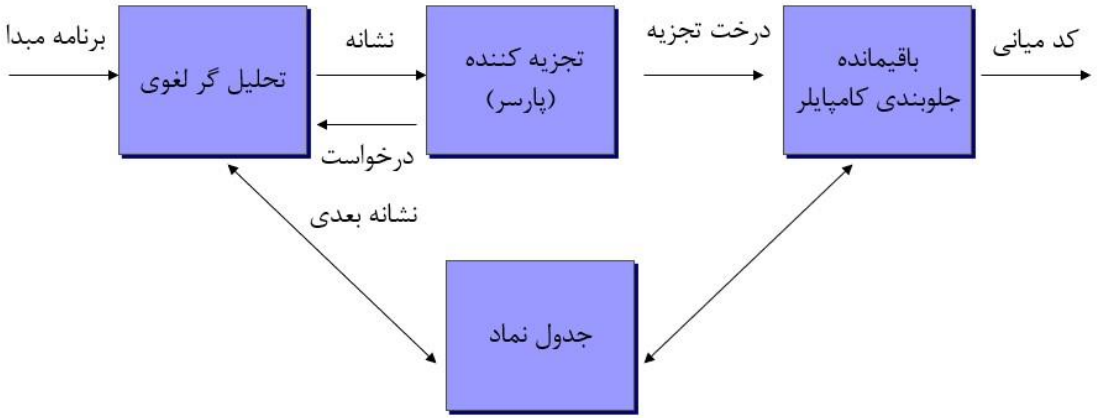
# تجزیه کننده

| |
|---|
| 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 |

## تجزیه کننده– ارتباطات




موقعیت تجزیه کننده در مدل کامپایلر

# تجزیه ( پارسینگ)

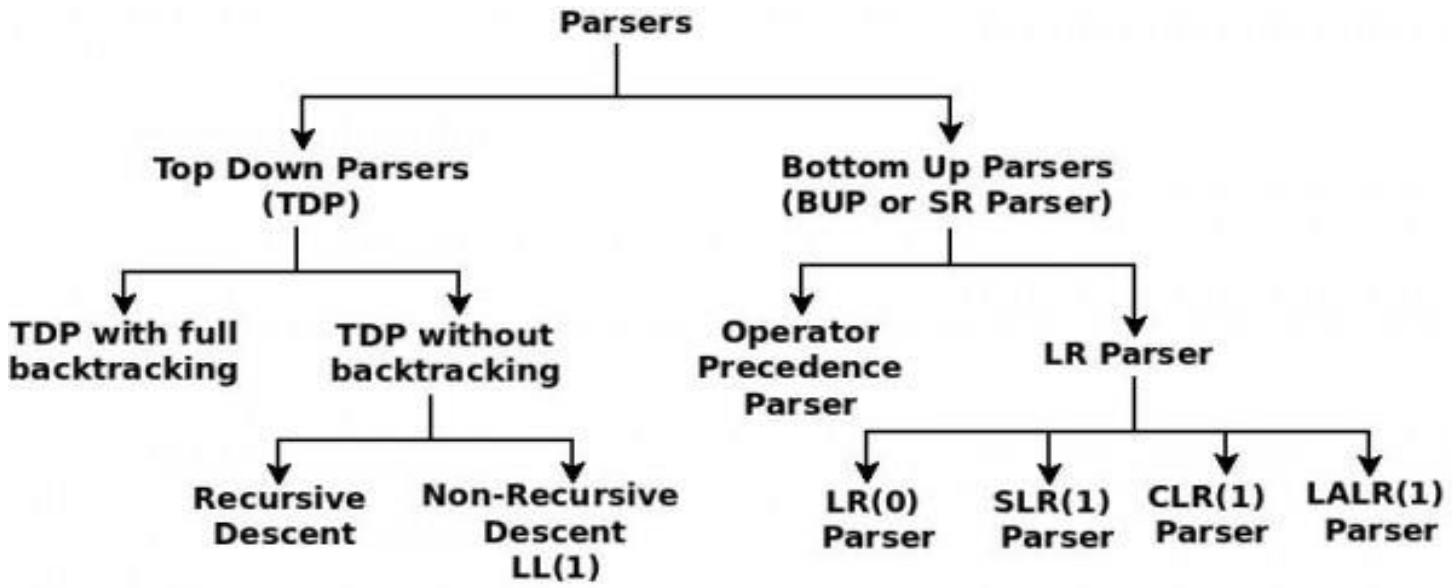
*تجزیه به کمک تحلیلگر نحوی و به نام تحلیل نحوی انجام می گیرد.

* در **تجزیه تعلق رشته ورودی به زبان مبدا** بررسی می شود. (وظیفه پارسر)

* بررسی طبق ساختار و نحو دستورات زبان مبدا انجام می گیرد.

***گذر (Pass)** تعداد دفعاتی که فایل ورودی مبدا یا فایل‌های مرتبط با آن از اول تا آخر خوانده می‌شود. در واقع هر بار مرور فایل ورودی مبدا یا فایل‌های مرتبط با آن را گذر می‌گویند.

**دسته بندی روش‌ها** روش های تجزیه بالا به پایین درخت تجزیه را از بالا به پایین میسازند، در حالی که روش های پایین به بالا بر عکس عمل می کنند یعنی درخت تجزیه را از پایین به بالا می سازند. در هر دو روش ورودی از **چپ به راست** در **هر گام تنها یک توکن** بررسی می شود.



$S \rightarrow AB$
$A \rightarrow aA \mid \lambda$
$B \rightarrow b \mid bB$

**خروجی تجزیه بالا به پائین**

سمت چپ ترین اشتقاق

| | |
|---|---|
| S | |
| AB | $S \rightarrow AB$ |
| aAB | $A \rightarrow aA$ |
| aaAB | $A \rightarrow aA$ |
| aaaAB | $A \rightarrow aA$ |
| aaa$\lambda$B | $A \rightarrow \lambda$ |
| aaab | $B \rightarrow b$ |

**خروجی تجزیه پائین به بالا**

سمت راست ترین اشتقاق

| | |
|---|---|
| aaab | |
| aaa$\lambda$b | |
| aaaAb | $A \rightarrow \lambda$ |
| aaAb | $A \rightarrow aA$ |
| aAb | $A \rightarrow aA$ |
| Ab | $A \rightarrow aA$ |
| AB | $B \rightarrow b$ |
| S | $S \rightarrow AB$ |

# مجموعه Follow و First

**First**

اگر رشته که a هر رشته ای از نمادهای گرامری باشد، مجموعه پایانه هایی رشته های مشتق شده ازآنها با a شروع می‌شوند، این مجموعه را با first (a) نشان می‌دهیم.

## محاسبه First ( A)

۱- اگر α پایانه باشد، First (α) برابرست با {α}

۲- اگر ε → α قانون گرامر باشد، ε به First (α) اضافه می‌شود.

**A → BCd**
**B → bB| e | $\lambda$**
**C → aC | $\lambda$**

Inpt: BCD
first(BCD) = {b, e, a, d}

**A → aA| aB**
**B → bB| c**

Input: aA
first(aA) = {a}

Input : aB
first(aB ) = {a}

**A → aA| bB| a**
**B → bB| b | $\lambda$**

first(A) = {a, b}
first(B ) = {b, $\lambda$}
first(bB ) = {b}
first(a) = {a}
first(b) = {b}

۳- اگر X غیرپایانه و $X \rightarrow Y1Y2\ldots Yn$ قانون گرامر، برای هر i, a در مجموعه First(Yi) باشد و ε در تمام مجموعه‌های First (Yn)…First ( Y1) قرار داشته باشد یعنی همه Yi بتوانند تهی را تولید کنند. در نتیجه X نیز می تواند ε را تولید کند در این صورت ε به مجموعه First (X) اضافه می شود.

$S \rightarrow AB$
$A \rightarrow aA \mid bB \mid a \mid \lambda$
$B \rightarrow bB \mid b \mid \lambda$

$First(S) = \{\lambda, a, b\}$

۴- اگر X غیرپایانه و $X \rightarrow Y1Y2\ldots Yn$ باشد، مجموعه First(Yi) (بجز ε )باشد به مجموعه First ( X) اضافه می‌گردد. زیرا First(Y1) مجموعه پایانه‌هایی هستند که در شروع رشته‌هایی که توسط Yi تولید می‌شوند قرار دارند از آنجایی که X با Yi شروع می‌شود، پس X با پایانه‌های First(Yi) شروع می‌شوند، در نتیجه First(Yi) به First(X) اضافه می‌گردد.

$A \rightarrow Bab$
$B \rightarrow c \mid d$

$First(B) = \{c, d\}$
$First(A) = \{c, d\}$

۵- اگر X غیرپایانه و $X \rightarrow Y1Y2\ldots Yn$ باشد، ε در مجموعه First(Yi) باشد (Yi نمی‌تواند ε را تولید کند)در این صورت علاوه بر First(Y1) (بجز ε)مجموعه First(Y2) (بجز ε) نیز به First ( X) اضافه می‌گردد.

**$A \rightarrow Bab$**
**$B \rightarrow c \mid d \mid \lambda$**

**First(B)={c, d, $\lambda$}**
**First(A)={a, c, d}**

## Follow

برای غیر پایانه A مجموعه ای از پایانهاست که در هر شبه جمله بلافاصله بعد از آن هستند.

پایانه ایی که بعد از غیرپایانه می توان دید.

$X \rightarrow aXAad \mid a$
$A \rightarrow Ac \mid Af \mid \lambda$

$Follow(A) = \{a, c, f\}$

### محاسبه Follow ( A)

۱-اگر S نماد شروع باشد $ (نشان دهنده آخر رشته ورودی است.) به Follow (S) اضافه می‌شود.

۲- اگر قانونی به صورت $A \rightarrow EBF$ وجود دارد آنگاه هرچیزی در First (F) بجز ε به مجموعه Follow (B) اضافه می‌شود.

۳- اگر قانونهایی بشکل $A \rightarrow EB$ یا $A \rightarrow EBF$ که First (F) حاوی $\lambda$ باشد، هر چیزی در مجموعه Follow(A) به Follow (B) اضافه می شود.

$A \rightarrow AXZ \mid \lambda$
$Z \rightarrow aZ \mid bZ \mid c \mid \lambda$
$X \rightarrow a$

First (Z)= {a, b, c, $\lambda$}
Follow(Z)={a, b, c}

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow c|e|D$
$d \rightarrow dD|d$

Follow(A) = {c, e, d}

$A \rightarrow AXb$
$X \rightarrow d|dB|eBE$
$E \rightarrow a|\lambda$
$B \rightarrow b$

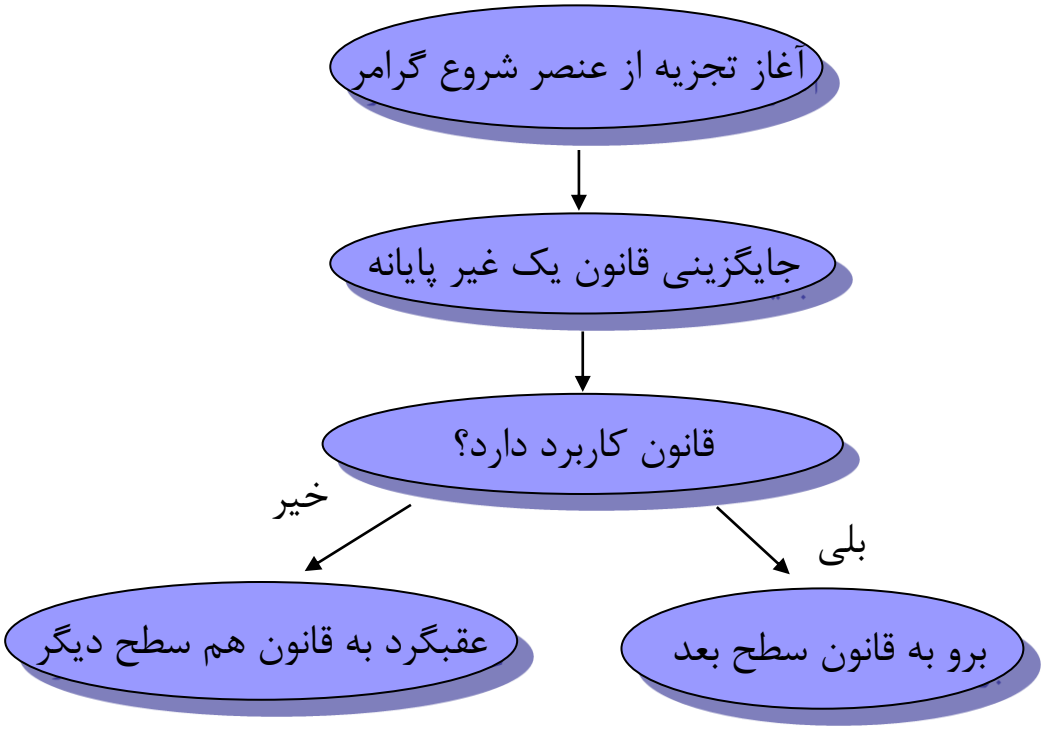
First(b)= {b}
First(E)= {a, $\lambda$}
Follow(X) = {b}
Follow(B) = {a, b}

# 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

زمانی که یک تجزیه‌کننده، درخت تجزیه را از نماد آغازین می‌سازد و سپس تلاش می‌کند تا نماد آغازین را به ورودی تبدیل کند، این کار تجزیه بالا به پایین نامیده می‌شود.



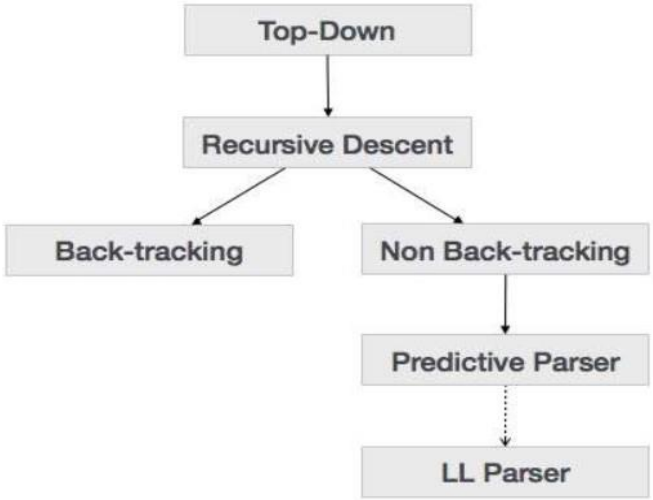
## **انواع پارسرهای بالا به پایین**(بازگشتی-کاهشی)

### **۱-تجزیه کننده بازگشتی Recursive descent Backtracking (**گرامرهای با عقبگرد**) پس‌گرد**

منظور از پس‌گرد این است که اگر اشتقاق یک ترکیب ناموفق باشد، تحلیل‌گر نحوی فرایند کاری خود را با استفاده از قواعد متفاوت از همان ترکیب مجدداً شروع می‌کند. در این تکنیک برای تعیین ترکیب صحیح ممکن است رشته ورودی بیش از یک بار پردازش شود.

### **۲-تجزیه کننده غیر بازگشتی Recursive descent non Backtracking** (گرامرهای بدون عقبگرد) **(تجزیه پایین‌گرد)**

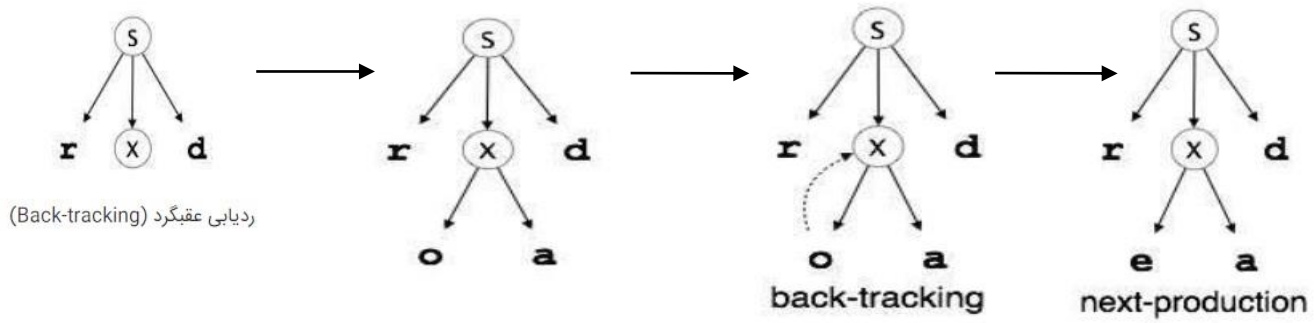
این نوع از تجزیه که تجزیه بازگشتی-کاهشی نیز نامیده می‌شود، از جمله روش‌های تجزیه بالا به پایین رایج است. دلیل این که این تجزیه بازگشتی Recursiveنامیده شده، این است که از رویه‌های بازگشتی برای پردازش ورودی استفاده می‌کند. تجزیه پایین‌گرد از مشکل پس‌گرد Backtrackingرنج می‌برد.



## ۱-تجزیه کننده  پس گردی Back-tracking

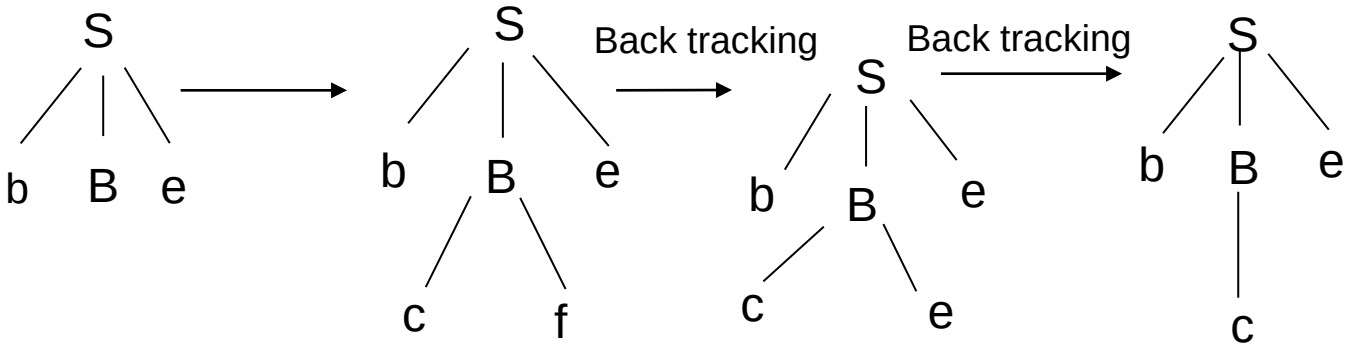
تجزیه کننده از Sشروع می کند و با بررسی قواعد ترکیب، خروجی را با چپ ترین حرف ورودی بعدی یعنی rمطابقت می دهد. اولین ترکیب S → rXd با آن مطابقت دارد. بنابراین تجزیه کننده بالا به پایین به سمت حرف ورودی بعدی یعنی eپیش می رود. تجزیه گر سعی دارد Xغیر پایانی را بسط داده و ترکیب آن را از سمت چپ X->oaبررسی کند. این عبارت با نماد ورودی بعدی مطابقت ندارد. بنابراین تجزیه کننده بالا به پایین برای به دست آوردن قانون ترکیب بعدی X، X →ea پس گرد می کند.backtrackاکنون که تجزیه کننده تمام حروف ورودی را به شکلی مرتب شده مطابقت داده. رشته پذیرفته شده است. این روش بسیار کند و زمانبر است.

**S → rXd | rZd**
**X → oa | ea**
**Z → ai**




**S → bBe**
**B → cf | ce |c**

**Input bce**



```
void A() {
        choose an A-production, A->X1X2..Xk
        for (i=1 to k) {
                if (Xi is a nonterminal
                        call procedure Xi();
                else if (Xi equals the current input symbol a)
                        advance the input to the next symbol;
                else /* an error has occurred */
        }
}
```

در روش بازگشتی-کاهشی ممکن است نیاز به عقبگرد باشد.

به این منظور، کد بیان شده تغییر میکند.

در حالت کلی نمیتوان یک قانون خاص را به سادگی انتخاب کرد.

بنابراین باید انتخابهای ممکن بررسی شوند.

اگر یکی از انتخابها شکست بخورد، پوینتر ورودی باید به ابتدا برگردد و توکن دیگری انتخاب شود.

پارسرهای بازگشتی کاهشی برای گرامرهایی که بازگشتی چپ دارند قابل استفاده نیستند.

**S → bab | bA**
**A → d | cA**

**Input: bcd**

| | | |
|---|---|---|
| S | bcd | Try S –> bab |
| bab | bcd | match b |
| ab | cd | dead-end, backtrack |
| S | bcd | Try S –> bA |
| bA | bcd | match b |
| A | cd | Try A –> d |
| d | cd | dead-end, backtrack |
| A | cd | Try A –> cA |
| cA | cd | match c |
| A | d | Try A –> d |
| d | d | match d |
| | | Success! |

به ازای هر غیرپایانهٔ گرامر یک تابع می نویسیم.

برای تجزیهٔ یک جمله توابع متناظر با هر یک از قواعد گرامر که برای تولید آن جمله باید مورد استفاده قرار گیرند، را فراخوانی می‌کنیم.