فصل ۴

تحلیلگر نحوی (تحلیل ساختار دستور)

اهداف رفتاری:

دانشجو پس از مطالعه این فصل با مفاهیم زیر آشنا خواهد شد:

- **تحلیل گر نحوی و خطاهای نحوی**
- **گرامر و گرامر مستقل از متن**
- **تجزیه بالا به پایین و پایین به بالا**
- **تجزیه پیشگو**
- **گرامرهای LL(1)، LR، SLR و LALR**

۱-۴ فواید گرامرها

۱- نمایش دقیق و قابل فهمی برای زبان

۲- امکان ایجاد پارسرهای کارآمد با قابلیت تشخیص ساختارهای نحوی درست و دقیق

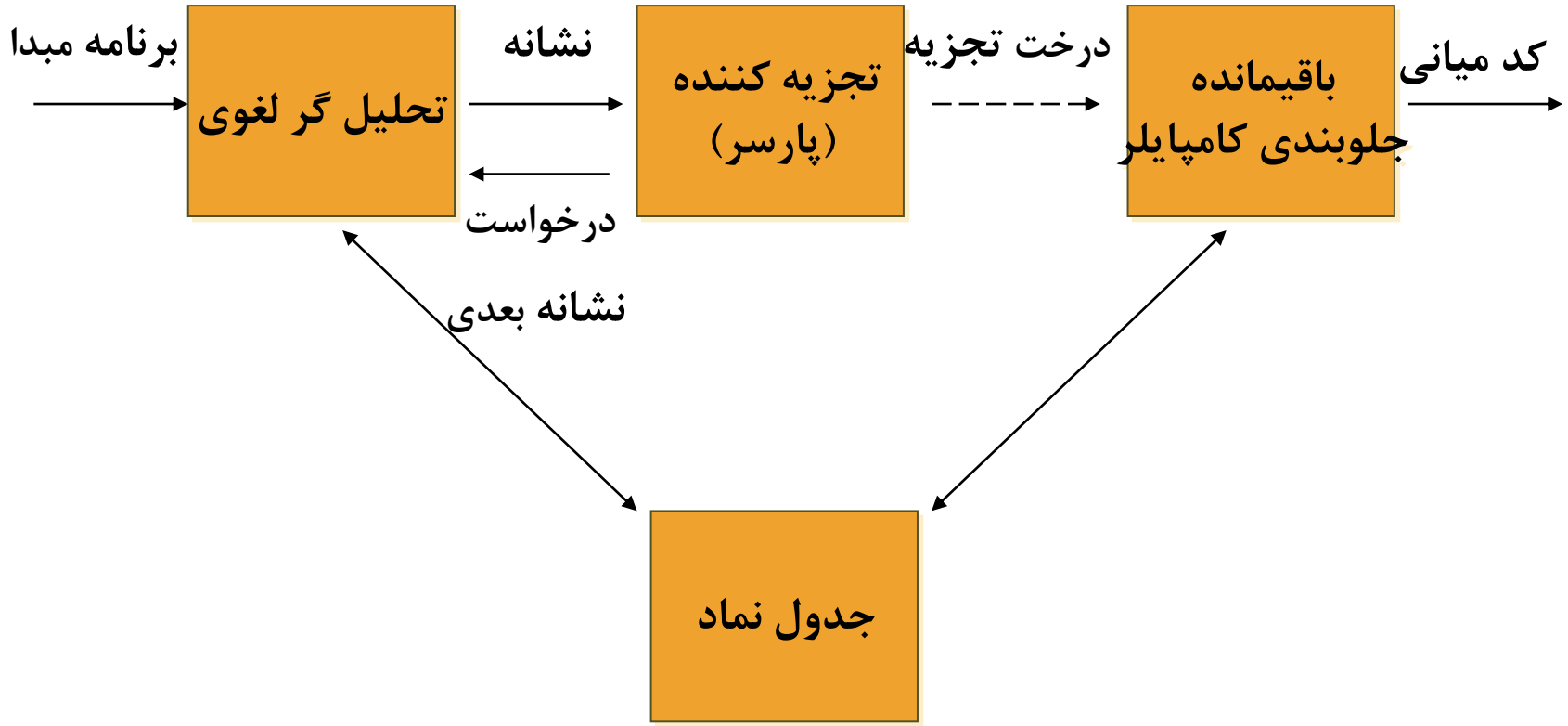
۳- ایجاد ساختاری مناسب برای زبان جهت ترجمه صحیح و آشکارسازی خطا توسط گرامر درست طراحی شده

۴- سادگی اضافه نمودن ساختارهای جدید به زبان

۲-۴ تجزیه کننده (پارسر)

دریافت رشته ای از نشانه ها از تحلیل گر لغوی و بررسی تعلق رشته به زبان توسط گرامر
انجام بررسی طبق ساختارهای نحوی زبان و هر مرحله گزارش خطاهای نحوی به اداره کننده خطا
رفع خطا برای پردازش ادامه ورودی بر اساس خطاهای متداول

۴-۲-۱ تجزیه کننده- ارتباطات



موقعیت تجزیه کننده در مدل کامپایلر

پارسر

- سه نوع پارسر عمومی برای گرامرها وجود دارد:
  - الگوریتم CYK (Cocke-Younger-Kasami)
  - الگوریتم Early
  - آنقدر کارا نیستند در تولید کامپایلرها
- دو دسته کلی روش هایی که برای کامپایلرها استفاده می شوند:
  - بالا به پایین
  - پایین به بالا
  - هر دو چپ به راست و هر بار یک نماد از ورودی را پویش می کنند.
  - نمونه: LL و LR

بررسی خطاهای نحوی (دستوری)

- برنامه نویسان، اغلب برنامه های نادرست می نویسند.
- یک کامپایلر خوب باید در پیدا کردن خطاها و تعیین محل آن ها به برنامه نویس کمک کند.
- طراحی درست کنترل و بررسی خطا از ابتدای طراحی کامپایلر می تواند باعث ساده شدن ساختار کامپایلر شود و چگونگی پاسخگویی به خطاها را بهبود بخشد.
- قسمت عمده آشکارسازی و تصحیح خطا در یک کامپایلر، در فاز تحلیل لغوی آن است زیرا بسیاری از خطاها، دستوری هستند.

۳-۴ خطای نحوی

الف- سطوح خطا

۱- **لغوی** مانند دیکته نادرست شناسه، کلمه کلیدی یا عملگر

۲- **نحوی (دستوری)** مانند عبارت محاسباتی با پرانتزهای نامتوازن

۳- **معنایی** مانند استفاده از عملگر با عملوندهای ناسازگار

۴- **منطقی** مانند فراخوانی بازگشتی بی نهایت

۳-۴ خطای نحوی

ب- ویژگی اداره کننده خطای نحوی

- توانایی گزارش حضور خطاها را با وضوح و با دقت
- پوشش هر خطا با سرعت کافی به جهت امکان آشکارسازی خطاهای بعدی
- عدم کاهش بیش از حد سرعت پردازش برنامه های صحیح

۳-۴ خطای نحوی

ج- استراتژی های پوشش خطای نحوی

- حالت اضطراری (تعجیلی)
- سطح عبارت
- قوانین تولید خطا
- تصحیح سراسری

-Panic Mode

-Phrase level

-Error production

-Global correction

۳-۴ خطای نحوی - Panic mode

ویژگی

- ساده ترین روش پوشش
- قابل استفاده اکثر روشهای تجزیه
- وارد حلقه بی نهایت نمی شود.

روش کار

صرف نظر از یک نماد در هر مرحله تا زمان پیداشدن نشانه هماهنگ با زبان

۳-۴ خطای نحوی - Phrase Level

ویژگی

- استفاده از تصحیح موضعی
- عدم ورود به حلقه بی نهایت با دقت در انتخاب جایگزینی
- ضعف در برخورد با خطاهای اصلی قبل از نقطه تشخیص
- قادر به تصحیح هر رشته ورودی

روش کار

پیشوندی از باقیمانده ورودی را جایگزین رشته ای می نماید که امکان ادامه تجزیه باشد.

۳-۴ خطای نحوی - Error production

روش کار

اضافه نمودن ساختارهای مولد خطا به زبان از قبل تشخیص آنها در زمان تجزیه در رشته ورودی

۳-۴ خطای نحوی- Global Correction

روش کار

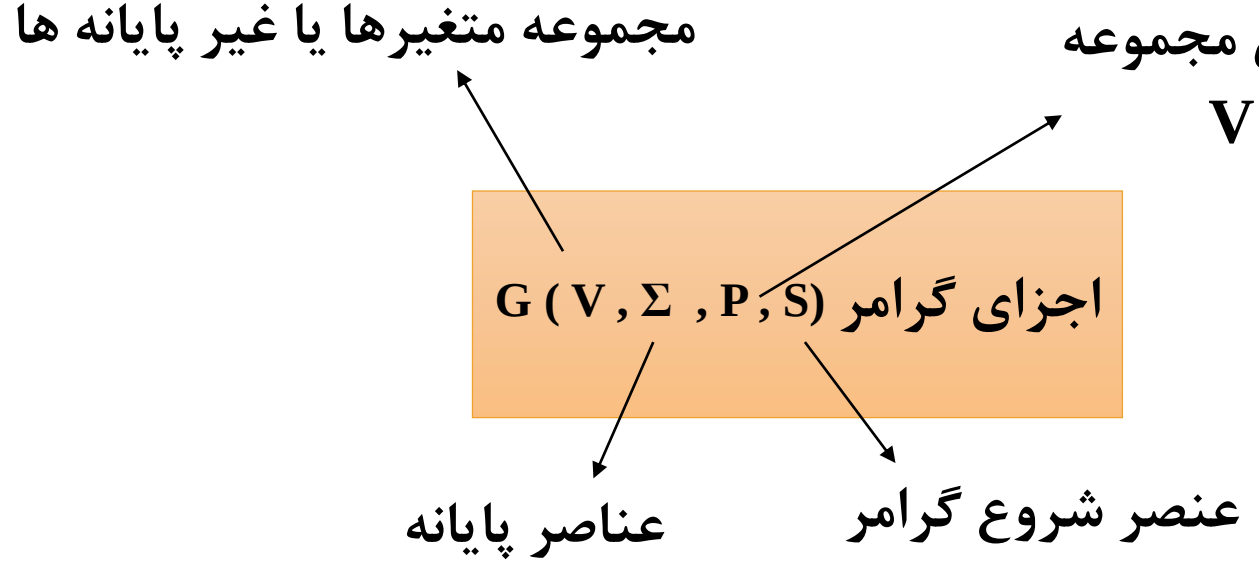
انتخاب الگوریتم های تصحیح خطا با قابلیت ایجاد کمترین تغییرات در ورودی برای رفع خطا

رخ دادن حداقل تعداد درج ها، حذفها در رشته ورودی

۴-۴ گرامر مستقل از متن

قوانین : اعضای مجموعه

$V^*(V \cup \Sigma)$



ویژگی زبان

$S \to \lambda$ ($\lambda \in L$ اگر و تنها اگر)

$A \to u \; (A \in V, u \in (\Sigma \cup V)^+$

۱-۴-۴ گرامر مستقل از متن - تعاریف

فرم جمله ای	رشته $w \in (V \cup \Sigma)$ یک فرم جمله ایست اگر یک اشتقاق از S. به w وجود داشته باشد
جمله	رشته $w \in \Sigma^*$ یک جمله است اگر یک اشتقاق از S به w وجود داشته باشد
زبان	زبان G که با $L(G)$ نشان می دهند، مجموعه $\{w \in \Sigma^*\}$ است.

۴- ۴گرامر مستقل از متن نمونه اشتقاق های یک رشته

$S \rightarrow AA$

$A \rightarrow AAA \mid bA \mid Ab \mid a$

رشته **ababaa**

$S \rightarrow AA$	$S \rightarrow AA$	$S \rightarrow AA$	$S \rightarrow AA$
$\rightarrow aA$	$\rightarrow AAAA$	$\rightarrow Aa$	$\rightarrow aA$
$\rightarrow aAAA$	$\rightarrow aAAA$	$\rightarrow AAAa$	$\rightarrow aAAA$
$\rightarrow abAAA$	$\rightarrow abAAA$	$\rightarrow AAbAa$	$\rightarrow aAAa$
$\rightarrow abaAA$	$\rightarrow abaAA$	$\rightarrow AAbaa$	$\rightarrow abAAa$
$\rightarrow ababAA$	$\rightarrow ababAA$	$\rightarrow AbAbaa$	$\rightarrow abAbAa$
$\rightarrow ababaA$	$\rightarrow ababaA$	$\rightarrow Ababaa$	$\rightarrow ababAa$
$\rightarrow ababaa$	$\rightarrow ababaa$	$\rightarrow ababaa$	$\rightarrow ababaa$

مثال گرامرهای مستقل از متن

$S \rightarrow aSb \mid aSbb$

$S \rightarrow aSdd \mid A$
$A \rightarrow bAc \mid bc$

$S \rightarrow aSa \mid aBa$
$B \rightarrow bB \mid b$

$S \rightarrow aS \mid aB$
$B \rightarrow bB \mid b$

$S \rightarrow abScB$
$B \rightarrow bB \mid b$

قراردادهای نمادگذاری

- نمادهای پایانی
  - حروف کوچک ابتدایی حروف الفبای انگلیسی مانند a,b,c
  - نمادهای عملگر مانند + و −
  - نمادهای علامت گذاری مانند پرانتز، کاما و ...
  - ارقام
  - رشته های بولد شده مانند **id** و **if**
- نمادهای غیرپایانی
  - حروف بزرگ ابتدایی حروف الفبای انگلیسی مانند A, B, C
  - حرف S که معمولا نماد شروع است
  - نام های ایتالیک با حروف کوچک مانند *expr* و *stmt*
- حروف بزرگ انتهایی حروف الفبای انگلیسی مانند X، Y و Z که نمادهای گرامر را نشان می دهند یعنی غیرپایانی هستند.
- حروف کوچک انتهایی حروف الفبای انگلیسی مانند u, v, …, z که رشته ای از پایانی ها را نشان می دهند.

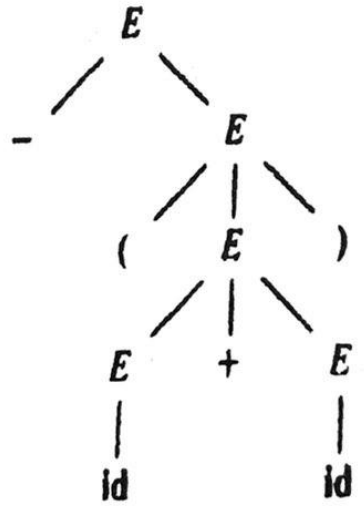
مثال قراردادهای نمادگذاری

$expr \rightarrow expr\ op\ expr$
$expr \rightarrow (expr)$
$expr \rightarrow -expr$
$expr \rightarrow \mathbf{id}$
$op \rightarrow +$
$op \rightarrow -$
$op \rightarrow *$
$op \rightarrow /$
$op \rightarrow \uparrow$

$E \rightarrow EAE \mid (E) \mid -E \mid \mathrm{id}$
$A \rightarrow +\mid -\mid *\mid /\mid \uparrow$

درخت اشتقاق

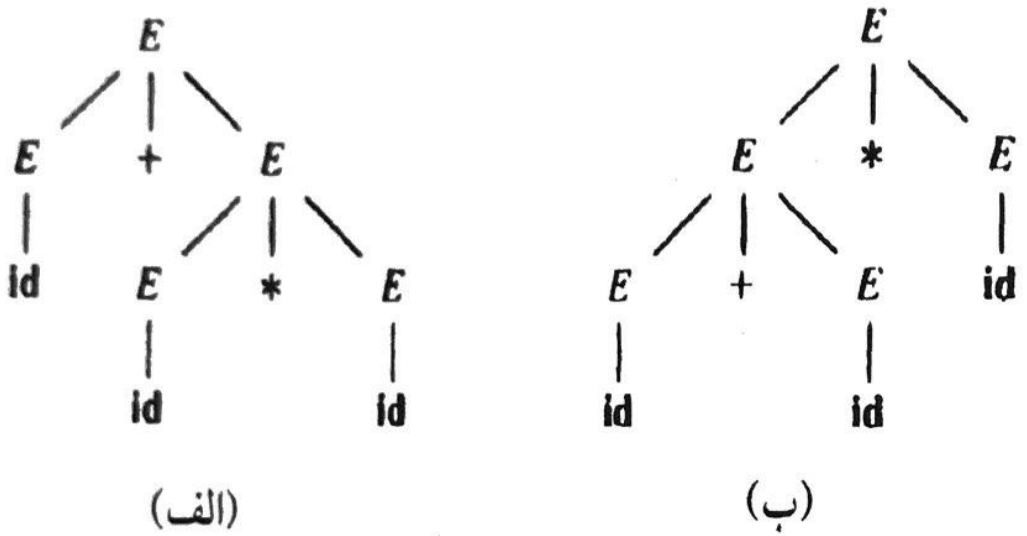
$$E \rightarrow EAE \mid (E) \mid -E \mid \text{id}$$
$$A \rightarrow + \mid - \mid * \mid / \mid \uparrow$$



شکل ۴-۲: درخت تجزیهٔ (id+id)− .

ابهام

$$E \rightarrow EAE \mid (E) \mid -E \mid \text{id}$$
$$A \rightarrow + \mid - \mid * \mid / \mid \uparrow$$



شکل ۴-۴: دو درخت تجزیه برای id+id*id.

۴-۵ عبارات منظم- دلایل استفاده برای نحو زبان

۱- عدم نیاز به نمایش نوع قوی مانند گرامر برای توصیف قوانین ساده لغوی زبان

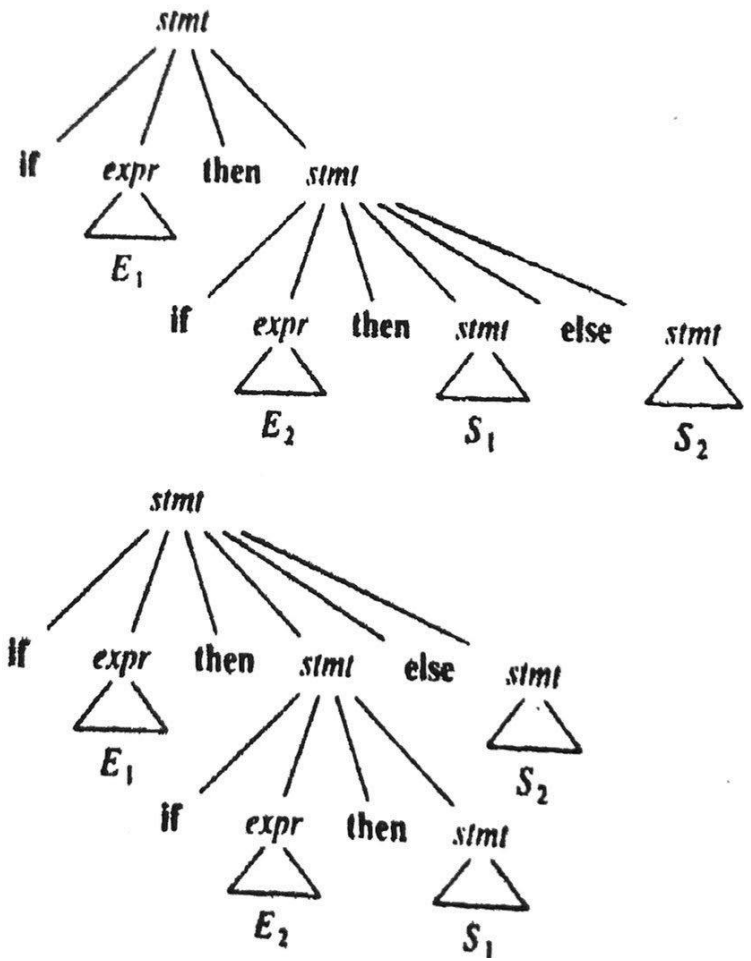
۲- امکان نمایش مختصرتر و قابل فهم تری برای نشانه ها نسبت به گرامر

۳- ایجاد تحلیل گرهای لغوی کاراتر به صورت خودکار

۴- راه مناسبی برای پیمانه سازی جلوبندی کامپایلر به دو بخش قابل مدیریت با تقسیم ساختار زبان به لغوی و غیرلغوی

رفع ابهام

if E_1 **then if** E_2 **then** S_1 **else** S_2



مثال else سرگردان:

$stmt \rightarrow$ **if** $expr$ **then** $stmt$

| **if** $expr$ **then** $stmt$ **else** $stmt$

|**other**

$stmt \rightarrow m_{stmt} | unm_{stmt}$

$m_{stmt} \rightarrow$ **if** $expr$ **then** m_{stmt} **else** m_{stmt}| **other**

$unm_{stmt} \rightarrow$ **if** $expr$ **then** $stmt$

| **if** $expr$ **then**unm_{stmt} **else** unm_{stmt}

حذف بازگشتی چپ مستقیم

- به جای قوانین بازگشتی چپ $A \to A\alpha|\beta$ می توان قوانین غیر بازگشتی چپ زیر را قرار داد.

$$A \to \beta A'$$
$$A' \to \alpha A'|\lambda$$

- مثال: حذف بازگشتی چپ مستقیم از گرامر زیر

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

$$E \to TE'$$
$$E' \to +TE'|\lambda$$
$$T \to FT'$$
$$T' \to * FT'|\lambda$$
$$F \to (E)|\mathbf{id}$$

حذف بازگشتی چپ غیر مستقیم

- در قواعد $A \to A\alpha|\beta$ اگر α معادل λ باشد باعث بازگشتی چپ غیر مستقیم می شود.
- بازگشتی چپی را که شامل اشتقاق های دو یا چند مرحله ای هستند به روش قبل نمی توان حذف کرد.
- مثال بازگشتی چپ غیرمستقیم: در گرامر $S \to Aa|b$ و $A \to Ac|Sd|\lambda$ غیرپایانی S بازگشتی چپ است زیرا $S \Rightarrow Aa \Rightarrow Sda$

الگوریتم ۴-۱ : حذف بازگشتیِ چپ .

ورودی : گرامر G که هیچ حلقه‌ای یا قوانین تولید ε ندارد .

خروجی : یک گرامر معادل بدون بازگشتی چپ .

روش کار : الگوریتم شکل ۴-۷ را روی گرامر G اعمال کنید . توجه داشته باشید که گرامر غیربازگشتی چپ حاصل می‌تواند قوانین تولید ε داشته باشد .

۱- غیرپایانی‌ها را با ترتیب A_1، A_2، ...، A_n مرتب کنید .

```
for i := 1 to n do
    for j := 1 to i - 1 do begin
        replace each production of the form A_i → A_j γ by
            the productions A_i → δ_1 γ | δ_2 γ | ··· | δ_k,
            where A_j → δ_1 | δ_2 | ··· | δ_k are all the
            current A_j-productions;
        eliminate the immediate left recursion among the A_i-productions
    end
```

شکل ۴-۷: الگوریتم حذف بازگشتی چپ از یک گرامر.

مثال:

$S \to Aa|b$

$A \to Ac|Sd|\lambda$

1) i=1

2) i=2

$A \to Ac|Aad|bd|\lambda$

3) حذف بازگشتی چپ

$S \to Aa|b$

$A \to bdA'$

$A' \to cA'|adA'|\lambda$

فاکتورگیری از چپ

- یک تبدیل گرامر است که جهت تولید گرامر مناسب برای تجزیه کننده پیشگو مفید می باشد.
- روشن کردن انتخاب کدام یک از دو قانون تولید هنگامی که نمادهای ورودی با بیش از یک قانون تولید تطابق داشته باشد.
- مثال $stmt \rightarrow \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt | \textbf{if } expr \textbf{ then } stmt$
- اگر $A \rightarrow \alpha\beta_1 | \alpha\beta_2$ با فاکتورگیری چپ خواهیم داشت:

$$A \rightarrow \alpha A'$$
$$A' \rightarrow \beta_1 | \beta_2$$

مثال فاکتورگیری چپ

$$S \to iEtS|iEtSeS|a$$
$$E \longrightarrow b$$

$$S \longrightarrow iEtSS'|a$$
$$S' \to eS|\lambda$$
$$E \to b$$

۶-۴ تجزیه- نوع بالا به پایین

۱- سعی در یافتن سمت چپ ترین اشتقاق برای رشته ورودی دارد.
۲- سعی در ساختن درخت تجزیه برای رشته ورودی با شروع از ریشه و ایجاد گره های درخت بصورت پیش ترتیب

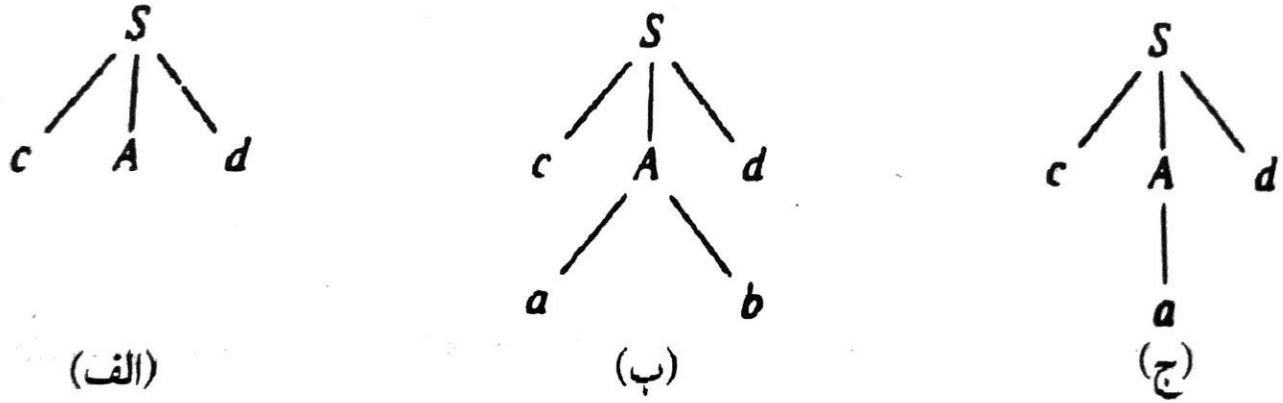
انواع پارسرهای بالا به پایین

- **۱- بدون عقبگرد - تجزیه کننده پیشگو- LL (1)**
- **۲- با عقبگرد- بازگشتی نزولی- در تجزیه زبان طبیعی کاربرد ندارد**

تجزیه کننده های بازگشتی نزولی

مثال: w=cad

$$S \rightarrow cAd$$
$$A \rightarrow ab \mid a$$



شکل ۴-۹: مراحل تجزیهٔ بالا به پائین.

۶-۴ تجزیه- نوع بالا به پایین

تجزیه کننده بازگشتی – کاهشی (پیشگو)

نوشتن دقیق یک گرامر، حذف بازگشتی چپ، فاکتورگیری چپ

$stmt \rightarrow$ **if** $expr$ **then** $stmt$ **else** $stmt$
| **while** $expr$ **do** $stmt$
| **begin** $stmt\_list$ **end**

تهیه مجموعه ای از پایانه هایی که در هر قانون برای یک غیر پایانه در سمت راست ظاهر می شوند.

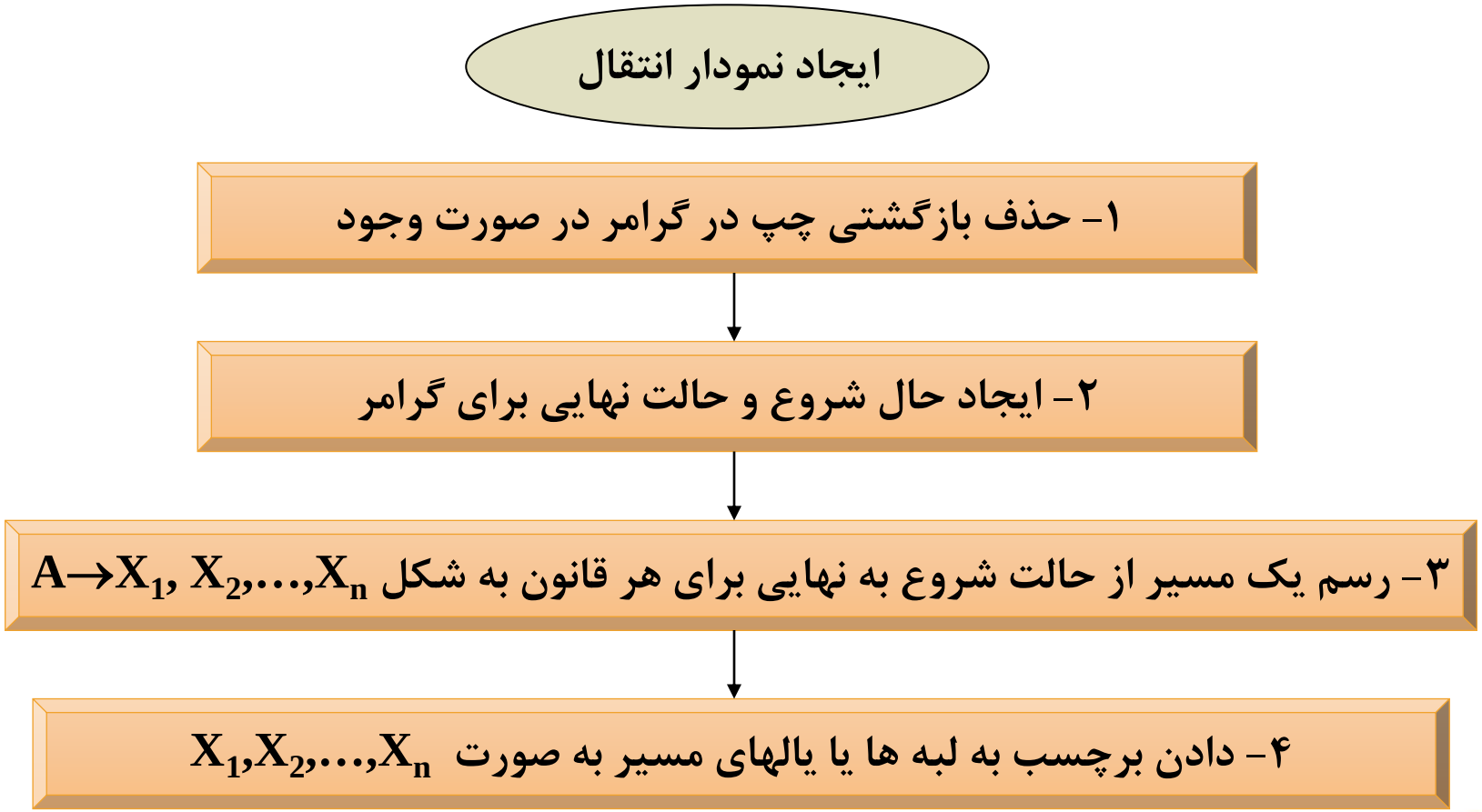
↓

بررسی دنباله رشته ورودی بر طبق مجموعه بالا

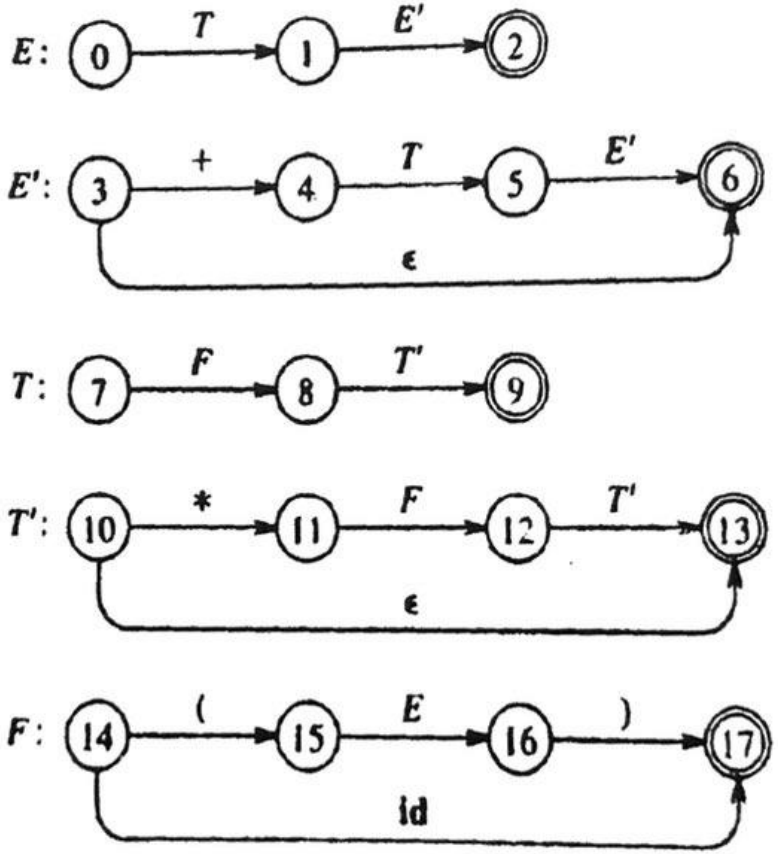
↓

بررسی بالا با مقایسه نماد پیش نگر در ورودی با عناصر مجموعه بالا

۴-۶-۱ تجزیه کننده پیشگو – پیاده سازی



مثال تجزیه کننده پیشگو- نمودار انتقال



شکل ۴-۱۰: نمودارهای تبدیل حالت برای گرامر (۴-۱۱).

$E \rightarrow TE'$

$E' \rightarrow +TE'|\lambda$

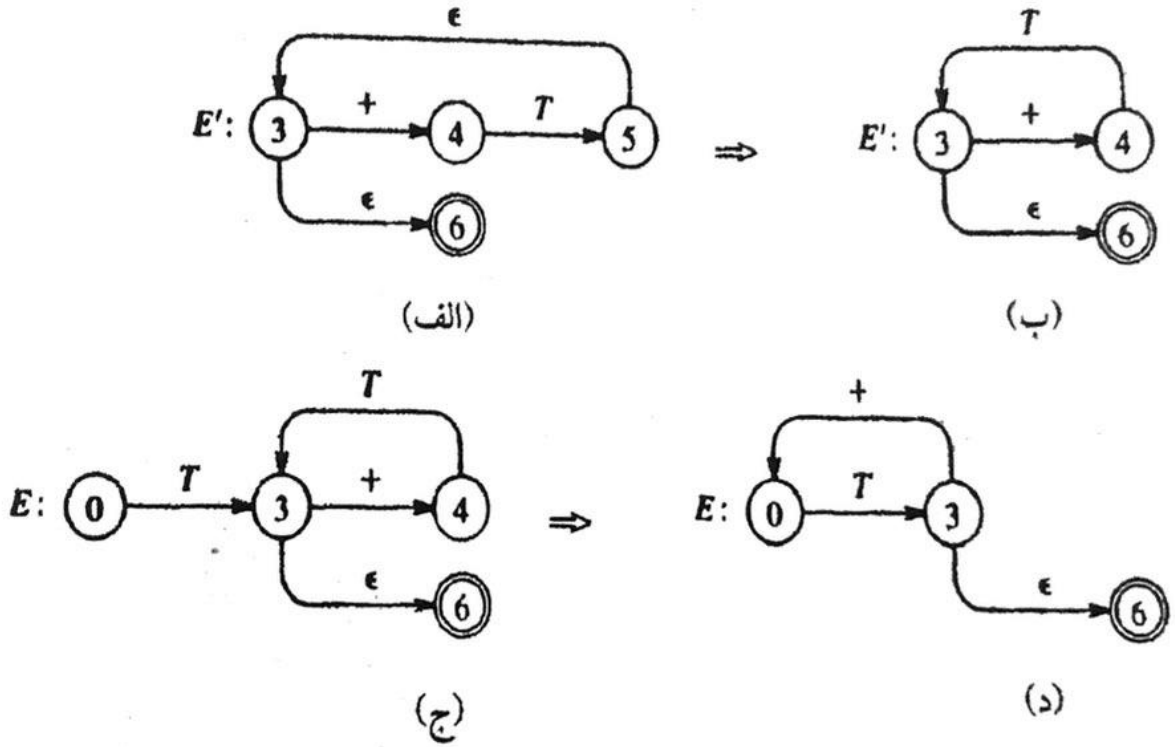
$T \rightarrow FT'$

$T' \rightarrow * FT'|\lambda$

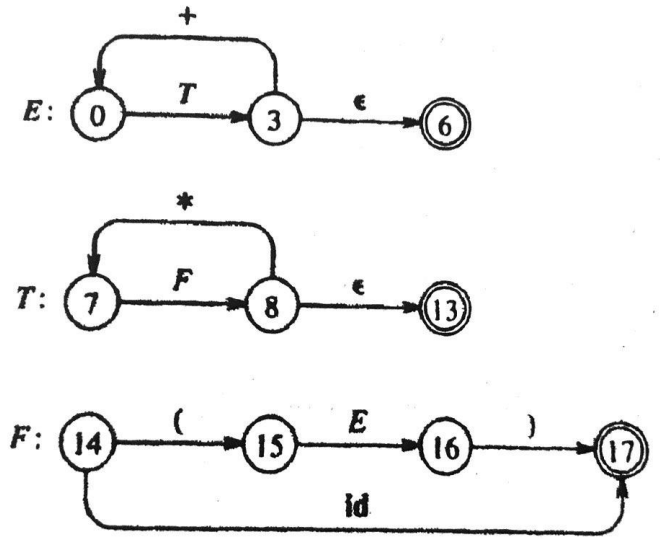
$F \rightarrow (E)|\mathbf{id}$

مثال تجزیه کننده پیشگو- نمودار انتقال

ساده سازی نمودارهای تبدیل حالت با جایگزینی نمودارها در یکدیگر

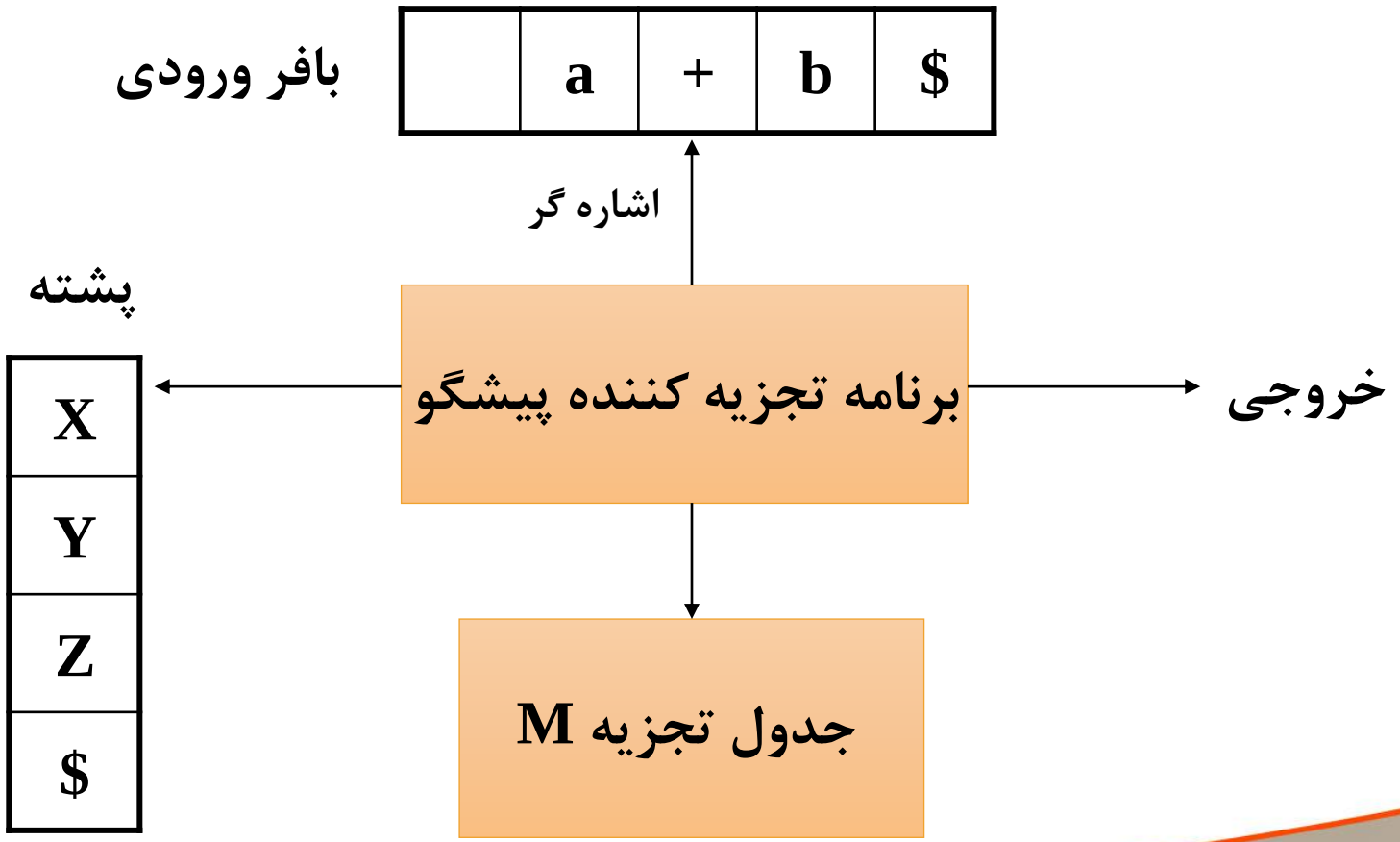


شکل ۴-۱۱: نمودارهای تبدیل حالتِ ساده شده.



شکل ۴-۱۲: نمودارهای تبدیل حالتِ ساده شده برای عبارت‌های محاسباتی.

۴-۶-۲ تجزیه کننده پیشگوی غیر بازگشتی



۴-۶-۲ تجزیه کننده پیشگوی غیر بازگشتی

بخش های یک تجزیه کننده غیر بازگشتی

بافر ورودی — حاوی رشته ورودی برای تجزیه با علامت $ در انتهای آن

پشته — دنباله ای از نمادهای گرامر در هر لحظه برای تجزیه با $ در انتهای آن

جدول تجزیه — آرایه دو بعدی M[A,a] : A غیرپایانی، a پایانی یا نماد $

دنباله خروجی — دنباله تجزیه شده تا آن زمان

۲-۶-۴ تجزیه غیر بازگشتی پیشگو – عملکرد برنامه تجزیه کننده

این برنامه X را به عنوان نماد بالای پشته و a را به عنوان نماد ورودی فعلی در نظر می گیرد. این دو نماد عمل تجزیه کننده را تعیین می کنند. سه حالت پیش می آید:

۱- اگر X= a = $ باشد: توقف تجزیه کننده، اعلام خاتمه موفق تجزیه

۲- اگر X= a ≠ $ باشد: خروج X از پشته، انتقال اشاره گر ورودی به نماد بعدی در ورودی

۳- اگر X غیرپایانی باشد: بررسی درایه M[X,a] از جدول تجزیه M. این درایه یا قانون تولید X از گرامر یا درایه خطا. جایگزینی قانون مناسب آن از جدول یا فراخوانی تابع تصحیح خطا

مثال تجزیه کننده غیر بازگشتی پیشگو

جدول تجزیه M

	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \lambda$	$E' \to \lambda$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \lambda$	$T' \to * FT'$		$T' \to \lambda$	$T' \to \lambda$
F	$F \to \text{id}$			$F \to (E)$		

$E \to TE'$
$E' \to +TE' | \lambda$
$T \to FT'$
$T' \to * FT' | \lambda$
$F \to (E) | \text{id}$

مثال تجزیه کننده غیر بازگشتی پیشگو

Stack	Input	Output
$E	id + id * id$	
$E' T	id + id * id$	E → TE'
$E' T' F	id + id * id$	T → FT'
$E' T' id	id + id * id$	F → id
$E' T'	+id * id$	
$E'	+id * id$	T' → λ
$E' T+	+id * id$	E→ +TE'
$E' T	id * id$	
$E' T' F	id * id$	T → FT'
$E' T' id	id * id$	F → id
$E' T'	*id$	
$E' T' F*	*id$	T' → * FT'
$E' T' F	id$	
$E' T' id	id$	F → id
$E' T'	$	
$E'	$	T' → λ
$	$	E' → λ

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \lambda$	$\boldsymbol{E' \rightarrow \lambda}$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \lambda$	$T' \rightarrow * FT'$		$T' \rightarrow \lambda$	$\boldsymbol{T' \rightarrow \lambda}$
F	$F \rightarrow$ id			$F \rightarrow (E)$		

۷-۴ مجموعه Follow و First

First

اگر α هر رشته ای از نمادهای گرامر باشد **First(α)** مجموعه ای از پایانی ها است که رشته های مشتق شده از α با آن ها شروع می شود. اگر $\alpha \overset{*}{\Rightarrow} \lambda$ آنگاه λ نیز در **First(α)** است.

Follow

برای غیرپایانی **A** به صورت مجموعه ای از پایانی های **a** است که می توانند بلافاصله در سمت راست قانون **A** ظاهر شود.

۴-۷-۱ محاسبه First (A)

۱- اگر X پایانه باشد، First (X) برابر است با {X}

۲- اگر $X \rightarrow \lambda$ قانون گرامر باشد، λ به First (X) اضافه می شود.

۳- اگر X غیرپایانه و $X \rightarrow Y_1Y_2\ldots Y_i$ یک قانون تولید باشد آنگاه a را در صورتی در First (X) قرار دهید که برای یک i، a در First(Yi) باشد و λ در تمام مجموعه های $First(Y_1)\ldots First(Y_{i-1})$ قرار داشته باشد.

۱-۷-۴ محاسبه Follow (A)

۱- $ در Follow (S) قرار داده می شود که در آن S نماد شروع گرامر است و $ نماد پایانی سمت راست رشته ورودی است.

۲- اگر قانونی به صورت $A \rightarrow \alpha B \beta$ وجود دارد آنگاه هر چیزی در First (β) بجز λ به مجموعه Follow (B) اضافه می شود.

۳- اگر قانون تولید $A \rightarrow \alpha\beta$ یا قانون تولید $A \rightarrow \alpha B \beta$ وجود داشته باشند که در آن First (β) حاوی λ است (یعنی $\beta \overset{*}{\Rightarrow} \boldsymbol{\lambda}$) آنگاه هر چیزی در مجموعه Follow(A) به Follow (B) اضافه می شود.

مثال مجموعه های Follow و First

$$E \rightarrow TE'$$
$$E' \rightarrow +TE'|\lambda$$
$$T \rightarrow FT'$$
$$T' \rightarrow * FT'|\lambda$$
$$F \rightarrow (E)|\mathrm{id}$$

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

First (E') = { + , λ }

First (T ') = { * , λ }

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

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

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

۸-۴ ایجاد جدول تجزیه

۱- برای هر قانون از گرامر بصورت $A \rightarrow \alpha$ مراحل ۲ و ۳ را انجام دهید.

۲- برای هر پایانی a در مجموعه First (α)، $A \rightarrow \alpha$ را به M[A , a] اضافه کنید.

۳- اگر λ در First (α) باشد، برای هر پایانه b در Follow (A)، $A \rightarrow a$ را به M[A , b] اضافه کنید. اگر λ در First (α) و $ در Follow (A) باشد، $A \rightarrow a$ به M [A , $] اضافه نمایید.

مثال جدول تجزیه

$E \rightarrow TE'$
$E' \rightarrow +TE'|\lambda$
$T \rightarrow FT'$
$T' \rightarrow * FT'|\lambda$
$F \rightarrow (E)|\text{id}$

First (E) = First (T) = First (F) = { (, id }
First (E') = { + , λ }
First (T ') = { * , λ }

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

	id	+	*	(	)	**$**
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \lambda$	$\boldsymbol{E' \rightarrow \lambda}$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \lambda$	$T' \rightarrow * FT'$		$T' \rightarrow \lambda$	$\boldsymbol{T' \rightarrow \lambda}$
F	$F \rightarrow \text{id}$			$F \rightarrow (E)$		

گرامرهای LL(1)

- الگوریتم قبل را می توان بر هر گرامر G اعمال کرد تا جدول تجزیه M بدست آید. با وجود این، برای برخی از گرامرها، M تعدادی درایه ورودی با تعریف چندگانه دارند.

S→iEtSS' | a

S'→ eS | λ

E→b

	a	b	e	I	t	$
S	S→a			S→iEtss'		
S'			S' →**λ** S' →es			S'→**λ**
E		E→b				

Follow(S')={e,$}

گرامر مبهم

۹-۴ شناسایی گرامر LL(1)

گرامری است که جدول تجزیه اش هیچ درایه ای با تعریف چندگانه ندارد. در LL(1)، L اول به معنی پویش ورودی از سمت چپ، L دوم به معنی تولید سمت چپ ترین اشتقاق و ۱ به معنی استفاده از یک نماد ورودی از پیش نگر در هر مرحله به منظور تصمیم گیری در عمل تجزیه می باشند.

ویژگی های یک گرامر LL(1) عبارتند از:

- ابهام یا بازگشتی چپ ندارد.
- اگر و فقط اگر $A \rightarrow \alpha \mid \beta$ دو قانون تولید متمایز از G باشند آنگاه خواهیم داشت:

۱. برای هیچ پایانی a، هر دوی α و β رشته های شروع شونده با a تولید نمی کنند ($First(\alpha) \cap First(\beta) = \emptyset$).

۲. حداکثر یکی از α و β می توانند رشته تهی (λ) تولید کنند.

۳. اگر $\beta \overset{*}{\Rightarrow} \boldsymbol{\lambda}$، آنگاه α رشته ای شروع شونده با پایانی ای که در Follow (A) است تولید نمی کند ($Follow(A) \cap First(\alpha) = \emptyset$).

۱۰-۴ تصحیح خطا در تجزیه پیشگو

نوع خطا

۱- عدم تطابق پایانی بالای پشته با نماد ورودی بعدی

۲- خالی بودن درایه M[A,a] در جدول تجزیه برای غیر پایانی A در بالای پشته و پایانی a بعنوان نماد ورودی بعدی

تصحیح خطا در panic mode

کنار گذاشتن نمادهای ورودی تا زمان ظاهر شدن نشانه ای متعلق به مجموعه ای از نشانه های هماهنگ کننده

۴-۱۰-۱ انتخاب مجموعه هماهنگ کننده

۱- گذاردن تمام نمادهای Follow(A) در مجموعه هماهنگ کننده غیر پایانی A تا زمان دیدن یک عضو از مجموعه Follow(A). با خروج A از پشته انتظار داریم تجزیه بتواند ادامه یابد.

۲- گذاردن مجموعه نمادهای شروع کننده ساختارهای سطح بالاتر زبان به همراه مجموعه هماهنگ کننده ساختارهای سطوح پایین تر زبان مانند کلمات کلیدی به اضافه مجموعه Follow ها

۳- گذاردن مجموعه First (A) به مجموعه هماهنگ کننده غیر پایانی A

۴- قانون های تولید کننده غیر پایانی λ برای غیر پایانی های قادر به اشتقاق λ

۵- در صورت عدم انطباق پایانی بالای پشته یک ایده ساده این است که آن پایانی را از بالای پشته خارج کنیم و پیغام دهیم که پایانی درج شده است و به تجزیه ادامه دهیم. در نتیجه این روش، مجموعه هماهنگ کننده یک نشانه را شامل تمام نشانه های دیگر در نظر می گیرد.

مثال بروز خطا در تجزیه پیشگو- panic mode

۱- در صورت خالی بودن درایه M[A,a] جدول تجزیه، حذف نماد ورودی a

۲- اگر آن درایه synch باشد (از مجموعه نشانه های هماهنگ کننده حاصل از مجموعه Follow)
خروج غیر پایانه بالای پشته برای امکان ادامه تجزیه

۳- در صورت عدم تطابق نشانه بالای پشته با نماد ورودی، خروج نشانه از پشته

$E \rightarrow TE'$
$E' \rightarrow +TE'|\lambda$
$T \rightarrow FT'$
$T' \rightarrow * FT'|\lambda$
$F \rightarrow (E)|\text{id}$

مثال بروز خطا در تجزیه پیشگو- panic mode

رشته ورودی دارای خطای)id*+id(

مرحله ۱

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \lambda$	$\boldsymbol{E' \rightarrow \lambda}$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \lambda$	$T' \rightarrow * FT'$		$T' \rightarrow \lambda$	$\boldsymbol{T' \rightarrow \lambda}$
F	$F \rightarrow \text{id}$			$F \rightarrow (E)$		

مثال (ادامه)

$E \rightarrow TE'$
$E' \rightarrow +TE'|\lambda$
$T \rightarrow FT'$
$T' \rightarrow * FT'|\lambda$
$F \rightarrow (E)|\text{id}$

مرحله ۲

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

First (E') = { + , λ }

First (T ') = { * , λ }

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

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

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

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \lambda$	$E' \rightarrow \lambda$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \lambda$	$T' \rightarrow * FT'$		$T' \rightarrow \lambda$	$T' \rightarrow \lambda$
F	$F \rightarrow$ id	synch	synch	$F \rightarrow (E)$	synch	synch

مثال (ادامه)

مرحله ۳

STACK	INPUT	REMARK
$E	)id *+ id$	error , skip)
$E	id *+ id$	id is in First (E)
$E`T	id *+ id$	
$E`T`F	id *+ id$	
$E`T`id	id *+ id$	
$E`T`	*+ id$	
$E`T`F*	*+ id$	
$E`T`F	+ id$	Error M [F, +] = synch
$E`T`	+ id$	F has been poped
$E`	+ id$	
$E`T+	+ id$	
$E`T	id$	
$E`T`F	id$	
$E`T`id	id$	
$E`T`	$	
$E`	$	
$	$	

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \lambda$	$E' \rightarrow \lambda$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \lambda$	$T' \rightarrow * FT'$		$T' \rightarrow \lambda$	$T' \rightarrow \lambda$
F	$F \rightarrow$ id	synch	synch	$F \rightarrow (E)$	synch	synch

بروز خطا در تجزیه پیشگو در عبارت- سطح

- پر کردن درایه های خالی در جدول تجزیه پیشگو با اشاره گر هایی به روال های خطا
- روال ها می توانند:
  - تغییر نمادهای ورودی
  - درج یا اضافه
  - ارسال پیغام مناسب
  - خروج از پشته
- تغییر نمادهای پشته یا اضافه کردن نمادهای جدید؟
  - امکان وقوع حلقه بی نهایت
    - بررسی این که هر عمل تصحیح خطا در نهایت باعث استفاده از نماد ورودی می شود

۴-۱۱ تجزیه پایین به بالا– کاهش- انتقال

نمونه ها:
تجزیه عملگر- اولویت
تجزیه LR

سعی در ایجاد درخت تجزیه با شروع از برگها و رفتن به سمت ریشه ⇐ کاهش

جایگزینی یک زیررشته خاص منطبق با سمت راست یک قانون با نماد
سمت چپ همان قانون (سمت راست ترین اشتقاق به صورت عکس ردیابی می شود)

۴-۱۱ تجزیه پایین به بالا– کاهش- انتقال

دنباله ورودی **abbcde**

abbcde	انتقال
aAbcde	کاهش
aAde	انتقال
aABe	کاهش
S	

S → aABe
A → Abc | b
B → d

S → aABe → aAde → aAbcde → abbcde اشتقاق

۴-۱۱-۱ تجزیه انتقال کاهش – دستگیره (Handle)

زیر رشته ای منطبق بر سمت راست یک قانون و ایجاد کننده یک کاهش به غیر پایانه سمت چپ آن قانون
(یک مرحله از عمل معکوس سمت راست ترین اشتقاق)

ویژگی دستگیره

زیر رشته ای که با عمل کاهش با توانایی هدایت تجزیه کننده به عنصر شروع گرامر

مثال دستگیره

دنباله ورودی abbcde

abbcde	انتقال
aAbcde	کاهش
aAde	انتقال
aABe	کاهش
S	

$S \rightarrow aABe$
$A \rightarrow Abc \mid b$
$B \rightarrow d$

دستگیره ها

$A \rightarrow b$

$A \rightarrow Abc$

$B \rightarrow d$

مثال دستگیره

گرامر

(1) E → E + E

(2) E → E * E

(3) E → (E)

(4) E → id

سمت راست ترین اشتقاق

مرحله ۱

E → <u>E + E</u>

→ E + <u>E * E</u>

→ E + E * <u>id3</u>

→ E + <u>id2</u> * id3

→ <u>id1</u> + id2 * id3

مثال دستگیره

دستگیره	جمله
id1	id1 + id2 * id3
id2	E + id2 * id3
id3	E + E * id3
E * E	E + E * E
E+E	E+E
	E

دستگیره ها

مثال دستگیره (گرامر مبهم)

گرامر

(1) E → E + E

(2) E → E * E

(3) E → (E)

(4) E → id

سمت راست ترین اشتقاق

مرحله ۱

E → <u>E * E</u>

→ E * <u>id3</u>

→ <u>E + E</u> * id3

→ E + <u>id2</u> * id3

→ <u>id1</u> + id2 * id3

۴-۱۱-۱-۱ دستگیره- هرس کردن دستگیره ها

مزیت هرس نمودن

توانایی تولید معکوس سمت راست ترین اشتقاق

مراحل هرس نمودن

۱- اگر W جمله گرامر برای تجزیه باشد، آنگاه W = Yn شبه جمله راست n ام از سمت راست ترین اشتقاق نامشخص .
۲- یافتن دستگیره Bn-1 در Yn-1 و کاهش آن تا بدست آمدن شبه جمله راست Yn-2.
۳- توقف عملیات در صورت رسیدن به عنصر شروع گرامر S

مثال هرس نمودن دستگیره

(1) E → E + E

(2) E → E * E

(3) E → (E)

(4) E → id

id1 + id2 * id3

قانون کاهشی	دستگیره	شبه جمله راست
E → id	id1	id1 + id2 * id3
E → id	id2	E + id2 * id3
E → id	id3	E + E * id3
E → E * E	E * E	E + E * E
E → E + E	E + E	E + E
		E

۴-۱۱-۲ مشکلات هرس نمودن دستگیره

۱- تعیین زیر رشته مناسب برای کاهش در یک شبه جمله راست

۲- انتخاب قانون مناسب در موارد وجود دو یا بیشتر قانون با زیر رشته یکسان در سمت راست

۱۲-۴ پیاده سازی تجزیه کاهش- انتقال با پشته

استفاده از پشته به منظور نگهداری نمادهای گرامر
استفاده از بافر ورودی برای ذخیره رشته W ورودی جهت تجزیه

روند تجزیه

۱- انتقال صفر یا چند نماد به پشته توسط تجزیه کننده

۲- ادامه مرحله ۱ تا زمان پیدا شدن یک دستگیره در بالای پشته

۳- کاهش دستگیره پیدا شده به سمت چپ قانون گرامری مناسب آن

۴- پایان کار با پیدا کردن خطا یا حالتی که پشته برابر $S و ورودی برابر $ باشد

۴-۱۲-۱ عملیات کاهش- انتقال با پشته

انتقال ← انتقال نماد بعدی ورودی به بالای پشته

کاهش ← وجود انتهای سمت راست دستگیره در بالای پشته و یافتن سمت چپ آن و تصمیم گیری برای جایگزینی

پذیرش ← اعلام تکمیل موفقیت آمیز عمل تجزیه

خطا ← تشخیص خطای نحوی و فراخوانی رویه پوشش خطا

مثال کاهش- انتقال با پشته

(1) E → E + E

(2) E → E * E

گرامر ←

(3) E → (E)

(4) E → id

E → E + E

→ E + E * E

→ E + E * id3

→ E + id2 * id3

→ id1 + id2 * id3

رشته id1 + id2 * id3 ←

مثال تجزیه کاهش- انتقال با پشته

پشته	ورودی	عمل
$	id1 + id2 * id3 $	Shift
$id1	+ id2 * id3 $	Reduce by E → id
$E	+ id2 * id3 $	Shift
$E +	id2 * id3 $	Shift
$E + id2	* id3 $	Reduce by E → id
$E + E	* id3 $	Shift
$E + E *	id3 4	Shift
$E + E * id3	$	Reduce by E → id
$ E + E * E	$	Reduce by E → E * E
$ E + E	$	Reduce E → E + E
$E	$	accept

۴- ۱۳ تجزیه کاهش- انتقال: پیشوندهای قابل وقوع

مجموعه پیشوندهای شکل های جمله ای راست ظاهر شونده در پشته یک تجزیه کننده کاهش - انتقال

یا

یک شکل جمله ای راست که بعد از انتهای راستِ سمت راست ترین دستگیره آن شکل جمله ای ادامه نَیابد

۱۴-۴ تجزیه کاهش- انتقال: برخوردها

برخورد انتقال - کاهش

تردید در عمل انتقال یا عمل کاهش در زمان تصمیم گیری برای تجزیه کننده

برخورد کاهش- کاهش

وجود چند قانون برای کاهش یک رشته در یک زمان

مثال برخورد کاهش- انتقال

گرامر مبهم $\Leftarrow$ LR نیست

$stmt \rightarrow$ **if** *expr* **then** *stmt*
| **if** *expr* **then** *stmt* **else** *stmt*
| **other**

Stack	Input
… **if** expr **then** stmt	**else** …$

مثال برخورد کاهش- کاهش

(1) *stmt* → **id** (*parameter_list*)
(2) *stmt* → *expr* := *expr*
(3) *parameter_list* → *parameter_list* , *parameter*
(4) *parameter_list* → *parameter*
(5) *parameter* → **id**
(6) *expr* → **id** (*expr_list*)
(7) *expr* → **id**
(8) *expr_list* → *expr_list* , *expr*
(9) *expr_list* → *expr*

Stack	Input
… id (id	**, id) …**

۱۴-۴ تجزیه کننده اولویت- عملگر

- بزرگ ترین کلاس از گرامرها که می توان برای آن ها تجزیه کننده های کاهش- انتقال را با موفقیت ایجاد کرد گرامرهای LR هستند.
- کلاسی کوچک ولی مهم از گرامرها که به آسانی می توان تجزیه کننده های کاهش- انتقال کارآمدی را به صورت دستی ساخت دارای ویژگی های زیر هستند:

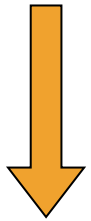
۱. سمت راست هیچ قانون تولیدی، λ نیست.
۲. سمت راست هیچ قانون تولیدی، دو غیر پایانی مجاور نیست.

گرامر نوع عملگر

مثال اولویت- عملگر: گرامر عملگر

$$E \to EAE|(E) - E|id$$
$$A \to +| - | * | /| \uparrow$$

وجود دو غیر پایانه در سمت راست قانون



گرامر عملگر نیست

۴-۱۴-۱ مزایا و معایب روش اولویت- عملگر

- معایب
  - دشوار بودن اداره نمودن نشانه هایی مانند علامت منها با دو اولویت متفاوت (دودیی یا یگانی)
  - عدم اطمینان از نتیجه درست تجزیه به دلیل رابطه نزدیک بین گرامر زبان در حال تجزیه و تجزیه کننده اولویت- عملگر
  - قابلیت تجزیه بر روی تنها کلاس کوچکی از گرامرها
- مزایا
  - پیاده سازی آسان
  - استفاده از روش بازگشتی- کاهشی
  - برای تمام زبان ها ساخته شده اند.

۴-۱۴-۲ روش اولویت- عملگر: تعیین اولویت ها

تعریف ۳ رابطه اولویت مجزای بین هر زوج از پایانی ها

رابطه های اولویت منتهی به انتخاب دستگیره شده و دارای معانی زیر هستند:

رابطه	مفهوم
a .< b	**اولویت a کمتر از b است.**
a ≐ b	**اولویت a و b یکسان است.**
a .> b	**اولویت a بیشتر از b است.**

۴-۱۴-۲ اولویت- عملگر: روش های تعیین اولویت

۱- استفاده از شرکت پذیری و اولویت موجود بین خود عملگرها در زبان مانند اولویت های زیر که ابهام در گرامر را از بین می برد. (علامت منهای یگانی مشکل دارد)

* .> + و + <. *

۲- ایجاد گرامر غیر مبهمی برای زبان که درخت تجزیه آن با قابلیت انعکاس شرکت پذیری و اولویت صحیح بین عناصر پایانی در درخت باشد.

مثال اولویت عملگر– تعیین اولویت ها

$E \rightarrow TE'$
$E' \rightarrow +TE'|\lambda$
$T \rightarrow FT'$
$T' \rightarrow * FT'|\lambda$
$F \rightarrow (E)|\mathrm{id}$

	+	*	(	)	id	$
+	.>	<.	<.	.>	<.	.>
*	.>	.>	<.	.>	<.	.>
(	<.	<.	<.	≐	<.	.>
)	.>	.>	<.	.>		.>
id	.>	.>		.>		.>
$	<.	<.	<.		<.	

۴-۱۴-۳ استفاده از اولویت ها

۱- حذف غیر پایانه ها از جمله ورودی

۲- قرار دادن علامت $ ابتدا و انتهای رشته ورودی به همراه اولویت آن نسبت به اولین پایانه و آخرین پایانه رشته ($<.b و b>.$)

۳- قرار دادن روابط اولویت بین پایانه ها در رشته ورودی به تجزیه

۴- پویش از انتهای چپ رشته تا رسیدن به اولین اولویت <.

۵- پویش به عقب (چپ) از همان نقطه با پشت سرگذاردن هر ≐ تا رسیدن به .> (نهایتا تا $)

۶- تعیین دستگیره شامل هر چیزی در سمت چپ اولین <. و در راست .> در مرحله ۵

مثال استفاده از اولویت ها

رشته ورودی به تجزیه کننده id + id * id

$ <. id .> + <. id .> * <. id .> $.

مرحله ۱: زمان دیدن اولین <. از سمت چپ بین اولین id و +

مرحله ۲: پویش به عقب و ردشدن از روی هر ≐ در صورت وجود برخورد با اولین <.

مثال استفاده از اولویت ها

مرحله ۳: دستگیره بین اولین <. و .> یعنی اولین id سمت چپ و تبدیل آن به E
(E+id*id)

مرحله ۴: تشخیص سایر دستگیره های مشابه { id2 , id3 } و باقیمانده دنباله ورودی به شکل
$ <. * . > + .> $ (شکل جمله ای راست E+E*E)

مرحله ۵: دستگیره بعدی بین + و * و انتهای راست آن بین * و $ یعنی E * E دستگیره و تبدیل به E

مرحله۶: دستگیره بعدی بین + و $ یعنی E + E و تبدیل به E و پایان تجزیه انتقال کاهش – عملگر اولویت

```
(1)  set ip to point to the first symbol of w$;
(2)  repeat forever
(3)      if $ is on top of the stack and ip points to $ then
(4)          return
         else begin
(5)          let a be the topmost terminal symbol on the stack
                 and let b be the symbol pointed to by ip;
(6)          if a <· b or a ≐ b then begin
(7)              push b onto the stack;
(8)              advance ip to the next input symbol;
             end;
(9)          else if a ·> b then          /* reduce */
(10)             repeat
(11)                 pop the stack
(12)             until the top stack terminal is related by <·
                     to the terminal most recently popped
(13)         else error()
         end
```

رابطه های اولویت عملگر با استفاده از شرکت پذیری و اولویت

۱- اگر عملگر A اولویت بیشتر از عملگر B داشته باشد، روابط A . > B و B <. A برقرار است بین آنها

۲- اگر A و B عملگرهای با اولویت یکسان باشند، اگر هر دو شرکت پذیر از راست هستند: A <. B , B <. A و شرکت پذیر از چپ :A .> B , B .> A

↑ <. ↑ , + .> + , + .> - , - .> - , -.> + ,

این روابط تضمین می کنند که در E-E+E به عنوان دستگیره E-E و در E ↑ E ↑ E آخرین دستگیره E ↑ E انتخاب می شوند.

۴-۱۴-۴ اولویت عملگر– اولویتهای بدیهی

۳- برای تمام عملگرهای مانند A روابط زیر برقرار است.

$ <. A A .> $ A .>)) .> A (<. A

A <. (id .> A A <. id

۴- روابط زیر همیشه برقرارند:

(≐) (<. ((<. id

$ <. (id .> $ id .>)

$ <. id) .> $) .>)

مثال رابطه های اولویت عملگر با استفاده از شرکت پذیری و اولویت

فرضیات

۱- ↑ (توان) بالاترین اولویت و شرکت پذیر از راست

۲- * و / بالاترین اولویت بعدی و شرکت پذیر از چپ

۳- + و – پایین ترین اولویت و شرکت پذیر از چپ

مثال عملگر اولویت - جدول اولویت

	+	-	*	/	↑	id	(	)	$
+	.>	.>	<.	<.	<.	<.	<.	.>	.>
-	.>	.>	<.	<.	<.	<.	<.	.>	.>
*	.>	.>	.>	.>	<.	<.	<.	.>	.>
/	.>	.>	.>	.>	<.	<.	<.	.>	.>
↑	.>	.>	.>	.>	<.	<.	<.	.>	.>
id	.>	.>	.>	.>	.>			.>	.>
(	<.	<.	<.	<.	<.	<.	<.	=	
)	.>	.>	.>	.>	.>			.>	.>
$	<.	<.	<.	<.	<.	<.	<.		

فضاهای خالی، درایه های خطا

جدول اولویت ها

جدول را با ورودی id*(id ↑id)-id/id آزمایش و اجرا کنید.

۴-۱۴-۵ اولویت عملگر- توابع اولویت

دو تابع f و g برای کدگذاری جدول اولویت برای پارسر که نمادهای پایانی را به اعداد صحیح تبدیل می کنند.

۱- اگر a <. b آنگاه f(a) <. g (b)

۲- اگر a ≐ b آنگاه f(a) = g (b)

۳- اگر a .> b آنگاه f(a) > g (b)

مثال عملگر اولویت - توابع اولویت

	+	-	*	/	↑	(	)	id	$
f	2	2	4	4	4	0	6	6	0
g	1	1	3	3	5	5	0	5	0

جدول اولویت

*** < . id ⟹ f (*) < g(id)**

id .> id ⟹ f (id) > g (id)

اما در واقع هیچ رابطه اولویتی بین id و id برقرار نیست.

الگوریتم ۴-۶: ساخت توابع اولویت.

ورودی: یک ماتریس اولویت - عملگر.

خروجی: توابع اولویتی که ماتریس ورودی را نشان می‌دهد یا بیان می‌کند چنین ماتریسی وجود ندارد.

روش کار:

۱- برای هر a که یک پایانی یا $ است نمادهای f_a و g_a را ایجاد کنید.

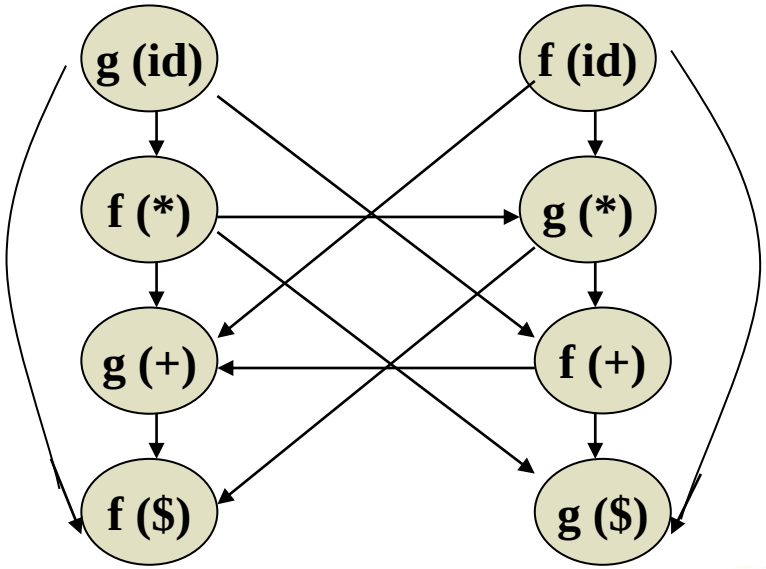
۲- نمادهای ایجاد شده را تا حد ممکن به چند گروه تقسیم بندی کنید به صورتی که اگر $a \doteq b$، آنگاه f_a و g_a در یک گروه قرار گیرند. توجه دارید که ممکن است مجبور شویم نمادها را در یک گروه قرار دهیم حتی اگر با $\doteq$، هیچ رابطه‌ای با هم نداشته باشند. برای مثال اگر $a \doteq b$ و $c \doteq b$ آنگاه f_a و f_c باید در یک گروه باشند زیرا هر دو در همان گروه g_b قرار دارند. علاوه بر این، اگر $c \doteq d$ آنگاه f_a و g_d در یک گروه قرار می‌گیرند، حتی اگر $a \doteq d$ برقرار نباشد.

۳- یک گراف جهت دار رسم کنید که گره‌های آن، گروه‌های به دست آمده در مرحلهٔ (۲) باشند. برای هر a و b، اگر $a \lessdot b$ یک یال از گروه g_b به گروه f_a رسم کنید. توجه داشته باشید که یک یال یا مسیر از f_a به g_b به معنی آن است که f(a) باید بزرگتر از g(b) باشد. یک مسیر از g_b به f_a به معنی آن است که g(b) باید بزرگتر از f(a) باشد.

۴- اگر گراف ساخته شده در مرحلهٔ (۳)، یک دور (حلقه) داشته باشد آنگاه توابع اولویت وجود ندارد. اگر هیچ دوری وجود نداشته باشد آنگاه f(a) را برابر طول بزرگترین مسیری قرار دهید که از گروه f_a شروع می‌شود، g(a) را طول بزرگترین مسیر از گروه g_a قرار دهید.

مثال عملگر اولویت - گراف روابط اولویت

	+	*	id	$
f	2	4	4	0
g	1	3	5	0



هیچ دوری در گراف وجود ندارد پس توابع اولویت وجود دارند.

طولانی ترین مسیر g_+ دارای طول ۱ پس g(+)=1.

طولانی ترین مسیر g_{id} دارای طول ۵ پس g(id)=5.

۴-۱۴-۵ تجزیه عملگر اولویت- پوشش خطا

انواع خطا

۱- اگر هیچ رابطه اولویتی بین پایانی بالای پشته و ورودی فعلی وجود نداشته باشد.

۲- اگر دستگیره ای پیدا شود اما هیچ قانون تولیدی که سمت راست آن، این دستگیره باشد، وجود نداشته باشد.

پوشش خطا

در حالت اول: قرار گرفتن اشاره گرهایی به توابع رفع خطا و فراخوانی آنها هنگام وقوع خطا

۴- ۱۵ تجزیه کننده های LR

دلایل پر طرفداربودن تجزیه کننده های LR

۱- قابلیت تشخیص ساختارهای زبانهای مستقل از متن

۲- عمومی ترین روش تجزیه انتقال کاهش غیر بازگشتی

۳- توانایی تجزیه رده گرامرهای قابل تجزیه پیش گو

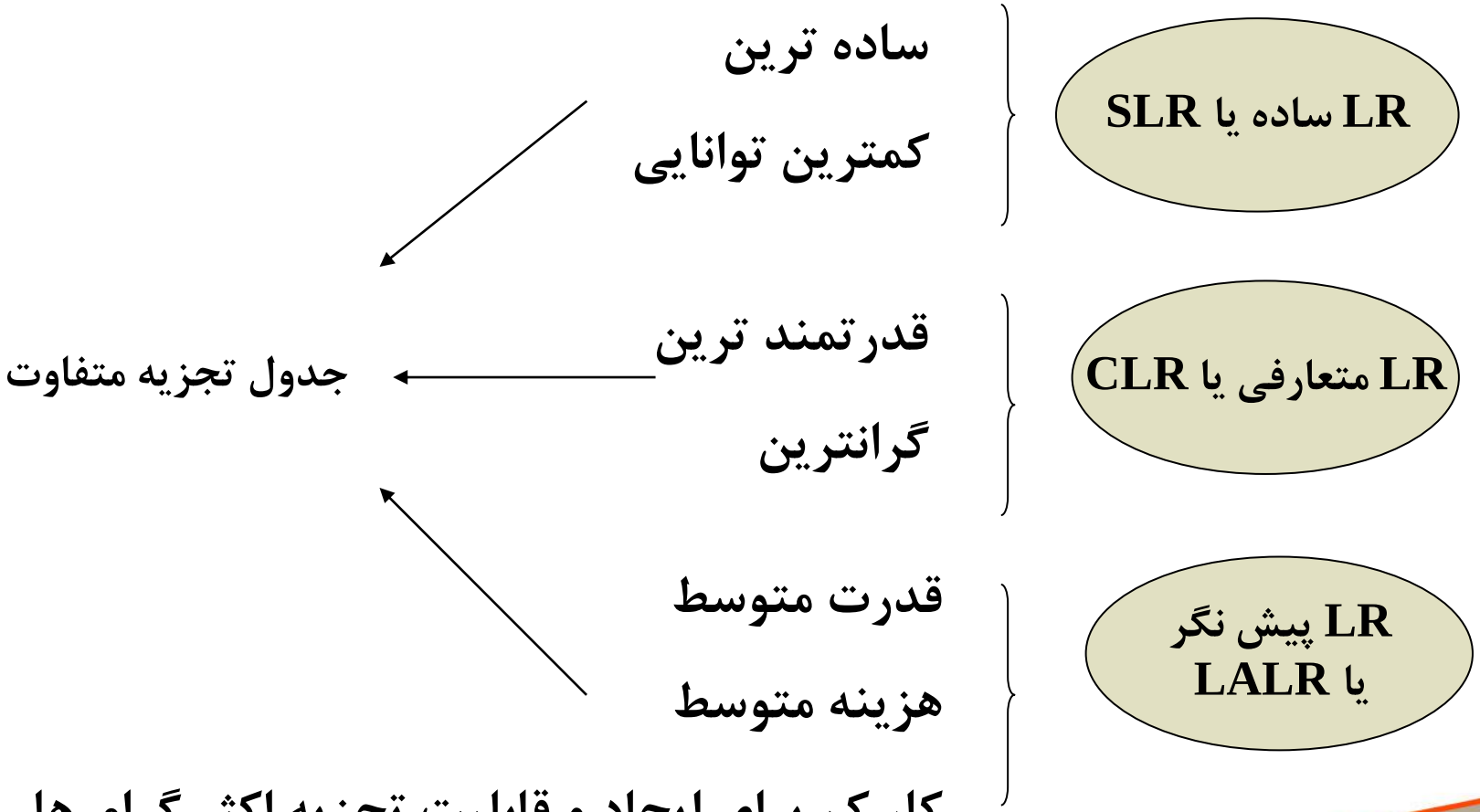
۴- سریعترین تشخیص خطای نحوی با پویش چپ به راست

۴-۱۵-۱ تجزیه LR- نقاط ضعف

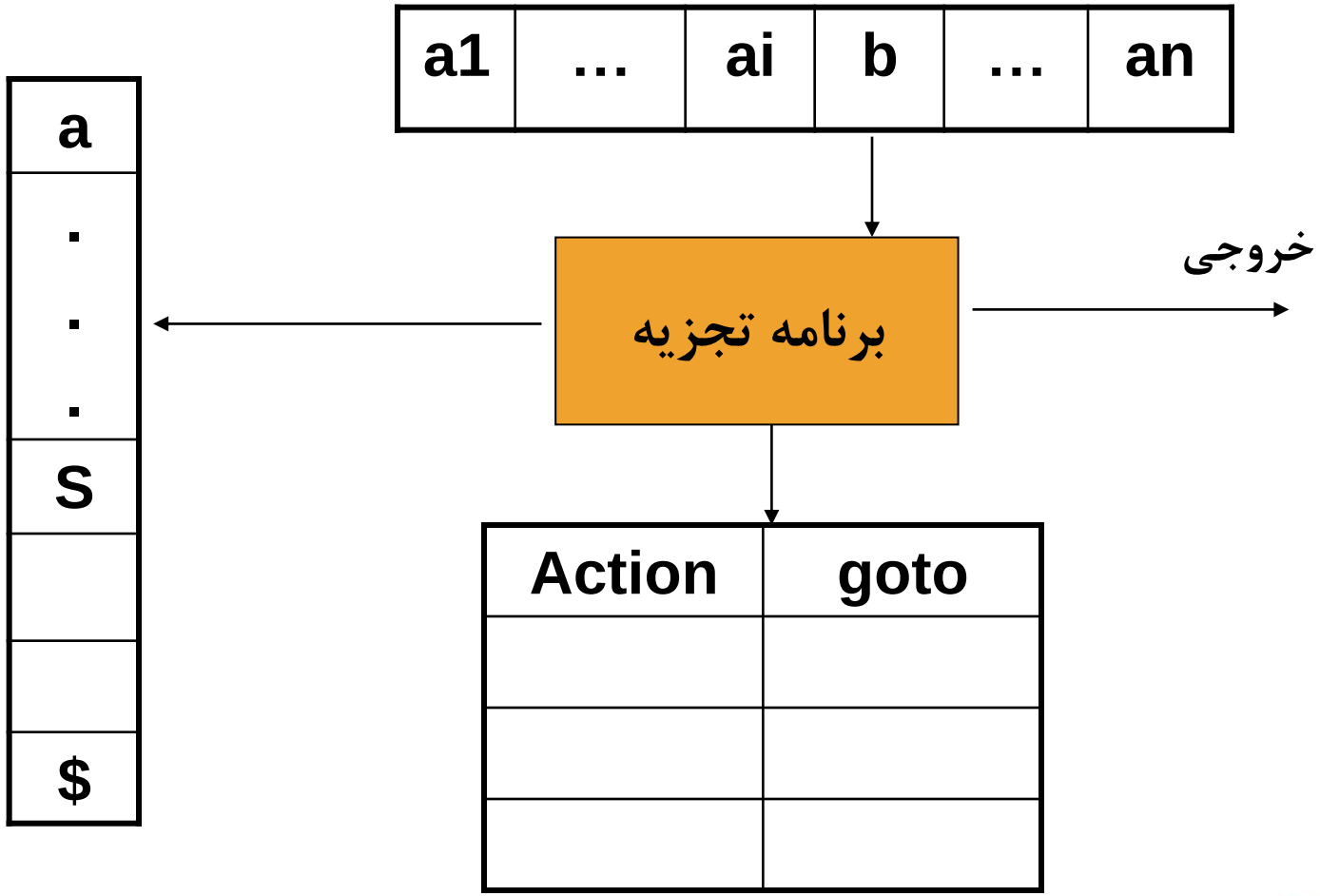
۱-کار زیاد در ساخت آن برای گرامر زبان بشکل دستی

۲- نیازمند ابزار مولد تجزیه کننده LR برای ایجاد

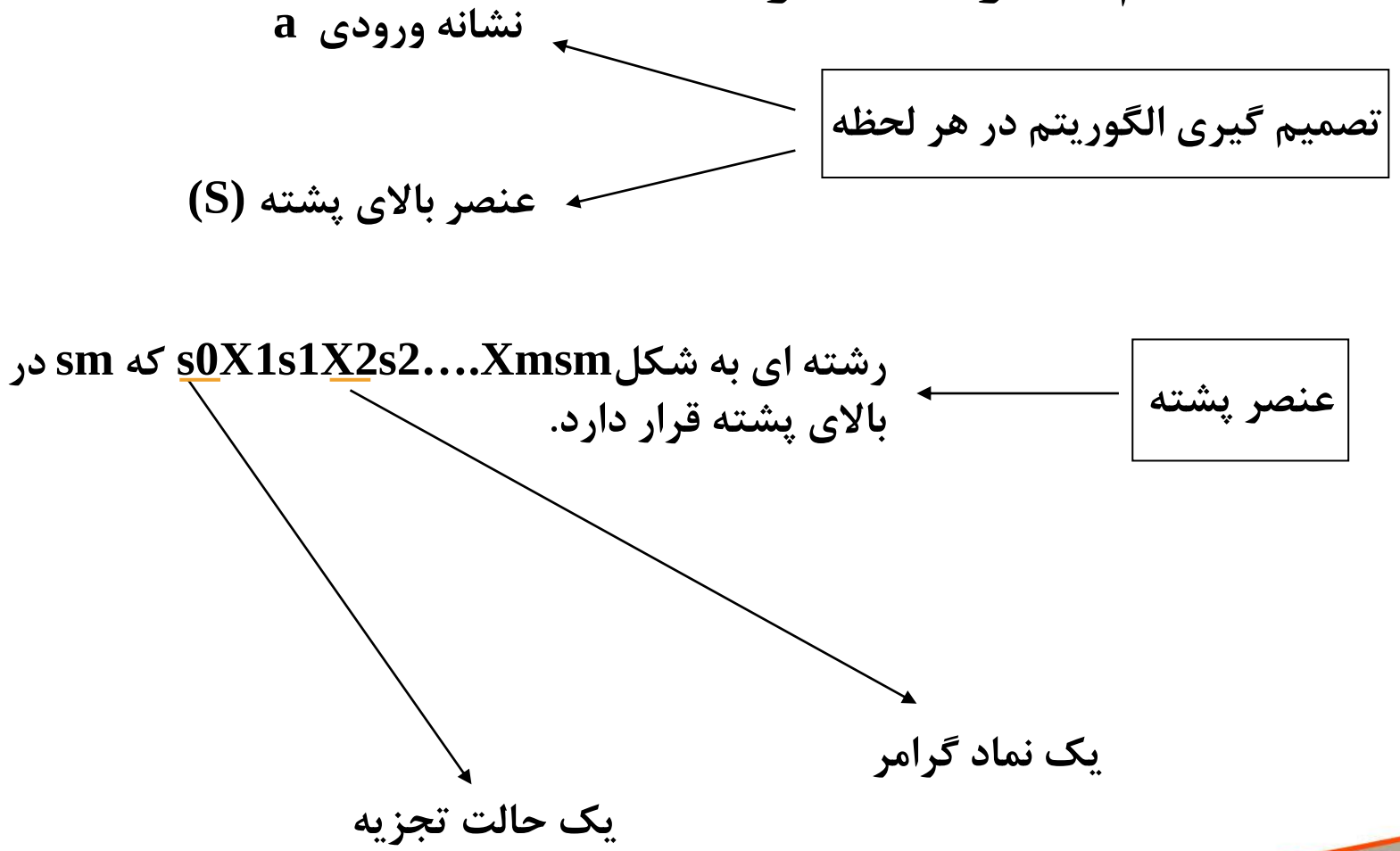
۴-۱۵-۲ تجزیه کننده LR- انواع



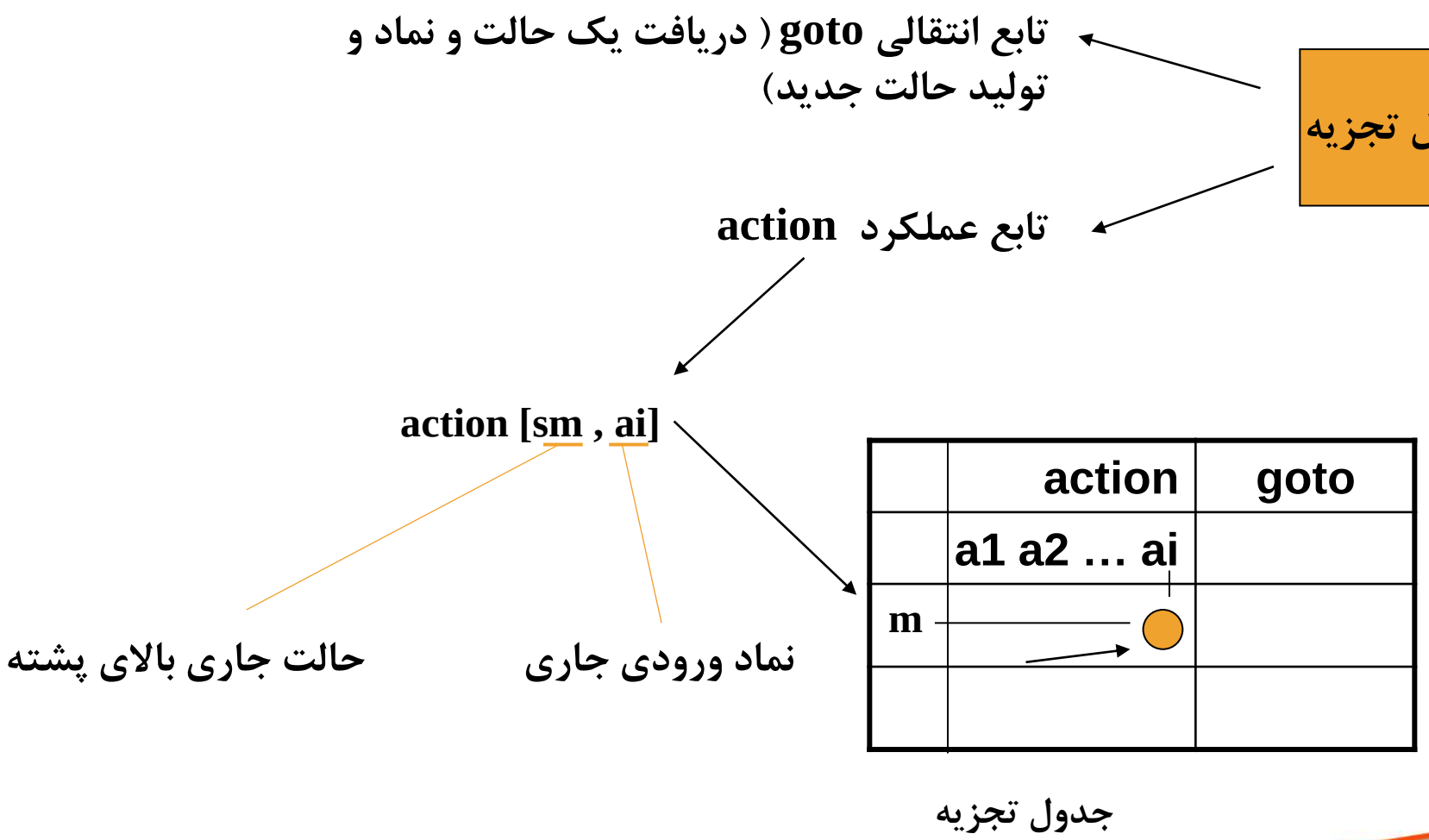
۴-۱۵-۳ تجزیه LR - اجزا



۴-۱۵-۴ تصمیم گیری تجزیه LR

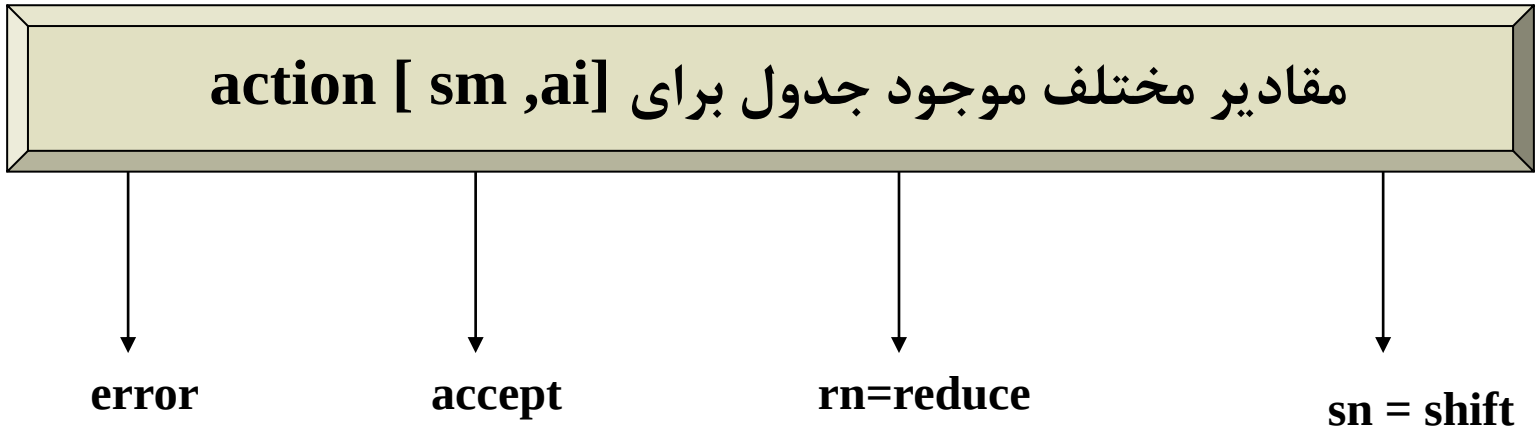


۴-۱۵-۵ تجزیه LR – جدول تجزیه



جدول تجزیه

۴-۱۵-۵ تجزیه LR – جدول تجزیه



خطای نحوی

پایان موفق تجزیه

انجام عمل کاهش با دستور n ام پشته

a روی پشته قرار داده و بعد n به روی پشته می رود

۴-۱۵-۶ تجزیه LR – روال تجزیه

۱- قرار دادن رشته ورودی w به همراه علامت $ در انتهای آن در میانگیر ورودی

۲- گذاردن s0 در پشته به عنوان حالت اولیه

۳- خواندن وارده جدول تجزیه برای [$, s0]

۴- اجرای عمل در نظر گرفته شده در جدول

[shift , reduce, accept , error]

۴-۱۵-۶ تجزیه LR – روال تجزیه

۵- اگر پیکربندی پشته در یک لحظه بصورت:

S0 X1 s1 X2 s2….Xm sm , ai ai+1 …. an $

۶- با خواندن a نماد ورودی جاری و s حالت بالای پشته مراجعه به جدول و انجام یک مورد :

Shift , reduce

۷- تبدیل رشته روی پشته پس از Shift s بصورت :

S0 X1 s1 X2 s2…Xm sm ai s, ai+1…an $

۴-۱۵-۶ تجزیه LR – روال تجزیه

۸- تبدیل رشته روی پشته پس از reduce A بصورت:

S0 X1 s1 X2 s2… Xm-r sm-r A s , ai ai+1… an $

۹- اعلام پایان موفق تجزیه با دیدن accept در جدول

۱۰- فراخوانی رویه پوشش خطا با دیدن error

الگوریتم ۴-۷: الگوریتم تجزیهٔ LR.

ورودی: رشتهٔ ورودی w و جدول تجزیهٔ LR به همراه تابع‌های action و goto برای گرامر G.

خروجی: اگر w در L(G) باشد تجزیهٔ پائین به بالای w، در غیر این صورت، نمایش خطا.

روش کار: در آغاز، تجزیه‌کنندهٔ s_0 را در پشته دارد که در آن، s_0، حالت اولیه و w$ در بافر ورودی است. آنگاه تجزیه‌کنندهٔ برنامهٔ شکل ۴-۳۰ را تا زمانی اجرا می‌کند که عمل accept (پذیرش) یا error (خطا) اتفاق افتاده است.

```
set ip to point to the first symbol of w$;
repeat forever begin
    let s be the state on top of the stack and
        a the symbol pointed to by ip;
    if action[s, a] = shift s' then begin
        push a then s' on top of the stack;
        advance ip to the next input symbol
    end
    else if action[s, a] = reduce A → β then begin
        pop 2*|β| symbols off the stack;
        let s' be the state now on top of the stack;
        push A then goto[s', A] on top of the stack;
        output the production A → β
    end
    else if action[s, a] = accept then
        return
    else error()
end
```

شکل ۴-۳۰: برنامهٔ تجزیهٔ LR.

مثال تجزیه LR

تجزیه id * id + id

	Stack	Input	Action
(1)	0	id * id + id $	s5
(2)	0 id 5	*id + id $	r6 (F→id)
(3)	0 F 3	*id + id $	r4 (T →F)
(4)	0 T 2	*id + id $	s7
(5)	0 T 2 * 7	id + id $	s5
(6)	0 T 2 * 7 id 5	+ id $	r6 (F→id)
(7)	0 T 2 * 7 F 10	+ id $	r3 (T→ T ∗ F)
(8)	0 T 2	+ id $	r2 (E→ T)
(9)	0 E 1	+ id $	s6
(10)	0 E 1 + 6	id $	s5
(11)	0 E 1 + 6 id 5	$	r6 (F→id)
(12)	0 E 1 + 6 F 3	$	r4 (T→F)
(13)	0 E 1 + 6 T 9	$	r1 (E→E+T)
(14)	0 E 1	$	acc

State	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

۷-۱۵-۴ تفاوت گرامر LL و LR

گرامر LL(K)

توانایی تشخیص وقوع سمت راست یک رشته با دیدن تمام آنچه که از آن سمت راست مشتق شده ، با استفاده از K نماد پیش نگر

گرامر LR(K)

توانایی تشخیص وقوع سمت راست تنها با دیدن اولین K نماد از آنچه توسط سمت راست آن مشتق شده

۱۶-۴ تجزیه SLR

تعریف یک قلم

یک قلم از LR(0) قانونی از گرامر با یک نقطه در مکانی در سمت راست آن

مثال اقلام مختلف قانون

$A \rightarrow XYZ$ ⟶

- $A \rightarrow . XYZ$
- $A \rightarrow X .YZ$
- $A \rightarrow XY .Z$
- $A \rightarrow XYZ.$

قانون تولید $A \rightarrow \lambda$ تنها یک قلم $A \rightarrow .$ را تولید می کند.

۱۶-۴ تجزیه SLR تقسیم بندی اقلام

اقلام هسته

شامل قلم اولیه $S' \rightarrow S$ و تمام اقلامی که نقطه آنها در انتهای چپ نیست. هدف از این قانون تولید شروع جدید، نشان دادن زمان به تجزیه کننده است که باید عمل تجزیه را متوقف کند و پذیرفته شدن ورودی را اعلام نماید.

اقلام غیر هسته

اقلامی که نقطه در انتهای چپ است.

۴-۱۶-۱ تجزیه SLR-ایجاد قلم

$S \rightarrow AaAb$
$S \rightarrow BbBa$
$A \rightarrow \varepsilon$
$B \rightarrow \varepsilon$

مرحله۱: تعیین یک نقطه شروع برای گرامر

$S' \rightarrow S$
$S \rightarrow AaAb$
$S \rightarrow BbBa$
$A \rightarrow \varepsilon$
$B \rightarrow \varepsilon$

مرحله۲: گذاردن نقطه در اولین مکان سمت راست قانونها

$S' \rightarrow . S$
$S \rightarrow .AaAb$
$S \rightarrow .BbBa$
$A \rightarrow .$
$B \rightarrow .$

۴-۱۶-۱ تجزیه SLR-ایجاد قلم

مرحله۳: گذاردن نقطه در مکانهای بعدی در سمت راست قانونها

$S' \rightarrow S .$
$S \rightarrow A .aAb$
$S \rightarrow B .bBa$

$S \rightarrow Aa .Ab$
$S \rightarrow Bb .Ba$
$A \rightarrow .$
$B \rightarrow .$

$S \rightarrow AaA .b$
$S \rightarrow BbB .a$

$S \rightarrow AaAb .$
$S \rightarrow BbBa .$

۴-۱۶-۲ تجزیه SLR - گروه اقلام

گروه اقلام LR(0) یا گروه LR(0) متعارف فراهم کننده مبنای ساخت تجزیه کننده های SLR

↓

ساخت به وسیله دو تابع closure و goto به همراه گرامر افزوده

۴-۱۶-۲ تجزیه SLR - گروه اقلام

تابع closure(I)

۱- اضافه شدن هر قلم موجود در مجموعه اقلام I به مجموعه closure

۲- اضافه شدن قلم K. → B درصورت وجودM .BN → A درclosure
و وجود قانون K → B در گرامر

مثال تجزیه SLR - گروه اقلام

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

یک گروه قلم داده شده $I = \{ [E' \rightarrow .E] \}$
آنگاه clouser(I) حاوی اقلام زیر است:

$$\text{Closure (I)} = \begin{cases} E' \rightarrow .E \\ E \rightarrow .E + T \\ E \rightarrow .T \\ T \rightarrow .T * F \\ T \rightarrow .F \\ F \rightarrow .(E) \\ F \rightarrow .id \end{cases}$$

```
function closure ( I );
begin
    J := I;
    repeat
        for each item A → α·Bβ in J and each production
            B → γ of G such that B → ·γ is not in J do
                add B → ·γ to J
    until no more items can be added to J;
    return J
end
```

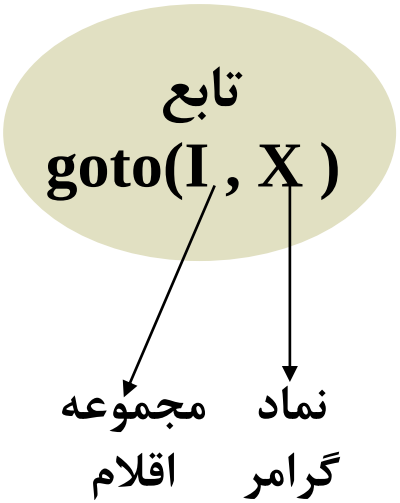
شکل ۴-۳۳: محاسبهٔ closure.

۴-۱۶-۲ تجزیه SLR - عمل goto

مجموعه closure بر روی مجموعه $[A \rightarrow \alpha\ X\ .\beta]$ با شرط وجود $[A \rightarrow \alpha\ .\ X\ \beta]$ در I .

یا

مجموعه ای از اقلام معتبر برای پیشوند قابل وقوع $X\ \gamma$ با وجود مجموعه اقلام معتبر برای γ در I .



مثال تجزیه SLR – عمل goto

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

اگر I مجموعه دو اقلام $I = \{ [E' \rightarrow E.] , [E \rightarrow E. + T] \}$ باشد، آنگاه goto(I, X) از اقلام زیر تشکیل می شود:

$goto(I , +) =$

$E \rightarrow E + . T$

$T \rightarrow .T * F$

$T \rightarrow .F$

$F \rightarrow .(E)$

$F \rightarrow .id$

۴-۱۶-۲ تجزیه SLR - ایجادگروه اقلام LR(0)

۱- گذاردن $C = \{clouser(\{[S' \rightarrow .S]\})\}$ در مجموعه گروه C

۲- برای هر مجموعه اقلام I در C و هر نماد مانند X انجام بده:

۲- ۱- اگر goto(I , X) تهی نیست و در C نیست انجام بده:

۲-۱-۱- goto (I , X) را به C اضافه کن.

عقبگرد، تا زمانی که هیچ مجموعه ای برای اضافه شدن نمانده

```
procedure items(G');
begin
    C := {closure({[S' → ·S]})};
    repeat
        for each set of items I in C and each grammar symbol X
            such that goto(I, X) is not empty and not in C do
                add goto(I, X) to C
    until no more sets of items can be added to C
end
```

شکل ۴-۳۴: ساخت مجموعه‌هایی از اقلام.

مثال تجزیه SLR - ایجادگروه اقلام

E'→ E
E → E + T | T
T → T * F | F
F → (E) | id

I0:
E' → .E
E → .E + T
E → .T
T → .T * F
T → .F
F → .(E)
F → .id

I1:
E' → E.
E → E. + T

I2:
E → T.
T → T. * F

I3:
T → F.

I4:
F → (.E)
E → .E + T
E → .T
T → .T * F
T → .F
F → .(E)
F → .id

I5:
F → id.

I6:
E → E + .T
T → .T * F
T → .F
F → .(E)
F → .id

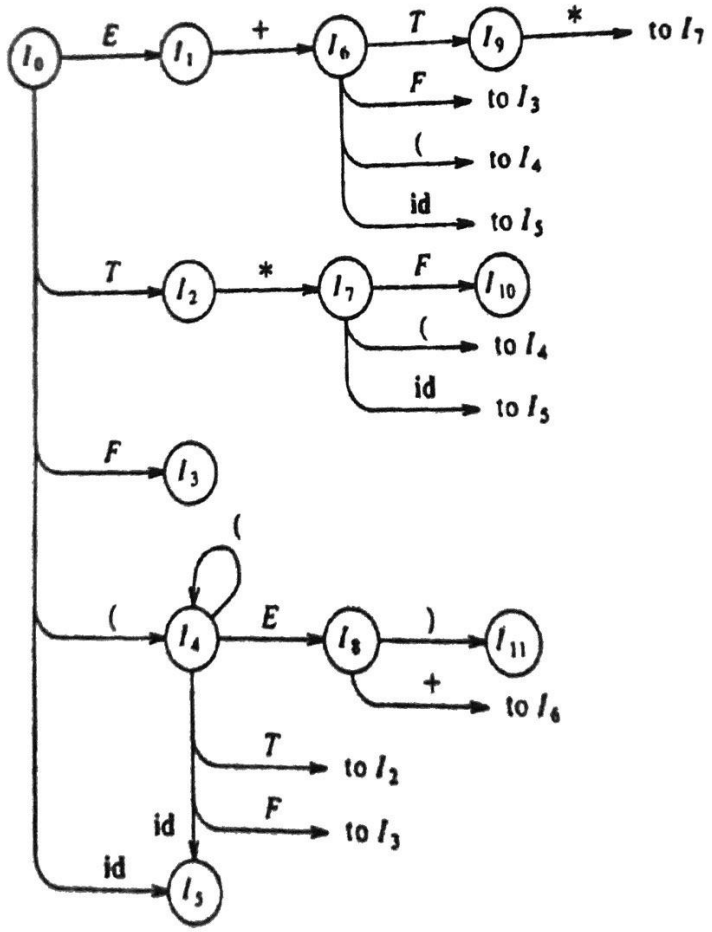
I7:
T → T * .F
F → .(E)
F → .id

I8:
F → (E.)
E → E. + T

I9:
E → E + T.
T → T. * F

I10:
T → T * F.

I11:
F → (E).



۳۶-۴: نمودار تبدیل حالت از یک DFA به نام D برای پیشوندهای قابل وقوع.

۴-۱۶-۲ تجزیه SLR - اقلام معتبر

یک قلم به صورت $A \rightarrow \beta_1 . \beta_2$ برای پیشوند قابل وقوع $\alpha\beta_1$ معتبر است اگر:

اشتقاقی به صورت $S' \overset{*}{\Rightarrow} \alpha A w \overset{*}{\Rightarrow} \alpha\beta_1\beta_2 w$ وجود داشته باشد.

مثال تجزیه SLR - اقلام معتبر

$E' \to E$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid id$

یک پیشوند قابل وقوع ۱ → E + T *

۲ → پس از خواندن آن و رفتن به حالت I7

I7: $T \to T * .F$
$F \to .(E)$
$F \to .id$

اقلام معتبر

$$E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T * F$$

$$E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow E + T * (E)$$

$$E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow E + T * id$$

۴-۱۶-۲ تجزیه SLR - ایجاد جدول تجزیه

الگوریتم ۴-۸: ساخت جدول تجزیهٔ SLR.

ورودی: گرامر افزوده شدهٔ G'.

خروجی: توابع action و goto از جدول تجزیهٔ SLR برای G'.

روش کار:

۱- $C=\{I_0,I_1,...,I_n\}$ یعنی گردایهٔ مجموعه‌های اقلام LR(0) را برای G' بسازید.

۲- حالت i از I_i ساخته می‌شود. action های تجزیهٔ حالت i به صورت زیر تعریف می‌شود:

الف) اگر $[A \rightarrow \alpha.a\beta]$ در I_i و $goto[I_i,a]=I_j$ باشد آنگاه action[i,a] را برابر "shift j" قرار دهید. در اینجا a باید یک پایانی باشد.

ب) اگر $[A \rightarrow \alpha.]$ در I_i باشد آنگاه action[i,a] را برای تمام a در FOLLOW(A) برابر با "reduce $A \rightarrow \alpha$" قرار دهید. در اینجا، A نمی‌تواند S' باشد.

ج) اگر $[S' \rightarrow S.]$ در I_i باشد آنگاه action[i,$] را برابر "accept" قرار دهید.

اگر هر برخوردی در action ها توسط قوانین بالا تولید شود می‌گوئیم این گرامر، SLR(1) نیست. در این حالت، الگوریتم نمی‌تواند یک تجزیه‌کننده را تولید کند.

۳- تبدیل‌های حالت goto برای حالت i، برای تمام غیرپایانی‌های A با استفاده از این قانون ساخته می‌شود: اگر $goto(I_i,A)=I_j$ آنگاه $goto(i,A)=j$.

۴- تمام درایه‌های تعریف نشده با قوانین (2) و (3)، "error" در نظر گرفته می‌شوند.

۵- حالت اولیهٔ تجزیه‌کننده، حالتی است که با استفاده از مجموعه اقلام حاوی $[S' \rightarrow .S]$ ساخته می‌شود.

مثال تجزیه SLR - ایجاد جدول تجزیه

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

I0:
$E' \rightarrow .E$
$E \rightarrow .E + T$
$E \rightarrow .T$
$T \rightarrow .T * F$
$T \rightarrow .F$
$F \rightarrow .(E)$
$F \rightarrow .id$

I1:
$E' \rightarrow E.$
$E \rightarrow E. + T$

I2:
$E \rightarrow T.$
$T \rightarrow T. * F$

I3:
$T \rightarrow F.$

I4:
$F \rightarrow (.E)$
$E \rightarrow .E + T$
$E \rightarrow .T$
$T \rightarrow .T * F$
$T \rightarrow .F$
$F \rightarrow .(E)$
$F \rightarrow .id$

I5:
$F \rightarrow id.$

I6:
$E \rightarrow E + .T$
$T \rightarrow .T * F$
$T \rightarrow .F$
$F \rightarrow .(E)$
$F \rightarrow .id$

I7:
$T \rightarrow T * .F$
$F \rightarrow .(E)$
$F \rightarrow .id$

I8:
$F \rightarrow (E.)$
$E \rightarrow E. + T$

I9:
$E \rightarrow E + T.$
$T \rightarrow T. * F$

I10:
$T \rightarrow T * F.$

I11:
$F \rightarrow (E).$

مثال تجزیه SLR - ایجاد جدول تجزیه

مرحله ۱

I0:

$E' \rightarrow .E$

$E \rightarrow .E + T$

$E \rightarrow .T$

$T \rightarrow .T * F$

$T \rightarrow .F$

⟵ no action

$F \rightarrow .(E)$ ⟵ action [0 , ( ] = shift 4

$F \rightarrow .id$ ⟵ action [0 , id] = shift 5

مرحله ۲

I1:

$E' \rightarrow E.$ ⟵ action [1, $] = accept

$E \rightarrow E. + T$ ⟵ action [1 , +] = shift 6

مرحله ۳

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

I2:

$E \rightarrow T.$ ⟵ action[2, $] = action[2, +]=action[2,)] = reduce E→T

$T \rightarrow T. * F$ ⟵ action [2 , *] = shift 7

ادامه مراحل مطابق نمونه ها.

مثال تجزیه SLR

I_0: $S' \to \cdot S$
$S \to \cdot L = R$
$S \to \cdot R$
$L \to \cdot *R$
$L \to \cdot \mathbf{id}$
$R \to \cdot L$

I_1: $S' \to S \cdot$

I_2: $S \to L \cdot = R$
$R \to L \cdot$

I_3: $S \to R \cdot$

I_4: $L \to * \cdot R$
$R \to \cdot L$
$L \to \cdot *R$
$L \to \cdot \mathbf{id}$

I_5: $L \to \mathbf{id}$

I_6: $S \to L = \cdot R$
$R \to \cdot L$
$L \to \cdot *R$
$L \to \cdot \mathbf{id}$

I_7: $L \to *R \cdot$

I_8: $R \to L \cdot$

I_9: $S \to L = R$

S → L = R
S → R
L → * R
L → id
R → L

$action[2, =] = shift\ 6$

follow(R) = {$, =} $\Rightarrow$ $action[2, =] = reduce\ R \to L$

⇓

برخورد کاهش- انتقال

گرامر مبهم نیست اما SLR(1) نیز نیست.

۱۷-۴ تجزیه CLR یا LR متعارف- تعریف قلم

$[A \rightarrow \alpha.\beta , a]$
قلم LR(1)

شکل کلی یک قلم است که در آن $A \rightarrow \alpha\beta$ یک قانون تولید در گرامر و a یک پایانی یا علامت انتهای سمت راست ($) می باشد.

عملکرد

اعلام کاهش با A در زمان مشاهده a به عنوان نماد ورودی بعدی

۱۷-۴ تجزیه CLR-پیشوند قابل وقوع

شرایط قلم معتبر $[A \rightarrow \alpha.\beta , a]$ برای یک پیشوند قابل وقوع γ

۱- وجود اشتقاق $S \overset{*}{\Rightarrow} \delta Aw \overset{*}{\Rightarrow} \delta\alpha\beta w$ که در آن:

۱- $\gamma = \delta\alpha$ و

۲- a اولین نماد w یا w برابر λ و a برابر $ باشد

۱۷-۴ تجزیه CLR- مثال

$S \rightarrow BB$
$B \rightarrow aB \mid b$

سمت راست ترین اشتقاق $\longrightarrow$ $S \overset{*}{\Rightarrow} aaBab \Rightarrow aaaBab$ $\longrightarrow$

$\delta = aa$
$A = B$
$w = ab$
$\alpha = a$
$\beta = B$

قلم [B→a.B,a] برای پیشوند قابل وقوع γ=aaa مجاز است.

۱۷-۴ تجزیه CLR- ایجاد مجموعه اقلام LR(1)

محاسبه closure

۱- برای هر قلم مثل $[A \rightarrow \alpha.B\beta, a]$ در مجموعه I انجام بده :

برای قانونهای مانند $B \rightarrow \gamma$ در گرامر و هر پایانی مانند b در First (βa):

که قلم $[B \rightarrow .\gamma , b]$ در I نیست
$[B \rightarrow .\gamma , b]$ را به I اضافه کن

برگشت به مرحله ۱
اگر قلمی باقی مانده

الگوریتم ۴-۹: ساختن مجموعه اقلام LR(1).

ورودی: گرامر افزوده شدهٔ G'.

خروجی: مجموعه اقلام LR(1) که برای یک یا چند پیشوند قابل وقوع G'، مجموعه اقلام مجاز هستند.

روش کار: زیربرنامه‌های (تابع‌های) closure و goto و تابع اصلی items برای ساختن مجموعه اقلام، در شکل ۴-۳۸ ارائه شده‌اند.

```
function closure (I);
begin
    repeat
        for each item [A → α·Bβ, a] in I,
            each production B → γ in G',
            and each terminal b in FIRST(βa)
            such that [B → ·γ, b] is not in I do
                add [B → ·γ, b] to I;
    until no more items can be added to I;
    return I
end;
```

۱۷-۴ تجزیه CLR- ایجاد مجموعه اقلام

محاسبه goto

فرض کنید J برابر مجموعه قلم های $[A \rightarrow \alpha X.\beta , a]$ است
که $[A \rightarrow \alpha.X\beta , a]$ در I موجودند.

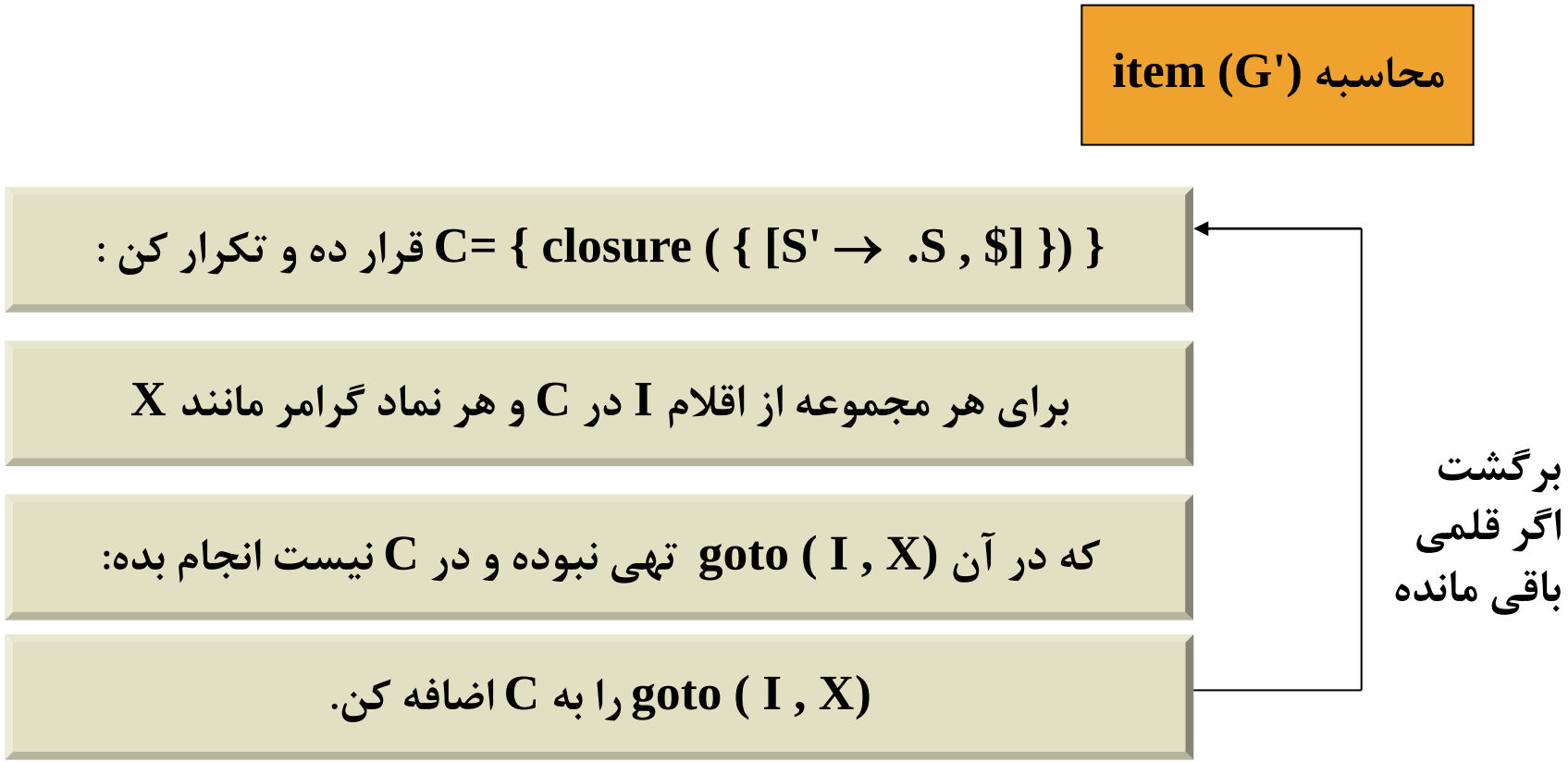
closure (J) را برگردان.

```
begin
      let J be the set of items [A → αX·β, a] such that
            [A → α·Xβ, a] is in I;
      return closure(J)
end;

procedure items(G');
begin
      C := {closure({[S' → ·S, $]})};
      repeat
            for each set of items I in C and each grammar symbol X
                  such that goto(I, X) is not empty and not in C do
                        add goto(I, X) to C
      until no more sets of items can be added to C
end
```

شکل ۴-۳۸: ساخت مجموعه اقلام LR(1) برای گرامر G'.

۱۷-۴ تجزیه CLR- ایجاد مجموعه اقلام



مثال تجزیه CLR- ایجاد مجموعه اقلام

$S' \rightarrow S$
$S \rightarrow CC$
$C \rightarrow cC \mid d$

I0
$S' \rightarrow .S\ ,\ \$$
$S \rightarrow .CC\ ,\ \$$
$C \rightarrow .cC\ ,\ c \mid d$
$C \rightarrow .d\ ,\ c \mid d$

I1
$S' \rightarrow S.\ ,\ \$$

I2
$S \rightarrow C.C\ ,\ \$$
$C \rightarrow .cC\ ,\ \$$
$C \rightarrow .d\ ,\ \$$

I3
$C \rightarrow c.C\ ,\ c \mid d$
$C \rightarrow .cC\ ,\ c \mid d$
$C \rightarrow .d\ ,\ c \mid d$

I4
$C \rightarrow d.\ ,\ c \mid d$

I5
$S \rightarrow CC.\ ,\ \$$

I6
$C \rightarrow c.C\ ,\ \$$
$C \rightarrow .cC\ ,\ \$$
$C \rightarrow .d\ ,\ \$$

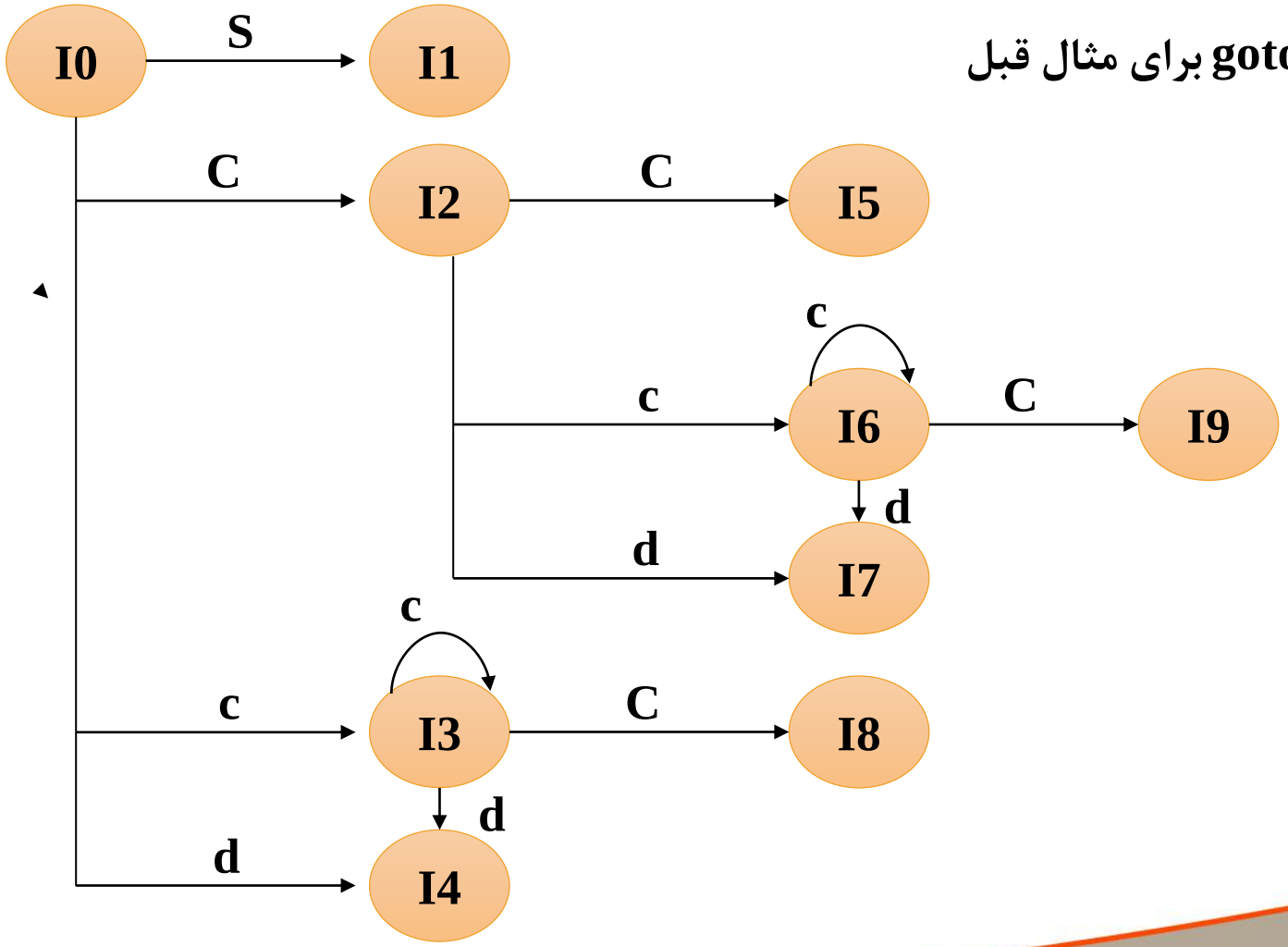
I7
$C \rightarrow d.\ ,\ \$$

I8
$C \rightarrow cC.\ ,\ c \mid d$

I9
$C \rightarrow cC.\ ,\ \$$

مثال تجزیه CLR- ایجاد مجموعه اقلام

گراف goto برای مثال قبل



الگوریتم ۴-۱۰: ساختن جدول تجزیهٔ LR متعارف.

ورودی: گرامر افزوده شدهٔ G'.

خروجی: توابع action و goto از جدول تجزیهٔ LR متعارف برای G'.

روش کار:

۱- $C=\{I_0,I_1,\ldots,I_n\}$ یعنی گردایه‌ای از مجموعه‌هائی از اقلام LR(1) را برای G' بسازید.

۲- با استفاده از I_i، حالت i از تجزیه‌کنندهٔ ساخته می‌شود. action های تجزیه برای i به صورت زیر تعیین می‌شود:

الف) اگر $[A \to \alpha.a\beta,b]$ در I_i و $goto[I_i,a]=I_j$ باشد آنگاه action[i,a] را برابر "shift j" قرار دهید. در اینجا، لازم است a برابر پایانی باشد.

ب) اگر $[A \to \alpha.,a]$ در I_i، $A\neq S'$ باشد آنگاه action[i,a] را برابر "reduce $A \to \alpha$" قرار دهید.

ج) اگر $[S' \to S.,\$]$ در I_i باشد action[i,$] را برابر "accept" قرار دهید.

اگر برخوردی از قوانین بالا به وجود آید در آن صورت، این گرامر، LR(1) نیست، و این الگوریتم، ناموفق است.

۳- تبدیل حالت‌های goto برای حالت i به صورت زیر تعیین می‌شوند: اگر $goto[I_i,A]=I_j$، آنگاه $goto[i,A]=j$.

۴- تمام درایه‌های تعریف نشده با قوانین (۲) و (۳) را برابر "error" قرار دهید.

۵- حالت اولیهٔ تجزیه‌کننده، حالتی است که با استفاده از مجموعه‌ای حاوی قلم $[S' \to .S,\$]$ ساخته می‌شود.

$S \rightarrow CC$
$C \rightarrow cC \mid d$

State	action			goto	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

۴-۱۸ تجزیه LALR- ساخت جدول تجزیه

الگوریتم ۴-۱۱: یک روش ساده ولی با مصرف حافظهٔ زیاد برای ساختن جدول LALR.

ورودی: گرامر افزوده شدهٔ G'.

خروجی: تابع‌های action و goto از جدول تجزیهٔ LALR برای گرامر G'.

روش کار:

۱- $C=\{I_0,I_1,\ldots,I_n\}$ یعنی گردایه‌ای از مجموعه‌هائی از اقلام LR(1) را بسازید.

۲- برای هر core که در میان مجموعه اقلام LR(1) وجود دارد تمام مجموعه‌هایی را به دست آورید که دارای آن core هستند و به جای این مجموعه‌ها، اجتماع آنها را قرار دهید.

۳- فرض کنید $C'=\{J_0,J_1,\ldots,J_m\}$ مجموعه حاصل از اقلام LR(1) باشد. عملیات (action) تجزیهٔ مربوط حالت i با استفاده از J_i مانند الگوریتم ۴-۱۰ ساخته می‌شود. اگر برخوردی در action تجزیه وجود داشته باشد این الگوریتم، تجزیه کنندهٔ را نمی‌تواند تولید کند و این گرامر، LALR(1) نامیده نمی‌شود.

۴- جدول goto به صورت زیر ساخته می‌شود. اگر J اجتماع یک یا چند مجموعه از اقلام LR(1) باشد یعنی اگر $J=I_1 \cup I_2 \cup \ldots \cup I_k$ باشد آنگاه core های $goto(I_1,X)$، $goto(I_2,X)$،...، $goto(I_k,X)$ یکسان هستند زیرا $I_1, I_2 \ldots I_k$ همگی دارای یک core هستند. فرض کنید اجتماع تمام مجموعه‌هایی از اقلام دارای core یکسانی مانند $goto(I_1,X)$ باشند. آنگاه $goto(J,X)=K$ است.

مثال تجزیه LALR- ساخت جدول تجزیه

$S' \rightarrow S$
$S \rightarrow CC$
$C \rightarrow cC \mid d$

I0
$S' \rightarrow .S\ ,\ \$$
$S \rightarrow .CC\ ,\ \$$
$C \rightarrow .cC\ ,\ c \mid d$
$C \rightarrow .d\ ,\ c \mid d$

I1
$S' \rightarrow S.\ ,\ \$$

I2
$S \rightarrow C.C\ ,\ \$$
$C \rightarrow .cC\ ,\ \$$
$C \rightarrow .d\ ,\ \$$

I3
$C \rightarrow c.C\ ,\ c \mid d$
$C \rightarrow .cC\ ,\ c \mid d$
$C \rightarrow .d\ ,\ c \mid d$

I4
$C \rightarrow d.\ ,\ c \mid d$

I5
$S \rightarrow CC.\ ,\ \$$

I6
$C \rightarrow c.C\ ,\ \$$
$C \rightarrow .cC\ ,\ \$$
$C \rightarrow .d\ ,\ \$$

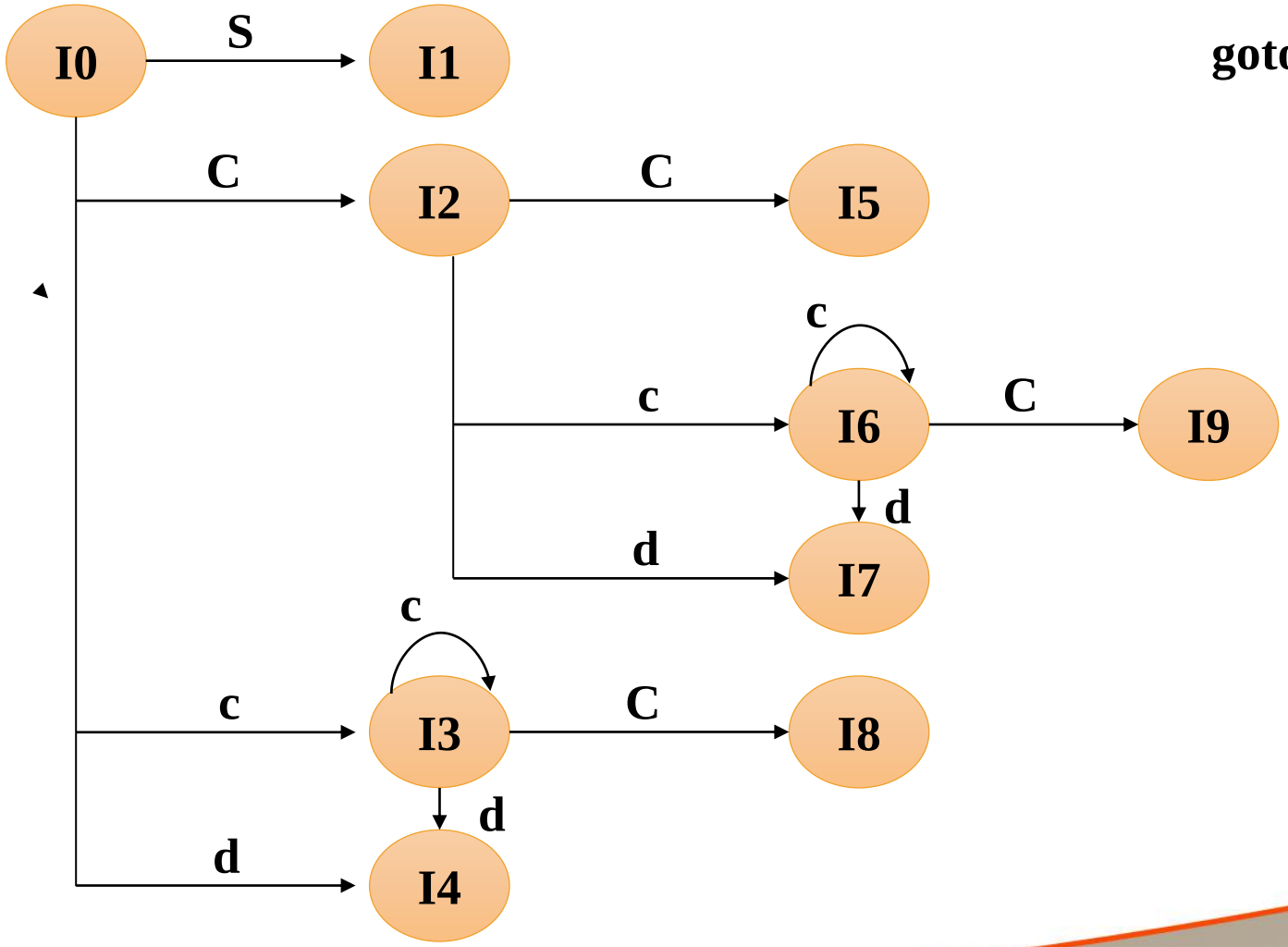
I7
$C \rightarrow d.\ ,\ \$$

I8
$C \rightarrow cC.\ ,\ c \mid d$

I9
$C \rightarrow cC.\ ,\ \$$

مثال تجزیه LALR- ساخت جدول تجزیه

گراف goto



مثال تجزیه LALR- ساخت جدول تجزیه

1 S'→ S
2 S → CC
3 C → cC | d

ادغام مجموعه های اقلام و جایگزین شدن با اجتماعشان

I3 , I6

C → c.C , c | d | $

C → .cC, c | d | $

C → .d , c | d | $

I4 , I7

C → d. , c | d | $

I8 , I9

C → cC. , c | d | $

مثال تجزیه LALR- ساخت جدول تجزیه

$S \rightarrow CC$
$C \rightarrow cC \mid d$

State	action			goto	
	c	d	$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

مثال پوشش خطا در تجزیه LR و LALR

$S \rightarrow CC$
$C \rightarrow cC \mid d$

LR

State	action			goto	
	c	d	$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

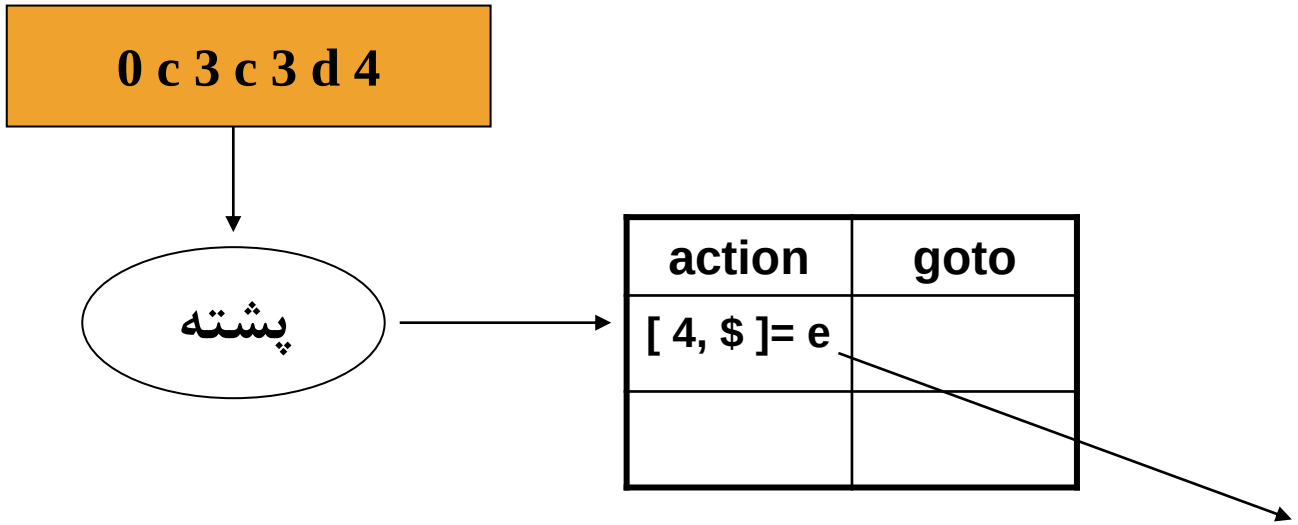
LALR

State	action			goto	
	c	d	$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

مثال پوشش خطا در تجزیه LR و LALR

برخورد با خطا در تجزیه LR

ورودی دارای خطای $ccd



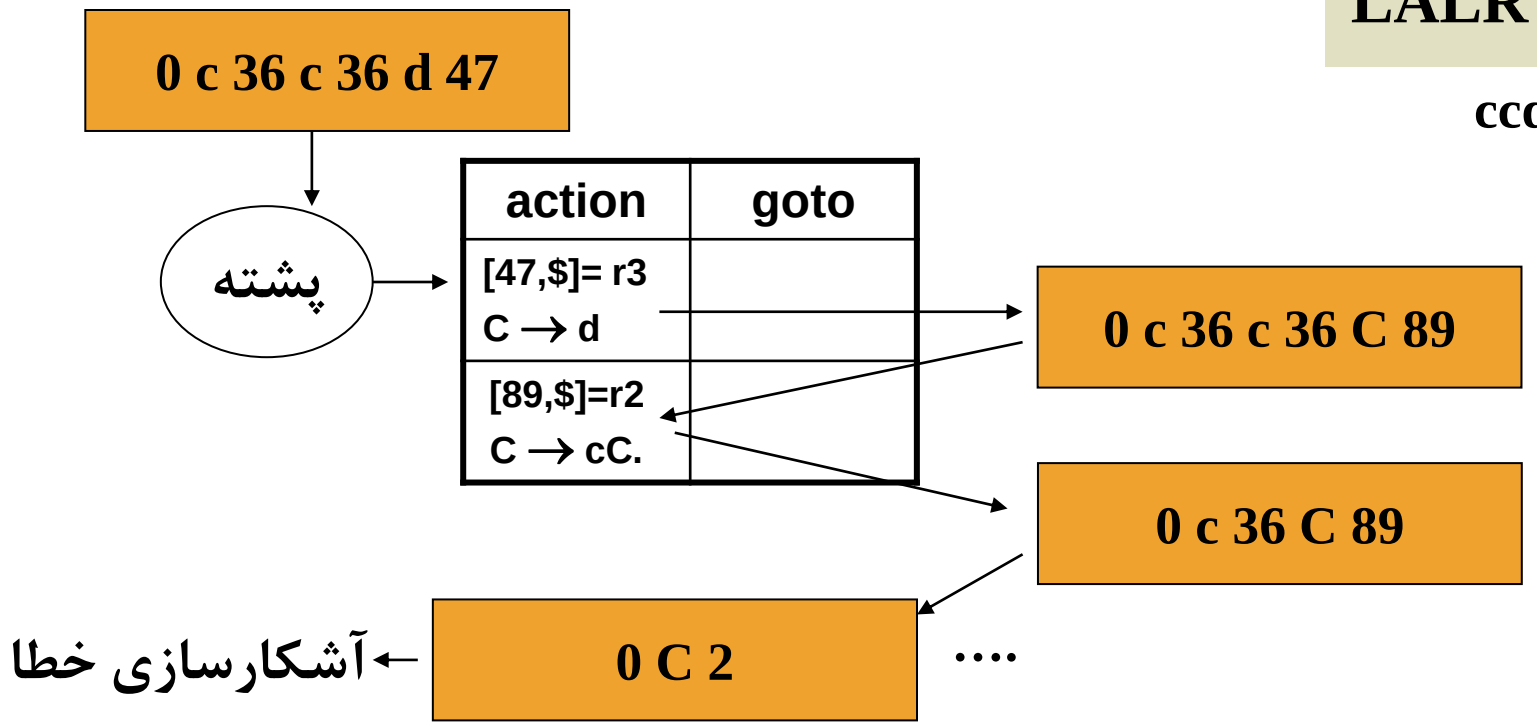
آشکارسازی خطا

تشخیص خطا در یک مرحله

مثال پوشش خطا در تجزیه LR و LALR

برخورد با خطا در تجزیه LALR

ورودی دارای خطای $ccd



تشخیص خطا پس از چند کاهش

۴-۱۸ تجزیه LALR- تعیین پیش نگرها

الگوریتم ۴-۱۲ : تعیین پیش‌نگرها.

ورودی : هستهٔ k از مجموعه اقلام LR(0) در I و نماد گرامر X.

خروجی : پیش‌نگرهای خود به خود تولید شده توسط اقلام موجود در اقلام هستهٔ goto(I,X) و اقلام اکه با استفاده از آنها ، پیش‌نگرها به اقلام هسته در goto(I,X) انتشار یافته‌اند.

روش کار : این الگوریتم در شکل ۴-۴۳ داده شده است. این الگوریتم از نماد پیش‌نگر ساختگی # برای کشف موقعیت‌هایی استفاده می‌کند که در آن ، پیش‌نگرها انتشار یافته‌اند.

```
for each item B → γ·δ in K do begin
    J' := closure({[B → γ·δ, #]});
    if [A → α·Xβ, a] is in J' where a is not # then
        lookahead a is generated spontaneously for item
            A → αX·β in goto(I, X);
    if [A → α·Xβ, #] is in J' then
        lookaheads propagate from B → γ·δ in I to
            A → αX·β in goto(I, X)
end
```

شکل ۴-۴۳: تعیین پیش‌نگرهای انتشار یافته و خود به خودی.

۱۸-۴ تجزیه LALR- ساخت جدول تجزیه بهینه

چند اصلاح در الگوریتم ساخت جدول تجزیه

الگوریتم ۴-۱۳ : محاسبهٔ کارای هسته‌های گردایهٔ LALR(1) از مجموعه هایی از اقلام.

ورودی : گرامر افزوده شدهٔ G′.

خروجی : هسته‌های گردایهٔ LALR(1) از مجموعه‌هایی از اقلام.

روش کار :

۱- با استفاده از روش بالا، هسته‌های مجموعه‌هایی از اقلام LR(0) را برای G ایجاد کنید.

۲- الگوریتم ۴-۱۲ را بر هستهٔ هر مجموعه از اقلام LR(0) و نماد گرامر X اجرا کنید تا مشخص کند کدام یک از پیش‌نگرها، خود به خود، برای اقلام هستهٔ موجود در goto(I,X) تولید می‌شوند، و کدام یک از اقلام در I، پیش‌نگرها به اقلام هسته در goto(I,X) انتشار یافته‌اند.

۳- جدولی ایجاد کنید که برای هر قلم از هسته در هر مجموعه از اقلام، پیش‌نگرهای مربوطه‌اش را مشخص می‌کند. در آغاز، به هر قلم، آن دسته از پیش‌نگرهایی مربوط می‌شود که در (۲)، خود به خود تولید شده‌اند.

۴- در تمام مجموعه‌ها، بر روی اقلام هسته، گذرهای مکرر انجام دهید. هنگامی که قلم A → α·, a را ملاقات می‌کنید اقلام هسته را جستجو کنید که با آن، a با استفاده از اطلاعات موجود در (۲)، پیش‌نگرهای خود را منتشر می‌کند. مجموعهٔ جاری از پیش‌نگرهای مربوط به A → α· به پیش‌نگرهایی اضافه می‌شوند که قبل از این، هر یک با اقلامی همراه بودند که با آن، a پیش‌نگرهای خود را منتشر می‌کند. ایجاد گذرها بر روی اقلام هسته را تا آنجا ادامه دهید که پیش‌نگرهای جدید دیگری برای انتشار وجود نداشته باشد.

مثال LALR- محاسبه هسته های گروه اقلام

$S' \rightarrow S$
$S \rightarrow L=R \mid R$
$L \rightarrow * R \mid id$
$R \rightarrow L$

مرحله ۱ : بدست آوردن اقلام LR

I0: $S' \rightarrow .S$

I1: $S' \rightarrow S.$

I2: $S \rightarrow L.=R$

$R \rightarrow L.$

I3: $S \rightarrow R.$

I4: $L \rightarrow *.R$

I5: $L \rightarrow id.$

I6: $S \rightarrow L=.R$

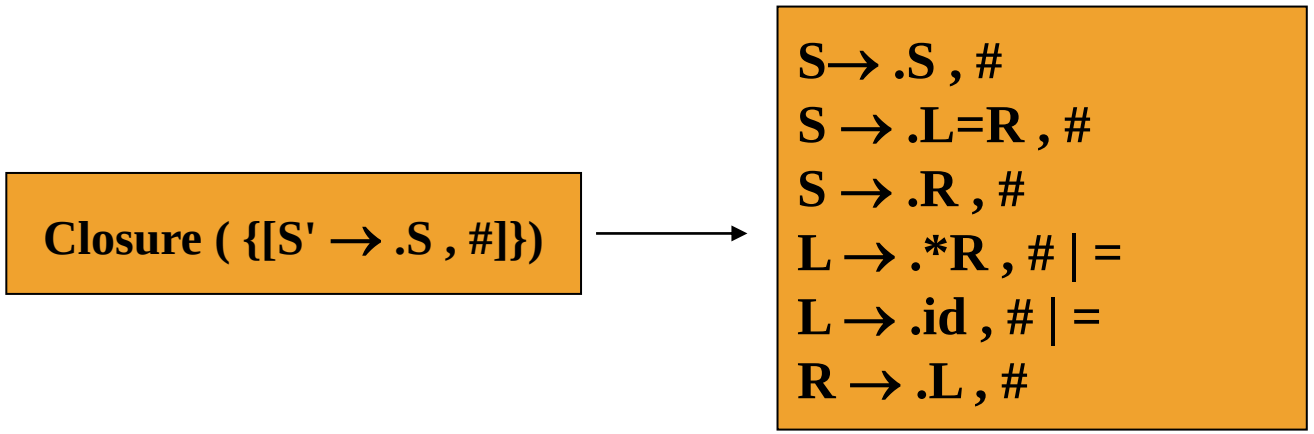
I7: $L \rightarrow *R.$

I8: $R \rightarrow L.$

I9: $S \rightarrow L=R.$

مثال LALR- محاسبه هسته های گروه اقلام

مرحله ۲ : اجرای الگوریتم محاسبه پیش نگرها



مثال LALR- محاسبه هسته های گروه اقلام

مرحله ۳ : انتشار پیش نگرها در بین اقلام هسته

از	به
I0: S' → .S	I1: S' → S. I2: R → L.=R R → L. I3: S → R. I4: L → *.R I5: L → id.
I2: R → L.=R	I6: S → L=.R

مثال LALR- محاسبه هسته های گروه اقلام

از	به
I4: L → *.R	I4: L → *.R I5: L → id. I7: L → *R. I8: R → L.
I6: S → L=.R	I4: L → *.R I5: L → id. I8: R → L. I9: S → L=R.

مثال LALR- محاسبه هسته های گروه اقلام

مرحله ۴ : مقداردهی جدول پیش نگرها و انجام گذرها

قلم مجموعه		پیش نگرها			
		INIT	PASS1	PASS2	PASS3
I0:	S' → .S	$	$	$	$
I1:	S' → S.		$	$	$
I2:	R → L.=R		$	$	$
I2:	R → L.		$	$	$
I3:	S → R.		$	$	$
I4:	L → *.R	=	=|$	=|$	=|$
I5:	L → id.	=	=|$	=|$	=|$
I6:	S → L=.R			$	$
I7:	L → *R.		=	=|$	=|$
I8:	R → L.		=	=|$	=|$
I9:	S → L=R.				$