

CHƯƠNG III

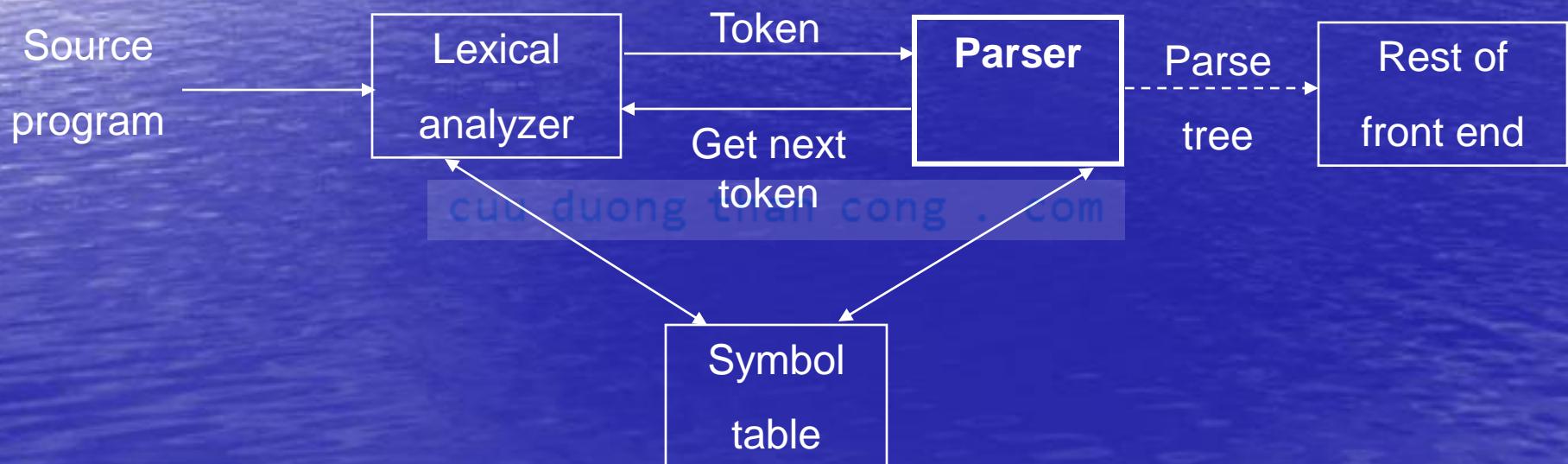
Phân tích cú pháp

Mục tiêu:

- Nắm được vai trò của giai đoạn phân tích cú pháp
- Văn phạm phi ngữ cảnh (context-free grammar), cách phân tích cú pháp từ dưới lên- từ trên xuống (top-down and bottom-up parsing)
- Bộ phân tích cú pháp LR

Vai trò của bộ phân tích cú pháp

- Đây là giai đoạn thứ 2 của quá trình biên dịch
- Nhiệm vụ chính: Nhận chuỗi các token từ bộ phân tích từ vựng và xác định chuỗi đó có được sinh ra bởi văn phạm của ngôn ngữ nguồn không



- Các phương pháp phân tích cú pháp (PTCP) chia làm hai loại: Phân tích từ trên xuống (top- down parsing) và phân tích từ dưới lên (bottom- up parsing)
- Trong quá trình biên dịch xuất hiện nhiều lỗi trong giai đoạn PTCP do đó bộ phân tích cú pháp phải phát hiện và thông báo lỗi chính xác cho người lập trình đồng thời không làm chậm những chương trình được viết đúng

Văn phạm phi ngữ cảnh

- Để định nghĩa cấu trúc của ngôn ngữ lập trình ta dùng văn phạm phi ngữ cảnh (Context-free grammars) hay gọi tắt là một văn phạm
- Một văn phạm bao gồm:
 - Các kí hiệu kết thúc (**terminals**): Chính là các token
 - Các kí hiệu chưa kết thúc (**nonterminals**): Là các biến kí hiệu tập các xâu kí tự
 - Các luật sinh (**productions**): Xác định cách thức hình thành các xâu từ các kí hiệu kết thúc và chưa kết thúc

Ví dụ 3.1: Văn phạm sau định nghĩa các biểu thức số học đơn giản

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

$A \rightarrow + \mid - \mid * \mid / \mid \uparrow$

Trong đó E, A là các kí tự chưa kết thúc (E còn là kí tự bắt đầu), các kí tự còn lại là các kí tự kết thúc

cuu duong than cong . com

- Dẫn xuất (derivation): Ta nói $\alpha A\beta \Rightarrow \alpha\tilde{\gamma}\beta$ nếu $A \rightarrow \tilde{\gamma}$ là một luật sinh (\Rightarrow đọc là dẫn xuất hoặc suy ra)
- Nếu $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$ thì ta nói rằng α_1 dẫn xuất α_n
- Kí hiệu: \Rightarrow^* là dẫn xuất ≥ 0 bước, \Rightarrow^+ là dẫn xuất ≥ 1 bước
- Cho văn phạm G với kí tự bắt đầu là S, L(G) là ngôn ngữ được sinh bởi G. Mọi xâu trong L(G) chỉ chứa các kí hiệu kết thúc của G

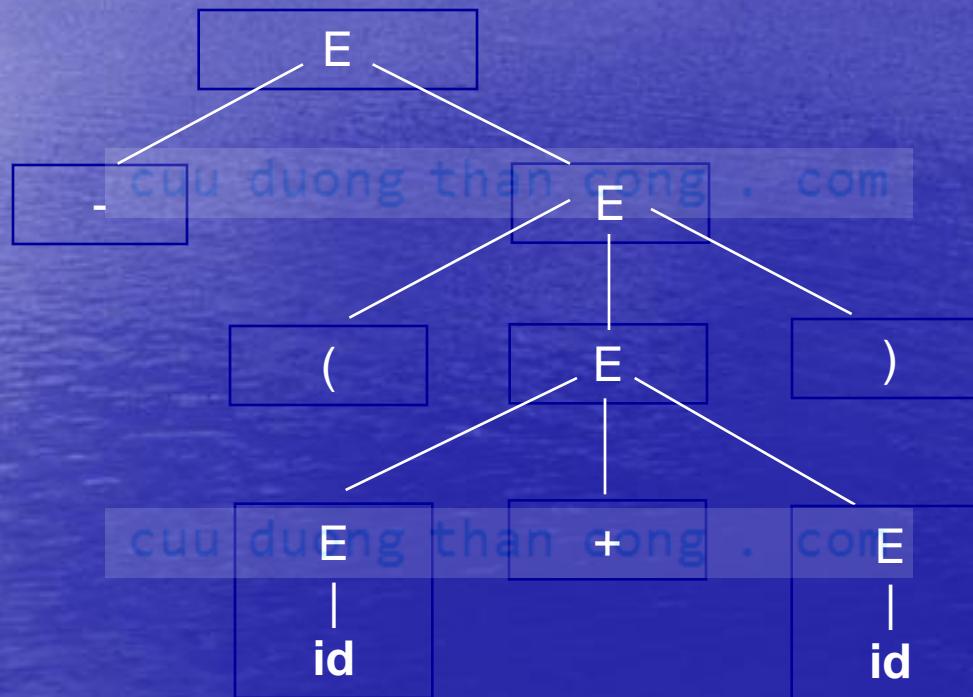
- Ta nói một xâu $w \in L(G)$ nếu và chỉ nếu $S \Rightarrow^+ w$, w được gọi là một câu (sentence) của văn phạm G
- Một ngôn ngữ được sinh bởi văn phạm phi ngữ cảnh được gọi là ngôn ngữ phi ngữ cảnh (context-free language)
- Hai văn phạm được gọi là tương đương nếu sinh ra cùng một ngôn ngữ
- Nếu $S \Rightarrow^* \alpha$ (α có thể chứa kí hiệu chưa kết thúc) thì ta nói α là một dạng câu (sentence form) của G . Một câu là một dạng câu không chứa kí hiệu chưa kết thúc

Ví dụ 3.2: Xâu $-(\text{id}+\text{id})$ là một câu của văn phạm trong ví dụ 3.1 vì

$$\begin{aligned} E &\Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(id+E) \Rightarrow \\ &(id+id) \end{aligned}$$

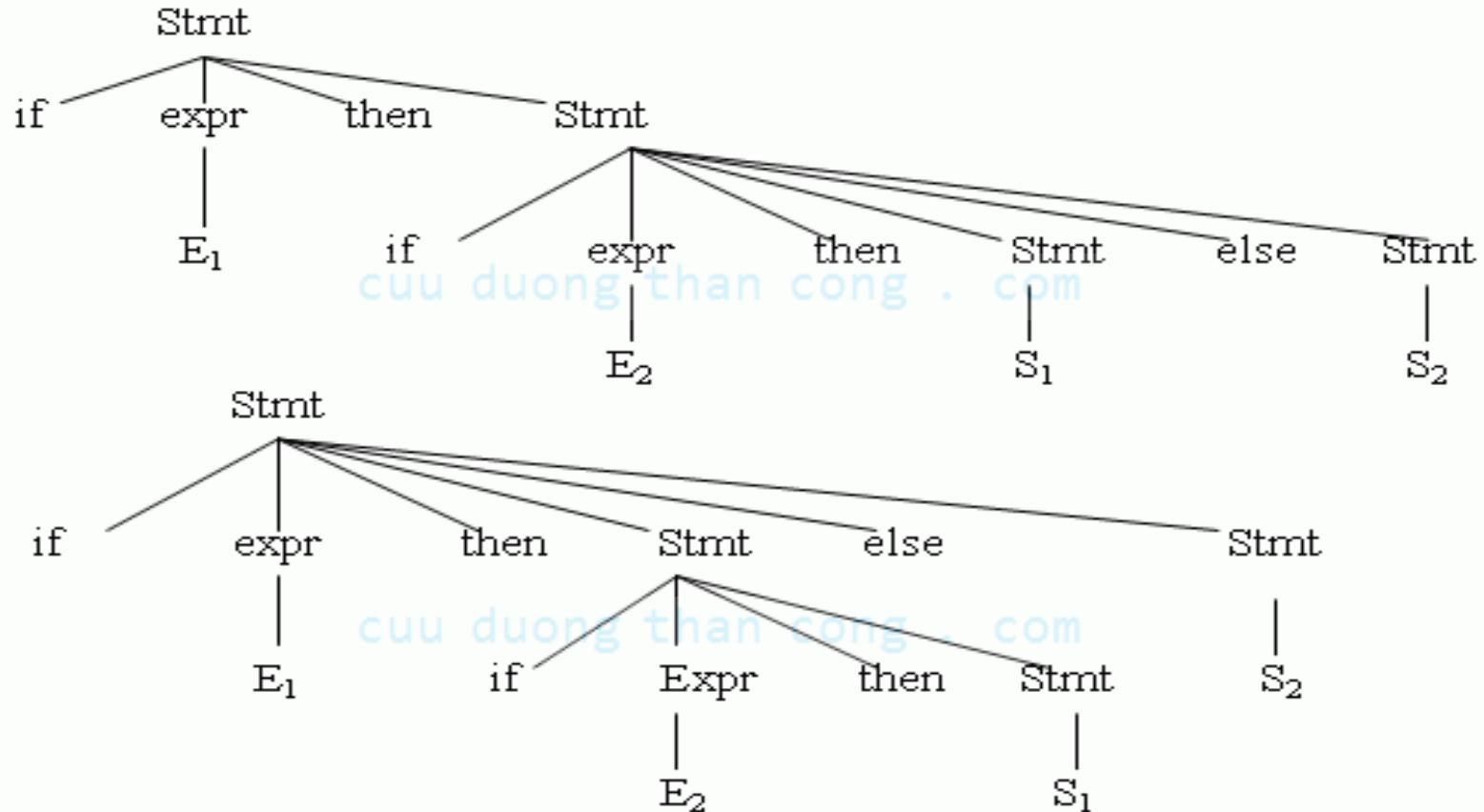
- Một dẫn xuất được gọi là trái nhất (leftmost) nếu tại mỗi bước kí hiệu chưa kết thúc ngoài cùng bên trái được thay thế, kí hiệu \Rightarrow_{lm} . Nếu $S \Rightarrow^*_{lm} \alpha$ thì α được gọi là dạng câu trái
- Tương tự ta có dẫn xuất phải nhất (rightmost) hay còn gọi là dẫn xuất chính tắc, kí hiệu \Rightarrow_{rm}

- Cây phân tích cú pháp (parse tree) là dạng biểu diễn hình học của dân xuất. Ví dụ parse tree cho biểu thức $-(id+id)$ là:



- Tính mơ hồ của văn phạm (ambiguity): Một văn phạm sinh ra nhiều hơn một parse tree cho một câu được gọi là văn phạm mơ hồ. Nói cách khác một văn phạm mơ hồ sẽ sinh ra nhiều hơn một dẫn xuất trái nhất hoặc dẫn xuất phải nhất cho cùng một câu.
- Loại bỏ sự mơ hồ của văn phạm: Ta xét ví dụ văn phạm sau
 Stmt → if expr then stmt
 | if expr then stmt else stmt
 | other

- Văn phạm trên là mơ hồ vì với cùng một câu lệnh "**if E1 then if E2 then S1 else S2**" sẽ có hai parse tree:



- Để loại bỏ sự mơ hồ này 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

 | **other**

unmatched_stmt → **if** expr **then** Stmt

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

matched_stmt **else**

 unmatched_stmt

- Loại bỏ đệ qui trái: Một văn phạm được gọi là đệ qui trái (left recursion) nếu tồn tại một dẫn xuất có dạng $A \Rightarrow^+ A\alpha$ (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

- Với đệ qui trái trực tiếp: Ta nhóm các luật sinh thành

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

Thay luật sinh trên bởi các luật sinh sau:

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

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$$

Ví dụ 3.3: Thay luật sinh $A \rightarrow A\alpha \mid \beta$ bởi

$$A \rightarrow \beta A'$$

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

- Với đê qui trái gián tiếp: Ta dùng thuật toán sau

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

2. **for** $i:=1$ **to** n **do**

begin

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

begin

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

$A_i \rightarrow \beta_1\gamma | \beta_2\gamma | \dots | \beta_k\gamma$ trong đó

$A_j \rightarrow \beta_1 | \beta_2 | \dots | \beta_k$ là tất cả các luật sinh hiện tại

end;

- Tạo ra nhân tố trái (left factoring) là một phép biến đổi văn phạm rất có ích để có được một văn phạm thuận tiện cho việc phân tích dự đoán
- Ý tưởng cơ bản là khi không rõ luật sinh nào trong hai luật sinh khả triển có thể dùng để khai triển một ký hiệu chưa kết thúc A, chúng 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.

Ví dụ 3.3: Ta có hai luật sinh

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

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

- Cách tạo nhân tố trái: Giả sử có luật sinh $A \rightarrow \alpha \beta_1 \mid \alpha \beta_2 \mid \dots \mid \alpha \beta_n \mid \gamma$ (α là tiền tố chung dài nhất của các luật sinh, γ không bắt đầu bởi α)

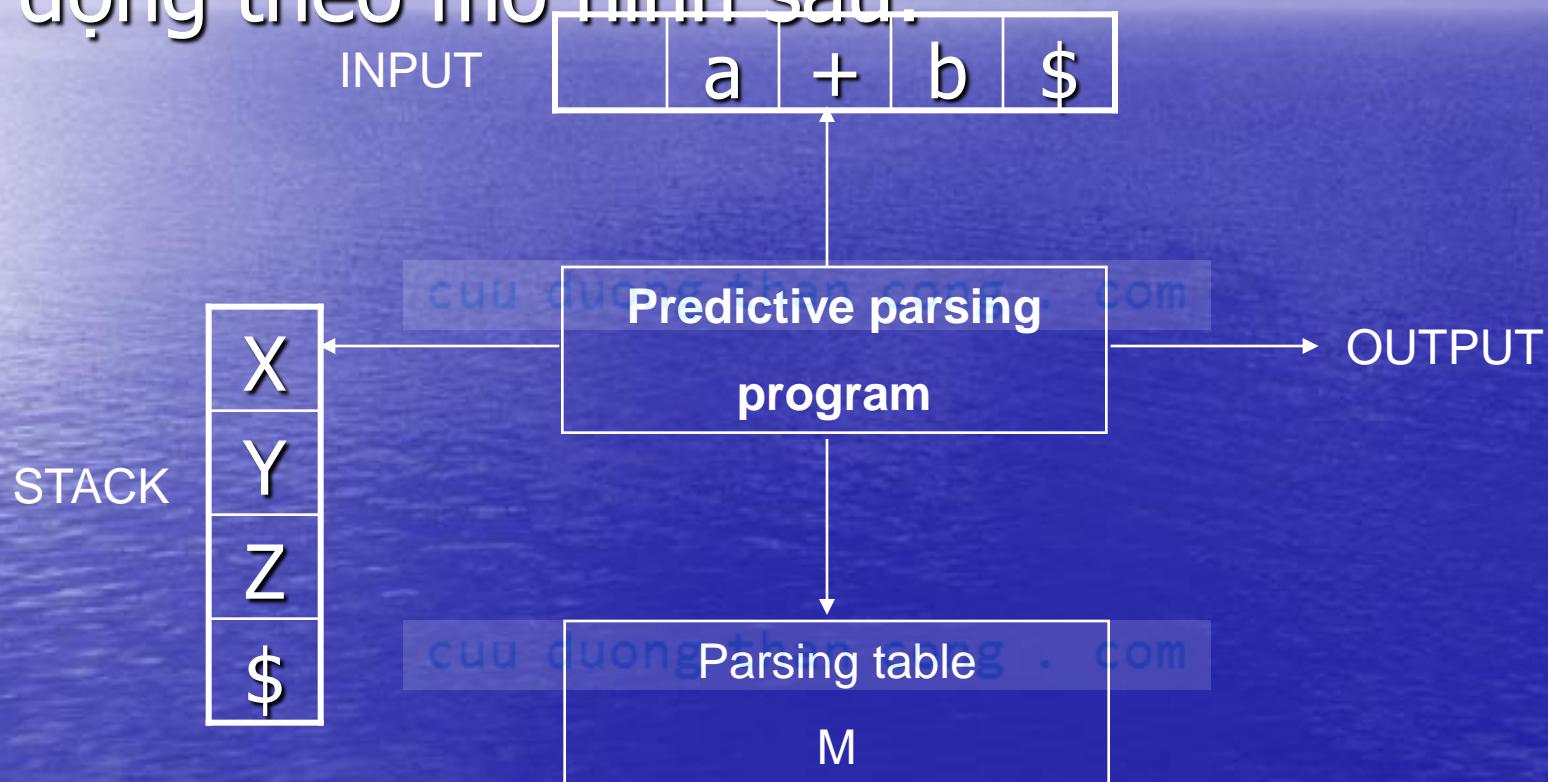
Luật sinh trên được biến đổi thành:

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

Phân tích cú pháp từ trên xuống

- Phân tích cú pháp (PTCP) từ trên xuống được xem như một cỗ gǎng tìm kiếm một dẫn xuất trái nhất cho chuỗi nhập. Nó cũng có thể xem như một cỗ gǎng xây dựng cây phân tích cú pháp bắt đầu từ nút gốc và phát sinh dần xuống lá
- PTCP từ trên xuống đơn giản hơn PTCP từ dưới lên nhưng bị giới hạn về mặt hiệu quả
- Có một số kĩ thuật PTCP từ trên xuống như: PTCP đệ qui lùi, PTCP đoán trước,

- PTCP đoán trước không đệ qui (nonrecursive predictive parsing) hoạt động theo mô hình sau:



- **INPUT** là bộ đệm chứa chuỗi cần phân tích, kết thúc bởi ký hiệu \$
- **STACK** chứa một chuỗi các ký hiệu văn phạm với ký hiệu \$ nằm ở đáy STACK. Khoảng đầu STACK chứa kí hiệu bắt đầu S trên đỉnh
- **Parsing table M** là một mảng hai chiều dạng $M[A,a]$, trong đó A là ký hiệu chưa kết thúc, a là ký hiệu kết thúc hoặc \$.
- Bộ phân tích cú pháp được điều khiển bởi **Predictive parsing program**

- **Predictive parsing program** hoạt động như sau: Chương trình xét ký hiệu X trên đỉnh Stack và ký hiệu nhập hiện hành a
 1. Nếu $X = a = \$$ thì quá trình PTCP kết thúc thành công
 2. Nếu $X = a \neq \$$, đẩy X ra khỏi Stack và đọc ký hiệu nhập tiếp theo.
 3. Nếu X là ký hiệu chưa kết thúc thì chương trình truy xuất đến phần tử $M[X,a]$ trong **Parsing table M**:
 - Nếu $M[X,a]$ là một luật sinh có dạng $X \rightarrow UYV$ thì đẩy X ra khỏi đỉnh Stack và đẩy V, Y, U vào Stack (với U trên đỉnh Stack), đồng thời bộ xuất ra OUTPUT luật sinh $X \rightarrow UYV$

Ví dụ 3.4: Xét văn phạm

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

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

$$F \rightarrow (E) \mid id$$

Loại bỏ đệ qui trái ta thu được

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

$$F \rightarrow (E) \mid id$$

Giả sử xâu input nhập vào là **id+id*id**

● Parsing table M cho văn phạm trên như

Sau Non- termin al	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	$F \rightarrow id$			$F \rightarrow (E)$		

STACK	INPUT	OUTPUT
\$ E	id + id * id \$	
\$ E' T	id + id * id \$	E → T E'
\$ E' T' F	id + id * id \$	T → F T'
\$ E' T' id	id + id * id \$	F → id
\$ E' T'	+ id * id \$	
\$ E'	+ id * id \$	T' → ε
\$ E' T +	+ id * id \$	E' → + T E'
\$ E' T	id * id \$	
\$ E' T' F	id * id \$	T → F T'
\$ E' T' id	id * id \$	F → id
\$ E' T'	* id \$	
\$ E' T' F *	* id \$	T' → * F T'
\$ E' T' F	id \$	
\$ E' T' id	id \$	F → id
\$ E' T'	\$	
\$ E'	\$	T' → ε

- Hàm FIRST và FOLLOW: Là các hàm xác định các tập hợp cho phép xây dựng bảng phân tích M và phục hồi lỗi
- Nếu α là một xâu thì $\text{FIRST}(\alpha)$ là tập hợp các ký hiệu kết thúc mà nó bắt đầu một chuỗi dẫn xuất từ α . Nếu $\alpha \Rightarrow^* \epsilon$ thì ϵ thuộc $\text{FIRST}(\alpha)$
- Nếu A là một kí hiệu chưa kết thúc thì $\text{FOLLOW}(A)$ là tập các kí hiệu kết thúc mà nó xuất hiện ngay bên phải A trong một dạng câu . Nếu $S \Rightarrow^* \alpha A$ thì $\$$ thuộc $\text{FOLLOW}(A)$

- Qui tắc tính các tập hợp FOLLOW
 1. Đặt \$ vào FOLLOW(S) (S là kí hiệu bắt đầu)
 2. Nếu $A \rightarrow \alpha B \beta$ thì mọi phần tử thuộc FIRST(β) ngoại trừ ϵ đều thuộc FOLLOW(B)
 3. Nếu $A \rightarrow \alpha B$ hoặc $A \rightarrow \alpha B \beta$ và $\beta \Rightarrow^* \epsilon$ thì mọi phần tử thuộc FOLLOW(A) đều thuộc FOLLOW(B)

Ví dụ 3.5: Xét văn phạm

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

Khi đó:

$\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{ (, id \}$

$\text{FIRST}(E') = \{ +, \epsilon \}$

$\text{FIRST}(T') = \{ *, \epsilon \}$

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

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

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

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

1. Với mỗi luật sinh $A \rightarrow \alpha$ của văn phạm, thực hiện bước 2 và 3
2. Với mỗi ký hiệu kết thúc $a \in \text{FIRST}(\alpha)$, thêm $A \rightarrow \alpha$ vào $M[A,a]$
3. Nếu $\varepsilon \in \text{FIRST}(\alpha)$ thì đưa luật sinh $A \rightarrow \alpha$ vào $M[A,b]$ với mỗi ký hiệu kết thúc $b \in \text{FOLLOW}(A)$. Nếu $\varepsilon \in \text{FIRST}(\alpha)$ và $\$ \in \text{FOLLOW}(A)$ thì đưa luật sinh $A \rightarrow \alpha$ vào $M[A,\$]$.
4. Ô còn trống trong bảng tương ứng với lỗi (error).

Phân tích cú pháp từ dưới lên

- Giới thiệu một kiểu phân tích cú pháp từ dưới lên tổng quát gọi là phân tích cú pháp Shift –Reduce
- Một phương pháp tổng quát hơn của kỹ thuật Shift - Reduce là phân tích cú pháp LR (LR parsing) sẽ được thảo luận
- Shift –Reduce parsing sẽ cố gắng xây dựng một parse tree cho một xâu nhập vào từ nút lá lên nút gốc. Nói cách khác ta "reducing" từng bước xâu nhập vào đến khi thu được kí hiệu bắt đầu của văn

Ví dụ 3.6: Cho văn phạm :

$S \rightarrow a A B e$

$A \rightarrow A b c | b$

$B \rightarrow d$

Câu **abbcde** có thể thu gọn về S theo các bước sau:

a b b c d e

a A b c d e

a A d e

a A B e

S

Đảo ngược lại quá trình trên ta thu được dẫn xuất phải nhất:

$S \Rightarrow_{rm} a A B e \Rightarrow_{rm} a A d e \Rightarrow_{rm} a A b c d e \Rightarrow_{rm} a b b c d e$

- Handles: Handle của một right-sentential form γ là một luật sinh $A \rightarrow \beta$, một xâu α sao cho $\gamma = \alpha\beta\omega$ và $S \Rightarrow_{rm}^* \alpha A \omega \Rightarrow_{rm} \alpha\beta\omega$. Đôi khi ta còn gọi β là một handle, xâu ω bên phải β chỉ chứa các kí hiệu kết thúc
- Nếu một văn phạm là không mơ hồ thì với mỗi right-sentential form có duy nhất một handle của nó

Ví dụ 3.7: Trong dẫn xuất $S \Rightarrow_{rm}^* aA\overline{Be} \Rightarrow_{rm}^* aA\overline{de} \Rightarrow_{rm}^* a\overline{Abcde} \Rightarrow_{rm}^* \overline{abbcde}$ các handle được gạch chân

Ví dụ 3.8: Xét văn phạm mơ hồ

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

Với cùng một biểu thức $id + id * id$ sẽ có hai dẫn xuất phải nhất (các handle được gạch chéo). Cùng một right sentence form $E + E * id3$ trong trường hợp đầu $id3$ là handle còn trường hợp thứ 2 handle là

$$E + E$$

$$\Rightarrow_{rm} \underline{E + E}$$

$$\Rightarrow_{rm} E + \underline{E * E}$$

$$\Rightarrow_{rm} E + E * \underline{id3}$$

$$\Rightarrow_{rm} E + \underline{id2 * id3}$$

$$\Rightarrow_{rm} \underline{id1} + id2 * id3$$

$$E \Rightarrow_{rm} \underline{E * E}$$

$$\Rightarrow_{rm} E * \underline{id3}$$

$$\Rightarrow_{rm} \underline{E + E * id3}$$

$$\Rightarrow_{rm} E + \underline{id2 * id3}$$

$$\Rightarrow_{rm} \underline{id1} + id2 * id3$$

• Biểu diễn stack của shift-reduce parsing

STACK	INPUT	ACTION
\$	$\text{id}_1 + \text{id}_2 * \text{id}_3$	shift
\$ id_1	\$	reduce by $E \rightarrow \text{id}$
\$ E	$+ \text{id}_2 * \text{id}_3$ \$	shift
\$ $E +$	$+ \text{id}_2 * \text{id}_3$ \$	shift
\$ $E + \text{id}_2$	$\text{id}_2 * \text{id}_3$ \$	reduce by $E \rightarrow \text{id}$
\$ $E + E$	$* \text{id}_3$ \$	shift
\$ $E + E *$	$* \text{id}_3$ \$	shift
\$ $E + E$	id_3 \$	reduce by $E \rightarrow \text{id}$
* id_3	\$	reduce by $E \rightarrow E *$
\$ $E + E *$	\$	E
\$ $E + E$	\$	reduce by $E \rightarrow E +$
\$ E	\$	E

Phân tích cú pháp LR (LR parser)

- LR(k) là một kỹ thuật phân tích cú pháp từ dưới lên hiệu quả, có thể sử dụng để phân tích một lớp rộng các văn phạm phi ngữ cảnh.
 - **L(Left-to-right):** Duyệt chuỗi nhập từ trái sang phải
 - **R(Rightmost derivation):** Xây dựng chuỗi dẫn xuất phải nhất đảo ngược
 - **k:**Số lượng ký hiệu lookahead tại mỗi thời điểm, dùng để đưa ra quyết định phân tích. Khi không đề cập đến k, chúng ta hiểu

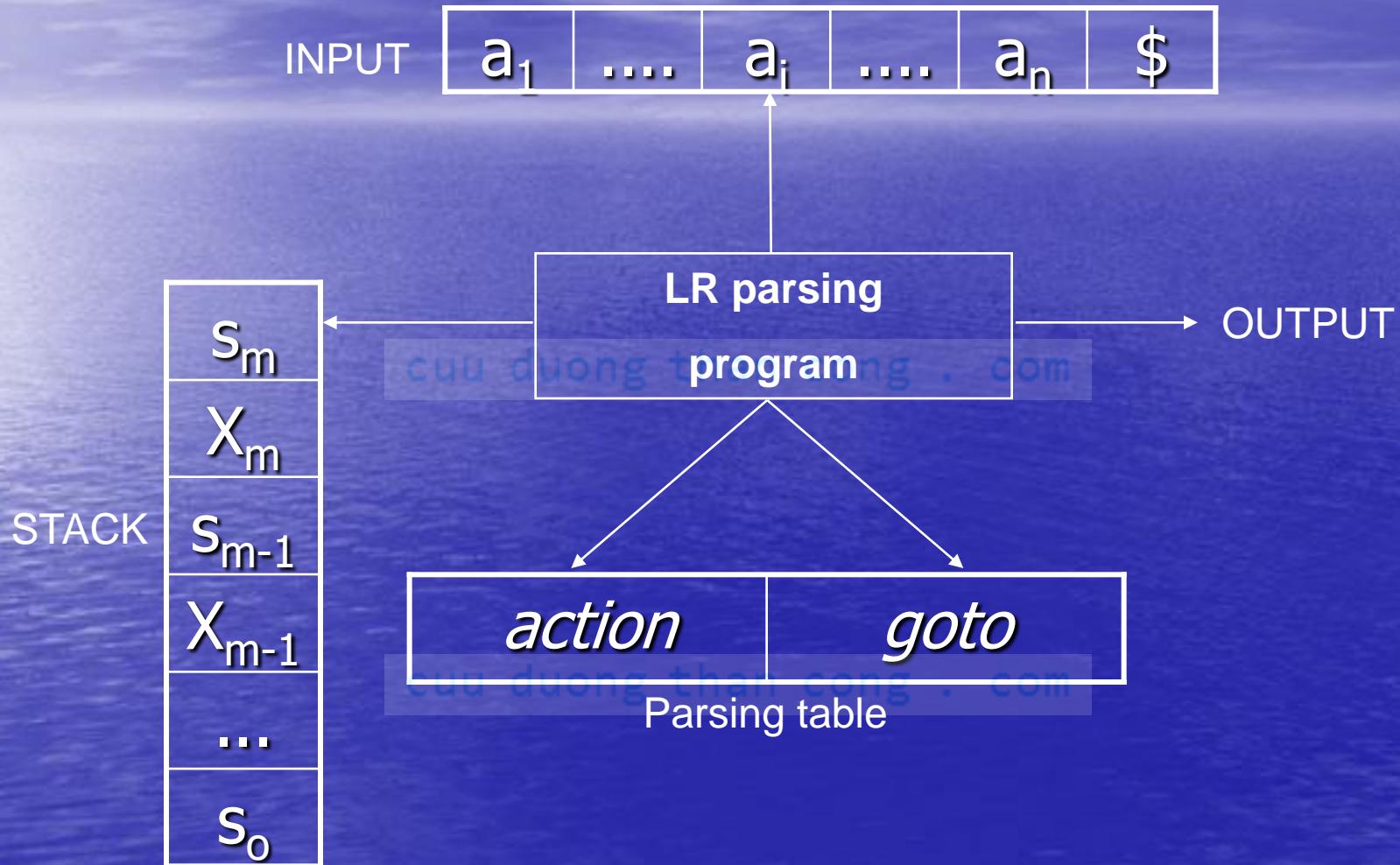
● Ưu điểm của LR:

- Có thể nhận biết hầu như tất cả các ngôn ngữ lập trình được tạo ra bởi văn phạm phi ngữ cảnh
- Phương pháp phân tích cú pháp LR là phương pháp tổng quát của phương pháp shift-reduce không quay lui
- Lớp văn phạm có thể dùng phương pháp LR là một lớp rộng lớn hơn lớp văn phạm có thể sử dụng phương pháp dự đoán
- Có thể xác định lỗi cú pháp nhanh ngay trong khi duyệt dòng nhập từ trái sang phải

● Nhược điểm của LR:

- Xây dựng LR parser khá phức tạp

● Mô hình của LR parser



- STACK lưu chuỗi $s_0 X_1 s_1 X_2 s_2 \dots X_m s_m$ trong đó s_m nằm trên đỉnh STACK một ký hiệu văn phạm, s_i là một trạng thái tóm tắt thông tin chứa trong STACK bên dưới nó
- Parsing table bao gồm 2 phần : Hàm **action** và hàm **goto**
 - $\text{action}[s_m, a_i]$ có thể có một trong 4 giá trị :
 1. **shift s**: Đẩy s, trong đó s là một trạng thái
 2. **reduce**: Thu gọn bằng luật sinh $A \rightarrow \beta$
 3. **accept**: Chấp nhận

- Cấu hình (configuration) của một bộ phân tích cú pháp LR là một cặp thành phần $(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \$)$. Cấu hình biểu diễn right-sentential form $X_1 X_2 \dots X_m a_i a_{i+1} \dots a_n$
- Sự thay đổi cấu hình theo hàm action như sau:
 - Nếu $\text{action}[s_m, a_i] = \text{shift } s$, cấu hình chuyển thành $(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m a_i s, a_{i+1} \dots a_n \$)$, trong đó $s = \text{action}[s_m, a_i]$
 - Nếu $\text{action}[s_m, a_i] = \text{reduce } A \rightarrow \beta$, cấu hình chuyển thành

Ví dụ 3.9: Xét văn phạm cho các phép toán số học + và *

(1) $E \rightarrow E + T$

(2) $E \rightarrow T$

(3) $T \rightarrow T * F$

(4) $T \rightarrow F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow id$

Ý nghĩa :

s_i : shift s_i

r_j : reduce by production j

acc: accept

blank: error

Stat e	Action						Goto		
	id	+	*	()	\$	E	T	F
0	s_5				S_4		1	2	3
1		s_6				acc			
2		r_2	s_7			r_2	r_2		
3		r_4	r_4	.		r_4	r_4		
4	s_5				s_4		8	2	3
5		r_6	r_6			r_6	r_6		
6	s_5				s_4			9	3
7	s_5			s_4					10
8		s_6				s_{11}			
9		r_1	s_7			r_1	r_1		
10		r_3	r_3			r_3	r_3		
11		r_E	r_E			r_E	r_E		

- Với chuỗi nhập $\text{id}^* \text{id} + \text{id}$ quá trình phân tích như sau:

	STACK	INPUT	ACTION
(1)	0	$\text{id}^* \text{id} + \text{id} \$$	shift
(2)	0 id 5	$* \text{id} + \text{id} \$$	reduce by $F \rightarrow \text{id}$
(3)	0 F 3	$* \text{id} + \text{id} \$$	reduce by $T \rightarrow F$
(4)	0 T 2	$* \text{id} + \text{id} \$$	shift
(5)	0 T 2 * 7	$\text{id} + \text{id} \$$	shift
(6)	0 T 2 * 7 id	$+ \text{id} \$$	reduce by $F \rightarrow \text{id}$
(7)	5	$+ \text{id} \$$	reduce by $T \rightarrow T * F$
(8)	0 T 2 * 7 F	$+ \text{id} \$$	reduce by $E \rightarrow T$
(9)	10	$+ \text{id} \$$	shift
(10)	0 T 2	$\text{id} \$$	shift
(11)	0 E 1	\$	reduce by $F \rightarrow \text{id}$
(12)	0 E 1 + 6	\$	reduce by $T \rightarrow F$
(13)	0 E 1 + 6 id 5	\$	reduce by $E \rightarrow E + T$
(14)	0 E 1 + 6 F	\$	accept

Xây dựng SLR parsing table

- Có 3 phương pháp xây dựng một bảng phân tích cú pháp LR từ văn phạm là Simple LR (SLR), Canonical LR và Lookahead- LR (LALR), các phương pháp khác nhau về tính hiệu quả cũng như tính dễ cài đặt
- Phương pháp SLR, là phương pháp yếu nhất nếu tính theo số lượng văn phạm có thể xây dựng thành công, nhưng đây lại là phương pháp dễ cài đặt nhất
- Một văn phạm có thể xây dựng được SLR parser được gọi là một văn phạm SLR

- Một mục LR(0) (hoặc item) của một văn phạm G là một luật sinh của G với một dấu chấm tại vị trí nào đó trong vế phải

Ví dụ 3.10: Luật sinh $A \rightarrow XYZ$ có 4 mục như sau:

$A \rightarrow .XYZ$

$A \rightarrow X.YZ$

$A \rightarrow XY.Z$

$A \rightarrow XYZ.$

- Luật sinh $A \rightarrow \epsilon$ chỉ tạo ra một mục $A \rightarrow .$

- Văn phạm tăng cường (Augmented Grammar): G là một văn phạm với ký hiệu bắt đầu S, thêm một ký hiệu bắt đầu mới S' và luật sinh $S' \rightarrow S$ để được văn phạm mới G' gọi là văn phạm tăng cường
- Phép toán bao đóng (Closure): Giả sử I là một tập các mục của văn phạm G thì bao đóng closure(I) là tập các mục được xây dựng từ I như sau:
 1. Tất cả các mục của I được thêm vào closure(I).
 2. Nếu $A \rightarrow \alpha.B\beta \in \text{closure}(I)$ và $B \rightarrow \gamma$ là một luật sinh thì thêm $B \rightarrow .\gamma$ vào closure(I) nếu nó chưa có trong đó. Lặp lại bước này cho đến khi không thể

Ví dụ 3.11: Xét văn phạm tăng cường

$$E' \rightarrow E$$

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

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

$$F \rightarrow (E) \mid id$$

Nếu $I = \{E' \rightarrow \cdot E\}$ thì closure(I) bao gồm các mục sau:

$$E' \rightarrow \cdot E$$

$$E \rightarrow \cdot E + T$$

$$E \rightarrow \cdot T$$

$$T \rightarrow \cdot T * F$$

$$T \rightarrow \cdot F$$

$$F \rightarrow \cdot (E)$$

- Phép toán goto: Nếu I là một tập các mục và X là một ký hiệu văn phạm thì $\text{goto}(I, X)$ là bao đóng của tập hợp các mục $A \rightarrow \alpha X \beta$ sao cho $A \rightarrow \alpha X \beta \in I$
- Cách tính $\text{goto}(I, X)$:
 1. Tạo một tập $I' = \emptyset$
 2. Nếu $A \rightarrow \alpha X \beta \in I$ thì đưa $A \rightarrow \alpha X \beta$ vào I' , tiếp tục quá trình này cho đến khi xét hết tập I.
 3. $\text{goto}(I, X) = \text{closure}(I')$

Ví dụ 3.12: Giả sử $I = \{E' \rightarrow E., E \rightarrow E. + T\}$

Ta có $I' = \{ E \rightarrow E. + . T\}$

goto ($I, +$) = closure(I') bao gồm các mục :

$E \rightarrow E. + . T$

$T \rightarrow . T$ cuuduongthancong.com

$T \rightarrow . F$

$F \rightarrow . (E)$

$F \rightarrow . id$

cuuduongthancong.com

- Giải thuật xây dựng họ tập hợp các mục LR(0) (kí hiệu là C) của văn phạm G'
procedure Item (G')

begin

C := {closure({ S' → .S}) };

repeat

hiệu
X) \neq Φ và

For Với mỗi tập các mục I ∈ C và mỗi ký
văn phạm X sao cho goto (I,
X)

goto(I, X) ∈ C thì thêm goto(I, X) vào C;

until Không còn tập hợp mục nào có thể thêm
vào C;

Ví dụ 3.13: Xây dựng họ tập hợp các mục trong ví dụ 3.11

closure($\{E' \rightarrow E\}$)	$E' \rightarrow \cdot E$	goto ($I_0,$	$F \rightarrow id \cdot$
$I_0:$	$E \rightarrow \cdot E + T$	id)	$E \rightarrow E + \cdot T$
	$E \rightarrow \cdot T$	goto ($I_1,$	$T \rightarrow \cdot T * F$
	$T \rightarrow \cdot T * F$	+)	$T \rightarrow \cdot F$
	$T \rightarrow \cdot F$		$F \rightarrow \cdot (E)$
	$F \rightarrow \cdot (E)$		$F \rightarrow \cdot id$
	$F \rightarrow \cdot id$		$T \rightarrow T^* \cdot F$
	$E' \rightarrow E \cdot$		$F \rightarrow \cdot (E)$
goto ($I_0,$	$E \rightarrow E \cdot + T$	goto ($I_2,$	$F \rightarrow \cdot id$
$E)$	$E \rightarrow T \cdot$	$*)$	$F \rightarrow (E \cdot)$
	$T \rightarrow T \cdot * F$		$E \rightarrow E \cdot + T$
goto ($I_0,$	$T \rightarrow F \cdot$		$E \rightarrow E + T \cdot$
$T)$	$I_1:$	$F \rightarrow (\cdot E)$	$T \rightarrow T \cdot * F$
		goto ($I_4,$	$T \rightarrow T * F \cdot$
goto	$E \rightarrow \cdot E + T$	$E)$	$F \rightarrow (E) \cdot$
($I_0, F)$	$E \rightarrow \cdot T$	$I_8:$	
	$T \rightarrow \cdot T * F$	goto	
goto ($I_0, ($	$T \rightarrow \cdot F$	($I_6, T)$	
)	$F \rightarrow \cdot (E)$		$I_9:$
	$E \rightarrow \cdot id$	goto	
		($I_5, E)$	

● Xây dựng SLR parsing table

1. Xây dựng họ tập hợp các mục của G' : $C = \{ I_0, I_1, \dots, I_n \}$
2. Trạng thái i được xây dựng từ I_i . Các action tương ứng trạng thái i xác định như sau:
 - a) Nếu $A \rightarrow \alpha.a\beta \in I_i$ và $\text{goto}(I_i, a) = I_j$ thì $\text{action}[i, a] = \text{"shift } j\text{"}$,
a là ký hiệu kết thúc
 - b) Nếu $A \rightarrow \alpha. \in I_i$ thì $\text{action}[i, a] = \text{"reduce } (A \rightarrow \alpha)"$, với mọi $a \in \text{FOLLOW}(A)$, $A \neq S'$
 - c) Nếu $S' \rightarrow S \cdot \in I_i$ thì $\text{action}[i, \$] = \text{"accept"}.$

Nếu một action đụng độ được sinh ra bởi các luật trên, ta nói văn phạm không phải là SLR(1). Giải thuật thất bại

Xây dựng bảng phân tích LR chính tắc

- LR chính tắc (canonical LR) là kĩ thuật chung nhất để xây dựng LR parsing table cho một văn phạm
- Một mục LR(1) (item) là một cặp $[A \rightarrow \alpha.\beta, a]$ trong đó $A \rightarrow \alpha\beta$ là một luật sinh, a - là kí tự lookahead là một kí hiệu kết thúc hoặc $\$$
- Nếu $\beta \neq \varepsilon$ thì a không có ý nghĩa nhưng nếu $\beta = \varepsilon$ thì việc reduce theo luật $A \rightarrow \alpha$ chỉ được thực hiện nếu kí tự đọc vào tiếp theo là a

- Phép toán bao đóng (Closure): Giả sử I là một tập các mục LR(1) của văn phạm G thì bao đóng closure(I) là tập các mục được xây dựng từ I như sau:
 1. Tất cả các mục của I được thêm vào closure(I).
 2. Nếu $[A \rightarrow \alpha.B\beta, a] \in \text{closure}(I)$, $B \rightarrow \gamma$ là một luật sinh và $b \in \text{FIRST}(\beta a)$ thì thêm $[B \rightarrow .\gamma, b]$ vào closure(I) nếu nó chưa có trong đó. Lặp lại bước này cho đến khi không thể thêm vào closure(I) được nữa
- Phép toán goto: Nếu I là một tập các mục và X là một ký hiệu văn phạm thì goto(I, X) là bao đóng của tập hợp các mục $[A \rightarrow \dots, X \rightarrow \dots]$ sao cho $[A \rightarrow \dots, X \rightarrow \dots] \in I$

- Giải thuật xây dựng họ tập hợp các mục LR(1) (kí hiệu là C) của văn phạm G'
procedure Item (G')
begin

C := {closure({[S' → .S, \$]})};

cuu duong than cong . com

repeat

hiệu
X) \neq Φ và

For Với mỗi tập các mục I \in C và mỗi ký
văn phạm X sao cho goto (I,
X)

cuu duong than cong . com

goto(I, X) \notin C thì thêm goto(I, X) vào C;

until Không còn tập hợp mục nào có thể thêm
vào C;

Ví dụ 3.14: Xây dựng họ tập hợp các mục LR(1) cho văn phạm dưới đây

$$S' \rightarrow S$$

(1) $S \rightarrow CC$

(2) $C \rightarrow cC$

(3) $C \rightarrow d$

$\text{closure}(\{S' \rightarrow S\})$

$I_0:$

goto ($I_0,$

$S)$

$I_1:$

goto ($I_0,$

$C)$

$I_2:$

$$S' \rightarrow \cdot S, \$$$

goto (I_0, d

$$C \rightarrow d \cdot, c/d$$

$$S \rightarrow \cdot CC, \$$$

) $I_4:$

$$S \rightarrow CC \cdot, \$$$

$$C \rightarrow \cdot cC, c/d$$

$$C \rightarrow \cdot d, c/d$$

goto ($I_2,$

$$S' \rightarrow S \cdot, \$$$

$C)$ $I_5:$

$$C \rightarrow c \cdot C, \$$$

$$S \rightarrow C \cdot C, \$$$

goto ($I_2,$

$$C \rightarrow \cdot d, \$$$

) $I_6:$

$$C \rightarrow d \cdot, \$$$

$$C \rightarrow c \cdot C, c/d$$

$$C \rightarrow \cdot cC, c/d$$

goto ($I_2,$

$$C \rightarrow \cdot d, c/d$$

$d)$

$I_7:$

$$C \rightarrow cC \cdot, c/d$$

$$C \rightarrow cC \cdot, \$$$

● Xây dựng canonical LR parsing table

1. Xây dựng họ tập hợp các mục LR(1) của G' : $C = \{I_0, I_1, \dots, I_n\}$
2. Trạng thái i được xây dựng từ I_i . Các action tương ứng trạng thái i xác định như sau:
 - a) Nếu $[A \rightarrow \alpha.a\beta, b] \in I_i$ và $\text{goto}(I_i, a) = I_j$ thì $\text{action}[i, a] = \text{"shift } j\text{"}$, a là ký hiệu kết thúc
 - b) Nếu $[A \rightarrow \alpha., a] \in I_i$ thì $\text{action}[i, a] = \text{"reduce } (A \rightarrow \alpha.)"$, $A \neq S'$
 - c) Nếu $[S' \rightarrow S \cdot, \$] \in I_i$ thì $\text{action}[i, \$] = \text{"accept"}.$
Nếu một action đụng độ được sinh ra bởi các luật trên, ta nói văn phạm không phải là LR(1). Giải thuật thất bại
3. Nếu $\text{goto}(I_i, A) = I_j$, thì $\text{goto}[i, A] = j$, A là kí hiệu

- Canonical LR parsing table cho văn phạm trong ví dụ 3.14

State	Action			Goto	
	c	d	\$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7	tan cong . com		5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Xây dựng bảng phân tích LALR

- LALR là phương pháp canonical parsing trong đó các trạng thái được nhóm lại với nhau nhờ đó bảng phân tích cấu trúc có kích thước nhỏ hơn (có thể so sánh với SLR)
- Hạt nhân (core) của một tập hợp mục LR(1) có dạng $\{[A \rightarrow \alpha.\beta, a]\}$, trong đó $A \rightarrow \alpha\beta$ là một luật sinh và a là ký hiệu kết thúc có hạt nhân (core) là tập hợp $\{A \rightarrow \alpha.\beta\}$.
- Trong họ tập hợp các mục LR(1) $C = \{I_0, I_1, \dots, I_n\}$ có thể có các tập hợp các mục có chung một hạt nhân.

● Xây dựng LALR parsing table

1. Xây dựng họ tập hợp các mục LR(1) của G' : $C = \{I_0, I_1, \dots, I_n\}$
2. Nhóm các mục có cùng core trong C được $C' = \{J_0, J_1, \dots, J_m\}$
3. Trạng thái i được xây dựng từ J_i . Các action tương ứng trạng thái i xác định tương tự như canonical LR
Nếu một action đụng độ được sinh ra bởi các luật trên, ta nói văn phạm không phải là LALR(1). Giải thuật thất bại
3. Xây dựng bảng goto : Giả sử $J = I_1 \cup I_2 \cup \dots \cup I_k$. Vì I_1, I_2, \dots, I_k có chung hạt nhân nên $\text{goto}(I_1, X)$, $\text{goto}(I_2, X), \dots, \text{goto}(I_k, X)$ cũng có chung hạt nhân.

- LALR parsing table cho văn phạm trong ví dụ 3.14

State	Action			Goto	
	c	d	\$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Công cụ phân tích cú pháp Yacc

- Giống như Lex, Yacc (yet another compiler compiler) là câu lệnh sẵn có của UNIX và là một công cụ hữu hiệu cho phép xây dựng bộ phân tích cú pháp một cách tự động
- Yacc được tạo bởi S. C. Johnson vào những năm đầu của thập kỉ 70
- Yacc sử dụng phương pháp LALR

