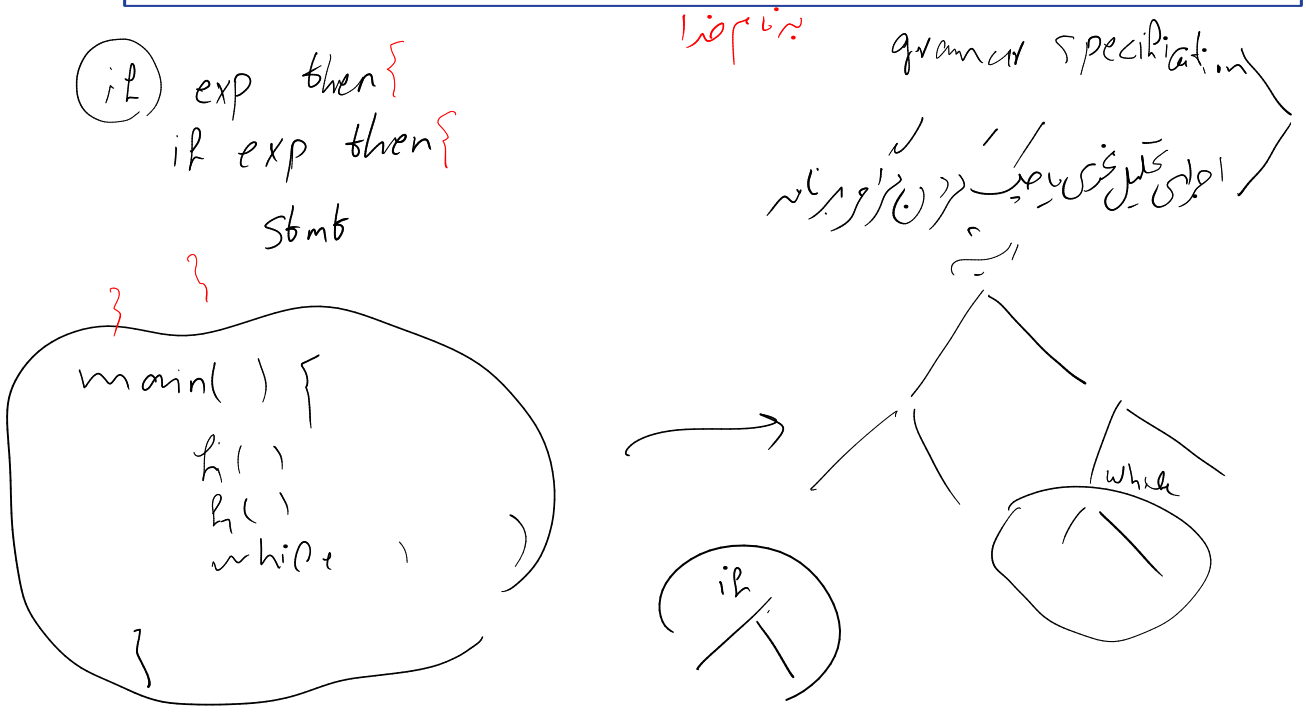
به نام خدا



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

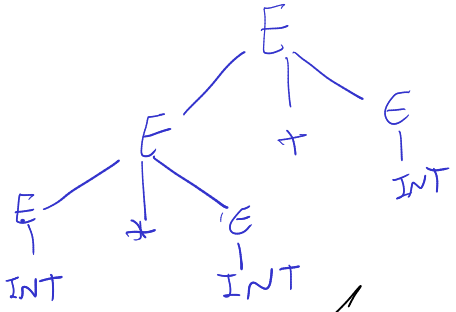
ضعف RE برای تشخیص ساختارهای گرامری زبان‌های برنامه‌نویسی؟

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

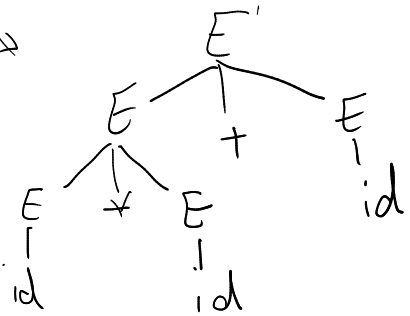
۳- مرحله ۲ را تکرار می‌کنیم به طوری که نهایتا خروجی سمبل غیرپایانی دیگری نداشته باشد و رشته‌های با رشته مورد نظر تطابق داشته باشد

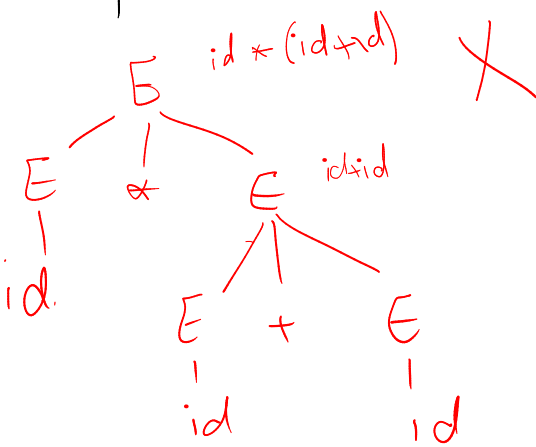
$E \rightarrow E+E \rightarrow E*E+E \rightarrow id*E+E \rightarrow id*id+E \rightarrow id*id+id$

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

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

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
    }
}
```

$X_i \in T \cup N$

$A \to X_1 X_2 \dots X_k$

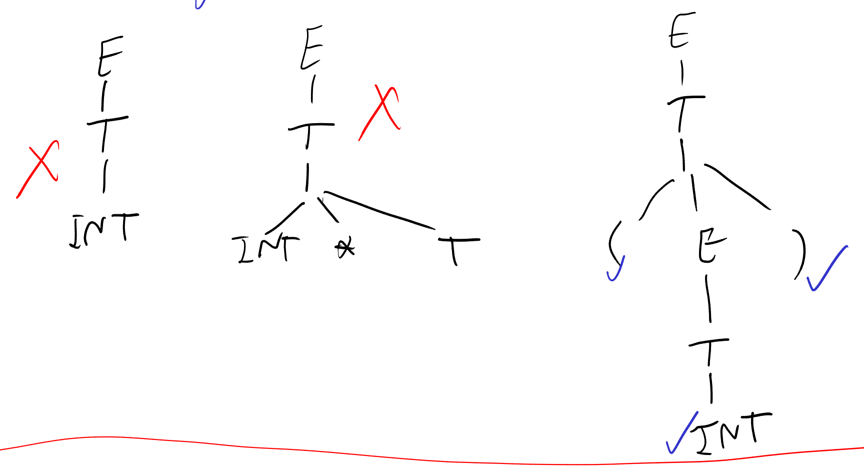
*next

ورودی $a_1 a_2 \dots a_n$

مثال) $E \rightarrow T \mid T + E$ (E_1: T, E_2: T + E)

$T \rightarrow INT \mid INT * T \mid (E)$ (T_1: INT, T_2: INT * T, 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 —— a_n (↓ next)

```
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() ) )
}
```

```
bool T1 ( ) {
    return match( INT)
}
bool T2( ) {
    return ( match(INT) && match ('*') && T( ) )
}
bool T3 ( ) {
    return ( match ('(') && E( ) && match (')') );
}
```

برای تمرین تجزیه

۱- next به ابتدای رشته ورودی اشاره کند

۲- S( ) فراخوانی شود (کمل تمرین)