طراحی کامپایلرها

تحلیل نحوی (قسمت دوم)

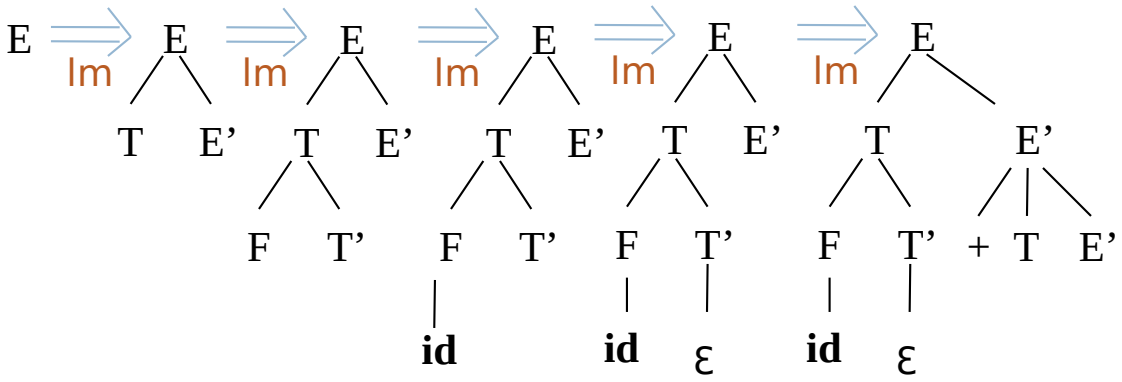
قمرناز تدین تبریزی

Top Down Parsing

مقدمه

- هدف پارسر بالا به پایین ایجاد یک درخت تجزیه از ریشه به سمت برگها با خواندن ورودی از چپ به راست است. (سمت چپ ترین اشتقاق)
- مثال: id+id*id

E -> TE’
E’ -> +TE’ | Ɛ
T -> FT’
T’ -> *FT’ | Ɛ
F -> (E) | **id**



تجزیه بازگشتی–کاهشی Recursive descent

- از مجموعه ای از روایهها تشکیل می شود. (یکی برای هر غیرپایانی)
- اجرا با رویه نماد شروع آغاز می شود.
- مثال: یک رویه برای غیرپایانیها:

```
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 */
    }
}
```

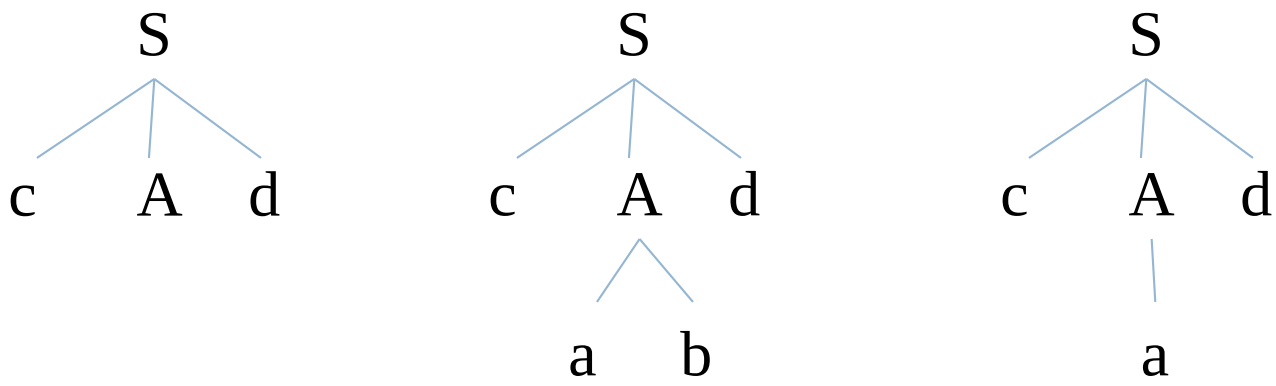
تجزیه بازگشتی–کاهشی (ادامه)

- 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 راخ دیگری انتخاب شود.
- پارسرهای بازگشتی کاهشی برای گرامرهایی که بازگشتی چپ دارند قابل استفاده نیستند.

Example

S->cAd
A->ab | a

Input: cad



تجزیه

- هدف ؟
- دو روش اصلی
  - بالا به پائین (Top Down)
  - پائین به بالا (Bottom Up)
- مثلا برای گرامر زیر این کار را بصورت دستی انجام می دهیم.

S AB

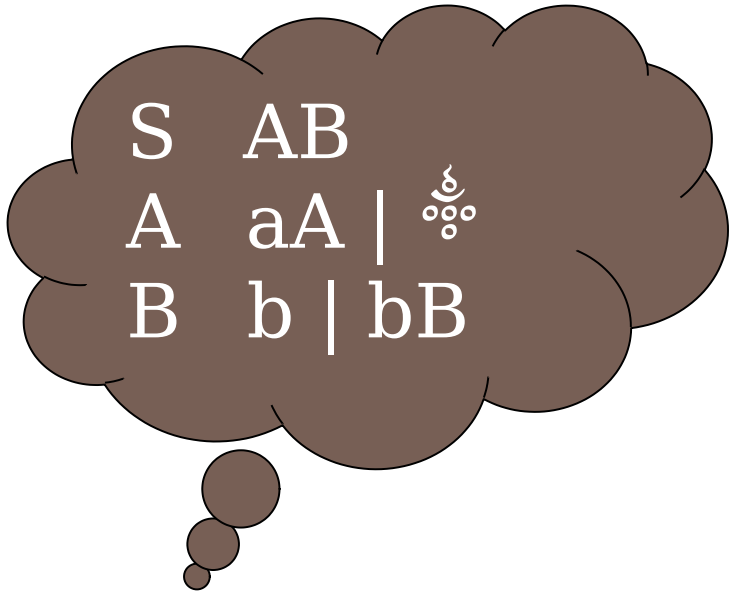
A aA | ε

B b | bB

خروجی تجزیه بالا به پائین

S

AB	S	AB
aAB	A	aA
aaAB	A	aA
aaaAB	A	aA
aaaεB	A	ε
aaab	B	b



- سمت چپ ترین اشتقاق

تجزیه گر پائین به بالا

aaab

aaaεb

aaaAb A ε

aaAb A aA

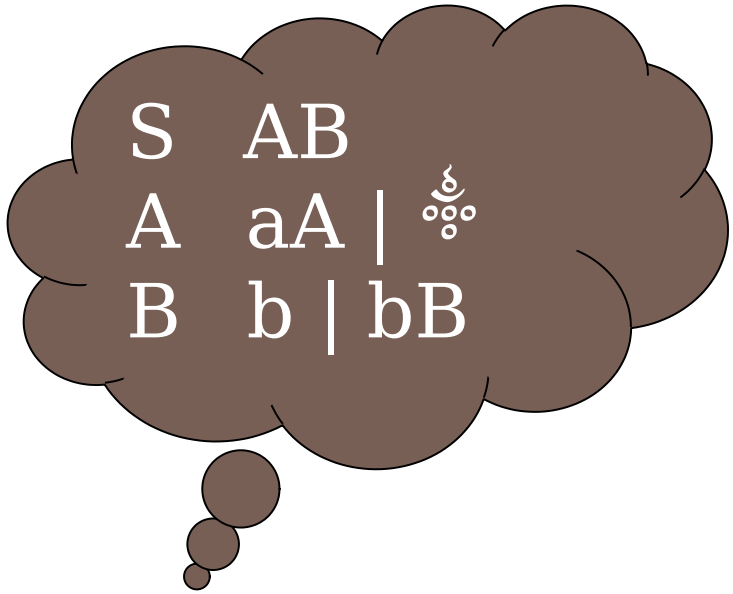
aAb A aA

Ab A aA

AB B b

S S AB

- سمت راست ترین اشتقاق



تجزیه گر ها

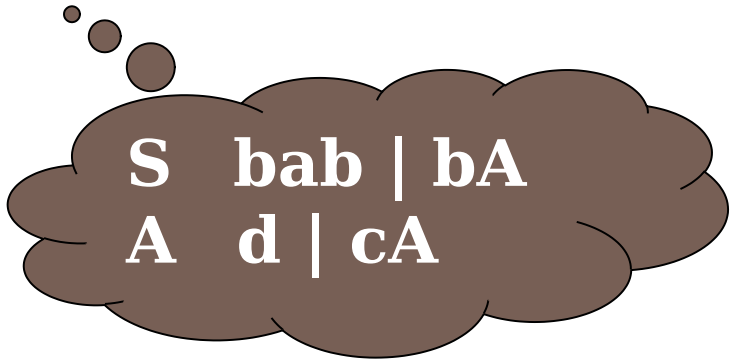
- در دو مثال قبل توانستیم به راحتی قواعد قابل اعمال را انتخاب کنیم، اما یک تجزیه گر نمی تواند این گونه عمل کند. چرا ؟
- تجزیه گر ها معمولا فقط از یک ترم پیش بینی برای انتخاب استفاده می کنند.
- مثال زیر را در نظر بگیرید :

B b

B bB

Backtracking

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!



تجزیه گر کاهینه بازگشتی (1)

- 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 می کنیم.
- به عنوان مثال برای گرامر زیر یک تجزیه گر تولید می کنیم.

program function_list

function_list function_list function | function

function FUNC identifier (parameter_list) statements

```
//function   FUNC identifier ( parameter_list ) statements
void ParseFunction() {
    if (lookahead != T_FUNC)  {
    printf("syntax error \n");
    exit(0);
    } else
        lookahead = yylex();
    ParseIdentifier();
    if (lookahead != T_LPAREN) {
    printf("syntax error \n");
    exit(0);
    } else
        lookahead = yylex();
    ParseParameterList();
    if (lookahead!= T_RPAREN) {
    printf("syntax error \n");
    exit(0);
    } else
        lookahead = yylex();
    ParseStatements();
}
```

```
void MatchToken(int expected) {
    if (lookahead != expected) {
    printf("syntax error, expected %d, got %d\n", expected,lookahead);
    exit(0);
    } else
        lookahead = yylex();
}

//function   FUNC identifier ( parameter_list ) statements
void ParseFunction(){
    MatchToken(T_FUNC);
    ParseIdentifier();
    MatchToken(T_LPAREN);
    ParseParameterList();
    MatchToken(T_RPAREN);
    ParseStatements();
}
```

تجزیه گر کاهینه بازگشتی (2)

- مثال دیگر

if_st IF expression THEN statement ENDIF |
IF expression THEN statement ELSE statement ENDIF

- آیا این گرامر مبهم است؟
- تجزیه گر کاهینه بازگشتی برای این گرامر ؟
- و حالا این گرامر :

if_st IF expression THEN statement close_if

close_if ENDIF | ELSE statement ENDIF

```
//if_st   IF expression THEN statement close_if
void ParseIfSt() {
    MatchToken(T_IF);
    ParseExpression();
    MatchToken(T_THEN);
    ParseStatement();
    ParseCloseIf();
}

//close_if   ENDIF | ELSE statement ENDIF
void ParseCloseIf() {
    if (lookahead == T_ENDIF)
        lookahead = yylex();
    else {
        MatchToken(T_ELSE);
        ParseStatement();
        MatchToken(T_ENDIF);
    }
}
```

تجزیه گر کاهینه بازگشتی (3)

- اگر بخواهیم تابع متناظر با قواعد زیر را بنویسیم، با چه مشکلی روبرو هستیم ؟

statement assg_statement | return_statement |
print_statement | null_statement| if_statement |
while_statement
| block_of_statements

- آیا برای انتخاب قاعدهٔ قابل اعمال با توجه به وروحی بعدی با مشکلی مواجه هستیم ؟
- برای حل این مشکل نیاز به تعریف مجموعه آغازین(First) داریم.

First and Follow

- First() مجموعه پایانیهایی‌استکه رشته های‌مشتق‌شده با آن شروع میشوند. *
- If α=>ε then is also in First(ε)

- در تجزیه پیشگو، اگرA-> α|β را داشته باشیم، درصورتیکه First(α) و First(β) دو مجموعه غیرالحاقی باشند، با دریافت ورودی میتوان یک قانون مناسب را انتخاب کرد.
- Follow(A) , برای‌هر غیرپایانیA، مجموعه ایپایانی‌هایa است که میتولنند بلافاصله بعد از A در یک‌شبه جمله ظاهر شوند. *
  - اگر برای برخی α و β داشته باشیم S => αAaβ آنگاه a در Follow(A) است.
  - اگر A سمت راست ترین نماد در شبه جمله باشد آنگاه $ در Follow(A)است.

LL(1) Grammars

- پارسرهای پیشگو، پارسرهای بازگشتی کاهشی هستند که نیازی به عقبگرد ندارند.
- به گرامرهایی که میتوان برای آنها پارسر پیشگو نوشت LL(1) گفته می شود.
  - L اول به معنی خواندن ورودی از چپ به راست است
  - L دوم به معنی اشتقاق چپ است
  - 1 به معنی یک نماد ورودی به عنوان lookahead است.

- گرامر LL(1) ، G است اگر و تنها اگردر صورتی که A-> α|β دو قانون از G باشند، شرایط زیر را داشته باشیم:
  - برای هیچ پایانی a هر دو α و β با a شروع نشوند.
  - حداکثر یکی از α و β رشته تهی را تولید کنند.
  - اگر α=>* ε آنگاه β هیچ رشته ای تولید نکند که با پایانی ای در Follow(A) شروع شود.

Computing First

- برای محاسبه First(X) برای نماد X گرامر، قوانین زیر به کار میروند تا زمانی که هیچ ترمینال دیگر یا نماد ε را نتوان به مجموعه First اضافه کرد.
  1. اگر X یک پایانی است، First(X) = {X} .
  2. اگر X غیرپایانی است و X->Y1Y2…Yk یک قانون وk>=1 است، آنگاه a در First(X) است اگر برای برخی i ها، a در First(Yi) باشد و ε در همه ,First(Y1*) …,First(Yi-1) باشد یعنی ε => Y1…Yi-1. اگرε در First(Yj) for j=1,…,k آنگاه ε به First(X) اضافه میشود.
  3. اگرε <-X آنگاه ε به First(X) اضافه می شود.

مجموعه آغازین (2)

- حالا فرض کنید قاعده ای به شکل زیر داریم :

$$\ldots \mid A \quad u_1 \mid u_2$$

- با توانائی محاسبهٔ مجموعهٔ آغازین انتخاب یکی از این قوانین راحت تر شده است.
- در صورت وجود چه شرطی می توان یک تجزیه گر کاهینه بازگشتی برای این گرامر نوشت؟

```
void ParseA() {
    switch (lookahead)  {
        case First(u1): A   u 1
            /* code to recognize u1 */
            return;
        case First( u 2): // A   u2
            /* code to recognize u 2 */
            return;
        ...
        default:
            printf("syntax error \n");
            exit(0);
    }
}
```

مجموعه آغازین (3)

- اگر غیر پایانه ای که در حال تجزیه اش هستیم، قابل تهی شدن (nullable) باشد، چه اتفاقی می افتد ؟
  - غیرپایانهٔ X را قابل تهی شدن می نامند اگر اشتقاقی وجود داشته باشد که طی آن X به ε برود.
  - بر اساس مجموعهٔ آغازین چه زمان یک غیر پایانه قابل تهی شدن است؟
- برای برخورد با این مشکل مجموعهٔ پیرو (Follow) را تعریف می کنیم.

مجموعهٔ پیرو

- مجموعهٔ پیرو برای یک غیر پایانه مثل X را با Follow(X) نمایش می دهند.
- برای هر شبه جمله مثل $\underline{u}X\underline{v}$ $S \Rightarrow^* \underline{u}X\underline{v}$ که در آن $\underline{v}$ با پایانه آغاز می شود، آن پایانه در مجموعهٔ پیرو X قرار دارد.
- کاربرد مجموعهٔ پیرو در اینجا، برای زمانی است که یک غیرپایانه قابل تهی شدن باشد.

```
void ParseA() {
    switch (lookahead) {
        case First(u1) :
            /* code to recognize u1 */
            return;
        case First(u2) :
            /* code to recognize u2 */
            return;
        ...
        case Follow(A): // if A is nullable
            /* ??? */
        default:
            printf("syntax error \n");
            exit(0);
    }
}
```

قواعد دارای بازگشت چپ (1)

- آیا این قواعد مشکلی ایجاد می کنند؟
- مثال زیر را در نظر بگیرید :

```
function_list   function_list function | function
function   FUNC identifier ( parameter_list ) statement

void ParseFunctionList() {
    ParseFunctionList();
    ParseFunction();
}
```

قواعد دارای بازگشت چپ (2)

- برای رفع مشکل گرامر را بازنویسی کرده و تابع مربوطه را می نویسیم:

function_list function more_functions

more_functions function more_functions | ε

```
void ParseFunctionList()  {
    ParseFunction();
    ParseMoreFunctions();
}
```

ساخت مجموعه آغازین (جمع بندی)

- برای محاسبهٔ مجموعه آغازین u، در زمانی که u به فرم X1X2...Xn باشد، به شکل زیر عمل می کنیم:
  - اگر X1 پایانه است، آن را به مجموعهٔ First(u) می افزائیم و کار تمام شده است.
  - در غیر اینصورت اگر X1 غیرپایانه است،
    - ε - First(X1) را به First(u) می افزائیم.
    - اگر X1 قابل تهی شدن است، ε - First(X2) را به First(u) می افزائیم.
      - اگر X2 هم قابل تهی شدن باشد، ε - First(X3) را هم به First(u) می افزائیم، و ... (تا زمانی که به یک نماد غیر قابل تهی شدن برسیم)
      - اگر X1X2...Xn * ε باشد، ε را به مجموعهٔ آغازین می افزائیم.

مثالی از محاسبهٔ مجموعه های آغازین و پیرو

- برای گرامر زیر مجموعهٔ آغازین و پیرو را حساب می کنیم.

S AB

A Ca | ε

B BaAC | c

C b | ε

تجزیه بالا به پائین پیشگو

دو نوع تجزیه گر بالا به پائین پیشگو

- کاهینه بازگشتی (Recursive descent)
- مبتنی بر جدول (Table driven)

تجزیه بالا به پائین – تجزیه مبتنی بر جدول

در تجزیه گر کاهینه بازگشتی،

- قواعد چگونه ذخیره شده اند؟
- پشته چطور؟

- در تجزیه گر مبتنی بر جدول،
  - قواعد را در یک جدول ذخیره می کنیم و
  - از یک پشته جهت انجام عمل تجزیه استفاده می کنیم.

تجزیه مبتنی بر جدول (1)

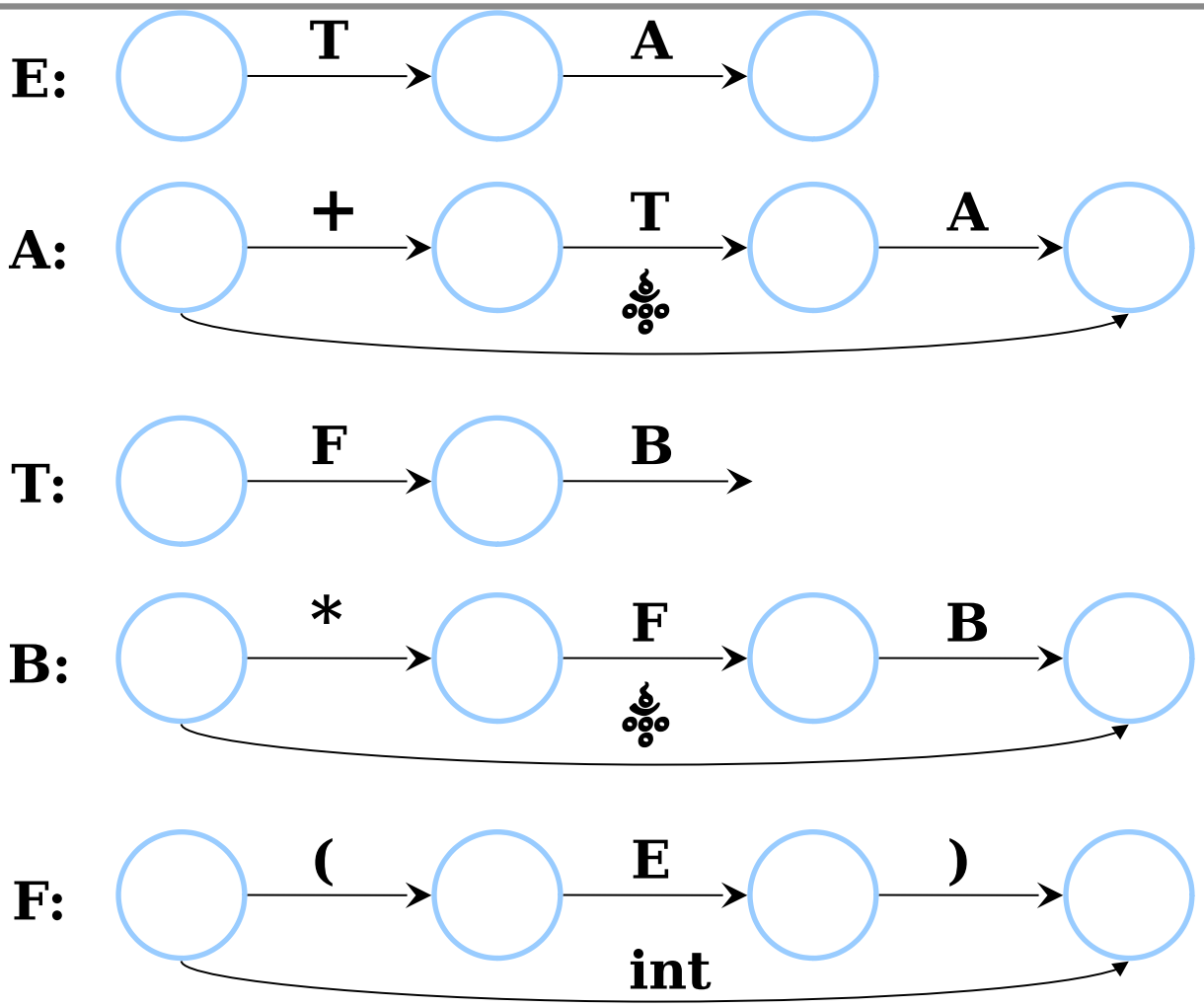
- گرامر زیر را در نظر بگیرید:

E E + T | T
T T * F | F
F (E) | int

- پس از حذف بازگشت چپ داریم :

E TA
A +TA | ε
T FB
B *FB | ε
F (E) | id

تجزیه مبتنی بر جدول (2)

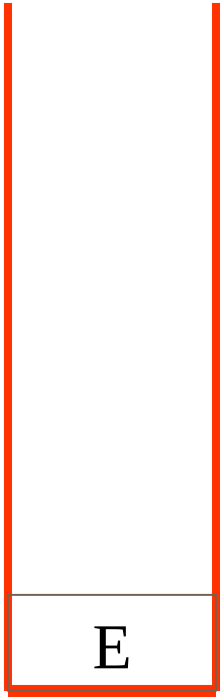


E TA
A +TA | ε
T FB
B *FB | ε
F (E) | id

String:
int + int * int

1. Push the start symbol on the stack
2. top_of_stack = pop()
3. rc = lookup(top_of_stack, next_token, new_stack_symbols)
4. push(new_stack_symbols)
5. top_of_stack = pop()
6. if (top_of_stack = next_token)
7. lex(next_token)
8. top_of_stack = pop()
9. goto 6
10. if (top_of_stack != empty_stack)
11. goto 3
12. else push(top_of_stack)

Stack

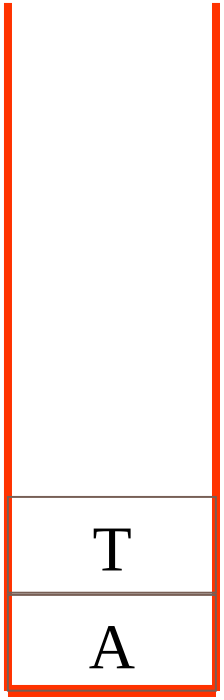


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

id	+	**id**	**$**

رشته ورودی

Stack

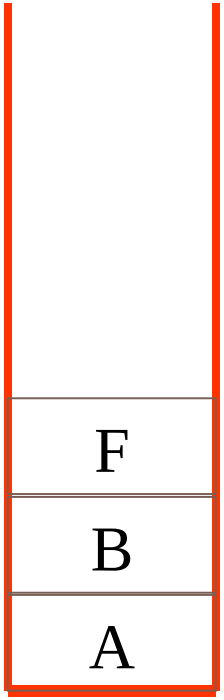


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

id	+	**id**	**$**

رشته ورودی

Stack

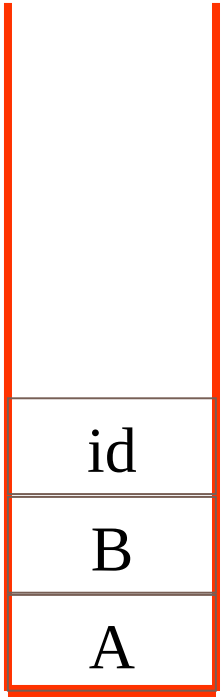


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

id	+	**id**	**$**

رشته ورودی

Stack

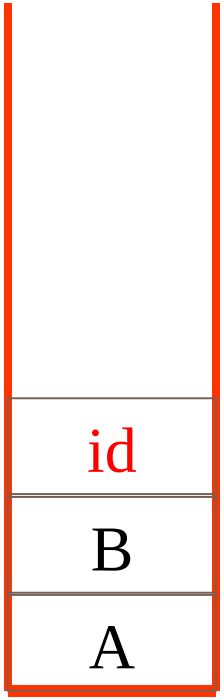


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

id	+	**id**	**$**

رشته ورودی

Stack

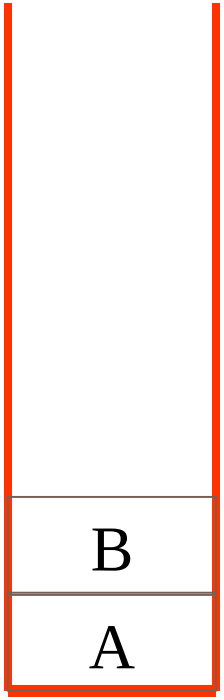


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

id	+	**id**	**$**

رشته ورودی

Stack

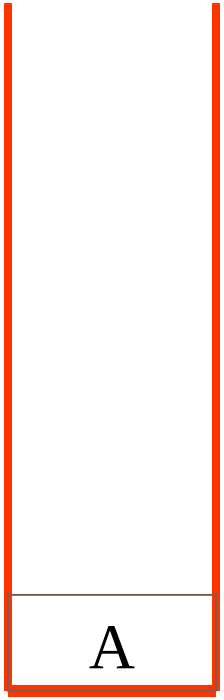


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

+	**id**	**$**

رشته ورودی

Stack

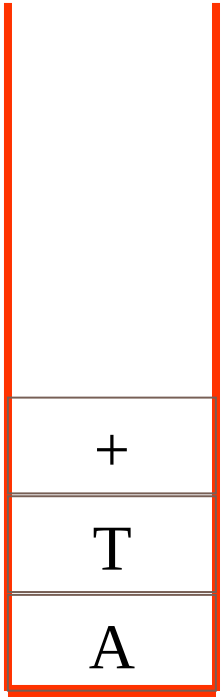


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

+	**id**	**$**

رشته ورودی

Stack

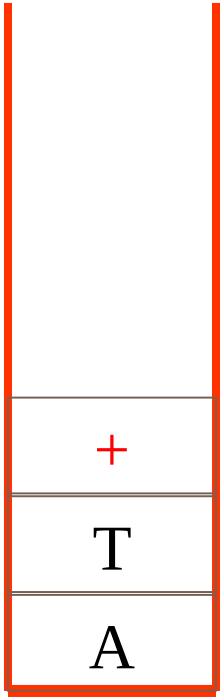


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

+	**id**	**$**

رشته ورودی

Stack

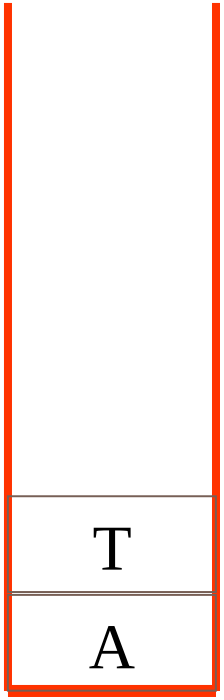


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

+	**id**	**$**

رشته ورودی

Stack

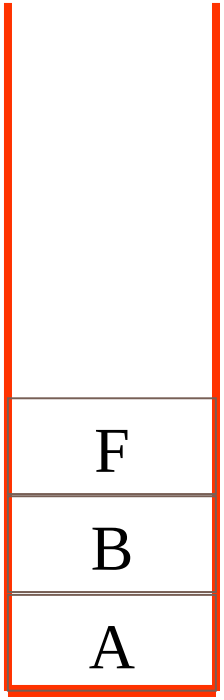


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

id	**$**

رشته ورودی

Stack

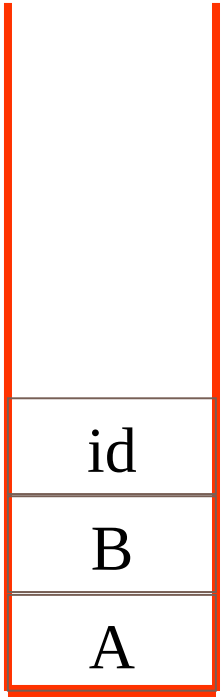


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

id	**$**

رشته ورودی

Stack

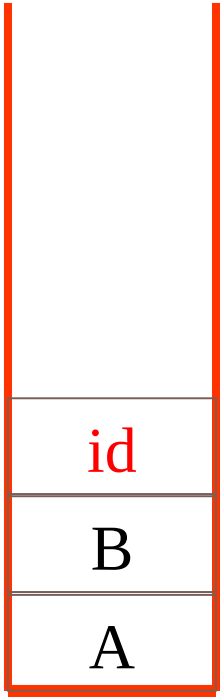


Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

id	**$**

رشته ورودی

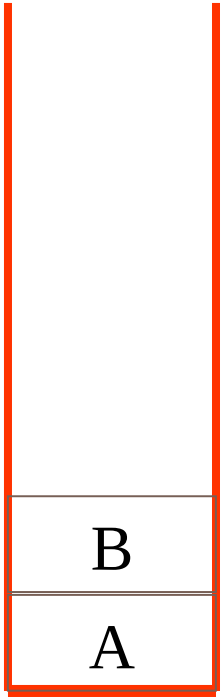
Stack



Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

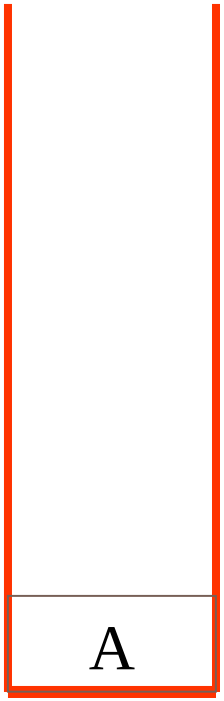
id	$

رشته ورودی



Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

رشته ورودی



Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

رشته ورودی

Stack

Nonterminal	(	id	+	*	)	$
E	TA	TA				
A			+TA		ε	ε
T	FB	FB				
B			ε	*FB	ε	ε
F	(E)	id				

رشته ورودی

انجام عمل تجزیه برای رشتهٔ id + id

Stack	Input	Action
E$	id + id	
TA$		
FBA$		
intBA$		
BA$		
A$		
+TA$		
TA$		
FBA$		

Stack	Input	Action
intBA$		
BA$		
*FBA$		
FBA$		
intBA$		
BA$		
A$		
$		

نحوهٔ پر کردن جدول تجزیه

- برای هر قاعده ای از گرامر مثل A → u
  - بازای هر پایانهٔ موجود در مجموعهٔ آغازین u مثل a، در خانهٔ [A,a] جدول قاعدهٔ A → u قرار می دهیم.
  - اگر u قابل تهی شدن بود (یا به عبارتی A قابل تهی شدن بود)، در تمام خانه های [A,b] که در آن، b عضو مجموعهٔ پیرو A می باشد، A → u قرار می دهیم. در این حالت اگر $ عضو مجموعهٔ پیرو A بود، A → u را در خانهٔ [A,$] هم قرار می دهیم.

نحوهٔ پر کردن جدول تجزیه – سوال

- خانه هائی از جدول که در آنها هیچ مقداری قرار نگرفته است، چگونه تفسیر می شوند؟
- ممکن است برای بعضی از گرامر ها در یک خانه از جدول دو مقدار قرار بگیرد، این حالت نشانهٔ چیست؟

Nonterminal	(	id	+	*	)	$
E						
A						
T						
B						
F						

چگونگی شناخت گرامر های LL(1)

- یک تجزیه گر پیشگو (کاهینه بازگشتی، مبتنی بر جدول) را فقط برای گرامرهائی می توان ساخت که LL(1) باشند.
- راههای آشکار و سادهٔ شناخت یک گرامر غیر LL(1) چیست ؟
  - وجود فاکتور چپ مشترک
  - وجود بازگشت چپ
  - وجود ابهام
- راه مطمئن تر برای تشخیص LL(1) بودن یک گرامر، ساخت جدول تجزیه مربوطه می باشد.

ویژگی های گرامر های LL(1)

- گرامر G، یک گرامر LL(1) است؛ در صورتی که اگر قواعد A u|v در این گرامر وجود داشت، شرایط زیر نیز برقرار باشد:
  - مجموعه های آغازین u و v اشتراک نداشته باشند.
  - از u و v ، حداکثر یکی قابل تهی شدن باشد.
  - اگر $v \Rightarrow^{*} \varepsilon$ ، آنگاه اشتراک مجموعهٔ پیرو A و مجموعهٔ آغازین u تهی باشد.

پوشش خطا – ایدهٔ اول

- ایدهٔ اول : اگر یک غیرپایانه در بالای پشته قرار گرفته است و به خطا برخورد کردیم، تا زمانی که در رشتهٔ ورودی، به یک نشانهٔ هماهنگی (synch) نرسیده ایم، نشانه های ورودی را نادیده می گیریم؛ پس از رسیدن به نشانهٔ هماهنگی ، غیرپایانهٔ بالای پشته را حذف می کنیم.
- نشانهٔ هماهنگی چه ویژگی دارد که با رسیدن به آن می توان غیرپایانهٔ بالای پشته را حذف نمود؟
- این روش ممکن است منجر به بروز خطاهای آبشاری شود.

پوشش خطا – ایدهٔ اول – مثال (1)

- گرامر زیر را در نظر بگیرید. برای این گرامر جدول تجزیه را تشکیل می‌دهیم.

S AaS | b

A cB | dB | eCDBfDB

B DB | ε

C c | d | eCDBf

D gC | hC

پوشش خطا – ایدهٔ اول – مثال (2)

NT	a	b	c	d	e	f	g	h	$
S									
A									
B									
C									
D									

پوشش خطا – ایدهٔ اول – مثال (3)

NT	a	b	c	d	e	f	g	h	$
S		b	AaS	AaS	AaS				Syn
A	Syn		cB	dB	eCDBfDB				
B	⚘					⚘	DB	DB	
C	Syn		c	d	eCDBf	Syn	Syn	Syn	
D	Syn					Syn	gC	hC	

پوشش خطا
ایدهٔ اول
مثال (4)

NT	a	b	c	d	e	f	g	h	$
S		b	AaS	AaS	AaS				Synch
A	Synch		cB	dB	eCDBfDB				
B	ε					ε	DB	DB	
C	Synch		c	d	eCDBf	Synch	Synch	Synch	
D	Synch					Synch	gC	hC	

STACK	CURRENT INPUT	PRODUCTION TO APPLY
$S\$$	$cgah\$$	$S \rightarrow AaS$
$AaS\$$	$cgah\$$	$A \rightarrow cB$
$cBaS\$$	$cgah\$$	match
$BaS\$$	$gah\$$	$B \rightarrow DB$
$DBaS\$$	$gah\$$	$D \rightarrow gC$
$gCBaS\$$	$gah\$$	match
$CBaS\$$	$ah\$$	error, synch
$BaS\$$	$ah\$$	$B \rightarrow \epsilon$
$aS\$$	$ah\$$	match
$S\$$	$h\$$	error
$S\$$	$\$$	error, synch
$\$$	$\$$	parse complete

پوشش خطا – ایدهٔ دوم و سوم

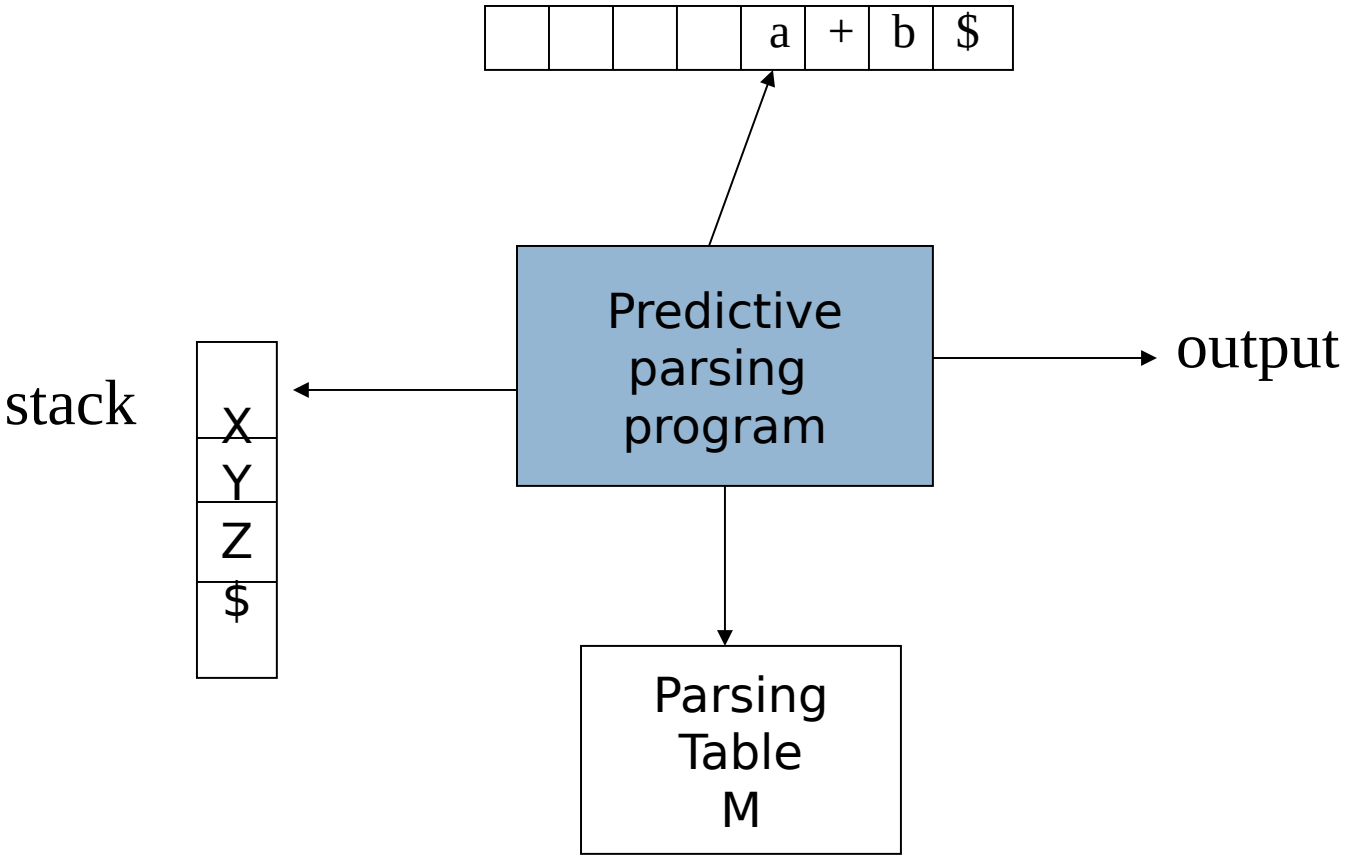
- در زمانی که یک غیرپایانه در بالای پشته بود و به خطا برخورد کردیم، تا زمان رسیدن به یکی از عناصری که در مجموعهٔ آغازین آن غیرپایانه قرار دارند، ورودی را نادیده می گیریم.
- برای پیاده سازی این ایده چه تغییری باید در جدول نمادها بدهیم؟
- در صورتی که در بالای پشته به یک پایانه مثل a برخورد کردیم، می توانیم پیغام inserting "a" in input را صادر کنیم.

Another example

S -> iEtSS’ | a
S’ -> eS | ε
E -> b

Non - terminal	Input Symbol					
	a	b	e	i	t	$
SS -> a			S -> iEtSS’			
S’			S’ -> ε S’ -> eS			S’ -> ε
E		E -> b				

Non-recursive predicting parsing



Predictive parsing algorithm


```
Set ip point to the first symbol of w;
Set X to the top stack symbol;
While (X<>$) { /* stack is not empty */
   if (X is a) pop the stack and advance ip;
   else if (X is a terminal) error();
   else if (M[X,a] is an error entry) error();
   else if (M[X,a] = X->Y1Y2..Yk) {
   output the production X->Y1Y2..Yk;
   pop the stack;
   push Yk,…,Y2,Y1 on to the stack with Y1 on top;
   }
   set X to the top stack symbol;
}
```