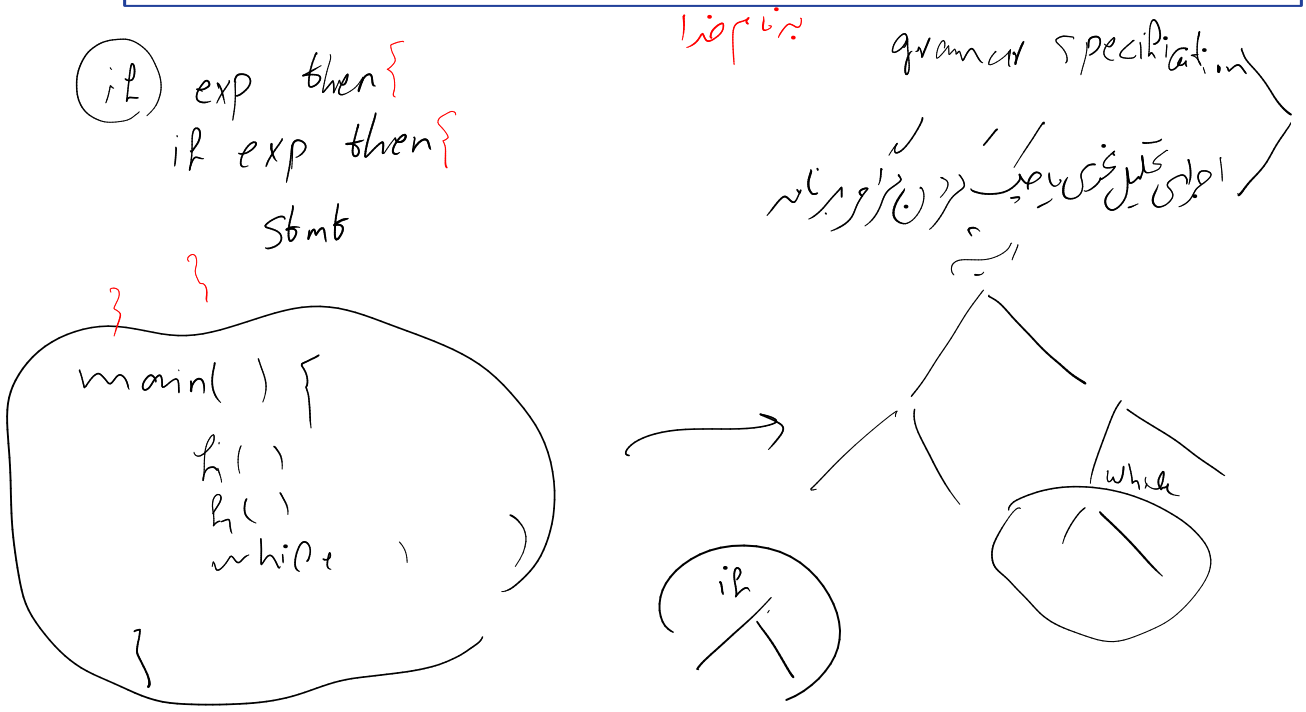
به نام خدا



تحلیل نحوی دنباله‌ای از توکن‌ها را دریافت کرده و ساختارهای مختلف زبان را از آن استخراج می‌کند
بدین منظور یک درخت پارس تولید می‌شود

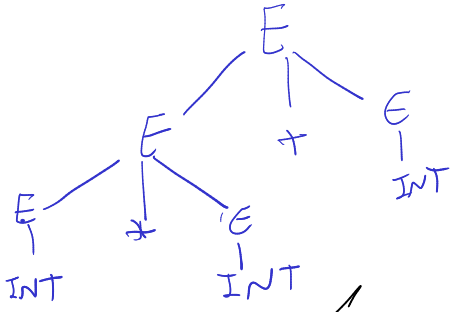
ضعف RE برای تشخیص ساختارهای گرامری زبان‌های برنامه‌نویسی؟

قدرت RE برای ساختارهای زبان کافی نیست مثلاً توسط RE
نمی‌توان تعادل پرانتزها را شمارش کرد.

(( ) + ( ))

Context Free Grammar

مثال) 3x4+5



راه کار طی

زبان‌های برنامه‌نویسی را از طریق CFG ها توصیف کنیم و سپس کامپایلر به ازای هر رشته از توکن‌ها تشخیص دهد آن رشته توسط زبان توصیف شده تولید می‌شود یا نه.

CFG

- T : مجموعه سمبل‌های پایانی یا ترمینال
- N : مجموعه سمبل‌های غیرپایانی
- $S \in N$ سمبل شروع

قوانین $X \rightarrow Y_1 \dots Y_n$ ، $X \in N$ ، $Y_i \in N \cup T \cup \{\varepsilon\}$

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

رشته $(3*4+5)$

مراحل تجزیه یک رشته با گرامر

۱- از سمبل شروع آغاز می‌کنیم

۲- یک سمبل غیرپایانی در رشته را با سمت راست یکی از قوانین جایگزین می‌کنیم

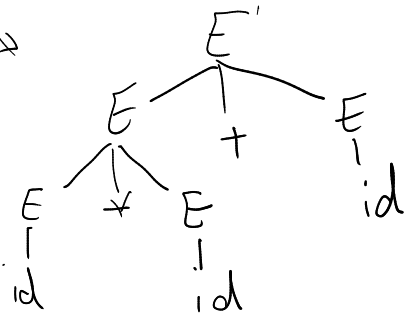
۳- مرحله ۲ را تکرار می‌کنیم به طوری که نهایتا خروجی سمبل غیرپایانی دیگری نداشته باشد و رشته‌های با رشته مورد نظر تطابق داشته باشد

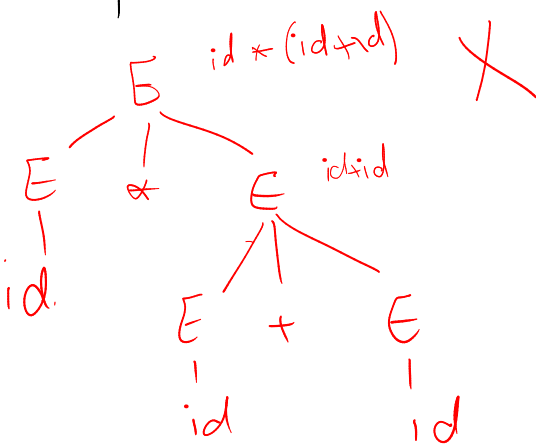
$E \rightarrow E+E \rightarrow E*E+E \rightarrow id*E+E \rightarrow id*id+E \rightarrow id*id+id$

$E \rightarrow E*E \rightarrow E*E+E \rightarrow E*id+id \rightarrow id*id+id$

$L(G) = \{a_1 \dots a_n \mid \forall a_i \in T \wedge S \xrightarrow{*} a_1 \dots a_n\}$

روش پارس





برنامه خدا

دنباله‌ای از قوانین گرامر که از سمبل شروع آغاز می‌شود و به رشته مورد نظر ختم می‌شود — Derivation (اشتقاق)

به ازای هر اشتقاق یک درخت پارس بدست می‌آید

* به ازای یک رشته، ممکن است اشتقاق‌های زیادی وجود داشته باشد ولی همه اشتقاق‌ها برای پارس مورد قبول نیست

در هر مرحله چپ‌ترین غیرترمینال را جایگزین کن

دو نمونه اشتقاق:
- left-most derivation
- right-most derivation

روش‌های تجزیه یا پارس:
- بالا به پایین:
  - backtracked recursive (تجزیه‌گر بازگشتی)
  - recursive (بازگشتی)
  - با استفاده از جدول (غیربازگشتی) LL(1)
- پایین به بالا

$A \to \beta_1 | \beta_2 | \dots | \beta_m$

```
bool while A( ){ {
    choose an    A-production
    for( i=1 to k){
      if ( Xi ∈ N)
          Xi ()
      else  if ( Xi = *next)
          next++i
      else
          break    // error
    }
}
```

$X_i \in T \cup N$

$A \to X_1 X_2 \dots X_k$

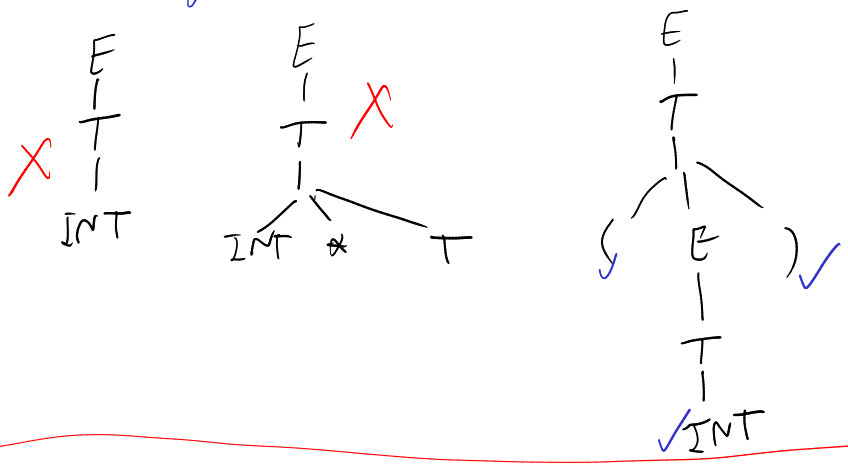
*next

ورودی $a_1 a_2 \dots a_n$

مثال) $E \to \underbrace{T}_{E_1} \mid \underbrace{T + E}_{E_2}$

$T \to \underbrace{INT}_{T_1} \mid \underbrace{INT * T}_{T_2} \mid \underbrace{(E)}_{T_3}$

رشته ورودی (INT)



پارس تمام می‌شود

روش پارس Recursive

- فرض می‌کنیم که توکن‌ها از lexer نوبت‌به‌نوبت آمده‌اند
- متغیر سراسری next به ورودی بعدی اشاره می‌کند
- به ازای هر non terminal یک تابع boolean که چک کردن قانون S_n جهت match شدن

- همه قوانین را با پیاده‌کردن یک match امتحان کنیم

```
bool E() {
    Token *save = next
    return ( E1() || (next=save, E2()) )
}

bool E1() {
    return T()
}

bool E2() {
    return (T() && match('+') && E())
}

bool T() {
    Token *save = next
    return ( T1() || (next=save, T2()) || (next=save, T3()) )
}

match( Token tok) {
    return *next++ == tok
}
```

$E_2 \to T + E$

Algorithm 4.19 : Eliminating left recursion.

INPUT: Grammar G with no cycles or ϵ-productions.

OUTPUT: An equivalent grammar with no left recursion.

METHOD: Apply the algorithm in Fig. 4.11 to G. Note that the resulting non-left-recursive grammar may have ϵ-productions. $\square$

1) arrange the nonterminals in some order $A_1, A_2, \ldots, A_n$.
2) **for** (each i from 1 to n) {
3) **for** (each j from 1 to $i-1$) {
4) replace each production of the form $A_i \rightarrow A_j\gamma$ by the productions $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$, where $A_j \rightarrow \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ are all current A_j-productions
5) }
6) eliminate the immediate left recursion among the A_i-productions
7) }

bool E() {

return E() && match('('); match(')');

}

برای تزریق تخریب

۱- max+ به ابتدای رشته ورودی اشاره کند

۲- S() فراخوانی شود (کمل تزریق)

به نام خدا

مشکلات روش Recursive

ترتیب فراخوانی توابع روی ورودی

مثال ورودی INT * INT

$E \to \overbrace{T}^{E_1} \mid \overbrace{T + E}^{E_2}$

$T \to \underbrace{INT}_{T_1} \mid \underbrace{INT * T}_{T_2} \mid \underbrace{(E)}_{T_3}$

$T \to INT\ T'$

$T' \to *T \mid \varepsilon$

$E \to E_1 \to T \to INT$

match ، true برمی‌گرداند و بنابراین به صورت بازگشتی همه توابع تا E1 ، true برمی‌گردانند و کار متوقف و تمام می‌شود

حالتی که رشته ورودی تمام نشده

مشکل! این روش خطاهای گرامری را درست کار می‌کند و در آن‌ها برای هیچ ترمینال حداقل یک قانون نمی‌تواند true برگرداند و قانون‌ها شروع یک می‌نداشته باشند

راه‌حل فاکتورگیری از چپ گرامر

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \cdots \alpha\beta_n \mid \gamma \rightsquigarrow$ با شروع رشته

صبر کن!

$A \to \alpha A' \mid \gamma$

$A' \to \beta_1 \mid \beta_2 \cdots \mid \beta_n$

گرامر $E \to E+E \mid (E) \mid id$

ترتیب فراخوانی‌ها: E, E, E, ...

مشکل فراخوانی توابع در loop افتاده این مشکل گرامری که left recursion دارد پیش می‌آید

راه‌حل حذف left recursion

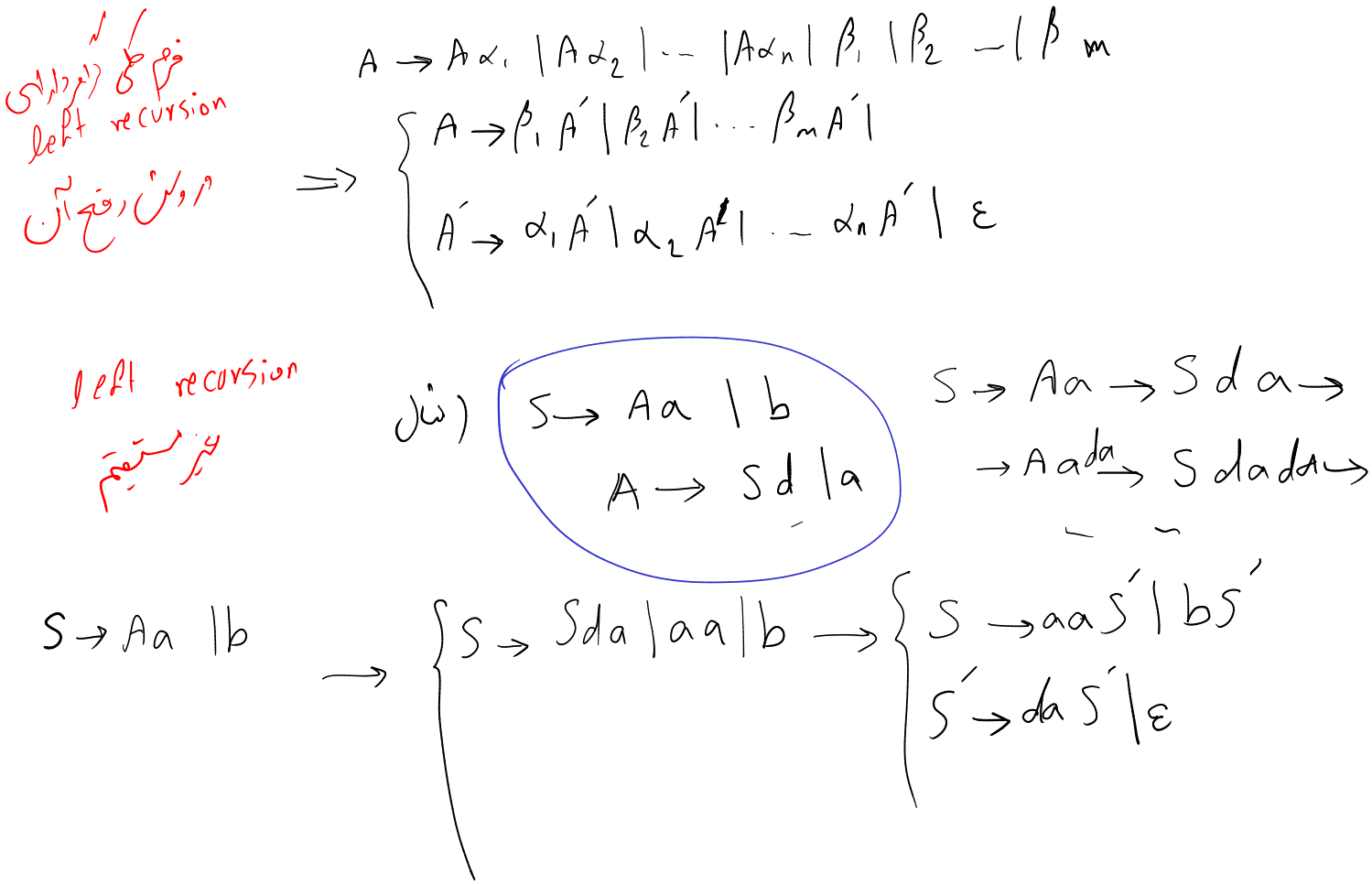
$A \to A\alpha \mid \beta \rightsquigarrow \begin{cases} A \to \beta A' \\ A' \to \alpha A' \mid \varepsilon \end{cases}$

مثال

$E \to E * E \mid (E) \mid id$

$E \to (E)E' \mid id\,E'$

$E' \to *E\,E' \mid \varepsilon$



Algorithm 4.19 : Eliminating left recursion.

INPUT: Grammar G with no cycles or ϵ-productions.

OUTPUT: An equivalent grammar with no left recursion.

METHOD: Apply the algorithm in Fig. 4.11 to G. Note that the resulting non-left-recursive grammar may have ϵ-productions. □

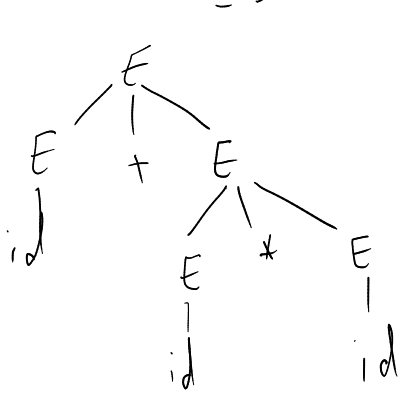
```
1)  arrange the nonterminals in some order A_1, A_2, ..., A_n.
2)  for ( each i from 1 to n ) {
3)      for ( each j from 1 to i - 1 ) {
4)          replace each production of the form A_i → A_j γ by the
                productions A_i → δ_1 γ | δ_2 γ | ··· | δ_k γ, where
                A_j → δ_1 | δ_2 | ··· | δ_k are all current A_j-productions
5)      }
6)      eliminate the immediate left recursion among the A_i-productions
7)  }
```

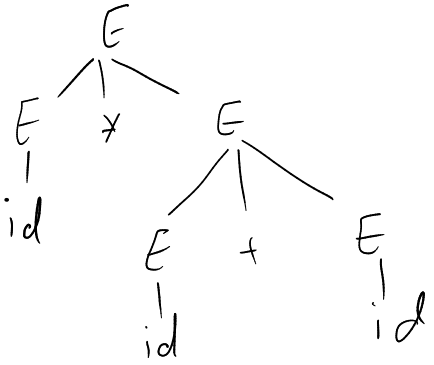
(۳) ابهام در گرامر

مثال) $E \to E+E \mid E-E \mid E*E \mid E/E \mid E\hat{}E \mid -E \mid (E) \mid id$

ورودی id+id*id

دو درخت متفاوت پارس برای این ورودی از گرامر می‌توانند بدست بیایند





حل مشکل

گرامر مورد نظر نباید ابهام داشته باشد می‌توان گرامر را در صورت امکان رفع ابهام کرد

یعنی گرامر مبهم را تبدیل به گرامری کرد که زبان تولید شده توسط گرامر جدید بدون ابهام دقیقا معادل گرامر اول بهم باشد.

رفع ابهام مثال

ترتیب اولویت ها $(E)\ ,\ -\ ,\ \hat{}\ ,\ *\ ,\ /\ ,\ +\ ,\ -$

ترکیب پذیری چپ راست راست چپ

گرامر معادل بدون ابهام

$E \to E+T \mid E-T \mid T$

$T \to T*F \mid T/F \mid F$

$F \to G\hat{}F \mid G$

$G \to -G \mid H$

$H \to (E) \mid id$

مثال دوم برای جلسه بعد

نام ورودی شان

رهنمود گرامر ابهام دارد

پس نمی‌شود گرامر

جدید معادلی که بدون

ابهام است بنویسید

$st \to if\ \ exp\ \ then\ \ st \mid$

$if\ \ exp\ then\ st\ else\ st \mid s$

$exp \to e$

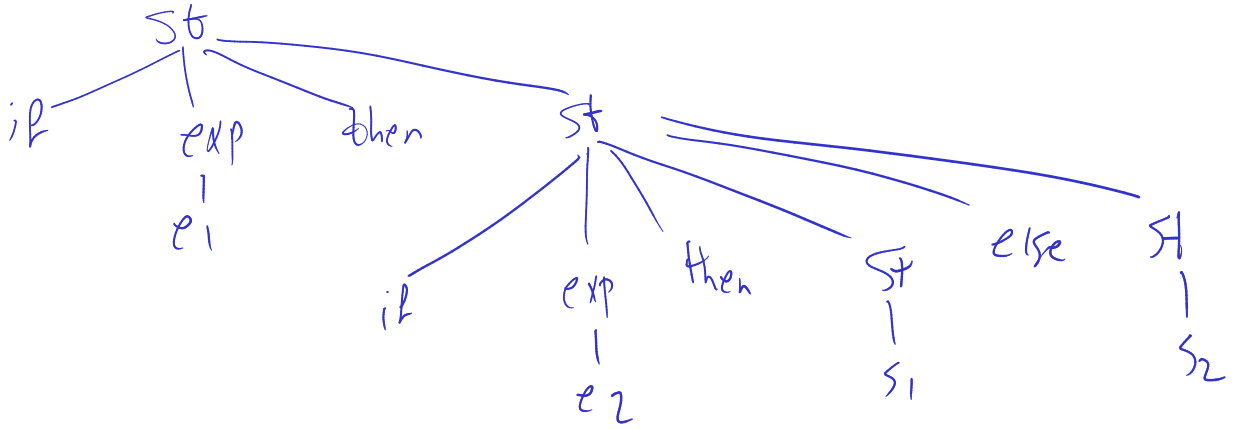
برنامه ضدا

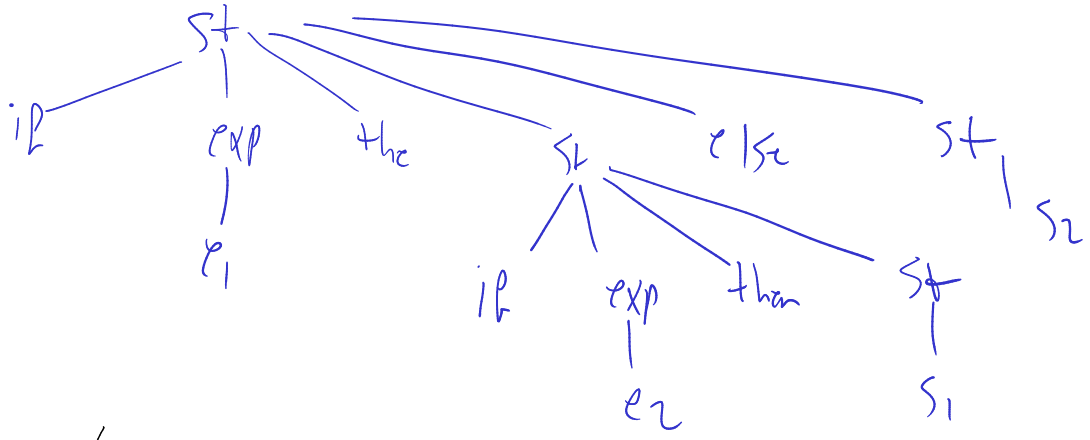
مثال رفع ابهام از گرامر if

st → if exp then st |
if exp then st else st | s

exp → e

مثال) if e_1 then if e_2 then s_1 else s_2



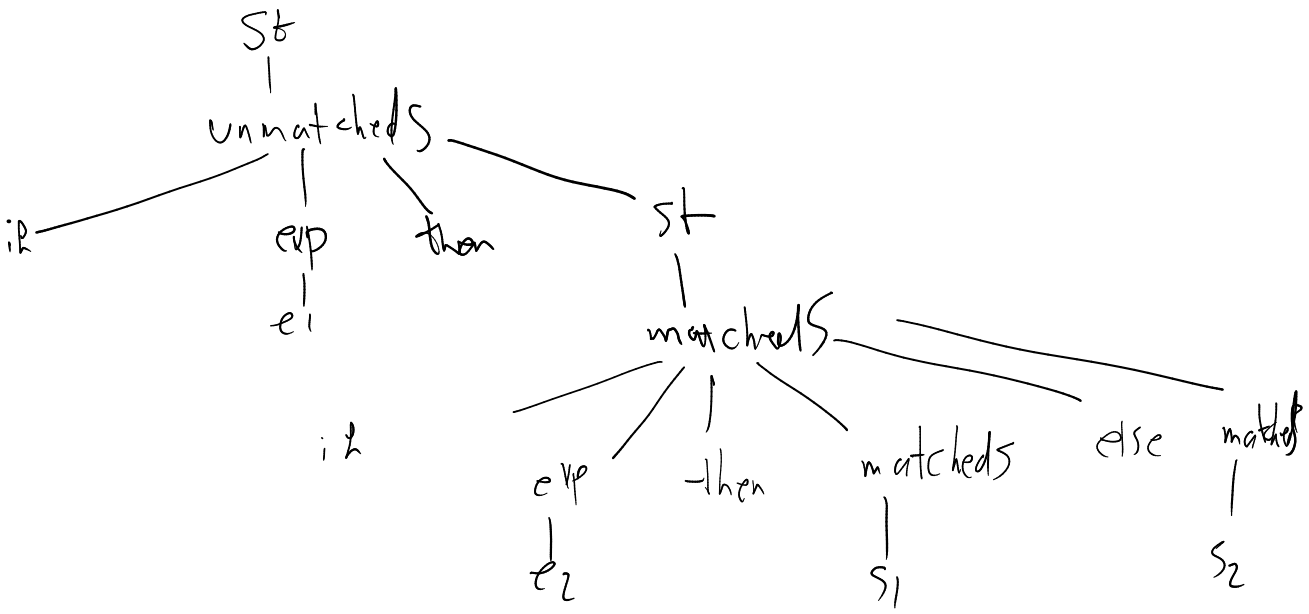


St → matchedS | UnmatchedS

matchedS → if exp then matchedS else matchedS | s

UnmatchedS → if exp then St | if exp then matchedS else UnmatchedS

exp →


St
unmatchedS
if
exp
then
St
e1
matchedS
if
exp
then
matchedS
else
matched
e2
S1
S2

Predictive Recursive Descent

هدف: تجربه به غیرپایانه A رسیدیم و می‌خواهیم از بین production‌های A، production مناسب را انتخاب کنیم

ورودی *next

$A \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_k$

۱- اگر $A \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_k$ باید مشخص کنیم هر کدام از α_iها با چه توکن‌هایی می‌توانند شروع شوند و آن α_i را انتخاب کنیم که شروع‌شونده با توکن جاری ورودی مطابق باشد (first(α_i) را بدست آوریم)

* پس لازم است $first(\alpha_i) \cap first(\alpha_j) = \emptyset$ تا قانون یکتا برای توکن موجود ورودی بدست آید

۲- اگر $A \to \varepsilon$ (یا $A \Rightarrow^* \varepsilon$) در آن صورت بدون مصرف ورودی return کنیم

با مثالی که بعداز A در قانون فراخوانی شده قبل آمده است ادامه می‌دهیم. در این صورت باید توکن جاری ورودی بتواند با follow(A) مطابق شود

$S \Rightarrow^* \beta A \beta' \to \beta \beta'$

$a_1 \dots a_i \dots a_n$ (next ↑ a_i)

* اگر $A \to \varepsilon$ آنگاه $first(\alpha_i) \cap follow(A) = \emptyset$

روش predictive

برای هر غیرپایانه A تابع زیر را می‌توان نوشت

$A \to \alpha_1 \mid \dots \mid \alpha_k \mid \varepsilon$

*next، توکن جاری ورودی

```
A() {
  if (*next ∈ first(α1)) continue parsing the input with α1
  else if (*next ∈ first(α2))            α2
  else . . . first(αk)                   αk
  else if (*next ∈ follow(A))
      return ;
  } else syntax Error ;
```

مثال) $E \rightarrow E+T \mid E-T \mid T$ صورت LR

$T \rightarrow id$

$E \rightarrow TE'$

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

$T \rightarrow id$

```
E() {
  if(*next ∈ first(TE'))
    T(); E'()
  else
    Syn Err
}
T() {
  if(*next == id)
    match('id')
  else Syn Err
}
```

```
E'() {
  if(next ∈ first(+TE'))
    match('+') T() E'()
  elseif(next ∈ first(-TE'))
    match('-'); T() E'()
  else if(*next ∈ follow(E'))
    return
  else syn Err
}
```

به نام خدا

شرایط استفاده از الگوریتم بدون عقب گرد برای تجزیه Top down

برای هر α و β از A-productionی $A \to \alpha \mid \beta$ شرایط زیر برقرار باشد:

۱- $first(\alpha) \cap first(\beta) = \phi$

۲- $\beta \Rightarrow^* \varepsilon$ آنگاه باید $follow(A) \cap first(\alpha) = \phi$ باشد

* این شرایط، شرایط لازم و کافی برای LL(1) بودن یک گرامر است.

تعریف گرامر LL(1)

به تجزیه گری می گوییم که می تواند در هر قدم، قاعده صحیح را برای هر NonTerminal انتخاب کند (بدون backTrack) تجزیه گر پیشگو گوییم. اگر در هر مرحله با توجه به یک توکن از ورودی، قاعده صحیح انتخاب شود تجزیه گر را LL(1) یا یا گرامر

LL(1) : Left to right - Left most derivation

روش های تبدیل گرامر به LL(1)

۱- حذف left recursion

۲- فاکتور گیری از چپ

۳- تبدیل گرامر به گرامر غیر مبهم

- استخراج زبان گرامر و تعریف گرامر جدید LL(1)

محاسبه first و follow

$first(\alpha)$ مجموعه سمبل‌های انتهایی که ابتدای رشته‌های به‌دست‌آمده از α قرار می‌گیرند.

روش محاسبه $first(\alpha)$

- اگر α یک سمبل انتهایی باشد $first(\alpha)=\alpha$
- اگر α یک غیرانتهایی باشد، $\alpha \to Y_1 Y_2 \dots Y_k$ یکی از قوانین α باشد
- اگر برای یک i، $a \in first(Y_i)$، برای همه $j<i$، $\varepsilon \in first(Y_j)$ آنگاه $a \in first(\alpha)$
- اگر به ازای همه $1<i\le k$ $\varepsilon \in first(Y_i)$ آنگاه $\varepsilon \in first(\alpha)$
- اگر $\alpha \to \varepsilon$ آنگاه $\varepsilon \in first(\alpha)$

(مثال)

$S \to ABCDE \mid f$
$A \to aA \mid B$
$B \to bB \mid \varepsilon$
$C \to cC \mid d$
$D \to dD \mid Be$
$E \to gE \mid \varepsilon$

$first(S)=\{f,a,b,c,d\}$
$first(A)=\{a,b,\varepsilon\}$
$first(B)=\{b,\varepsilon\}$
$first(C)=\{c,d\}$
$first(D)=\{d,b,e\}$
$first(E)=\{g,\varepsilon\}$

روش بدست آوردن follow

- اگر S سمبل شروع باشد $\$ \in follow(S)$
- اگر قانونی به صورت $A \to \alpha B \beta$، همه سمبل‌های عضو $first(\beta)$ غیر از ε در $follow(B)$ قرار می‌گیرند
- اگر قانونی به صورت $A \to \alpha B$ یا $A \to \alpha B \beta$ و آنگاه $\beta \Rightarrow^* \varepsilon$ آنگاه $follow(A) \subset follow(B)$

$C \to \beta_1 A \beta_2 \Rightarrow^* \beta_1 \alpha B \beta_2$

پس همه آنچه در $follow(A)$ قرار می‌گیرد می‌تواند به‌واسطه این جمله در $follow(B)$ هم قرار گیرد.

مثال)

$S \to ABCSDE \mid \varepsilon$

$A \to aA \mid B$

$B \to bB \mid \varepsilon$

$C \to cC \mid d$

$D \to dD \mid Be$

$E \to gE \mid \varepsilon$

① $First(BCSDE) \subseteq Follow(A)$

$A \to B \Rightarrow Follow(A) \subseteq Follow(B)$ ②

③ $First(SDE) \subseteq Follow(C)$

$S \to ABCSDE \Rightarrow Follow(S) \subseteq Follow(E)$ ④

$First(E) \subseteq Follow(D)$ ⑤

$S \to ABCSDE \Rightarrow First(DE) \subseteq Follow(S)$

$E \to \varepsilon \Rightarrow Follow(S) \subseteq Follow(D)$

$Follow(S) = \{\$, d, h, e\}$

$Follow(A) = \{b, c, d\}$ ①

$Follow(B) = \{b, c, d\}$ ②

$Follow(C) = \{f, a, b, c, d\}$ ③

$Follow(E) = \{\$, d, h, e\}$ ④

$Follow(D) = \{g, \$, d, h, e\}$ ⑤

$First(bB) \cap Follow(B) = \{b\} \cap \{b, c, d\} = \{b\}$

$B \to bB \mid \varepsilon$

گرامر LL(1) نیست

روش پارس ورودی با کمک جدول LL(1)

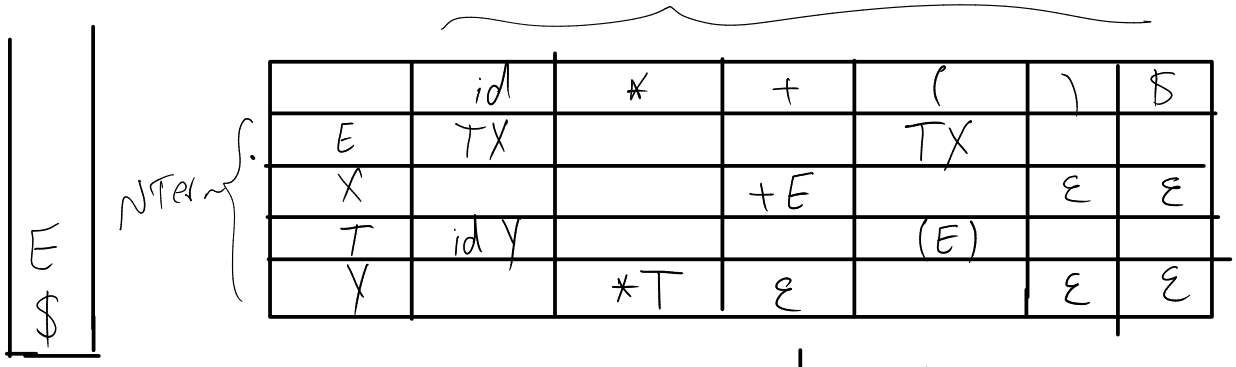
مثال) $E \to T+E \mid T$
$T \to id \mid id*T \mid (E)$

بعد از فاکتورگیری چپ:

$E \to TX$
$X \to +E \mid \varepsilon$
$T \to idY \mid (E)$
$Y \to *T \mid \varepsilon$

ورودی: $id+id*id$

$E \to TX \to idYX \to idX \to id+E$
$\to id+TX \to id+idYX \to id+id*TX \to$
$id+id*idYX \to id+id*idX \to$
$id+id*id$



Stack	ورودی	action
$E\$$	$id+id*id\$$	$E \to TX$
$TX\$$	$id+id*id\$$	$T \to idY$
$idYX\$$	$id+id*id\$$	match(id)
$YX\$$	$+id*id\$$	$Y \to \varepsilon$
$X\$$	$+id*id\$$	$X \to +E$
$+E\$$	$+id*id\$$	match(+) ✓
$E\$$	$id*id\$$	$E \to TX$
$TX\$$	$id*id\$$	$T \to idY$
$idYX\$$	$id*id\$$	match(id)
$YX\$$	$*id\$$	$Y \to *T$
$*TX\$$	$*id\$$	match(*)
$TX\$$	$id\$$	$T \to idY$
$idYX\$$	$id\$$	match(id)
$YX\$$	$\$$	$Y \to \varepsilon$
$X\$$	$\$$	$X \to \varepsilon$
$\$$	$\$$	accept

الگوریتم طی تجزیه با توجه به جدول LL(1) یا یک گرامر LL(1)

initialize stack < S $ >
* next = next token at input (initialize by first token at input)

Repeat
case stack of
< X rest > : if T [X, *next] = Y1 — Yn
X ∈ NonTerminals then stack ← < Y1 -- Yn rest > ;
else error ;

< t rest > : if t = *next
t ∈ Terminals then stack ← < rest >
else error ;

Until stack = < >

Algorithm 4.31: Construction of a predictive parsing table.

INPUT: Grammar G.

OUTPUT: Parsing table M.

METHOD: For each production $A \to \alpha$ of the grammar, do the following:

1. For each terminal a in FIRST(α), add $A \to \alpha$ to $M[A, a]$.
2. If ϵ is in FIRST(α), then for each terminal b in FOLLOW(A), add $A \to \alpha$ to $M[A, b]$. If ϵ is in FIRST(α) and \$ is in FOLLOW(A), add $A \to \alpha$ to $M[A, \$]$ as well.

مثال S → i E t S | i E t S e S | a
E → b