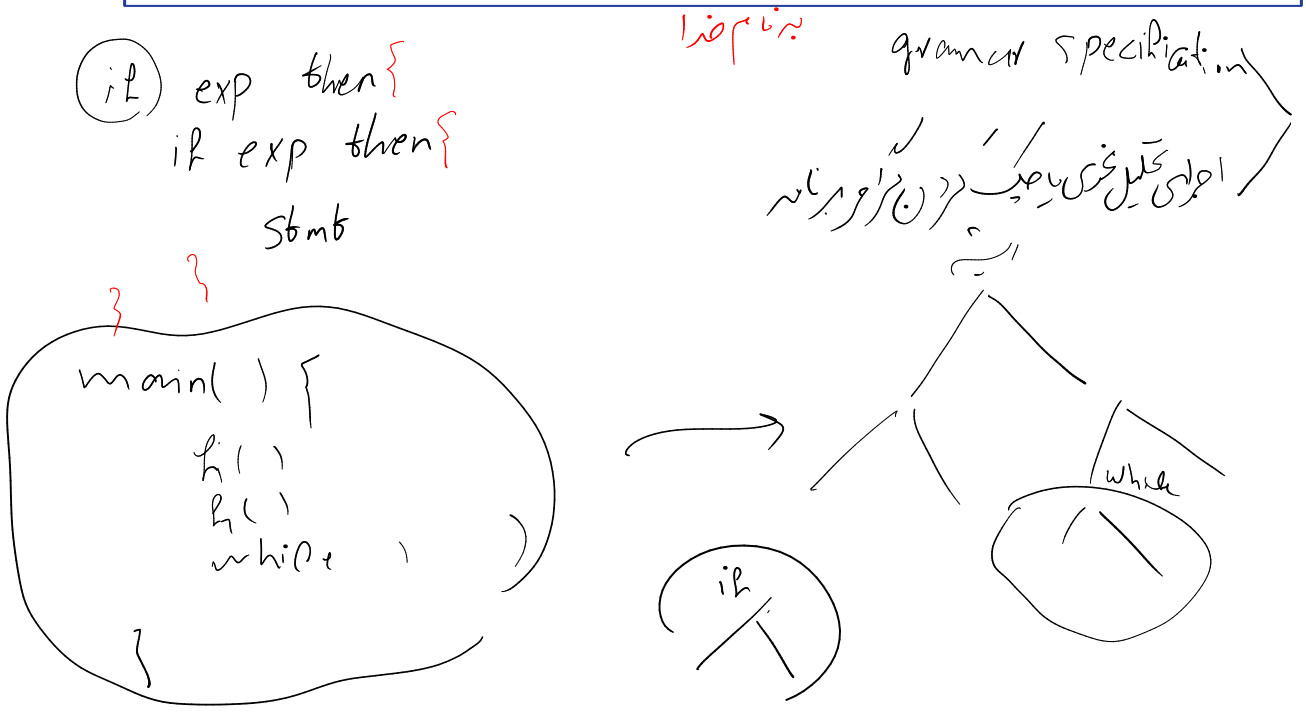
به نام خدا



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

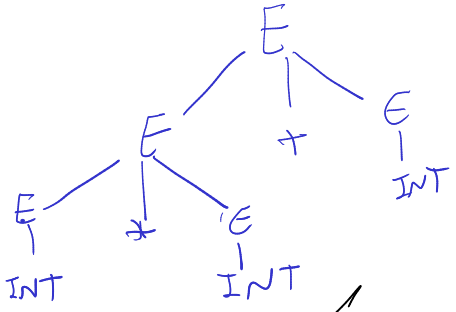
ضعف RE برای تشخیص ساختارهای گرامری زبان‌های برنامه‌نویسی؟

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

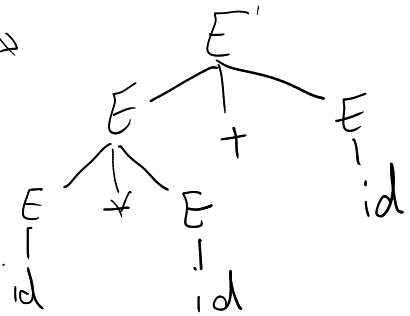
۳- مرحله ۲ را تکرار می‌کنیم به طوری که نهایتا خروجی سمبل غیرپایانی دیگری نداشته باشد و رشته‌های با رشته مورد نظر تطابق داشته باشد

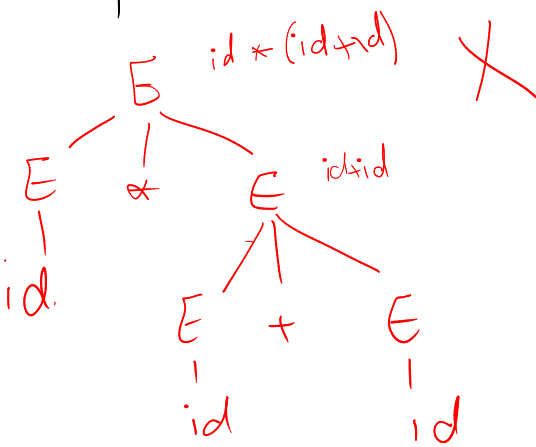
$E \rightarrow E+E \rightarrow E*E+E \rightarrow id*E+E \rightarrow id*id+E \rightarrow id*id+id$

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

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

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
    }
}
```

$X_i \in T \cup N$

$A \to X_1 X_2 \dots X_k$

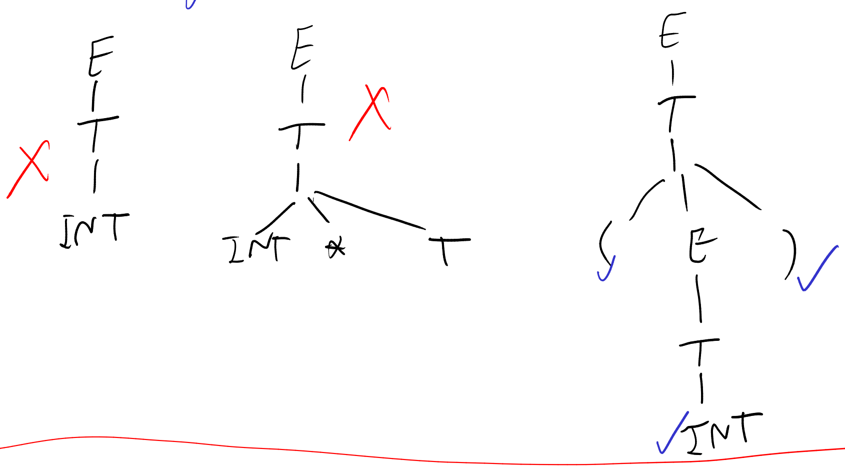
*next

ورودی $a_1 a_2 \dots a_n$

مثال) $E \rightarrow \overset{E_1}{T} \mid \overset{E_2}{T + E}$

$T \rightarrow \underset{T_1}{INT} \mid \underset{T_2}{INT * T} \mid \underset{T_3}{(E)}$

رشته ورودی (INT)



پارس تمام می‌شود

روش پارس Recursive

- فرض می‌کنیم که توکن‌ها از lexer بدست می‌آمده‌اند
- متغیر سراسری next به ورودی بعدی اشاره می‌کند
- به ازای هر non terminal (S) یک تابع boolean که چک کردن قانون Sn جهت match شدن
- همه قوانین را با پیاده کردن یک match امکان‌پذیر کنیم

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

$a_1 \overset{\downarrow next}{\_\_\_} a_n$

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

```
bool E1( ) {
  return T()
}
bool E2( ){
  return (T() && match('+') && E())
}
```

$E_2 \rightarrow T + E$

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

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. □

```
arrange the nonterminals in some order A_1, A_2, ..., A_n.
for ( each i from 1 to n ) {
    for ( each j from 1 to i - 1 ) {
        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
    }
    eliminate the immediate left recursion among the A_i-productions
}
```

```
bool
    return (match('(') && E() && match(')'));
}
```

برای شروع تجزیه

۱- next به ابتدای رشته ورودی اشاره کند

۲- S() فراخوانی شود (عمل شروع)

به نام خدا

مثال) ورودی INT*INT

$E \to \overset{E_1}{T} \mid \overset{E_2}{T+E}$

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

$T \to INT\ T'$

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

مشکلات روش Recursive

ترتیب فراخوانی توابع روی ورودی

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

match ، true برمی‌گرداند و بنابراین به صورت بازگشتی همه توابع تا E، true برمی‌گردانند و کار متوقف و تمام می‌شود در حالی که رشته ورودی تمام نشده

(۱) ریشه مشکل: این روش فقط برای گرامرهایی درست کار می‌کند که در آن برای هر غیرترمینال حداکثر یک قانون بتواند true برگرداند و قانون‌ها شروع یکسانی نداشته باشند

راه حل: فاکتورگیری از چپ گرامر

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \alpha\beta_n \mid \gamma$ ← با α شروع شده

جایگزین

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

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

(۲) تکرار $E \to E+E \mid (E) \mid id$

ترتیب فراخوانی‌ها: $E, E, E, \dots$

مشکل: فراخوانی توابع در loop می‌افتد. این مشکل در گرامری که left recursion دارد پیش می‌آید

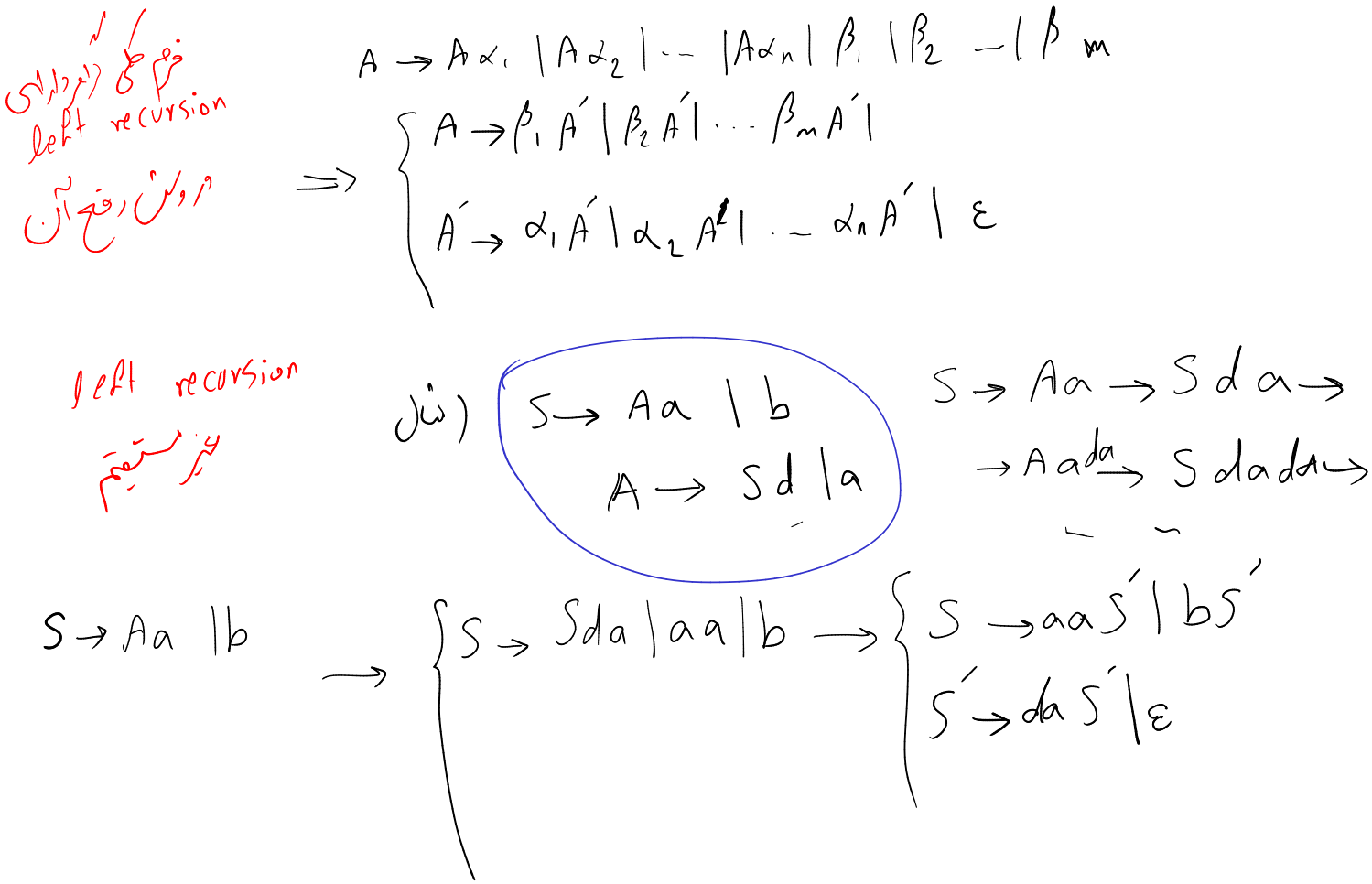
راه حل: حذف left recursion

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

مثال) $E \to E*E \mid (E) \mid id$

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

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



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. □

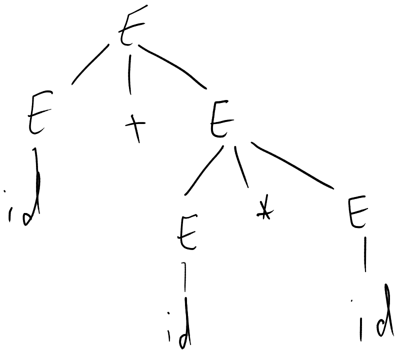
```
1)  arrange the nonterminals in some order A_1, A_2, ..., A_n.
2)  for ( each i from 1 to n ) {
3)      for ( each j from 1 to i - 1 ) {
4)          replace each production of the form A_i → A_j γ by the
                productions A_i → δ_1 γ | δ_2 γ | ··· | δ_k γ, where
                A_j → δ_1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

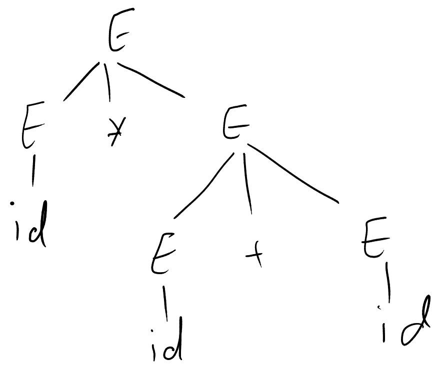
(۳) ابهام در گرامر

مثال) $E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid E^\wedge E \mid -E \mid (E) \mid id$

ورودی) $id+id*id$

دو درخت متفاوت پارس برای این ورودی از گرامر می تواند بدست بیاید





حل مشکل

گرامر مورد نظر نباید ابهام داشته باشد می توان 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 مبهم باشد.

رفع ابهام مثال) ترتیب اولویت ها: (E) , $-$, $^\wedge$, $*$, $/$, $+$, $-$

ترتیب شرکت پذیری: راست ، راست ، راست ، چپ

گرامر معادل بدون ابهام:

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

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

$F \rightarrow G^\wedge F \mid G$

$G \rightarrow -G \mid H$

$H \rightarrow (E) \mid id$

مثال دوم برای جلسه بعد: با یک ورودی نشان دهید گرامر ابهام دارد سپس سعی کنید گرامر جدید معادلی که بدون ابهام است بنویسید

$st \rightarrow$ if exp then st | if exp then st else st | s

$exp \rightarrow e$