Bài 4.
PHÂN TÍCH CÚ PHÁP

Hoàng Anh Việt
Viện CNTT&TT - ĐHBKHN

Mục đích

- Sau khi học xong chương này, sinh viên sẽ nắm được:
  - Các phương pháp phân tích cú pháp
  - Cách cài đặt một bộ PTCP từ một Văn phạm phi ngữ cảnh
  - Các khái niệm và sử dụng công cụ sinh bộ PTCP: Yacc.

Điều kiện

- Kiến thức cần có:
  - Kiến thức cơ bản về Automat.
  - Kiến thức về văn phạm phi ngữ cảnh CFG.

Tài liệu tham khảo

[1] Slide bài giảng

[2] Compilers : Principles, Technique and Tools - Alfred V.Aho, Jeffrey D.Ullman - Addison - Wesley Publishing Company, 1986.

[3] Automata and Formal Language, An Introduction- Dean Kelley- Prentice Hall, Englewood Cliffs, New Jersey 07632

[4] Compilers course, CS 143 summer 2010, Standford University.

[5] Compiler Design – Reinhard Wilhelm, Dieter Maurer - Addison – Wesley Publishing Company, 1996.

Nội dung

1. Vai trò của bộ phân tích cú pháp (PTCP)
2. Văn phạm của ngôn ngữ lập trình
3. Phân tích cú pháp từ trên xuống
4. Phân tích cú pháp từ dưới lên
5. Bộ sinh bộ PTCP

Nội dung

1. **Vai trò của bộ phân tích cú pháp (PTCP)**
2. Văn phạm của ngôn ngữ lập trình
3. Phân tích cú pháp từ trên xuống
4. Phân tích cú pháp từ dưới lên
5. Bộ sinh bộ PTCP

1. Vai trò của bộ phân tích cú pháp

- Bộ phân tích cú pháp nhận chuỗi các token từ bộ phân tích từ vựng để tạo ra cấu trúc cú pháp của chương trình nguồn.
- Tồn tại ba loại bộ phân tích cú pháp:
  - Phương pháp tổng quát:
    - Cocke-Younger-Kasami.
    - Earley.
- Phương pháp thông dụng: Phân tích **từ trên xuống** hay phân tích **từ dưới lên**.

1. Vai trò của bộ phân tích cú pháp

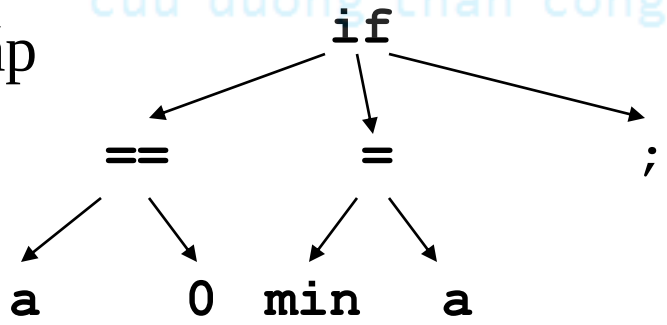
Mã nguồn (dãy các kí tự)

```
If (a == 0) min = a;
```

Dãy các từ tố (token)

If	(	Id:a	==	0	)	Id:min	=	Id:a	;

Cây cú pháp



Phân tích từ

Phân tích cú

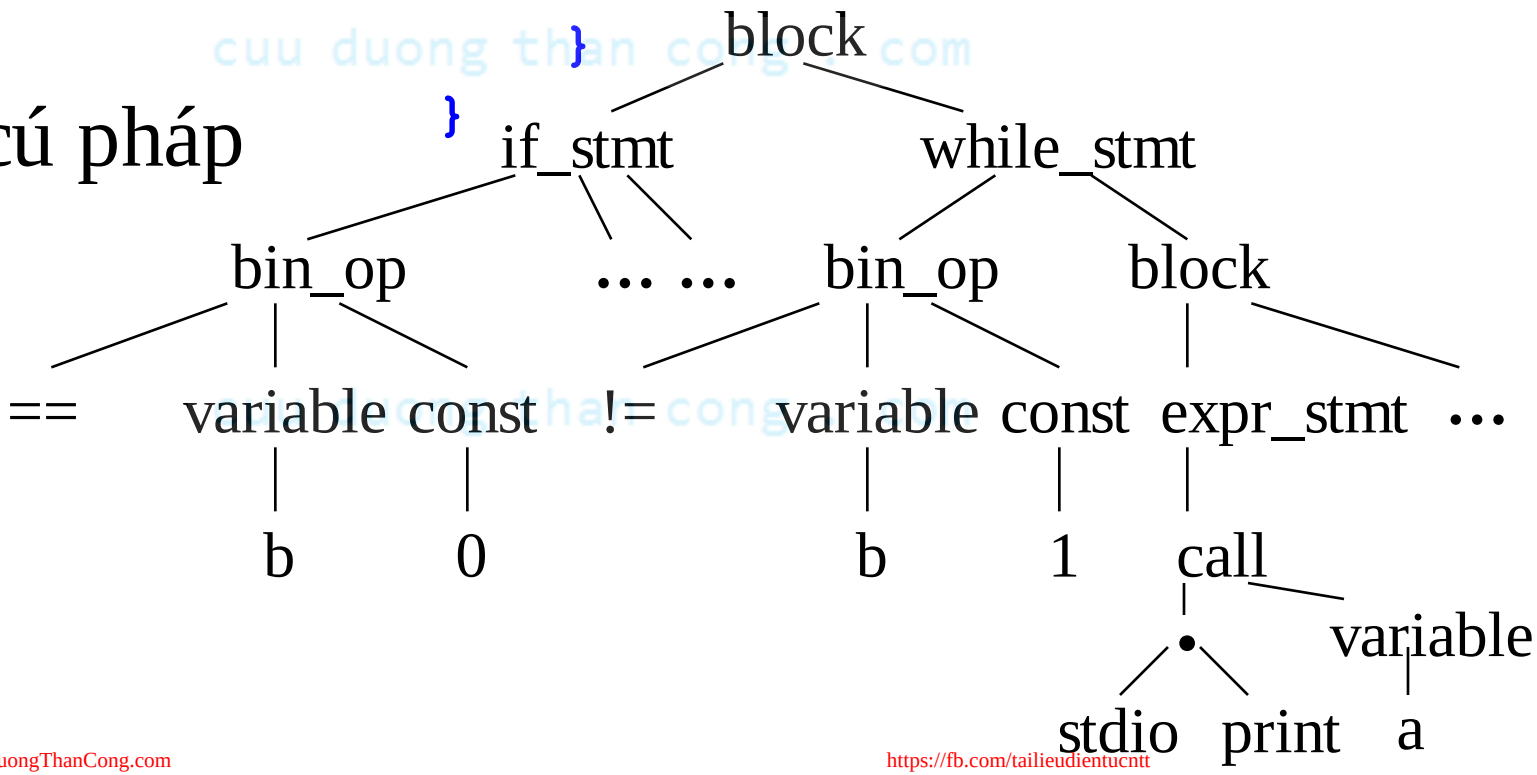
Phân tích ngữ

1. Vai trò của bộ phân tích cú pháp

- Mã nguồn

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

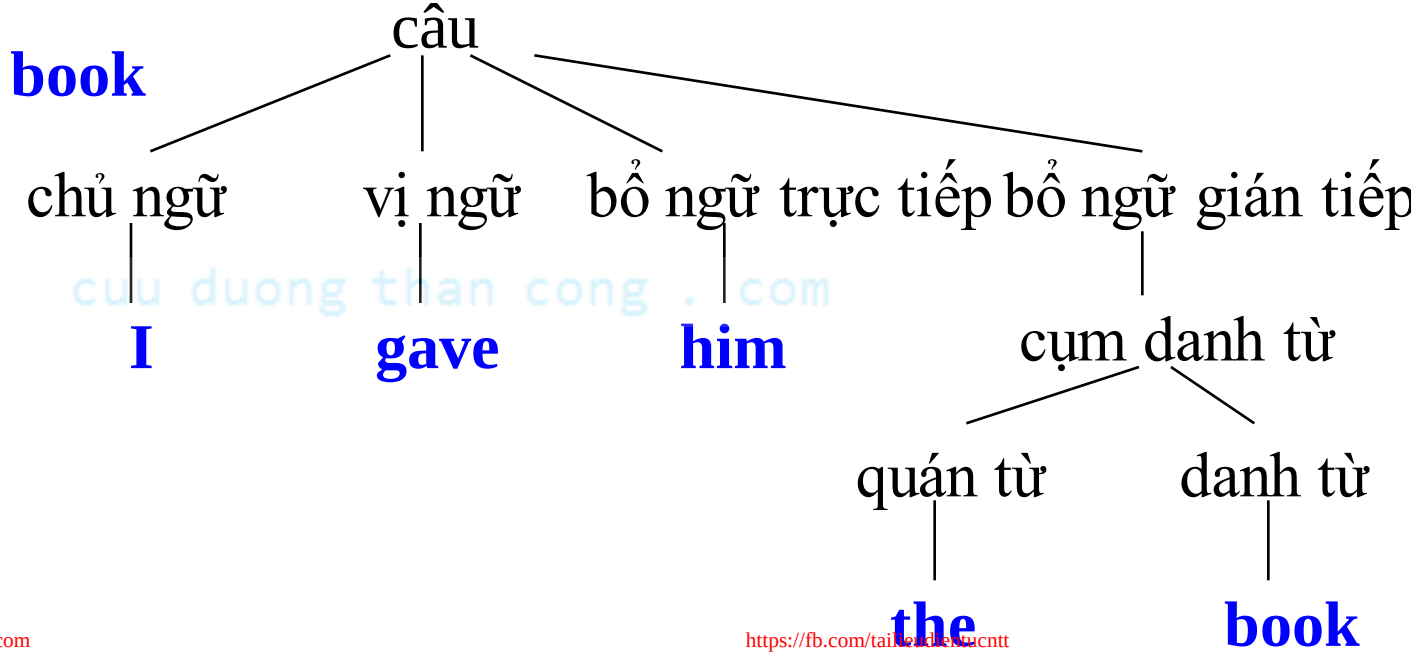
- Cây cú pháp



1. Vai trò của bộ phân tích cú pháp

- Kiểm tra tính đúng đắn về cú pháp của chương trình nguồn
- Xác định chức năng của các thành phần trong chương trình nguồn

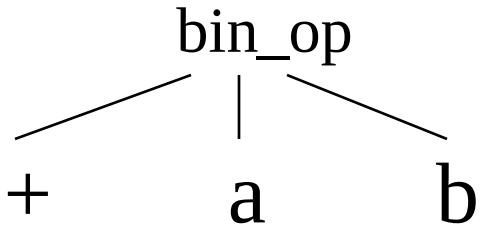
I gave him the book



1. Vai trò của bộ phân tích cú pháp

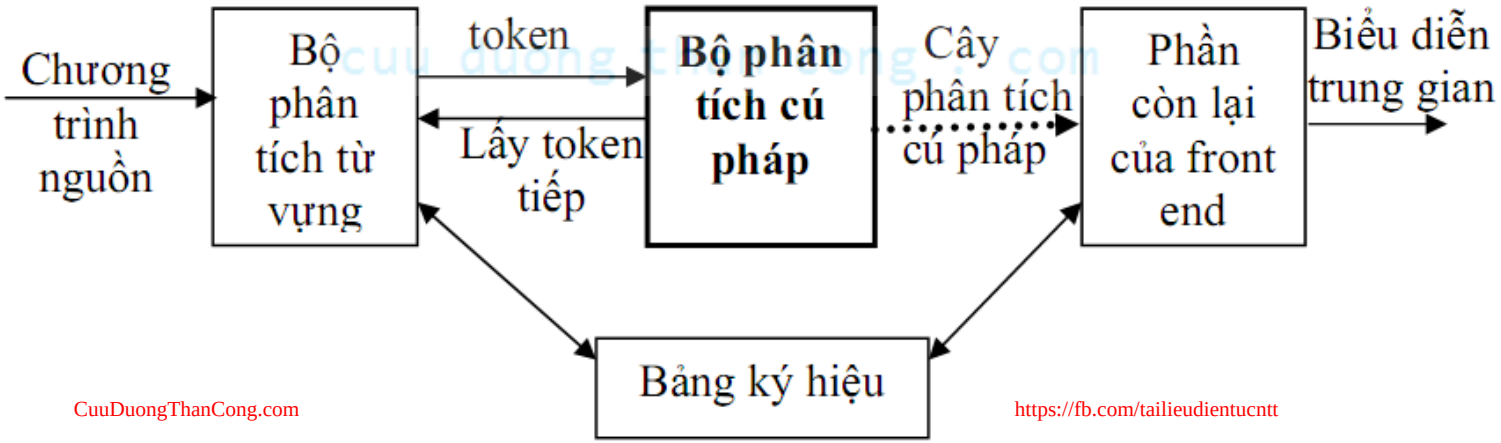
- **Input**: Dãy các từ tố
- **Output**: Cây cú pháp
- **Cài đặt**:
  - Duyệt qua dãy các từ tố
  - Xây dựng cây cú pháp
  - Loại bỏ các cú pháp thừa trong cây cú pháp

VD: a+b ≈ (a)+(b) ≈ ((a)+((b)))



1. Vai trò của bộ phân tích cú pháp

- Phân tích cú pháp không làm tất cả mọi công đoạn của chương trình dịch. Ví dụ: kiểm tra kiểu, khai báo biến, khởi tạo biến… =>Để lại cho phần phân tích ngữ nghĩa.
- Bộ PTCP có cơ chế ghi nhận và xử lý các lỗi cú pháp thường gặp.



Nội dung

1. Vai trò của bộ phân tích cú pháp (PTCP)
2. **Văn phạm của ngôn ngữ lập trình**
3. Phân tích cú pháp từ trên xuống
4. Phân tích cú pháp từ dưới lên
5. Bộ sinh bộ PTCP

2. Văn phạm của ngôn ngữ lập trình

2.1 Đặc tả cú pháp của ngôn ngữ

2.2 Văn phạm nhập nhằng

2.3 Loại bỏ nhập nhằng

cuu duong than cong . com

2.1. Đặc tả cú pháp của ngôn ngữ

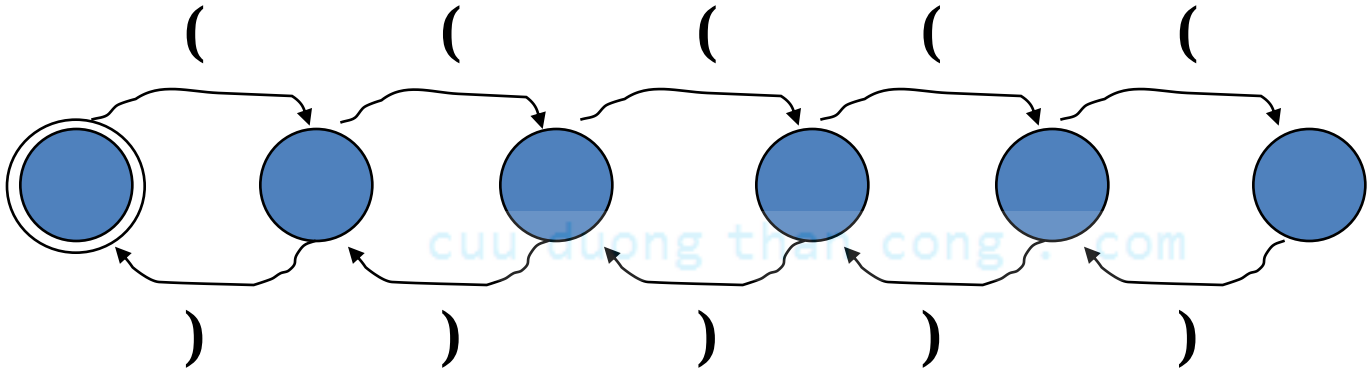
- Vấn đề: Làm thế nào để mô tả chính xác và dễ dàng cú pháp của ngôn ngữ tạo nên mã nguồn?
- Giống như từ tố được mô tả bằng REs
- REs dễ cài đặt (bằng NFA hoặc DFA)
- Có thể dùng REs để mô tả cú pháp của ngôn ngữ lập trình được không?

2.1. Đặc tả cú pháp của ngôn ngữ

- Cú pháp của ngôn ngữ lập trình không thuộc lớp ngôn ngữ chính quy → không thể mô tả bằng REs được
- Ví dụ: $L \subseteq \{ (,) \}^*$ sao cho L là tập các cách viết () đúng.
- Nếu dùng RE để biểu diễn L → phải đếm số lượng dấu "(" chưa có dấu ")" tương ứng → số đếm không bị giới hạn

2.1. Đặc tả cú pháp của ngôn ngữ

- Ta biết: RE ⇔ DFA
- Số đếm không giới hạn → số trạng thái không giới hạn → mâu thuẫn với cấu trúc của DFA (hữu hạn)



< 6: OK **>=6**

2.1. Đặc tả cú pháp của ngôn ngữ

- Nhiều NNLT có cấu trúc đệ quy mà nó có thể được định nghĩa bằng các VP PNC (context-free grammar) G với 4 thành phần G (V, T, P, S), trong đó:
  - **V** : là tập hữu hạn các ký hiệu chưa kết thúc hay các biến (variables)
  - **T** : là tập hữu hạn các ký hiệu kết thúc (terminals).
  - **P** : là tập luật sinh của văn phạm (productions).
  - **S** ∈ V: là ký hiệu bắt đầu của văn phạm (start symbol).

2.1. Đặc tả cú pháp của ngôn ngữ

- Ví dụ: mô tả ngôn ngữ L

  $S \rightarrow (S)S$

  $S \rightarrow \varepsilon$

- CFG sử dụng định nghĩa đệ quy
- CFG trực quan hơn REs

  $S \Rightarrow (S)S \Rightarrow ((S)S)S \Rightarrow ((\varepsilon)S)S \Rightarrow \ldots \Rightarrow (())$

- Một xâu nằm trong ngôn ngữ của CFG nếu có một dãy suy dẫn sử dụng các **sản xuất** của CFG tạo nên xâu đó

2.1. Đặc tả cú pháp của ngôn ngữ

- Cây PTCP có thể được xem như một dạng biểu diễn hình ảnh của một **dẫn xuất**.
- $\alpha A\beta$ **dẫn xuất** ra $\alpha\gamma\beta$ (ký hiệu: $\alpha A\beta \Rightarrow \alpha\gamma\beta$) nếu $A \rightarrow \gamma$ là một **luật sinh**, α và β là các chuỗi tùy ý các ký hiệu văn phạm.
- Nếu $\alpha 1 \Rightarrow \alpha 2 \Rightarrow .. .. \Rightarrow \alpha n$ ta nói α_1 dẫn xuất ra (suy ra) αn
- Ký hiệu: $\Rightarrow$: dẫn xuất ra qua 1 bước

  $\Rightarrow^*$: dẫn xuất ra qua 0 hoặc nhiều bước.

  $\Rightarrow^+$: dẫn xuất ra qua 1 hoặc nhiều bước.

2.1. Đặc tả cú pháp của ngôn ngữ

- Tính chất của Dẫn xuất:
  1. $\alpha \Rightarrow^* \alpha$ với $\forall\alpha$
  2. 2. $\alpha \Rightarrow^* \beta$ và $\beta \Rightarrow^* \gamma$ thì $\alpha \Rightarrow^* \gamma$
- **Ví dụ 4.1:** xét xâu (1 + 2 + (3 + 4)) + 5, và bộ luật sinh:

  S → E + S

  S → E

  E → số

  E → (S)

2.1. Đặc tả cú pháp của ngôn ngữ

S → E + S | E

E → số | (S)

kí hiệu không kết thúc – vế trái

vế phải của sản xuất

- Xâu (1 + 2 + (3 + 4)) + 5

S ⇒ **E** + S ⇒ **(S)** + S ⇒ (**E** + S) + S

⇒ (1 + **S**) + S ⇒ (1 + **E** + S) + S

⇒ (1 + 2 + **S**) + S ⇒ (1 + 2 + **E**) + S

⇒ (1 + 2 + (**S**)) + S ⇒ (1 + 2 + (**E** + S)) + S

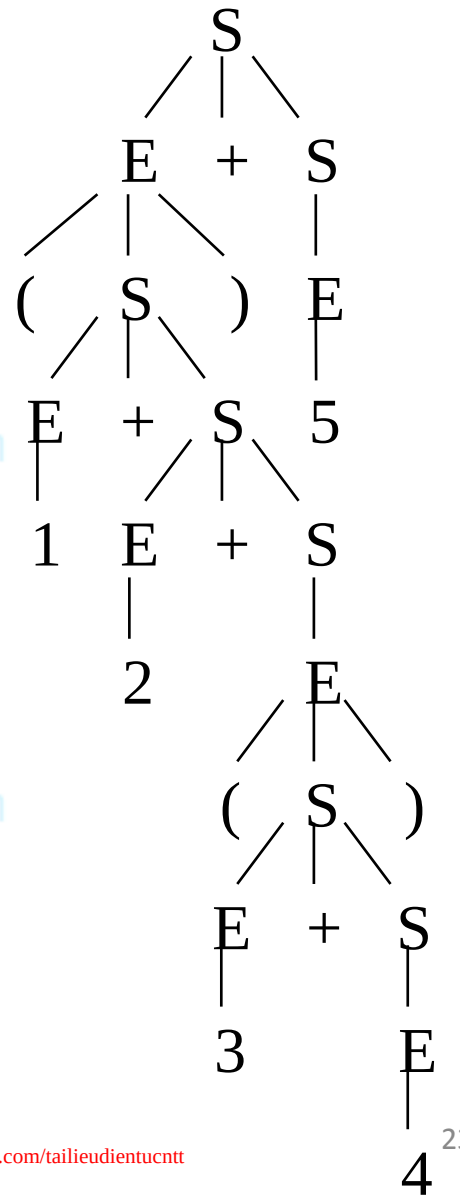
⇒ (1 + 2 + (3 + **S**)) + S ⇒ (1 + 2 + (3 + **E**)) + S

⇒ (1 + 2 + (3 + 4)) + **S** ⇒ (1 + 2 + (3 + 4)) + **E**

⇒ (1 + 2 + (3 + 4)) + 5

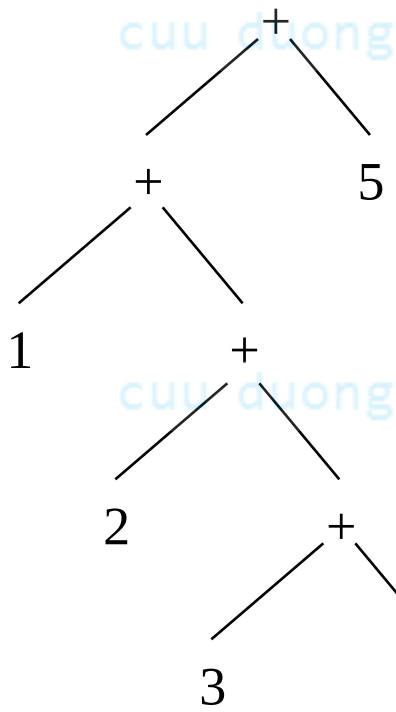
2.1. Đặc tả cú pháp của ngôn ngữ

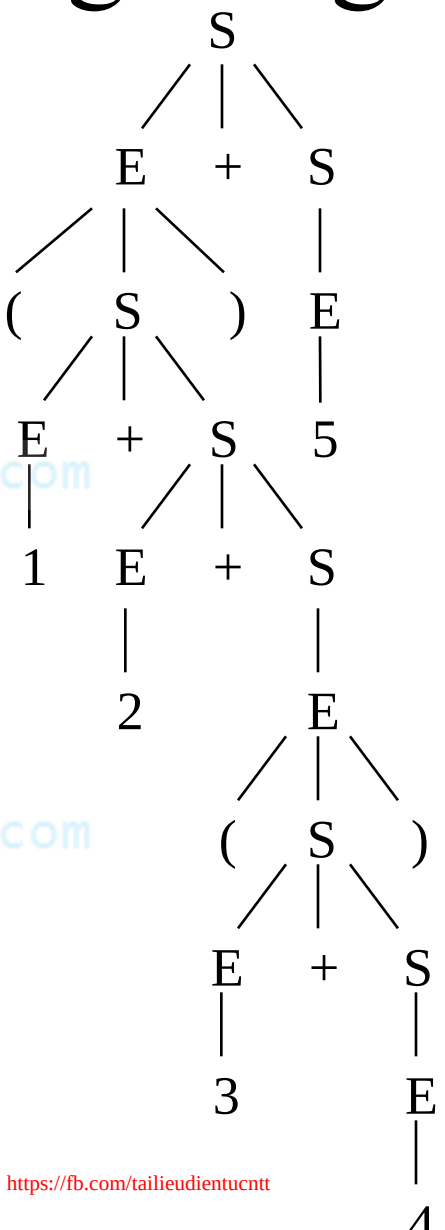
- Cây suy dẫn:
  - Một dãy dẫn xuất bắt đầu từ S có thể mô tả dưới dạng cây suy dẫn
    - Lá của cây là kí hiệu kết thúc; theo thứ tự duyệt sẽ tạo thành xâu vào
    - Nút trong của cây là các kí hiệu không kết thúc
    - Cây không chỉ rõ thứ tự của các dẫn xuất



2.1. Đặc tả cú pháp của ngôn ngữ

- **Cây cú pháp**:
  - Giản lược các thông tin thừa khỏi cây suy dẫn





2.1. Đặc tả cú pháp của ngôn ngữ

- Thứ tự dẫn xuất tùy ý, có thể chọn bất cứ một kí hiệu không kết thúc nào để áp dụng sản xuất
- Hai thứ tự dẫn xuất thường dùng:
  - Suy dẫn trái: chọn kí hiệu bên trái nhất
  - Suy dẫn phải: chọn kí hiệu bên phải nhất
- Được sử dụng trong nhiều kiểu phân tích cú pháp tự động (automatic parsing)

2.1. Đặc tả cú pháp của ngôn ngữ

- **Suy dẫn trái**

  $\boldsymbol{S} \Rightarrow \boldsymbol{E}+S \Rightarrow (\boldsymbol{S}) + S \Rightarrow (\boldsymbol{E} + S)+ S \Rightarrow (1 + \boldsymbol{S})+S$
  $\Rightarrow (1+\boldsymbol{E}+S)+S \Rightarrow (1+2+\boldsymbol{S})+S \Rightarrow (1+2+\boldsymbol{E})+S$
  $\Rightarrow (1+2+(\boldsymbol{S}))+S \Rightarrow (1+2+(\boldsymbol{E}+S))+S \Rightarrow (1+2+(3+\boldsymbol{S}))+S$
  $\Rightarrow (1+2+(3+\boldsymbol{E}))+S \Rightarrow (1+2+(3+4))+\boldsymbol{S} \Rightarrow (1+2+(3+4))+\boldsymbol{E}$
  $\Rightarrow (1+2+(3+4))+5$

- **Suy dẫn phải**

  $\boldsymbol{S} \Rightarrow E+\boldsymbol{S} \Rightarrow E+\boldsymbol{E} \Rightarrow \boldsymbol{E}+5 \Rightarrow (\boldsymbol{S})+5 \Rightarrow (E+\boldsymbol{S})+5$
  $\Rightarrow (E+E+\boldsymbol{S})+5 \Rightarrow (E+E+\boldsymbol{E})+5 \Rightarrow (E+E+(\boldsymbol{S}))+5$
  $\Rightarrow (E+E+(E+\boldsymbol{S}))+5 \Rightarrow (E+E+(E+\boldsymbol{E}))+5$
  $\Rightarrow (E+E+(\boldsymbol{E}+4))+5 \Rightarrow (E+\boldsymbol{E}+(3+4))+5$
  $\Rightarrow (\boldsymbol{E}+2+(3+4))+5 \Rightarrow (1+2+(3+4))+5$

- Cùng một cây suy dẫn, cùng sử dụng các dẫn xuất nhưng theo thứ tự khác nhau

2.2 Văn phạm nhập nhằng

- Xét văn phạm sau

$$S \rightarrow S + S \mid S * S \mid \text{number}$$

- Sử dụng dẫn xuất khác nhau cho ra các cây suy dẫn khác nhau => Văn phạm nhập nhằng

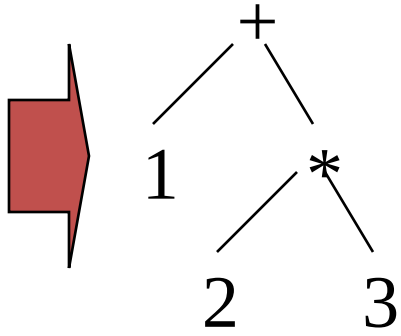
2.2 Văn phạm nhập nhằng

$$S \rightarrow S + S \mid S * S \mid \text{number}$$

- Nếu xâu vào là 1 + 2 * 3
- Suy dẫn 1:

  $S \Rightarrow S + S \Rightarrow 1 + S \Rightarrow 1 + S * S$

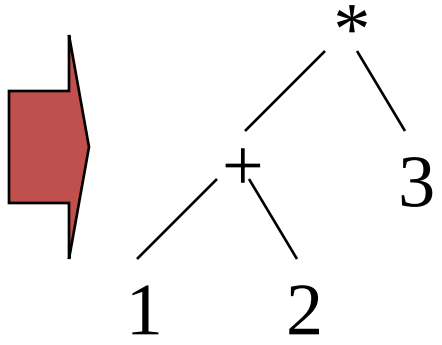
  $\Rightarrow 1 + 2 * S \Rightarrow 1 + 2 * 3$



- Suy dẫn 2:

  $S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow 1 + S * S$

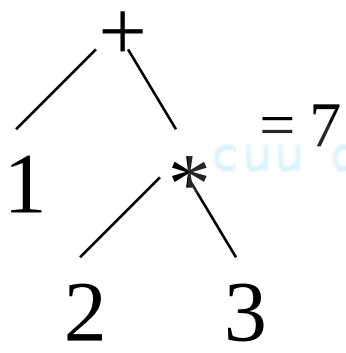
  $\Rightarrow 1 + 2 * S \Rightarrow 1 + 2 * 3$



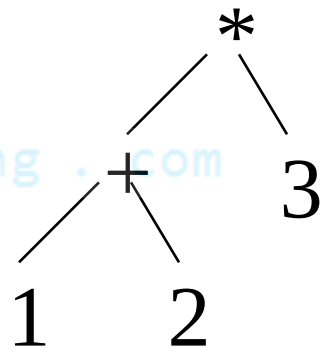
2.2 Văn phạm nhập nhằng

- Cây suy dẫn khác nhau cho kết quả tính toán khác nhau
- Văn phạm nhập nhằng

→ Có nhiều cách hiểu chương trình nguồn



= 7



= 9

2.3 Loại bỏ nhập nhằng

- Loại bỏ đệ quy trái
- Tạo yếu tố trái

2.3.1 Loại bỏ đệ quy trái

- Đệ quy trái:
  - Một văn phạm là đệ qui trái (left recursive) nếu nó có một ký hiệu chưa kết thúc A sao cho có một dẫn xuất , $A \Rightarrow^{+} A\alpha$ với α là một chuỗi nào đó.
  - Các phương pháp phân tích từ trên xuống không thể nào xử lý văn phạm đệ qui trái, do đó cần phải dùng một cơ chế biến đổi tương đương để loại bỏ các đệ qui trái.
  - Có 2 loại:
    - Trực tiếp: Dạng A -> A α
    - Gián tiếp: $A \Rightarrow^{i} A\alpha$ với i>=2

$S \rightarrow Aa \mid b$

$A \rightarrow Ac \mid Sd \mid \varepsilon$

$\Rightarrow S \Rightarrow Aa \Rightarrow Sda$

2.3.1 Loại bỏ đệ quy trái

- Với đệ quy trái trực tiếp: Luật sinh có dạng

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

Sẽ thay thế bởi:

$$\begin{cases} A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A' \\ A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon \end{cases}$$

- Với đệ quy trái gián tiếp: áp dụng giải thuật loại bỏ đệ quy (áp dụng cho đệ quy trái nói chung).

2.3.1 Loại bỏ đệ quy trái

Giải thuật 3.1: Loại bỏ đệ quy trái.

- **Input:** VP không tuần hoàn (không có luật sinh A=> $^{+}$A) và không có luật sinh ε (không có luật sinh A-> ε)
- **Output:** VP tương đương không đệ quy trái.
- **Các Bước:**

B1. Sắp xếp các ký hiệu không kết thúc theo thứ tự A_1 , A_2, …, A_n

2.3.1 Loại bỏ đệ quy trái

Giải thuật 4.1: Loại bỏ đệ quy trái (tiếp)

B2:

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 $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$

trong đó $A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ là tất cả các A_i luật sinh hiện tại;

end;

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

End;

2.3.1 Loại bỏ đệ quy trái

- Ví dụ: xét VP đệ quy

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sd \mid \varepsilon$$

1. Sắp xếp các ký hiệu chưa kết thúc theo thứ tự S, A.
2. Với i = 1, không có đệ qui trái trực tiếp nên không có điều gì xảy ra.

   Với i = 2, thay các S - luật sinh vào $A \rightarrow Sd$ được: $A \rightarrow Ac \mid Aad \mid bd \mid \varepsilon$

   Loại bỏ đệ qui trái trực tiếp cho các A luật sinh, ta được :

$$S \rightarrow Aa \mid b$$

$$A \rightarrow bdA'$$

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

2.3.2 Tạo yếu tố trái

- Là phép biến đổi văn phạm thuận tiện cho việc phân tích dự đoán (phân tích dự đoán?)
- 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.

2.3.2 Tạo yếu tố trái

- Một cách tổng quát: khi có luật $A \to \alpha\beta_1 \mid \alpha\beta_2$ thì biến đổi thành: $\begin{cases} A \to \alpha A' \\ A' \to \beta_1 \mid \beta_2 \end{cases}$
- **Giải thuật 4.2:** Tạo yêu tô trái cho văn phạm
  - Input: văn phạm G
  - Output: văn phạm tương đương với yếu tố trái.
  - Phương pháp: Tìm chuỗi **α** dẫn đầu chung của các vế phải luật sinh, thí dụ $A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma$

γ là chuỗi không bắt đầu bởi α. Ta thay các luật trên bằng các luật:

$A \to \alpha A' \mid \gamma$

$A' \to \beta 1 \mid \beta 2 \mid \beta 3 \ldots \mid \beta n$

2.3.2 Tạo yếu tố trái

- Ví dụ: Áp dụng thuật toán 3.2 cho văn phạm sau:

S -> iEtS | iEtSeS | a

E -> b

=> Văn phạm yếu tố trái tương đương:

$S \rightarrow \mathbf{i}\,E\,\mathbf{t}\,S\,S' \mid \alpha$

$S' \rightarrow eS \mid \varepsilon$

$E \rightarrow b$

Nội dung

1. Vai trò của bộ phân tích cú pháp (PTCP)
2. Văn phạm của ngôn ngữ lập trình
3. **Phân tích cú pháp từ trên xuống**
4. Phân tích cú pháp từ dưới lên
5. Bộ sinh bộ PTCP

3. Phân tích cú pháp từ trên xuống

3.1 Phân tích cú pháp đệ quy đi xuống

3.2 Phân tích cú pháp dự đoán

cuu duong than cong . com

3.1 Phân tích cú pháp đệ quy đi xuống

- Mục tiêu: xây dựng cây **suy dẫn trái** trong khi đọc dãy từ tố

Suy dẫn	Từ tố nhìn trước	Dãy từ tố Đã đọc / Chưa đọc
S	(	(1+2+(3+4))+5
E+S	(	(1+2+(3+4))+5
(S)+S	1	(1+2+(3+4))+5
(E+S)+S	1	(1+2+(3+4))+5
(1+S)+S	2	(1+2+(3+4))+5
(1+E+S)+S	2	(1+2+(3+4))+5
(1+2+S)+S	(	(1+2+(3+4))+5
(1+2+E)+S	(	(1+2+(3+4))+5
(1+2+(S))+S	3	(1+2+(3+4))+5

3.1 Phân tích cú pháp đệ quy đi xuống

- Ta muốn lựa chọn sản xuất dựa vào từ tố nhìn trước

S → E + S | E
E → num | (S)

(1) **S** ⇒ **E** ⇒ (**S**) ⇒ (**E**) ⇒ (1)

(1)+2 **S** ⇒ **E** + S ⇒ (**S**) + S ⇒ (**E**) + S

⇒ (1)+**E** ⇒ (1)+2

- Có thể quay lui để quét lại chuỗi nhập???

3.2 Phân tích cú pháp dự đoán (Predictive Parser)

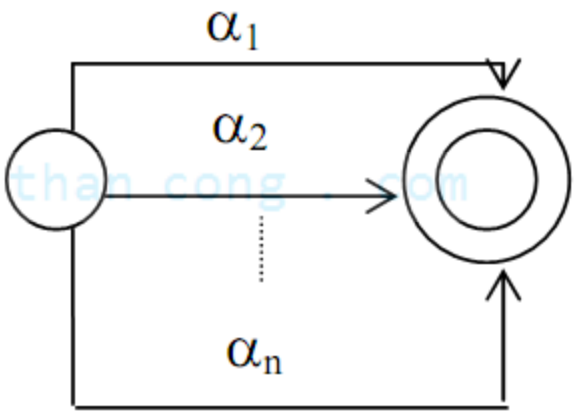
- Với bộ văn phạm đã tinh chỉnh (loại bỏ đệ quy, thêm yếu tố trái) => có thể sử dụng PTCP đệ quy đi xuống, **nhưng không cần quay lui,** gọi là phân tích cú pháp dự đoán.

3.2.1 Xây dựng sơ đồ dịch cho PTDD

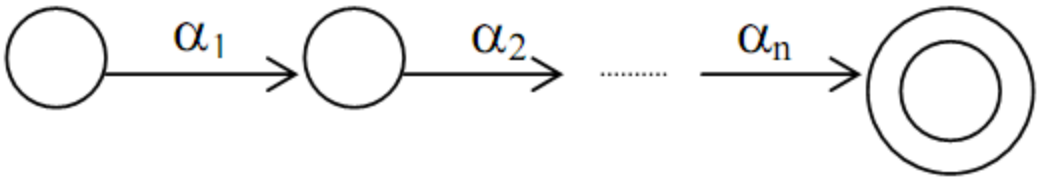
- Các bước xây dựng sơ đồ dịch cho mỗi ký tự chưa kết thúc A:
  - B1: Loại bỏ đệ quy trái, tạo yếu tố trái cho văn phạm.
  - B2: Tạo một trạng thái khởi đầu và một trạng thái kết thúc.
  - B3: Với mỗi luật sinh A → X1X2 ... Xn , tạo một đường đi từ trạng thái khởi đầu đến trạng thái kết thúc bằng các cạnh có nhãn X1X2 ... Xn

3.2.1 Xây dựng sơ đồ dịch cho PTDD

- Luật sinh có dạng $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ thì tương ứng sơ đồ dịch có dạng:



- Luật sinh có dạng $A \rightarrow \alpha_1 \ \alpha_2 \ldots \ldots \alpha_n$ thì tương ứng sơ đồ dịch có dạng:



3.2.1 Xây dựng sơ đồ dịch cho PTDD

- Ví dụ 4.2: Xét văn phạm cho bởi:

  E -> E + T |T

  T -> T*F|F

  F -> (E)|id

- Loại bỏ đệ quy trái trong văn phạm, ta được văn phạm tương đương:

$$E \rightarrow TE'$$

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

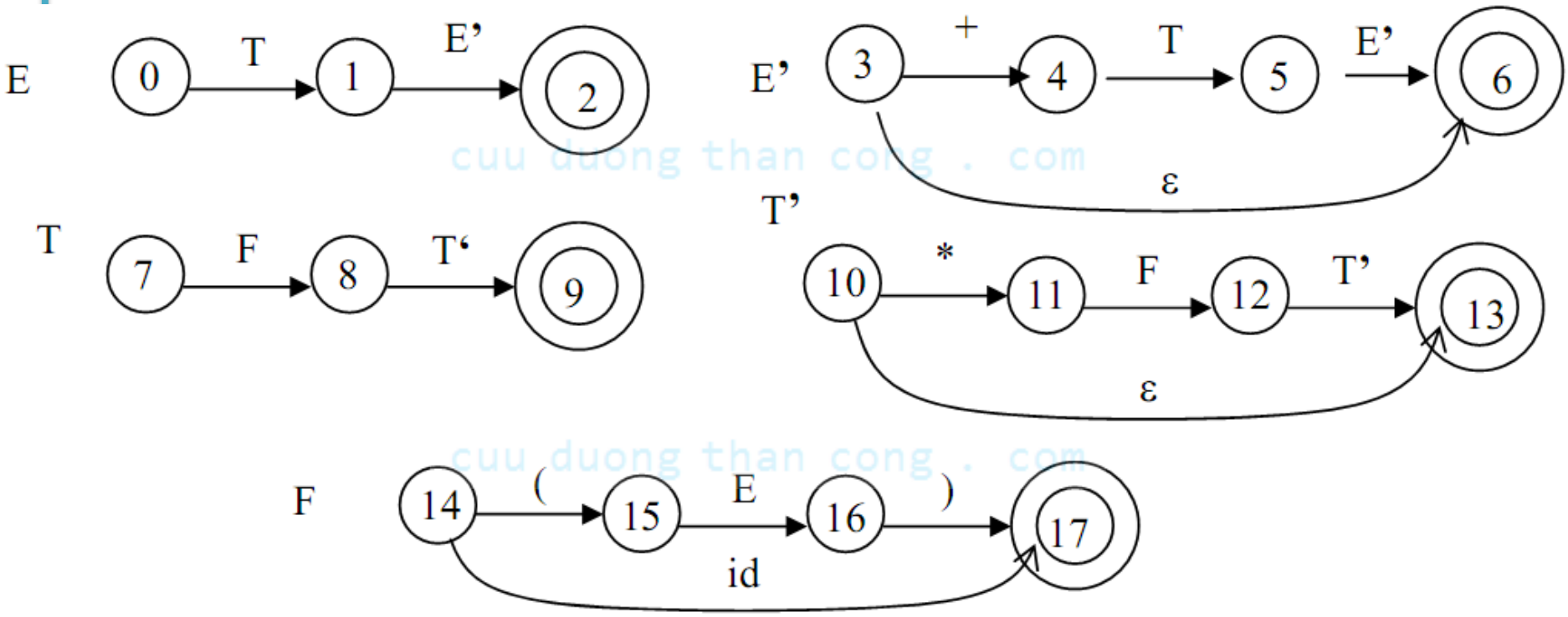
$$T \rightarrow FT'$$

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

$$F \rightarrow (E) \mid \mathbf{id}$$

3.2.1 Xây dựng sơ đồ dịch cho PTDD

- Các sơ đồ dịch tương ứng:



3.2.1 Xây dựng sơ đồ dịch cho PTDD

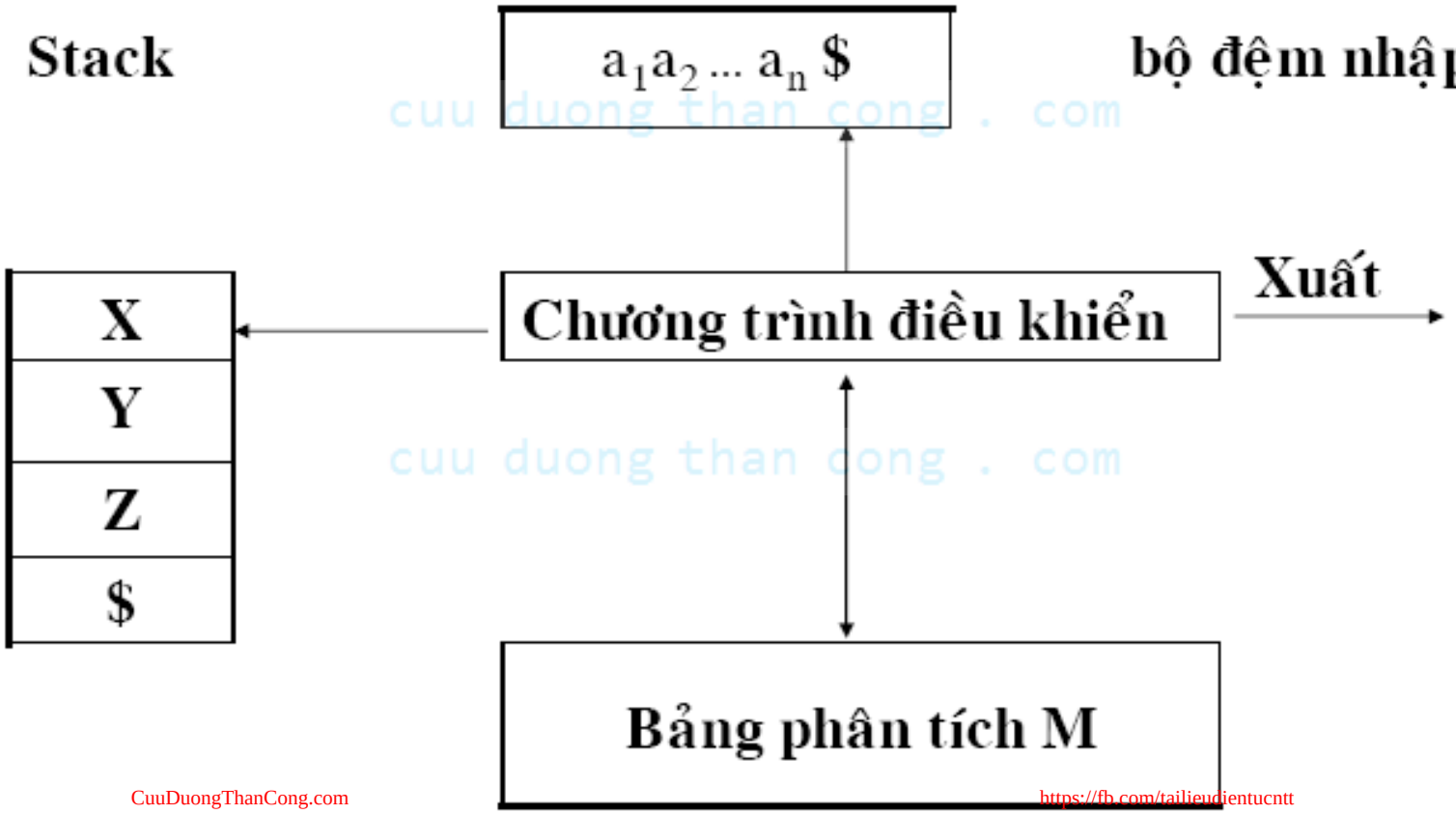
- Một chương trình PTCP dự đoán xây dựng dựa trên các sơ đồ dịch cho các ký hiệu chưa kết thúc.
- Duyệt chuỗi token:
  - So sánh ký hiệu chưa kết thúc với chuỗi token đầu vào.
  - Đưa ra lời gọi đệ quy mỗi khi đi theo cạnh có nhãn là ký hiệu chưa kết thúc.

3.2.2 Phân tích dự đoán không đệ quy

- Mục đích chính: xác định luật sinh được áp dụng cho bước tiếp theo.
- Giải pháp: Sử dụng một stack để thực hiện phân tích thay vì lời gọi đệ quy

3.2.2 Phân tích dự đoán không đệ quy

- Cấu tạo bộ phân tích dự đoán



3.2.2 Phân tích dự đoán không đệ quy

- **Input**: là bộ đệm chứa chuỗi token cần phân tích, kết thúc bởi ký hiệu $.
- **STACK** chứa một chuỗi các ký hiệu văn phạm với ký hiệu $ nằm ở đáy Stack.
- **Bảng phân tích 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 $. *(Mỗi văn phạm có một bảng phân tích M tương ứng)*

3.2.2 Phân tích dự đoán không đệ quy

- **Hoạt động:** Chương trình xét ký hiệu **X** trên đỉnh Stack và ký hiệu nhập hiện hành **a**:
  1. Nếu X = a = $ bộ phân tích dừng và báo thành công.
  2. Nếu X = a <> $ bộ phân tích sẽ đẩy X ra khỏi Stack và dịch đầu đọc đến ký hiệu nhập kế tiếp.
  3. Nếu X là ký hiệu không kết thúc bộ phân tích sẽ xét bảng ma trận tại **M[X,a]** để tìm luật sinh hoặt lỗi:
     a) Nếu M[X,a] là một luật sinh có dạng X → UVW thì Pop X ra khỏi đỉnh Stack và Push W, V, U vào Stack (với U trên đỉnh Stack), đồng thời bộ xuất sinh ra luật sinh X → UVW.
     b) Nếu M[X,a] = error, gọi chương trình phục hồi lỗi.

3.2.2 Phân tích dự đoán không đệ quy

Giải thuật 4.3:

- **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$.

3.2.2 Phân tích dự đoán không đệ quy

Giải thuật 4.3:

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 = $

3.2.2 Phân tích dự đoán không đệ quy

Ví dụ 4.3: Giả sử chúng ta có văn phạm G:

E -> E + T | T

T -> T *F | F

F -> (E) | id

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

E -> TE'

E'-> +TE' | ε

T -> FT'

T' ->*FT'| ε

F -> (E) | id

3.2.2 Phân tích dự đoán không đệ quy

- Bây giờ chúng ta sẽ phân tích cú pháp cho câu nhập **w = id + id * id** bằng bảng phân tích M cho trước *(cách xây dựng bảng ở phần sau)*

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)		

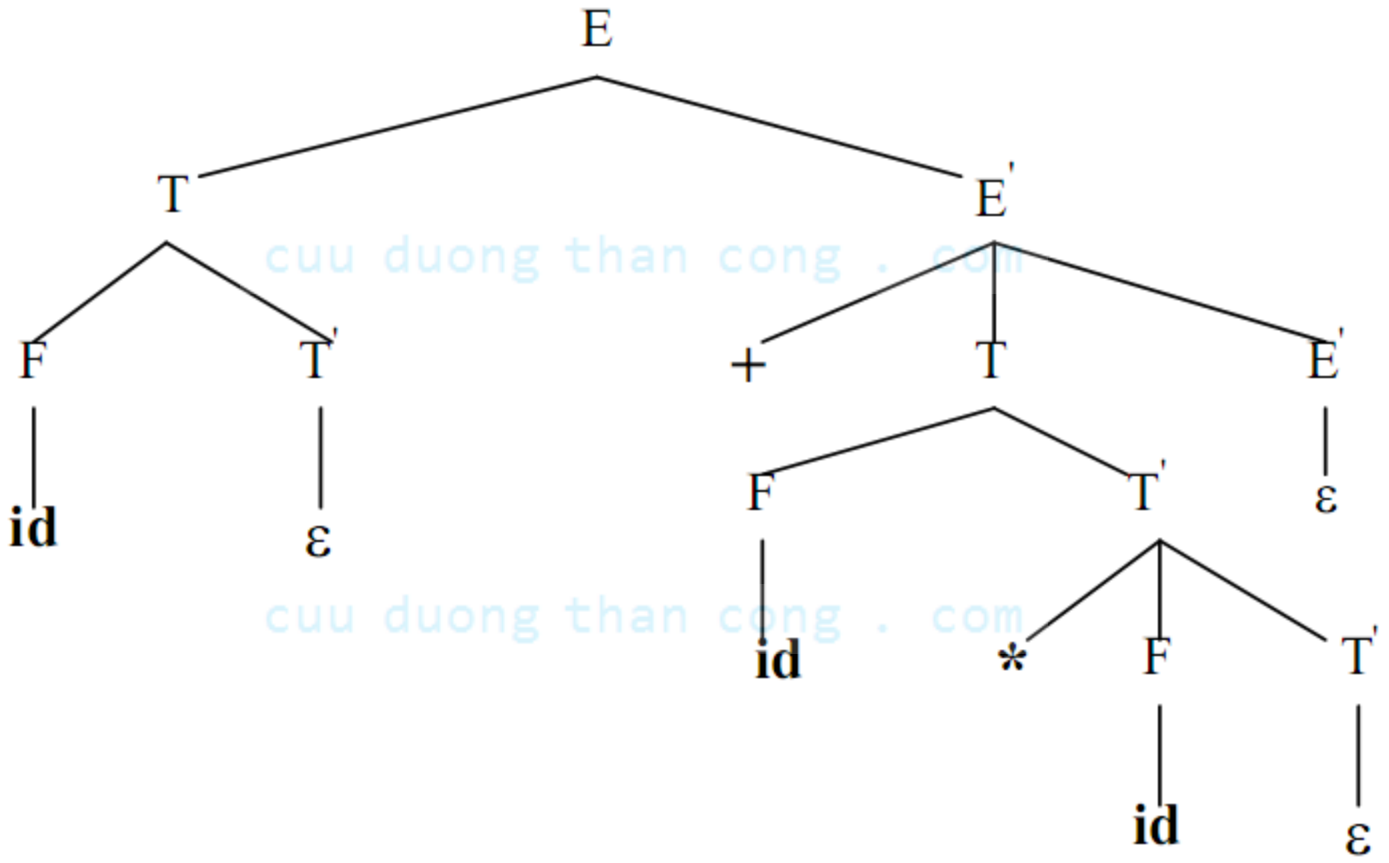
3.2.2 Phân tích dự đoán không đệ quy

=> Các bước phân tích cú pháp cho w:

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' → ∈

3.2.2 Phân tích dự đoán không đệ quy

=> Cây suy dẫn được xây dựng:



3.2.3 Xây dựng bảng phân tích M

- Xây dựng bảng M: dựa vào tập hợp **First** và **Follow.**
- Trong đó:
  - Định nghĩa **FIRST(α)**: Giả sử α là một chuỗi các ký hiệu văn phạm (bao gồm cả kết thúc và không kết thúc), FIRST(α) là tập hợp các ký hiệu kết thúc mà nó bắt đầu một chuỗi dẫn xuất từ α. Nếu $\alpha \Rightarrow^* \varepsilon$ thì $\varepsilon \in$ FIRST(α).
  - Định nghĩa **FOLLOW(A)**: (với A là một ký hiệu chưa kết thúc) là tập hợp các ký hiệu kết thúc a mà nó xuất hiện ngay sau A (bên phía phải của A) trong một dạng câu nào đó. Tức là tập hợp các ký hiệu kết thúc a, sao cho tồn tại một dẫn xuất dạng $S \Rightarrow^* \alpha A a \beta$. Chú ý rằng nếu A là ký hiệu phải nhất trong một dạng câu nào đó thì \$ ∈ FOLLOW(A) (\$ là ký hiệu kết thúc chuỗi nhập).

3.2.3 Xây dựng bảng phân tích M

Các quy tắc tính First(X) với X là ký hiệu văn phạm (*Thực hiện cho đến khi không còn có ký hiệu kết thúc nào hoặc ε có thể thêm vào tập FIRST(X)*):

- Nếu X là ký hiệu kết thúc thì first(X) = {X}
- Nếu X-> ε là luật sinh thì ta thêm ε vào first(X)
- Nếu X là ký hiệu không kết thúc và X ->Y1Y2Y3 ...Yk là một luật sinh thì:
  - Thêm tất cả các ký hiệu kết thúc khác ε của FIRST(Y1) vào FIRST(X).
  - Nếu ε ∈ FIRST(Y1) thì tiếp tục thêm vào FIRST(X) tất cả các ký hiệu kết thúc khác ε của FIRST(Y2).
  - Nếu ε ∈ FIRST(Y1) ∩ FIRST(Y2) thì thêm tất cả các ký hiệu kết thúc khác ε ∈ FIRST(Y3) ...
  - Cuối cùng thêm ε vào FIRST(X) nếu $\varepsilon \in \bigcap_{i=1}^{k} FIRST(Yi)$

3.2.3 Xây dựng bảng phân tích M

- **Ví dụ 4.4** xét văn phạm:

  E -> TE’

  E’-> +TE’ | ε

  T -> FT’

  T’ ->*FT’| ε

  F -> (E) | id

- Vì E' → ε ⇒ ε ∈ FIRST(E').

  Do E' → + TE' mà FIRST(+) = {+} ⇒ FIRST(E') = {+, ε }

- Tương tự FIRST(T') = {*, ε }
- Vì F ⇒ (E) | id ⇒ FIRST(F) = { (, id }
- Từ T → F T' và ε ∉ FIRST(F) ⇒ FIRST(T) = FIRST(F)= { (, id }
- Từ E → T E' và ε ∉ FIRST(T) ⇒ FIRST(E) = FIRST(T)= { (, id }

3.2.3 Xây dựng bảng phân tích M

- Tính FIRST(γ)? *(γ là một chuỗi)*
  - FIRST(X) ⊇ FIRST(γ) nếu X → γ
  - FIRST(aβ) = {a}
  - FIRST(Xβ) ⊇ FIRST(X)
  - FIRST(Xβ) ⊇ FIRST(β) nếu X-> ε
- **Thuật toán**: Giả sử với mọi γ, FIRST(γ) rỗng, áp dụng các luật trên liên tục để xây dựng các tập FIRST.

3.2.3 Xây dựng bảng phân tích M

Các quy tắc tính follow(A) cho tất cả các ký hiệu không kết thúc A *(Áp dụng các quy tắc sau cho đến khi không thể thêm gì vào mọi tập FOLLOW được nữa).*

1. Cho ký hiệu $ vào follow(S), S là ký hiệu bắt đầu văn phạm, $ là ký hiệu kết thúc chuỗi nhập.
2. Tồn tại luật A-> αBβ, tất cả các ký hiệu thuộc first(β) sẽ cho vào follow(B) trừ ε.
3. Tồn tại luật A-> αB hoặc A-> αBβ mà first(β) = {ε} thì tất cả các ký hiệu follow(A) sẽ cho vào follow(B).

$FOLLOW(B) \supseteq FIRST(\beta)$

$FOLLOW(B) \supseteq FOLLOW(A)$ nếu $\beta \rightarrow \varepsilon$

3.2.3 Xây dựng bảng phân tích M

- Với văn phạm ở **ví dụ 4.4**:

Áp dụng luật 2 cho luật sinh $F \rightarrow (E) \Rightarrow) \in FOLLOW(E) \Rightarrow$ FOLLOW(E) ={$,) }

Áp dụng luật 3 cho $E \rightarrow TE' \Rightarrow), \$ \in FOLLOW(E') \Rightarrow$ FOLLOW(E') ={$,) }

Áp dụng luật 2 cho $E \rightarrow TE' \Rightarrow + \in FOLLOW(T)$.

Áp dụng luật 3 cho $E' \rightarrow +TE'$, $E' \rightarrow \varepsilon$

$\Rightarrow FOLLOW(E') \subset FOLLOW(T) \Rightarrow$ FOLLOW(T) = { +,), $ }.

Áp dụng luật 3 cho $T \rightarrow FT'$ thì FOLLOW(T') = FOLLOW(T) ={+,), $ }

Áp dụng luật 2 cho $T \rightarrow FT' \Rightarrow * \in FOLLOW(F)$

Áp dụng luật 3 cho $T' \rightarrow * F T'$, $T' \rightarrow \varepsilon$

$\Rightarrow FOLLOW(T') \subset FOLLOW(F) \Rightarrow$ FOLLOW(F) = {*, +,), $ }.

3.2.3 Xây dựng bảng phân tích M

- Thuật toán xây dựng bảng M:
  - **Nhập**: Văn phạm G.
  - **Xuất**: Bảng phân tích M.
  - **Phương pháp**:

Đặt A → α:
- Dòng A, các cột FIRST(**α**)
- Dòng A, các cột FOLLOW(A) nếu **α->ε**

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 ε thuộc first(α), thêm A -> ε vào M[A, b] sao cho b thuộc follow(A). Nếu $ thuộc follow(A) thì thêm A -> ε vào M [A, $].
4. Những phần tử của bảng M trống, hãy đánh dấu lỗi.

3.2.3 Xây dựng bảng phân tích M

- Ví dụ xây dựng bảng M trong **ví dụ 4.3**:

Luật sinh E → TE' : Tính FIRST(TE') = FIRST(T) = {(,id}

⇒ M[E,id] và M[E,( ] chứa luật sinh E → TE'

Luật sinh E'→ + TE' : Tính FIRST(+TE') = FIRST(+) = {+}

⇒ M[E',+] chứa E' → +TE'

Luật sinh E' → ε : Vì ε ∈ FIRST(E') và FOLLOW(E') = {), $ }

⇒ E → ε nằm trong M[E',)] và M[E',$]

Luật sinh T'→ * FT' : FIRST(* FT') = {* }

⇒ T' → * FT' nằm trong M[T',*]

Luật sinh T' → ε: Vì ε ∈ FIRST(T') và FOLLOW(T') = {+,), $}

⇒ T' → ε nằm trong M[T', +] , M[T',)] và M[T',$]

Luật sinh F→ (E) ; FIRST((E)) = { (}

⇒ F → (E) nằm trong M[F, (]

Luật sinh F → id ; FIRST(id) = {id}

⇒ F → id nằm trong M[F, id]

3.2.3 Xây dựng bảng phân tích M

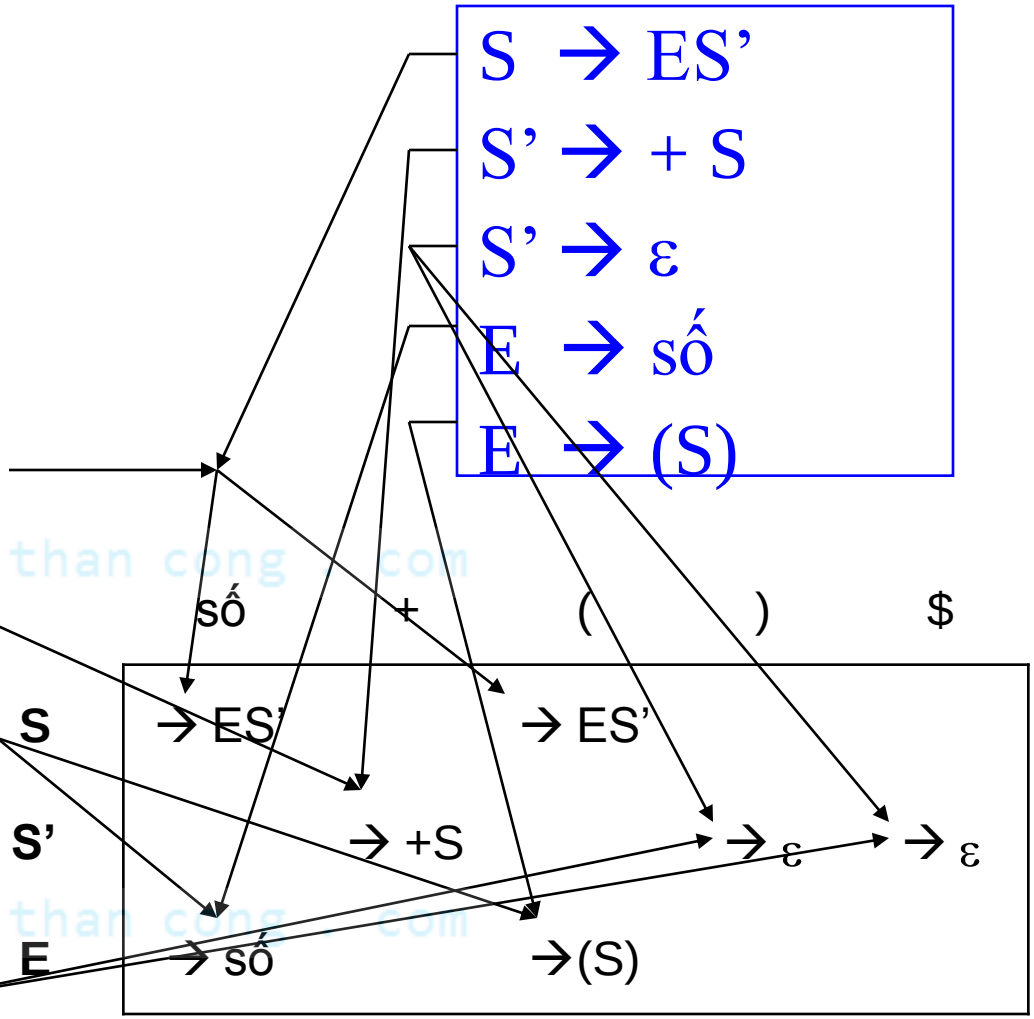
- **Bài tập**: Cho G:

S → ES’
S’ → + S
S’ → ε
E → số
E → (S)

Xây dựng bảng phân tích M?

Ví dụ

- Triệt tiêu được
  - Chỉ có S' triệt tiêu được
- FIRST
  - FIRST(E S') = {**số**, **(** }
  - FIRST(+S) = { **+** }
  - FIRST(**số**) = {**số**}
  - FIRST(**(**S**)**) = { **(** }
  - FIRST(S') = { + }
- FOLLOW
  - FOLLOW(S) = { **$**, **)** }
  - FOLLOW(S') = {**$**, **)**}
  - FOLLOW(E) = { **+**, **)**, **$**}



3.2.3 Văn phạm LL(1)

- Giải thuật tạo bảng M có thể áp dụng cho văn phạm G bất kỳ.
- Nếu G mơ hồ hoặc đệ quy trái => trong M có đa trị (nhiều luật sinh trong một ô)
- **Ví dụ 4.5:** cho văn phạm G

S -> iEtSS' | a

S'-> eS' | €

E -> b

first(S) = {i,a}, first(S') = {e,€}, first(E) = {b}

follow(S) = {e,$}, follow(S') = {e,$}, follow(E) = {t}

3.2.3 Văn phạm LL(1)

=> Bảng phân tích M cho **ví dụ 4.5**

Các ký hiệu không KT	*Ký hiệu nhập*					
	a	b	e	i	t	$
S	S → a			S → iEtSS'		
S'			S' → ∈ S' → eS			S' → ∈
E		E → b				

3.2.3 Văn phạm LL(1)

- **Ví dụ 4.5** *(tiếp)* Đây là một văn phạm mơ hồ và sự mơ hồ này được thể hiện qua việc chọn luật sinh khi gặp ký hiệu e. Ô tại vị trí M [S', e] được gọi là ô đa trị.
- 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)**:
  - L: Left-to-right parse (mô tả hành động quét chuỗi nhập từ trái sang phải)
  - L: Leftmost-derivation (biểu thị việc sinh ra một dẫn xuất trái cho chuỗi nhập)
  - 1: 1-symbol lookahead (tại mỗi một bước, đầu đọc chỉ đọc trước được một token để thực hiện các quyết định phân tích cú pháp)

3.2.3 Văn phạm LL(1)

- Tính chất của VP LL(1):
  - Không là VP đệ quy trái hay mơ hồ
  - Với các 2 luật sinh phân biệt: A ->α|β :
    1. Không có một ký hiệu kết thúc a nào mà cả α và β đều dẫn xuất ra các chuỗi bắt đầu bằng a.
    2. Tối đa chỉ có α hoặc chỉ có β có thể dẫn xuất ra chuỗi rỗng.
    3. Nếu $\beta \Rightarrow^* \varepsilon$ thì α không dẫn xuất được chuỗi nào bắt đầu bằng một ký hiệu kết thúc thuộc tập FOLLOW(A).

Biến Văn phạm mơ hồ/đệ quy => LL(1) ?

3.2.4 Khắc phục lỗi

- 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.
- Ta cho tất cả các ký hiệu trong follow(A) vào tập token đồng bộ của A (**synch**). Chúng ta làm như vậy cho mỗi ký hiệu không kết thúc A.

3.2.4 Khắc phục lỗi

- Ví dụ 4.5: Xét VP G:

  E -> TE’

  E’-> +TE’ | ε

  T -> FT’

  T’ ->*FT’| ε

  F -> (E) | id

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

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

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

3.2.4 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*

Bảng phân tích cú pháp M phục hồi lỗi

3.2.4 Khắc phục lỗi

- Bảng này được sử dụng như sau:
  - Nếu M[A,a] là rỗng thì bỏ qua token a.
  - Nếu M[A,a] là "synch" thì lấy A ra khỏi Stack nhằm tái hoạt động quá trình phân tích.
  - Nếu một token trên đỉnh Stack không phù hợp với token trong dòng nhập thì lấy token ra khỏi Stack.
- Với chuỗi nhập: + id * + id, bộ phân tích cú pháp và cơ chế phục hồi lỗi thực hiện:

3.2.4 Khắc phục lỗi

STACK	INPUT	OUTPUT
$ E	+ id * + id $	**error, nhảy qua +**
$ E	id * + id $	E → T E'
$ E' T	id * + id $	T → F T'
$ E' T' F	id * + id $	F → id
$ E' T' id	id * + id $	
$ E' T'	* + id $	T' → * F T'
$ E' T' F *	* + id $	
$ E' T' F	+ id $	**error**, M[F,+] = **synch** pop F
$ E' T'	+ id $	T → ε
$ E'	+ id $	E' → + T E'
$ E' T +	+ id $	
$ E' T	id $	T' → F T'
$ E' T' F	id $	F→ id
$ E' T' id	id $	T' → ε
$ E' T'	$	E' → ε
$ E'	$	
$	$	

3.2.4 Khắc phục lỗi

- **Bài tập:** Phân tích và khắc phục lỗi cho chuỗi nhập:

  W =)id*+id

Kiểm tra

Bài 1

- Cho văn phạm G chứa các luật sinh sau:

  S → xAB

  A → Ayz | y

  B → t

1. Khử đệ quy trái
2. Xây dựng bảng phân tích M
3. Sử dụng bộ phân tích cú pháp trên để xây dựng cây suy dẫn *(chỉ rõ các bước)*:

xyyzt

Bài 2

- Cho Văn phạm G chứa các luật sinh:

  E → E op E | (E) | num

  op → + | * | ^

1. Khử đệ quy trái
2. Xây dựng bảng phân tích M
3. Sử dụng bộ phân tích cú pháp trên để xây dựng cây suy dẫn *(chỉ rõ các bước)*:

2 ^ 3 + 4 * 5

Kiểm tra (2)

Bài 3:

Xét ngôn ngữ sử dụng các thẻ (tags) được mô tả như sau:

- Ký hiệu kết thúc: { < , > , / , = , **word** }
- Mỗi thẻ bắt đầu bằng < và kết thúc bằng >
- Có hai loại thẻ: thẻ mở và thẻ đóng
- Thẻ mở có dạng <**word word** = **word** … >, tức là bắt đầu bằng **word**, tiếp theo là các cặp **word** được nối với nhau bằng dấu =, thể hiện các thuộc tính của thẻ.
- Thẻ đóng có dạng </**word**>
- Mỗi thẻ mở phải có một thẻ đóng tương ứng phía sau. Giữa cặp thẻ mở và đóng đó có thể có dãy các **word** dài tuỳ ý.
- Ví dụ: xâu <**word word=word word=word**><**word**>**word word word**</**word**><**word**></**word**></**word**> thuộc ngôn ngữ trên.

a, Hãy viết văn phạm phi ngữ cảnh cho ngôn ngữ trên .

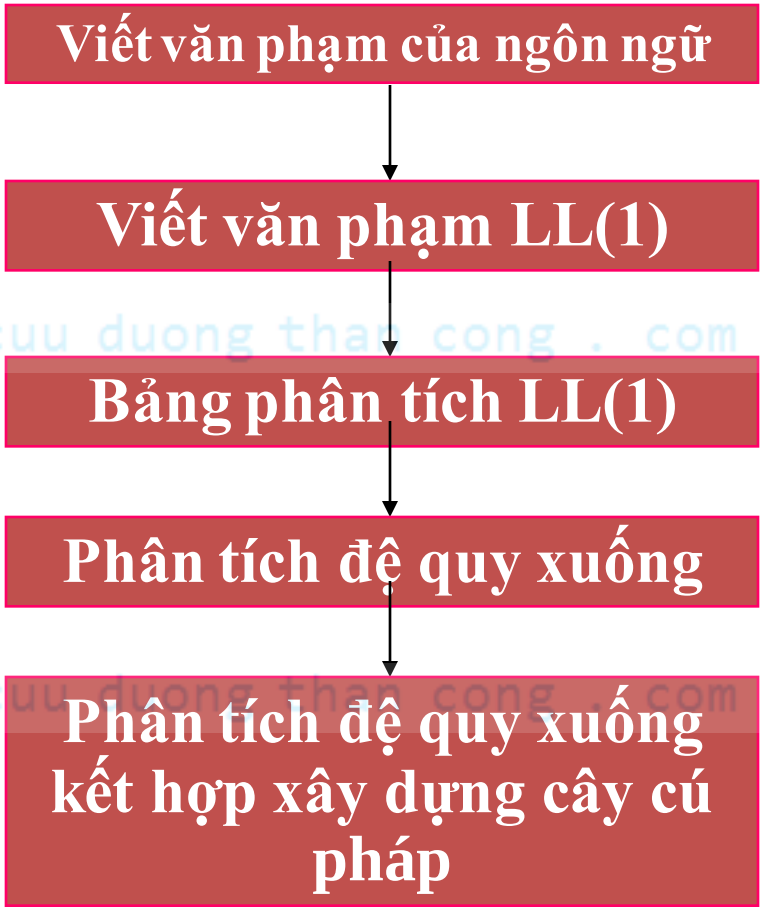
b, Tìm các kí hiệu có thể triệt tiêu được.

c, Tính các tập FIRST, FOLLOW cho văn phạm trên.

d, Lập bảng phân tích M, chỉ ra những vị trí xung đột trên bảng.

e, Hãy chỉ ra nguyên nhân khiến văn phạm này không phải là LL(1).

Xây dựng bộ PTCP trên xuống



Nội dung

1. Vai trò của bộ phân tích cú pháp (PTCP)
2. Văn phạm của ngôn ngữ lập trình
3. Phân tích cú pháp từ trên xuống
4. **Phân tích cú pháp từ dưới lên**
5. Bộ sinh bộ PTCP

Tổng kết Bài 4

- Các kiến thức cần nhớ:
- Về nhà đọc thêm:

cuu duong than cong . com

Bài học phần sau

Bài 5

Thảo luận