# اصول طراحی کامپایلرها

حسین کارشناس

دانشکده مهندسی کامپیوتر

ترم اول ۹۸ – ۹۷

# تحلیل دستوری (تجزیه)

# تحلیل دستوری

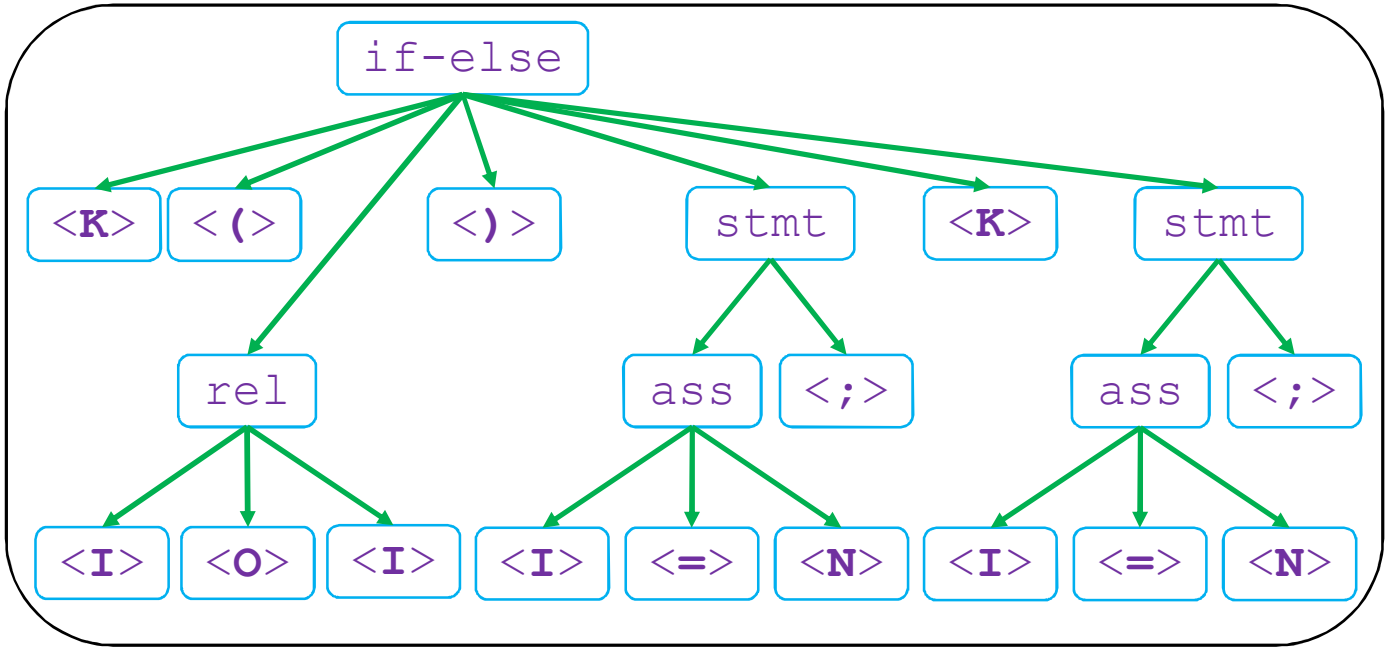
- گرامرهای مستقل از متن
- روش‌های تجزیه
- تجزیه نزول بازگشتی
- تجزیه پیشگویانه
- تجزیه انتقال کاهش
- تجزیه LR
  - SLR، CLR، LALR

# تحلیل دستوری

- بررسی صحت ساختار برنامه با توجه به قوانین دستوری زبان
  - هدف شناسایی برنامه‌های خوش شکل (well-formed) است
  - ورودی: دنباله‌ای از نمادها (tokens)
  - خروجی: درخت تجزیه (parse tree)

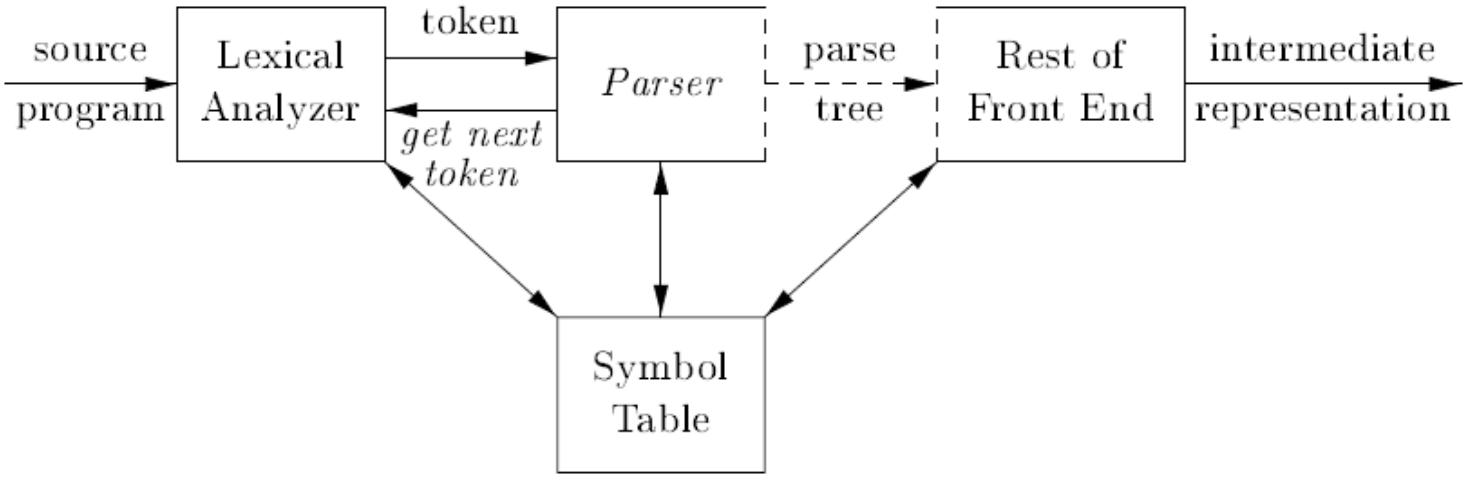
```
if (i == j)
    Z = 0;
else
    Z = 1;
```

```
K ( I
O I )
I = N
; K I
= N ;
```



# تحلیل دستوری

- ارتباط با سایر مراحل کامپایلر




- امکان انجام سایر مراحل کامپایل در حین تجزیه ورودی
  - اطلاعات حاصل از تجزیه (درخت تجزیه) به صورت ضمنی به سایر مراحل داده می‌شود
  - ترجمه با راهنمایی دستور زبان (syntax-directed translation)

# تحلیل دستوری

- وظیفه تجزیه‌گر (parser) تفکیک دنباله نمادهای معتبر از نامعتبر است
  - هر دنباله‌ای از نمادها یک برنامه معتبر (valid) نیست
  - نیاز به توصیف دنباله نمادهای معتبر

- توصیف ساختارهای مورد استفاده در برنامه‌نویسی
  - استفاده از زبان‌های منظم؟
    - مثال: زبان $\{(^i)^i \mid i \geq 0\}$
      - کاربرد: بررسی متعادل بودن تعداد پرانتزهای باز و بسته
  - ماشین‌های متناهی قادر به بررسی (شمارش) برابری در ورودی‌ها نیستند

# تحلیل دستوری

- برخی ساختارهای متداول برنامه‌نویسی
  - وجود ساختارهای تو در تو
    - مثال: در بدنه دستور if یک دستور if دیگر می‌تواند وجود داشته باشد
    - نیاز به بررسی تعادل در کاربرد ساختارهای تو در تو
  - وجود ساختارهای بازگشتی
    - مثال: در زبان برنامه‌نویسی C
      - یک برنامه حاوی چندین تابع
      - هر تابع دارای چندین تعریف (declaration) و دستور (statement)
      - هر دستور ترکیبی از عبارت‌ها (expressions)
      - یک عبارت می‌تواند ترکیبی از عبارت‌های دیگر باشد: $a + 2 < b$

# گرامرهای مستقل از متن

- توصیف با گرامرهای مستقل از متن (context-free grammars)
  - امکان توصیف نظام‌مند (systematic) قوانین دستوری زبان‌های برنامه‌نویسی
  - پشتیبانی از بسیاری از ساختارهای متداول زبان‌های برنامه‌نویسی

- تعریف گرامر مستقل از متن
  - نشانه‌های پایانی
  - نشانه‌های غیرپایانی
  - نشانه‌ی شروع
  - مجموعه قواعد تولید (production rules)

# گرامرهای مستقل از متن

- توصیف برخی قوانین زبان‌های برنامه‌نویسی
  - مثال: گرامر پرانتزهای باز و بسته متعادل

$$S \rightarrow ( S )$$
$$S \rightarrow \epsilon$$

  - مثال: نمونه‌ای از گرامر دستورهای انشعاب و عبارت‌های شرطی

$$\begin{array}{lcl} stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\ & | & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\ & | & \epsilon \\ expr & \rightarrow & term \textbf{ relop } term \\ & | & term \\ term & \rightarrow & \textbf{id} \\ & | & \textbf{number} \end{array}$$

  - مثال: نمونه‌ای از گرامر عبارت‌های ریاضی

$$E \rightarrow E + E \mid E * E \mid - E \mid ( E ) \mid \textbf{id}$$

# گرامرهای مستقل از متن

- اشتقاق (derivation)
  - بکارگیری دنباله‌ای از گام‌های (steps) اشتقاق با استفاده از قواعد تولید گرامر
  - یک گام اشتقاق
    - جایگزینی نشانه غیرپایانی سمت چپ یک قاعده تولید با نشانه‌های سمت راست آن
  - اشتقاق یک رشته از نشانه شروع گرامر: $S \overset{*}{\Rightarrow} w$
    - مثال: اشتقاق رشته $-(\mathrm{id} + \mathrm{id})$ از گرامر زیر

$$E \rightarrow E + E \mid E * E \mid - E \mid ( E ) \mid \mathbf{id}$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$$

  - رشته جمله‌ای (sentential form): حاوی نشانه‌های پایانی و غیرپایانی
    - معمولاً با حروف کوچک یونانی نشان داده می‌شود

# گرامرهای مستقل از متن

- ترتیب جایگزینی نشانه‌های غیرپایانی در یک رشته جمله‌ای
  - اشتقاق چپ (leftmost): سمت چپ‌ترین نشانه غیرپایانی انتخاب می‌شود
  - اشتقاق راست (rightmost): سمت راست‌ترین نشانه غیرپایانی انتخاب می‌شود
    - اشتقاق استاندارد (canonical)
- زبان یک گرامر
  - مجموعه تمام رشته‌های قابل اشتقاق از نشانه شروع یک گرامر

$$\{a_1 \cdots a_n | \forall_i \, a_i \in T, S \overset{*}{\Rightarrow} a_1 \cdots a_n\}$$

  - نشانه‌های پایانی گرامر در زبان‌های برنامه‌نویسی باید دسته نمادهای شناسایی شده در مرحله تحلیل واژه‌ای باشند

# گرامرهای مستقل از متن

- سوال: کدامیک از رشته‌ها عضو زبان گرامر داده شده است؟

$$S \rightarrow aXa$$
$$X \rightarrow \varepsilon \mid bY$$
$$Y \rightarrow \varepsilon \mid cXc$$

- ☐ abcba
- ☐ acca
- ☐ aba
- ☐ abcbcba

# گرامرهای مستقل از متن

- سوال: کدامیک از رشته‌ها یک اشتقاق معتبر از گرامر داده شده است؟

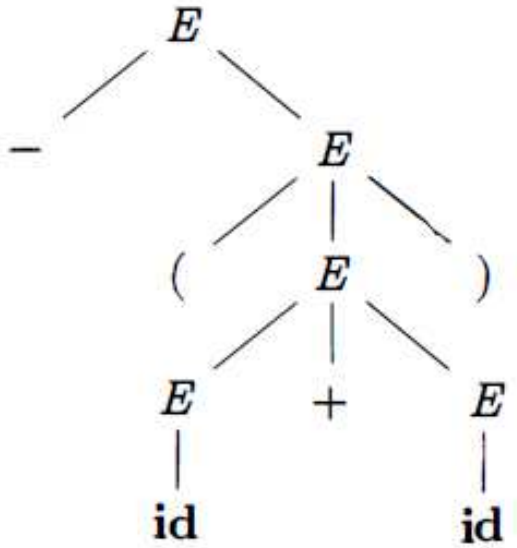
$$S \rightarrow aXa$$
$$X \rightarrow \varepsilon \mid bY$$
$$Y \rightarrow \varepsilon \mid cXc \mid d$$

- S
  aXa
  abYa
  acXca
  acca
- S
  aa
- S
  aXa
  abYa
  abcXca
  abcbYca
  abcbdca
- S
  aXa
  abYa
  abcXcda
  abccda

# گرامرهای مستقل از متن

- درخت تجزیه
- نمایش گرافیکی اشتقاق به صورت درختی (سلسله‌مراتبی)
- گره متناظر با نشانه غیرپایانی سمت چپ قاعده تولید با یال به گره‌های متناظر با نشانه‌های سمت راست آن قاعده تولید به عنوان فرزند وصل می‌شوند
- مثال:

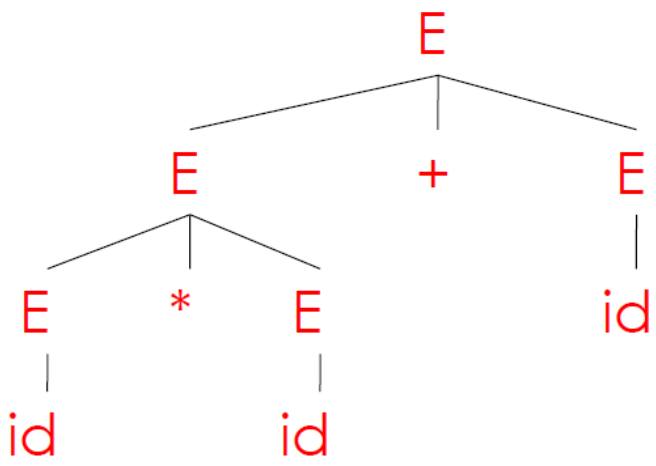
$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$$



# گرامرهای مستقل از متن

- درخت تجزیه (ادامه)
  - نشان دهنده ترتیب انجام عملیات
    - پیمایش چپ به راست گره‌های پایانی دنباله نمادهای ورودی را تولید می‌کند
  - برای هر درخت تجزیه می‌تواند چندین اشتقاق معادل وجود داشته باشد
  - هر درخت تجزیه دارای دقیقاً یک اشتقاق چپ و راست معادل است

$$
\begin{aligned}
& E \\
\rightarrow\ & E+E \\
\rightarrow\ & E * E+E \\
\rightarrow\ & id * E + E \\
\rightarrow\ & id * id + E \\
\rightarrow\ & id * id + id
\end{aligned}
$$




$$
\begin{aligned}
& E \\
\rightarrow\ & E+E \\
\rightarrow\ & E+id \\
\rightarrow\ & E * E + id \\
\rightarrow\ & E * id + id \\
\rightarrow\ & id * id + id
\end{aligned}
$$

# گرامرهای مستقل از متن

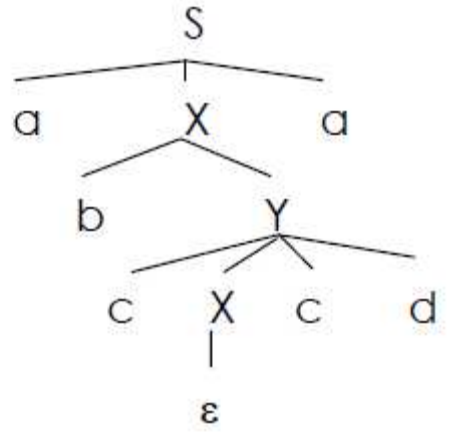
- سوال: کدامیک از درخت‌های تجزیه زیر برای گرامر داده شده معتبر است؟

$S \rightarrow aXa$

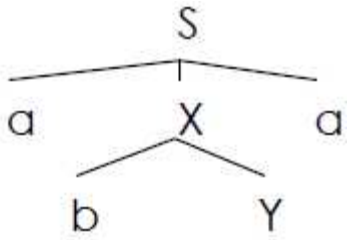
$X \rightarrow \varepsilon \mid bY$

$Y \rightarrow \varepsilon \mid cXc \mid d$

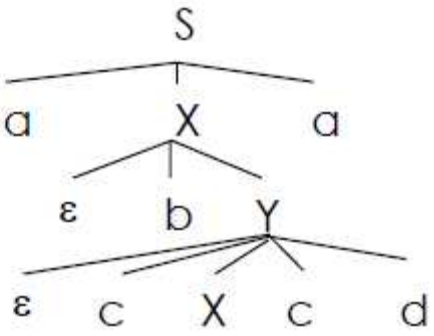
○



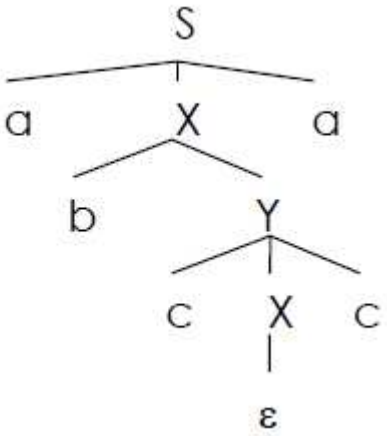

○




○



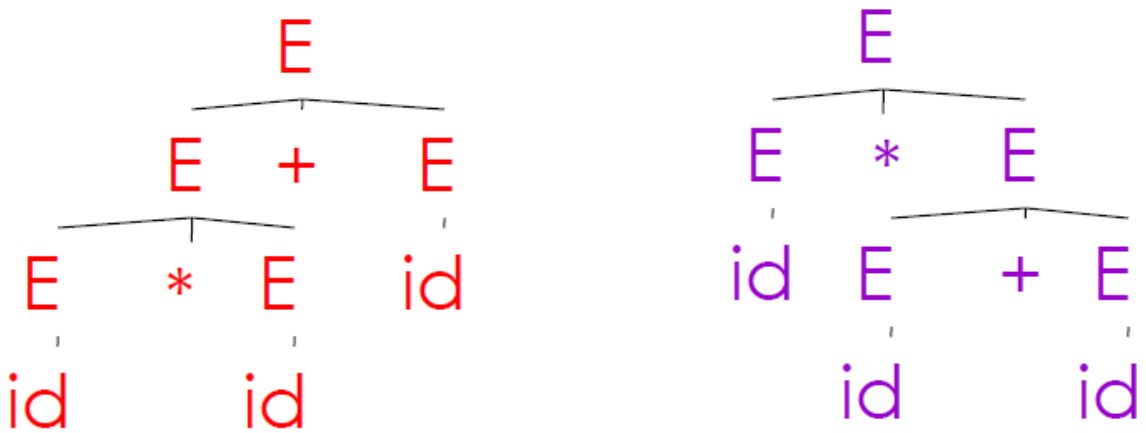

○



# گرامرهای مستقل از متن

- ابهام (ambiguity) در گرامر
  - برای حداقل یک رشته بیش از یک درخت تجزیه قابل تولید باشد
    - بیش از یک اشتقاق چپ یا اشتقاق راست وجود داشته باشد
  - مثال: درخت تجزیه برای رشته id * id + id با گرامر زیر

$$E \rightarrow E + E \mid E * E \mid ( E ) \mid \mathbf{id}$$



# ابهام در گرامرهای مستقل از متن

- یک ویژگی مشکل‌ساز در پیاده‌سازی تجزیه‌گرها
  - معنی بعضی از برنامه‌ها قابل تفسیر است
  - کامپایلر باید خود یکی از تفاسیر (معانی) برنامه را برای تجزیه انتخاب کند
    - رویکرد نامطلوب!
- یک راهکار: رفع ابهام از گرامر
  - بازنویسی گرامر بصورت غیر مبهم

$$E \rightarrow E + E \mid E * E \mid ( E ) \mid \mathbf{id}$$

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow ( E ) \mid \mathbf{id} \end{aligned}$$

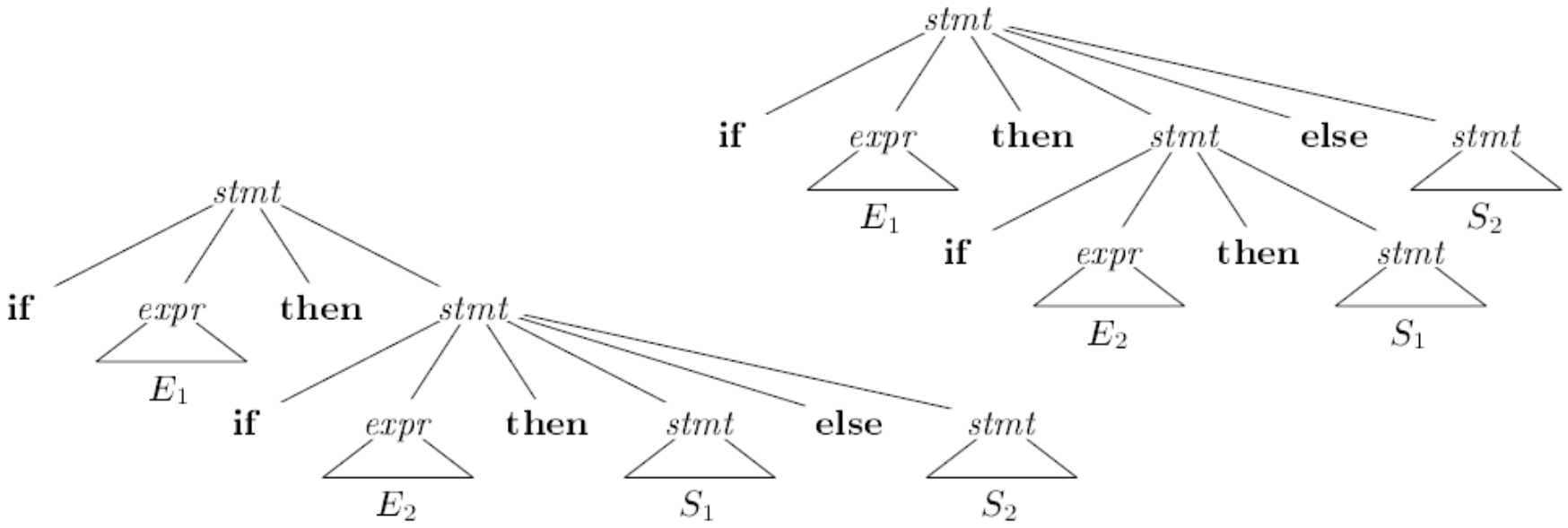
# ابهام در گرامرهای مستقل از متن

- مثال: else آویزان (dangling)

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & | & \textbf{other}
\end{array}
$$

- نحوه تجزیه جمله روبرو؟

$$\textbf{if } E_1 \textbf{ then if } E_2 \textbf{ then } S_1 \textbf{ else } S_2$$



# ابهام در گرامرهای مستقل از متن

- مثال: else آویزان (ادامه)
  - بازنویسی گرامر بصورت غیر مبهم

$$
\begin{array}{rcl}
stmt & \rightarrow & matched\_stmt \\
 & | & open\_stmt \\
matched\_stmt & \rightarrow & \textbf{if } expr \textbf{ then } matched\_stmt \textbf{ else } matched\_stmt \\
 & | & \textbf{other} \\
open\_stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } matched\_stmt \textbf{ else } open\_stmt
\end{array}
$$

- نتیجه: هر else متناظر با نزدیکترین then (if) است
  - نمونه‌ای از یک قانون ابهام‌زدا (disambiguating)

# ابهام در گرامرهای مستقل از متن

- سوال
  - کدامیک از گرامرهای زیر مبهم هستند؟

- ☐ $S \rightarrow SS \mid a \mid b$
- ☐ $E \rightarrow E + E \mid id$
- ☐ $S \rightarrow Sa \mid Sb \mid \varepsilon$
- ☐ $E \rightarrow E' \mid E' + E$

  $E' \rightarrow -E' \mid id \mid (E)$

# ابهام در گرامرهای مستقل از متن

- سوال
- کدامیک از گرامرهای زیر معادل غیر مبهم برای گرامر مبهم داده شده است؟

$$S \rightarrow SS \mid a \mid b$$

- $S \rightarrow Sa \mid Sb \mid a \mid b$
- $S \rightarrow SS'$
  $S' \rightarrow a \mid b$
- $S \rightarrow S \mid S'$
  $S' \rightarrow a \mid b$
- $S \rightarrow Sa \mid Sb$

# ابهام در گرامرهای مستقل از متن

- راهکار چیست؟
  - بازنویسی تمام گرامرهای مبهم بصورت غیرمبهم؟

- در حالت کلی تبدیل خودکار گرامر مبهم به غیرمبهم همیشه امکان‌پذیر نیست
- امکان توصیف طبیعی‌تر قوانین دستوری زبان با استفاده از گرامرهای مبهم
  - بعضی مواقع گرامرهای مبهم موجزتر، ساده‌تر (قابل درک‌تر) و تغییر آنها راحت‌تر است
  - سرعت بیشتر تجزیه با گرامرهای مبهم

$$E \rightarrow E + E \mid E * E \mid ( E ) \mid \mathbf{id}$$

---

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow ( E ) \mid \mathbf{id} \end{aligned}$$

# ابهام در گرامرهای مستقل از متن

- بکارگیری گرامرهای مبهم برای تجزیه
  - نیاز به ساز و کارهای برای ابهام زدایی در تفسیر دارند

- قوانین ابهام‌زدای متداول
  - شرکت‌پذیری (associativity) عملگرها
    - شرکت‌پذیری از چپ
    - شرکت‌پذیری از راست

9-5-2 ➡ (9-5)-2

a=b=c ➡ a=(b=c)

| | | |
|---|---|---|
| *right* | → | *letter* = *right* \| *letter* |
| *letter* | → | a \| b \| ··· \| z |

# ابهام در گرامرهای مستقل از متن

- قوانین ابهام‌زدای متداول (ادامه)
  - اولویت (precedence) عملگرها

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & ( E ) \mid \mathbf{id}
\end{array}
$$

| | | |
|---|---|---|
| 1 | Parenthesis | () [] |
| 1 | Structure Access | . -> |
| 2 | Unary | ! ~ ++ -- - * & |
| 3 | Mutiply, Divide, Modulus | * / % |
| 4 | Add, Subtract | + - |
| 5 | Shift Right, Left | >> << |
| 6 | Greater, Less Than, etc | > < = |
| 7 | Equal, Not Equal | == != |
| 8 | Bitwise AND | & |
| 9 | Bitwise Exclusive OR | ^ |
| 10 | Bitwise OR | \| |
| 11 | Logical AND | && |
| 12 | Logical OR | \|\| |
| 13 | Conditional Expression | ?: |
| 14 | Assignment | = += -= etc |
| 15 | Comma | , |

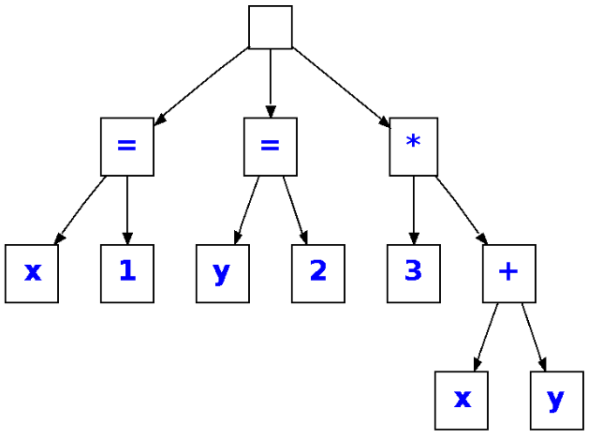
# درخت دستور انتزاعی

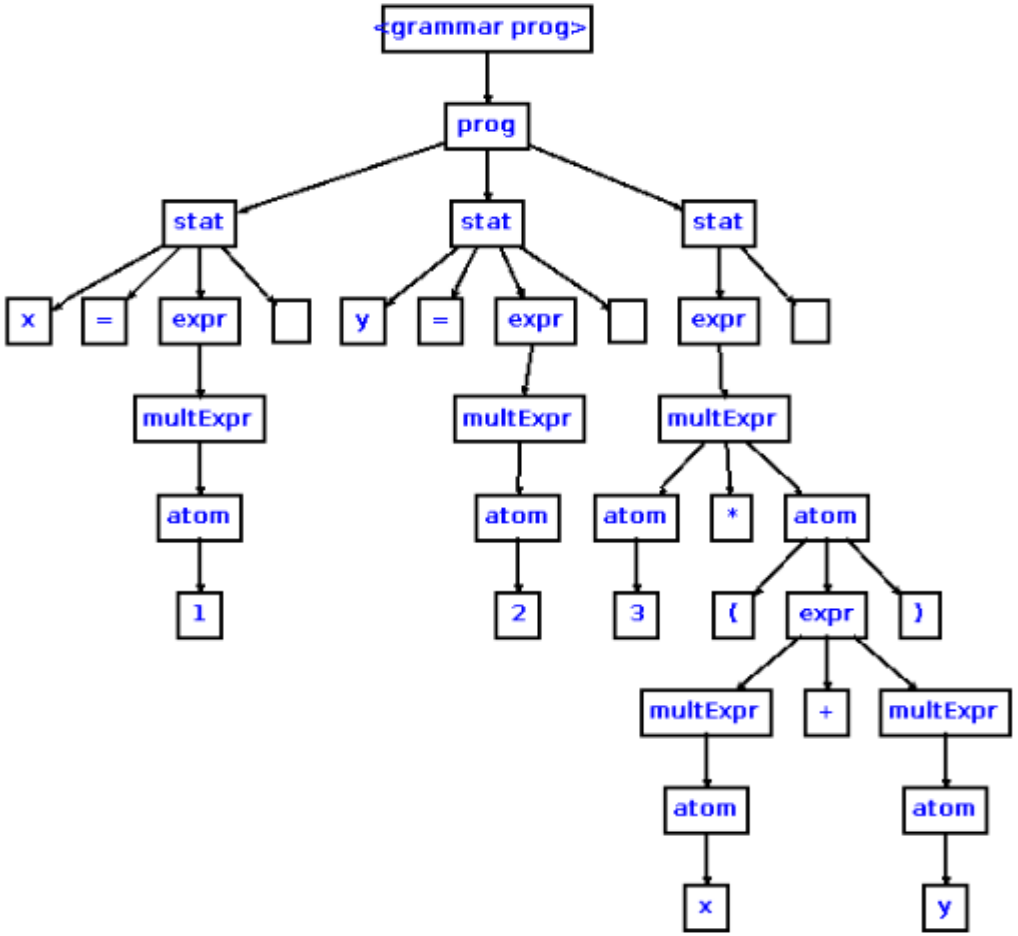
- تجزیه‌گر نحوه اشتقاق دنباله نمادها را بررسی می‌کند
  - تعیین دنباله‌های معتبر
- نیاز به یک نمایش ساختاری از برنامه در مراحل بعدی کامپایلر
  - در قالب یک ساختمان داده حاوی اطلاعات ساختاری

- درخت دستور انتزاعی (AST – abstract syntax tree)
  - مانند درخت تجزیه ولی فاقد برخی از جزئیات
    - در سطح بالاتری از انتزاع قرار دارد
    - به درخت تجزیه، درخت دستور عینی (concrete syntax tree) هم گفته می‌شود

# درخت دستور انتزاعی

- مثال: مقایسه درخت تجزیه و AST



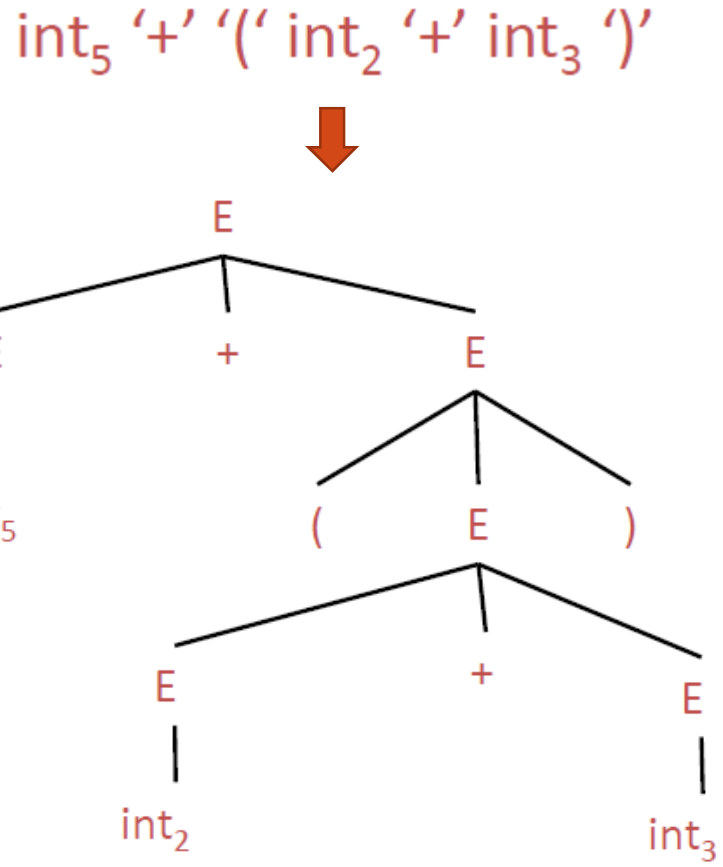



# درخت دستور انتزاعی

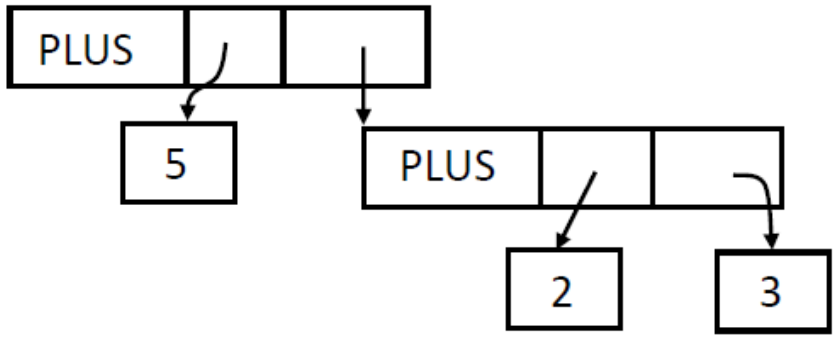
- تفاوت با درخت تجزیه
  - درخت تجزیه نشان دهنده نحوه اشتقاق یک برنامه از نشانه شروع گرامر
  - AST نشان دهنده نحوه ترکیب اجزاء برنامه با هم از لحاظ دستوری
- در نظر گرفتن درخت تجزیه یا AST؟
  - پیچیدگی زیاد درخت‌های تجزیه
    - نحوه بکارگیری قواعد تولید
    - قواعد تولید تکی برای تولید نشانه‌های پایانی
    - نحوه پرانتزبندی برنامه
  - AST توان نشان دادن ساختار تو در تو (سلسه‌مراتبی) برنامه را دارد
    - بسیار موجزتر از درخت تجزیه و کار با آن ساده‌تر است

# درخت دستور انتزاعی

- مثال: درخت تجزیه و AST برای عبارت $5 + (2 + 3)$




$E \rightarrow int \mid ( E ) \mid E + E$



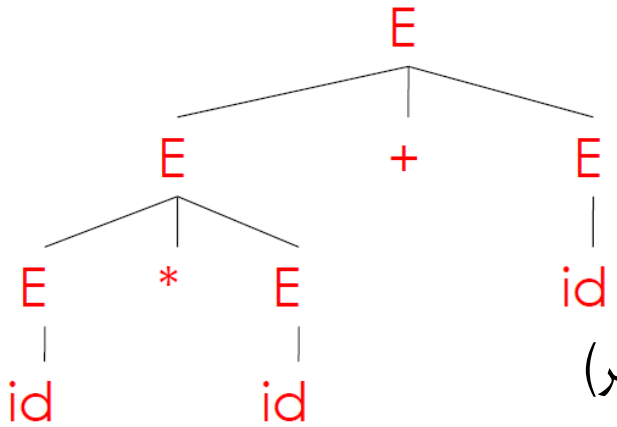
# تحلیل دستوری

- گرامرهای مستقل از متن
- روش‌های تجزیه
- تجزیه نزول بازگشتی
- تجزیه پیشگویانه
- تجزیه انتقال کاهش
- تجزیه LR
  - SLR، CLR، LALR

# تجزیه

# تجزیه

- فرآیند بررسی نحوه تولید یک رشته با استفاده از یک گرامر
  - ساختن درخت تجزیه (بدست آوردن اشتقاق)
- دو رویکرد اصلی در تجزیه
  - بالا به پایین (top-down)
    - شروع ساخت درخت تجزیه از ریشه (نشانه شروع گرامر)
    - شباهت به نحوه تجزیه دستی توسط انسان
  - پایین به بالا (bottom-up)
    - شروع ساخت درخت تجزیه از برگ‌ها (نمادها)
    - مورد استفاده در بسیاری از ابزارهای خودکار تولید تجزیه‌گر
    - پشتیبانی از دسته بزرگتری از گرامرها



# تجزیه

- الگوریتم‌های تجزیه
  - وجود الگوریتم‌های کلی تجزیه رشته‌ها برای همه گرامرهای مستقل از متن
    - چندان کارامد نیستند
  - الگوریتم‌های کاراتر برای دسته‌های خاصی از گرامرها
    - گرامرهای LL و LR
      - امکان توصیف اکثر ساختارهای متداول برنامه‌نویسی را دارند
    - افزایش کارایی تجزیه برای این دسته از گرامرها ($O(n)$)
  - بررسی دنباله (نمادهای) ورودی همیشه از چپ به راست (left-to-right)

# تجزیه بالا به پایین

- روند تجزیه بالا به پایین
  - در ابتدا تنها نشانه شروع گرامر در درخت قرار دارد
  - تکرار مراحل زیر تا وقتی که تمام برگ‌ها حاوی نشانه‌های پایانی باشد
    - انتخاب برگی حاوی یک نشانه غیرپایانی (A)
    - انتخاب یکی از قواعد تولید این نشانه غیرپایانی ($A \rightarrow X_1 X_2 \ldots X_k$)
    - افزودن نشانه‌های موجود در بدنه این قاعده تولید به عنوان فرزندان برگ انتخاب شده

- تجزیه نزول بازگشتی (recursive descent)
  - یک روش کلی برای تجزیه بالا به پایین
  - اعمال قواعد تولید گرامر به صورت بازگشتی تا رسیدن (نزول) به رشته ورودی

# تجزیه نزول بازگشتی

- تجزیه‌گر به صورت مجموعه‌ای از رویه‌ها پیاده‌سازی می‌شود
  - برای هر نشانه‌های غیرپایانی گرامر یک رویه وجود دارد
  - تجزیه با فراخوانی رویه مربوط به نشانه شروع گرامر آغاز می‌شود
  - در هر رویه
    - بررسی امکان ساختن رشته ورودی با یکی از قواعد تولید نشانه غیرپایانی مربوطه
      - رویه یک نشانه غیرپایانی می‌تواند رویه نشانه غیرپایانی دیگر را فرخوانی کند
      - فرخوانی‌های بازگشتی

- برای نشانه‌های پایانی گرامر یک رویه تطبیق با ورودی‌ها پیاده‌سازی می‌شود
  - اطمینان از درستی نسبی مسیر طی شده پس از تطبیق پایانی‌ها با ورودی

# تجزیه نزول بازگشتی

- نمونه‌ای از یک رویه برای یک نشانه غیرپایانی

```
void A() {
    Choose an A-production, A → X1 X2 ··· 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 */;
    }
}
```

- انتخاب یک قاعده تولید دیگر
- بازگرداندن ورودی‌های رد شده
- اعلام عدم توفیق در این رویه

- اعلام موفقیت در این رویه
- انجام عملیات تکمیلی متناسب با پذیرفتن یک رشته

# تجزیه نزول بازگشتی

- نیاز به عقب‌گرد (backtrack)
  - اگر قاعده تولید در حال بررسی قابل تطبیق با ورودی نباشد (انتخاب اشتباه)
  - عقب‌گرد به قبل از آخرین تصمیم گرفته شده و بررسی یک تصمیم دیگر
    - بررسی یک قاعده تولید دیگر بجای قاعده تولید قبلی
    - اگر در سطح آخرین تصمیم، امکان تصمیم‌گیری دیگری نباشد رفتن به سطح بالاتر
      - در سطح ریشه اعلام عدم پذیرش رشته ورودی
        - عدم وجود گزینه دیگر برای تصمیم‌گیری

- احتمال نیاز به پویش مجدد ورودی
  - برگرداندن ورودی‌های رد شده توسط قاعده تولید فعلی

# تجزیه نزول بازگشتی

- مثال

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$

E

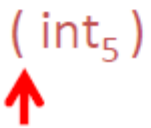
$( int_5 )$

# تجزیه نزول بازگشتی

- مثال

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$

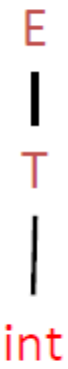



# تجزیه نزول بازگشتی

- مثال
  - عدم تطبیق با ورودی ’)‘ !
  - عقب‌گرد

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$




$( int_5 )$

# تجزیه نزول بازگشتی

- مثال

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$

E

|

T

$( int_5 )$

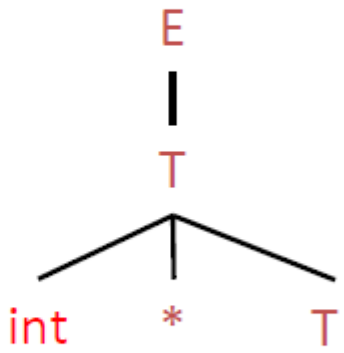
↑

# تجزیه نزول بازگشتی

- مثال
  - عدم تطبیق با ورودی ’)‘ !
  - عقب‌گرد

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$




$( int_5 )$

# تجزیه نزول بازگشتی

- مثال

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$

E

|

T

( $int_5$ )

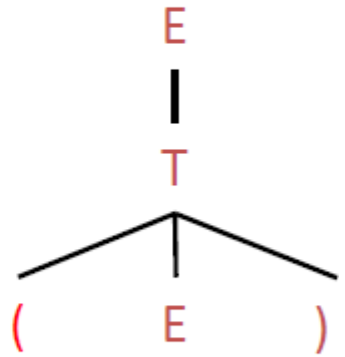
↑

# تجزیه نزول بازگشتی

- مثال
  - تطبیق با ورودی ’)‘
  - جلو بردن ورودی

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$



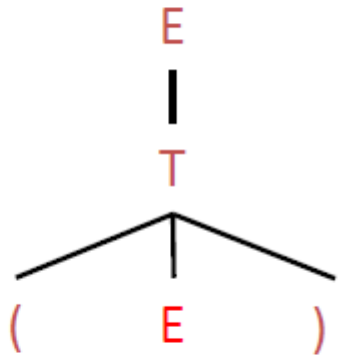

$( int_5 )$

# تجزیه نزول بازگشتی

- مثال

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$



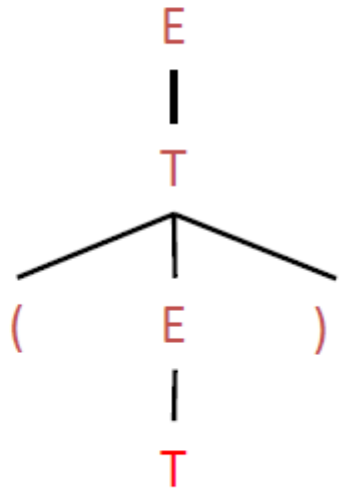

$( int_5 )$

# تجزیه نزول بازگشتی

- مثال

$E \rightarrow T \mid T + E$

$T \rightarrow \text{int} \mid \text{int} * T \mid ( E )$




$( \text{int}_5 )$

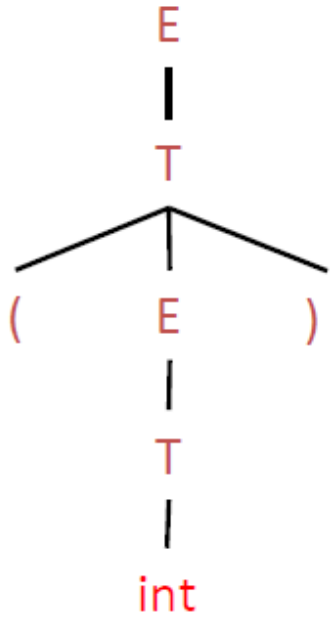
# تجزیه نزول بازگشتی

- مثال
  - تطبیق با ورودی int
  - جلو بردن ورودی

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$

$( int_5 )$



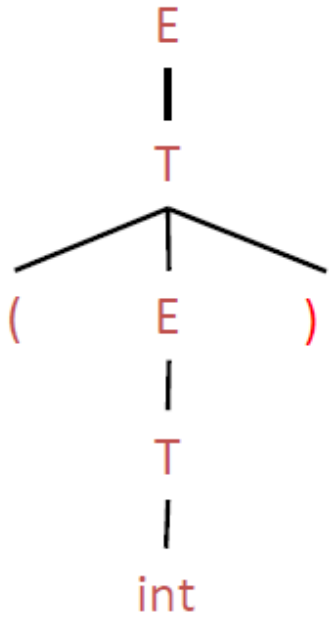
# تجزیه نزول بازگشتی

- مثال
  - تطبیق با ورودی ’(‘
  - جلو بردن ورودی

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$

$( int_5 )$

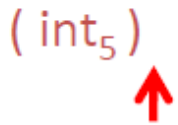


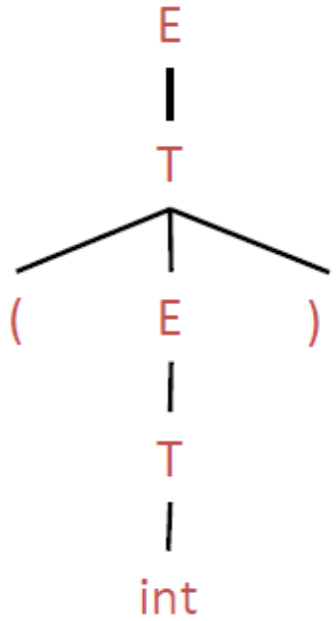
# تجزیه نزول بازگشتی

- مثال
  - پایان ورودی
  - پذیرفتن رشته

$E \rightarrow T \mid T + E$

$T \rightarrow int \mid int * T \mid ( E )$



# تجزیه نزول بازگشتی

- سوال: با گرامر داده شده، کدامیک از اشتقاق‌های زیر یک تجزیه نزول بازگشتی معتبر برای رشته id + id را نشان می‌دهد؟ جایگزینی‌هایی که بعد از آنها عقب‌گرد انجام شده به رنگ قرمز نشان داده شده‌اند.

$E \rightarrow E' \mid E' + E$

$E' \rightarrow -E' \mid id \mid (E)$

○
E
E'
E' + E
id + E
id + E'
id + id

○
E
E' + E
id + E
id + E'
id + id

○
E
E'
-E'
id
(E)
E' + E
-E' + E
id + E
id + E'
id + -E'
id + id

○
E
E'
id
E' + E
id + E
id + E'
id + id

# تجزیه نزول بازگشتی

- تمرین: تجزیه‌گر نزول بازگشتی زیر برای گرامر مورد نظر پیاده‌سازی شده است:

$E \rightarrow T \mid T + E$
$T \rightarrow int \mid int * T \mid ( E )$

```
bool term(TOKEN tok) { return *next++ == tok; }

bool E1() { return T(); }
bool E2() { return T() && term(PLUS) && E(); }

bool E()  {TOKEN *save = next; return    (next = save, E1())
                                      || (next = save,  E2());   }
bool T1() { return term(INT); }
bool T2() { return term(INT) && term(TIMES) && T(); }
bool T3() { return term(OPEN) && E() && term(CLOSE); }

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

- توالی فراخوانی رویه‌ها برای بررسی رشته‌های "(int)"، "int*int" و "int+int" را بنویسید.
- این پیاده‌سازی چه ایرادهایی دارد؟ راهکار چیست؟

# تجزیه نزول بازگشتی

- فاکتوربندی چپ (left factoring)
  - یک تبدیل روی گرامر برای تسهیل تجزیه بالا به پایین
  - استخراج بزرگترین پیشوند مشترک در بدنه قواعد تولید یک نشانه غیرپایانی

$$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \cdots \mid \alpha\beta_n \mid \gamma \quad \Rightarrow \quad \begin{array}{l} A \to \alpha A' \mid \gamma \\ A' \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n \end{array}$$

  - تعویق انتخاب بین قواعد تولید تا مشاهده ورودی‌های بیشتر
  - مثال

$$\begin{array}{l} E \to T + E \mid T \\ T \to \text{int} \mid \text{int} * T \mid ( E ) \end{array} \quad \Rightarrow \quad \begin{array}{l} E \to T\ X \\ T \to ( E ) \mid \text{int}\ Y \\ X \to + E \mid \varepsilon \\ Y \to * T \mid \varepsilon \end{array}$$

# تجزیه نزول بازگشتی

- سوال: کدامیک از گزینه‌های زیر فاکتوربندی چپ صحیح برای گرامر داده شده است؟

```
EXPR → if BOOL then { EXPR }
     | if BOOL then { EXPR } else { EXPR }
     | ...
BOOL → true | false
```

○
```
EXPR → if true then { EXPR }
     | if false then { EXPR }
     | if true then { EXPR } else { EXPR }
     | if false then { EXPR } else { EXPR }
     | ...
```

○
```
EXPR → EXPR' | EXPR' else { EXPR }
EXPR' → if BOOL then { EXPR }
      | ...
BOOL → true | false
```

○
```
EXPR → if BOOL EXPR'
     | ...
EXPR' → then { EXPR }
      | then { EXPR } else { EXPR }
BOOL → true | false
```

○
```
EXPR → if BOOL then { EXPR } EXPR'
     | ...
EXPR' → else { EXPR } | ε
BOOL → true | false
```

# تجزیه نزول بازگشتی

- مشکل بازگشت چپ (left recursion)
  - یک گرامر بازگشتی چپ است اگر برای نشانه غیرپایانی $A$ داشته باشیم

$$A \overset{+}{\Rightarrow} A\alpha$$

  - تجزیه نزول بازگشتی برای گرامرهای بازگشتی چپ قابل اعمال نیست
    - الگوریتم در حلقه تکراری گیر خواهد کرد
    - باید اول سمت چپ رشته را تطبیق دهد (بررسی ورودی از چپ به راست)
    - مثال: رویه مربوط به نشانه غیرپایانی $A$ در گرامر روبرو

$$A \rightarrow A\alpha \mid \beta$$

بصورت بازگشتی فراخوانی خواهد شد

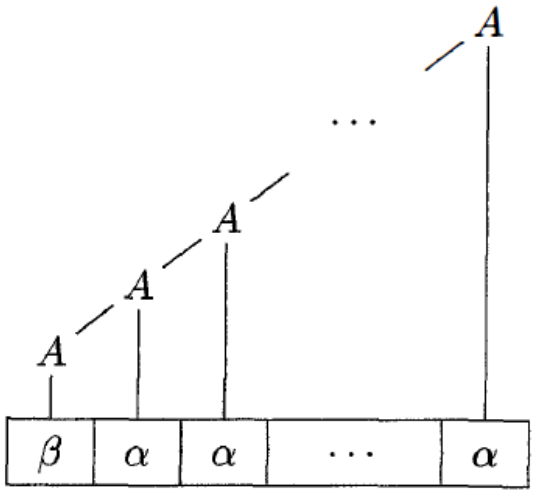
    - در هر بار فراخوانی هیچ ورودی مصرف نخواهد شد
    - شرایط تصمیم‌گیری در فراخوانی‌های متفاوت فرقی نخواهد کرد

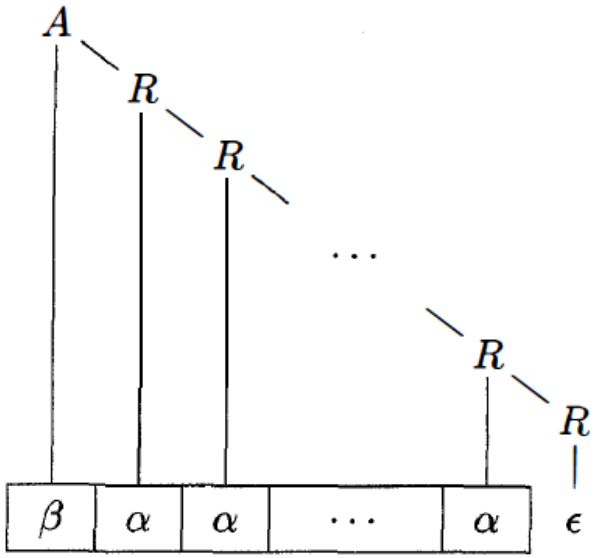
# تجزیه نزول بازگشتی

- رفع بازگشت چپ
- بازنویسی گرامر بازگشتی چپ به صورت بازگشتی راست
  - مثال: صفر یا بیشتر زیررشته α بعد از زیررشته β

$$A \rightarrow A\alpha \mid \beta$$

$$\begin{aligned} A &\rightarrow \beta R \\ R &\rightarrow \alpha R \mid \epsilon \end{aligned}$$



# تجزیه نزول بازگشتی

- رفع بازگشت چپ بی‌واسطه (immediate) در حالت کلی
  - تفکیک نقش قواعد تولید نشانه غیرپایانی بازگشتی

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

$$\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{aligned}$$

- مثال: گرامر عبارت‌های ریاضی

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (\ E\ ) \mid \mathbf{id} \end{aligned}$$

$$\begin{aligned} E &\to T\ E' \\ E' &\to +\ T\ E' \mid \epsilon \\ T &\to F\ T' \\ T' &\to *\ F\ T' \mid \epsilon \\ F &\to (\ E\ ) \mid \mathbf{id} \end{aligned}$$

# تجزیه نزول بازگشتی

- امکان وجود بازگشت چپ باواسطه
  - مثال: گرامر روبرو بازگشتی چپ است

$S \to A\,\alpha \mid \delta$

$A \to S\,\beta$

$S \overset{2+}{\Rightarrow} S\beta\alpha$

- امکان حذف بازگشت چپ باواسطه وجود دارد (الگوریتم در کتاب مرجع درس)
  - در نظر گرفتن یک ترتیب بین نشانه‌های غیرپایانی
  - جایگزینی هر غیرپایانی زودتر با بدنه قواعدش در بدنه قواعد تولید غیرپایانی‌های دیرتر
    - تبدیل بازگشت چپ باواسطه به بی‌واسطه
  - حذف بازگشت چپ بی‌واسطه در بدنه قواعد تولید هر نشانه غیرپایانی
- امکان حذف خودکار بازگشت چپ در حالت کلی
  - ولی معمولاً جهت مشخص بودن گرامر مورد استفاده، بصورت دستی انجام می‌شود

# تجزیه نزول بازگشتی

- سوال: حذف بازگشت چپ برای گرامر داده شده در کدامیک از گرامرهای زیر به درستی انجام شده است؟

$E \rightarrow E + T \mid T$

$T \rightarrow id \mid (E)$

- $E \rightarrow E + id \mid E + (E) \mid id \mid (E)$
- $E \rightarrow TE'$ <br> $E' \rightarrow + TE' \mid \varepsilon$ <br> $T \rightarrow id \mid (E)$
- $E \rightarrow E' + T \mid T$ <br> $E' \rightarrow id \mid (E)$ <br> $T \rightarrow id \mid (E)$
- $E \rightarrow id + E \mid E + T \mid T$ <br> $T \rightarrow id \mid (E)$

# تحلیل دستوری

- گرامرهای مستقل از متن
- روش‌های تجزیه
- تجزیه نزول بازگشتی
- تجزیه پیشگویانه
- تجزیه انتقال کاهش
- تجزیه LR
  - SLR، CLR، LALR

# تجزیه پیشگویانه (predictive)

# تجزیه پیشگویانه

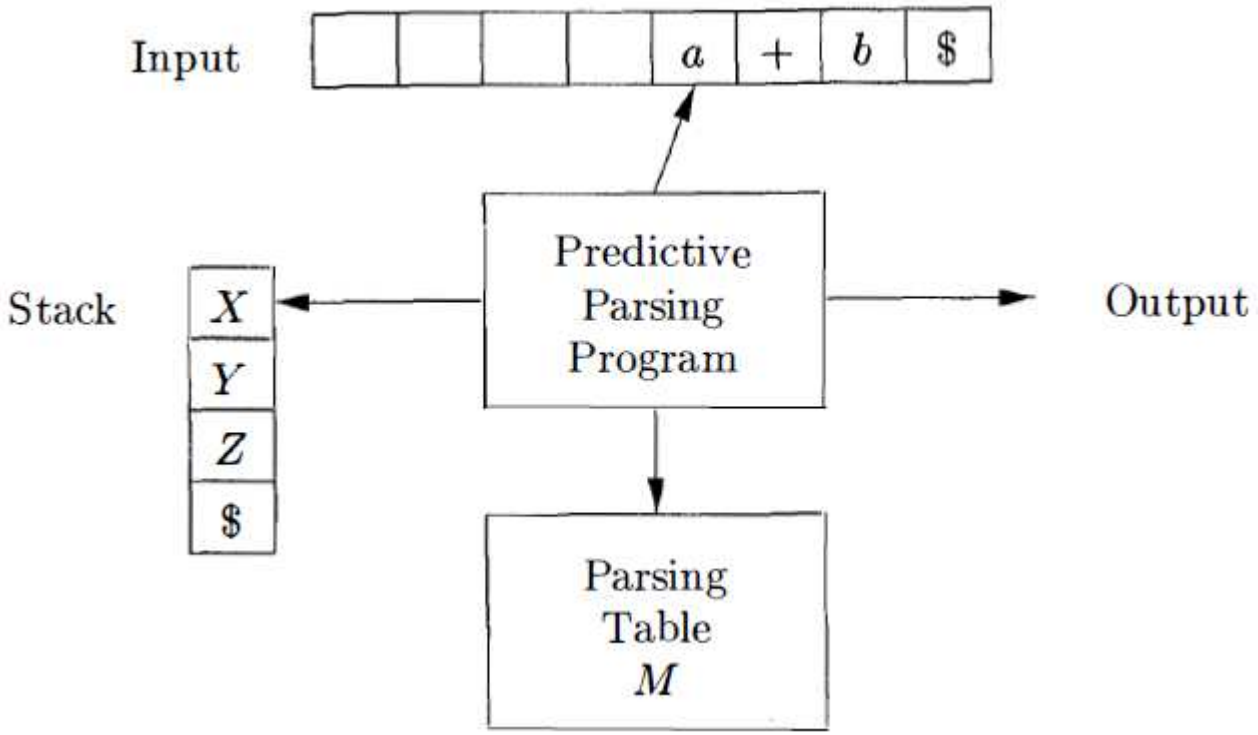
- گونه‌ای از تجزیه بالا به پایین
  - نیازی به عقب‌گرد ندارد
    - تصمیم‌گیری یکتا در مورد قاعده تولید بعدی که باید استفاده شود (پیشگویی)
  - در نظر گرفتن تعدادی از ورودی‌های بعدی (lookahead) برای تصمیم‌گیری
  - محدودیت در گرامرهای قابل استفاده در تجزیه
    - دسته گرامرهای LL(k)
      - Left to right scan
      - Leftmost derivation
      - **k** lookahead input symbols
    - در عمل معمولاً $k = 1$ در نظر گرفته می‌شود
      - پیاده‌سازی گرامرهای LL(1)

# تجزیه پیشگویانه

- عملکرد تجزیه‌گر پیشگویانه کاملاً معین (deterministic) است
  - همیشه سمت‌چپ‌ترین نشانه غیرپایانی جایگزین می‌شود
  - در هر گام تجزیه حداکثر یک قاعده تولید از نشانه غیرپایانی قابل استفاده است
    - استفاده از هر قاعده تولید دیگر قطعاً اشتباه است
    - ممکن است هیچ قاعده تولیدی قابل استفاده نباشد
      - خطا در تجزیه
- تجزیه‌گر پیشگویانه برای گرامرهای LL(1)
  - نیاز به شناسایی قاعده تولید مناسب با در نظر گرفتن ورودی فعلی
    - هر نشانه پایانی باید توسط یک قاعده تولید بدست آید
  - گرامر باید فاکتوربندی چپ شده باشد

# تجزیه پیشگویانه

- امکان پیاده‌سازی تجزیه‌گر پیشگویانه بصورت مجموعه‌ای از رویه‌ها
  - نوعی از تجزیه نزول بازگشتی که کاملاً معین است
- پیاده‌سازی تجزیه‌گر پیشگویانه بصورت غیربازگشتی با استفاده از پشته



# تجزیه پیشگویانه

- تجزیه‌گر پیشگویانه غیربازگشتی
  - استفاده از یک جدول تجزیه بصورت M[A, a] برای انتخاب قواعد تولید
    - به ازای هر نشانه غیرپایانی (A) یک سطر در جدول وجود دارد
    - به ازای هر نشانه پایانی (a) یک ستون در جدول وجود دارد
    - هر خانه جدول، قاعده تولید مورد استفاده برای نشانه غیرپایانی A با دیدن نماد a در ورودی را نشان می‌دهد
    - خانه‌های خالی نشان دهنده خطا در تجزیه هستند
  - یک نشانه ویژه ($) برای نشان دادن پایان ورودی در نظر گرفته می‌شود
    - نباید جزء نشانه‌های پایانی گرامر باشد
    - برای این نشانه هم یک ستون در جدول وجود دارد

# تجزیه پیشگویانه

- تجزیه‌گر پیشگویانه غیربازگشتی (ادامه)
  - مثالی از یک جدول تجزیه پیشگویانه

$E \rightarrow T\ X$
$T \rightarrow (\ E\ )\ |\ int\ Y$
$X \rightarrow +\ E\ |\ \varepsilon$
$Y \rightarrow *\ T\ |\ \varepsilon$

| | int | * | + | ( | ) | $ |
|---|---|---|---|---|---|---|
| E | T X | | | T X | | |
| X | | | + E | | ε | ε |
| T | int Y | | | ( E ) | | |
| Y | | * T | ε | | ε | ε |

- پشته دربردارنده نشانه‌های (پایانی و غیرپایانی) بررسی نشده است
  - نشان دهنده نشانه‌های در انتظار بررسی در درخت تجزیه
  - در بالای پشته همیشه سمت چپ‌ترین نشانه قرار دارد

# تجزیه پیشگویانه

- الگوریتم غیربازگشتی تجزیه‌گر پیشگویانه (ماشین پذیرنده پشته‌ای)
  - در شروع بکار تجزیه‌گر، جمله w$ در ورودی و S$ در پشته قرار دارد

```
set ip to point to the first symbol of w;                    ← نماد a اولیه
set X to the top stack symbol;
while ( X ≠ $ ) { /* stack is not empty */
        if ( X is a ) pop the stack and advance ip;          ← تطبیق ورودی
        else if ( X is a terminal ) error();                 ← عدم تطبیق ورودی
        else if ( M[X, a] is an error entry ) error();
        else if ( M[X, a] = X → Y1 Y2 ··· Yk ) {
                output the production X → Y1 Y2 ··· Yk;      ← عملیات تکمیلی تجزیه
                pop the stack;
                push Yk, Yk−1, ..., Y1 onto the stack, with Y1 on top;
        }
        set X to the top stack symbol;
}
```

- اعلام پذیرش اگر با خالی شدن پشته به پایان ورودی برسیم
  - نشانه $ در بالای پشته با نشانه $ در پایان ورودی تطبیق یابد

# تجزیه پیشگویانه

- نحوه کار تجزیه‌گر پیشگویانه غیربازگشتی

| | int | * | + | ( | ) | $ |
|---|---|---|---|---|---|---|
| E | T X | | | T X | | |
| X | | | + E | | ε | ε |
| T | int Y | | | ( E ) | | |
| Y | | * T | ε | | ε | ε |

| Stack | Input | Action |
|---|---|---|
| E $ | int * int $ | T X |
| T X $ | int * int $ | int Y |
| int Y X $ | int * int $ | terminal |
| Y X $ | * int $ | * T |
| * T X $ | * int $ | terminal |
| T X $ | int $ | int Y |
| int Y X $ | int $ | terminal |
| Y X $ | $ | ε |
| X $ | $ | ε |
| $ | $ | ACCEPT |

$E \rightarrow T\ X$
$T \rightarrow (\ E\ ) \mid int\ Y$
$X \rightarrow +\ E \mid \varepsilon$
$Y \rightarrow *\ T \mid \varepsilon$

# تجزیه پیشگویانه

- سوال: با در نظر گرفتن جدول تجزیه، گرامر و وضعیت فعلی تجزیه‌گر، وضعیت بعدی چیست؟ (رشته ورودی: if true then { true } else { if false then { false } } $)

| | if | then | else | { | } | true | false | $ |
|---|---|---|---|---|---|---|---|---|
| E | if B then { E } E' | | | | ε | B | B | ε |
| E' | | | else { E } | | ε | | | ε |
| B | | | | | | true | false | |

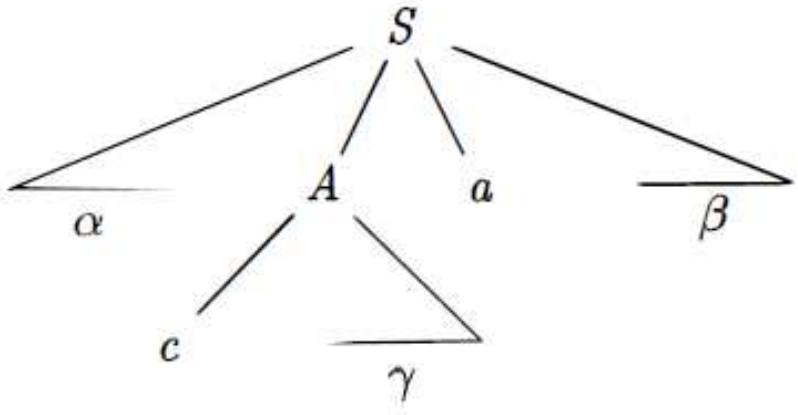
| | Stack | Input |
|---|---|---|
| Current | E' $ | else { if false then { false } } $ |
| ○ | $ | $ |
| ○ | else { E } $ | else { if false then { false } } $ |
| ○ | E } $ | if false then { false } } $ |
| ○ | else {if B then { E } E' } $ | else { if false then { false } } $ |

E → if B then { E } E'
| B | ε
E' → else { E } | ε
B → true | false

# تجزیه پیشگویانه

- انتخاب قاعده تولید برای یک نشانه غیرپایانی با دیدن ورودی فعلی
  - اگر $A \longrightarrow \alpha$ یک قاعده تولید باشد، هدف بررسی $M[A, a] \stackrel{?}{=} \alpha$
    - آیا اشتقاق $\alpha \stackrel{*}{\Rightarrow} a\beta$ ممکن است؟
    - آیا هر دو اشتقاق $\alpha \stackrel{*}{\Rightarrow} \epsilon$ و $S \stackrel{*}{\Rightarrow} \gamma Aa\beta$ ممکن است؟
  - استفاده از مفهوم مجموعه‌های اولین (First) و پسین (Follow)
    - مجموعه اولین برای رشته $\alpha$ از نشانه‌های گرامر ($First(\alpha)$)
      - نشانه‌های پایانی موجود در ابتدای هر رشته اشتقاق‌پذیر از $\alpha$ (یا $\epsilon$)
    - مجموعه پسین برای نشانه $X$ از گرامر ($Follow(X)$)
      - نشانه‌های پایانی موجود بلافاصله پس از $X$ در هر رشته اشتقاق‌پذیر از $S$
        - فرض: نشانه ویژه $ در پایان تمام رشته‌ها وجود دارد

# تجزیه پیشگویانه




- مجموعه‌های اولین و پسین
  - مثال: طبق درخت تجزیه روبرو داریم
    - $c \in First(A)$
    - $a \in Follow(A)$
- شرط استفاده از قاعده تولید $A \longrightarrow \alpha$ در تجزیه پیشگویانه
  - اگر ورودی فعلی عضو مجموعه اولین بدنه قاعده تولید ($\alpha$) باشد
  - اگر بدنه قاعده تولید ($\alpha$) تهی‌پذیر (nullable) باشد و ورودی فعلی عضو مجموعه پسین نشانه غیرپایانی سمت چپ (سر – head) قاعده تولید (A) باشد
    - امکان حذف نشانه غیرپایانی سمت چپ قاعده تولید از رشته جمله‌ای

# تجزیه پیشگویانه

- نحوه محاسبه مجموعه اولین
  - اگر a یک نشانه پایانی گرامر باشد: $First(a) = \{a\}$
  - اگر A نشانه غیرپایانی گرامر با قواعد تولید $A \rightarrow \alpha_1 \mid \cdots \mid \alpha_n$ باشد:

$$First(A) = First(\alpha_1) \cup \cdots \cup First(\alpha_n)$$

    - اگر یکی از قواعد تولید $(\forall i)\ \alpha_i = \epsilon$ باشد: $\epsilon \in First(A)$
  - اگر $\alpha = X_1 \ldots X_k$ یک دنباله از نشانه‌های گرامر باشد:

$$First(\alpha) = \{First(X_1) - \epsilon\} \cup \{First(X_2) - \epsilon : nullable(X_1)\} \cup \cdots \cup \{First(X_k) - \epsilon : \forall i<k, nullable(X_i)\} \cup \{\epsilon : \forall i, nullable(X_i)\}$$

    - اگر $X_i$ یک نشانه غیرپایانی باشد: $\epsilon \in First(X_i) \Rightarrow nullable(X_i)$

# تجزیه پیشگویانه

- محاسبه مجموعه اولین
  - مثال: مجموعه اولین برای نشانه‌های گرامر روبرو

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

  - $First(+) = \{+\}$
  - $First(*) = \{*\}$
  - $First(() = \{(\}$
  - $First()) = \{)\}$
  - $First(\mathbf{id}) = \{\mathbf{id}\}$
  - $First(E) = First(T) = First(F) = \{(, \mathbf{id}\}$
  - $First(E') = \{+, \epsilon\}$
  - $First(T') = \{*, \epsilon\}$

# تجزیه پیشگویانه

- نحوه محاسبه مجموعه پسین برای هر نشانه X از گرامر
  - اگر X نشانه شروع گرامر باشد: $\$ \in Follow(X)$
  - بررسی تمام قواعد تولیدی که نشانه X در بدنه آنها استفاده شده است
    - باید مکان کاربرد نشانه بررسی شود
    - بررسی قواعد تولید همه نشانه‌های غیرپایانی گرامر برای نشانه X
  - اگر $A \rightarrow \alpha X \beta$ یک قاعده تولید گرامر باشد که شامل نشانه X است:

$$\{First(\beta) - \epsilon\} \cup \{Follow(A) : nullable(\beta)\} \subseteq Follow(X)$$

- مجموعه پسین به صورت افزایشی (incremental) ساخته می‌شود

# تجزیه پیشگویانه

- محاسبه مجموعه پسین
- مثال: مجموعه پسین برای نشانه‌های گرامر روبرو

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

- $\text{Follow(E)} = \text{Follow(E}') = \{\$, )\}$
- $\text{Follow(T)} = \text{Follow(T}') = \{+, \$, )\}$
- $\text{Follow(F)} = \{*, +, \$, )\}$
- $\text{Follow}(+) = \text{Follow}(*) = \text{Follow}(() = \{(, \mathbf{id}\}$
- $\text{Follow}()) = \text{Follow}(\mathbf{id}) = \{*, +, \$, )\}$

# تجزیه پیشگویانه

- ساختن جدول تجزیه پیشگویانه (M[A, a])
  - a می‌تواند هر یک از نشانه‌های پایانی گرامر یا نشانه ویژه $ باشد
  - A می‌تواند هر یک از نشانه‌های غیرپایانی گرامر باشد
  - محاسبه مجموعه اولین برای بدنه تمام قواعد تولید
  - محاسبه مجموعه پسین برای تمام نشانه‌های غیرپایانی

- برای هر قاعده تولید $A \longrightarrow \alpha$ قرار می‌دهیم $M[A, a] = \alpha$ اگر
  - $a \in First(\alpha)$
  - $a \in Follow(A)$ و $nullable(\alpha)$
- در خانه‌هایی که با هیچ یک از قواعد تولید پر نشود خطا اعلام می‌کنیم

# تجزیه پیشگویانه

- ساختن جدول تجزیه پیشگویانه
- مثال: جدول تجزیه پیشگویانه برای گرامر روبرو

$$
\begin{aligned}
E &\rightarrow T\ E' \\
E' &\rightarrow +\ T\ E' \mid \epsilon \\
T &\rightarrow F\ T' \\
T' &\rightarrow *\ F\ T' \mid \epsilon \\
F &\rightarrow (\ E\ ) \mid \mathbf{id}
\end{aligned}
$$

| NON-TERMINAL | INPUT SYMBOL | | | | | |
|---|---|---|---|---|---|---|
| | **id** | + | * | ( | ) | $ |
| $E$ | $E \rightarrow TE'$ | | | $E \rightarrow TE'$ | | |
| $E'$ | | $E' \rightarrow +TE'$ | | | $E' \rightarrow \epsilon$ | $E' \rightarrow \epsilon$ |
| $T$ | $T \rightarrow FT'$ | | | $T \rightarrow FT'$ | | |
| $T'$ | | $T' \rightarrow \epsilon$ | $T' \rightarrow *FT'$ | | $T' \rightarrow \epsilon$ | $T' \rightarrow \epsilon$ |
| $F$ | $F \rightarrow \mathbf{id}$ | | | $F \rightarrow (E)$ | | |

# تجزیه پیشگویانه

- گرامر LL(1)
  - برای هر دو قاعده تولید $A \longrightarrow \alpha \mid \beta$ هر سه شرط زیر برقرار باشد:
    - $\{First(\alpha) - \epsilon\} \cap \{First(\beta) - \epsilon\} = \emptyset$
    - اگر $nullable(\beta)$ آنگاه $First(\alpha) \cap Follow(A) = \emptyset$
    - اگر $nullable(\alpha)$ آنگاه $First(\beta) \cap Follow(A) = \emptyset$
  - وجود حداکثر یک قاعده تولید در هر خانه از جدول تجزیه پیشگویانه
    - در صورت وجود خانه‌هایی به صورت چندگانه (multiply defined entries)، تجزیه‌گر پیشگویانه مبتنی بر جدول، عملکرد معین (انتخاب فقط یک قاعده تولید) نخواهد داشت
- خیلی از گرامرها LL(1) نیستند
  - مثلاً گرامرهای مبهم، بازگشتی چپ و فاکتوربندی چپ نشده

# مدیریت خطا
# (Error Handling)

# مدیریت خطا

- وظایف اصلی کامپایلر
  - تشخیص برنامه‌های معتبر از نامعتبر
  - ترجمه برنامه‌های معتبر
- ضرورت مدیریت خطا در کامپایلر
  - گزارش دقیق و شفاف خطا
  - فرآیند سریع بازیابی از خطا (error recovery)
  - جلوگیری از کند شدن روند کامپایل برنامه‌های معتبر
- توصیف زبان‌ها معمولاً نحوه مدیریت خطا را مشخص نمی‌کند
  - طراح کامپایلر باید خود در مورد مدیریت خطاها تصمیم‌گیری کند

# مدیریت خطا

- برنامه‌های ورودی می‌تواند در سطوح مختلف حاوی خطا باشد
  - خطاهای واژه‌ای (lexical)
    - استفاده از نویسه‌های غیرمجاز در برنامه
      - مثلاً **$** در برنامه‌های جاوا
  - خطاهای دستوری (syntactic)
    - عدم رعایت دستور زبان در نوشتن برنامه
      - مثال: فقدان ’**;**‘ در انتهای یک دستور برنامه، یا عبارت ناصحیح **x * %**
  - خطاهای معنایی (semantic)
    - مثلاً عدم تطابق نوع: **int x; y = x[3];**
  - خطاهای منطقی (logical)
    - مثال: بکارگیری ’**=**‘ بجای ’**==**‘ در یک عبارت توسط برنامه‌نویس

# مدیریت خطا

- برخی روش‌های بازیابی از خطا (error recovery)
  - قواعد تولید خطا (error productions)
    - پیش‌بینی خطاهای پرتکرار
    - جلوگیری از عملکرد پیش‌بینی نشده کامپایلر
  - تصحیح خطا (error correction)
    - هدف پیدا کردن یک برنامه معتبر نزدیک به برنامه ورودی
    - تغییر برنامه ورودی برای یافتن برنامه معتبر
  - روال ترس (panic mode)
    - خروج از روال عادی کار کامپایلر
    - سعی در برطرف کردن مشکل تطبیق برنامه ورودی در روال ترس

# مدیریت خطاهای واژه‌ای

- تنوع خطاهای واژه‌ای
  - معمولاً شناسایی خطا در مرحله تحلیل واژه‌ای به تنهایی ممکن نیست
    - مثال: دستور شرطی یا فراخوانی تابع؟ `fi ( a == f(x)) ...`
  - خطا: عدم امکان تطبیق هر پیشوندی از باقیمانده ورودی با یکی از الگوها


- روال ترس
  - هنگامی که دیگر امکان تطبیق نویسه‌ها وجود نداشته باشد
  - حذف پی‌درپی تعدادی از نویسه‌ها از باقیمانده ورودی
  - ادامه اینکار تا یافتن یک تکواژه معتبر در پیشوند باقیمانده ورودی

# مدیریت خطاهای واژه‌ای

- تصحیح ورودی
  - بکارگیری مجموعه‌ای از تبدیلات روی باقیمانده ورودی
    - حذف یک نویسه
    - درج یک نویسه مفقود (missing)
    - جایگزینی یک نویسه با نویسه دیگر
    - جابجایی دو نویسه مجاور
  - سعی در یافتن تکواژه‌ای معتبر با اعمال یک تبدیل به باقیمانده ورودی
    - معمولاً اکثر خطاهای واژه‌ای فقط شامل یک نویسه هستند
- یافتن کمترین تعداد تبدیلاتی که کل برنامه ورودی را تصحیح کند؟
  - منجر به برنامه‌ای می‌شود که فقط شامل تکواژه‌های معتبر است
  - به دلیل پیچیدگی بالا در عمل امکان‌پذیر نیست

# مدیریت خطاهای دستوری

- قابلیت تحلیل دستوری در شناسایی خطاها
  - امکان شناسایی سریع خطا به علت دقیق بودن روند تحلیل دستوری
  - بیشتر خطاها از نوع دستوری هستند
    - بروز بسیاری از خطاهای واژه‌ای و معنایی به شکل خطای دستوری
- بکارگیری قواعد تولید خطا
  - اضافه کردن قواعد تولید جدید به گرامر زبان برای خطاهای متداول
    - منجر به پیچیدگی گرامر
  - مثال: قاعده تولید $\text{stmt} \rightarrow \{\text{stmts}\}$
  - مثال: قاعده تولید $E \rightarrow EE$ برای عبارت‌های ضرب (مثلاً **2a**)

# مدیریت خطاهای دستوری

- تصحیح ورودی
  - اعمال مجموعه‌ای از تغییرات روی برنامه ورودی
    - حذف، اضافه، جایگزینی یا جابجایی نمادها
  - تصحیح خطا در سطح محلی یا سراسری؟
    - تغییرات در پیشوندی از باقیمانده نمادهای ورودی یا در کل برنامه ورودی؟
- جستجو به دنبال برنامه‌ای معتبر نزدیک به برنامه ورودی
  - نحوه تعریف فاصله دو برنامه از هم
    - مثلاً تعداد تغییرات لازم برای تبدیل یک برنامه به دیگری
  - یافتن مجموعه‌ای کمینه از تغییرات که برنامه ورودی را تصحیح کند
  - بیشتر یک روش تئوری برای ارزیابی سایر روش‌های بازیابی از خطا

# مدیریت خطاهای دستوری

- مشکلات یافتن نزدیکترین برنامه معتبر
  - پیاده‌سازی آن سخت و بسیار زمانبر است (یک مسأله بهینه‌سازی ترکیباتی)
  - تجزیه برنامه‌های معتبر را کند می‌کند
  - نزدیک‌ترین برنامه صحیح ممکن است مدنظر برنامه‌نویس نباشد
- روال ترس
  - حذف تعدادی از نمادها تا رسیدن به نمادی با نقش مشخص
    - مجموعه نمادهای هماهنگ‌کننده (synchronizing)
      - دارای نقشی مشخص در زبان هستند
        - خاتمه دهنده توابع، دستورات و عبارت‌ها
      - مثلاً رسیدن به حائل‌ها (delimiters) مانند ‘ ; ’ یا ‘ } ’

# مدیریت خطاهای دستوری

- بازیابی از خطا در تجزیه پیشگویانه
  - دو شرط وقوع خطا در تجزیه پیشگویانه
    - عدم تطبیق نشانه پایانی بالای پشته با نماد ورودی فعلی
    - عدم امکان انتخاب قاعده تولید برای نشانه غیرپایانی بالای پشته و نماد ورودی فعلی
      - خانه خالی در جدول تجزیه
  - برخی راهکارها
    - برداشتن نشانه پایانی از بالای پشته در صورت عدم تطبیق با ورودی فعلی
      - مانند درج نماد مربوطه در دنباله ورودی
    - رد کردن ورودی‌ها در بررسی نشانه غیرپایانی تا رسیدن به مجموعه هماهنگ کننده
      - مانند حذف برخی نمادها در دنباله ورودی (روال ترس)

# مدیریت خطاهای دستوری

- روال ترس در تجزیه پیشگویانه
- برخی گزینه‌ها برای مجموعه هماهنگ کننده برای نشانه غیرپایانی A
  - استفاده از Follow(A)
    - سعی در تجزیه ساختار بعد از A در ورودی
    - برداشتن A از بالای پشته با رسیدن به مجموعه هماهنگ کننده
  - استفاده از First(A)
    - سعی در تجزیه مجدد A با دیدن نشانه پیشین A در ورودی
- افزودن آدرس رویه‌های تصحیح‌کننده به خانه‌های خالی در جدول
  - رد کردن ورودی
  - برداشتن از بالای جدول

# مدیریت خطاهای دستوری

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

- روال ترس در تجزیه پیشگویانه (ادامه)
  - مثال: جدول تجزیه برای بازیابی خطا

| NON-TERMINAL | INPUT SYMBOL | | | | | |
|---|---|---|---|---|---|---|
| | **id** | + | * | ( | ) | $ |
| $E$ | $E \rightarrow TE'$ | | | $E \rightarrow TE'$ | synch | synch |
| $E'$ | | $E' \rightarrow +TE'$ | | | $E' \rightarrow \epsilon$ | $E' \rightarrow \epsilon$ |
| $T$ | $T \rightarrow FT'$ | synch | | $T \rightarrow FT'$ | synch | synch |
| $T'$ | | $T' \rightarrow \epsilon$ | $T' \rightarrow *FT'$ | | $T' \rightarrow \epsilon$ | $T' \rightarrow \epsilon$ |
| $F$ | $F \rightarrow \mathbf{id}$ | synch | synch | $F \rightarrow (E)$ | synch | synch |

- خانه‌های خالی: ورود به روال ترس برای رد کردن ورودی تا رسیدن به synch
- رویه synch برای برداشتن نشانه بالای پشته
  - استفاده از Follow به عنوان مجموعه هماهنگ کننده
- برای نشانه پایانی: در صورت عدم تطبیق با ورودی از بالای پشته برداشته می‌شود

# مدیریت خطا

- تغییر نیازها در طول زمان
  - کامپایل کردن کندتر ولی شناسایی خطاهای بسیار زیاد
    - هنگامی که فرآیند برنامه‌نویسی سخت و طولانی باشد
  - کامپایل کردن سریع و بکارگیری روش‌های ساده مدیریت خطا
    - عدم نیاز به روش‌های پیچیده بازیابی خطا
  - کاربران معمولاً تمایل به اصلاح یک خطا در هر بار کامپایل کردن دارند