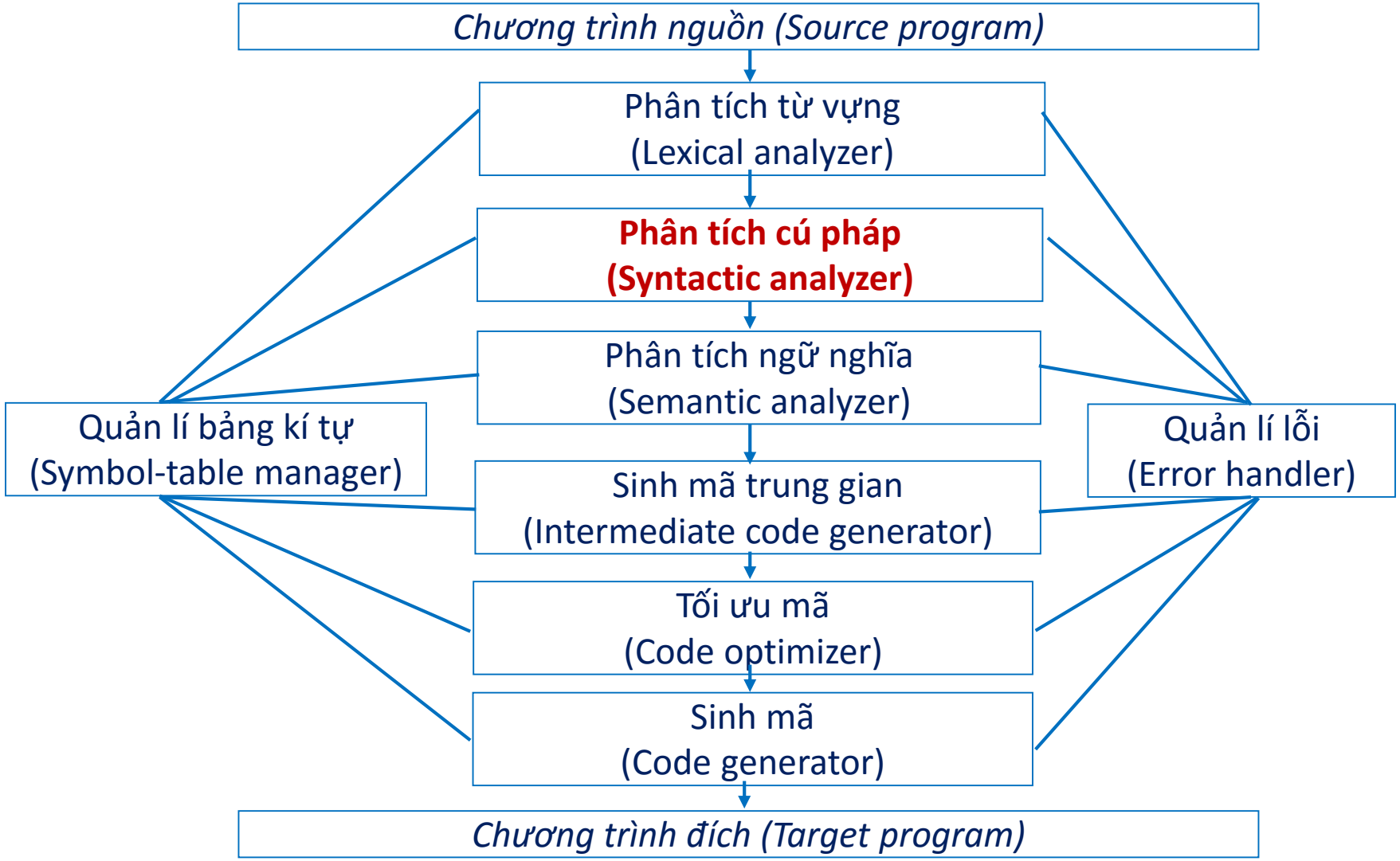
Lecture 7. Syntax Analysis (Phân tích cú pháp)

Hà Chí Trung, BM KHMT, KCNTT, HVKTQS

hct2009@yahoo.com

01685-582-102

Vị trí của phân tích cú pháp



Vị trí của phân tích cú pháp

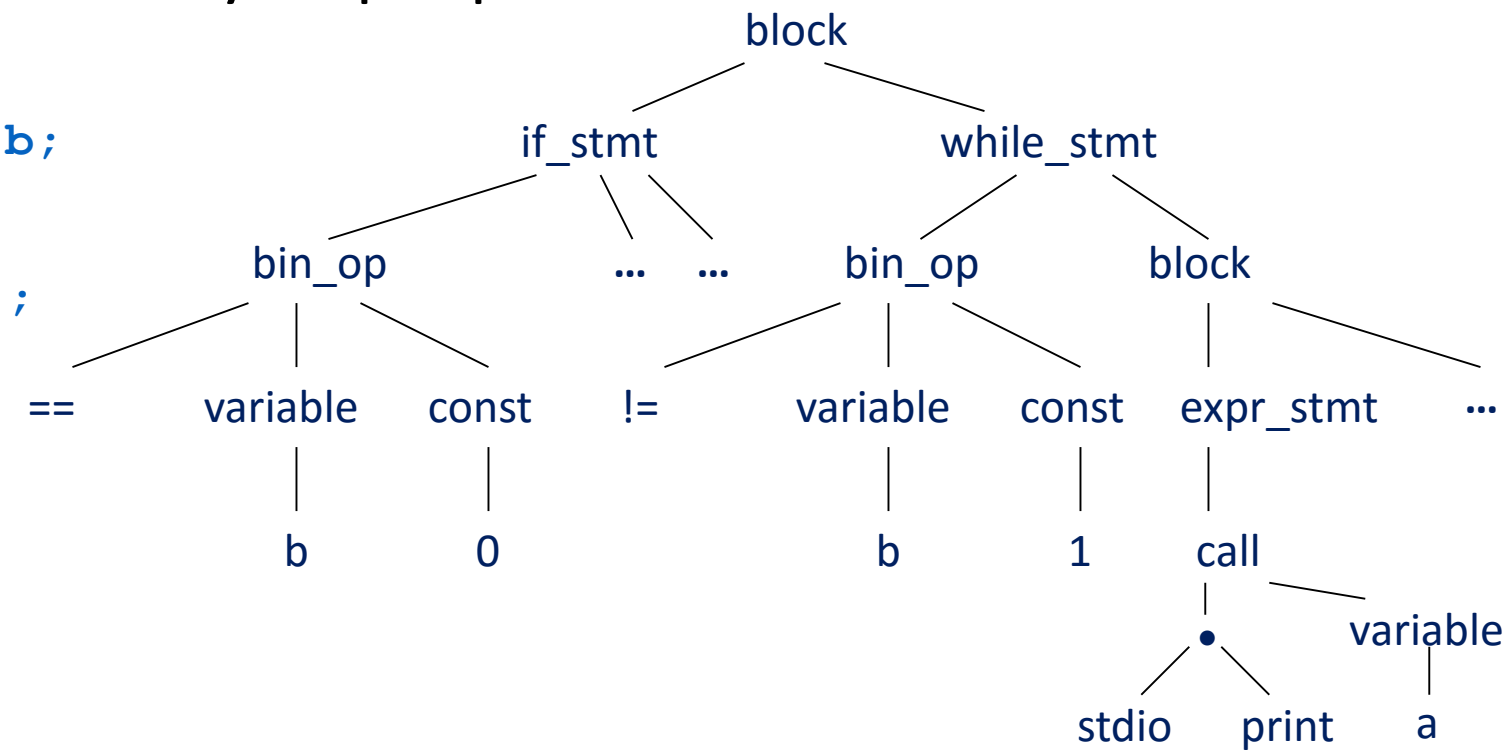
- Cú pháp của một **PL** có thể được miêu tả bởi một **CFG**. Thông thường người ta sử dụng dạng **EBNF** (**Extended Backus-Naur Form**) để diễn đạt chúng.
- Nhiệm vụ chính của **SA** (**Syntax Analyzer, parser**):
  - kiểm tra xem SP của chương trình có thỏa mãn cú pháp của ngôn ngữ hay không, bằng cách nhận chuỗi các token từ bộ phân tích từ vựng và xác định chuỗi đó có được sinh ra bởi văn phạm của SL không.
  - Thông thường, parser sẽ chỉ ra cấu trúc cú pháp của mã nguồn, cấu trúc này trong hầu hết trường hợp có thể biểu diễn như là một parse tree.
  - Ngược lại, trả về các thông báo lỗi.

Vị trí của phân tích cú pháp

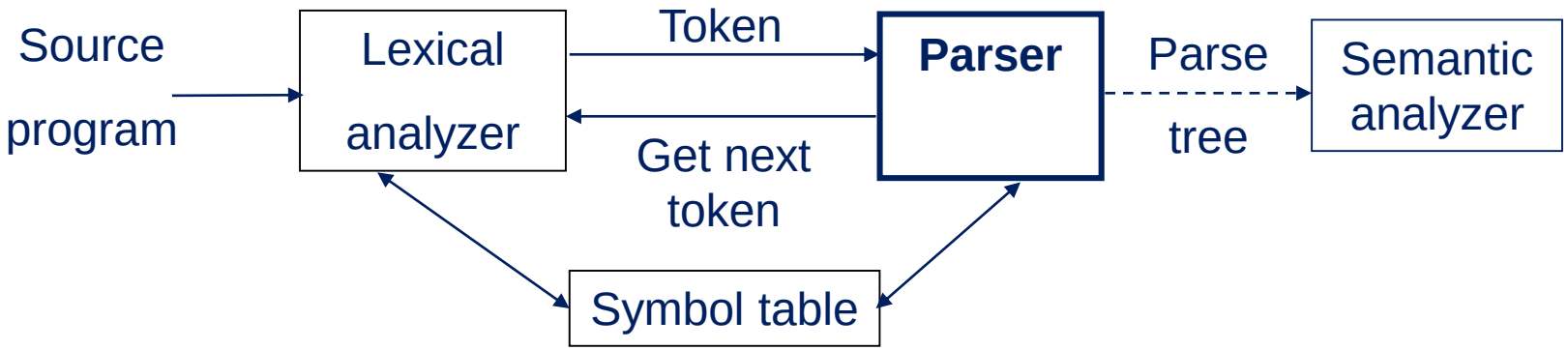
- Mã nguồn:

```
{
    if (b == (0)) a = b;
    while (a != 1) {
        stdio.print(a);
        a = a - 1;
    }
}
```

Cây cú pháp



Vị trí của phân tích cú pháp



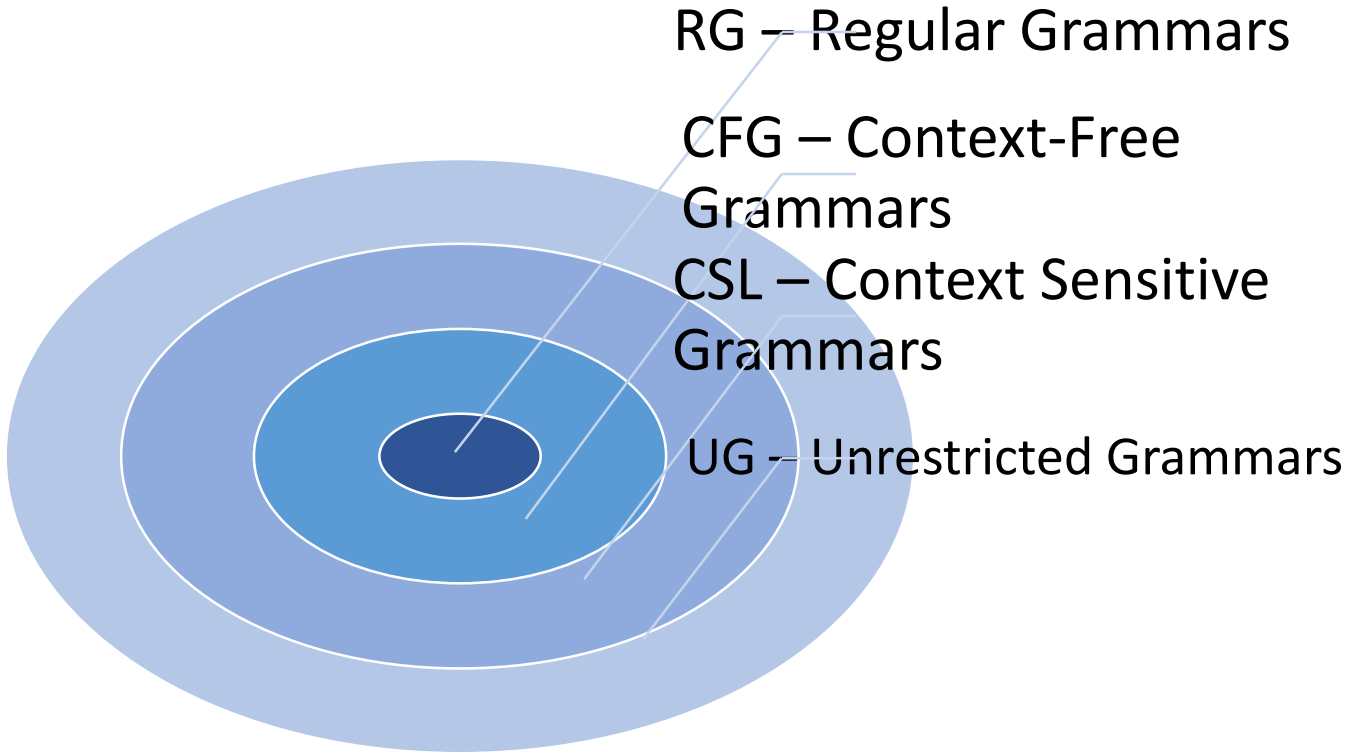
- Các chiến lược phân tích cú pháp:
  - Phân tích từ trên xuống (**top-down parsing**): Cây cú pháp được tạo từ trên xuống, bắt đầu từ gốc;
  - Phân tích từ dưới lên (**bottom-up parsing**): Cây cú pháp được tạo từ dưới lên, bắt đầu từ các lá.
- Cả hai dạng parsers đều quét luồng dữ liệu vào từ trái qua phải (tại mỗi thời điểm thường là 1 token).

Văn phạm phi ngữ cảnh

- Văn phạm phi ngữ cảnh **(CFG – Context-Free Grammar):** có luật sinh dạng $A \rightarrow \alpha$ với A là một biến đơn và α là chuỗi các ký hiệu thuộc $(\Sigma \cup \Delta)^*$;
- VD:

$G = (\{a, b\}, \{S\}, S , P)$

$$P = \begin{cases} S \rightarrow aSb \\ S \rightarrow ab \end{cases}$$



Văn phạm phi ngữ cảnh

- **Một số khái niệm:** dẫn xuất trực tiếp, gián tiếp trái nhất, phải nhất
- **Dẫn xuất trái nhất:**

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

- **Dẫn xuất phải nhất:**

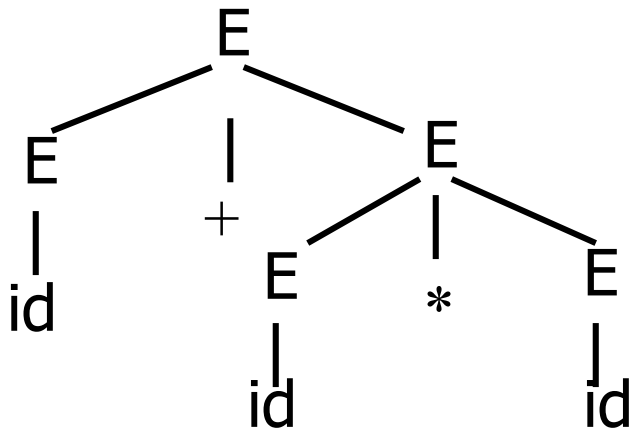
$$\mathbf{E} \rightarrow -E \rightarrow -(E) \rightarrow -(E+E) \rightarrow -(E+id) \rightarrow -(id+id)$$

- **VD:** Cho G: S → AB; A → aA | a; B → bB | b;
  (a) S → AB → aAB → aaAB → aaaB → aaabB → aaabb
  (b) S → AB → AbB → Abb → aAbb → aaAbb → aaabb
  (c) S → AB → aAB → aAbB → aAbb → aaAbb → aaabb
  (d) S → AB → aAB → aaAB → aaAbB → aaabB → aaabb

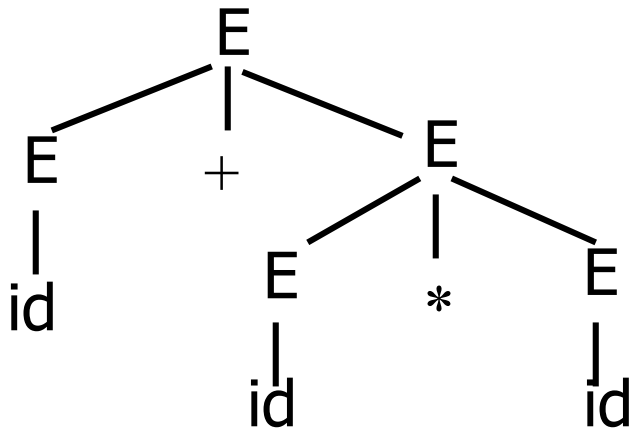
Văn phạm phi ngữ cảnh

- **Nhận xét:**
  - **Top-down parsing** cố gắng tìm ra dẫn xuất trái nhất của chương trình nguồn.
  - **Bottom-up parsing** cố tìm ra cây dẫn xuất phải nhất của chương trình nguồn.
  - **Parse tree** là dạng biểu diễn hình học của dẫn xuất.

Top-down v.s. Bottom-up



Top-down parsing



Bottom-up parsing

$E \Rightarrow E + E$

$\Rightarrow id + E$

$\Rightarrow id + E * E$

$\Rightarrow id + id * E$

$\Rightarrow id + id * id$

$E \Rightarrow E + E$

$\Rightarrow E + E * E$

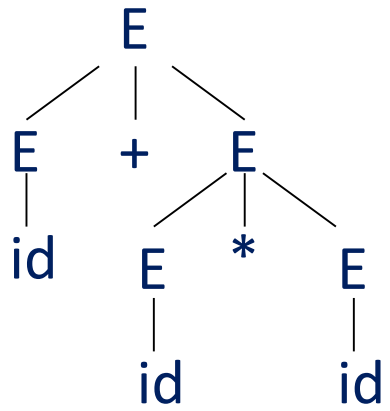
$\Rightarrow E + E * id$

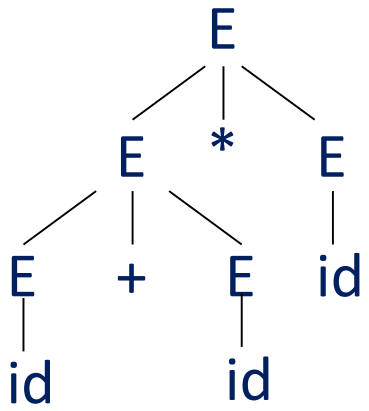
$\Rightarrow E + id * id$

$\Rightarrow id + id * id$

Văn phạm nhập nhằng

- Tính nhập nhằng của văn phạm (**ambiguity**): Một văn phạm sinh ra nhiều hơn một **parse tree** cho một câu được gọi là văn phạm nhập nhằng (mơ hồ, đa nghĩa). Nói cách khác một văn phạm nhập nhằng sẽ sinh ra nhiều hơn một dẫn xuất trái nhất hoặc dẫn xuất phải nhất cho cùng một câu.
  - $E \rightarrow E+E \rightarrow id+E \rightarrow id+E*E \rightarrow id+id*E \rightarrow id+id*id$
  - $E \rightarrow E*E \rightarrow E+E*E \rightarrow id+E*E \rightarrow id+id*E \rightarrow id+id*id$





Văn phạm nhập nhằng

- **VD**: Xét văn phạm sau:

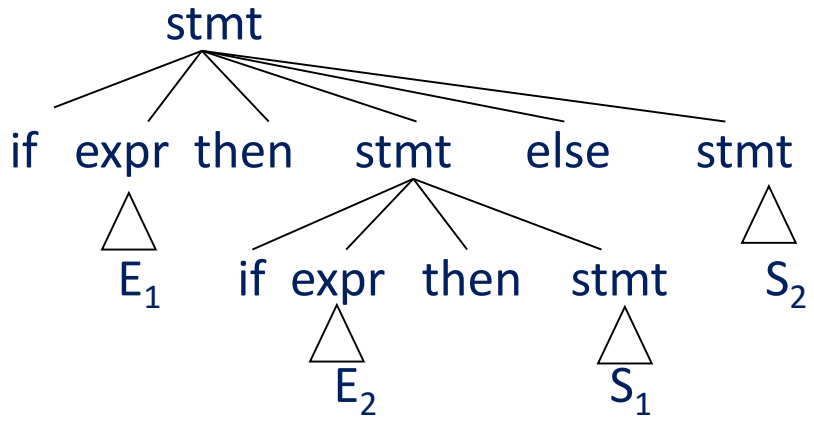
stmt → **if** expr **then** stmt
| **if** expr **then** stmt **else** stmt
| otherstmt

- Với cùng một câu lệnh:

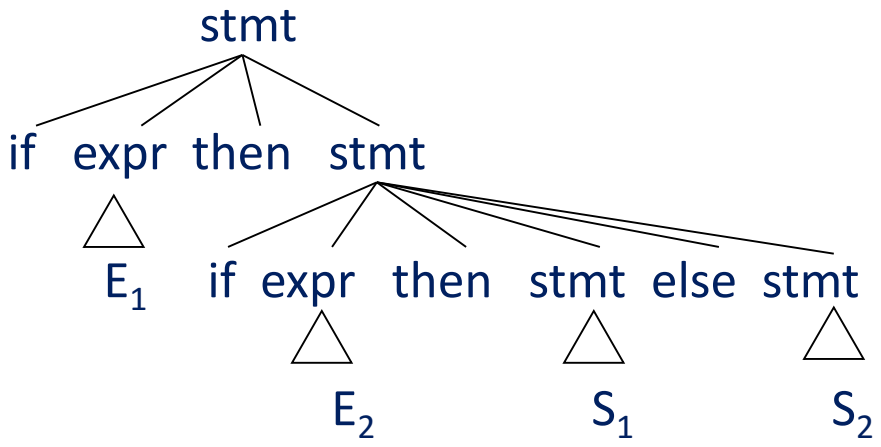
if E_1 **then if** E_2 **then** S_1 **else** S_2

sẽ có hai parse tree:

1:



2:



Văn phạm nhập nhằng

- Vấn đề với CFG:
  - Khó khăn trong chuyển đổi sang **EBNF**
  - Không thể quyết định nên áp dụng luật sinh nào tại mỗi bước
  - Không thể xác định khi nào thì áp dụng luật sinh ε: $A \rightarrow \varepsilon$

Đối với hầu hết parsers, đòi hỏi văn phạm phải là văn phạm không nhập nhằng.

Văn phạm nhập nhằng

- **VD** : Có thể loại bỏ sự nhập nhằng trong vd trước, ta đưa ra qui tắc "Khớp mỗi **else** với một **then** chưa khớp gần nhất trước đó". Với qui tắc này, ta viết lại văn phạm trên như sau:

  Stmt → matched_stmt | unmatched_stmt

  matched_stmt → **if** expr **then** matched_stmt
  else matched_stmt
  | otherstmt

  unmatched_stmt → **if** expr **then** stmt
  | **if** expr **then** matched_stmt
  else unmatched_stmt

Văn phạm đệ quy trái

- Một văn phạm được gọi là đệ qui trái (left recursion) nếu tồn tại một dẫn xuất có dạng $A \xrightarrow{+} A\alpha$ (trong đó A là 1 kí hiệu chưa kết thúc, α là một xâu).
- Các phương pháp phân tích từ trên xuống không thể xử lí văn phạm đệ qui trái, do đó cần phải biến đổi văn phạm để loại bỏ các đệ qui trái.
- Đệ qui trái có hai loại:
  - Loại trực tiếp: Có dạng $A \rightarrow A\alpha$
  - Loại gián tiếp: Gây ra do dẫn xuất của hai hoặc nhiều bước

  VD: $S \rightarrow Aa \mid b$; $A \rightarrow Sc \mid d$

Văn phạm đệ quy trái

- **Loại bỏ đệ quy trái trực tiếp:**
  - Ta nhóm các luật sinh thành

    $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid ..... \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid ..... \mid \beta_n$

    trong đó $\beta_1 \ldots \beta_n$ không bắt đầu với A
  - Thay luật sinh trên bởi các luật sinh sau:

    $A \rightarrow \beta_1 A' \mid \beta_2 A' \mid ..... \mid \beta_n A'$

    $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid ..... \mid \alpha_m A' \mid \varepsilon$
- **VD**: Thay luật sinh $A \rightarrow A\alpha \mid \beta$ bởi

  $A \rightarrow \beta A'$

  $A' \rightarrow \alpha A' \mid \varepsilon$

Văn phạm đệ quy trái

- **Loại bỏ đệ qui trái gián tiếp:**

1. Sắp xếp các ký hiệu không kết thúc theo thứ tự $A_1, A_2, ..., A_n$

2. Áp dụng thuật toán sau:

for i:=1 **to** n **do {**

for j:=1 **to** i -1 **do {**

Thay luật sinh dạng $A_i \rightarrow A_j\gamma$ bởi

$A_i \rightarrow \beta_1\gamma \mid \beta_2\gamma \mid ..... \mid \beta_k\gamma$, trong đó

$A_j \rightarrow \beta_1 \mid \beta_2 \mid ..... \mid \beta_k$

}

Loại bỏ đệ qui trái trực tiếp trong số các luật sinh A_i

};

Văn phạm đệ quy trái

- **VD:** Cho văn phạm sau:

$$S \rightarrow Aa \mid b$$
$$A \rightarrow Ac \mid Sd \mid f$$

- Nếu ta sắp theo thứ tự S, A, khi đó đối với S không có đệ quy trái trực tiếp, Với A:
  - thay thế $A \rightarrow Sd$ bằng $A \rightarrow Aad \mid bd$. Khi đó ta thu được các luật $A \rightarrow Ac \mid Aad \mid bd \mid f$
  - Sau đó khử đệ quy trực tiếp của A nhận được:

$$A \rightarrow bdA' \mid fA' \mid A' \rightarrow cA' \mid adA' \mid \varepsilon$$

  - Kết quả ta nhận được văn phạm không đệ quy trái:

$$S \rightarrow Aa \mid b$$
$$A \rightarrow bdA' \mid fA'$$
$$A' \rightarrow cA' \mid adA' \mid \varepsilon$$

Văn phạm đệ quy trái

- Nếu ta sắp theo thứ tự A, S, khi đó ta khử đệ quy trái trực tiếp đối với A:

  $A \rightarrow SdA' \mid fA'$

  $A' \rightarrow cA' \mid \varepsilon$

  - Với S, thay thế $S \rightarrow Aa$ bằng $S \rightarrow SdA'a \mid fA'a$. Khi đó ta thu được các luật $S \rightarrow SdA'a \mid fA'a \mid b$
  - Sau khi khử đệ quy trực tiếp của S nhận được:

    $S \rightarrow fA'aS' \mid bS'$

    $S' \rightarrow dA'aS' \mid \varepsilon$
  - Kết quả ta nhận được văn phạm không đệ quy trái:

    $S \rightarrow fA'aS' \mid bS'$

    $S' \rightarrow dA'aS' \mid \varepsilon$

    $A \rightarrow SdA' \mid fA'$

    $A' \rightarrow cA' \mid \varepsilon$

Phân tích cú pháp từ trên xuống

- **Top-down parser**: Cây cú pháp được tạo từ gốc tới lá. Có một số kĩ thuật SA từ trên xuống như:
  - SA đệ quy lùi (Recursive-Descent parsing);
  - SA đoán trước (Predictive parsing);
  - SA đoán trước đệ quy (recursive predictive parsing);
  - SA đoán trước không đệ quy (non-recursive predictive parsing);
  - ...
- **Recursive-Descent Parsing**:
  - Backtracking (nếu lựa chọn luật sinh này không thỏa mãn thì quay lui thử áp dụng luật sinh khác).
  - Kỹ thuật tổng quát, nhưng ko được sử dụng rộng rãi;
  - Không hiệu quả…

Phân tích cú pháp từ trên xuống

- **Predictive Parsing**: O(n)
  - Không quay lui, nhưng đòi hỏi văn phạm phải được thừa số hóa trái (left-factored);
  - Chỉ áp dụng cho một lớp con của CFG là văn phạm LL(k) (Left-to-right parse, Leftmost-derivation, k-symbol lockahead);
  - Phép thừa số hóa trái (left-factoring) là phép biến đổi CFG để có được một văn phạm thuận tiện cho việc phân tích dự đoán.
- Ý tưởng: khi không rõ luật sinh nào trong hai luật sinh có thể dùng để khai triển một ký hiệu chưa kết thúc A, ta có thể viết lại các A-luật sinh nhằm "hoãn" lại việc quyết định cho đến khi thấy đủ yếu tố cho một lựa chọn đúng.

Phép thừa số hóa trái

- **VD**: Ta có hai luật sinh

  stmt → **if** expr **then** stmt **else** stmt

  | **if** expr **then** stmt

  sau khi đọc if, ta không thể ngay lập tức quyết định sẽ dùng luật sinh nào để mở rộng stmt

- **Phép thừa số hóa trái:**
  - Giả sử có luật sinh

    $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots\ldots \mid \alpha\beta_n \mid \gamma$

    (α là tiền tố chung dài nhất của các luật sinh, γ không bắt đầu bởi α)
  - Luật sinh trên được biến đổi thành:

    $A \rightarrow \alpha A' \mid \gamma$

    $A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots\ldots \mid \beta_n$

Phép thừa số hóa trái

- **VD**:

$A \rightarrow \underline{a}bB \mid \underline{a}B \mid cdg \mid cdeB \mid cdfB$

$\Downarrow$

$A \rightarrow aA' \mid \underline{cdg} \mid \underline{cde}B \mid \underline{cdf}B$

$A' \rightarrow bB \mid B$

$\Downarrow$

$A \rightarrow aA' \mid cdA''$

$A' \rightarrow bB \mid B$

$A'' \rightarrow g \mid eB \mid fB$

- **VD**:

$A \rightarrow ad \mid a \mid ab \mid abc \mid b$

$\Downarrow$

$A \rightarrow aA' \mid b$

$A' \rightarrow d \mid \varepsilon \mid b \mid bc$

$\Downarrow$

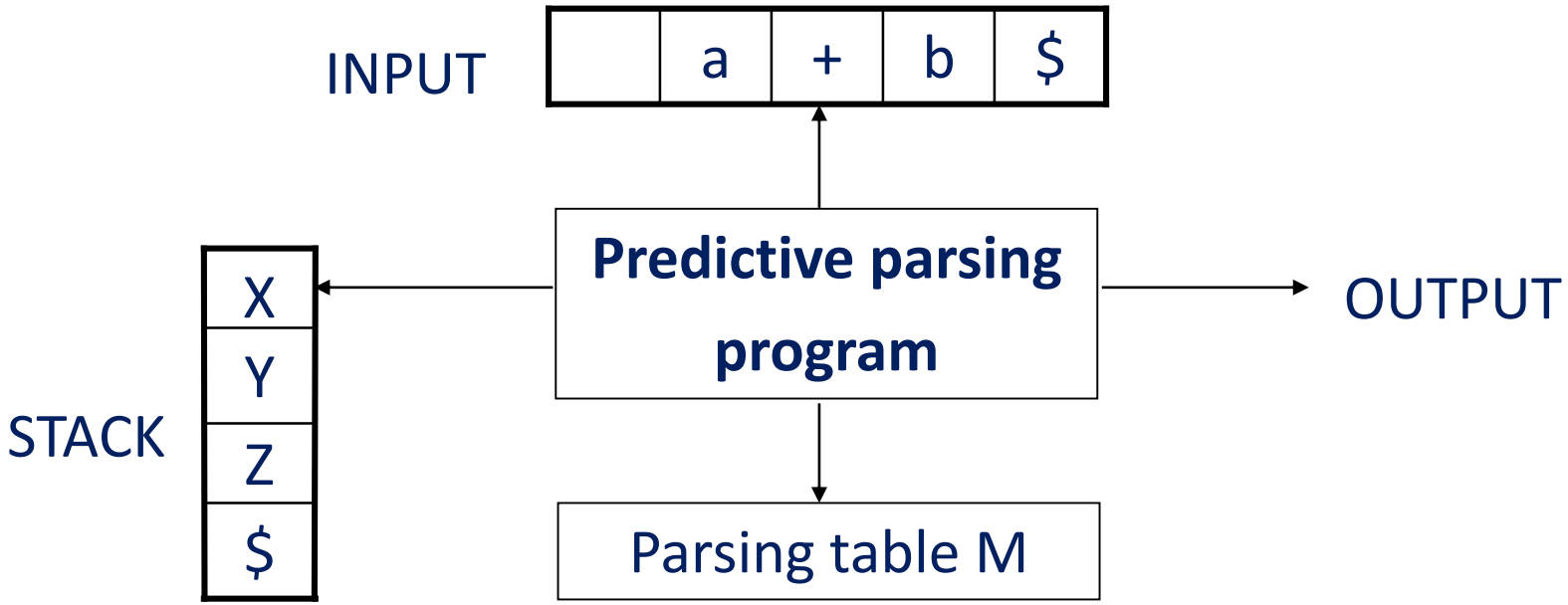
$A \rightarrow aA' \mid b$

$A' \rightarrow d \mid \varepsilon \mid bA''$

$A'' \rightarrow \varepsilon \mid c$

Phân tích cú pháp LL(1)

- **SA** đoán trước không đệ quy còn gọi là LL(1) parser hoặc **table-driven parser**(phân tích dựa trên bảng) hoạt động theo mô hình sau:



- Bộ phân tích cú pháp được điều khiển bởi **Predictive parsing program**

Phân tích cú pháp LL(1)

- **Input buffer:** Dòng dữ liệu cần phân tích. Ở cuối ta thêm vào một ký hiệu đặc biệt $.
- **Output:** Luật sinh được áp dụng trong từng bước dẫn xuất (left-most derivation).
- **Stack:**
  - Các ký hiệu của văn phạm;
  - Stack lúc khởi tạo chỉ chứa $ và ký hiệu bắt đầu S.
  - Khi stack rỗng (i.e. chỉ còn $), quá trình phân tích kết thúc.
- **Parsing table**
  - Mảng hai chiều M[A, a], mỗi hàng là 1 ký hiệu chưa kết thúc, mỗi cột là 1 ký hiệu kết thúc hoặc là ký hiệu đặc biệt $
  - Mỗi ô chỉ chứa không quá 1 luật sinh.

Phân tích cú pháp LL(1)

- **Thuật toán phân tích:** Stack (S$) và bộ đệm chứa chuỗi nhập dạng w$, con trỏ ip trỏ tới ký hiệu đầu tiên của w$;

```
while (X ≠ $)
        X = top (Stack)  và ip trỏ đến a;
        if  (X là ký hiệu kết thúc hoặc $)
                if  (X = a) pop(X) và dịch chuyển ip;
                else error ( ) ;  //gọi chương trình phục hồi lỗi;
        else if ( M[X,a] = X → Y1 Y2 .... Yk ){     // X là non-terminal ;
                pop(X);
                push Yk ,Yk-1, ... ,Y1  vào Stack;
                Xuất ra luật sinh X → Y1 Y2 ... Yk;
        }
        else error ( ) /* Stack rỗng */
}
```

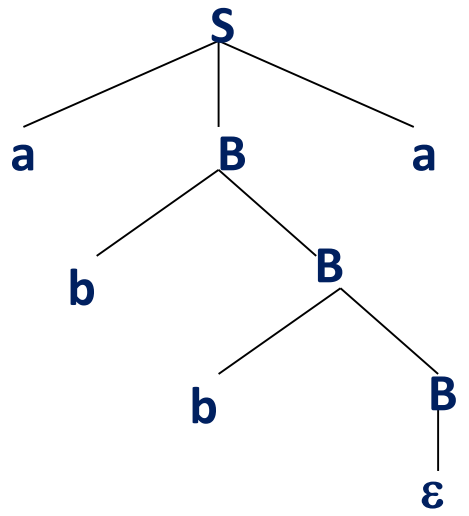
Phân tích cú pháp LL(1)

- **VD**: Cho văn phạm sau: $S \rightarrow aBa$; $B \rightarrow bB \mid \varepsilon$
- Xét quá trình phân tích chuỗi nhập

Stack	input	output
$S	abba$	$S \rightarrow aBa$
$aBa	abba$	
$aB	bba$	$B \rightarrow bB$
$aBb	bba$	
$aB	ba$	$B \rightarrow bB$
$aBb	ba$	
$aB	a$	$B \rightarrow \varepsilon$
$a	a$	
$	$	thành công

- Outputs: $S \rightarrow aBa$ $B \rightarrow bB$ $B \rightarrow bB$ $B \rightarrow \varepsilon$
- Derivation(left-most): $S \Rightarrow aBa \Rightarrow abBa \Rightarrow abbBa \Rightarrow abba$

	a	b	$
S	$S \rightarrow aBa$		
B	$B \rightarrow \varepsilon$	$B \rightarrow bB$	



Phân tích cú pháp LL(1)

- **VD**: Xét văn phạm

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid id$

- Áp dụng loại bỏ đệ qui trái ta thu được

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid id$

Phân tích cú pháp LL(1)

- **Parsing table M** cho văn phạm trên như sau:

Non-terminal	Input symbol					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Phân tích cú pháp LL(1)

- Với chuỗi nhập có dạng id+id:

stack	input	output
$E	id+id$	$E \rightarrow TE'$
$E'T	id+id$	$T \rightarrow FT'$
$E'T'F	id+id$	$F \rightarrow id$
$ E'T'id	id+id$	
$ E'T'	+id$	$T' \rightarrow \varepsilon$
$ E'	+id$	$E' \rightarrow +TE'$
$ E'T+	+id$	
$ E'T	id$	$T \rightarrow FT'$
$ E'T'F	id$	$F \rightarrow id$
$ E'T'id	id$	
$ E'T'	$	$T' \rightarrow \varepsilon$
$ E'	$	$E' \rightarrow \varepsilon$
$	$	accept

FIRST và FOLLOW

- **Xây dựng bảng phân tích cho văn phạm LL(1):**
- Hàm **FIRST** và **FOLLOW**: Là các hàm xác định các tập hợp cho phép xây dựng bảng phân tích M và phục hồi lỗi theo chiến lược panic-mode.
  - Nếu α là một xâu thì FIRST(α) là tập hợp các ký hiệu kết thúc mà nó có thể bắt đầu ở một chuỗi được dẫn xuất từ α. Nếu $\alpha \Rightarrow^* \varepsilon$ thì ε thuộc FIRST(α)
  - Nếu A là một kí hiệu chưa kết thúc thì FOLLOW(A) là tập các kí hiệu kết thúc mà nó có thể xuất hiện ngay bên phải A trong một dẫn xuất từ S. Nếu $S \Rightarrow^* \alpha A$ thì $ thuộc FOLLOW(A)

Tìm các tập FIRST

- **Quy tắc xác định hàm FIRST:**
  1. Nếu X là kí hiệu kết thúc thì FIRST(X) là {X}
  2. Nếu X → ε là một luật sinh thì thêm ε vào FIRST(X).
  3. Nếu X → $Y_1Y_2Y_3$ ...Y_k là một luật sinh thì:
     - thêm tất cả các ký hiệu kết thúc khác ε của FIRST(Y_1) vào FIRST(X).
     - Nếu ε ∈ FIRST(Y_1) thì tiếp tục thêm vào FIRST(X) tất cả các ký hiệu kết thúc khác ε của FIRST(Y_2).
     - Nếu ε ∈ FIRST(Y_1) ∩ FIRST(Y_2) thì thêm tất cả các ký hiệu kết thúc khác ε ∈ FIRST(Y_3) ...
     - Cuối cùng thêm ε vào FIRST(X) nếu ε ∈ $\bigcap_{i=1}^{k}$ FIRST(Yi)

Tìm các tập FIRST

For all terminals a, First(a) = {a}

For all nonterminals A, First(A) := {}

While there are changes to any First(A)

For each rule $A \rightarrow X_1\ X_2 \ldots X_n$

For each X_i in $\{X_1, X_2, \ldots, X_n\}$

If for all $j<i$ First(X_j) contains λ,

Then

add First(X_i)-{λ} to First(A)

If λ is in First(X_1), First(X_2), ..., and First(X_n)

Then add λ to First(A)

If A is a terminal or λ, then First(A) = {A}.

If A is a nonterminal, then for each rule $A \rightarrow X_1\ X_2 \ldots X_n$, First(A) contains First(X_1) - {λ}.

If also for some $i<n$, First(X_1), First(X_2), ..., and First(X_i) contain λ, then First(A) contains First(X_{i+1})-{λ}.

If First(X_1), First(X_2), ..., and First(X_n) contain λ, then First(A) also contains λ.

Tìm các tập FIRST

exp → term exp’

exp’ → addop term exp’ | λ

addop → + | -

term → factor term’

term’ → mulop factor term’ | λ

mulop → *

factor → (exp) | num

	First
exp	
exp’	λ
addop	+ -
term	(num
term’	λ
mulop	*
factor	(num

Tìm các tập FIRST

- **VD**: Cho văn phạm:

  $E \rightarrow TE'$

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

  $T \rightarrow FT'$

  $T' \rightarrow *FT' \mid \varepsilon$

  $F \rightarrow (E) \mid id$

- Ta tính được hàm FIRST:

  FIRST(F) = {(,id}

  FIRST(T′) = {*, ε}

  FIRST(T) = {(,id}

  FIRST(ε) = {ε}

  FIRST(E′) = {+, ε}

  FIRST(E) = {(,id}

  FIRST(TE′) = {(,id}

  FIRST(+TE′) = {+}

  FIRST(FT′) = {(,id}

  FIRST(*FT′) = {*}

  FIRST((E)) = {(}

  FIRST(id) = {id}

Tìm các tập FOLLOW

- **Qui tắc tính các tập hợp FOLLOW** (chỉ áp dụng cho ký hiệu chưa kết thúc)
  1. Đặt $ vào FOLLOW(S) (S là kí hiệu bắt đầu)
  2. Nếu $A \to \alpha B \beta$ thì mọi phần tử thuộc FIRST(β) ngoại trừ ε đều thuộc FOLLOW(B)
  3. Nếu $A \to \alpha B$ hoặc $A \to \alpha B \beta$ và $\beta \Rightarrow^* \varepsilon$ thì mọi phần tử thuộc FOLLOW(A) đều thuộc FOLLOW(B)
  4. áp dụng các quy tắc trên đến khi không thể thêm gì vào các tập FOLLOW

Tìm các tập FOLLOW

Follow(S) = {$}

FOR each A in V-{S}

Follow(A)={}

WHILE change is made to some Follow sets

FOR each production A → X_1 X_2 ... X_n,

FOR each nonterminal X_i

Add First(X_{i+1} X_{i+2}...X_n)-{λ}
into Follow(X_i).
(NOTE: If i=n, X_{i+1} X_{i+2}...X_n= λ)

IF λ is in First(X_{i+1} X_{i+2}...X_n) THEN

Add Follow(A) to Follow(X_i)

If A is the start symbol, then $ is in Follow(A).

If there is a rule A → Y X Z, then First(Z) - {λ} is in Follow(X).

If there is production B → X A Y and λ is in First(Y), then Follow(A) contains Follow(B).

Tìm các tập FOLLOW

exp → term exp’

exp’ → addop term exp’ | λ

addop → + | -

term → factor term’

term’ → mulop factor term’ |λ

mulop → *

factor → (exp) | num

	First	Follow
exp	(num	$)
exp’	λ + -	$)
addop	+ -	
term	(num	+ - $)
term’	λ *	
mulop	*	
factor	(num	

Tìm các tập FOLLOW

VD: Cho văn phạm:

$E \rightarrow TE'$

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

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id$

- Ta tính được hàm **FOLLOW**:

FOLLOW(E) = { $,) }

FOLLOW(E′) = { $,) }

FOLLOW(T) = { +,), $ }

FOLLOW(T′) = { +,), $ }

FOLLOW(F) = {+, *,), $ }

- **VD**: Xét văn phạm

$E \rightarrow TE'$

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

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id$

- Khi đó:

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }

FIRST(E') = {+, ε }

FIRST(T') = {*, ε }

FOLLOW(E) = FOLLOW(E') = { $,) }

FOLLOW(T) = FOLLOW(T') = { +,), $ }

FOLLOW(F) = {*,+,), $ }

Xây dựng bảng phân tích LL(1)

- Thuật giải xây dựng **Parsing table M** của văn phạm G:

A. FOR EARCH nonterminal A và luật sinh $A \rightarrow X$

FOR EACH ký hiệu kết thúc a trong First(X)

cho $A \rightarrow X$ vào M[A, a]

IF ε có trong First(X) THEN

FOR EACH ký tự a trong Follow(A)

thêm $A \rightarrow X$ vào M[A, a]

B. Ô còn trống trong bảng tương ứng với lỗi (error).

Xây dựng bảng phân tích LL(1)

	First	Follow
exp	{(, num}	{$,)}
exp’	{+,-, λ}	{$,)}
addop	{+,-}	{(,num}
term	{(,num}	{+,-,),$}
term’	{*, λ}	{+,-,),$}
mulop	{*}	{(,num}
factor	{(, num}	{*,+,-,),$}

1 exp → term exp’
2 exp’ → addop term exp’
3 exp’ → λ
4 addop → +
5 addop → -
6 term → factor term’
7 term’ → mulop factor term’
8 term’ → λ
9 mulop → *
10 factor → (exp)
11 factor → num

	(	)	+	-	*	n	$
exp	1					1	
exp’		3	2	2			3
addop			4	5			
term	6					6	
term’		8	8	8	7		8
mulop					9		
factor	10					11	

Xây dựng bảng phân tích LL(1)

- Áp dụng giải thuật ta xây dựng được **Parsing table M** cho văn phạm trên như sau:

Non-terminal	Input symbol					
	id	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Văn phạm LL(1)

- **Văn phạm LL(1):** Văn phạm mà bảng phân tích trên mỗi ô có không quá 1 dòng (1 luật sinh) được gọi là văn phạm LL(1).
- VD: Văn phạm sau không phải là văn phạm LL(1).

$S \rightarrow i\ C\ t\ S\ E \mid a$
$E \rightarrow e\ S \mid \varepsilon$
$C \rightarrow b$

FIRST(iCtSE) = {i}
FIRST(a) = {a}
FIRST(eS) = {e}
FIRST(ε) = {ε}
FIRST(b) = {b}

FOLLOW(S) = { $,e }
FOLLOW(E) = { $,e }
FOLLOW(C) = { t }

	a	b	e	i	t	$
S	$S \rightarrow a$			$S \rightarrow iCtSE$		
E			$E \rightarrow e\ S$ $E \rightarrow \varepsilon$			$E \rightarrow \varepsilon$
C		$C \rightarrow b$				

Văn phạm LL(1)

- **Tính chất của văn phạm LL(1):** Văn phạm G là LL(1) khi và chỉ khi những điều kiện sau thỏa mãn đối với 2 luật phân biệt bất kỳ $A \to \alpha$ và $A \to \beta$
  1. Cả α và β không dẫn ra từ các xâu có ký hiệu đầu giống nhau ($First(\alpha) \cap First(\beta) = \varnothing$).
  2. Không đồng thời ε thuộc $First(\alpha)$ và $First(\beta)$.
  3. Nếu β dẫn tới ε, thì α không thể dẫn ra xâu bắt đầu bằng ký hiệu trong FOLLOW(A). (nếu $\varepsilon \in First(\beta)$ thì $Follow(A) \cap First(\alpha) = \varnothing$)

Văn phạm LL(1)

- **Văn phạm không phải là LL(1):**
  - Văn phạm đệ quy trái không thể là LL(1) grammar.

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

➔ mọi ký hiệu kết thúc trong FIRST(β) cũng có mặt trong FIRST($A\alpha$) vì rằng $A\alpha \Rightarrow \beta\alpha$.

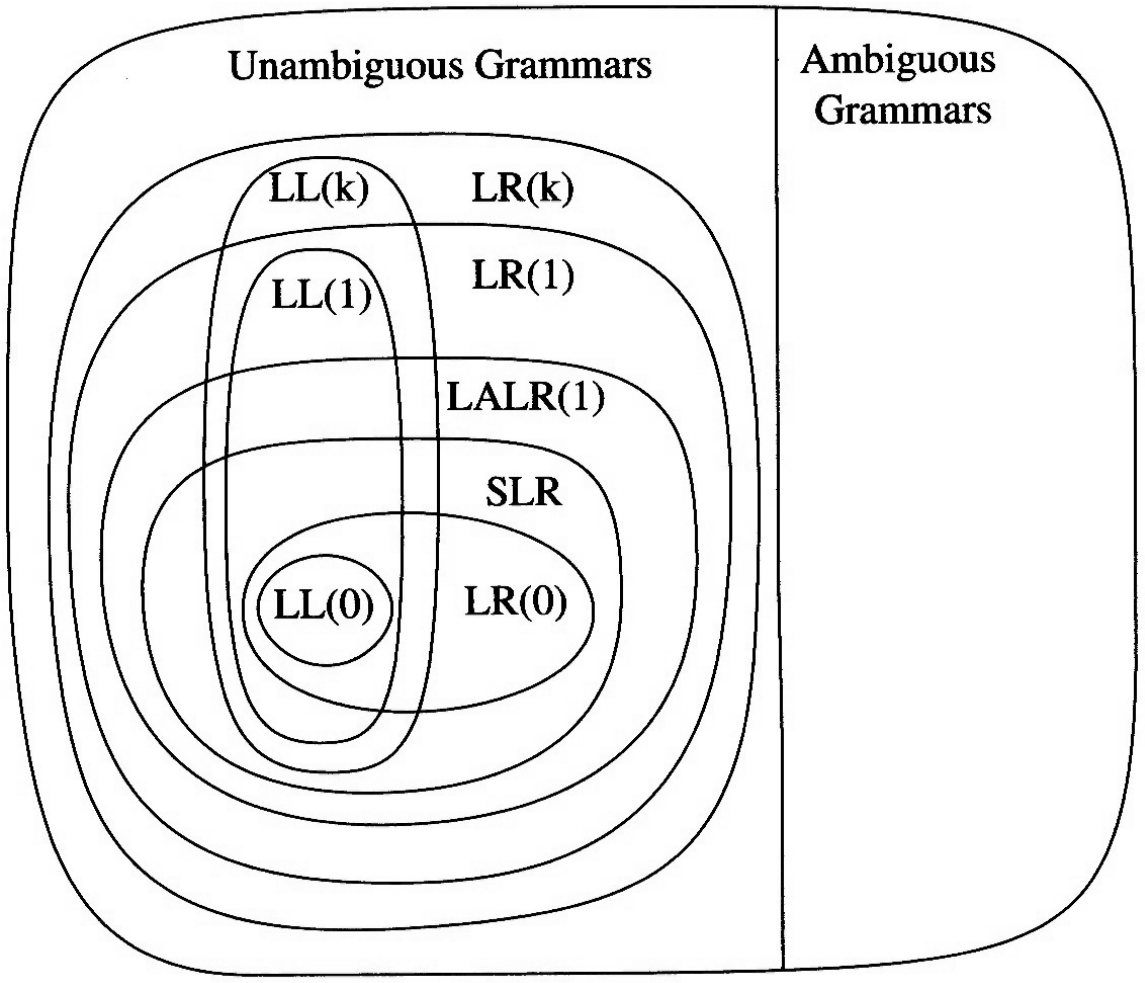
➔ Nếu β là ε, mọi ký hiệu kết thúc trong FIRST(α) cũng có trong FIRST($A\alpha$) và FOLLOW(A).

  - Văn phạm chưa thừa số hóa trái, không thể là LL(1) grammar

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$$

➔mọi ký hiệu kết thúc trong FIRST($\alpha\beta_1$) cũng có trong FIRST($\alpha\beta_2$).

  - Văn phạm nhập nhằng không thể là LL(1) grammar.



- **LL(*k*):**
  - —**L**eft-to-right, **L**eftmost derivation, *k* tokens lookahead
- **LR(*k*):**
  - —**L**eft-to-right, **R**ightmost derivation, *k* tokens lookahead
- **SLR:**
  - —**S**imple **LR** (uses “follow sets”)
- **LALR:**
  - —**L**ook**A**head **LR** (uses “lookahead sets”)