CHƯƠNG 4

PHÂN TÍCH CÚ PHÁP

4.1. Vai trò của bộ phân tích cú pháp

- Phương pháp tổng quát: Cocke-Younger-Kasami và Earley.
- Phân tích từ trên xuống.
- Phân tích từ dưới lên.

4.2. Xây dựng văn phạm cho ngôn ngữ lập trình

Loại bỏ sự không tường minh

stmt → **if** exp **then** stmt

if exp **then** stmt **else** stmt

| other

Thí dụ: phát biểu: **if** E_1 **then if** E_2 **then** S_1 **else** S_2 là phát biểu không tường minh

- Loại bỏ sự không tường minh.

Quy ước hoặc sửa văn phạm.

stmt → matched-stmt

lunmatched-stmt

matched-stmt→ **if** exp **then** matched-stmt **else** matched-stmt1

| other

unmatched-stmt → if exp then stmt

| **if** exp **then** matched-stmt **else** unmatched-stmt

Loại bỏ đệ quay trái

Văn phạm gọi là đệ quy trái nếu tồn tại dẫn xuất.

$A \Rightarrow A\alpha$, với $\alpha \subset (V_t \cup V_n)$

Đệ quy trái là bao gồm đệ quy trái đơn giản (trực tiếp) và đệ quy trái tổng quát.

Để loại bỏ đệ quy đơn giản, ta sẽ thay thế tập luật sinh:

$A \rightarrow A\alpha_1 | A\alpha_2 | \ldots\ldots | A\alpha_m | \beta_1 | \beta_2 | \ldots\ldots | \beta_n$

bằng cặp luật sinh

$A \rightarrow \beta_1 A' | \beta_2 A' | \ldots | \beta_n A.'$

$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \ldots\ldots | \alpha_m A' | \in$

Thí dụ 4.1. Loại bỏ đệ quy trái cho văn phạm:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Giải thuật 4.1. Loại bỏ đệ qy trái

Nhập: Văn phạm G không có vòng lặp hội luật sinh rỗng.

Xuất : Văn phạm tương đương G' không có đệ quy trái.

Phương pháp: Áp dụng giải thuật ở mô phỏng 4.1 cho G. G' không còn đệ quy trái nhưng có thể có luật sinh rỗng.

Sắp xếp caucus ký hiệu không kết thúc theo một thứ tự nào đó: A_1, A_2, ... A_n .

Mô phỏng 4.1. *Giải thuật loại bỏ đệ quy trái từ văn phạm*

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

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

- Thay các luật sinh có dạng $A_i \rightarrow A_j \gamma$ bằng các luật sinh $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid ..... \mid \delta_k\gamma$
- Với A_j luật sinh có dạng $A_i \rightarrow \delta_1 \mid \delta_2 \mid .... \mid \delta_k$
- Loại tất cả cả các luật sinh có đệ quy trái trực tiếp trong các A_i luật sinh

end;

Thí dụ: Chúng ta có áp dụng giải thuật 4.1 vào văn phạm sau để loại bỏ đệ quy trái.

$S \rightarrow Aa \mid b$ $\qquad\qquad$ $A \rightarrow Ac \mid Sd \mid \in$

Thừa số trái: Thí dụ ta có hai luật sinh:

stmt → **if** exp **then** stmt **else** stmt

| **if** exp **then** stmt

Cả hai luật sinh đều có **if** dẫn đầu nên ta sẽ không biết chọn luật sinh nào để triển khai. Vì thế để làm chậm lại quyết định lựa chọn chúng ta sẽ tạo ra thừa số trái.

***Giải thuật 4.2.* Tạo văn phạm có thừa số trái**

Nhập: cho văn phạm G.

Xuất: văn phạm G' có thừa số trái tương đương.

Phương pháp: Tìm chuỗi dẫn đầu chung của các vế phải luật sinh, thí dụ: $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid ..... \mid \alpha\beta_n \mid \gamma$. γ là chuỗi không bắt đầu bởi α. Ta thay các luật trên bằng các luật $A \rightarrow \alpha A'$ $\quad$ $A' \rightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n$

Thí dụ: Ta áp dụng giải thuật trên cho văn phạm phát biểu if, nước văn phạm tương đương

$S \rightarrow i\,E\,t\,SS' \mid a$ $\quad$ $S' \rightarrow e\,S \mid \in$ $\quad$ $E \rightarrow b$

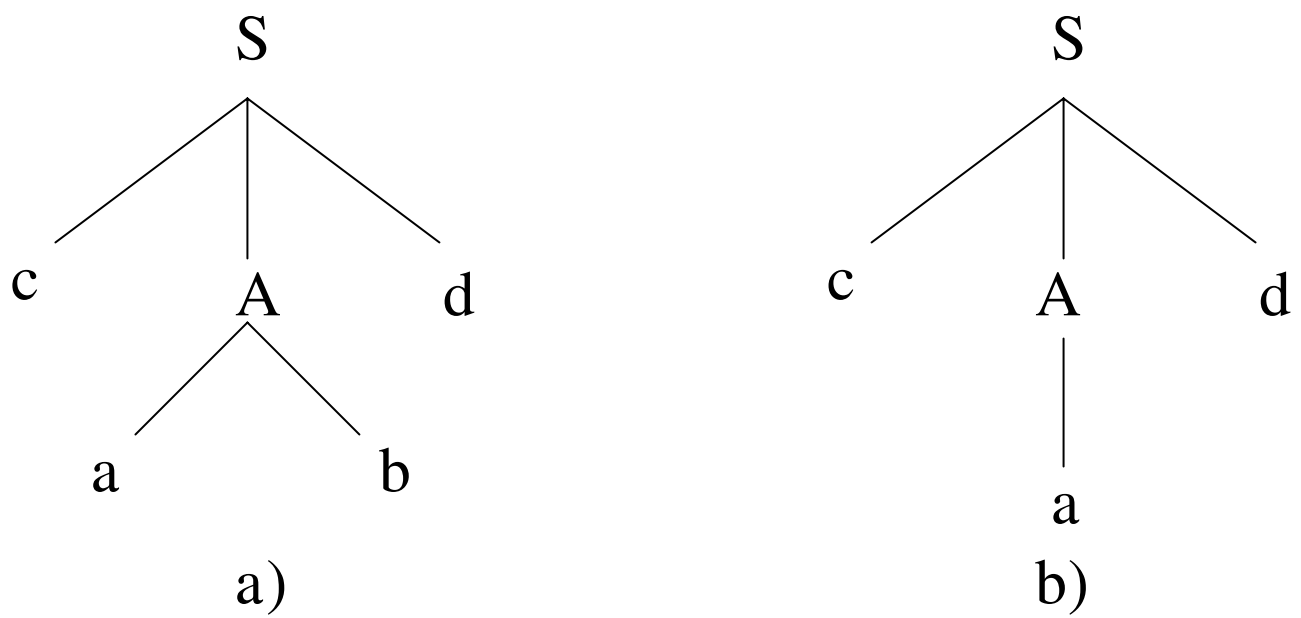
4.3. Phân tích cú pháp từ trên xuống

Phân tích cú pháp đệ quy đi xuống.

Phân tích cú pháp đoán nhận trước.

1. Phân tích cú pháp đệ quy đi xuống

Thí dụ: Cho văn phạm G : $S \rightarrow cAd \quad A \rightarrow ab \mid a$



Hình 4.4. Các bước phân tích cú pháp từ trên xuống

2. *Phân tích cú pháp đoán nhận trước*

- Hãy loại bỏ đệ quy trái cho văn phạm mà chúng ta thiết kế.
- Hãy tạo văn phạm có thừa số trái nếu cần thiết.

Sơ đồ dịch cho bộ phân tích đoán nhận trước

Sơ đồ này có đặc điểm như sau:

- Mỗi ký hiệu không kết thúc có một sơ đồ.
- Tên các cạnh là token và các ký hiệu không kết thúc.

Sự truyền trên token sẽ được thực hiện nếu ký hiệu nhập trùng với token đó. Nếu có sự truyền trên ký hiệu không kết thúc A thì ta thực hiện một lệnh gọi thủ tục A.

Để xây dựng sơ đồ chúng ta sẽ tiến hành các bước sau đây:

1. Tạo trạng thái bắt đầu và kết thúc.
2. Với mỗi luật sinh có dạng $A \rightarrow X_1X_2...X_n$, ta xây dựng đường đi từ trạng thái bắt đầu đến trạng thái kết thúc sao cho các cạnh có tên X_1, X_2, $X_3...X_n$.

Cơ chế hoạt động của bộ phân tích đoán nhận trước

Thí dụ 4.3. Chúng ta hãy tạo sơ đồ dịch cho văn phạm

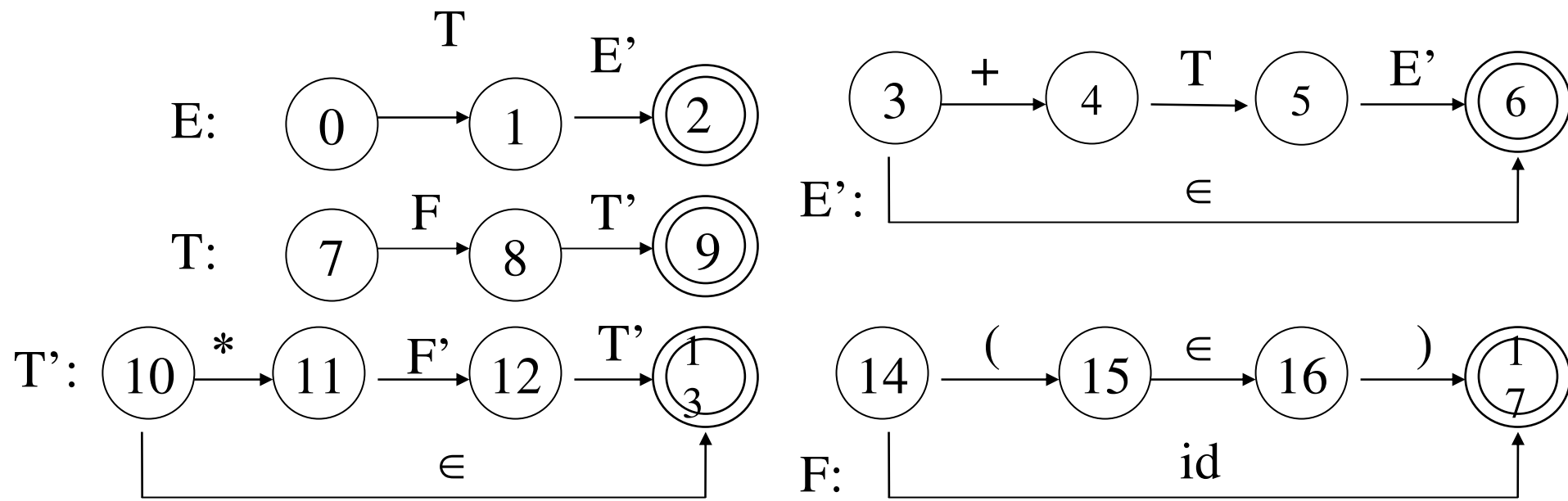
G: E → TE’

E’ → + TE’ |∈

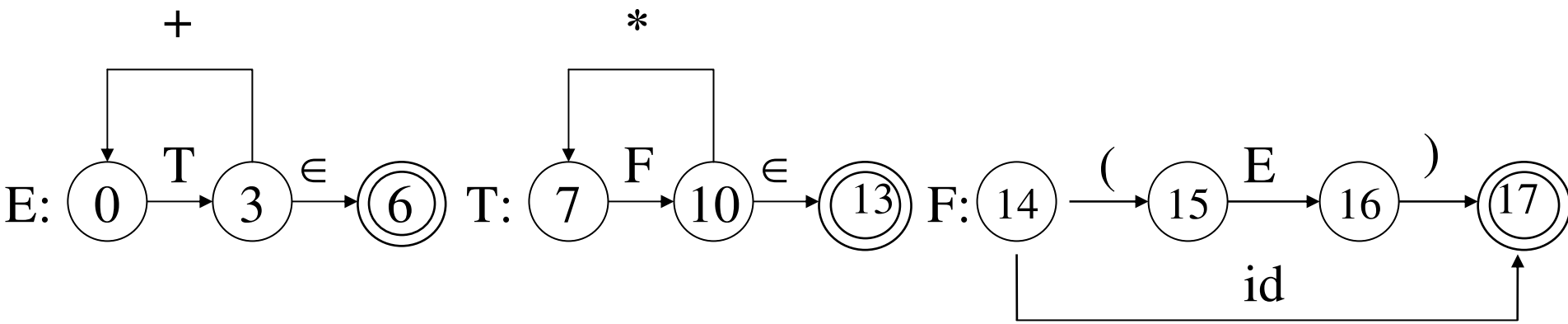
T → FT’

T’ → * FT’ |∈

F’ → (E) | id



Hình 4.5. Sơ đồ dịch của các ký hiệu không kết thúc của G



Hình 4.6. Sơ đồ dịch của các ký hiệu không kết thúc của G, đã được thu giảm

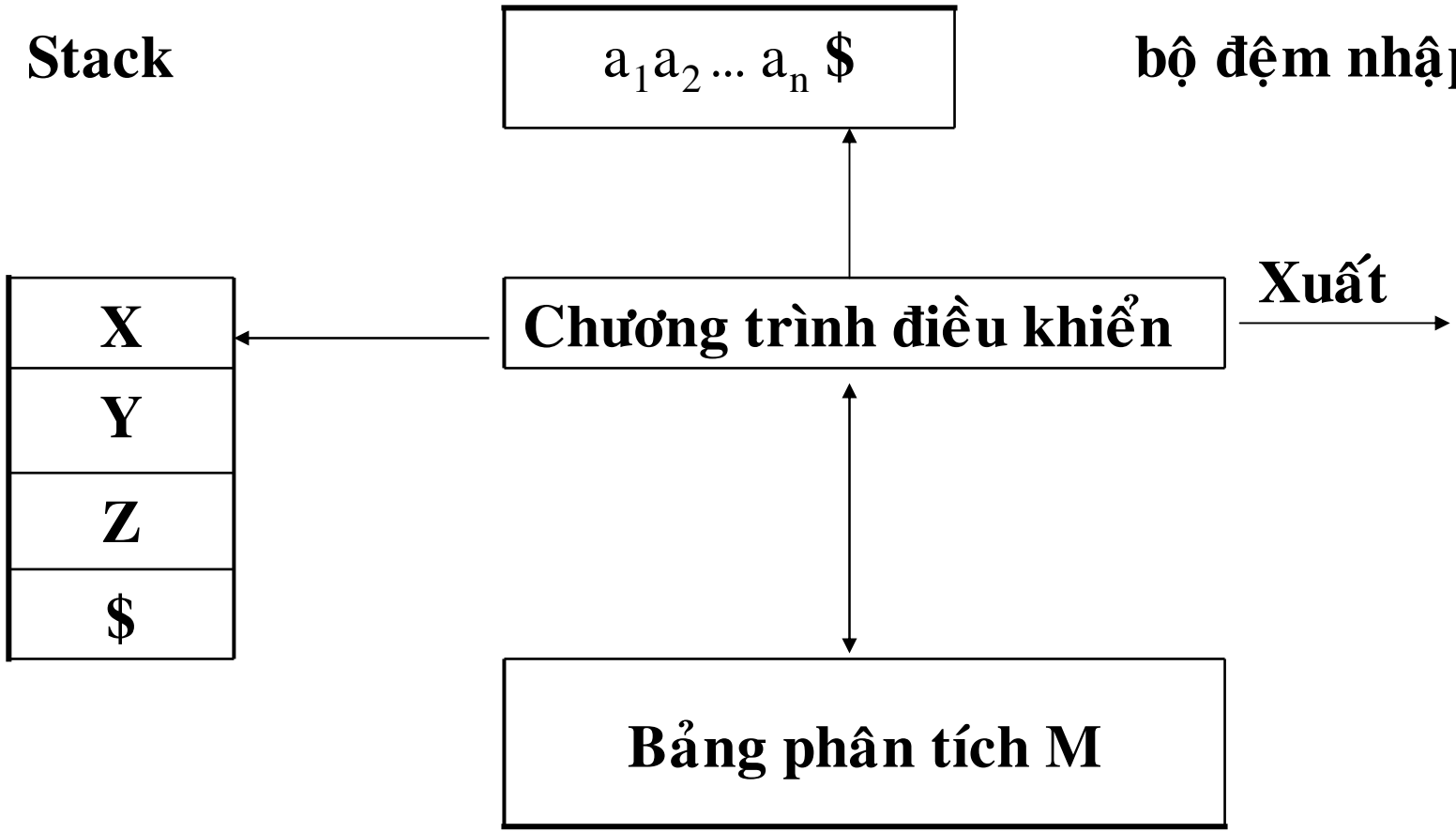
Giải thuật:

```
procedure E;
procedure T;
    procedure F;
        begin nextchar (c);
        if c = '(' then begin
                match ('('); E;
                match (')'); end
        else if c = id then match (id)
                       else error;
```

```
end; {F}
begin
        F;
        while c = '*' do F;
end; {T}
begin
        T;
        while c = '+' do T;
end; {E}
```

3. Phân tích cú pháp đoán nhận trước không đệ quy

Cấu tạo của bộ phân tích cú pháp



Hình 4.7. Mô hình cấu tạo của bộ phân tích đoán nhận trước không đệ quy.

Hoạt động của bộ phân tích

Ở trạng thái bắt đầu, stack chỉ chứa các ký hiệu mục tiêu của văn phạm nằm trên $, trên đỉnh stack. Bảng phân tích M là ma trận. Hai ký hiệu X và a sẽ xác định hành vi của bộ phân tích. Bộ phân tích có ba hành vi như sau:

1. Nếu X = a = $.
2. Nếu X = a ≠ $.
3. Nếu X là ký hiệu không kết thúc.

Giải thuật 4.2. **Phân tích cú pháp đoán nhận trước không đệ quy.**

Nhập: chuỗi nhập w và bảng phân tích M cho văn phạm G.

Xuất: nếu w thuộc L (G), sẽ tạo ra dẫn xuất trái của w, ngược lại sẽ báo lỗi.

Phương pháp: lúc đầu cấu hình của bộ phân tích là (S, w) với S là ký hiệu mục tiêu của G. Đặt ip (là con trỏ hoặc còn gọi là đầu đọc của bộ phân tích) vào ký hiệu nhập đầu tiên của w$.

Mô phỏng 4.2. *Chương trình phân tích cú pháp đoán nhận trước*

repeat

X trên stack và ký hiệu a đang được đầu đọc ip đọc;

if X là *ký hiệu kết thúc* hoặc *$* **then**

if X = a **then begin**

- đẩy X ra khỏi stack;
- dịch đầu đọc đến ký hiệu nhập kế tiếp; **end**

else error ()

else if $M[X, a] = X \rightarrow X_1X_2...X_k$ **then begin**

- đẩy X ra khỏi stack;
- đẩy $X_kX_{k-1}... X_1$ lên stack (X_1 trên đỉnh stack);
- xuất luật sinh $X \rightarrow X_1X_2 ... X_k$ **; end**

else error ()

until X = $

Thí dụ 4.4. Giả sử chúng ta có văn phạm G.

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Chúng ta sẽ thực hiện loại bỏ đệ quy trái, nhận được G':

E → TE'
E' → + TE' |∈
T → FT'
T → * FT' |∈
F → (E) | id

Bây giờ chúng ta sẽ phân tích cú pháp cho câu nhập w = id + id * id bằng bảng phân tích M cho trước, ở Bảng 4.1.

Bảng 4.1. *Bảng phân tích M cho văn phạm G*

Ký hiệu không kết thúc	***Ký hiệu nhập***					
	id	**+**	*****	**(**	**)**	**$**
E	E → TE'			E → TE'		
E'		E → +TE'			E' → ∈	E' → ∈
T	T → FT'			T → FT'		
T'		T' → ∈	T → * FT'		T' → ∈	T' → ∈
F	F → id			F → (E)		

Quá trình phân tích cú pháp câu nhập w = id + id * id sẽ được trình bày ở bảng 4.2.

Bảng 4.2. *Các bước phân tích cú pháp câu id + id * id*

Stack	*Chuỗi nhập*	*Xuất*	*Stack*	*Chuỗi nhập*	*Xuất*
$E	id + id * id $		$E'T'F	id * id $	T → FT'
$E'T	id + id * id $	E → TE'	$E'T'id	id * id $	F → id
$E'T'F	id + id * id $	T → FT'	$E'T'	* id $	
$E'T'id	id + id * id $	F → id	$E'T'F*	* id $	T' → *FT'
$E'T'	+ id * id $		$E'T'F	id $	
$E'	+ id * id $	T' → ∈	$E'T'id	id $	F → id
$E'T+	+ id * id $	E' → +TE'	$E'T'	$	
$E'T	id * id $		$E'	$	T' → ∈
			$	$	E' → ∈

Xây dựng bảng phân tích M

a. **first** và **follow**

first(α) là tập c ký hiệu kết thúc a, dẫn đầu các chuỗi được dẫn xuất từ α, $\alpha \Rightarrow a\gamma$. Nếu $\alpha \Rightarrow \in$ thì $\in$ thuộc first(α).

follow(A) là tập các ký hiệu kết thúc a, xuất hiện ngay bên phải A trong dạng câu. Như vậy tồn tại dẫn xuất $S \Rightarrow \alpha Aa\beta$. Nếu giữa A và a tồn tại chuỗi ký hiệu, thì nó sẽ dẫn xuất ra chuỗi rỗng. Nếu A ở tận cùng bên phải của dạng câu thì \$ thuộc **follow**(A).

- Các quy tắc tính **first**(X) với X là ký hiệu văn phạm.
- Các quy tắc tính **follow**(A) cho tất cả các ký hiệu không kết thúc A.

Thí dụ 4.5. Cho văn phạm G.

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

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

Toàn bộ các hàm **first** và **follow** của các ký hiệu văn phạm của G :

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

first(E') = {+, ∈} ; first(T') = {*, ∈}

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

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

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

b. Xây dựng bảng phân tích M

***Giải thuật 4.3*. Xây dựng bảng phân tích M.**

Nhập: văn phạm G.

Xuất: bảng phân tích M.

Phương pháp:

1. Với mỗi luật sinh A → α hãy thực thi bước 2 và 3.
2. Với mỗi ký hiệu kết thúc a thuộc first(α), thêm A → α vào M[A, a].
3. Nếu ký hiệu ∈ thuộc first(α), thêm A → α vào M[A, b] sao cho b thuộc follow(A). Nếu $ thuộc follow(A) thì thêm A → α vào M [A, $].
4. Những phần tử của bảng M trống, hãy đánh dấu lỗi.

Văn phạm LL (1)

Thí dụ 4.7. Cho văn phạm G.

$$S \rightarrow iEtSS' \mid a \; ; \; S' \rightarrow eS \mid \in \; ; \; E \rightarrow b$$

Bảng 4.3. *Bảng phân tích M cho thí dụ 4.7.*

Các ký hiệu không KT	***Ký hiệu nhập***					
	a	**b**	**e**	**i**	**t**	**$**
S	$S \rightarrow a$			$S \rightarrow iEtSS'$		
S'			$S' \rightarrow \in$ $S' \rightarrow eS$			$S' \rightarrow \in$
E		$E \rightarrow b$				

- Văn phạm không có phần tử nào của bảng phân tích M có nhiều hơn một trị thì được gọi là văn phạm LL (1).
- Văn phạm LL (1) có các tính chất sau.

Khắc phục lỗi trong phân tích cú pháp đoán nhận trước

Lỗi xuất hiện trong các trường hợp sau: Một là ký hiệu kết thúc trên stack không trùng với ký hiệu nhập đang được đọc. Hai là A là ký hiệu không kết thúc trên đỉnh stack, a trên chuỗi nhập, được đọc, mà M [A, a] là trống.

Một số heuristics được áp dụng cho việc khắc phục lỗi.

Thí dụ 4.8. Cho văn phạm

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

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

first(E') = {+, }∈; first (T') = {*, }∈

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

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

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

Bảng 4.4. *Phân tích M có ký hiệu khắc phục lỗi.*

Ký hiệu không KT	***Ký hiệu nhập***					
	id	**+**	*****	**(**	**)**	**$**
E	E → TE'			E → TE'	***synch***	***synch***
E'		E' → +TE'			E' → ∈	E' → ∈
T	T → FT'	***synch***		T → FT'	***synch***	***synch***
T'		T' → ∈	T' → * FT'		T' → ∈	T' → ∈
F	F → id	***synch***	***synch***	F → (E)	***synch***	***synch***

4.4. Phân tích cú pháp từ dưới lên

Phân tích cú pháp từ dưới lên được hiểu là phân tích đẩy và thu giảm (Shift-Reduce parsing) là phương pháp phân tích LR.

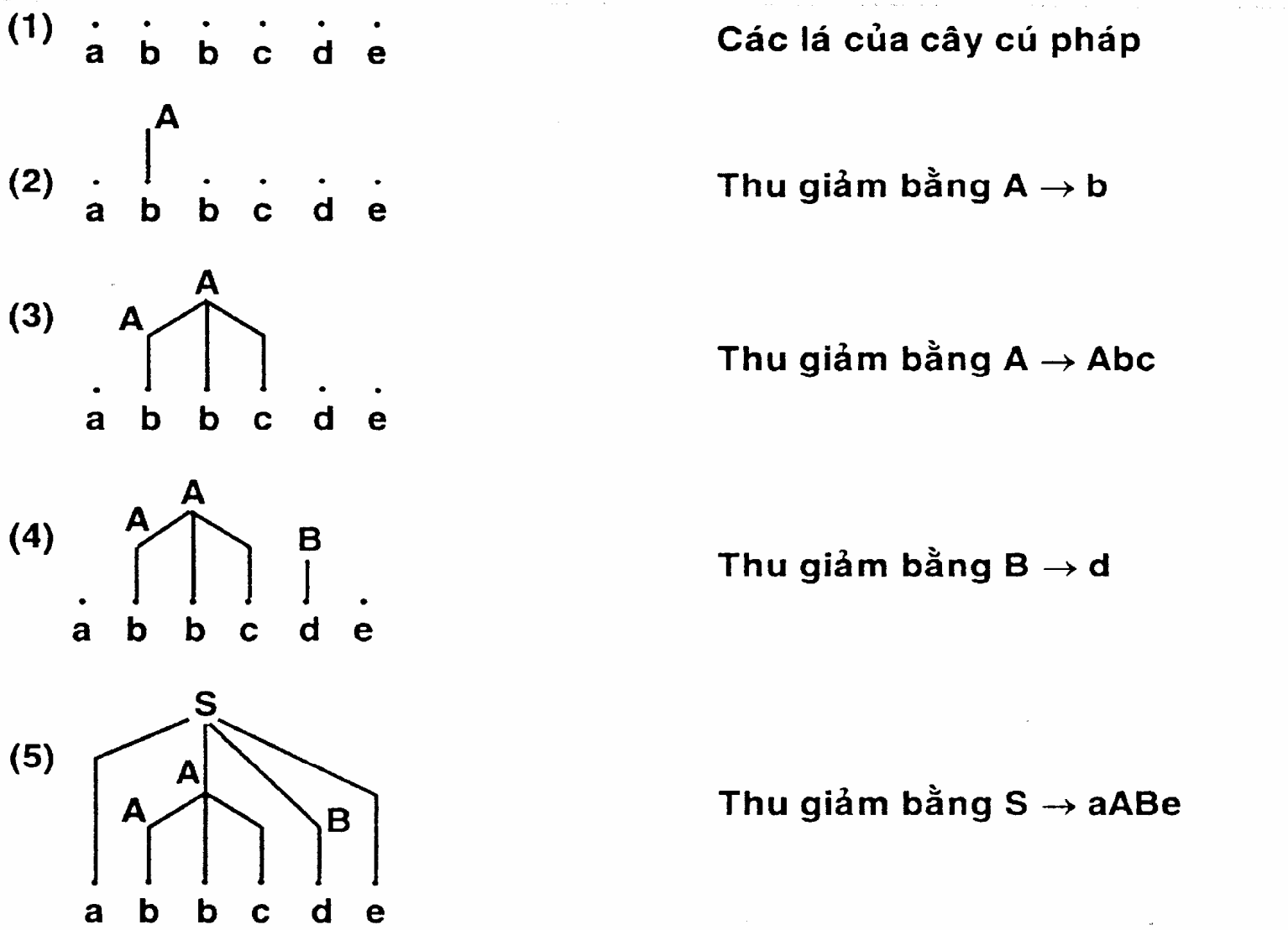
Thí dụ 4.9. Cho văn phạm G.

$$S \rightarrow aABe \; ; \; A \rightarrow Abc \mid b \; ; \; B \rightarrow d$$

Phân tích câu w = abbcde.

Tóm tắt các bước thu giảm như sau:

Quá trình thu giảm nếu theo chiều ngược lại thì đó chính là quá trình dẫn xuất phải. Quá trình này đã sinh cây cú pháp của câu phân tích từ dưới lên.



Hình 4.8. Cây cú pháp được xây dựng từ dưới lên của câu w = abbcde.

Handle

Tìm kiếm handle

Bắt đầu từ chuỗi cần phân tích w, ta đặt w = γn . γn là dạng câu được dẫn xuất ở lần thứ n.

$$S = \gamma_0 \overset{1}{\underset{r_m}{\Rightarrow}} \gamma_1 \overset{2}{\underset{r_m}{\Rightarrow}} \gamma_2 \underset{r_m}{\Rightarrow} \ldots \overset{r-1}{\underset{r_m}{\Rightarrow}} \gamma_{n-1} \overset{n}{\underset{r_m}{\Rightarrow}} \gamma_n = w$$

Xây dựng dẫn xuất phải ngược từ $w = \gamma_n$. Ta tìm βn trong γn sao cho β_n là vế phải luật sinh $A_n \rightarrow \beta_n$. Thay β_n trong γ_n bằng A_n , ta nhận được dạng câu thứ (n – 1) là γ_{n-1}.

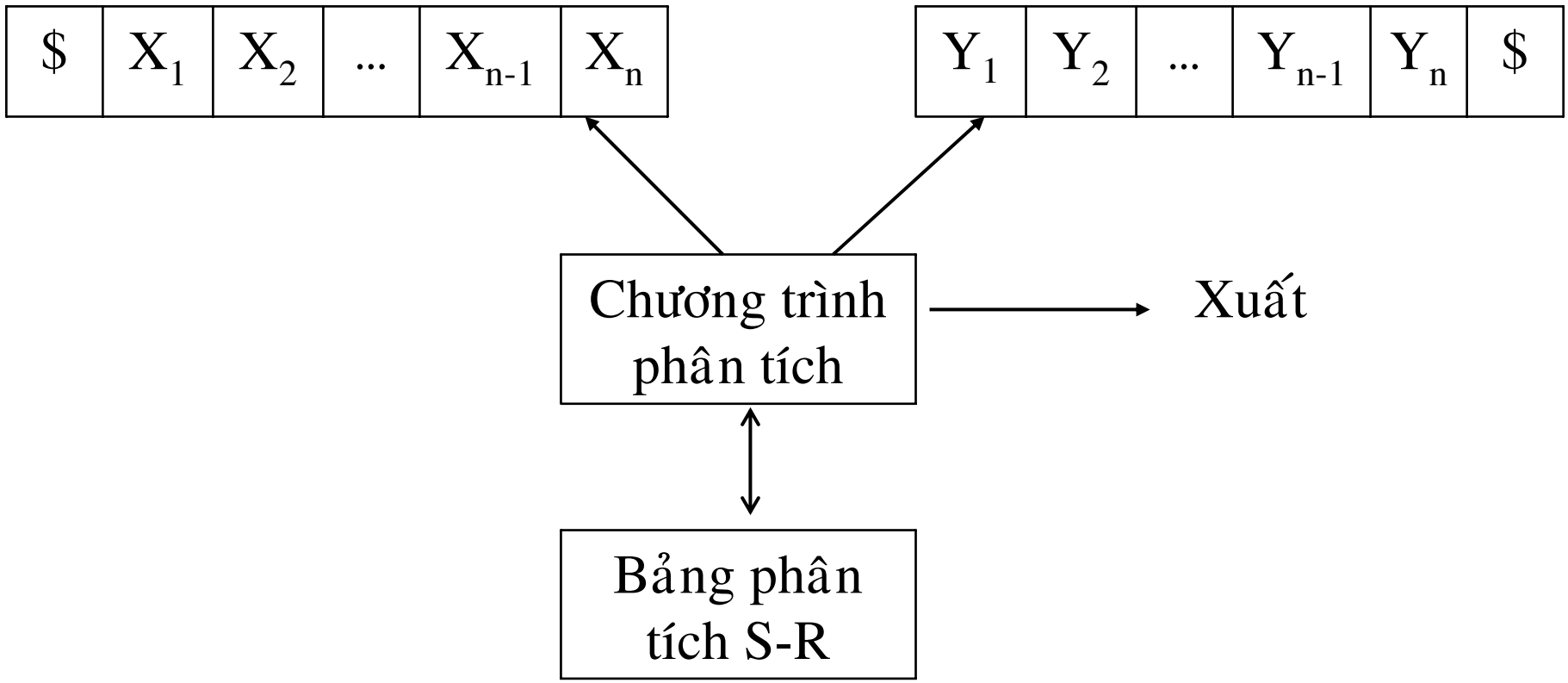
Quá trình thu giảm cứ tiếp tục như vậy cho đến khi đạt được γ_o chỉ còn là một ký hiệu không kết thúc và là ký hiệu mục tiêu.

1. Phân tích cú pháp thứ tự yếu

Văn phạm có tính chất: không có luật sinh nào có vế phải là chuỗi rỗng (A → ∈) hoặc ở vế phải không có hai ký hiệu không kết thúc đứng kề nhau gọi là văn phạm thứ tự yếu.

Bộ phân tích cú pháp thứ tự yếu

1. Cấu tạo



Hình 4.9. Mô hình bộ phân tích cú pháp thứ tự yếu

2. Hoạt động

Thí dụ 4.10. Cho văn phạm của phát biểu gán

< assign stmt > → id = < exp >

< exp > → < exp > + < term > | <term>

< term > → < term > * < factor > | < factor >

< factor > → id | (< exp >)

Ký hiệu <assign stmt> là ký hiệu mục tiêu.

Bảng 4.6. Bảng phân tích S-R cho văn phạm ở thí dụ 4.10.

	id	*	+	(	)	=	$
<assign stmt>							R*
<exp>			S		S		R
<term>		S	R		R		R
<factor>		R	R		R		R
id		R	R		R	S	R
*	S			S			
+	S			S			
(	S			S			
)		R	R		R		R
=	S			S			
$	S						

Giải thuật 4.4. **Phân tích cú pháp thứ tự yếu**

Mô phỏng 4.3. Giải thuật của chương trình phân tích thứ tự yếu

- Lúc đầu stack trạng thái chỉ có ký hiệu $. Stack nhập chứa chuỗi nhập, được kết thúc bởi dấu $; c:=false ;

repeat

if *Ký hiệu mục tiêu ở trên đỉnh và ký hiệu $ ở đáy stack trạng thái, đồng thời stack nhập chỉ chứa $* **then**

c:=true /*phân tích thành công, cây cú pháp xây dựng xong*/

else begin

- X ở trên đỉnh stack trạng thái, Y ở trên đỉnh stack nhập.
- Giả sử T là trị của phần tử S-R [X, Y];

if *T là rỗng* **then** error ()

else if *T = R* **then**

if *trên đỉnh stack có chứa vế phải của luật sinh nào đó*

then begin

- Gọi A → $X_1 X_2 ... Xn$ là luật sinh nào có vế phải dài nhất so trùng với chuỗi trên stack trạng thái: (a) Giải tỏa $X_1 X_2 ... X_n$ ra khỏi stack; (b) Thay A lên stack. (c) Tạo nút mới A trên cây cú pháp, có các con là $X_1 X_2 ... X_n$

end

```
                else error ()
        else begin
                (a) Giải tỏa Y ra khỏi stack nhập; (b) Đẩy Y lên đỉnh
stack trạng thái; (c) Tao nút mới tên Y trên cây cú pháp;
        end;
    end;
until c;
```

3. Xây dựng bảng phân tích S-R

Định nghĩa các quan hệ $<\bullet$, $\doteq$, $\bullet >$:

- Chúng ta nói X $<\bullet$ Y nếu và chỉ nếu tồn tại một luật sinh mà vế phải có dạng ... XA với A là ký hiệu không kết thúc và sinh ra một chuỗi bắt đầu bằng Y ($A \Rightarrow Y...$).
- X $\bullet >$ Y nếu và chỉ nếu tồn tại một luật sinh mà vế phải có dạng ...AB. A sinh ra một chuỗi ký hiệu được kết thúc bằng X ($A \Rightarrow ...X$). B sinh ra một chuỗi được bắt đầu bằng Y ($B \Rightarrow Y...$), hoặc B = Y.

Ở đây có hai trường hợp xảy ra trong quá trình tìm các mối quan hệ cho cặp (X, Y):

Trường hợp 1: Y là ký hiệu kết thúc

Trường hợp 2: Y là ký hiệu không kết thúc.

a. Tồn tại $ $\leq\bullet$ A với A là ký hiệu mục tiêu của văn phạm cho trước.

b. Nếu vế phải luật sinh có X nằm kề ngay Y về phía trái (...XY...) thì X $\leq\bullet$ Y

c. Nếu X $\leq\bullet$ Y mà tồn tại một luật sinh Y $\rightarrow Z_1 \dots Z_n$ thì X $\leq\bullet$ Z_1

d. Tồn tại A $\bullet>$ $ với A là ký hiệu mục tiêu

e. Nếu X $\leq\bullet$ Y và tồn tại một luật sinh X $\rightarrow Z_1 \dots Z_n$ thì $Z_n > Y$

g. Nếu X $\bullet>$ Y và tồn tại một luật sinh Y $\rightarrow Z_1 \dots Z_n$ thì $X > Z_1$

4. Văn phạm thứ tự yếu

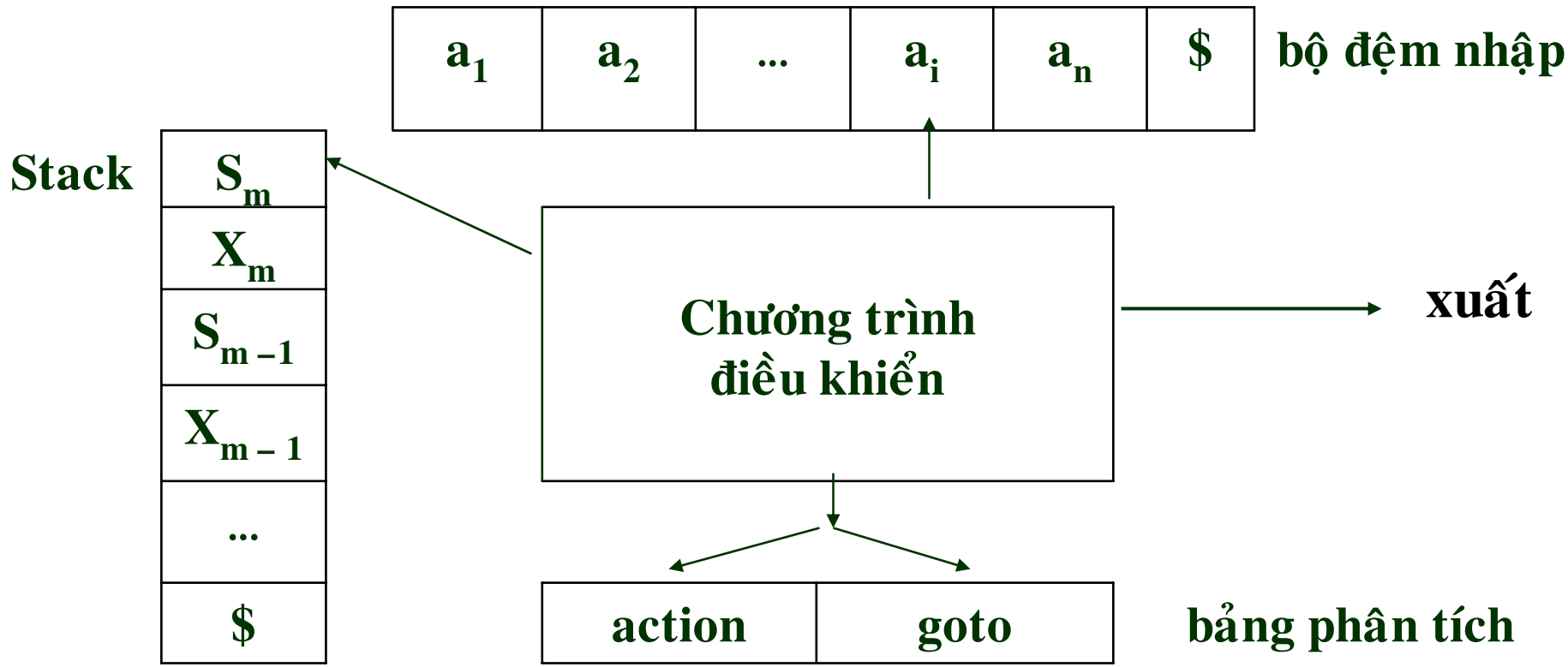
Một văn phạm được gọi là thứ tự yếu nếu thỏa các điều kiện sau:

1. Không có hai luật sinh có cùng một vế phải.
2. Không có phần tử S-R [X, Y] nào của bảng S-R vừa có trị S vừa có trị R.
3. Nếu tồn tại luật sinh A$\rightarrow X_1 X_2 \dots X_n$ và luật sinh B$\rightarrow X_i X_{i+1} \dots X_n$ thì không tồn tại quan hệ X_{i-1} $\leq\bullet$ B.

2. Bộ phân tích cú pháp LR

- Các tính chất của phương pháp phân tích LR
- Giải thuật phân tích cú pháp LR

1. Bộ phân tích cú pháp có cấu tạo như sau:



Hình 4.11. Mô hình bộ phân tích cú pháp LR

2. Hoạt động

Stack được dùng để chứa chuỗi ký hiệu có dạng $s_0 X_1 s_1 X_2 ... X_m s_m$. Cặp ($s_m$, a_i) sẽ xác định một trị được lưu chứa trong bảng phân tích. Bảng phân tích gồm hai phần biểu thị bởi hàm action và goto. Cấu hình (configuration) của bộ phân tích LR:

$s_0 X_1 s_1 ... X_i s_i ... X_m s_m$, $a_i a_{i+1} ... a_n$ \$). Cấu hình này cho chúng ta dạng câu $X_1 X_2 ... X_m a_i a_{i+1} ... a_n$.

***Giải thuật 4.5.* Phân tích cú pháp LR**

Nhập: chuỗi nhập w, bảng phân tích action goto của văn phạm G.

Xuất: nếu w thuộc L (G), nó tạo ra sự phân tích từ dưới lên. Ngược lại, bộ phân tích sẽ báo lỗi.

Phương pháp:

- Thời điểm ban đầu stack có trạng thái s_0.
- Chuỗi w\$ nằm trên bộ đệm nhập.
- Bộ phân tích đặt đầu đọc (con trỏ ip) vào ký hiệu nhập đầu tiên của w.

c:=false; /*c là biến luận lý, báo cho biết quá trình phân tích kết thúc*/

repeat

- Đặt s là trạng thái trên đỉnh stack a là ký hiệu nhập được ip chỉ đến

if *action [s, a] = shift(s')* **then begin**

(a)đẩy a lên stack (b)sau đó đẩy s' lên đỉnh stack (c)chuyển ip sang ký hiệu nhập kế tiếp; **end**

else if *action [s, a] = reduce(A → β)* **then**

begin

(a)đẩy (2*|β|) ký hiệu ra khỏi stack, s' là trạng thái trên đỉnh stack

(b)Tìm j = goto [s', A]; (c)đẩy A và sau đó là j lên đỉnh stack; (d)xuất luật A → β

end

else if *action [s, a] = accept* **then** c := true

else error ();

until c;

Thí dụ 4.12. Cho văn phạm G

(1) E → E + T (2) E → T (3) T → T * F

(4) T → F (5) F → (E) (6) F → id

Bảng 4.8. *Bảng phân tích cho văn phạm G ở thí dụ 4.12.*

Trạng thái	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}	s_{11}			
9		r_1	s_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

Thí dụ: Phân tích câu w = id * id + id

Văn phạm LR

Xây dựng bảng phân tích SLR

Định nghĩa thực thể LR (0)

Thí dụ: G có luật sinh A → XYZ, sẽ cho bốn thực thể:

A→•XYZ; A→X•YZ; A→XY•Z; A→XYZ•

Nếu A → ∈ sẽ cho ta thực thể A → •

Ý tưởng cơ bản của giải thuật xây dựng bảng phân tích SLR là từ văn phạm, chúng ta đi tìm DFA, nhận dạng chuỗi dẫn đầu bên trái của dạng câu (viable prefixe).

Định nghĩa văn phạm gia tố: nếu G là văn phạm, thì G' là văn phạm gia tố, là G có S' là ký hiệu mục tiêu và có thêm luật sinh

S' → S.

Phép bao đóng – Closure.

Giải thuật tính closure.

Mô phỏng 4.4. *Giải thuật tính hàm closure*

```
function closure (I : item) : item;
    begin J := I;
        repeat
            for với mỗi thực thể A → α.Bβ trong J và với mỗi
                                                luật sinh
                B → γ trong G sao cho
                thực thể B →  γ chưa có trong J do
                thêm B →  γ vào J;
    until không thể thêm thực thể mới vào J;
        closure := J;
    end;
```

Giải thuật tính goto: hàm goto (I, X) với I là tập các thực thể, X là ký hiệu văn phạm. Goto (I, X) là closure của tập các thực thể có dạng A → αX.β sao cho thực thể A → α.Xβ ở trong I.

Mô phỏng 4.5. *Giải thuật tính tập tuyển các tập thực thể*

procedure items (G');
begin
C := {closure ({S' → S}}}
repeat
for với mỗi tập thực thể I trong C và với mỗi ký hiệu văn phạm X sao cho phép goto (I, X) không rỗng và không có trong C **do**
thêm goto (I, X) vào C;
until không thể thêm tập thực thể mới vào C;
end;

Thí dụ 4.13. Cho văn phạm gia tố G'

E' → E ; E → E + T ; E → T

T → T* F ; T → F ; F → (E) ; F → id

Hãy tìm tập C và sơ đồ DFA.

Xây dựng bảng phân tích SLR

Giải thuật 4.6. **Xây dựng bảng phân tích**

Nhập: văn phạm gia tố G'

Xuất: bảng phân tích SLR với hàm action và goto cho văn phạm G'

Phương pháp:

1. Xây dựng C = $\{I_o, I_1, \ldots I_n\}$.
2. i là trạng thái đại diện cho tập thực thể I_i.

   a. Nếu A → α•aβ là thực thể ở trong I_i và goto $(I_i, a) = I_j$ thì phần tử action [i, a] = shift(j), với a phải là ký hiệu kết thúc.

   b. Nếu A → α• ở trong I_i thì action [i, a] = reduce(A →α) với a là tất cả các ký hiệu nằm trong follow (A). A không phải là S' (ký hiệu mục tiêu mới).

   c. Nếu S' → S• ở trong I_i thì action [i, $] = accept.
3. Cho tất cả các ký hiệu không kết thúc A. Nếu goto $[I_i, A]=I_j$ thì hàm goto [i, A]=j.
4. Tất cả các phần tử của bảng phân tích không được xác định bằng quy tắc 2 và 3, chúng ta coi là lỗi.
5. Trạng thái bắt đầu của bộ phân tích là tập thực thể có chứa thực thể S' →•S.

Thí dụ 4.14. Xây dựng bảng phân tích SLR cho văn phạm G ở thí dụ 4.13.

Thí dụ 4.15. Cho văn phạm G.

(1) $S \rightarrow L = R$
(2) $S \rightarrow R$
(3) $L \rightarrow * R$
(4) $L \rightarrow id$
(5) $R \rightarrow L$

Ta nhận thấy đụng độ khi action [2, =] = s_6 đồng thời action [2, =] = r_5 và action [2, $] = r_5. Do đó tại phần tử action [2, =] có hai trị s_6 và r_5. Như vậy G không phải là văn phạm SLR.

Xây dựng bảng phân tích Canonical LR

Dạng tổng quát của thực thể là [A → α.β, a] với A → αβ là luật sinh và a là ký hiệu kết thúc hoặc dấu $. Thực thể có dạng như thế được gọi là thực thể LR (1). Nếu β = ∈ thì thực thể sẽ có dạng [A → α , a]. Lúc này chúng ta thực hiện thu giảm bằng luật sinh A → α chỉ với điều kiện ký hiệu nhập kế tiếp là a.

Chúng ta nói thực thể LR (1) [A → α.β, a] là hợp lệ với chuỗi ký hiệu dẫn đầu dạng câu γ nếu tồn tại dẫn xuất phải:

$$S \underset{rm}{\Rightarrow} \delta Aw \underset{rm}{\Rightarrow} \delta\alpha\beta w \quad \text{với}$$

1. $\gamma = \delta\alpha$ và
2. hoặc a là ký hiệu dẫn đầu của w, hoặc $w = \in$ thì a là $.

Thí dụ 4.16. Cho văn phạm G

S → BB

B → aB | b

Tính tập tuyển các thực thể LR (1)

Phép tính closure

Giải thuật 4.7. Xây dựng các tập thực thể LR (1).

Mô phỏng 4.7. *Giải thuật tính các tập thực thể LR (1) cho văn phạm gia tố G'*

```
function closure (I: items): items;
    begin
        repeat
            for với mỗi thực thể [A → α  Bβ, a] trong , với mỗi
            luật sinh B → η trong G' và với mỗi ký hiệu kết
            thúc b thuộc first sao cho thực thể [B →  η, b]
            không có trong I do
            thêm thực thể [B → η, b] vào I
        until không thể thêm thực thể mới vào I;
closure := I;
    end;
function goto (I :items; X: symbol): items;
    begin
        j là tập các thực thể có dạng [A → αX  β, a] sao cho
        thực thể [A → α  Xβ, a]  ở trong I ; goto := closure (J);
    end;
```

```
procedure items (G');
        begin
            d := {closure ({S' → S, $}};
            repeat
                for với mỗi tập thực thể I ở trong C và với mỗi ký
                hiệu văn phạm X sao cho goto (I, X) không rỗng và
                chưa có trong C do thêm goto (I, X) vào C;
            until không thể thêm tập thực thể mới vào C;
    end;
```

Thí dụ 4.17. Xây dựng các tập thực thể LR (1) cho văn phạm gia tố G': S' → .S ; S → CC ; C → cC|d

***Giải thuật 4.8.* Xây dựng bảng phân tích Canonical LR.**

Nhập: văn phạm gia tố G'

Xuất: bảng phân tích Canonical LR với hai hàm action và goto cho G'

Phương pháp:

1. Xây dựng C = $\{I_o, I_1, ..., I_n\}$.
2. Trạng thái i đại diện cho I_i.

a. Nếu thực thể [A → α.aβ, b] ở trong Ii và goto (I_i , a) = I_j thì phần tử action [i, a] = shift(j), a phải là ký hiệu kết thúc.

b. Nếu [A → α• , a] ở trong I_i, A ≠ S' thì action[i, a]=reduce(A→α)

c. Nếu [S' → S• , $] ở trong I_i thì action [i, $] = accept.

3. Nếu goto (I_i , A) = Ij thì phần tử goto [i, A] = j.
4. Tất cả các phần tử không áp dụng được quy tắc 2 và 3 thì là lỗi.
5. Trạng thái bắt đầu của bộ phân tích cú pháp là tập thực thể co chứa thực thể [S' → •S , $].

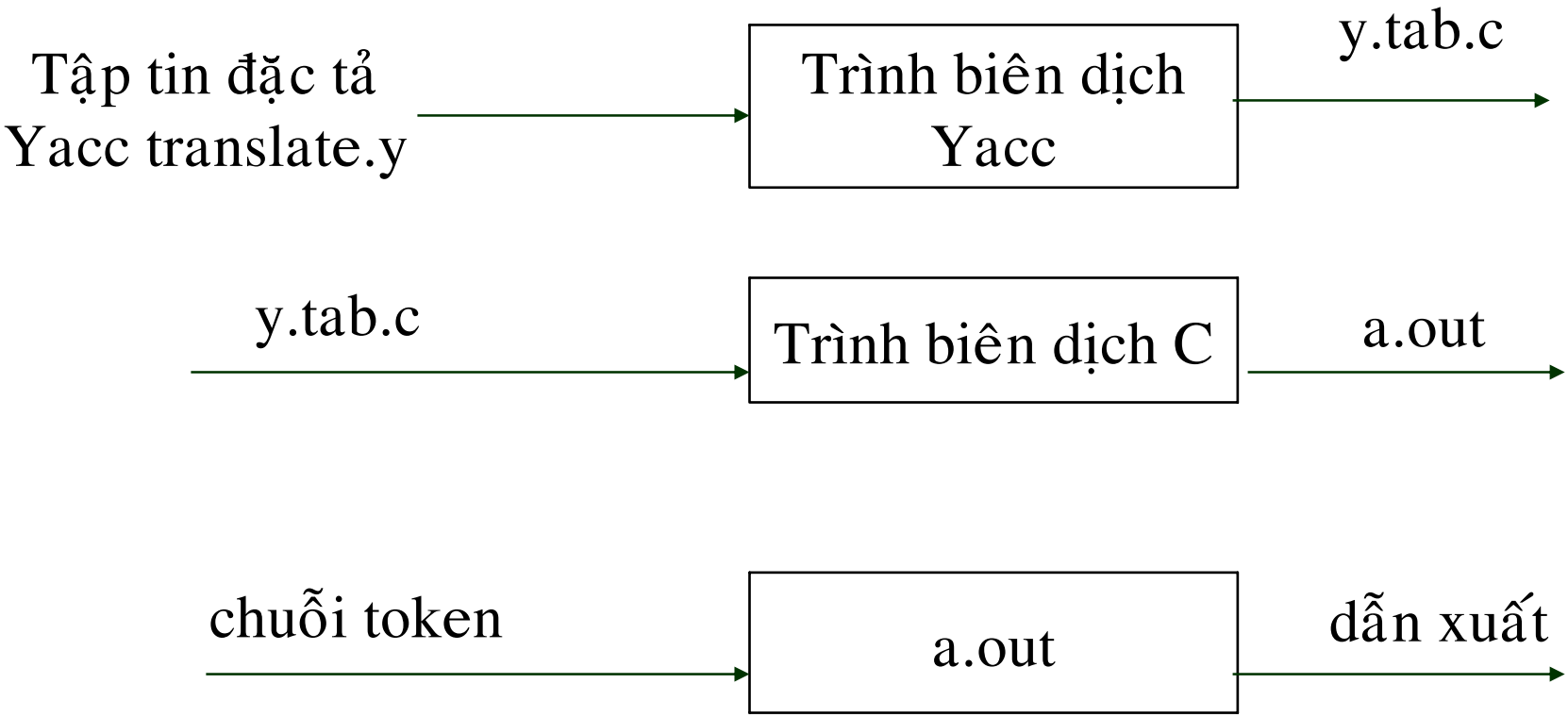
Bảng phân tích Canonical LR cho văn phạm ở thí dụ 4.17. được xây dựng dựa vào giải thuật 4.7.

Bảng 4.10. *Bảng phân tích Canonical LR*

Trạng thái	*action*			*goto*	
	c	**d**	**$**	**S**	**C**
0	s_3	s_4		1	2
1			acc		
2	s_6	s_7			5
3	s_3	s_4			8
4	r_3	r_3			
5			r_1		
6	s_6	s_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

Bộ sinh phân tích cú pháp

Bộ sinh phân tích cú pháp Yacc



Hình 4.14. Tạo bộ phân tích cú pháp bằng Yacc.

Thí dụ 4.23. Chúng ta sẽ tạo tập tin đặc tả văn phạm cho Yacc của văn phạm G.

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

Mô phỏng 4.10. *Tập tin đặc tả văn phạm cho Yacc ở thí dụ 4.23.*

```
% { # include <ctype.h>
        % }
        % token DIGIT
        % %
        line : exp '\n'{printf ("% d\n", $1) ;}
                    ;
        exp : exp '+' term {$$ = $1 + $3;}
                    : term
                    ;
        term : term '*' factor {$$ = $1 + $3;}
                    : factor
                    ;
```

```
factor : '('exp')' {$$ = $2;}
                : DIGIT
                ;
%%
        yylex ( ) {
                int c ;
                c = getchar ( ) ;
                if (isdigit (c)) {   yylval = c - '0' ;
                                        return DIGIT;
                                     }
                        return c;
                        }
```

Phần đặc tả

Phần các luật biên dịch:

<vế trái luật sinh> → <vế phải thứ nhất> |... | <vế phải thứ n>

Luật biên dịch trong Yacc:

<vế trái LS> : <vế phải 1> {hành vi ngữ nghĩa 1}
: <vế phải 2> {hành vi ngữ nghĩa 2}
...
: <vế phải n> {hành vi ngữ nghĩa n}

Phần các chương trình con C phụ trợ