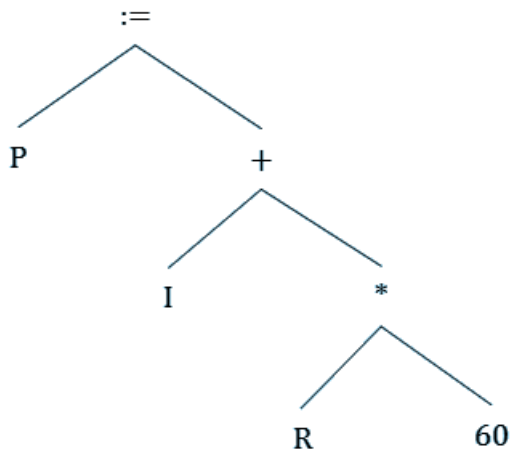
$S \rightarrow id:=E$

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

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

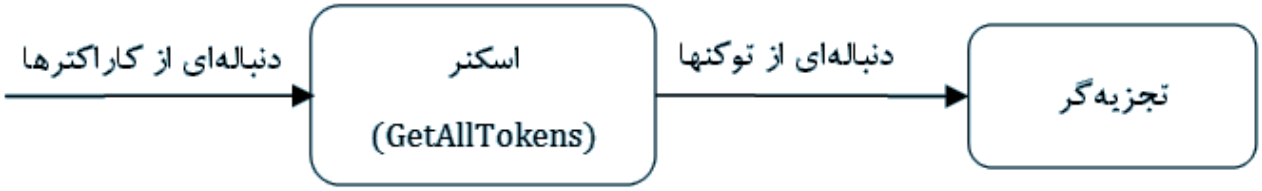
$F \rightarrow (E) \mid id \mid num$

درخت تجزیه

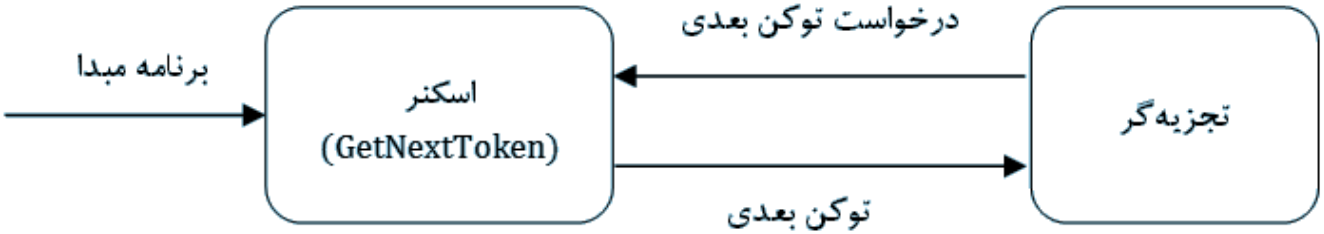


درخت نحو یا ارزشیابی

رابطه منطقی اسکنر و تجزیه‌گر: عمل کامپایل با اسکنر شروع میشود و بعد از شناسایی همه توکنها ، تجزیه‌گر به عمل کامپایل ادامه میدهد.



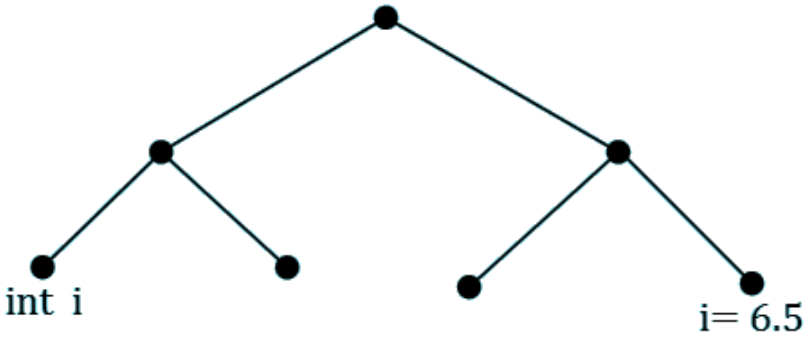
رابطه اسکنر و تجزیه‌گر در پیاده‌سازی: عمل کامپایل با تجزیه‌گر شروع میشود و اسکنر با درخواستهای تجزیه‌گر فراخوانی میشود.



فصل اول- بخش تحلیلگر معنایی

هدف: در این درس با تحلیلگر معنایی که سومین فاز کامپایلر است آشنا میشویم. این آشنایی در حد مقدماتی و به صورت مفهومی است. جزئیات الگوریتمی مربوط به این فاز در فصل آخر بررسی میشود.

دلیل مطرح شدن این فاز: از نظر تجزیه‌گر درستی یک ساختار فقط از درستی اجزای آن حاصل میشود، در حالی که از نظر زبان برنامه‌نویسی منطقی بودن یک ساختار گاهی با در نظر گرفتن منطق ساختارهای دیگر حاصل میشود.



بنابراین، تجزیه‌گر در تشخیص منطق یک برنامه ضعف دارد و از آنجاییکه تجزیه‌گر طبق گرامر زبان عمل میکند این ضعف در گرامر زبان نهفته شده است.

روشهای حل

- گرامر بهتری برای توصیف زبان انتخاب شود
- فاز دیگری برای رفع این ضعف افزوده شود

گرامرهای حساس به متن

- مزیت : قدرت توصیف بالاتری دارند
- عیب : به سادگی به برنامه تبدیل نمیشوند

گرامرهای مستقل از متن

- مزیت : به راحتی به برنامه تبدیل میشوند
- عیب : دارای ضعف بیانی هستند

راه حل : یک فاز دیگر به نام تحلیلگر معنایی افزوده شود تا درستی ساختار را بر مبنای مشخصات ساختارهایی که جزئی از آن نیستند بررسی کند.

وظایف تحلیلگر معنایی

کنترل نوع : اندیس آرایه نمیتواند اعشاری باشد ، در عبارت a+b هر دوی a و b باید از انواع سازگار باشند.

تبدیل نوع ضمنی : چنانچه در عبارت a+b متغیر b صحیح و a اعشاری باشد کامپایلر عبارت داده شده را به a+IntToFloat(b) تبدیل میکند.

متغیر استفاده نشده : کامپایلر به کاربر هشدار میدهد و برای آن متغیر حافظه‌ای در نظر نمیگیرد.

متغیر تعریف نشده : کامپایلر خطای مناسبی را تولید میکند.

سازگاری پارامترهای فرمال و واقعی : برای زیربرنامه void P(int N) هر دو فراخوانی P(2.5) و P(3,5) نادرست هستند.

کنترل محدوده اندیس آرایه‌ها : با فرض A[1..10] عبارت A[15] نادرست است و خطای آن در زمان کامپایل گزارش میشود. ولی درست یا نادرست بودن عبارت A[i] به زمان اجرا واگذار میشود.

کنترلهای ایستا و پویا: هر یک از وظایف تحلیلگر معنایی که در زمان ترجمه انجام شود کنترل ایستا نام دارد و هر کدام که در زمان اجرا انجام شود کنترل پویا نام دارد. کنترل محدوده اندیس آرایه‌ها اغلب در زمان اجرا انجام میگیرد.

جدول نمادها: مکانی برای نگهداری شناسه‌ها و مشخصات آنهاست. از نظر تئوری، شناسه‌ها در فاز اول تشخیص داده میشوند و یک ورودی برای آنها ایجاد میشود و سایر مشخصات آنها در فاز دوم به جدول اضافه میشود.

```
var Day: integer;
```

	جدول نمادها (پس از تحلیل نحوی)
	...
p1	Day, var, integer
	...

	جدول نمادها (پس از تحلیل واژه‌ای)
	...
p1	Day
	...

اشکال دیدگاه تئوری: اسکنر موقعیت شناسه‌ها را در اختیار ندارد و نمیداند آنها در موقعیت تعریف قرار دارند یا استفاده. برای نمونه برای تکه کد زیر اسکنر شناسه Day را سه بار به جدول نمادها اضافه میکند.

```
var Day: Integer;
Day:= Day+ 1;
```

در عمل: مدیریت جدول نمادها توسط کنشهایی انجام میگیرد که لابلای تجزیه‌گر گذاشته میشوند.

فصل اول- بخش مولد کد میانی

هدف: در این درس با مولد کد میانی که چهارمین فاز کامپایلر است آشنا میشویم. الگوریتمهایی که در این بخش گفته میشوند در حد مفهومی و قابل درک برای ذهن انسان است. روش دقیق تولید کد میانی و جزئیات الگوریتمی سطح پایین و قابل پیاده‌سازی آن برای ماشین در فصل آخر بررسی میشوند.

دستور سه آدرسه: شکل اصلی دستور سه آدرسه از نظر منطقی و در حافظه کامپیوتر به صورت زیر است:

(op, A, B, C)

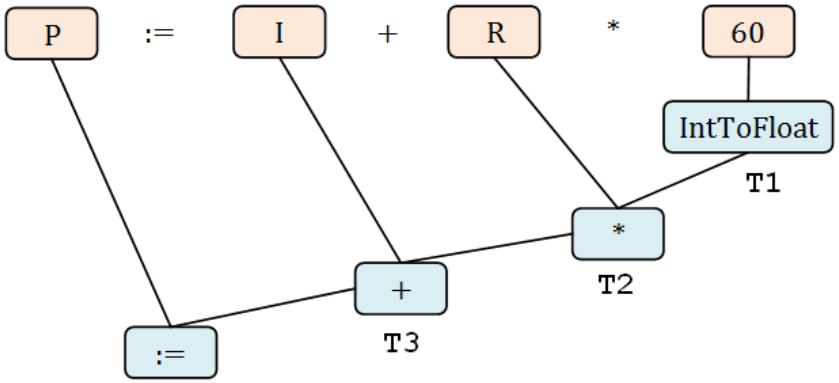
Op	Adr_1	Adr_2	Target
op	A	B	C

شرح دستور	شکل نوشتاری	شکل منطقی
عملگر دودویی (ریاضی، منطقی، مقایسه‌ای)	C:= A op B	(op, A, B, C)
عملگر یگانی	B:= op A	(op, A, B)
کپی	B:= A	(:=, A, B)
پرش بدون شرط	Goto L	(J, L)
پرش شرطی (با نتیجه درست)	If C goto L	(JT, C, L)
پرش شرطی (با نتیجه نادرست)	Ifnot C goto L	(JF, C, L)
تبدیل نوع	B:= IntToFloat(A)	(IntToFloat, A, B)
افزایش (کاهش) مقدار	A:= A ± n	(Inc/Dec, A, n)

عبارتهای ریاضی: روند تبدیل یک عبارت ریاضی به کد میانی مثل پیمایش پسوندی درخت ارزشیابی آن از برگ به سمت ریشه است.

مثال: (فرض کنید متغیرهای داده شده اعشاری هستند)

```
P := I + R * 60
```



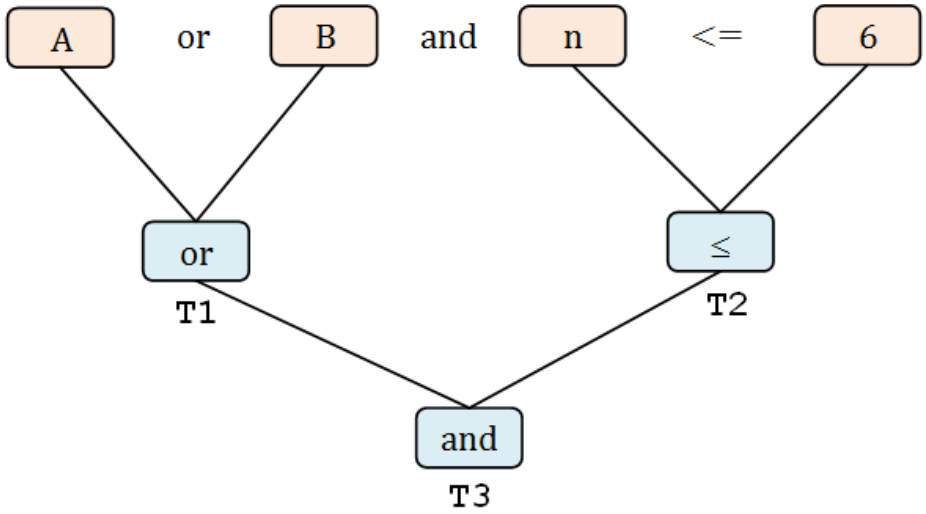

```
(IntToFloat, 60, T1)
(*, R, T1, T2)
(+, I, T2, T3)
(:=, T3, P)
```

Op	Adr_1	Adr_2	Target
IntToFloat	60		T1
*	R	T1	T2
+	I	T2	T3
:=	T3		P

عبارتهای منطقی (ارزشیابی کامل): در روش ارزشیابی کامل عبارت منطقی مثل یک عبارت ریاضی بـه طـور کامل ارزشیابی شده و نتیجه آن در یک متغیر ذخیره میشود.

مثال:

```
(A or B) and (n<= 6)
```




```
(or, A, B, T1)
(<=, n, 6, T2)
(and, T1, T2, T3)
```

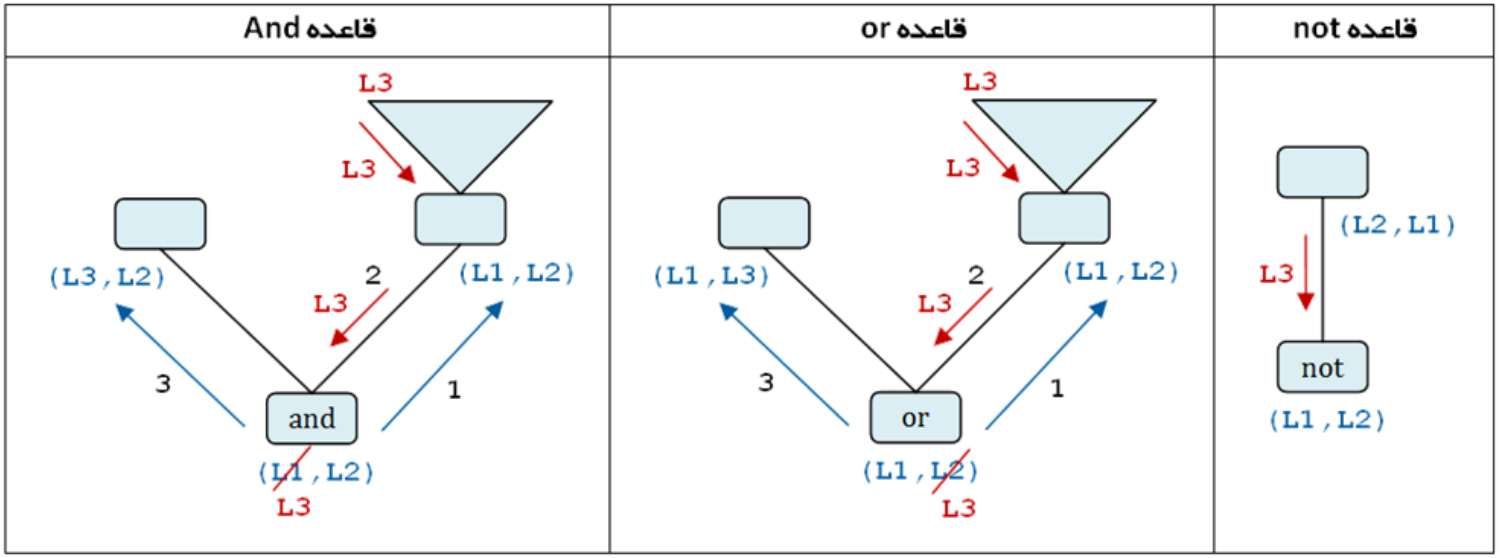
عبارتهای منطقی (**میانبر زدن**): در روش میانبر زدن هر واحد منطقی ارزشیابی کامل شده و پس از آن یک زوج پرش قرار میگیرد. این زوجها مشخص میکنند به ازای درست و یا نادرست بودن آن واحد منطقی به چه مقصدهایی باید پرش انجام شود. در روش اخیر امکان میانبر زدن از طریق عملگرهای and و or فراهم شده و کد سریعتری تولید میشود.

```
   L1       L2                  L3        L4    L5
(  A   or   B  )   and   (  n  <= 6  )  True  False
 (L3,L2) (L3,L5)             (L4,L5)
```

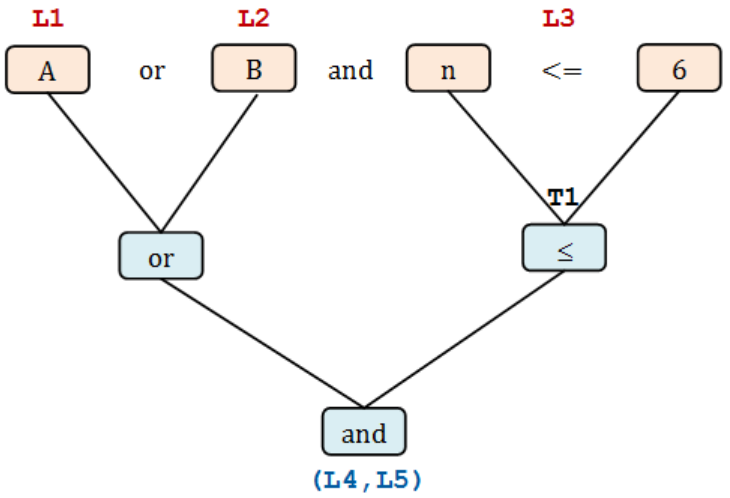
ترفند میانبر زدن: چنانچه در عبارت A and B، مولفه اول False باشد و در عبارت A or B، مولفه اول True باشد نیاز به ارزشیابی مولفه دوم نیست.

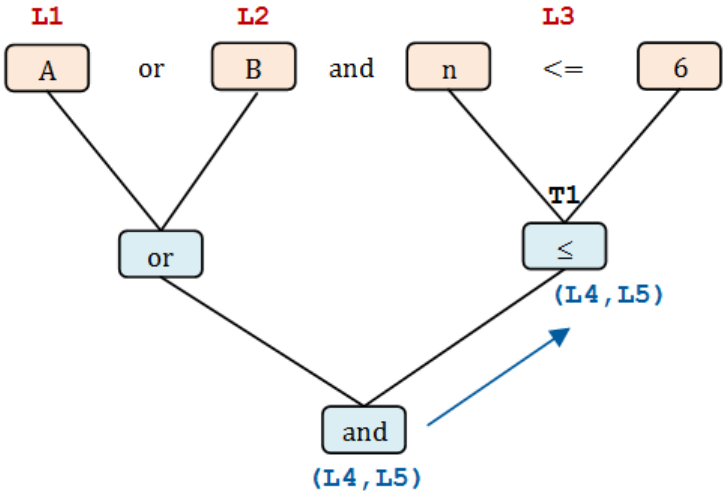
```
L1: (JT, A, L3)
    (J, L2)
L2: (JT, B, L3)
    (J, L5)
L3: (<=, n, 6, T1)
    (JT, T1, L4)
    (J, L5)
L4: {.True.}
L5: {.False.}
```

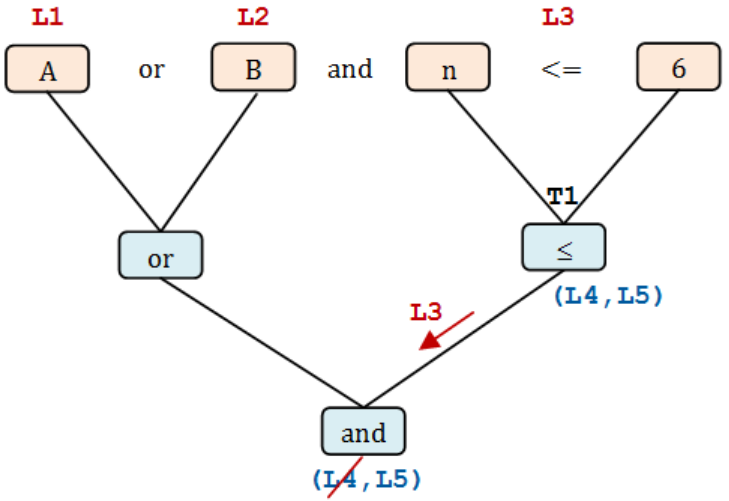
الگوریتم میانبر زدن: ۱- درخت ارزشیابی عبارت را تشکیل بده. ۲- برای هر واحد منطقی عبارت یک برچسب ایجاد کرده و دو برچسب هم برای مقصد درست و نادرست عبارت در نظر بگیر. ۳- برچسبهای ایجاد شده برای مقصدهای درست و نادرست را به یک زوج تبدیل کرده و آنها را به ریشه واگذار کن. ٤- با فرض اینکه (L1 , L2) زوج ارسال شده برای ریشه یک درخت باشد، آن درخت را به صورت زیر پیمایش کن: اگر محتوای گره یکی از عملگرهای and ، or یا not باشد ، قاعده مربوط به آن را اجرا کن وگرنه آن گره یک واحد منطقی است و برچسب ایجاد شده برای آن (در بند ۲) را برای والد خود برگردان.

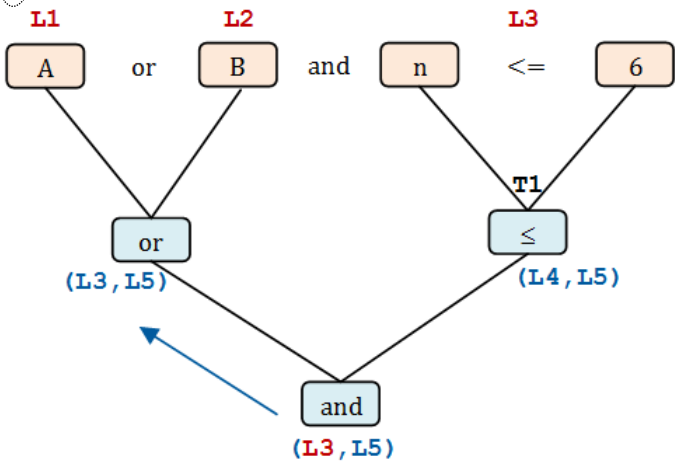


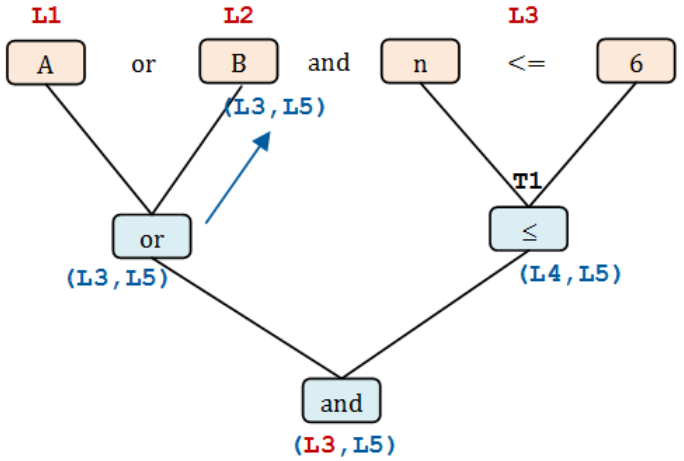
$$\left(\overbrace{\tilde{\tilde{A}}}^{L1} \; or \; \overbrace{\tilde{\tilde{B}}}^{L2}\right) and \left(\overbrace{n \leq 6}^{L3}\right) \qquad \overbrace{True}^{L4} \qquad \overbrace{False}^{L5}$$

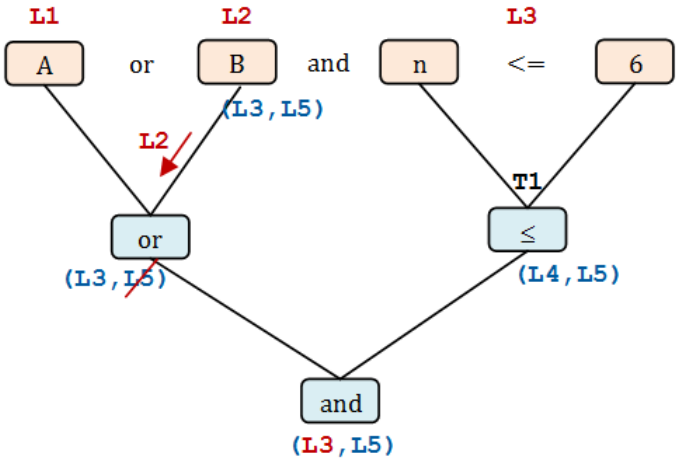


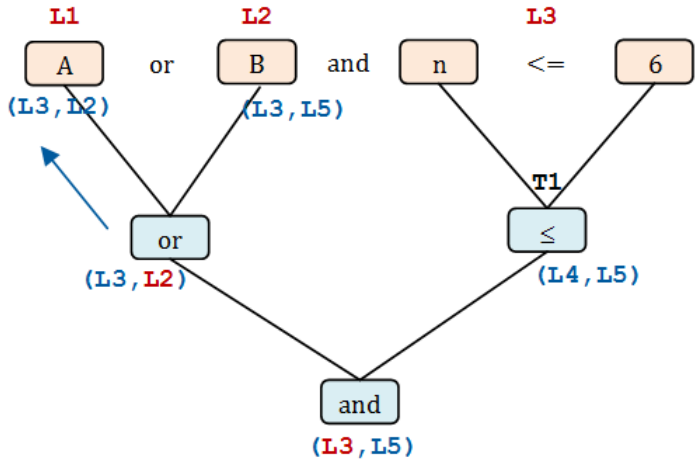





L1
L2
L3
A
or
B
and
n
<=
6
T1
or
≤
(L3,L5)
(L4,L5)
and
(L3,L5)


L1
L2
L3
A
or
B
and
n
<=
6
(L3,L5)
T1
or
≤
(L3,L5)
(L4,L5)
and
(L3,L5)


L1
L2
L3
A
or
B
and
n
<=
6
(L3,L5)
L2
T1
or
≤
(L3,L5)
(L4,L5)
and
(L3,L5)


L1
L2
L3
A
or
B
and
n
<=
6
(L3,L2)
(L3,L5)
T1
or
≤
(L3,L2)
(L4,L5)
and
(L3,L5)

نتیجه اجرای الگوریتم:

```
   L1       L2                    L3        L4    L5
(  A   or    B  )   and   (  n  <= 6  )   True  False
 (L3,L2) (L3,L5)              (L4,L5)
```

تولید کد از روی زوجها: برای هر واحد منطقی با متغیر واقعی و یا موقتی v و زوج به دست آمده (L1,L2)

کد زیر را تولید کن:

```
(JT, V, L1)
(J, L2)
```

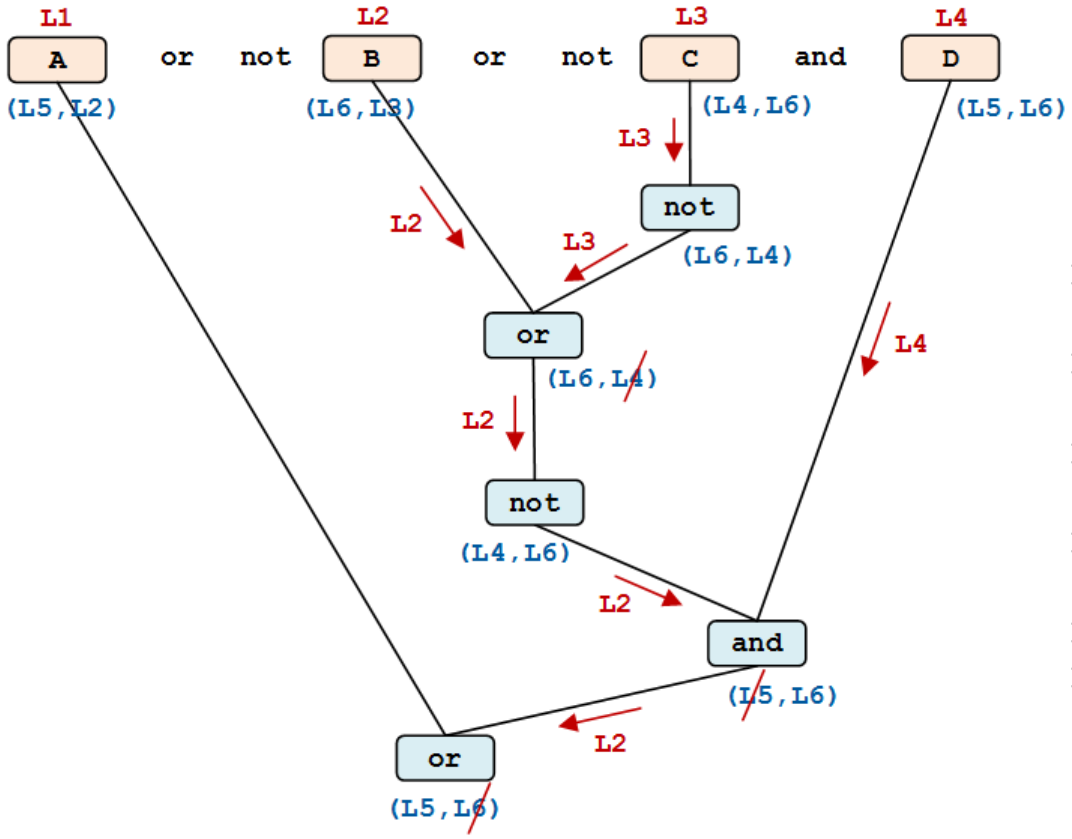
کد تولید شده:

```
L1: (JT, A, L3)
    (J, L2)
L2: (JT, B, L3)
    (J, L5)
L3: (<=, n, 6, T1)
    (JT, T1, L4)
    (J, L5)
L4: {.True.}
L5: {.False.}
```

مثال: عبارت منطقی زیر را به روش میانبر زدن به کد میانی تبدیل کنید:

```
A or not (B or not C) and D
```

$$\overset{L1}{\widetilde{A}}\ or\ not\ (\overset{L2}{\widetilde{B}}\ or\ not\ \overset{L3}{\widetilde{C}})\ and\ \overset{L4}{\widetilde{D}} \qquad \overbrace{True}^{L5} \qquad \overbrace{False}^{L6}$$




```
L1: (JT, A, L5)
    (J, L2)
L2: (JT, B, L6)
    (J, L3)
L3: (JT, C, L4)
    (J, L6)
L4: (JT, D, L5)
    (J, L6)
L5: {.True.}
L6: {.False.}
```

جملات شرطی تک شاخه‌ای:

```
if E then S
```

↓

```
if E JF_L1 then S L1
```

→

```
{E.Code}
(JF, E.Result, L1)
{S.Code}
L1:
```

مثال:

```
if A= 1 then S:= X
```

↓

```
if A= 1 JF_L1 then S:= X
L1
```

→

```
(=, A, 1, T1)
(JF, T1, L1)
(:=, X, S)
L1:
```

جملات شرطی دو شاخه‌ای:

```
    if E then S1
    else S2
```

↓

```
    if E JF_L1 then S1 J_L2
L1  else S2
L2
```

```
      {E.Code}
      (JF, E.Result, L1)
      {S1.Code}
      (J, L2)
L1:   {S2.Code}
L2:
```

مثال:

```
    if A= 1 then S:= X
    else S:= Y
```

↓

```
    if A= 1 JF_L1 then
      S:= X J_L2
L1  else S:= Y
L2
```

```
      (=, A, 1, T1)
      (JF, T1, L1)
      (:=, X, S)
      (J, L2)
L1:   (:=, Y, S)
L2:
```

طقه‌های while do:

```
    while E do S
```

↓

```
L1  while E JF_L2 do S J_L1
L2
```

→

```
L1:  {E.Code}
     (JF, E.Result, L2)
     {S.Code}
     (J, L1)
L2:
```

مثال:

```
    while S<= 10 do
      S:= S+ 1
```

↓

```
L1  while S<= 10 JF_L2 do
      S:= S+ 1 J_L1
L2
```

→

```
L1:  (<=, S, 10, T1)
     (JF, T1, L2)
     (+, S, 1, T2)
     (:=, T2, S)
     (J, L1)
L2:
```

طقه‌های repeat until:

```
repeat S until E
```

↓

```
L1 repeat S until E
JF L1
```

→

```
L1: {S.Code}
    {E.Code}
    (JF, E.Result, L1)
```

مثال:

```
repeat S:= S+ 1
until S> 10
```

↓

```
L1 repeat S:= S+ 1
until S> 10 JF L1
```

→

```
L1: (+, S, 1, T1)
    (:=, T1, S)
    (>, S, 10, T2)
    (JF, T2, L1)
```

طبقه‌های for:

```
for id:= E1 to E2 do S
```

```
for id:= E1 to E2 L1 ≤ JF_L2
do S Inc J_L1 L2
```

```
        {E1.Code}
        (:=, E1.Result, id)
        {E2.Code}
L1:     (<=, id, E2.Result, T1)
        (JF, T1, L2)
        {S.Code}
        (Inc, id)
        (J, L1)
L2:
```

مثال:

```
for i:= A+1 to N-1 do
  S:= S+ 1
```

```
for i:= A+1 to N-1 L1 ≤ JF_L2
do S:= S+ 1 Inc J_L1 L2
```

```
        (+, A, 1, T1)
        (:=, T1, i)
        (-, N, 1, T2)
L1:     (<=, i, T2, T3)
        (JF, T3, L2)
        (+, S, 1, T4)
        (:=, T4, S)
        (Inc, i)
        (J, L1)
L2:
```

فصل اول- بخش بهینه‌سازی کد میانی

چکیده: در این درس با بهینه‌سازی کد میانی که پنجمین فاز کامپایلر است آشنا میشویم. در این فاز کد میانی تولید شده به گونه‌ای دستکاری میشود تا حافظه کمتری اشغال کرده و سرعت اجرای بالاتری داشته باشد.

روشهای بهینه‌سازی

۱- جایگذاری ثابت: عبارتهایی که مقدار آنها در زمان کامپایل مشخص است را میتوان پیش محاسبه کرد تا نیازی به محاسبه آنها در زمان اجرا نباشد:

```
a:= b + 2 * 6    →    a:= b + 12

c:= 2 * 2 + 1    →    c:= 5
d:= c + 1             d:= 6
```

۲- ساده کردن جبری: گاهی با انتخاب عملگرهای کمتر و یا سریعتر برای یک عبارت میتوان سرعت اجرای آن را بالاتر برد:

```
A * B + A * C          →    A * (B + C)
-(A - B)               →    B - A
A - 4 > - A            →    A > 2

(A or B) and (A or C)  →    A or (B and C)
not B or not C         →    not (B and C)
not (A > B)            →    A <= B
A and (not A or B)     →    A and B
A and (A or B)         →    A
```

۳- حذف کد مرده: کدی است که اجرا شده ولی اجرای آن تاثیری در منطق برنامه ندارد.

الف- پرش به دستور بعدی:

```
    Goto L2
L2: A:= B+ C   →   L2: A:= B+ C
```

ب- دو انتساب فوری به یک متغیر:

```
B:= A+ 5
B:= C+ D       →   B:= C+ D
```

(شرط: B نام مستعار برای C یا D نباشد)

٤- حذف کد دسترس ناپذیر: کدی است که هیچگاه اجرا نمیشود و در نتیجه حذف آن تاثیری در منطق برنامه ندارد.

الف- شرطها یا حلقه‌ها

```
if 4 < 5 then X:= 1
else X:= 2;        →   if 4 < 5 then X:= 1
```

```
if 4 < 5 then X:= 1   →   X:= 1
```

(بخش شرط این جمله به دلیل کد مرده بودن حذف میشود)

ب- کدهای بعد از دستور return

```
<1>
return;
<2>
```

۵- بهینه‌سازی جریان انتقال:

الف- پرش از روی پرش:

```
     If C goto L2                    Ifnot C goto L3
     Goto L3          →      L2:     ....
L2:  ....
```

ب- پرش به پرش دیگر:

```
     Goto L1                          Goto L2
     ....                             ....
L1:  Goto L2          →       L1:     Goto L2
     ....                             ....
L2:                             L2:
```

```
     If C goto L1                     If C goto L2
     ....                             ....
L1:  Goto L2          →       L1:     Goto L2
     ....                             ....
L2:                             L2:
```

٦- انتشار کپی:

```
a:= b                          a:= b
c:= a+ 1          →            c:= b+ 1
```

اگر a بدون استفاده شود جمله a:=b حذف میشود

۷- فاکتورگیری از عبارتهای مشترک: اگر یک عبارت چند بار در کد تکرار شده باشد، به شرط آنکه در هر تکرار نتیجه یکسانی ایجاد کند میتوان آن را فقط یکبار محاسبه کرد.

```
A:= B + C + D
E:= B + C + F
```

→

```
T:= B + C
A:= T + D
E:= T + F
```

(شرط: A نام مستعار برای B یا C نباشد)

```
A[i]:= B[i]+ C[i]
```

آدرس A[i]= α+ 4*i
آدرس B[i]= β+ 4*i
آدرس C[i]= γ+ 4*i

میتوان از 4*i فاکتورگیری کرد

۸- بهینه‌سازی حلقه: عبارتهایی را که در هر بار اجرای یک حلقه نتیجه یکسانی را ایجاد میکنند به نقطه‌ای درست قبل از حلقه انتقال بدهیم. در این صورت این محاسبه تنها یک بار انجام میشود.

```
for i:= 1 to n do
  S:= S + i * (A + Sin(B))
```

→

```
T:= A + Sin(B);
for i:= 1 to n do
  S:= S+ i* T;
```

شرایط:

- حلقه دست کم باید 1 مرتبه اجرا شود.
- متغیرهای i و S نباید نام مستعار برای A یا B باشند.
- در ثابت حلقه نباید تابعی وجود داشته باشد که باعث تغییر در متغیرهای سراسری شود.

مثال:

```
P := I + R * 60
```

```
(IntToFloat, 60, T1)
(*, R, T1, T2)
(+, I, T2, T3)
(:=, T3, P)
```

→

```
(*, R, 60.0, T2)
(+, I, T2, P)
```

مثال:

```
   L1       L2                    L3        L4    L5
(  A   or   B  )   and   (  n  <= 6  )   True  False
 (L3,L2) (L3,L5)           (L4,L5)
```

```
L1: (JT, A, L3)
    (J, L2)
L2: (JT, B, L3)
    (J, L5)
L3: (<=, n, 6, T1)
    (JT, T1, L4)
    (J, L5)
L4: {.True.}
L5: {.False.}
```

→

```
L1: (JT, A, L3)
L2: (JF, B, L5)
L3: (<=, n, 6, T1)
    (JF, T1, L5)
L4: {.True.}
L5: {.False.}
```

شناسایی توکن

چکیده: در این درس با مطرح کردن دیدگاه دستی در طراحی اسکنر، وظیفه اصلی اسـکنر را بـدون در نظـر گرفتن وظایف فرعی آن مورد بررسی قرار میدهیم.

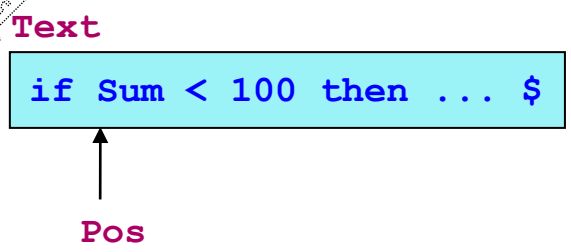
وظایف اسکنر

- تشخیص توکن (وظیفه اصلی)
- خواندن کاراکترها از فایل ورودی

خواندن از فایل ورودی

- با هر بار مراجعه به دیسک یک یا یک گروه از کاراکترها خوانده شود.
- با یک بار مراجعه به دیسک همه فایل خوانده شده و در حافظه بارگذاری شود.

تمرکز روی وظیفه اصلی: از آنجاییکه ورودی اسکنر (برنامه زبان مبدا) یک فایل متنی است دیدگاه دوم مناسبتر است. بنابراین در ادامه وظیفه اصلی اسکنر را بدون درگیر شدن با فایل ورودی بررسی میکنیم.



اجزای Inp

- `Text` : رشته ورودی را نگهداری میکند
- `Pos` : مکان‌نمای رشته است

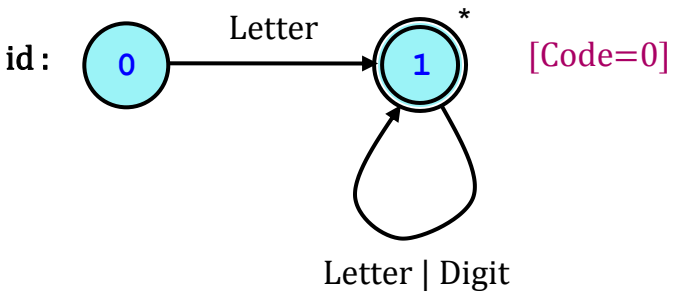
خواندن کاراکتر اضافه‌تر: در تشخیص توکن گاهی لازم است یک کاراکتر بیش از توکن دریافت شود. برای نمونه برای تشخیص شناسه باید یک کاراکتر غیر از حرف و رقم دریافت شود.

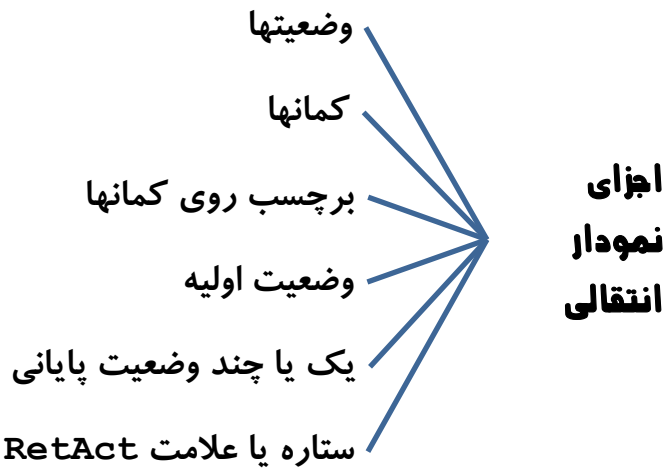
کاراکتر انتها: از آنجاییکه خواندن کاراکتر از یک مکان خالی خطای زمان اجرا ایجاد میکند لازم است در انتهای رشته ورودی کاراکتری که جزو هیچ یک از کاراکترهای توکن نیست (در اینجا $) قرار بگیرد.

متدهای Inp

- `GetChar` : کاراکتر بعدی را برگردانده و `Pos` را یکی جلو میبرد.
- `RetAction` : شرایط قبلی توکن را بازیابی میکند (در مواردی که یک کاراکتر اضافه‌تر دریافت شود).

نمودار انتقالی: روشی برای تشخیص توکن است و با افزودن یک الگوریتم ساده، به راحتی بـه برنامـه اسـکنر تبدیل میشود.

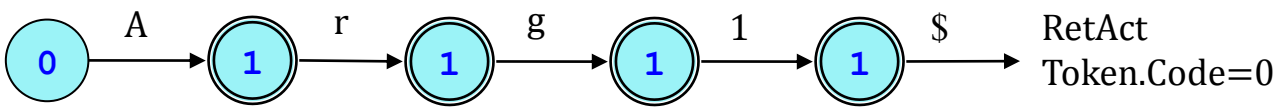




تشخیص توکن: پیدا کردن مسیری از وضعیت اولیه به یکی از وضعیتهای پایانی.

شرط ستاره‌دار بودن: چنانچه از وضعیت آخر یک نمودار کمانی خارج شود، لازم است آن وضعیت سـتاره‌دار شود (یک کاراکتر اضافه‌تر از توکن دریافت شود).

مثال: شکل زیر مشخص میکند برای دریافت شناسه Arg1 چه مسیری در نمودار دنبال میشود:



ابهام: توکن اعلام شده میتواند هر یک از رشته‌های Arg1 , Arg , Ar , A باشد ، ولـی چـرا Arg1 انتخـاب میشود؟

قانون بلندترین تطبیقها: اگر رشته A پیشوندی از رشته B باشد، B نسبت به A تقدم دارد و در مثال اخیر رشته Arg1 نسبت به سایر رشته‌ها تقدم داشته و باید توکن اعلام شده باشد.

مثال: کدام رشته تقدم دارد؟

abcd , ab → abcd

abcd , cd → هیچکدام

abcd , bc → هیچکدام

الگوریتم شناسایی توکن: با فرض اینکه (S1 , Ch , S2) یک انتقال از S1 به S2 با برچسب Ch باشد:

۱- متغیر S را برابر وضعیت اولیه نمودار قرار بده.

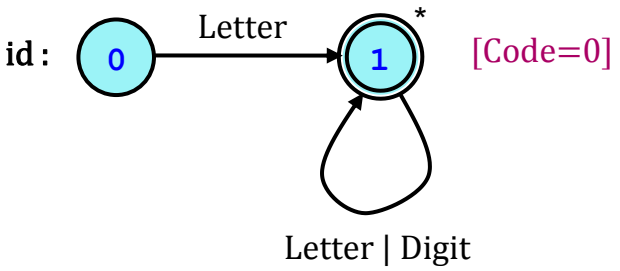
۲- تا زمانیکه توکن تشخیص داده نشده عملیات زیر را تکرار کن:

- اگر S یک وضعیت غیرپایانی باشد: کاراکتر بعدی را دریافت کن (Ch) ، اگر انتقالی با فرمت (S , Ch , S2) در نمودار موجود باشد S را برابر S2 قرار بده وگرنه یک کد نامعتبر برگردان.
- اگر S یک وضعیت پایانی ستاره‌دار باشد : کاراکتر بعدی را دریافت کن (Ch) ، اگر انتقالی با فرمت (S , Ch , S2) در نمودار موجود باشد S را برابر S2 قرار بده وگرنه عمل RetAct را انجام داده و کد توکن مربوط به آن وضعیت پایانی را برگردان.
- اگر S یک وضعیت پایانی بدون ستاره باشد : کد توکن مربوط به آن وضعیت پایانی را برگردان.

```
function FindAToken: Integer;
var
  State: Integer;  Ch: Char;
begin
  State:= 0;
  repeat
    case State of
      0:
        { some code }
      1:
        { some code }
      ...
    end;
  until False;
end;
```

```
int FindAToken()
{
  int State= 0; Char Ch;
  do
    switch (State)
    {
      case 0:
        // some code
        break;
      case 1:
        // some code
        break;
      ...
    }
  while (True);
}
```

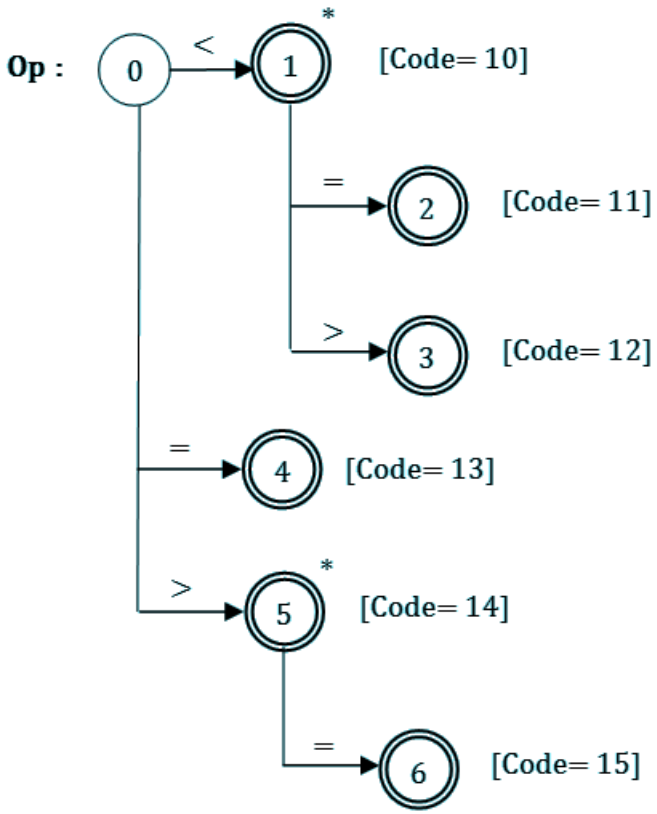
مثال: نمودار شناسه‌ها را کدنویسی کنید.




```
function InpFindId: Integer;
var
  State: Integer;  Ch: Char;
begin
  State:= 0;
  repeat
    case State of
      0:
        begin
          Ch:= Inp.GetChar;
          if Ch.IsLetter then State:= 1
          else Exit(-1);
        end;
      1:
        begin
          Ch:= Inp.GetChar;
          if Ch.IsLetterOrDigit then State:= 1
          else begin
            Inp.RetAction;
            Exit(0)
          end;
        end;
    end;
  until False;
end;
```

```
function InpFindOp: Integer;
var
  State: Integer;  Ch: Char;
begin
  State:= 0;
  repeat
    case State of
      0:
        begin
          Ch:= Inp.GetChar;
          if Ch= '<' then State:= 1
          else if Ch= '=' then State:= 4
          else if Ch= '>' then State:= 5
          else Exit(-1);
        end;
      1:
        begin
          Ch:= Inp.GetChar;
          if Ch= '=' then State:= 2
          else if Ch= '>' then State:= 3
          else begin
            Inp.RetAction;
            Exit(10)
          end;
        end;
      2: Exit(11);
      3: Exit(12);
      4: Exit(13);
      5:
        begin
          Ch:= Inp.GetChar;
          if Ch= '=' then State:= 6
          else begin
            Inp.RetAction;
            Exit(14)
          end;
        end;
      6: Exit(15);
    end;
  until False;
end;
```

مثال: op= {<, <=, <>, =, > , >=}

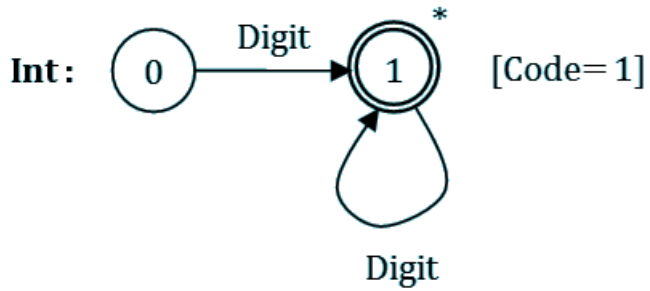


نمودار انتقالی توکنها

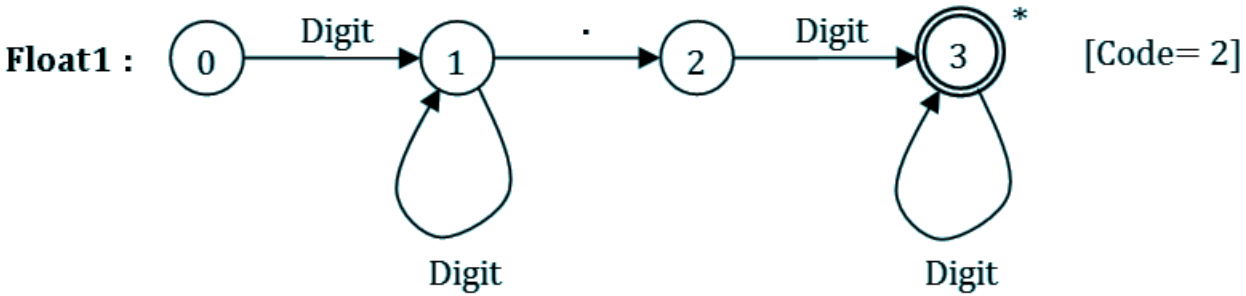
چکیده: در این درس نمودار انتقالی توکنهای باقاعده (شناسه‌ها، اعداد و رشته‌ها) ، توکن‌نماها (فضاهای خالی و کامنتها) و برخی از توکنهای بی‌قاعده (مثل کلمه‌های رزرو شده و نمادهای عملگر) را معرفی میکنیم.

نمودار شناسه‌ها: قبلا معرفی شده است.

نمودار اعداد صحیح:

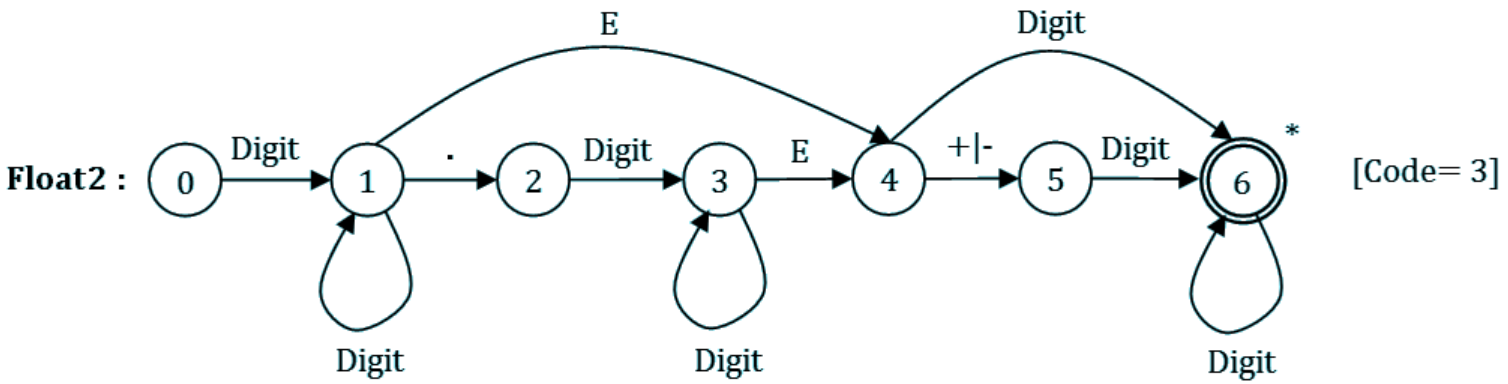


نمودار اعداد اعشاری ساده:



نمودار اعداد اعشاری کامل: عدد اعشاری کامل عددی است که بخش نمایی دارد. بخش نمایی یک بخش صحیح (علامتدار یا بدون علامت) است که در انتهای عدد و پس از نماد E قرار میگیرد.

```
12.34e4          12.34e+3
12.34e-3         1234e-02
```

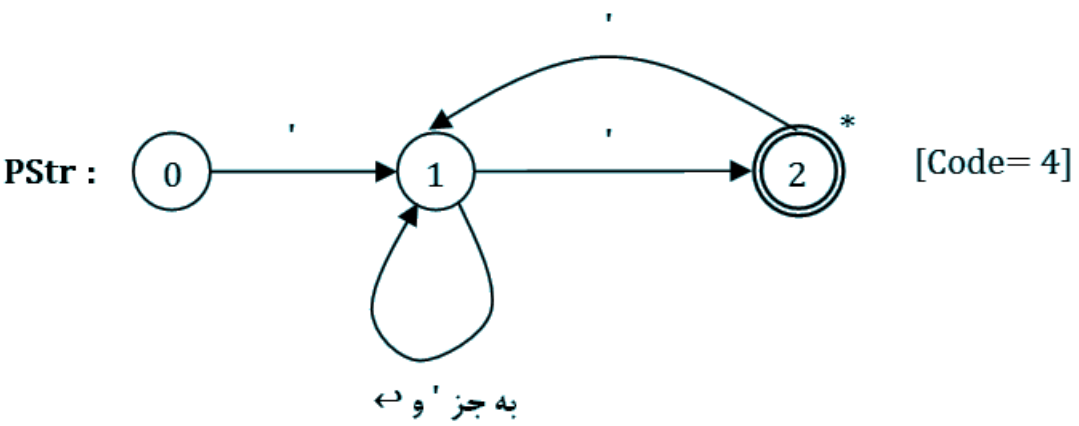


چرا نمودار اعداد علامتدار نیست؟ چون اگر نمودار اعداد علامتدار باشد هر علامت (-/+) قبل از عدد حتی اگر آن علامت عملگر دودویی باشد جزو عدد دریافت میشود. در نتیجه دریافت عبارتها مشکل‌ساز میشود.

```
A-15   →   A , -15          اگر علامت جزو عدد باشد.
A-15   →   A , - , 15       اگر علامت جزو عدد نباشد.
```

نمودار رشته‌های زبان پاسکال: رشته‌ها در زبان پاسکال (و دلفی) به صورتهای محدود به کوتیشن ، کنترلی و مخلوط به کار گرفته میشوند. ولی یک رشته محدود به کوتیشن با ' شروع شده ، با ' هم تمام میشود و محدود به یک سطر میباشد. برای استفاده از ' در رشته باید یک ' دیگر بعد از آن بیاید (به بیان دیگر هر دو ' کنار هم یک ' محسوب میشود).

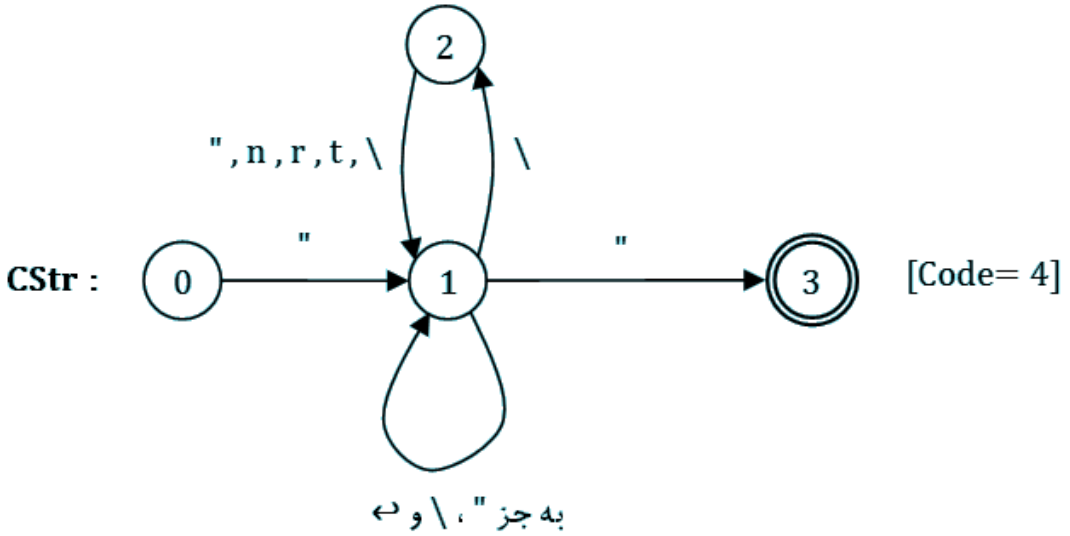
```
''' 1 2 " 3 4 '''   →   ' 1 2 " 3 4 '
'a''b''''c'         →   a'b''c
```



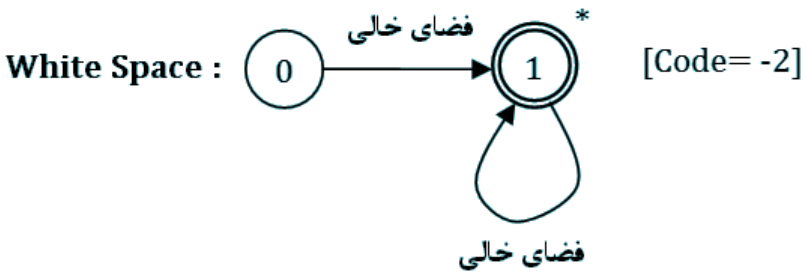
نمودار رشته‌های زبان سی: یک رشته در زبان سی با " شروع شده، با "هم تمام میشود و محدود به یک سطر میباشد. کاراکتر \ در رشته یک کاراکتر خاص است و بعد از آن کاراکترهای ویژه‌ای از جمله "، \و n قرار میگیرد. بنابراین اگر Ch کاراکتر بعد از \ باشد، آنگاه دنباله \Ch به یک کاراکتر جدید تفسیر شده و به خروجی ارسال میشود.

Seq	Char	Unicode
\"	"	
\n	↩	000A
\r	↩	000D
\t	Tab	0009
\\	\	

```
"ab\"cd\\ef"   →   ab"cd\ef
"\"\\\\b"      →   "\\b
"a\nb'c'd"     →   a
                   b'c'd
```



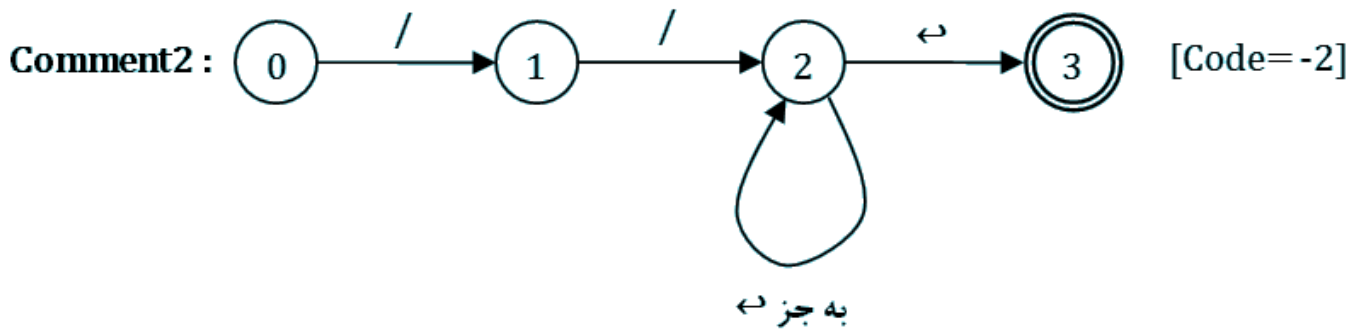
نمودار فضاهای خالی:




```
$0020 {SPACE}
$0009 {TAB}
$000A {LF}
$000B {LINE-TAB}
$000C {FF}
$000D {CR}
$00A0 {No-break Space}
$0085 {NEL}
```

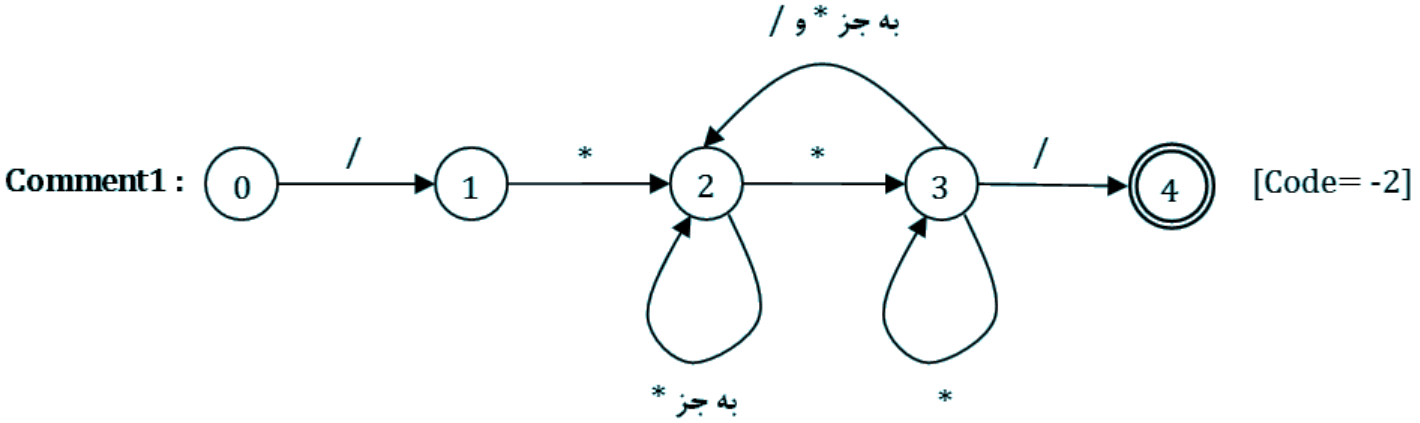
نمودار کامنتهای تک سطر: با // شروع شده و با کاراکتر انتهای خط هم تمام میشوند.

```
// 3333
/// Only /* One // Comment
```



نمودار کامنتهای چند سطر: با */ شروع شده و با /* هم تمام میشوند.

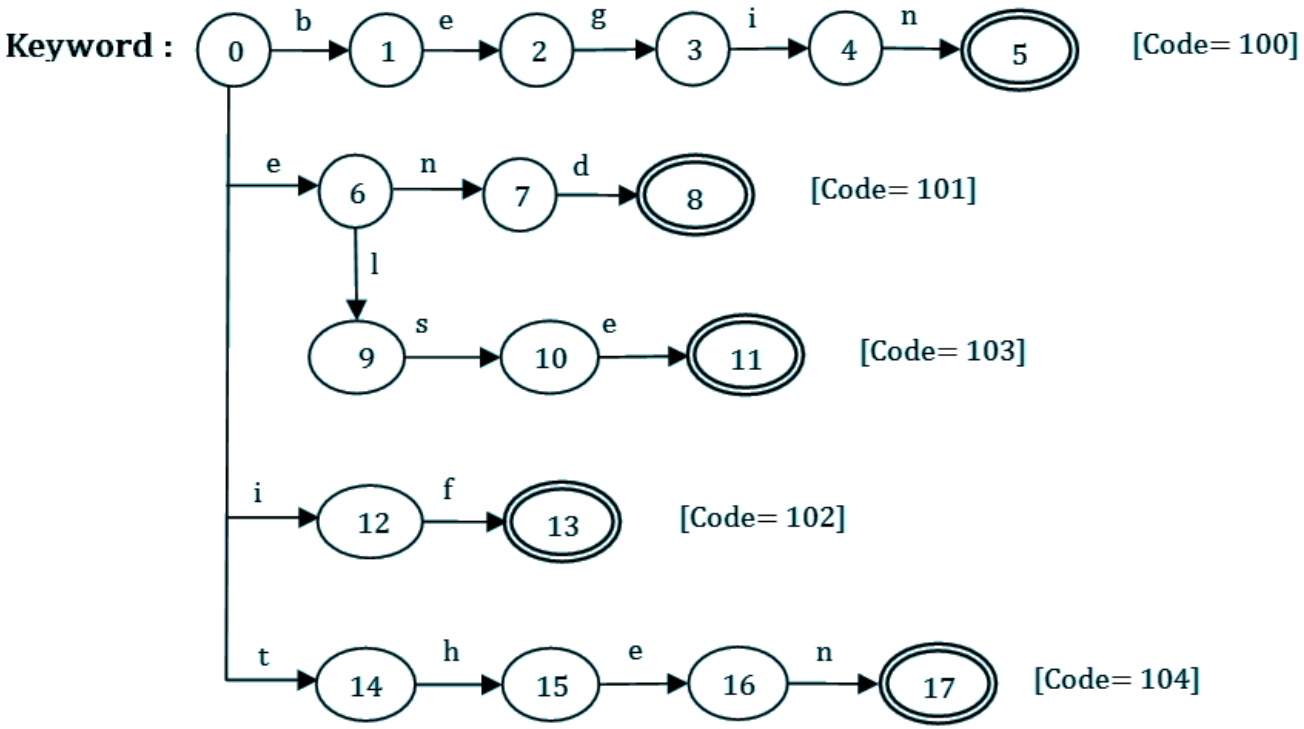
```
/*                  /*a*b***/
  Line1
  Line2             /**c**/
*/                  /*2/*2*2//2*/
```



نمودار عملگرها: قبلا معرفی شده است.

نمودار کلمه‌های رزرو شده: نمودار زیر میتواند کلمه‌های رزرو شده زیر را دریافت کند. بدیهی است کلمه‌های رزرو شده در یک زبان چند برابر این تعداد است و از یک زبان تا زبان دیگر هم تفاوت دارد.

```
Keywods= {begin, end, if, then, else}
```



ترکیب نمودارهای انتقالی

چکیده: در این درس نمودارهای طراحی شده برای توکنهای متداول را در هم ادغام میکنیم تا به یک نمـودار واحد برسیم. از آنجاییکه عمل ترکیب به صورت دستی انجام میگیرد نکات مربوط به ترکیب آنها را یک بـه یک بررسی میکنیم.

روش عقبگرد: وقتی چند نمودار انتقالی برای شناسایی توکنها داشته باشیم نمیدانیم کدام یک را دنبال کنیم. در این حالت مجبوریم با یکی از نمودارها شروع کرده و در صورت شکست از آن خارج شده و نمودار دیگری را انتخاب کنیم.

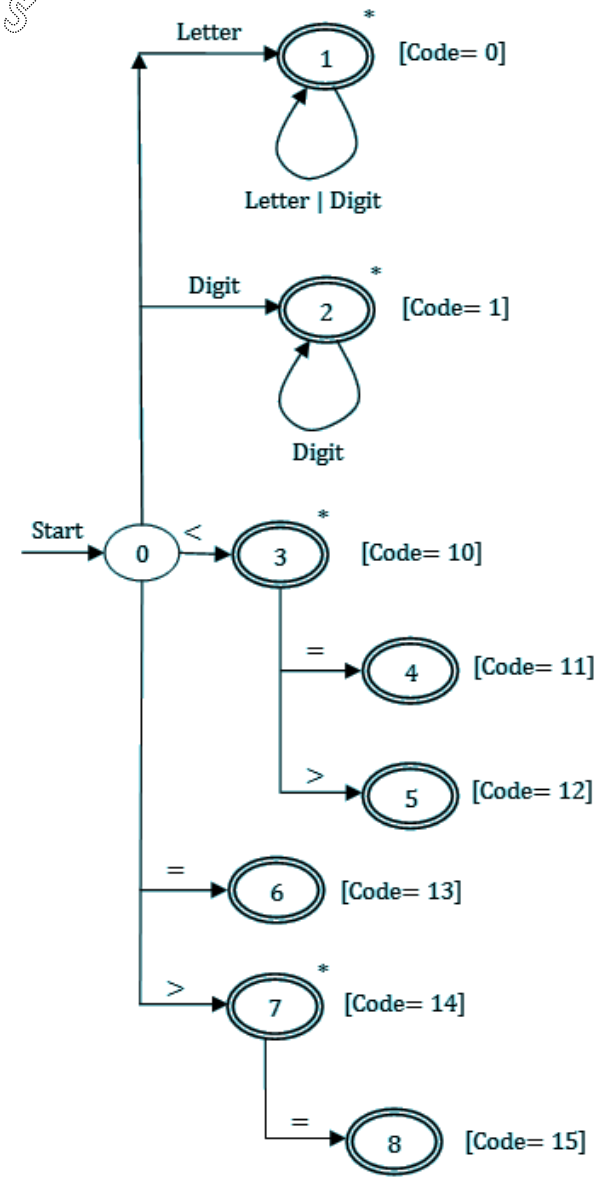
$$Token_1 \xrightarrow{\text{دستی}} diag_1$$
$$Token_2 \xrightarrow{\text{دستی}} diag_2$$
$$\dots$$
$$Token_n \xrightarrow{\text{دستی}} diag_n$$

ترکیب نمودارها: برای قطعی کردن روش تشخیص کافی است نمودارهای انتقالی را با هم ترکیب کنیم تا نیاز به عقبگرد و پیمایش نمودار بعدی از بین برود.

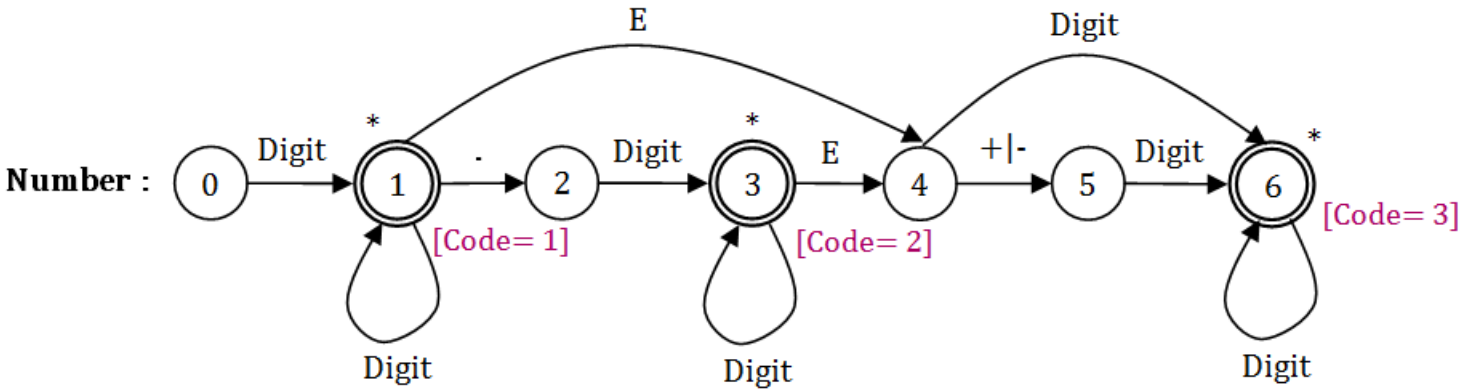
$$\left.\begin{array}{l} Token_1 \xrightarrow{\text{دستی}} diag_1 \\ Token_2 \xrightarrow{\text{دستی}} diag_2 \\ \dots \\ Token_n \xrightarrow{\text{دستی}} diag_n \end{array}\right\} \xrightarrow{\text{ادغام (دستی)}} Diag$$

ترکیب سه نمودار شناسه ،عدد صحیح و عملگرها:

ترکیب برخی از نمودارها بسیار آسان است برای نمونه شناسه‌ها ، اعداد صحیح و عملگرها به سادگی زیر با هم ترکیب میشوند.

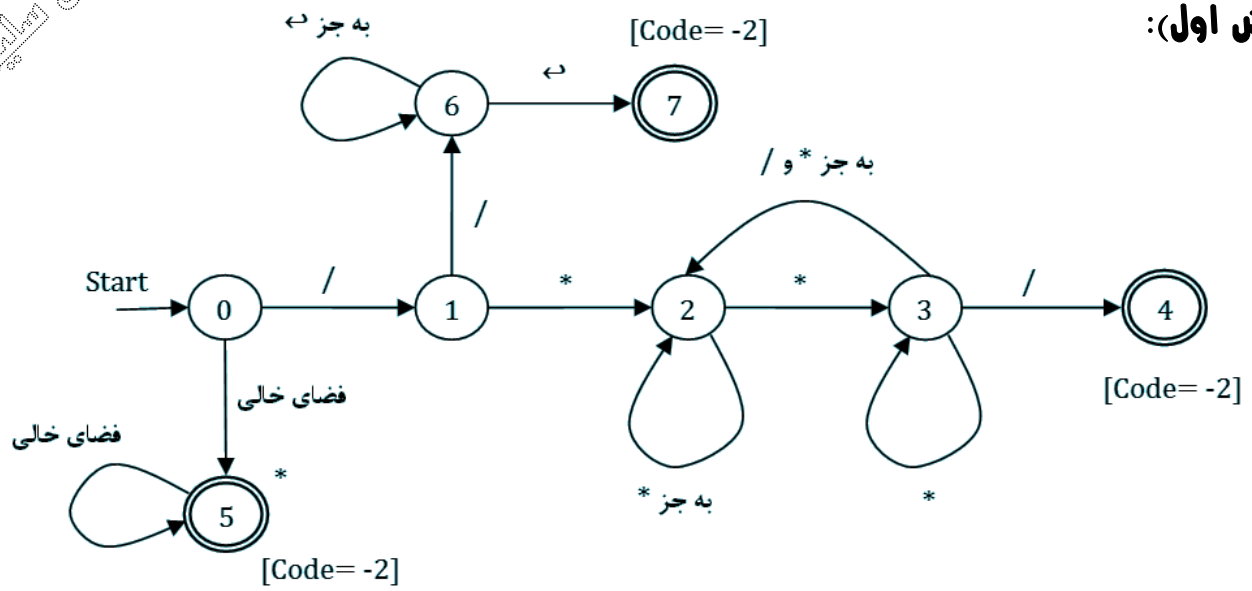


ترکیب نمودار اعداد: ترکیب سه نمودار عدد صحیح، اعشاری ساده و اعشاری کامل هم به صورت زیر است:



مشاهده میکنید که نمودار ترکیبی همانند نمودار اعداد اعشاری کامل است، با این تفاوت که وضعیتهای ۱ و ۳ به پایانی ستاره‌دار تبدیل شده‌اند.

ترکیب توکن‌نماها(روش اول):



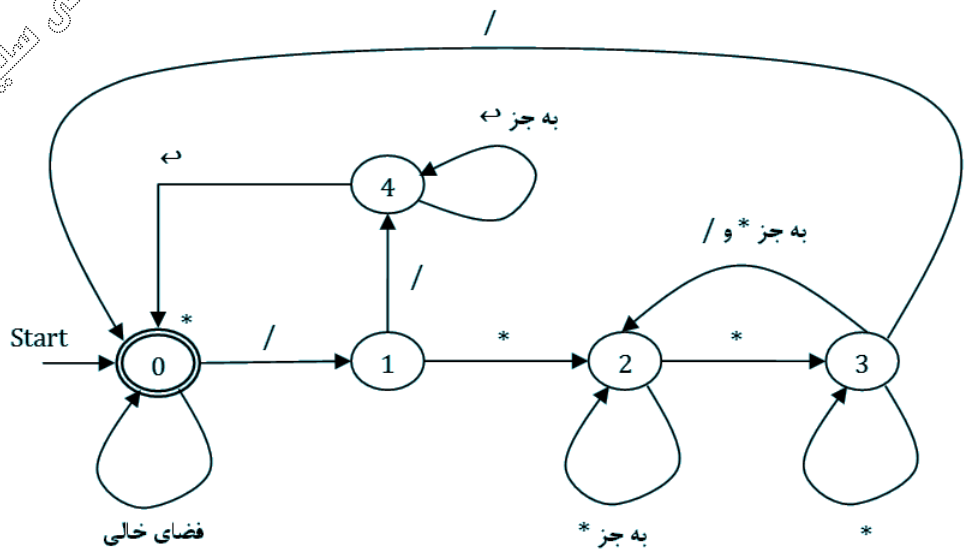
ویژگیها:

- در هر فراخوانی فقط یک توکن‌نما را شناسایی میکند.
- به راحتی میتوان آن را با سایر نمودارها ترکیب کرد.

روش کاربرد: با دریافت یک توکن‌نما نباید کدی به تجزیه‌گر ارسال شود.

```
repeat
  T:= FindNextToken;
until T.Code<> -2;
```

ترکیب توکن‌نماها(روش دوم):



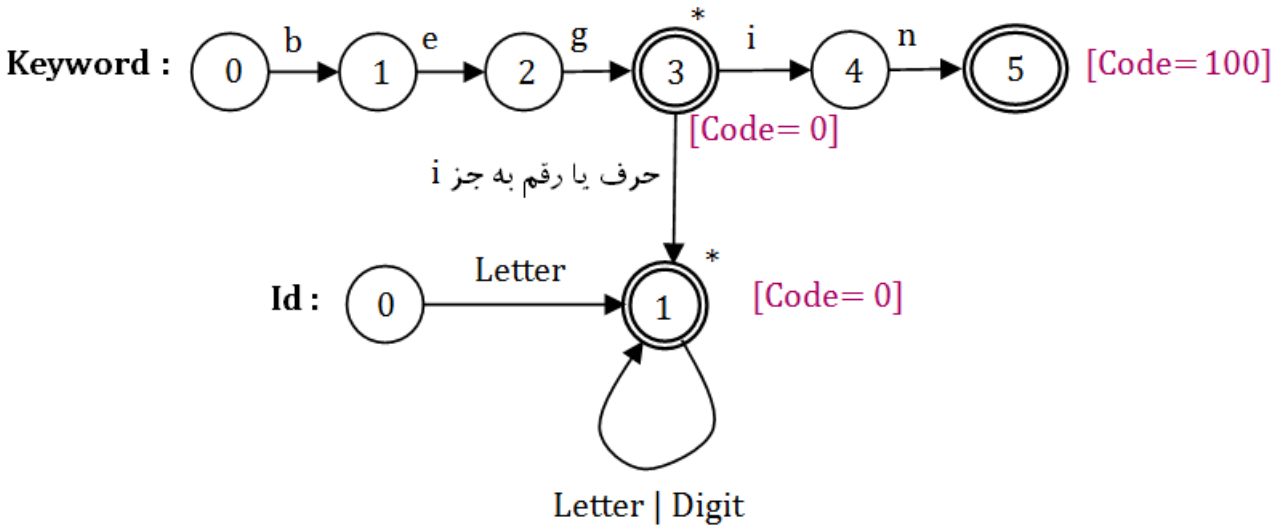
ویژگی: همه فضاهای خالی و کامنتهای پشت سر هم شناسایی شده و رد میشوند.

عیب: ترکیب آن با سایر نمودارها اشکال ایجاد میکند. برای نمونه اگر با نمودار Id ترکیب شود و رشته ورودی به صورت `/*Comment*/Max123` باشد، مشخصات (`Value,Code`) توکن شناسایی شده به صورت (`/*Comment*/Max123,IdCode`) خواهد بود. یعنی مقدار توکن برخلاف کد آن نادرست است.

روش کاربرد: با فرض اینکه نمودار توکن‌نماها `FindUnread` باشد روش کاربرد آن به صورت زیر است:

```
FindUnread;
T:= FindNextToken;
```

ترکیب شناسه‌ها و کلمه‌های رزرو شده: شکل زیر ترکیب این دو نمودار را در وضعیت ۳ نشان میدهد. مشاهده میکنید که این وضعیت به یک وضعیت پایانی ستاره‌دار تبدیل شده است.



جایگزین ترکیب: کلمه‌های رزرو شده همانند شناسه‌ها فرض شده و وقتی اسکنر واژه‌ای را تحت عنوان شناسه تشخیص میدهد توسط تابع `IsKeyword` کلمه رزرو بودن آن را بررسی میکند.

```
T:= FindNextToken;
if T.Code= 0 then
  IsKeyword(T);
```

تعریف زبان و معرفی گرامرها

چکیده: در این درس زبان را تعریف کرده و گرامرها را به عنوان یک روش مهم در توصیف رشته‌های یک زبان معرفی میکنیم. سپس با روند تولید یا بسط جمله‌های زبان آشنا میشویم. تعریفها و مفاهیم به کار رفته در این درس مقدمه‌ای در طراحی تجزیه‌گر خواهند بود.

الفبا: یک مجموعه متناهی از نمادها (یا کاراکترها) را الفبا میگویند.

$\Sigma = \{a, b, c\}$

رشته: دنباله‌ای از نمادهای یک الفبا را رشته (جمله و یا کلمه) میگویند.

$x = aab$ $\quad\quad$ $z = abccbacb$

$y = baca$ $\quad\quad$ $t = \in$

زبان: مجموعه‌ای از رشته‌هایی است که از الفبای خاصی به دست میآیند.

$L1 = \{a, ccab\}$ $\quad\quad$ $L3 = \{\}$

$L2 = \{\in, ca, bcb\}$ $\quad\quad$ $L4 = \{\in\}$

$L5 = \{\in, a, aa, aaa, aaaa, ...\}$

$L6 = \{ab, aabb, aaabbb, aaaabbbb, ...\}$

روشهای توصیف زبان

- **عبارتهای منظم**: قدرت توصیف پایین‌تری دارند و اغلب در تعریف توکنهای یک زبان برنامه‌نویسی استفاده میشوند.
- **گرامرها**: قدرت توصیف بالاتری دارند و اغلب در تعریف ساختارهای نحوی زبانهای برنامه‌نویسی به کار میروند.

اجزای گرامر

- **نمادهای پایانی (∑):** که الفبای زبان را تشکیل میدهند.
- **نمادهای غیرپایانی (N):** که بر حسب سایر نمادها تعریف میشوند.
- **قواعد زبان (P):** که غیرپایانه‌ها را بر حسب سایر نمادها تعریف میکنند. قواعد زبان با فرمت $\alpha \rightarrow \beta$ تعریف میشوند که در آن α و β رشته‌ای از نمادها هستند.
- **نماد شروع (S):** که عضو غیرپایانی‌های زبان است و اولین قاعده گرامر با آن شروع میشود.

مثال۱:

$\sum = \{ a , b \}$
$N = \{ S \}$
$S = \{ S \}$

$S \rightarrow aa$
$S \rightarrow bb$
$S \rightarrow aSa$
$S \rightarrow bSb$

$X \rightarrow \beta1 , X \rightarrow \beta2 , \dots , X \rightarrow \beta n$
$X \rightarrow \beta1 \mid \beta2 \mid \dots \mid \beta n$

$S \rightarrow aa \mid bb \mid aSa \mid bSb$

مثال۲:

$\sum = \{ (,) , id , + , - , * , / , ^ \}$
$N = \{ E , A \}$
$S = \{ E \}$

$E \rightarrow EAE \mid (E) \mid -E \mid id$
$A \rightarrow + \mid - \mid * \mid / \mid ^$

طبقه‌بندی گرامرها

نامحدود: به صورت $\alpha \rightarrow \beta$ نوشته میشوند. هیچ محدودیتی روی β نیست ولی α نمیتواند تهی باشد. مثال: $abc \rightarrow ac$ و $aSb \rightarrow Sab$

حساس به متن: به صورت $\alpha \rightarrow \beta$ نوشته میشوند. $|\beta| \geq |\alpha|$ و α دست کم شامل یک غیرپایانه است. مثال: $aSa \rightarrow abba$

مستقل از متن: به صورت $X \rightarrow \beta$ نوشته میشوند. X یک غیرپایانه است. در توصیف ساختارهای نحوی زبانهای برنامه‌نویسی استفاده میشوند. مثال: $S \rightarrow aSa$

منظم: به صورت $X \rightarrow \alpha$ یا $X \rightarrow \alpha Y$ (خطی راست) نوشته میشوند. X و Y غیرپایانه و α یک رشته (با طول صفر یا بیشتر) از پایانه‌هاست. معادل عبارتهای منظم هستند و در توصیف توکنهای زبان برنامه‌نویسی استفاده میشوند. مثال: $S \rightarrow abb$ ، $S \rightarrow baS$ و $S \rightarrow \in$

قدرت تولیدی گرامر: قدرت تولیدی یک گرامر مجموعه جملاتی است که تولید میکند. گرامرهای نامحدود دارای بیشترین قدرت تولیدی هستند و گرامرهای منظم کمترین قدرت تولیدی را دارند.

جایگزینی: چنانچه α_j با به کارگیری یک قاعده گرامر از α_i نتیجه شود، مینویسیم $\alpha_i \Rightarrow \alpha_j$. و چنانچه طی صفر یا چند جایگزینی از α_i به دست بیاید (مشتق شود) مینویسیم $\alpha_i \overset{*}{\Rightarrow} \alpha_j$. و چنانچه طی یک یا چند جایگزینی از α_i به دست بیاید مینویسیم $\alpha_i \overset{+}{\Rightarrow} \alpha_j$.

```
E → EAE | (E) | -E | id
A → + | - | * | / | ^
```

$$-(E) \Rightarrow -(EAE)$$

$$-(E) \overset{+}{\Rightarrow} -(id + E)$$

روند تولید یا بسط جملات: جمله w و گرامر G داده شده‌اند. میخواهیم بدانیم آیا w عضو L(G) هست یا نه. برای این منظور باید بتوانیم جمله w را از نماد شروع گرامر مشتق کنیم. در روند اشتقاق (بسط)، از نماد S شروع کرده و با انتخاب قواعد مناسب غیرپایانه‌ها را بر اساس سایر نمادها توسعه میدهیم تا در پایان جمله دلخواه که فقط شامل پایانه‌هاست به دست بیاید.

مثال: جمله w را با گرامر زیر تولید کنید.

```
w = bababbabab
```

```
S → aa | bb | aSa | bSb
```

```
S ⇒ bSb ⇒ baSab ⇒ babSbab ⇒ babaSabab ⇒ bababbabab
```

بسط چپ و راست: هرگاه در روند تولید یک جمله در هر جایگزینی سمت چپ‌ترین غیرپایانه جایگزین شود به آن بسط چپ (LMD) و اگر سمت راست‌ترین غیرپایانه جایگزین شود به آن بسط راست (RMD) میگویند.

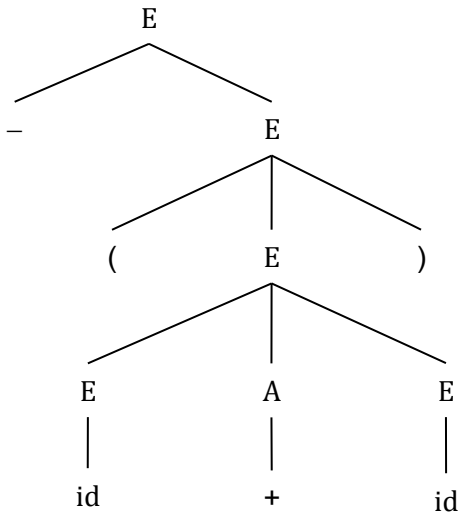
مثال: برای گرامر زیر، جمله w را با روند بسط چپ و راست تولید کنید.

w = -(id + id)

E → EAE | (E) | -E | id
A → + | - | * | / | ^

LMD: E ⇒ -E ⇒ -(E) ⇒ -(EAE) ⇒ -(id A E) ⇒ -(id + E) ⇒ -(id + id)
RMD: E ⇒ -E ⇒ -(E) ⇒ -(EAE) ⇒ -(E A id) ⇒ -(E + id) ⇒ -(id + id)

درخت تجزیه: نمایش گرافیکی بسط یک جمله است.



گرامرهای مبهم

چکیده: در این درس گرامر مبهم را تعریف کرده و برای ساختارهایی مثل عبارتها که از عملگرهای یگانی و دودویی تشکیل میشوند یک گرامر بدون ابهام می‌نویسیم.

گرامر مبهم: هرگاه برای یک جمله دو یا چند بسط چپ (یا راست) وجود داشته باشد گرامر مورد نظر گنگ یا مبهم است. به بیان دیگر اگر گرامری برای یک جمله بیش از یک درخت تجزیه داشته باشد مبهم است.

مثال: برای گرامر زیر، جمله **w** را با روند بسط چپ تولید کنید.

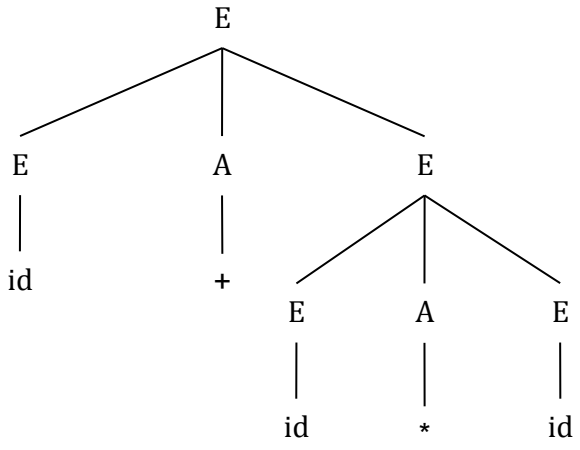
```
w = id + id * id
```

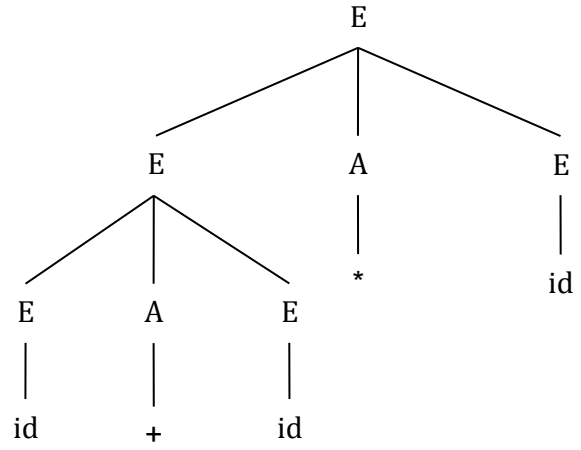
$E \rightarrow EAE \mid (E) \mid -E \mid id$

$A \rightarrow + \mid - \mid * \mid / \mid \hat{}$

LMD1: $E \Rightarrow EAE \Rightarrow idAE \Rightarrow id+E \Rightarrow id+EAE \Rightarrow id+idAE \Rightarrow id+id*E \Rightarrow id+id*id$

LMD2: $E \Rightarrow EAE \Rightarrow EAEAE \Rightarrow idAEAE \Rightarrow id+EAE \Rightarrow id+idAE \Rightarrow id+id*E \Rightarrow id+id*id$





ساختارهای مبهم

- عبارتهای ریاضی و منطقی
- جمله‌های شرطی تو در تو

```
id + id * id
```

```
id + (id * id)
```

```
(id + id) * id
```

```
if E1 then if E2 then S1
else S2
```

```
if E1 then
  if E2 then S1
  else S2
```

```
if E1 then
  if E2 then S1
else S2
```

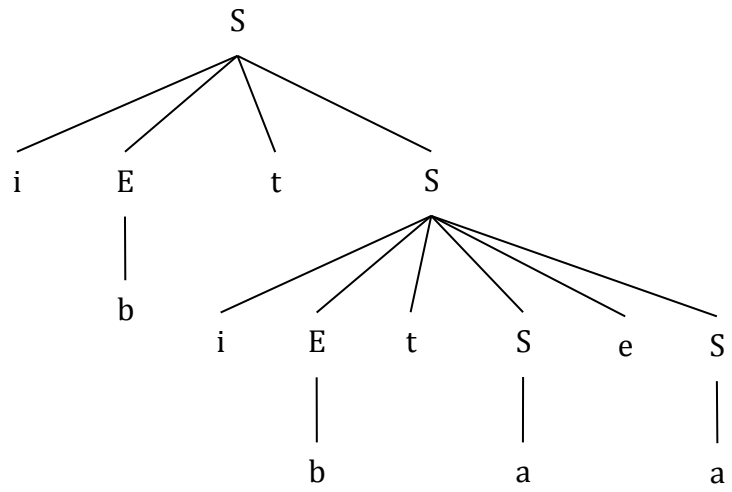
مثال: گرامر زیر برای توصیف جملات شرطی تو در تو به کار میرود. نشان دهید این گرامر برای جمله زیر دو بسط چپ ایجاد میکند.

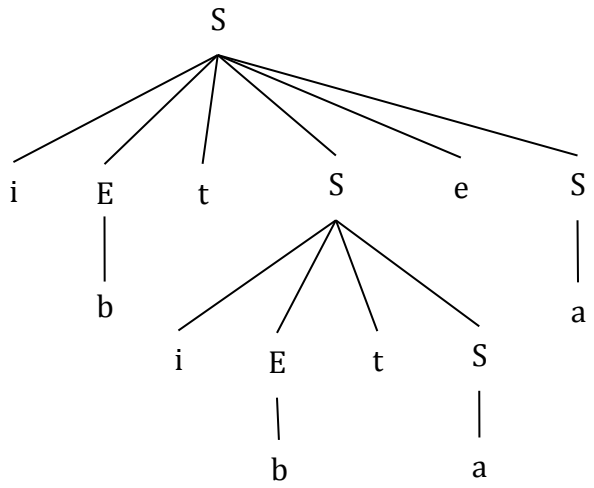
```
S → iEtS | iEtSeS | a     // i = if , t = then , e = else
E → b

w = ibtibtaea             // if b then if b then a else a
```

LMD1: $S \Rightarrow iEtS \Rightarrow ibtS \Rightarrow ibtiEtSeS \Rightarrow ibtibtSeS \Rightarrow ibtibtaeS \Rightarrow ibtibtaea$

LMD2: $S \Rightarrow iEtSeS \Rightarrow ibtSeS \Rightarrow ibtiEtSeS \Rightarrow ibtibtSeS \Rightarrow ibtibtaeS \Rightarrow ibtibtaea$





اشکال گرامرهای مبهم: میدانیم مشتق کردن w از S یک روش برای اثبات عضویت w در زبان گرامر است. بنابراین تا جایی که بحث عضویت مطرح باشد اشکالی به ساختارها و یا گرامرهای مبهم وارد نیست ولی از آنجاییکه هر مسیر عضویت درخت تجزیه متفاوتی را ایجاد میکند و هر درخت تجزیه هم کد میانی متفاوتی دارد ساختارهای مبهم برای طراحی تجزیه‌گرها مشکل ایجاد میکنند. به همبن دلبل باید ابهامی که در ساختارهای زبان وجود دارد به شکلی برطرف شود تا امکان طراحی تجزیه‌گرها ایجاد شود.

راهکارها: برای این منظور سه راهکار وجود دارد:

- با توجه به قواعد زبانهای برنامه‌نویسی (مثل تقدم عملگرها) برای ساختار داده شده یک گرامر غیر مبهم بنویسیم.
- زبان برنامه‌نویسی را به شکلی طراحی کنیم که ساختار مورد نظر در آن غیرمبهم باشد.
- گرامر را به صورت مبهم رها کنیم ولی با دستکاری در جدول تجزیه ابهام آن را برطرف کنیم.

رفع ابهام از گرامر عبارتها: برای ساختارهایی مثل عبارتها که از عملگرهای یگانی و دودویی تشکیل میشوند:

۱- تقدم عملگر: عملگرها را بر اساس سطح تقدم دسته‌بندی میکنیم و برای هر سطح یک غیرپایانی در نظر میگیریم.

۲- شرکت‌پذیری: اگر عملگر op در سطح N1 بوده و N2 نماد غیرپایانی یک سطح بالاتر باشد، با توجه به نوع عملگر و شرکت پذیری آن قواعد زیر را تولید میکنیم.

قاعده	شرکت پذیری	نوع عملگر
N1 → N1 op N2 | N2	از چپ	دودویی
N1 → N2 op N1 | N2	از راست	دودویی
N1 → op N1 | N2	-	یگانی چپ
N1 → N1 op | N2	-	یگانی راست

عملگر	غیرپایانی
+ , -	E (Expression)
* , /	T (Term)
^	F (Factor)
منهای یگانی	P (Primary)
پرانتز	M (Element)

E → E+T | E-T | T
T → T*F | T/F | F
F → P^F | P
P → –P | M
M → (E) | id

w = id + id * id

LMD: E ⇒ E+T ⇒ T+T ⇒ F+T ⇒ P+T ⇒ M+T ⇒ id+T ⇒ id+T*F⇒ id+F*F ⇒ id+P*F
⇒ id+M*F ⇒ id+id*F ⇒ id+id*P ⇒ id+id*M ⇒ id+id*id

گرامر غیر مبهم برای جملات شرطی: در زبانهای برنامه‌نویسی `else` به نزدیکترین `if` متعلق هست. با در نظر گـرفتن

این قاعده میتوان یک گرامر غیرمبهم برای جملات شرطی نوشت:

```
S → M | U            // M = Matched , U = Unmatched
M → iEtMeM | a
U → iEtS | iEtMeU
E → b

w = ibtibtaea        // if b then if b then a else a
```

LMD: $S \Rightarrow U \Rightarrow iEtS \Rightarrow ibtS \Rightarrow ibtiEtMeM \Rightarrow ibtibtMeM \Rightarrow ibtibtaeM \Rightarrow ibtibtaea$

اولین گام در طراحی تجزیه‌گر

چکیده: در این درس تجزیه‌گر را تعریف کرده و تلاش میکنیم روند بسط جمله را به شکل الگـوریتمی درآورده

و اولین گام را در طراحی تجزیه‌گر ایجاد کنیم.

تعریف تجزیه‌گر: برنامه‌ای است که جمله w را میگیرد و چنانچه w عضوی از زبان گرامر باشد یک ساختار نحوی (مثل درخت تجزیه) برای آن تشکیل داده وگرنه خطای نحوی برمیگرداند.

اولین گام: اگر بتوانیم یک الگوریتم قطعی برای بسط جمله پیدا کنیم یک تجزیه‌گر طراحی کرده‌ایم.

مثال۱: فرض کنید w با «-» شروع شود.

$E \rightarrow EAE \mid (E) \mid -E \mid id$

مثال۲: فرض کنید w با «a» شروع شود.

$S \rightarrow aa \mid bb \mid aSa \mid bSb$

روشهای طراحی تجزیه‌گر

- به کارگیری روش عقبگرد: در شرایط غیرقطعی تجزیه‌گر با یکی از قواعد تلاش میکند و چنانچه رشته ورودی به دست نیاید عقبگرد کرده و قاعده دیگر را انتخاب میکند.
- مشاهده نماد دوم: در شرایط غیرقطعی میتوان نماد دوم w را برای تصمیم‌گیری مشاهده کرد.
- تغییر گرامر: میتوان گرامر را تغییر داد تا به یک گرامر قطعی برای عمل تجزیه تبدیل شود. تجزیه‌گرهای پیشگوی بالا به پایین از روی گرامرهای قطعی شده ساخته میشوند.
- تغییر ابزار تشخیص: میتوان ابزار تشخیص را از گرامر به نمودار انتقالی (DFA) تغییر داد. تجزیه‌گرهای پایین به بالا بر همین مبنا ساخته میشوند.

شرط First-First: اگر First(β) اولین پایانه‌ای باشد که β با آن شروع میشود، آنگاه برای هر قاعده دو شاخه‌ای مثل زیر باید داشته باشیم:

$X \rightarrow \beta 1 \mid \beta 2$

$First(\beta 1) \cap First(\beta 2) = \{\}$

چنانچه یک لیست چند شاخه‌ای شرط F-F را نداشته باشد آن گرامر برای تجزیه‌گر بالا به پایین باید تغییر کند.

چگونه First(β) را پیدا کنیم؟

۱- برای رشته‌هایی مثل E– که با پایانه شروع میشوند یافتن First آسان است.

۲- برای رشته‌هایی مثل E+T که با غیرپایانه شروع میشوند تنها میتوان گفت رشته E+T ، First(E) را به ارث میبرد.

نتیجه۱: در مجموع هر رشته دلخواه چه با پایانه و چه غیر پایانه شروع شود، First اولین نماد خود را به ارث میبرد.

۳- اگر نماد E در E+T بتواند رشته نال را ایجاد کند آنگاه «+» هم به ارث برده میشود. و این موضوع برای نمادهای بعدی هم میتواند ادامه یابد. برای نمونه اگر در رشته XYZ غیرپایانه‌های X و Y هر دو نال باشند آنگاه علاوه بر First(X) و First(Y)، First(Z) هم به ارث برده میشود.

نتیجه۲: بنابراین قبل از یافتن First باید برای هر رشته α، نال بودن آن را بیابیم.

بررسی نال بودن رشته: رشته α نال است اگر و تنها اگر α طی صفر یا چند نتیجه‌گیری رشته تهی را ایجاد کند.

الگوریتم محاسبه Null(α) (نسخه بازگشتی): ورودی رشته α است و خروجی مقدار منطقی `True` یا `False` است.

۱- اگر $\alpha=\in$ آنگاه `Null(α) = True`.

۲- اگر α یک پایانه باشد آنگاه `Null(α) = False`.

۳- چنانچه داشته باشیم $\alpha\rightarrow\beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$ آنگاه:

$$Null(\alpha) = Null(\beta_1) \vee Null(\beta_2) \ldots \vee Null(\beta_n)$$

٤- چنانچه داشته باشیم $\alpha=x_1x_2\ldots x_n$ به طوری که x_i یک پایانه یا غیرپایانه در زبان باشد، آنگاه:

$$Null(\alpha) = Null(x_1) \wedge Null(x_2) \ldots \wedge Null(x_n)$$

بررسی الگوریتم: الگوریتم اخیر میتواند نال بودن هر رشته‌ای را بررسی کرده و اعلام کند. ولی به دلیل بازگشتی بودن، دنبال کردن آن برای ذهن کمی دشوار است و جنبه آموزشی پایینی دارد.

الگوریتم محاسبه Null(N) (نسخه غیر بازگشتی): ورودی مجموعه غیرپایانه‌هاست (N) و خروجی وضعیت نال بودن آنهاست.

۱- به صورت پیش فرض هر غیرپایانه در N را غیرنال کن.

۲- برای هر قاعده $X \rightarrow \beta$ که سمت چپ آن نال نشده است گام زیر را انجام بده:

- اگر همه نمادهای β نال باشند، X را نال کن.

۳- اگر در بند ۲ غیر پایانه جدیدی نال شود، بند ۲ را تکرار کن وگرنه از الگوریتم خارج شو.

S → VRTW | RVR | aVb
P → Y | cc
R → VXT | X | cYb
T → SW | b
V → VR | XR | dY
W → TS | aTb
X → dX | ∈
Y → SWP | Ybd

1	2	3	4	5
S → VRTW | RVR | aVb P → Y | cc R → VXT | X | cYb T → SW | b V → VR | XR | dY W → TS | aTb X → dX | ∈ Y → SWP | Ybd	S → VRTW | RVR | aVb P → Y | cc R → VT | ∈ | cYb T → SW | b V → VR | R | dY W → TS | aTb Y → SWP | Ybd	S → VTW | V | aVb P → Y | cc T → SW | b V → V | ∈ | dY W → TS | aTb Y → SWP | Ybd	S → TW | ∈ | ab P → Y | cc T → SW | b W → TS | aTb Y → SWP | Ybd	P → Y | cc T → W | b W → T | aTb Y → WP | Ybd
X	R	V	S	-

از آنجاییکه یک رشته پایانه‌دار نمیتواند نال باشد میتوان در الگوریتم محاسبه نال قواعدی که سمت راست آنها یک رشته پایانه‌دار باشد (مثل aVb) را از گرامر اولیه حذف کرده و آن گرامر را کوچک کرد. با مختصر کردن گرامر اولیه گرامرهای بازنویسی شده هم کوچکتر خواهند شد و به این ترتیب سرعت الگوریتم بالاتر میرود.

1	2	3	4	5
S → VRTW | RVR P → Y R → VXT | X T → SW V → VR | XR W → TS X → ∈ Y → SWP	S → VRTW | RVR P → Y R → VT | ∈ T → SW V → VR | R W → TS Y → SWP	S → VTW | V P → Y T → SW V → V | ∈ W → TS Y → SWP	S → TW | ∈ P → Y T → SW W → TS Y → SWP	P → Y T → W W → T Y → WP
X	R	V	S	-

~~S → VRTW | RVR | aVb~~
P → Y | cc
~~R → VXT | X | cYb~~
T → SW | b
~~V → VR | XR | dY~~
W → TS | aTb
~~X → dX | ∈~~ **X, R, V, S**
Y → SWP | Ybd

محاسبه مجموعه First

چکیده: شرط اول برای قطعی بودن گرامر این است که قواعد چند شاخه‌ای آن صریحا ابتدای مشترک نداشته باشند. بنابراین در این درس با محاسبه مجموعه `First` مشخص میکنیم قواعد چند شاخه‌ای گرامر ابتدای مشترک دارند یا نه.

مجموعه First: First(α) مجموعه پایانه‌هایی است که α با آن شروع میشود و تعریف آن به صورت زیر است:

$$First(\alpha) = \{a \mid \alpha \Rightarrow^{*} a\beta \ , \ \alpha,\beta \in (N \cup \sum)^{*} \ , \ a \in \sum\}$$

همچنین اگر α نال باشد، آنگاه ∈ هم به مجموعه First(α) اضافه میشود (با توجه به اینکه نماد Null همین هدف را دنبال میکند در ادامه از این بند چشم پوشی میکنیم. با این وجود اجتماع First و Null همان نتیجه را ایجاد میکند).

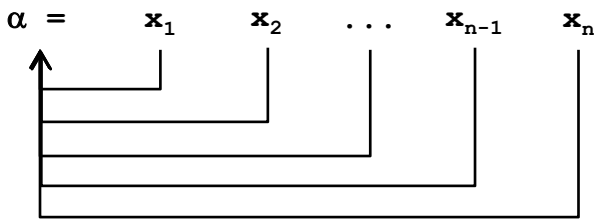
الگوریتم محاسبه First(α) (نسخه بازگشتی): ورودی رشته α است و خروجی مجموعه‌ای از پایانه‌ها است.

۱- اگر α یک پایانه باشد آنگاه First(α)={α}.

۲- چنانچه داشته باشیم $\alpha \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$ آنگاه برای هر β_i ، $First(\beta_i)$ را به First(α) اضافه کن.

۳- چنانچه داشته باشیم $\alpha = x_1 x_2 \ldots x_n$ آنگاه به ازای هر i از 1 تا n عملیات زیر را انجام بده:

- First(xi) را به First(α) اضافه کن.
- اگر xi نال باشد، بند ۳ را برای i بعدی تکرار کن وگرنه از بند ۳ خارج شو.



الگوریتم محاسبه First(N) (**نسخه غیر بازگشتی**): ورودی مجموعه غیرپایانه‌هاست (N) و خروجی مجموعه First آنهاست.

۱- یک گراف جهتدار تشکیل داده و برای هر غیرپایانه در N یک گره در گراف ایجاد کن.

۲- برای هر قاعده $Y \rightarrow x_1 x_2 \ldots x_n$ و برای هر i از 1 تا n عملیات زیر را انجام بده:

- اگر xi غیرپایانه باشد یک کمان از گره xi به گره Y ایجاد کن. و اگر xi پایانه باشد xi را به First(Y) اضافه کن (در حالت گرافیکی میتوانید xi را کنار گره Y بنویسید).
- اگر xi نال باشد، بند ۳ را برای i بعدی تکرار کن وگرنه از بند ۳ خارج شو.

۳- عمل انتقال پایانه‌ها را در گراف جهتدار انجام بده.

عمل انتقال: عمل انتقال در گراف میتواند به صورت زیر انجام بگیرد:

- برای هر گره غیر منزوی A که درجه ورودی آن در گراف صفر باشد عمل انتقال را برای خروجیهای A انجام بده. بعد از عمل انتقال یالهای خروجی A را حذف کرده و همین عمل را برای گراف باقیمانده انجام بده. عمل انتقال با منزوی شدن همه گره‌ها پایان میپذیرد (چنانچه گراف دارای چرخه باشد باید عمل انتقال از روی گراف مختصر آن انجام بگیرد).

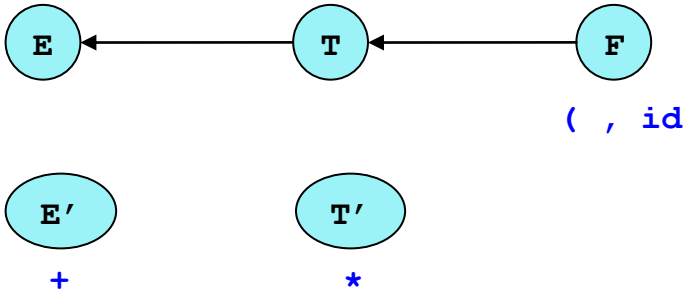
مثال۱: مجموعه First غیرپایانه‌های گرامر زیر را تعیین کرده و مشخص کنید شرط First-First برای قواعد آن برقرار هست یا نه.

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \in$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \in$
$F \rightarrow (E) \mid id$

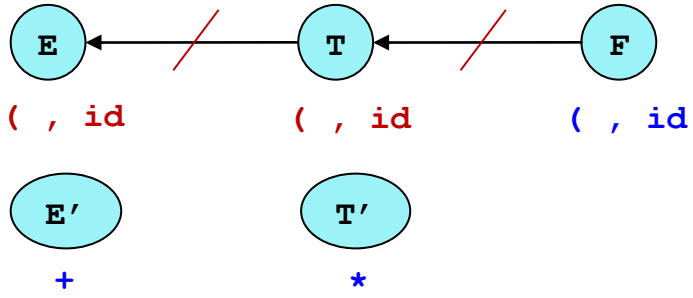
1	2
E → TE' E' → ∈ T → FT' T' → ∈	E → T T → F
E' , T'	-

الف- ابتدا وضعیت نال بودن غیرپایانه‌ها را مشخص میکنیم.

ب- سپس الگوریتم First را اجرا کرده و گراف جهتدار آن را به دست میآوریم.



ج- در پایان عمل انتقال را در گراف جهتدار انجام میدهیم:



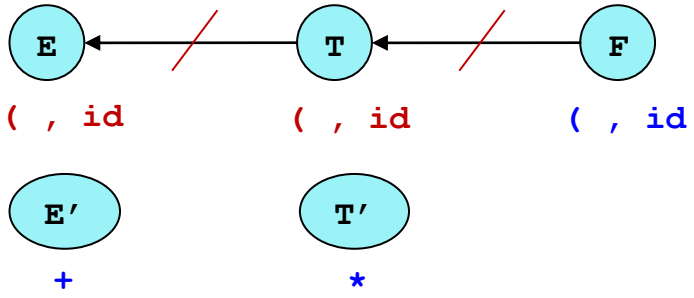
مثال ۱:

E → TE’
E’ → +TE’ | ∈
T → FT’
T’ → *FT’ | ∈
F → (E) | id

1	2
E → TE’ E’ → ∈ T → FT’ T’ → ∈	E → T T → F
E' , T'	-

نماد		First
E	-	(, id
E’	∈	+
T	-	(, id
T’	∈	*
F	-	(, id

میتوانید نتیجه الگوریتمهای Null و First را در جـــدول First غیرپایانه‌ها وارد کنید.



قاعده		First	F-F
E → TE’	-	(, id	
E’ → +TE’ E’ → ∈	- ∈	+	
T → FT’	-	(, id	
T’ → *FT’ T’ → ∈	- ∈	*	
F → (E) F → id	- -	(id	

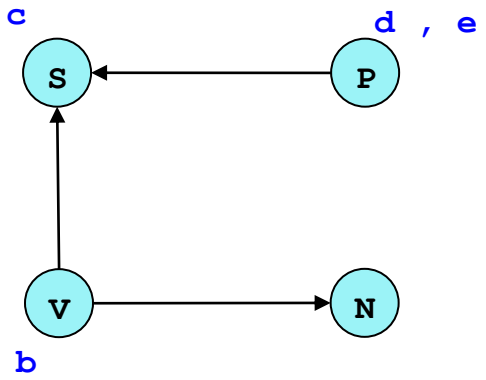
د- بـرای تعیـین شـرط First-First یـک جـدول دیگـر مشابه جدول First غیرپایانه‌ها ایجاد میکنیم. برای هر قاعده X→β یک ردیف ایجـاد کـرده و در مقابـل آن ، وضـعیت نـال بودن و مجموعه First رشته β را محاسبه میکنیم.

مثال۲:

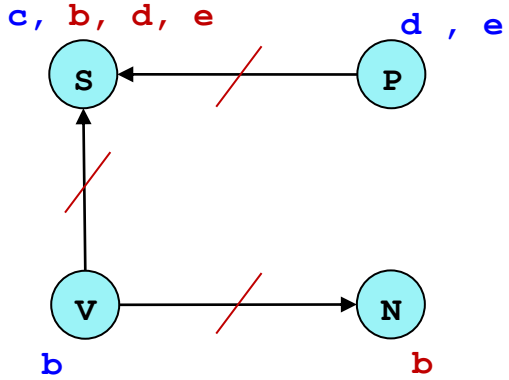
S → PaN | VP | c
P → dNP | e
N → Va | ∈
V → b

1	2
S → VP N → ∈	S → VP
N	-

نماد		First
S	-	b , c , d , e
P	-	d , e
N	∈	b
V	-	b



قاعده		First	F-F
S → PaN S → VP S → c	- - -	d , e b c	
P → dNP P → e	- -	d e	
N → Va N → ∈	- ∈	b	
V → b	-	b	

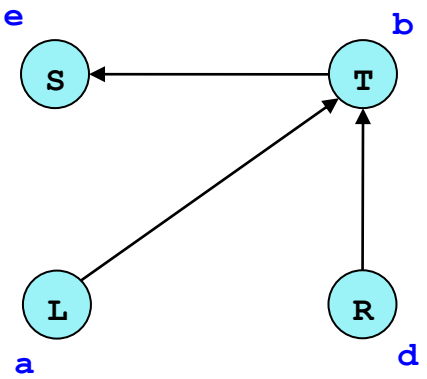


مثال۳:

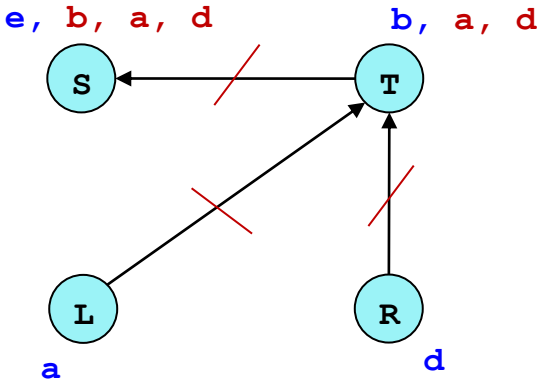
$S \rightarrow Te$
$L \rightarrow La \mid \in$
$T \rightarrow Tb \mid LRL$
$R \rightarrow dLT \mid \in$

1	2	3
$L \rightarrow \in$ $T \rightarrow LRL$ $R \rightarrow \in$	$T \rightarrow \in$	{Empty}
L , R	T	-

نماد		First
S	-	a , b , d , e
L	$\in$	a
T	$\in$	a , b , d
R	$\in$	d



قاعده		First	F-F
$S \rightarrow Te$	-	a , b , d , e	
$L \rightarrow La$ $L \rightarrow \in$	- $\in$	a	
$T \rightarrow Tb$ $T \rightarrow LRL$	- $\in$	a , b , d a , d	a , d
$R \rightarrow dLT$ $R \rightarrow \in$	- $\in$	d	

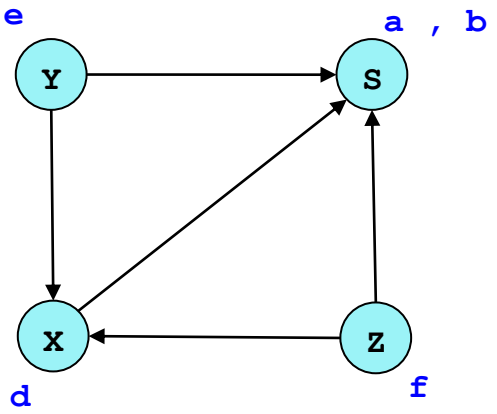


مثال٤:

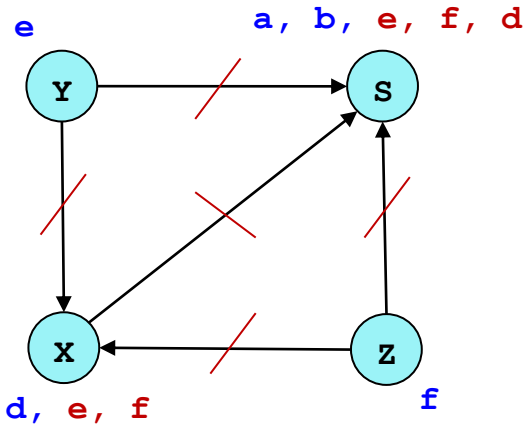
$S \rightarrow XZY \mid ZbY \mid Ya$
$X \rightarrow da \mid YZ$
$Y \rightarrow e \mid \in$
$Z \rightarrow f \mid \in$

1	2	3
S → XZY X → YZ Y → ∈ Z → ∈	S → X X → ∈	S → ∈
Y , Z	X	S

نماد		First
S	∈	a , b , d , e , f
X	∈	d , e , f
Y	∈	e
Z	∈	f

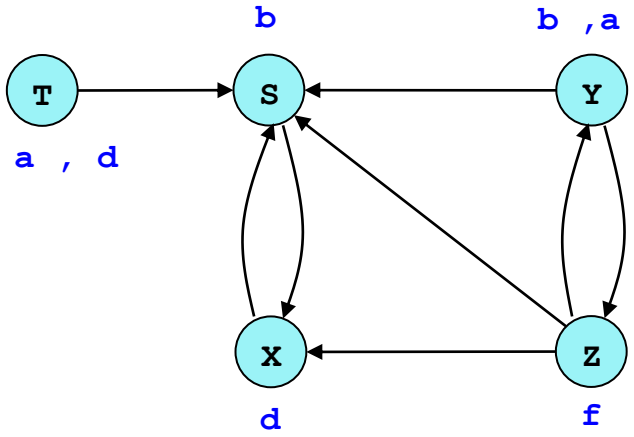


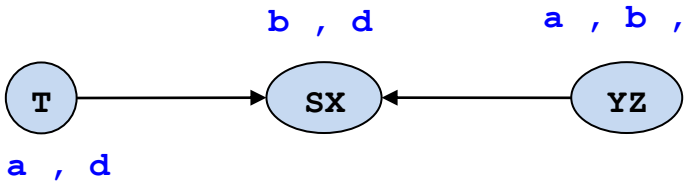
قاعده		First	F-F
S → XZY S → ZbY S → Ya	∈ - -	d , e , f f , b e , a	e , f
X → da X → YZ	- ∈	d e , f	
Y → e Y → ∈	- ∈	e	
Z → f Z → ∈	- ∈	f	



مثال۵:

$S \rightarrow XYa \mid Sb \mid ZT$
$X \rightarrow SZ \mid de$
$Y \rightarrow Zb \mid a$
$Z \rightarrow Yd \mid f \mid \in$
$T \rightarrow abX \mid d \mid \in$

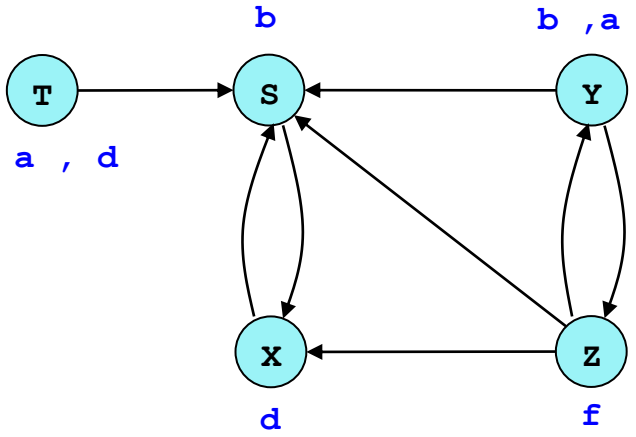


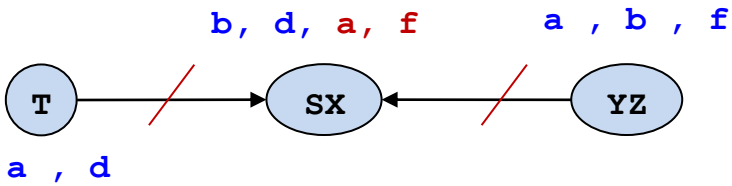


1	2	3	4
$S \rightarrow ZT$ $X \rightarrow SZ$ $Z \rightarrow \in$ $T \rightarrow \in$	$S \rightarrow \in$ $X \rightarrow S$	$X \rightarrow \in$	{Empty}
Z , T	S	X	-

مثال۵:

$S \rightarrow XYa \mid Sb \mid ZT$
$X \rightarrow SZ \mid de$
$Y \rightarrow Zb \mid a$
$Z \rightarrow Yd \mid f \mid \in$
$T \rightarrow abX \mid d \mid \in$





نماد		First
S	∈	a , b , d , f
X	∈	a , b , d , f
Y	-	a , b , f
Z	∈	a , b , f
T	∈	a , d

قاعده		First	F-F
S → XYa S → Sb S → ZT	- - ∈	a , b , d , f a , b , d , f a , b , f , d	a , b , d , f
X → SZ X → de	∈ -	a , b , d , f d	d
Y → Zb Y → a	- -	a , b , f a	a
Z → Yd Z → f Z → ∈	- - ∈	a , b , f f	f
T → abX T → d T → ∈	- - ∈	a d	

محاسبه مجموعه Follow

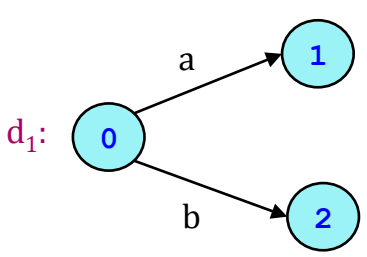
چکیده: با محاسبه مجموعه `First` و بررسی شرط `First-First` میتوان مشخص کرد قواعد چند شاخه‌ای گرامر ابتدای مشترک دارند یا نه. ولی برای قطعی بودن گرامر شرط `First-First` کافی نیست. در این درس با معرفی مجموعه `Follow` همه شرایط لازم برای قطعی بودن گرامر را مشخص میکنیم.

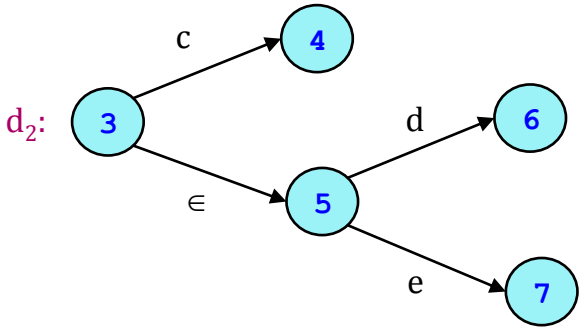
آیا First-First کافی است؟ فرض کنید w با «a» شروع شود.

$S \rightarrow Xab$
$X \rightarrow a \mid \in$

بدیهی است که شرط First-First برای این گرامر برقرار است. ولی نماد a هم میتواند با X→a دریافت شود و هم میتواند با نماد a در رشته Xab و بعد از به کارگیری قاعده X→∈ دریافت شود. این عدم قطعیت با جایگزینی قواعد X در S آشکارتر میشود:

$S \rightarrow aab \mid ab$





شرط انتخاب قاعده : یک قاعده مثل X→α در صورتی انتخاب میشود که نماد بعدی عضو ابتدای صریح یا پنهان آن باشد.

مجموعه Follow: برای هر غیرپایانه X ، Follow(X) مجموعه پایانه‌هایی است که بعد از X قرار میگیرند و تعریف رسمی آن به صورت زیر است:

$$Follow(X)= \{b \mid S \overset{*}{\Rightarrow} \alpha X b \beta \ , \ \alpha,\beta \in (N \cup \Sigma)^* \ , \ b \in \Sigma \ , \ S,X \in N \ , \ S \text{ is Start}\}$$

شرایط قطعی بودن گرامر: برای هر لیست دو شاخه‌ای مثل $X \rightarrow \beta_1 \mid \beta_2$ باید داشته باشیم:

- شرط First-First:

$$First(\beta_1) \cap First(\beta_2) = \phi$$

- شرط First-Follow:

$$\text{if } Null(\beta_1) \text{ then } Follow(X) \cap First(\beta_2) = \phi$$

- شرط Null-Null:

$$Null(\beta_1) \wedge Null(\beta_2) = False$$

نکته: اگر $\in$ را جزئی از First بگیریم Null-Null حالت خاصی از شرط First-First خواهد بود.

گرامرهای LL(1): یک گرامر LL(1) است اگر هر سه شرط قطعی بودن گرامر برای آن برقرار باشد.

الگوریتم محاسبه Follow(X): ورودی غیرپایانه X است و خروجی مجموعه‌ای از پایانه‌هاست وشامل ∈ نیست.

۱- اگر X نماد شروع گرامر باشد، $ را به مجموعه Follow (X) اضافه کن.

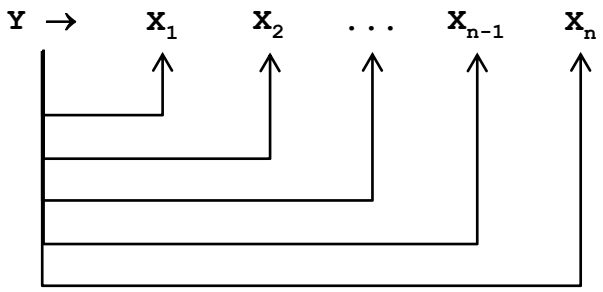
۲- برای هر قاعده با فرمت $Y \rightarrow \alpha X \beta$ ، First (β) را به Follow (X) اضافه کن.

۳- برای هر قاعده با فرمت $Y \rightarrow \alpha X$ یا $Y \rightarrow \alpha X \beta$ و $\beta \overset{*}{\Rightarrow} \in$ ، Follow (Y) را به Follow (X) اضافه کن.

نکته: بند سوم الگوریتم را به شکل زیر هم میتوان بیان کرد:

۳- اگر قاعده‌ای به شکل $Y \rightarrow \alpha X_1 X_2 \ldots X_{n-1} X_n$ وجود داشته باشد، به طوری که α تهی بوده و یا رشته‌ای از پایانه‌ها باشد و Xi یک غیرپایانه در گرامر باشد آنگاه:

- Follow (Y) را به $\text{Follow}(X_n)$ اضافه کن.
- اگر x_n نال باشد Follow (Y) را به $\text{Follow}(X_{n-1})$ اضافه کن.
- اگر x_{n-1} نال باشد Follow (Y) را به $\text{Follow}(X_{n-2})$ اضافه کن.
- ...
- اگر x_2 نال باشد Follow (Y) را به $\text{Follow}(X_1)$ اضافه کن.



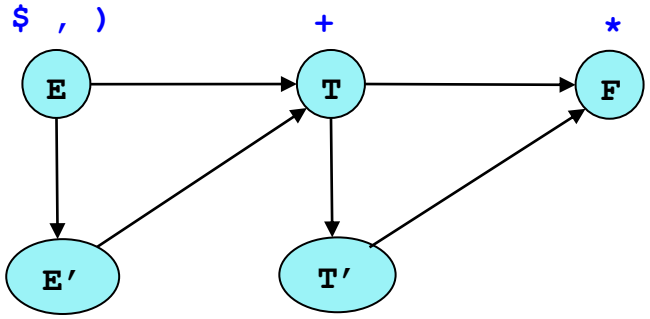
مثال ۱:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \in$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \in$
$F \rightarrow (E) \mid id$

T E' : **+**

F T' : ***

E) : **)**



نماد		First
E	-	(, id
E'	∈	+
T	-	(, id
T'	∈	*
F	-	(, id

قاعده		First
$E \rightarrow TE'$	-	(, id
$E' \rightarrow +TE'$ $E' \rightarrow \in$	- ∈	+
$T \rightarrow FT'$	-	(, id
$T' \rightarrow *FT'$ $T' \rightarrow \in$	- ∈	*
$F \rightarrow (E)$ $F \rightarrow id$	- -	(id

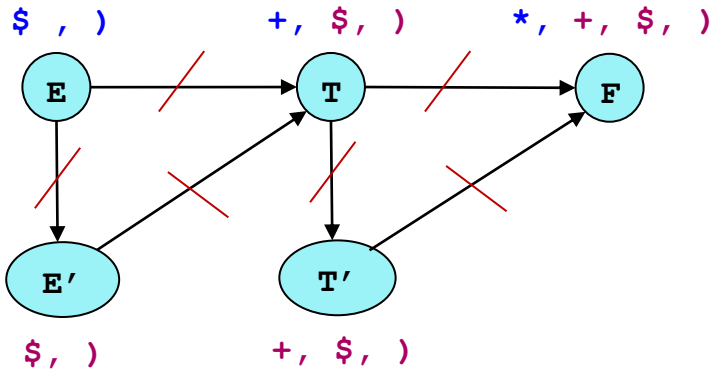
مثال ۱:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \in$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \in$
$F \rightarrow (E) \mid id$

T E' : **+**

F T' : ***

E) : **)**



نماد		First	Follow
E	-	(, id	$,)
E'	∈	+	$,)
T	-	(, id	+ , $,)
T'	∈	*	+ , $,)
F	-	(, id	* , + , $,)

قاعده		First	Follow	F-L
$E \rightarrow TE'$	-	(, id		
$E' \rightarrow +TE'$ $E' \rightarrow \in$	- ∈	+	$,)	
$T \rightarrow FT'$	-	(, id		
$T' \rightarrow *FT'$ $T' \rightarrow \in$	- ∈	*	+ , $,)	
$F \rightarrow (E)$ $F \rightarrow id$	- -	(id		

مثال۲:

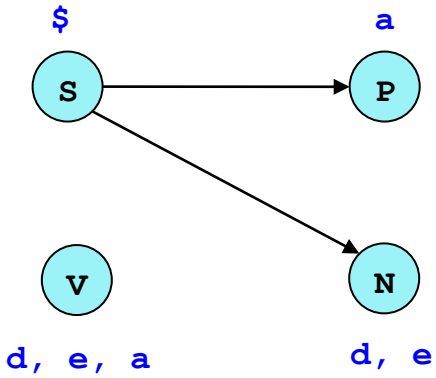
$S \rightarrow PaN \mid VP \mid c$
$P \rightarrow dNP \mid e$
$N \rightarrow Va \mid \in$
$V \rightarrow b$

P aN
a

N P
d, e

V P
d, e

V a
a



نماد		First
S	-	b , c , d , e
P	-	d , e
N	$\in$	b
V	-	b

قاعده		First
$S \rightarrow PaN$	-	d , e
$S \rightarrow VP$	-	b
$S \rightarrow c$	-	c
$P \rightarrow dNP$	-	d
$P \rightarrow e$	-	e
$N \rightarrow Va$	-	b
$N \rightarrow \in$	$\in$	
$V \rightarrow b$	-	b

مثال۲:

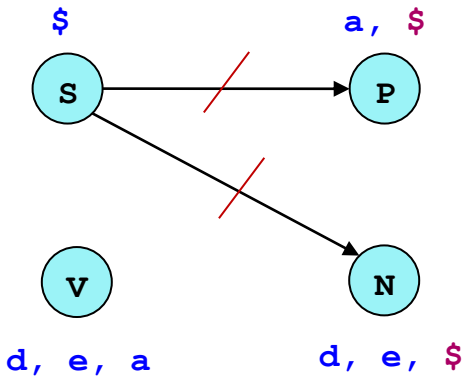
$S \rightarrow PaN \mid VP \mid c$
$P \rightarrow dNP \mid e$
$N \rightarrow Va \mid \in$
$V \rightarrow b$

P aN
a

N P
d, e

V P
d, e

V a
a



نماد		First	Follow
S	-	b , c , d , e	$
P	-	d , e	$, a
N	∈	b	$, d , e
V	-	b	d , e , a

قاعده		First	Follow	F-L
S → PaN S → VP S → c	- - -	d , e b c		
P → dNP P → e	- -	d e		
N → Va N → ∈	- ∈	b	$, d , e	
V → b	-	b		

مثال۳:

$S \rightarrow Te$
$L \rightarrow La \mid \in$
$T \rightarrow Tb \mid LRL$
$R \rightarrow dLT \mid \in$

T e
e

T b
b

R L
a

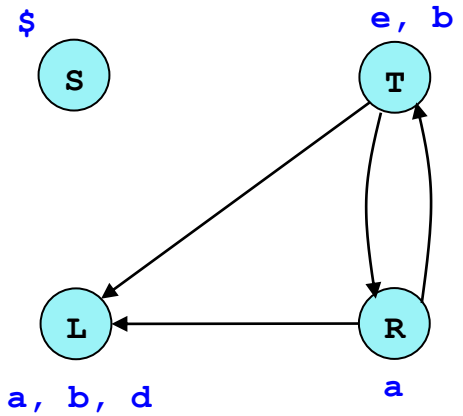
L a
a

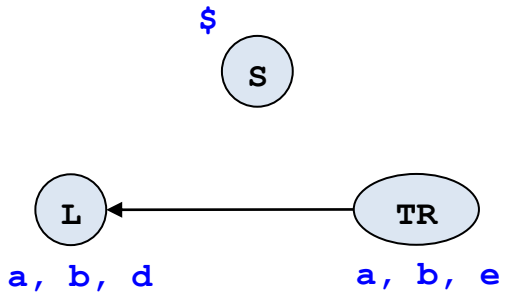
L RL
d, a

L T
a, b, d

نماد		First
S	-	a , b , d , e
L	∈	a
T	∈	a , b , d
R	∈	d

قاعده		First
S → Te	-	a, b, d, e
L → La L → ∈	- ∈	a
T → Tb T → LRL	- ∈	a, b, d a, d
R → dLT R → ∈	- ∈	d





مثال۳:

$S \rightarrow Te$
$L \rightarrow La \mid \in$
$T \rightarrow Tb \mid LRL$
$R \rightarrow dLT \mid \in$

T e
e

T b
b

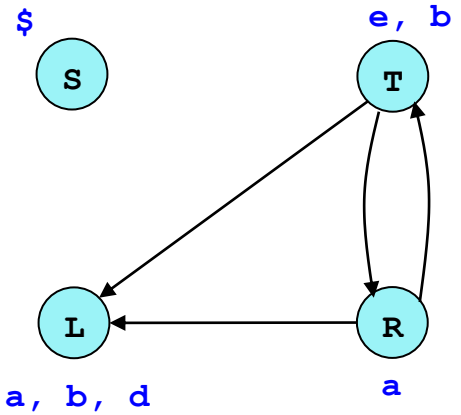
R L
a

L a
a

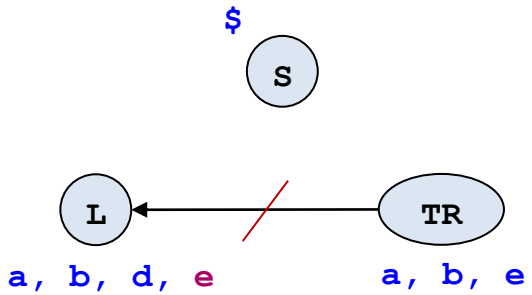
L RL
d, a

L T
a, b, d

نماد		First	Follow
S	-	a , b , d , e	$
L	∈	a	a , b , d , e
T	∈	a , b , d	a , b , e
R	∈	d	a , b , e



قاعده		First	Follow	F-L
S → Te	-	a, b, d, e		
L → La L → ∈	- ∈	a	a, b, d, e	a
T → Tb T → LRL	- ∈	a, b, d a, d	a , b , e	a, b
R → dLT R → ∈	- ∈	d	a , b , e	



مثال٤:

$S \rightarrow XZY \mid ZbY \mid Ya$
$X \rightarrow da \mid YZ$
$Y \rightarrow e \mid \in$
$Z \rightarrow f \mid \in$

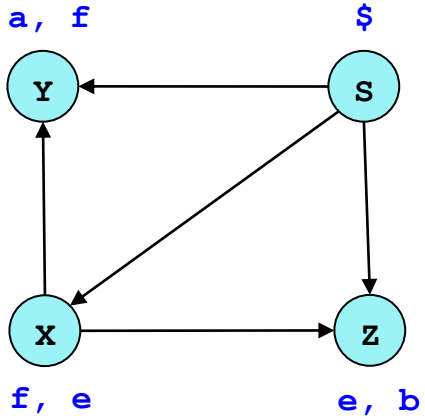
X ZY
f, e

Z bY
b

Y Z
f

Z Y
e

Y a
a



نماد		First
S	∈	a , b , d , e , f
X	∈	d , e , f
Y	∈	e
Z	∈	f

قاعده		First
S → XZY S → ZbY S → Ya	∈ - -	d , e , f f , b e , a
X → da X → YZ	- ∈	d e , f
Y → e Y → ∈	- ∈	e
Z → f Z → ∈	- ∈	f

مثال۴:

$S \rightarrow XZY \mid ZbY \mid Ya$
$X \rightarrow da \mid YZ$
$Y \rightarrow e \mid \in$
$Z \rightarrow f \mid \in$

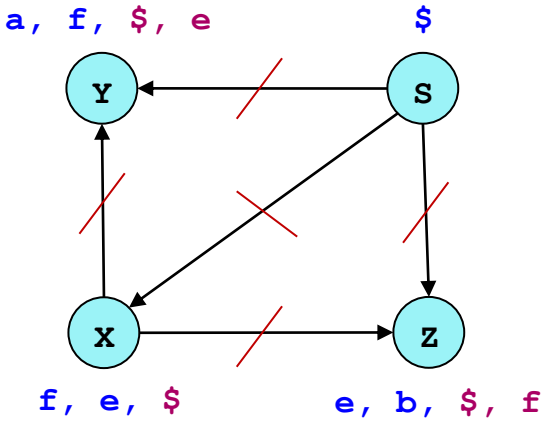
X ZY
f, e

Z bY
b

Y Z
f

Z Y
e

Y a
a



نماد		First	Follow
S	∈	a , b , d , e , f	$
X	∈	d , e , f	f , e , $
Y	∈	e	a , f , e , $
Z	∈	f	e , b , f , $

قاعده		First	Follow	F-L
S → XZY S → ZbY S → Ya	∈ - -	d , e , f f , b e , a	$	
X → da X → YZ	- ∈	d e , f	f , e , $	
Y → e Y → ∈	- ∈	e	a , f , e , $	e
Z → f Z → ∈	- ∈	f	e , b , f , $	f

مثال۵:

S → XYa | Sb | ZT
X → SZ | de
Y → Zb | a
Z → Yd | f | ∈
T → abX | d | ∈

X Ya
a, b, f

S b
b

S Z
a, b, f

Y d
d

Y a
a

Z T
a, d

Z b
b

نماد		First
S	∈	a , b , d , f
X	∈	a , b , d , f
Y	-	a , b , f
Z	∈	a , b , f
T	∈	a , d

قاعده		First
S → XYa	-	a , b , d , f
S → Sb	-	a , b , d , f
S → ZT	∈	a , b , f , d
X → SZ	∈	a , b , d, f
X → de	-	d
Y → Zb	-	a , b , f
Y → a	-	a
Z → Yd	-	a , b , f
Z → f	-	f
Z → ∈	∈	
T → abX	-	a
T → d	-	d
T → ∈	∈	

a, d
Y

$, a, b, f
T
S
X
Z
a, b, f
a, d, b

$, a, b, f
a, d, b
SXT
Z
Y
a, d

مثال۵:

$S \rightarrow XYa \mid Sb \mid ZT$
$X \rightarrow SZ \mid de$
$Y \rightarrow Zb \mid a$
$Z \rightarrow Yd \mid f \mid \in$
$T \rightarrow abX \mid d \mid \in$

X Ya
a, b, f

S b
b

S Z
a, b, f

Y d
d

Y a
a

Z T
a, d

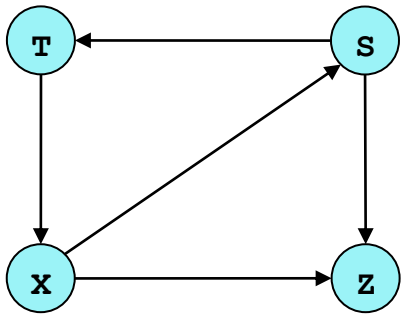
Z b
b

نماد		First	Follow
S	∈	a , b , d , f	$, a , b , f
X	∈	a , b , d , f	$, a , b , f
Y	-	a , b , f	a , d
Z	∈	a , b , f	$, a , d , b , f
T	∈	a , d	$, a , b , f

a, d

Y

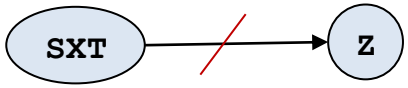
$, a, b, f



a, b, f a, d, b

قاعده		First	Follow	F-L
S → XYa	-	a , b , d , f		a , b , f
S → Sb	-	a , b , d , f		
S → ZT	∈	a , b , f , d	$, a , b , f	
X → SZ	∈	a , b , d, f	$, a , b , f	
X → de	-	d		
Y → Zb	-	a , b , f		
Y → a	-	a		
Z → Yd	-	a , b , f		a , b , f
Z → f	-	f		
Z → ∈	∈		$, a , d , b , f	
T → abX	-	a		a
T → d	-	d		
T → ∈	∈		$, a , b , f	

$, a, b, f a, d, b, $, f



Y

a, d

فاکتورگیری از چپ: چنانچه برای غیرپایانه X به دو قاعده برخورد کنیم که هر دو با α شروع شده باشند، شرط First- نقض شده و یک وضعیت غیرقطعی برای گرامر ایجاد میشود:

$X \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

$X \rightarrow \alpha X'$
$X' \rightarrow \beta_1 \mid \beta_2$

و در حالت کلی:

$X \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_m$

$X \rightarrow \alpha X' \mid \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_m$
$X' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$

مثال ۱ و ۲:

$S \rightarrow aXbc \mid aXbd \mid a$
$X \rightarrow daX \mid d \mid \in$

√ $S \rightarrow aS_1$
$S_1 \rightarrow Xbc \mid Xbd \mid \in$

√ $S_1 \rightarrow XbS_2 \mid \in$
√ $S_2 \rightarrow c \mid d$

√ $X \rightarrow dX' \mid \in$
√ $X' \rightarrow aX \mid \in$

$S \rightarrow iEtS \mid iEtSeS \mid a$
√ $E \rightarrow b$

√ $S \rightarrow iEtSS' \mid a$
√ $S' \rightarrow eS \mid \in$

حذف چپگردی: گرامر G را چپگرد میگوییم اگر برای غیرپایانه‌ای مثل X و دنباله α داشته باشیم:

$$X \overset{*}{\Rightarrow} X\alpha$$

چپگردی فوری: به صورت زیر تعریف شده و با $\beta\alpha^*$ معادل است:

$$X \rightarrow X\alpha \mid \beta$$

$$X \rightarrow \beta X'$$
$$X' \rightarrow \alpha X' \mid \in$$

و در حالت کلی:

$$X \rightarrow X\alpha_1 \mid X\alpha_2 \mid \dots \mid X\alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

$$X \rightarrow \beta_1 X' \mid \beta_2 X' \mid \dots \beta_n X'$$
$$X' \rightarrow \alpha_1 X' \mid \alpha_2 X' \mid \dots \mid \alpha_m X' \mid \in$$

مثال ۱، ۲ و ۳:

$$X \rightarrow Xc \mid Xad \mid bd \mid e$$

$$X \rightarrow bdX' \mid eX'$$
$$X' \rightarrow cX' \mid adX' \mid \in$$

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

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid -TE' \mid \in$$

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

$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid /FT' \mid \in$$

چپگردی پنهان: گرامرها به جز چپگردی فوری دارای چپگردی پنهان هم هستند:

$S \rightarrow Xa \mid b$
$X \rightarrow Xc \mid Sd \mid e$

$S \Rightarrow Xa \Rightarrow Sda$

الگوریتم حذف چپگردی پنهان: این الگوریتم از گرامرهایی که بسطهایی به فرم $X \overset{*}{\Rightarrow} X$ ندارند (دارای چرخه نیستند) و در آنها قواعد $X \rightarrow \in$ وجود ندارد چپگردی را حذف میکند. گرامر به دست آمده ممکن است قاعده $X \rightarrow \in$ داشته باشد.

۱- یک ترتیب برای غیرپایانه‌های گرامر به صورت $X_1, X_2, \ldots, X_n$ در نظر بگیر.

۲- برای هر مقدار i از 1 تا n:

الف- چنانچه داشته باشیم:

$X_j \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_k$

...

$X_i \rightarrow X_j\alpha \quad (j<i)$

قاعده‌های زیر را جایگزین کن:

$X_i \rightarrow \beta_1\alpha \mid \beta_2\alpha \mid \ldots \mid \beta_k\alpha$

و برای هر قاعده $X_i \rightarrow \beta_j\alpha$ چنانچه شرط الف برای آن وجود داشته باشد باید جایگزینی تکرار شود.

ب- چنانچه غیرپایانه X_i چپگردی فوری داشته باشد همه چپگردیها را از قاعده‌های X_i حذف کن.

مثال ۱ و ۲:

√ S → Xa | b
X → Xc | Y
Y → Sd | e

√ X → YX’
√ X’ → cX’ | ∈

Y → Xad | bd | e
Y → YX’ad | bd | e
√ Y → bdY’ | eY’
√ Y’ → X’adY’ | ∈

√ S → Xa | ab
X → Sd | Yb | c
Y → aS | Xbc | d

X → Xad | abd | Yb | c
√ X → abdX’ | YbX’ | cX’
√ X’ → adX’ | ∈

Y → aS | abdX’bc | YbX’bc | cX’bc | d
√ Y → aSY’ | abdX’bcY’ | cX’bcY’ | dY’
√ Y’ → bX’bcY’ | ∈

روش تجزیه بازگشتی-کاهشی

چکیده: تجزیه‌گرهای بالا به پایین به یکی از دو روش دستی و مبتنی بر جدول طراحی میشوند. روش دستی که با نام بازگشتی-کاهشی شناخته میشود مستقیما از روی جدول تصمیم گیری (First-Follow قواعد) و برای گرامرهای نوع LL(1) ساخته میشود.

روش بازگشتی-کاهشی: برای گرامرهای نوع LL(1) به کار میرود. در این روش برای هر یک از غیرپایانه‌ها یک زیربرنامه طبق الگوریتم تجزیه نوشته میشود. زیربرنامه‌ها به صورت بازگشتی همدیگر را فراخوانی میکنند تا سرانجام جمله (برنامه) مورد نظر تایید و یا رد شود.

$$\left.\begin{array}{l} N_1 + \text{الگوریتم تجزیه} \rightarrow proc_1 \\ N_2 + \text{الگوریتم تجزیه} \rightarrow proc_2 \\ \dots \\ N_m + \text{الگوریتم تجزیه} \rightarrow proc_n \end{array}\right\} = \text{تجزیه‌گر}$$

الگوریتم تجزیه بازگشتی-کاهشی: با فرض اینکه توکن بعدی در پیشنگر (Lookahead و به صورت مختصر L) نگهداری شده و GetNextToken تابعی باشد که با فراخوانی اسکنر توکن بعدی را دریافت کند:

۱- یک زیربرنامه به نام Match(T) به صورت زیر بنویس (این زیربرنامه مشخص میکند آیا توکن موجود در پیشنگر برابر T هست یا نه؟ اگر هست با دریافت توکن بعدی در پیشنگر آن را تایید کرده وگرنه پیغام خطا اعلام میکند):

```
procedure Match(T: TToken);
begin
  if T= Lookahead then
    Lookahead= GetNextToken;
  else
    Syntax-Error;
end;
```

۲- برای هر لیست چندشاخه‌ای مثل $X\rightarrow\beta_1 | \beta_2 | \ldots | \beta_n$ زیربرنامه‌ای به صورت زیر بنویس:

```
procedure MatchX;
begin
  if L in Selectors(X→β1) then
    Process β1
  else if L in Selectors(X→β2) then
    Process β2
  ...
  else if L in Selectors(X→βn) then
    Process βn
  else
    Syntax-Error;
end;
```

```
procedure MatchX;
begin
  case L of
    Selectors(X→β1): Process β1
    Selectors(X→β2): Process β2
    ...
    Selectors(X→βn): Process βn
    else
      Syntax-Error;
  end;
end;
```

نکته: برای قواعد تک شاخه‌ای مثل X→β میتوان هر یک از دو روش زیر را به کار گرفت:

```
procedure MatchX;
begin
  if L in Selectors(X→β) then
    Process β;
  else
    Syntax-Error;
end;
```

```
procedure MatchX;
begin
  Process β;
end;
```

الگوریتم پردازش رشته β: با فرض $\beta=x_1x_2...x_n$ ، به ازای هر i از 1 تا n :

- اگر xi یک غیرپایانه باشد زیربرنامه MatchXi وگرنه زیربرنامه Match(xi) را فراخوانی کن.

مثال: برای نمونه پردازش رشته bYS میتواند به صورت زیر نوشته شود:

```
Match('b');
MatchY;
MatchS;
```

قاعده		First	Follow
E → TE’	-	(, id	
E’ → +TE’	-	+	
E’ → ∈	∈		$,)
T → FT’	-	(, id	
T’ → *FT’	-	*	
T’ → ∈	∈		+ , $,)
F → (E)	-	(	
F → id	-	id	

```
procedure MatchE;
begin
  MatchT;
  MatchEp;
end;

procedure MatchEP;
begin
  case L of
    '+':
      begin
        Match('+');
        MatchT;
        MatchEp;
      end;
    '$', ')':
      {null}
    else
      SyntaxError('+ , $ , ) expected');
  end;
end;
```

```
procedure MatchT;
begin
  MatchF;
  MatchTp;
end;

procedure MatchTp;
begin
  case L of
    '*':
      begin
        Match('*');
        MatchF;
        MatchTp;
      end;
    '+', '$', ')':
      {null}
    else
      SyntaxError('*, + , $ , ) expected');
  end;
end;

procedure MatchF;
begin
  case L of
    '(':
      begin
        Match('(');
        MatchE;
        Match(')');
      end;
    'id': Match('id');
    else
      SyntaxError('(, id expected');
  end;
end;
```

ایجاد جدول LL(1)

چکیده: تجزیه‌گرهای بالا به پایین به یکی از دو روش دستی و مبتنی بر جدول طراحی میشوند. در روش مبتنی بر جدول عمل تجزیه از روی یک جدول به نام LL(1) انجام میگیرد. در این درس ابتدا ساختار جدول LL(1) را شرح داده و سپس الگوریتم ساخت جدول LL(1) را از روی جدول First-Follow قواعد بیان میکنیم.

شرط انتخاب قاعده : یک قاعده مثل $X \rightarrow \beta$ در صورتی انتخاب میشود که نماد بعدی عضو ابتدای صریح یا پنهان آن باشد.

گزینشگرهای قاعده : اجتماع ابتدای صریح و پنهان یک قاعده را گزینشگرهای آن قاعده مینامیم. یعنی همه نمادهایی که در مقابل آن قاعده در جدول First-Follow به دست آمدند.

انتخاب قاعده در لیست چند شاخه‌ای: اگر در یک لیست چند شاخه‌ای نماد بعدی عضو گزینشگرهای یکی از قواعد باشد آن قاعده انتخاب میشود وگرنه خطای نحوی هست. برای گرامرهای LL(1) نماد بعدی نمیتواند عضو گزینشگرهای دو یا چند قاعده باشد.

$X \rightarrow \beta 1 \mid \beta 2 \mid \ldots \mid \beta n$

مثال:

قاعده		First	Follow	Conflict
E → TE’	-	(, id		
E’ → +TE’ E’ → ∈	- ∈	+	$,)	
T → FT’	-	(, id		
T’ → *FT’ T’ → ∈	- ∈	*	+ , $,)	
F → (E) F → id	- -	(id		


تجزیه‌گرها
بالا به پایین
عقبگرد: گرامر (1) LL نباشد.
پیشگو: گرامر (1) LL باشد
بازگشتی-کاهشی: از تعدادی زیربرنامه بازگشتی تشکیل میشوند.
مبتنی بر جدول: عمل تجزیه از روی یک جدول تجزیه انجام میشود.
پایین به بالا
تقدم-عملگر: اغلب برای تجزیه عبارتها به کار میروند.
روشهای LR: گرامرهای بیشتری را نسبت به روشهای پیشگوی بالا به پایین پوشش میدهند.

تجزیه‌گرهای پیشگوی مبتنی بر جدول: پیش از عمل تجزیه یک جدول تجزیه برای گرامر ایجاد میشود. جدول تجزیه LL(1) نام دارد و توسط مجموعه‌های گزینشگر ایجاد میشود.

جدول LL(1): با فرض اینکه X1,X2,…,Xn غیرپایانه‌ها و a1,a2,…,am پایانه‌های گرامر باشند، جدول LL(1) ساختاری به صورت زیر دارد:

LL(1)	a_1	a_2	…	a_m
X_1				
X_2				
…				
X_n				

خانه‌های جدول

- خالی: به معنی خطای نحوی است.
- شامل Xi→β : در روند تجزیه غیرپایانه Xi با رشته β جایگزین میشود.
- تداخل داشته باشد: در این صورت گرامر LL(1) نیست.

الگوریتم ساخت جدول LL(1): برای هر پایانه a که عضو گزینشگرهای قاعده X→β هست ، قاعده X→β را در خانه M[X, a] درج کن. به بیان دیگر قاعده X→β در ردیف X و زیر پایانه‌هایی درج میشود که عضو گزینشگر آن قاعده هستند.

مثال۱: جدول تجزیه گرامر زیر را تشکیل دهید.

قاعده		First	Follow	Conflict
E → TE’	-	(, id		
E’ → +TE’ E’ → ∈	- ∈	+ 	 $,)	
T → FT’	-	(, id		
T’ → *FT’ T’ → ∈	- ∈	* 	 + , $,)	
F → (E) F → id	- -	(id		

LL(1)	id	+	*	(	)	$
E	E → TE’			E → TE’		
E’		E’ → +TE’			E’ → ∈	E’ → ∈
T	T → FT’			T → FT’		
T’		T’ → ∈	T’ → *FT’		T’ → ∈	T’ → ∈
F	F → id			F → (E)		

مثال۳: جدول تجزیه گرامر زیر را تشکیل دهید.

قاعده		First	Follow	Conflict
S → PaN S → VP S → c	- - -	d , e b c		
P → dNP P → e	- -	d e		
N → Va N → ∈	- ∈	b	$, d , e	
V → b	-	b		

LL(1)	a	b	c	d	e	$
S		S → VP	S → c	S → PaN	S → PaN	
P				P → dNP	P → e	
N		N → Va		N → ∈	N → ∈	N → ∈
V		V → b				

مثال۴: جدول تجزیه گرامر زیر را تشکیل دهید.

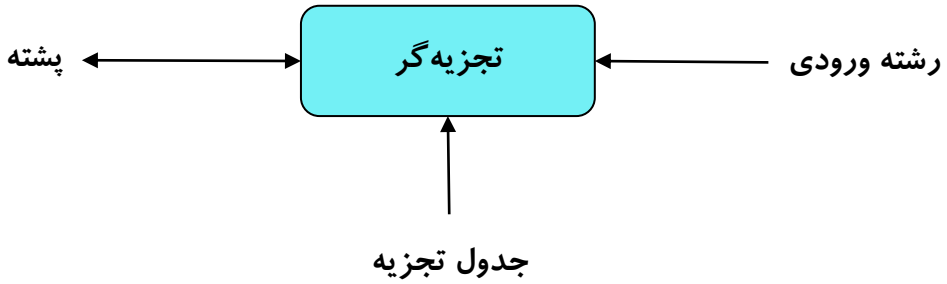
قاعده		First	Follow	Conflict
S → XZY S → ZbY S → Ya	∈ - -	d , e , f f , b e , a	$	e , f
X → da X → YZ	- ∈	d e , f	f , e , $	
Y → e Y → ∈	- ∈	e	a , f , e, $	e
Z → f Z → ∈	- ∈	f	e , b , f, $	f

LL(1)	a	b	d	e	f	$
S	S → Ya	S → ZbY	S → XZY	S → XZY S → Ya	S → XZY S → ZbY	S → XZY
X			X → da	X → YZ	X → YZ	X → YZ
Y	Y → ∈			Y → e Y → ∈	Y → ∈	Y → ∈
Z		Z → ∈		Z → ∈	Z → f Z → ∈	Z → ∈

تجزیه از روی جدول LL(1)

چکیده: تجزیه‌گرهای مبتنی بر جدول عمل تجزیه را با دریافت رشته ورودی ، جدول تجزیه و به کمک یک پشته انجام میدهند. در این درس الگوریتم تجزیه از روی جدول LL(1) را بیان میکنیم.

روش تجزیه پیشگوی مبتنی بر جدول : پس از ساخت جدول LL(1) عمل تجزیه با دریافت رشته ورودی ، جدول تجزیه و به کمک یک پشته انجام میگیرد.



ورودی: در انتهای رشته ورودی نماد $ قرار میگیرد:

$Inp = \alpha\$$

پشته: فرمتهای ابتدایی ، میانی و انتهایی پشته هم به صورت زیر هست:

$Stack_1 = \$S$

$Stack_2 = \$y_1y_2...y_m$

$Stack_3 = \$$

به طوری که هر y_i یک نماد گرامر (پایانه یا غیر پایانه) است.

الگوریتم تجزیه مبتنی بر جدول: وضعیت اولیه بافر ورودی (Inp) و پشته (Stack) را به صورت زیر تبدیل کن:

(Stack= $S , Inp= α$)

سپس بر مبنای محتوای بافر و پشته و جدول تجزیه (M) یکی از گامهای زیر را دنبال کن :

- پذیرفتن: در وضعیت زیر، جمله ورودی را پذیرفته و به الگوریتم تجزیه پایان بده.

(Stack= $, Inp= $)

- تطبیق: در وضعیت زیر، با عمل تطبیق پایانه a را از هر دو طرف حذف کن.

(Stack= $βa , Inp= aα$) →

(Stack= $β , Inp= α$)

- خطای نحوی: در وضعیت زیر، با یک خطای نحوی به الگوریتم تجزیه پایان بده.

(Stack= $βb , Inp= aα$)

- خطای نحوی: در وضعیت زیر، با یک خطای نحوی به الگوریتم تجزیه پایان بده.

(Stack= $βX , Inp= aα$, M[X, a]= "تهی")

- جایگزینی: در وضعیت زیر، نماد X را از پشته حذف کرده و رشته γ را به صورت وارونه جایگزین آن کن.

(Stack= $βX , Inp= aα$, M[X, a]= "X→γ") →

(Stack= $βγ′ , Inp= aα$)

(γ′ وارون رشته γ است، برای نمونه bYXa وارون aXYb است)

مثال۱:

LL(1)	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ∈	E' → ∈
T	T → FT'			T → FT'		
T'		T' → ∈	T' → *FT'		T' → ∈	T' → ∈
F	F → id			F → (E)		

- id + id * id
- (id + ( )

Stack	Inp	Action
$E	id+id*id$	E → TE'
$E'T	id+id*id$	T → FT'
$E'T'F	id+id*id$	F → id
$E'T'id	id+id*id$	Match
$E'T'	+id*id$	T' → ∈
$E'	+id*id$	E' → +TE'
$E'T+	+id*id$	Match
$E'T	id*id$	T → FT'
$E'T'F	id*id$	F → id
$E'T'id	id*id$	Match
$E'T'	*id$	T' → *FT'
$E'T'F*	*id$	Match
$E'T'F	id$	F → id
$E'T'id	id$	Match
$E'T'	$	T' → ∈
$E'	$	E' → ∈
$	$	Accept

Stack	Inp	Action
$E	(id+()$	E → TE'
$E'T	(id+()$	T → FT'
$E'T'F	(id+()$	F → (E)
$E'T')E(	(id+()$	Match
$E'T')E	id+()$	E → TE'
$E'T')E'T	id+()$	T → FT'
$E'T')E'T'F	id+()$	F → id
$E'T')E'T'id	id+()$	Match
$E'T')E'T'	+()$	T' → ∈
$E'T')E'	+()$	E' → +TE'
$E'T')E'T+	+()$	Match
$E'T')E'T	()$	T → FT'
$E'T')E'T'F	()$	F → (E)
$E'T')E'T')E(	()$	Match
$E'T')E'T')E	)$	Error
Error : 'id' , '(' Expected		

LMD: E ⇒ TE' ⇒ FT'E' ⇒ idT'E' ⇒ idE' ⇒ id+TE' ⇒ id+FT'E' ⇒ id+idT'E' ⇒ id+id*FT'E' ⇒ id+id*idT'E' ⇒ id+id*idE' ⇒ id+id*id

مثال۲:

LL(1)	a	b	d	$
S	S → XdY		S → XdY	
X	X → aX		X → ∈	
Y	Y → ∈	Y → bYS	Y → ∈	Y → ∈

- adbdbad
- dbdb

Stack	Inp	Action
$S	adbdbad$	S → XdY
$YdX	adbdbad$	X → aX
$YdXa	adbdbad$	Match
$YdX	dbdbad$	X → ∈
$Yd	dbdbad$	Match
$Y	bdbad$	Y → bYS
$SYb	bdbad$	Match
$SY	dbad$	Y → ∈
$S	dbad$	S → XdY
$YdX	dbad$	X → ∈
$Yd	dbad$	Match
$Y	bad$	Y → bYS
$SYb	bad$	Match
$SY	ad$	Y → ∈
$S	ad$	S → XdY
$YdX	ad$	X → aX
$YdXa	ad$	Match
$YdX	d$	X → ∈
$Yd	d$	Match
$Y	$	Y → ∈
$	$	Accept

Stack	Inp	Action
$S	dbdb$	S → XdY
$YdX	dbdb$	X → ∈
$Yd	dbdb$	Match
$Y	bdb$	Y → bYS
$SYb	bdb$	Match
$SY	db$	Y → ∈
$S	db$	S → XdY
$YdX	db$	X → ∈
$Yd	db$	Match
$Y	b$	Y → bYS
$SYb	b$	Match
$SY	$	Y → ∈
$S	$	Error
Error : 'a' , 'd' Expected		

LMD: $\underline{S} \Rightarrow \underline{X}dY \Rightarrow a\underline{X}dY \Rightarrow ad\underline{Y} \Rightarrow adb\underline{YS} \Rightarrow adb\underline{S} \Rightarrow adb\underline{X}dY \Rightarrow adbd\underline{Y} \Rightarrow adbdb\underline{YS} \Rightarrow adbdb\underline{S} \Rightarrow adbdb\underline{X}dY \Rightarrow adbdba\underline{X}dY \Rightarrow adbdbad\underline{Y} \Rightarrow adbdbad$

تجزیه از روی جدول LR

چکیده: تجزیه‌گرهای پایین به بالا (نوع LR) تنها به روش مبتنی بر جدول طراحی میشوند. در این روش عمل تجزیه از روی یک جدول به نام LR انجام میگیرد. از آنجاییکه الگوریتم نسبتا دشواری در ساخت جدول LR به کار میرد، در ادامه روش تجزیه از روی جدول را پیش از ساخت آن شرح میدهیم.

تجزیه‌گرهای LR: پیش از عمل تجزیه یک جدول تجزیه به نام LR برای گرامر ایجاد میشود. از آنجاییکه جدول LR از روی یک نمودار انتقالی ساخته میشود، جدول تجزیه از برخورد وضعیتها با نمادهای گرامر ایجاد میشود.

جدول تجزیه: با فرض اینکه $S_0, S_1, \ldots, S_n$ مجموعه وضعیتها ، $a_1, a_2, \ldots, a_m$ پایانه‌ها و $X_1, X_2, \ldots, X_p$ غیرپایانه‌های گرامر باشند، جدول تجزیه یک ساختار دو بخشی به صورت زیر دارد:

LR	بخش Action				بخش Goto			
	a_1	a_2	...	a_m	X_1	X_2	...	X_p
S_0								
S_1								
...								
S_n								

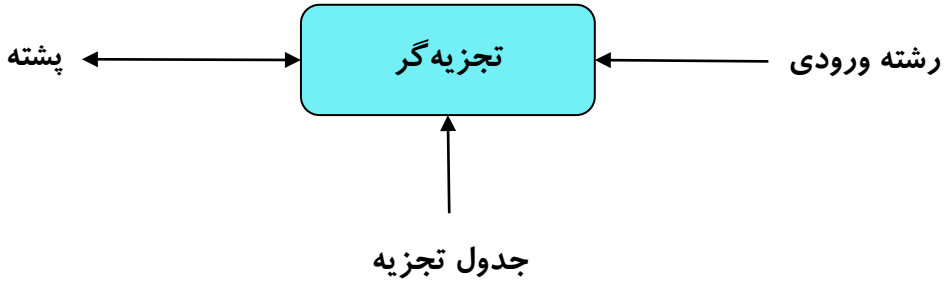
خانه‌های بخش Action

- خالی: به معنی خطای نحوی است.
- شامل یک فرمان: به یکی از فرمتهای Shift Sj ، Reduce X→β و Accept است.
- تداخل داشته باشد: در این صورت گرامر LR نیست.

خانه‌های بخش Goto

- خالی: هیچگاه به یک خانه خالی انتقالی انجام نمیشود.
- شامل یک فرمان: فرمانهای این بخش با فرمت Sj هستند.

روش تجزیه LR : پس از ساخت جدول LR عمل تجزیه با دریافت رشته ورودی ، جدول تجزیه و به کمک یک پشته انجام میگیرد.



ورودی: در انتهای رشته ورودی نماد $ قرار میگیرد:

$Inp = \alpha\$$

پشته: فرمتهای ابتدایی ، میانی و انتهایی پشته هم به صورت زیر هست:

$Stack_1 = 0$

$Stack_2 = s_0 y_1 s_1 y_2 s_2 \ldots y_m s_m$

$Stack_3 = 0S1$

به طوری که هر S_i یک وضعیت نمودار و هر y_i هم یک نماد گرامر (پایانه یا غیر پایانه) است.

الگوریتم تجزیه LR : وضعیت اولیه بافر ورودی (Inp) و پشته (Stack) را به صورت زیر تبدیل کن:

(Stack= 0 , Inp= α$)

سپس بر مبنای محتوای بافر و پشته و جدول تجزیه یکی از گامهای زیر را دنبال کن :

- پذیرفتن: در وضعیت زیر، جمله ورودی را پذیرفته و به الگوریتم تجزیه پایان بده.

(Stack= βS_i , Inp= $, A[S_i, $]= "Accept")

- خطای نحوی: در وضعیت زیر، با یک خطای نحوی به الگوریتم تجزیه پایان بده.

(Stack= βS_i , Inp= $a\alpha$, A[S_i, a]= "تهی")

- انتقال: در وضعیت زیر، با عمل انتقال پایانه a و سپس وضعیت S_j را روی پشته قرار بده.

(Stack= βS_i , Inp= $a\alpha$, A[S_i, a]= "Shift S_j") →
(Stack= $\beta S_i a S_j$, Inp= α$)

- کاهش: اگر وضعیت تجزیه‌گر به صورت زیر باشد، طی دو مرحله زیر آنرا خلاصه کن :
  - رشته $Y_1Y_2...Y_m$ از روی پشته برداشته شده و آن را با X جایگزین کن.

(Stack= $\beta S_i Y_1 S_1 Y_2 S_2 ... Y_m S_m$, Inp= $a\alpha$, A[S_m, a]= "Reduce $X \rightarrow Y_1Y_2...Y_m$") →
(Stack= $\beta S_i X$, Inp= aα$)

  - وضعیت S_j را در بخش Goto برداشته و روی پشته اضافه کن.

(Stack= $\beta S_i X$, Inp= aα$, G[S_i, X]= "S_j") →
(Stack= $\beta S_i X S_j$, Inp= aα$)

مثال۱:

LR	Action						Goto		
	id	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				Acc			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

1. E → E+T
2. E → T
3. T → T*F
4. T → F
5. F → (E)
6. F → id

- id + id * id
- (id + ( )

Stack	Inp	Action
0	id+id*id$	S5
0id5	+id*id$	F → id
0F3	+id*id$	T → F
0T2	+id*id$	E → T
0E1	+id*id$	S6
0E1+6	id*id$	S5
0E1+6id5	*id$	F → id
0E1+6F3	*id$	T → F
0E1+6T9	*id$	S7
0E1+6T9*7	id$	S5
0E1+6T9*7id5	$	F → id
0E1+6T9*7F10	$	T → T*F
0E1+6T9	$	E → E+T
0E1	$	Accept

Stack	Inp	Action
0	(id+()$	S4
0(4	id+()$	S5
0(4id5	+()$	F → id
0(4F3	+()$	T → F
0(4T2	+()$	E → T
0(4E8	+()$	S6
0(4E8+6	()$	S4
0(4E8+6(4	)$	Error
Error : 'id' , '(' Expected		

RMD: E ⇒ E+T ⇒ E+T*F ⇒ E+T*id ⇒ E+F*id ⇒ E+id*id ⇒ T+id*id ⇒ F+id*id ⇒ id+id*id

مثال۲:

1. S → XdY
2. X → aX
3. X → ∈
4. Y → bYS
5. Y → ∈

- adbdbad
- dbdb

LR	Action				Goto		
	a	b	d	$	S	X	Y
0	S3		R3		1	2	
1				Acc			
2			S4				
3	S3		R3			5	
4	R5	S7	R5	R5			6
5			R2				
6	R1		R1	R1			
7	R5	S7	R5	R5			8
8	S3		R3		9	2	
9	R4		R4	R4			

Stack	Inp	Action
0	adbdbad$	S3
0a3	dbdbad$	X → ∈
0a3X5	dbdbad$	X → aX
0X2	dbdbad$	S4
0X2d4	bdbad$	S7
0X2d4b7	dbad$	Y → ∈
0X2d4b7Y8	dbad$	X → ∈
0X2d4b7Y8X2	dbad$	S4
0X2d4b7Y8X2d4	bad$	S7
0X2d4b7Y8X2d4b7	ad$	Y → ∈
0X2d4b7Y8X2d4b7Y8	ad$	S3
0X2d4b7Y8X2d4b7Y8a3	d$	X → ∈
0X2d4b7Y8X2d4b7Y8a3X5	d$	X → aX
0X2d4b7Y8X2d4b7Y8X2	d$	S4
0X2d4b7Y8X2d4b7Y8X2d4	$	Y → ∈
0X2d4b7Y8X2d4b7Y8X2d4Y6	$	S → XdY
0X2d4b7Y8X2d4b7Y8S9	$	Y → bYS
0X2d4b7Y8X2d4Y6	$	S → XdY
0X2d4b7Y8S9	$	Y → bYS
0X2d4Y6	$	S → XdY
0S1	$	Accept

Stack	Inp	Action
0	dbdb$	X → ∈
0X2	dbdb$	S4
0X2d4	bdb$	S7
0X2d4b7	db$	Y → ∈
0X2d4b7Y8	db$	X → ∈
0X2d4b7Y8X2	db$	S4
0X2d4b7Y8X2d4	b$	S7
0X2d4b7Y8X2d4b7	$	Y → ∈
0X2d4b7Y8X2d4b7Y8	$	Error
Error : 'a' , 'd' Expected		

RMD: S ⇒ XdY ⇒ XdbYS ⇒ XdbYXdY ⇒ XdbYXdbYS ⇒ XdbYXdbYXdY ⇒ XdbYXdbYXd ⇒ XdbYXdbYaXd ⇒ XdbYXdbYad ⇒ XdbYXdbad ⇒ XdbYdbad ⇒ Xdbdbad ⇒ aXdbdbad ⇒ adbdbad

روش ساخت جدول LR: جدول LR از روی یک نمودار انتقالی و مجموعه Follow ایجاد میشود. نمودار انتقالی از نوع DFA است، با این تفاوت که برچسب هر کمان یک نماد گرامر (پایانه یا غیر پایانه) است و هر وضعیت ساختمان داده‌ای است که مجموعه‌ای از قواعد نقطه‌دار را در خود نگهداری میکند.

قاعده نقطه‌دار: قاعده‌ای است که یک نقطه در سمت راست خود داشته باشد و نشان میدهد چه بخشی از قاعده تایید شده است.

$S \rightarrow \alpha.\beta$

۱- قاعده کامل (دستگیره): چنانچه نقطه در انتهای قاعده باشد.

$S \rightarrow XdY.$

$X \rightarrow .$

۲- قاعده فعال: چنانچه نقطه در انتهای قاعده نباشد.

$S \rightarrow .XdY$

$S \rightarrow X.dY$

$S \rightarrow Xd.Y$

۳- قاعده اولیه: اگر هیچ بخشی از قاعده فعال تایید نشده باشد (نقطه در ابتدای آن باشد).

$S \rightarrow .XdY$

ایجاد وضعیت: همانگونه که اشاره شد هر وضعیت ساختمان داده‌ای است که مجموعه‌ای از قواعد نقطه‌دار را در خود نگهداری میکند. ولی این ساختمان داده با الگوریتم درج قاعده و با افزودن یک قاعده نقطه‌دار در آن ایجاد میشود:

الگوریتم درج قاعده: برای درج قاعده نقطه‌دار R در وضعیت (یا ساختمان داده) Si عملیات زیر را انجام بده:

۱- چنانچه R در Si موجود نباشد آن را به Si اضافه کرده وگرنه از الگوریتم خارج شو.

۲- اگر R یک قاعده فعال به صورت $X \to \alpha . Y\beta$ باشد (Y غیرپایانه است)، به ازای هر قاعده $Y \to \gamma$ در گرامر قاعده نقطه‌دار $Y \to . \gamma$ را به کمک الگوریتم درج قاعده به Si اضافه کن.

مثال: با فرض گرامر زیر ، درج قاعده نقطه‌دار $E \to . T$ در وضعیت Si باعث ایجاد مجموعه قواعد زیر میشود.

```
E → E+T | T
T → T*F | F
F → (E) | id
```

S_i:

```
E → .T
T → .T*F
T → .F
F → .(E)
F → .id
```

الگوریتم ساخت نمودار: برای ساخت نمودار عملیات زیر را انجام بده:

۱- **ایجاد وضعیت اولیه:** یک قاعده افزوده به صورت S'→S به مجموع قواعد گرامر اضافه کرده و قاعده اولیه S' → . S را طبق الگوریتم درج قاعده به وضعیت صفر (S0) اضافه کن.

۲- **جلو بردن نقطه و ایجاد وضعیتهای جدید:** به ازای هر نماد y (پایانه یا غیر پایانه) چنانچه قواعد نقطه‌داری به صورت X→α . yβ در وضعیت Si موجود باشند، یک وضعیت جدید به نام Sj در نمودار ایجاد کن و قواعد نقطه‌دار X→αy . β را طبق الگوریتم درج قاعده به وضعیت Sj اضافه کن.

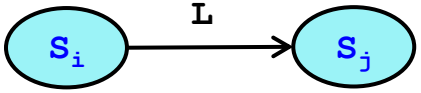
۳- **ایجاد کمان:** اگر محتوای Sj با هیچ یک از وضعیتهای ایجاد شده قبل یکسان نباشد کمان جهتداری با برچسب y از Si به Sj رسم کن. و اگر محتوای Sj با وضعیت ایجاد شده Sk یکسان باشد، Sj را حذف کرده و کمان جهتداری با برچسب y از Si به Sk ترسیم کن.

مثال: با فرض گرامر زیر ، جلوبردن نقطه روی نماد L در وضعیت Si باعث ایجاد وضعیت Sj میشود.

S’ → S
S → Te
L → La | ∈
T → Tb | LRL
R → dLT | ∈

S_i:
S' → .S
S → .Te
T → .Tb
T → .LRL
L → .La
L → .

S_j:
T → L.RL
L → L.a
R → .dLT
R → .



0: E’ → E
1, 2: E → E+T | T
3, 4: T → T*F | F
5, 6: F → (E) | id

نماد	Follow
E	+ ,) , $
T	+ , * ,) , $
F	+ , * ,) , $

0: **E'→ .E** $\overset{E}{\Rightarrow}$ 1 E → .E+T $\overset{E}{\Rightarrow}$ 1 E → .T $\overset{T}{\Rightarrow}$ 2 T → .T*F $\overset{T}{\Rightarrow}$ 2 T → .F $\overset{F}{\Rightarrow}$ 3 F → .(E) $\overset{(}{\Rightarrow}$ 4 F → .id $\overset{id}{\Rightarrow}$ 5	**1:** **E'→ E.** $\overset{\$}{\Rightarrow}$ Acc **E → E.+T** $\overset{+}{\Rightarrow}$ 6	**2:** **E → T.** $\overset{+)\$}{\Longrightarrow}$ R2 **T → T.*F** $\overset{*}{\Rightarrow}$ 7	**3:** **T → F.** $\overset{+*)\$}{\Longrightarrow}$ R4
4: **F → (.E)** $\overset{E}{\Rightarrow}$ 8 E → .E+T $\overset{E}{\Rightarrow}$ 8 E → .T $\overset{T}{\Rightarrow}$ 2 T → .T*F $\overset{T}{\Rightarrow}$ 2 T → .F $\overset{F}{\Rightarrow}$ 3 F → .(E) $\overset{(}{\Rightarrow}$ 4 F → .id $\overset{id}{\Rightarrow}$ 5 | **5:** **F → id.** $\overset{+*)\$}{\Longrightarrow}$ R6	**6:** **E → E+.T** $\overset{T}{\Rightarrow}$ 9 T → .T*F $\overset{T}{\Rightarrow}$ 9 T → .F $\overset{F}{\Rightarrow}$ 3 F → .(E) $\overset{(}{\Rightarrow}$ 4 F → .id $\overset{id}{\Rightarrow}$ 5	**7:** **T → T*.F** $\overset{F}{\Rightarrow}$ 10 F → .(E) $\overset{(}{\Rightarrow}$ 4 F → .id $\overset{id}{\Rightarrow}$ 5
8: **F → (E.)** $\overset{)}{\Rightarrow}$ 11 **E → E.+T** $\overset{+}{\Rightarrow}$ 6	**9:** **E → E+T.** $\overset{+)\$}{\Longrightarrow}$ R1 **T → T.*F** $\overset{*}{\Rightarrow}$ 7	**10:** **T → T*F.** $\overset{+*)\$}{\Longrightarrow}$ R3	**11:** **F → (E).** $\overset{+*)\$}{\Longrightarrow}$ R5

0: E'→ .E $\overset{E}{\Rightarrow}$ 1 E → .E+T $\overset{E}{\Rightarrow}$ 1 E → .T $\overset{T}{\Rightarrow}$ 2 T → .T*F $\overset{T}{\Rightarrow}$ 2 T → .F $\overset{F}{\Rightarrow}$ 3 F → .(E) $\overset{(}{\Rightarrow}$ 4 F → .id $\overset{id}{\Rightarrow}$ 5	1: E'→ E. $\overset{\$}{\Rightarrow}$ Acc E → E.+T $\overset{+}{\Rightarrow}$ 6	2: E → T. $\overset{+)\$}{\Longrightarrow}$ R2 T → T.*F $\overset{*}{\Rightarrow}$ 7	3: T → F. $\overset{+*)\$}{\Longrightarrow}$ R4
4: F → (.E) $\overset{E}{\Rightarrow}$ 8 E → .E+T $\overset{E}{\Rightarrow}$ 8 E → .T $\overset{T}{\Rightarrow}$ 2 T → .T*F $\overset{T}{\Rightarrow}$ 2 T → .F $\overset{F}{\Rightarrow}$ 3 F → .(E) $\overset{(}{\Rightarrow}$ 4 F → .id $\overset{id}{\Rightarrow}$ 5	5: F → id. $\overset{+*)\$}{\Longrightarrow}$ R6	6: E → E+.T $\overset{T}{\Rightarrow}$ 9 T → .T*F $\overset{T}{\Rightarrow}$ 9 T → .F $\overset{F}{\Rightarrow}$ 3 F → .(E) $\overset{(}{\Rightarrow}$ 4 F → .id $\overset{id}{\Rightarrow}$ 5	7: T → T*.F $\overset{F}{\Rightarrow}$ 10 F → .(E) $\overset{(}{\Rightarrow}$ 4 F → .id $\overset{id}{\Rightarrow}$ 5
8: F → (E.) $\overset{)}{\Rightarrow}$ 11 E → E.+T $\overset{+}{\Rightarrow}$ 6	9: E → E+T. $\overset{+)\$}{\Longrightarrow}$ R1 T → T.*F $\overset{*}{\Rightarrow}$ 7	10: T → T*F. $\overset{+*)\$}{\Longrightarrow}$ R3	11: F → (E). $\overset{+*)\$}{\Longrightarrow}$ R5

LR	Action						Goto		
	id	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				Acc			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

0: S’ → S
1: S → XdY
2, 3: X → aX | ∈
4, 5: Y → bYS | ∈

نماد	Follow
S	a , d , $
X	d
Y	a , d , $

0: S' → .S $\overset{S}{\Rightarrow}$ 1 S → .XdY $\overset{X}{\Rightarrow}$ 2 X → .aX $\overset{a}{\Rightarrow}$ 3 X → . $\overset{d}{\Rightarrow}$ R3	1: S' → S. $\overset{\$}{\Rightarrow}$ Acc	2: S → X.dY $\overset{d}{\Rightarrow}$ 4	3: X → a.X $\overset{X}{\Rightarrow}$ 5 X → .aX $\overset{a}{\Rightarrow}$ 3 X → . $\overset{d}{\Rightarrow}$ R3
4: S → Xd.Y $\overset{Y}{\Rightarrow}$ 6 Y → .bYS $\overset{b}{\Rightarrow}$ 7 Y → . $\overset{a\,d\,\$}{\Longrightarrow}$ R5	5: X → aX. $\overset{d}{\Rightarrow}$ R2	6: S → XdY. $\overset{a\,d\,\$}{\Longrightarrow}$ R1	7: Y → b.YS $\overset{Y}{\Rightarrow}$ 8 Y → .bYS $\overset{b}{\Rightarrow}$ 7 Y → . $\overset{a\,d\,\$}{\Longrightarrow}$ R5
8: Y → bY.S $\overset{S}{\Rightarrow}$ 9 S → .XdY $\overset{X}{\Rightarrow}$ 2 X → .aX $\overset{a}{\Rightarrow}$ 3 X → . $\overset{d}{\Rightarrow}$ R3	9: Y → bYS. $\overset{a\,d\,\$}{\Longrightarrow}$ R_4		

0: S' → .S $\overset{S}{\Rightarrow}$ 1 S → .XdY $\overset{X}{\Rightarrow}$ 2 X → .aX $\overset{a}{\Rightarrow}$ 3 X → . $\overset{d}{\Rightarrow}$ R3	**1:** S' → S. $\overset{\$}{\Rightarrow}$ Acc	**2:** S → X.dY $\overset{d}{\Rightarrow}$ 4	**3:** X → a.X $\overset{X}{\Rightarrow}$ 5 X → .aX $\overset{a}{\Rightarrow}$ 3 X → . $\overset{d}{\Rightarrow}$ R3
4: S → Xd.Y $\overset{Y}{\Rightarrow}$ 6 Y → .bYS $\overset{b}{\Rightarrow}$ 7 Y → . $\overset{a\,d\,\$}{\Longrightarrow}$ R5	**5:** X → aX. $\overset{d}{\Rightarrow}$ R2	**6:** S → XdY. $\overset{a\,d\,\$}{\Longrightarrow}$ R1	**7:** Y → b.YS $\overset{Y}{\Rightarrow}$ 8 Y → .bYS $\overset{b}{\Rightarrow}$ 7 Y → . $\overset{a\,d\,\$}{\Longrightarrow}$ R5
8: Y → bY.S $\overset{S}{\Rightarrow}$ 9 S → .XdY $\overset{X}{\Rightarrow}$ 2 X → .aX $\overset{a}{\Rightarrow}$ 3 X → . $\overset{d}{\Rightarrow}$ R3	**9:** Y → bYS. $\overset{a\,d\,\$}{\Longrightarrow}$ R_4		

LR	Action				Goto		
	a	b	d	$	S	X	Y
0	S3		R3		1	2	
1				Acc			
2			S4				
3	S3		R3			5	
4	R5	S7	R5	R5			6
5			R2				
6	R1		R1	R1			
7	R5	S7	R5	R5			8
8	S3		R3		9	2	
9	R4		R4	R4			

0: S' → S
1: S → Te
2, 3: L → La | ∈
4, 5: T → Tb | LRL
6, 7: R → dLT | ∈

نماد	Follow
S	$
L	a , b , d , e
T	a , b , e
R	a , b , e

0: **S' → .S** $\overset{S}{\Rightarrow}$ 1 S → .Te $\overset{T}{\Rightarrow}$ 3 T → .Tb $\overset{T}{\Rightarrow}$ 3 T → .LRL $\overset{L}{\Rightarrow}$ 2 L → .La $\overset{L}{\Rightarrow}$ 2 L → . $\overset{abde}{\Longrightarrow}$ R3	**1:** **S' → S.** $\overset{\$}{\Rightarrow}$ Acc	**2:** **T → L.RL** $\overset{R}{\Rightarrow}$ 4 **L → L.a** $\overset{a}{\Rightarrow}$ 5 R → .dLT $\overset{d}{\Rightarrow}$ 6 R → . $\overset{abe}{\Longrightarrow}$ R7	**3:** **S → T.e** $\overset{e}{\Rightarrow}$ 8 **T → T.b** $\overset{b}{\Rightarrow}$ 7
4: **T → LR.L** $\overset{L}{\Rightarrow}$ 9 L → .La $\overset{L}{\Rightarrow}$ 9 L → . $\overset{abde}{\Longrightarrow}$ R3	**5:** **L → La.** $\overset{abde}{\Longrightarrow}$ R2	**6:** **R → d.LT** $\overset{L}{\Rightarrow}$ 10 L → .La $\overset{L}{\Rightarrow}$ 10 L → . $\overset{abde}{\Longrightarrow}$ R3	**7:** **T → Tb.** $\overset{abe}{\Longrightarrow}$ R4
8: **S → Te.** $\overset{\$}{\Rightarrow}$ R1	**9:** **T → LRL.** $\overset{abe}{\Longrightarrow}$ R5 **L → L.a** $\overset{a}{\Rightarrow}$ 5	**10:** **R → dL.T** $\overset{T}{\Rightarrow}$ 11 **L → L.a** $\overset{a}{\Rightarrow}$ 5 T → .Tb $\overset{T}{\Rightarrow}$ 11 T → .LRL $\overset{L}{\Rightarrow}$ 2 L → .La $\overset{L}{\Rightarrow}$ 2 L → . $\overset{abde}{\Longrightarrow}$ R3	**11:** **R → dLT.** $\overset{abe}{\Longrightarrow}$ R6 **T → T.b** $\overset{b}{\Rightarrow}$ 7

0: S' → .S $\overset{S}{\Rightarrow}$ 1 S → .Te $\overset{T}{\Rightarrow}$ 3 T → .Tb $\overset{T}{\Rightarrow}$ 3 T → .LRL $\overset{L}{\Rightarrow}$ 2 L → .La $\overset{L}{\Rightarrow}$ 2 L → . $\overset{abde}{\Longrightarrow}$ R3	1: S' → S. $\overset{\$}{\Rightarrow}$ Acc	2: T → L.RL $\overset{R}{\Rightarrow}$ 4 L → L.a $\overset{a}{\Rightarrow}$ 5 R → .dLT $\overset{d}{\Rightarrow}$ 6 R → . $\overset{abe}{\Longrightarrow}$ R7	3: S → T.e $\overset{e}{\Rightarrow}$ 8 T → T.b $\overset{b}{\Rightarrow}$ 7
4: T → LR.L $\overset{L}{\Rightarrow}$ 9 L → .La $\overset{L}{\Rightarrow}$ 9 L → . $\overset{abde}{\Longrightarrow}$ R3	5: L → La. $\overset{abde}{\Longrightarrow}$ R2	6: R → d.LT $\overset{L}{\Rightarrow}$ 10 L → .La $\overset{L}{\Rightarrow}$ 10 L → . $\overset{abde}{\Longrightarrow}$ R3	7: T → Tb. $\overset{abe}{\Longrightarrow}$ R4
8: S → Te. $\overset{\$}{\Rightarrow}$ R1	9: T → LRL. $\overset{abe}{\Longrightarrow}$ R5 L → L.a $\overset{a}{\Rightarrow}$ 5	10: R → dL.T $\overset{T}{\Rightarrow}$ 11 L → L.a $\overset{a}{\Rightarrow}$ 5 T → .Tb $\overset{T}{\Rightarrow}$ 11 T → .LRL $\overset{L}{\Rightarrow}$ 2 L → .La $\overset{L}{\Rightarrow}$ 2 L → . $\overset{abde}{\Longrightarrow}$ R3	11: R → dLT. $\overset{abe}{\Longrightarrow}$ R6 T → T.b $\overset{b}{\Rightarrow}$ 7

LR	Action					Goto			
	a	b	d	e	$	S	L	T	R
0	R3	R3	R3	R3		1	2	3	
1					Acc				
2	S5,R7	R7	S6	R7					4
3		S7		S8					
4	R3	R3	R3	R3			9		
5	R2	R2	R2	R2					
6	R3	R3	R3	R3			10		
7	R4	R4		R4					
8					R1				
9	S5,R5	R5		R5					
10	S5,R3	R3	R3	R3			2	11	
11	R6	S7,R6		R6					

نماد	Follow
S	$
V	= , $
E	$

0: S’ → S
1, 2: S → V=E | id
3 : V → id
4, 5: E → V | num

LR	Action				Goto		
	=	id	num	$	S	V	E
0		S3			1	2	
1				Acc			
2	S4						
3	R3			R2,R3			
4		S7	S8			5	6
5				R4			
6				R1			
7	R3			R3			
8				R5			

0: **S' → .S** $\overset{S}{\Rightarrow}$ 1 S → .V=E $\overset{V}{\Rightarrow}$ 2 S → .id $\overset{id}{\Rightarrow}$ 3 V → .id $\overset{id}{\Rightarrow}$ 3	1: **S' → S.** $\overset{\$}{\Rightarrow}$ Acc	2: **S → V.=E** $\overset{=}{\Rightarrow}$ 4
3: **S → id.** $\overset{\$}{\Rightarrow}$ R2 **V → id.** $\overset{=\$}{\Rightarrow}$ R3	4: **S → V=.E** $\overset{E}{\Rightarrow}$ 6 E → .V $\overset{V}{\Rightarrow}$ 5 E → .num $\overset{num}{\Rightarrow}$ 8 V → .id $\overset{id}{\Rightarrow}$ 7	5: **E → V.** $\overset{\$}{\Rightarrow}$ R4
6: **S → V=E.** $\overset{\$}{\Rightarrow}$ R1	7: **V → id.** $\overset{=\$}{\Rightarrow}$ R3	8: **E → num.** $\overset{\$}{\Rightarrow}$ R5