# Kí hiệu đặc biệt

Anpha: α

Beta: β

Epsilon ε

Sigma: Σ

Delta: Δ

Gamma: γ

Rỗng: Ø

𝑉

→

Thuộc: ∈

𝑉**\***

𝑉**+**

α → β

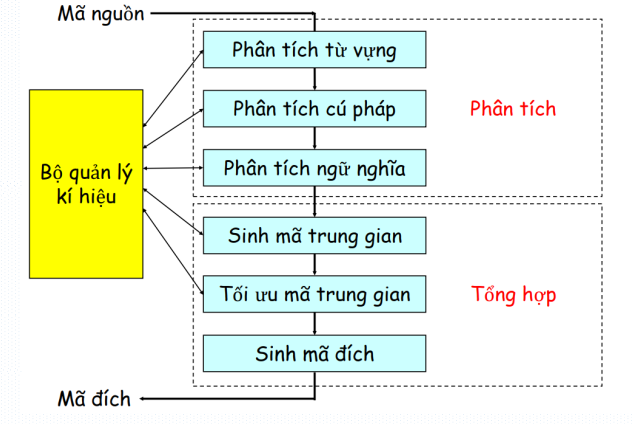
α ∈ 𝑉**+**

|α| <= |β|

Hợp: ∪

# Phần 1: Tổng quan CCD

Cấu trúc chi tiết của một chương trình dịch



# Phần 2: Văn Phạm

## Định nghĩa

G = (Σ, Δ, P, S) trong đó:

* Σ là tập hữu hạn các ký hiệu, gọi là ký hiệu kết thúc (terminal)
* Δ là tập hữu hạn các ký hiệu, gọi là ký hiệu không kết thúc (nonterminal)
* S gọi là ký hiệu khởi đầu.
* P là tập hữu hạn các cặp xâu (α, β) được gọi là các dẫn xuất hay luật cú pháp (α → β)

## Quy ước

1. Dùng các chữ in hoa A, B, C, D, E... hoặc cụm từ trong cặp ngoặc nhọn (như <chủ ngữ>): ký hiệu không kết thúc;
2. Dùng các chữ thường a, b, c, d, e... và các con số, các phép toán +, -, \*, /, cặp ngoặc đơn để trỏ các ký hiệu kết thúc.
3. Dùng các chữ in hoa X, Y, Z để trỏ các ký hiệu có thể là kết thúc hoặc không kết thúc;
4. Dùng các chữ thường u, v, w, x, y, z để trỏ các xâu ký hiệu cuối;
5. Dùng các chữ thường Hy lạp α, β, χ để trỏ các xâu gồm các biến và ký hiệu cuối;
6. Nếu có các sản xuất cùng vế trái A → α và A → β thì ta viết gộp lại cho gọn thành A → α | β . Các sản xuất có cùng một kí hiệu không kết thúc vế trái có thể gọi chung bằng tên kí hiệu vế trái. Ví dụ, sản xuất-A.

𝑉 = (Σ ∪ Δ) là một xâu (có thể rỗng) bao gồm cả ký hiệu không kết thúc và

ký hiệu kết thúc;

𝑉**\*** là tập tất cả các xâu V có thể có;

𝑉**+** cũng như vậy trừ xâu rỗng;

| | là ký hiệu độ dài xâu (ví dụ | α | là độ dài của xâu α);

Ký hiệu ε là một kí hiệu đặc biệt, chỉ xâu rỗng hoặc kí hiệu rỗng

## Phân loại văn phạm Chomsky

| Lớp | Dạng | Điều kiện | Đoán nhận |
| --- | --- | --- | --- |
| Lớp 0 | α → β | α ∈ 𝑉**+** , β ∈ 𝑉**\*** | Máy turing |
| Lớp 1 (Cảm ngữ cảnh) | α → β | |α| <= |β| | Automat tuyến tính giới nội |
| Lớp 2 (Phi ngữ cảnh NNPNC) | A → α | A ∈ Δ, α ∈ 𝑉**\*** | Automat đẩy xuống |
| Lớp 3 (Chính quy - NNCQ) | A → a, A → Ba  A → a, A → aB | A, B ∈ Δ, a ∈ Σ | Automat hữu hạn |

## Văn phạm đệ quy

Ví dụ: Ta có A → αAβ

G là văn phạm đệ quy trái nếu α → ε

G là văn phạm đệ quy phải nếu β → ε

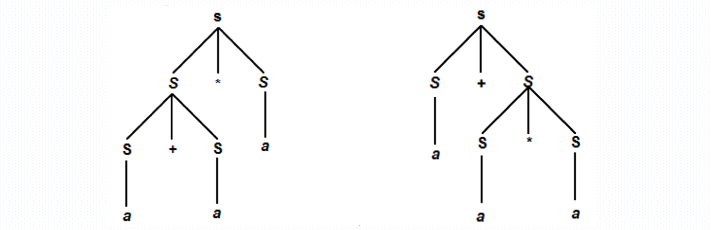
## Văn phạm nhập nhằng

Định nghĩa: Một văn phạm bị coi là nhập nhằng (ambiguity) nếu tồn tại chuỗi w có ít nhất hai cây phân tích tạo ra nó. Ngược lại, văn phạm không có nhập nhằng là văn phạm đơn nghĩa.

Ví dụ: Xét văn phạm sau: 𝑆 → 𝑆 + 𝑆 | 𝑆 ∗ 𝑆 | ( 𝑆 ) | 𝑎

Xây dựng cây phân tích của chuỗi 𝑎 + 𝑎 ∗ 𝑎

Ta có 2 cây phân tích, dẫn đến việc có 2 cách hiểu ngữ nghĩa của chuỗi (nếu thay a bằng số thì có 2 cách tính giá trị của chuỗi)



## Automat hữu hạn đơn định (DFA) và không đơn định (NFA)

Automat hữu hạn đơn định (Deterministic Finite Automata - DFA): Với 1 ký hiệu đầu vào, chỉ có thể chuyển sang tối đa một trạng thái tiếp theo (hoặc dừng và báo lỗi). Không chấp nhận kí hiệu ε.

Automat hữu hạn không đơn định (Nondeterministic Finite Automata - NFA): Với 1 ký hiệu đầu vào có thể chuyển sang nhiều trạng thái tiếp theo. Chấp nhận kí hiệu đầu vào là ε.

*DFA có thể coi là một trường hợp đặc biệt của NFA. DFA tương đương NFA về khả năng đoán nhận ngôn ngữ.*

## Biểu thức chính quy

| Kí hiệu | Nghĩa | Ví dụ |
| --- | --- | --- |
| | , ∪ | Hoặc | A ∪ B = A | B |
| + | Lặp lại 1 hoặc nhiều lần | a+ |
| \* | Lặp lại 0 hoặc nhiều lần | a+ |
| ? | Lặp lại 0 hoặc 1 lần | a? |
| ( ) | Để nhóm các thành phần | (a) |
| (A | B)\* | Dãy bất kì được tạo bởi A hoặc B, hoặc là dãy rỗng | (chữ cái | chữ số)\* |

# Phần 3: Phân tích từ vựng

## Nhiệm vụ PTTV

1. Đọc chương trình nguồn, loại bỏ các ký hiệu vô ích (khoảng trắng, dấu tab, xuống dòng, ghi chú,…).
2. Phát hiện một số lỗi cơ bản về từ vựng.
3. Xác định nội dung của từ vựng.
4. Xác định từ loại của từ vựng đó.
5. Đưa ra một số thông tin thuộc tính của từ vựng.

## Các bước PTTV

1. Xóa bỏ các ký tự không có nghĩa: chú thích, dòng trống, các ký tự xuống dòng, dấu tab, các khoảng trắng không cần thiết.
2. Nhận dạng các ký hiệu: các kí tự liền nhau tạo thành một kí hiệu. Các dạng kí hiệu gọi là các từ tố.
3. Số hoá ký hiệu.

Ví dụ

Đầu vào: position := initial + rate \* 60;

Đầu ra:

<tên, con trỏ đến position trên bảng ký hiệu>

<phép\_gán , >

<tên, con trỏ đến initial trên bảng ký hiệu>

<toán\_tử\_cộng , >

<tên, con trỏ đến rate trên bảng ký hiệu>

<toán\_tử\_nhân, >

<số, giá trị số nguyên 60>

<chấm\_phẩy, >

## Xác định từ tố

### Biển diễn từ tố

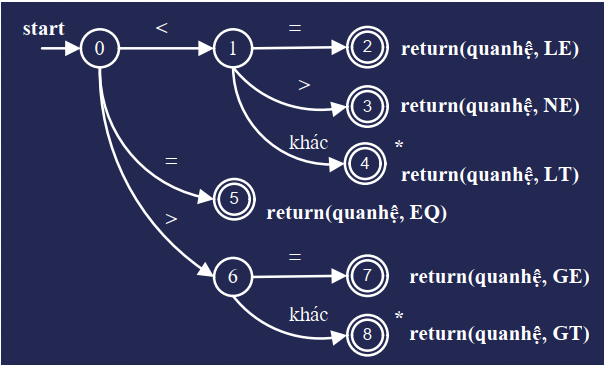
chữ cái → A | B | ... | Z | a | b | ... | z

chữ số → 0 | 1 | ... | 9

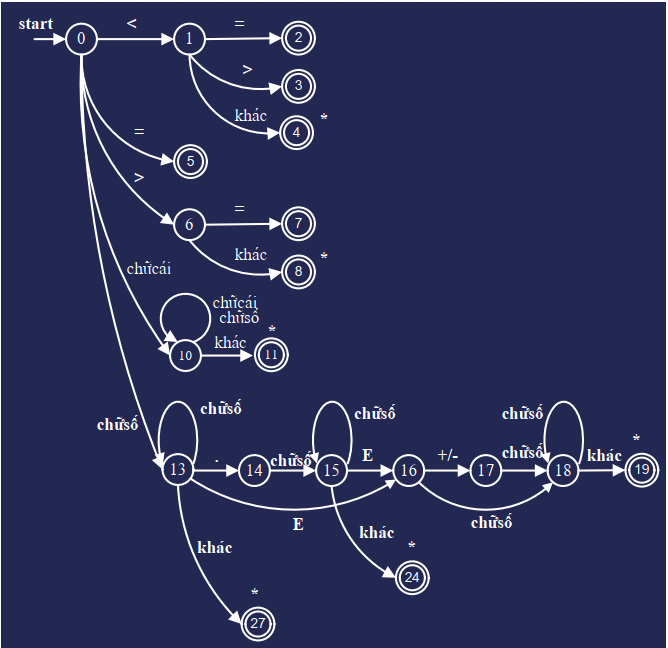
tên → chữcái ( chữcái | chữsố )\*

quan hệ → < | <= | = | <> | > | >=

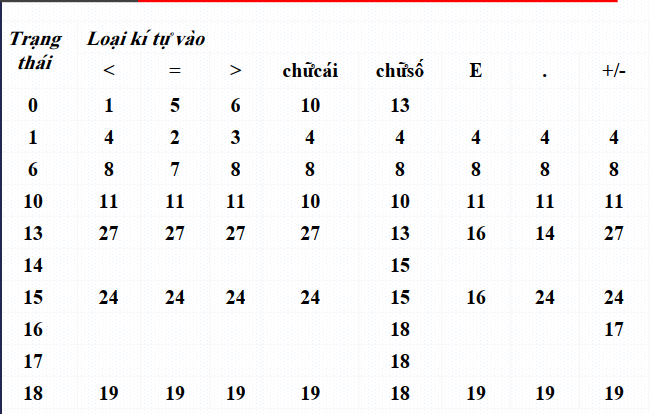
### Đồ thị chuyển



### Kết hợp các đồ thị chuyển



### Lập bộ phân tích từ vựng điều khiển bằng bảng



### Code

Dùng biểu thức chính quy xây dựng các khái niệm chữ số(số nguyên, số thập phân có

dấu) trong ngôn ngữ pascal? Xây dựng đồ thị chuyển đoán một từ tố có là chữ số hay

không? Sử dụng ngôn ngữ C/C++ viết một chương trình đơn giản?

#include <stdio.h>

#include <string.h>

#include <cctype>

#include <iostream>

struct trangThai {

int output\_chuso, output\_khac, output\_congtru, output\_cham, output\_e;

};

int ktKytu(char c) {

if (c >= '0' && c <= '9') {

return 0; // So la 0

} else if (c == 'E')

return 1; // E la 1

else if (c == '.')

return 2;

else if (c == '+' || c == '-')

return 3;

else

return 4;

}

using namespace std;

int main() {

struct trangThai s[12];

s[0].output\_chuso = 2;

s[0].output\_e = 11;

s[0].output\_cham = 11;

s[0].output\_congtru = 1;

s[0].output\_khac = 11;

s[1].output\_chuso = 2;

s[1].output\_e = 11;

s[1].output\_cham = 11;

s[1].output\_congtru = 1;

s[1].output\_khac = 11;

s[2].output\_chuso = 2;

s[2].output\_e = 5;

s[2].output\_cham = 3;

s[2].output\_congtru = 10;

s[2].output\_khac = 10;

s[3].output\_chuso = 4;

s[3].output\_e = 10;

s[3].output\_cham = 10;

s[3].output\_congtru = 10;

s[3].output\_khac = 10;

s[4].output\_chuso = 4;

s[4].output\_e = 5;

s[4].output\_cham = 9;

s[4].output\_congtru = 9;

s[4].output\_khac = 9;

s[5].output\_chuso = 7;

s[5].output\_e = 9;

s[5].output\_cham = 9;

s[5].output\_congtru = 6;

s[5].output\_khac = 9;

s[6].output\_chuso = 7;

s[6].output\_e = 9;

s[6].output\_cham = 9;

s[6].output\_congtru = 9;

s[6].output\_khac = 9;

s[7].output\_chuso = 7;

s[7].output\_e = 8;

s[7].output\_cham = 8;

s[7].output\_congtru = 8;

s[7].output\_khac = 8;

char token[100];

printf("Nhap vao 1 day\n");

scanf("%s", &token);

char t[10] = "+";

strcat(token, t);

int state = 0;

for (int i = 0; i < strlen(token); i++) {

int kt = ktKytu(token[i]);

switch (kt) {

case 0:

state = s[state].output\_chuso;

break;

case 1:

state = s[state].output\_e;

break;

case 2:

state = s[state].output\_cham;

break;

case 3:

state = s[state].output\_congtru;

break;

case 4:

state = s[state].output\_khac;

break;

}

if (state == 8) {

printf("so thuc mu\n");

// return 0;

}

if (state == 9) {

printf("so thuc \n");

// return 0;

}

if (state == 10) {

printf("so nguyen\n");

// return 0;

}

if (state == 11) {

printf("Loi cu phap\n");

return 0;

}

}

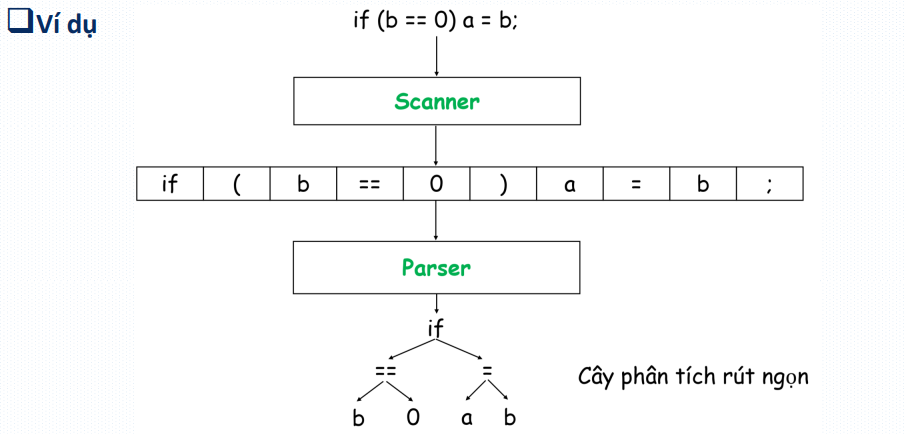
}

# Phần 4: Phân tích cú pháp

## Nhiệm vụ

1. Xây dựng cây cú pháp cho chuỗi vào
2. Thực hiện một số thao tác ngữ nghĩa phục vụ cho việc phân tích tiếp theo
3. Phát hiện các lỗi về văn phạm và lựa chọn phương pháp xử lý phù hợp
4. Xử lý lỗi để tiếp tục thực hiện việc phân tích
5. Đưa ra các gợi ý sửa lỗi cho mã nguồn

## Đầu vào, đầu ra của PTCP



## Các bước xây dựng bộ PTCP

1. Xây dựng bộ PTCP trực tiếp (dành cho các ngôn ngữ có độ phức tạp đơn giản)
2. Xây dựng bộ PTCP 2 bước

* Dựa trên văn phạm đầu vào, xây dựng automat đoán nhận
* Sử dụng automat để xử lý dãy từ tố từ PTTV

1. Xây dựng bộ PTCP vạn năng: trường hợp văn phạm quá phức tạp, có thể sử dụng các phương pháp phân tích vạn năng để xây dựng bộ PTCP

## Suy dẫn và biểu thức suy dẫn bằng cấu trúc cây

Nếu 𝛼1 ⇒ 𝛼2 ⇒ … ⇒ 𝛼𝑛 ta nói 𝛼1 suy dẫn ra 𝛼𝑛

Hệ thống kí hiệu

1. ⇒ Suy dẫn 1 bước
2. ⇒**\*** Suy dẫn ra 0 hoặc nhiều bước
3. ⇒**+** Suy dẫn ra 1 hoặc nhiều bước

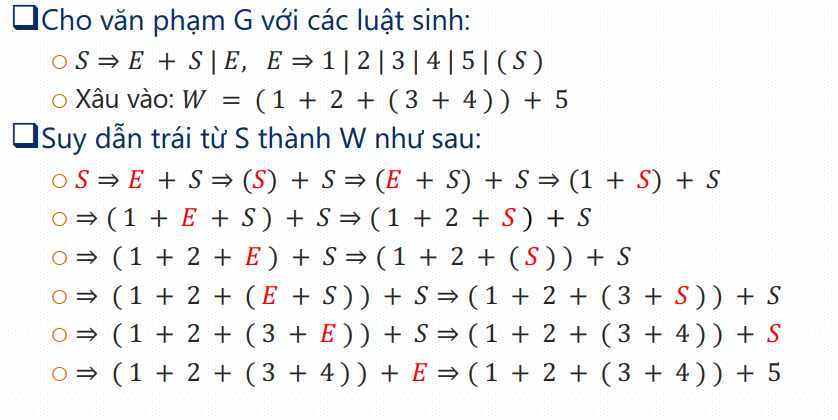
Một số tính chất

1. 𝛼 ⇒**\*** 𝛼 với ∀α
2. 𝛼 ⇒**\*** 𝛽 và 𝛽 ⇒**\*** 𝛾 thì 𝛼 ⇒**\*** 𝛾

Bài toán phân tích cú pháp thực chất là bài toán tìm chuỗi suy dẫn 𝑆 ⇒**\*** 𝛼 ⇒**\*** 𝛽, trong đó:

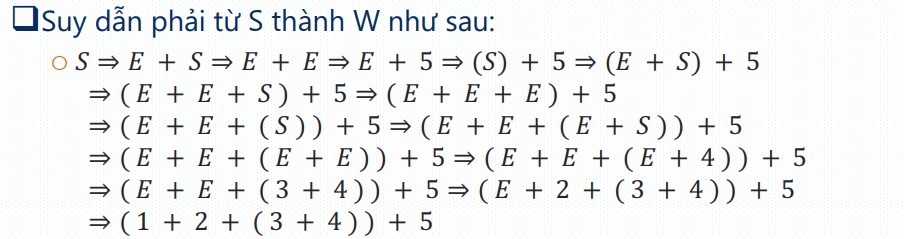
* S là kí hiệu gốc
* α là chuỗi có chứa kí hiệu trung gian
* β là chuỗi chỉ gồm các kí hiệu kết thúc

### Suy dẫn trái

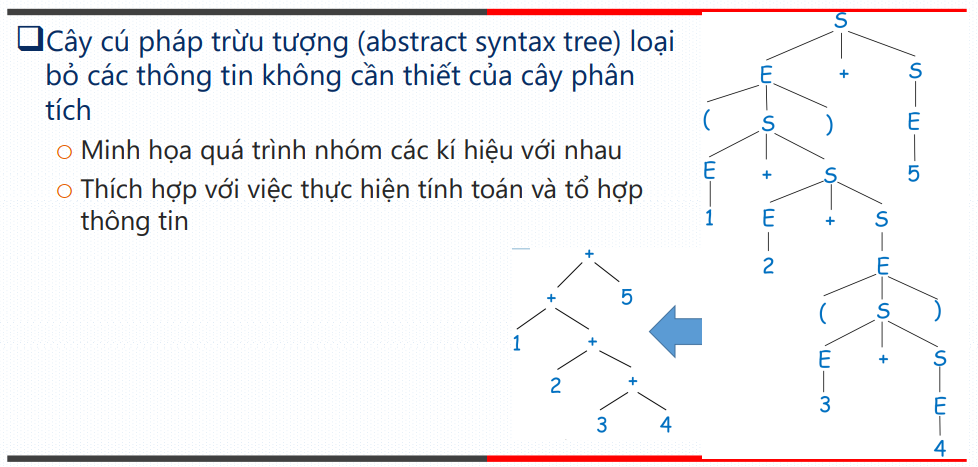


Giải thích: Nhìn theo những kí tự in màu đỏ, suy dẫn trái là chọn luật đúng nhất và áp dụng cho các ký tự từ trái qua phải

### Suy dẫn phải

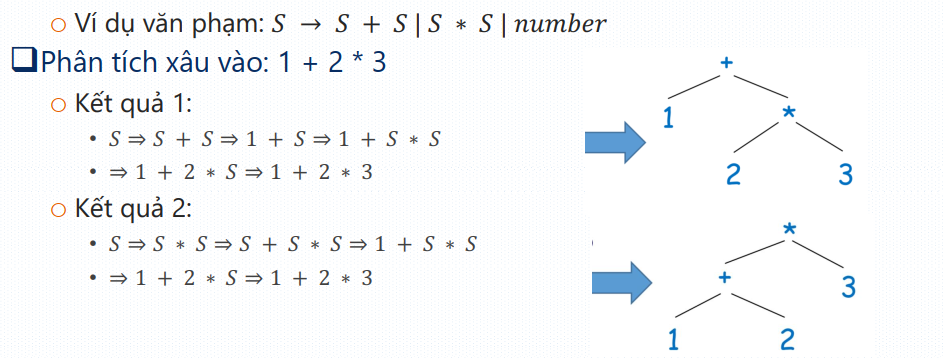


### Cây PTCP và cây PTCP trừu tượng



## Văn phạm có tính nhập nhằng

### Định nghĩa

Văn phạm tồn tại ít nhất một chuỗi w có từ 2 cây phân tích tương ứng trở lên gọi là văn phạm có nhập nhằng

### Khử nhập nhằng

1. Thêm vào các biến trung gian
2. Đưa ra các ràng buộc ngoài văn phạm (ví dụ như quy định mức độ ưu tiên của các phép toán)

Ví dụ văn phạm: 𝑆 → 𝑆 + 𝑆 | 𝑆 ∗ 𝑆 | 𝑛𝑢𝑚𝑏𝑒𝑟

Khử nhập nhằng bằng cách thêm biến trung gian:

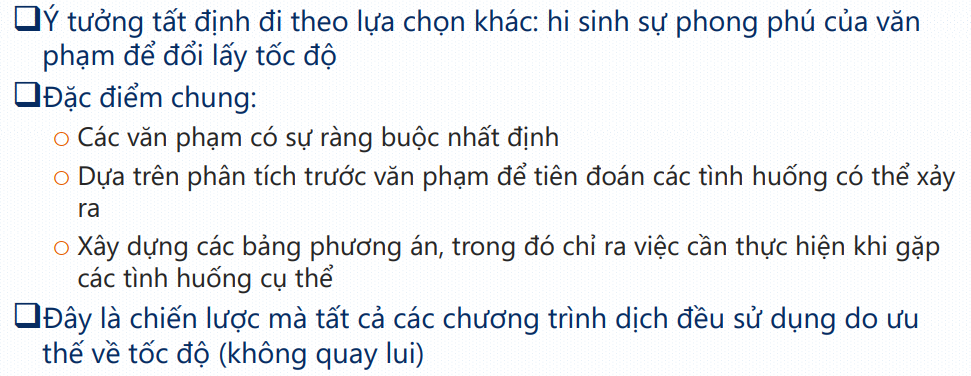
* 𝑆 → 𝑆 + 𝑇 | 𝑇
* 𝑇 → 𝑇 ∗ 𝑛𝑢𝑚𝑏𝑒𝑟 | 𝑛𝑢𝑚𝑏𝑒𝑟

## Các chiến lược PTCP

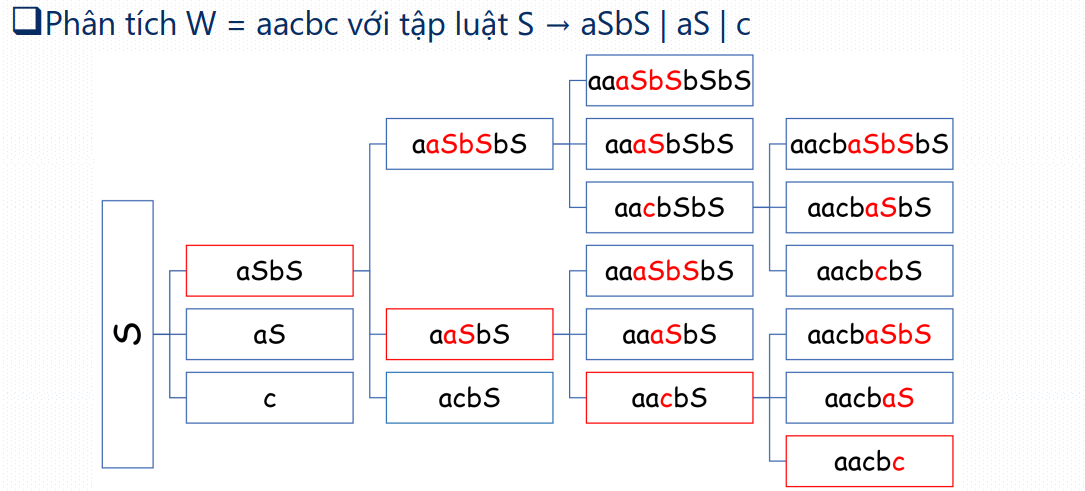
### Chiến lược thử-sai (quay lui): top-down, bottom-up

### Chiến lược quy hoạch động: CYK, Earley,…

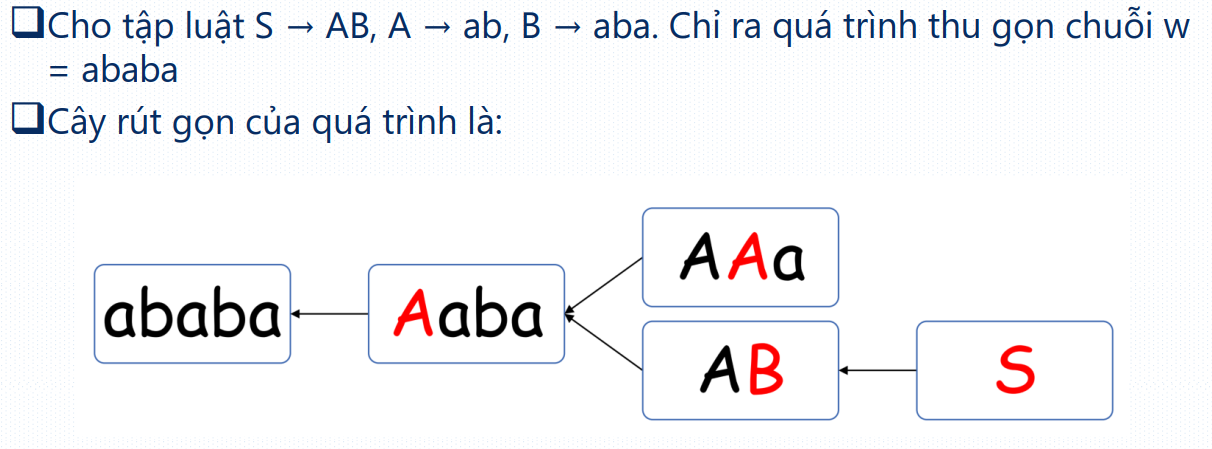
### Chiến lược tất định (deterministic): LL, LR,…



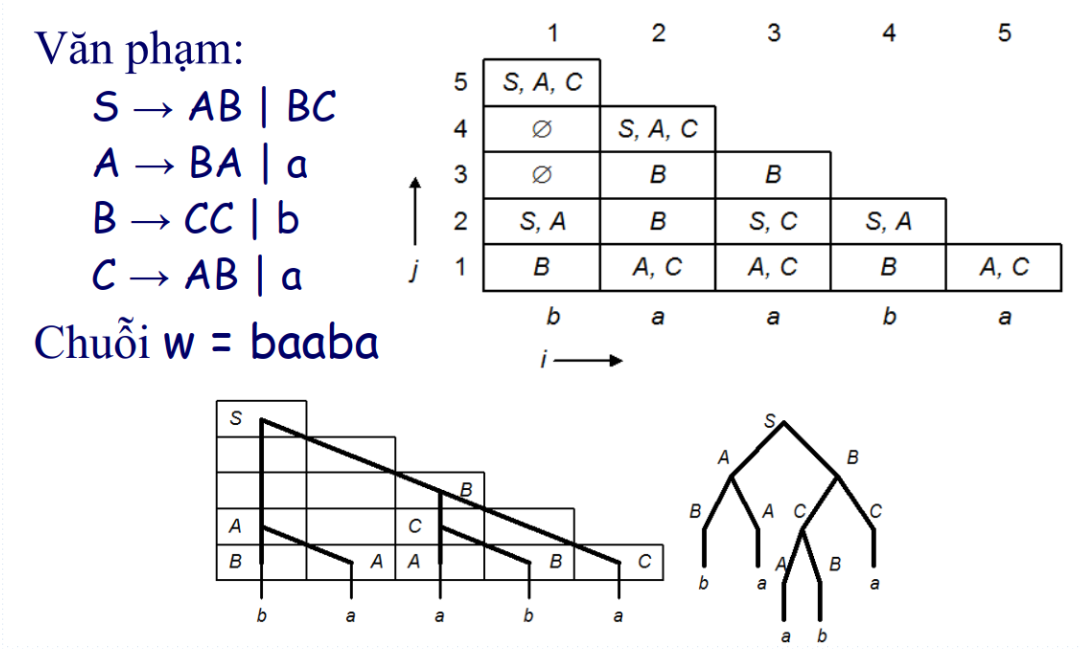
### Thuật toán top-down



### Thuật toán bottom-up



### Thuật toán CYK



Giải thích:

B1: Xét từng ký tự đơn b, a, a, b, a (Độ dài 1)

B2: Xét các chuỗi ba, aa, ab, ba (Độ dài 2)

Ta có tương ứng:

ba: BA & BC, => (S,A) do luật 1 và luật 2

aa: AA & AC & CA & CC, => (B) do luật 3

…

B3: Xét các chuỗi baa, aab, aba (Độ dài 3)

Ta lại có tương ứng:

baa → ba+a hoặc b+aa: SA & SC & AA & AC => Ø

…

B4, B5 làm tương tự

### Thuật toán Earley

### Thuật toán LL(1)

LL(1) nghĩa là gì? Viết tắt của “Left-to-right parse, Leftmost-derivation,

1-symbol lookahead".

Ký hiệu k trong LL(k) nghĩa là bộ phân tích sẽ nhìn trước k kí hiệu khi ra quyết định.

#### Ví dụ

Xét văn phạm G:

E → T E'

E' → + T E' | ε

T → F T'

T' → \* F T' | ε

F → (E) | id

Phân tích cú pháp cho chuỗi nhập: id + id \* id

#### Tính First

❑ Định nghĩa 𝐹𝐼𝑅𝑆𝑇(𝛼): giả sử α là một chuỗi các ký hiệu văn phạm,

𝐹𝐼𝑅𝑆𝑇(𝛼) là tập hợp các ký hiệu kết thúc mà nó bắt đầu một chuỗi dẫn xuất từ 𝛼:

* Giả sử 𝛼 = 𝑋1𝑋2 … 𝑋n
* Thêm vào 𝐹𝐼𝑅𝑆𝑇(𝛼): 𝐹𝐼𝑅𝑆𝑇(𝑋1) − {𝜀}
* Với mọi i= 2,3, … , 𝑛; nếu 𝐹𝐼𝑅𝑆𝑇(𝑋𝑘) chứa ε với mọi 𝑘 = 1,2 … , 𝑖 − 1 thì thêm vào 𝐹𝐼𝑅𝑆𝑇(𝛼): 𝐹𝐼𝑅𝑆𝑇(𝑋𝑖) − {𝜀}
* Nếu với mọi 𝑖 = 1,2, … 𝑛; nếu 𝐹𝐼𝑅𝑆𝑇(𝑋𝑖) chứa 𝜀 thì thêm 𝜀 vào 𝐹𝐼𝑅𝑆𝑇(𝛼)

Theo định nghĩa tập FIRST, ta có:

Vì F ⇒ (E) | id ⇒ FIRST(F) = { (, id }

Từ T → F T' và ε ∉ FIRST(F) ⇒ FIRST(T) = FIRST(F)

Từ E → T E' và ε ∉ FIRST(T) ⇒ FIRST(E) = FIRST(T)

Vì E' → ε ⇒ ε ∈ FIRST(E')

Mặt khác do E' → + TE' mà FIRST(+) = {+} ⇒ FIRST(E') = {+, ε }

Tương tự FIRST(T') = {\*, ε }

Vậy ta có :

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

FIRST(E') = {+, ε }

FIRST(T') = {\*, ε }

#### Tính Follow

❑ Tính FOLLOW (A): áp dụng các quy tắc sau cho đến khi không thể thêm gì vào mọi tập FOLLOW được nữa:

1. Ðặt $ vào follow(S), trong đó S là ký hiệu bắt đầu của văn phạm và $ là ký hiệu kết thúc chuỗi nhập.
2. Nếu có một luật sinh 𝐴 → 𝛼𝐵𝛽 thì thêm mọi phần tử khác ε của 𝐹𝐼𝑅𝑆𝑇 𝛽 vào trong 𝐹𝑂𝐿𝐿𝑂𝑊(𝐵).
3. Nếu có luật sinh 𝐴 → 𝛼𝐵 hoặc 𝐴 → 𝛼𝐵𝛽 mà 𝜀 ∈ 𝐹𝐼𝑅𝑆𝑇(𝛽) thì thêm tất cả các phần tử trong FOLLOW(A) vào FOLLOW(B).

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)

* Áp dụng luật 2 cho luật sinh F→ (E) ⇒ ) ∈ FOLLOW(E) ⇒ FOLLOW(E) ={$, ) }
* Áp dụng luật 3 cho E → TE' ⇒ ), $ ∈ FOLLOW(E') ⇒ FOLLOW(E') ={$, ) }
* Áp dụng luật 2 cho E → TE' ⇒ + ∈ FOLLOW(T).
* Áp dụng luật 3 cho E' → +TE' , E' → ε
* ⇒ FOLLOW(E') ⊂ FOLLOW(T) ⇒ FOLLOW(T) = { +, ), $ }.
* Áp dụng luật 3 cho T→ FT' thì FOLLOW(T') = FOLLOW(T) ={+, ), $ }
* Áp dụng luật 2 cho T→ FT' ⇒ \* ∈ FOLLOW(F)
* Áp dụng luật 3 cho T' → \* F T' , T’→ ε
* ⇒ FOLLOW(T') ⊂ FOLLOW(F) ⇒ FOLLOW(F) = {\*, +, ), $ }.
* Vậy ta có: FOLLOW(E) = FOLLOW(E') = { $, ) }
* FOLLOW(T) = FOLLOW(T') = { +, ), $ }
* FOLLOW(F) = {\*,+, ), $ }

#### Xây dựng bảng PTCP

Bảng phân tích cú pháp 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 $.

Thuật toán xây dựng bảng PTCP:

1. Với mỗi luật sinh A→ α của văn phạm, thực hiện:

* Với mỗi ký hiệu kết thúc a ∈ FIRST(α), thêm A→ α vào M[A,a]
* Nếu ε ∈ FIRST(α) thì đưa luật sinh A→ α vào M[A,b] với mỗi ký hiệu kết thúc b ∈ FOLLOW(A)
* Nếu ε ∈ FIRST(α) và $ ∈ FOLLOW(A) thì đưa luật sinh A→ α vào M[A,$]

2. Các ô trống trong bảng tương ứng với lỗi (error)

*Chú ý:* một ô trong bảng có thể chứa nhiều suy dẫn, tình huống này gọi là bảng

có nhập nhằng.

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(ε) và FOLLOW(T') = {+, ), $}

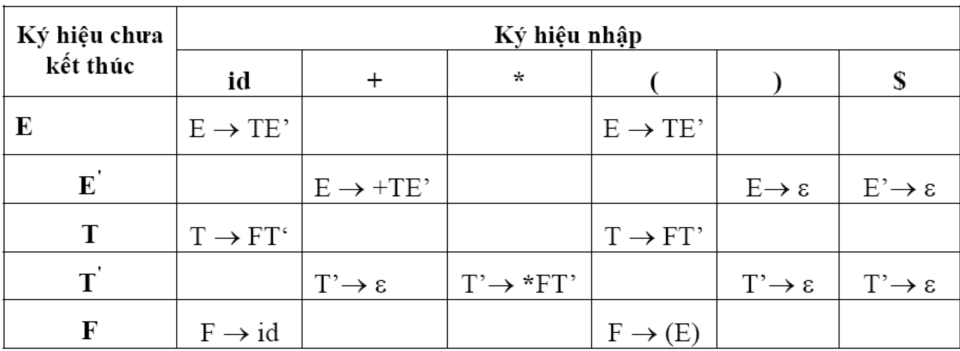
⇒ T' → ε nằm trong M[T', +] , M[T', )] và M[T',$]

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]



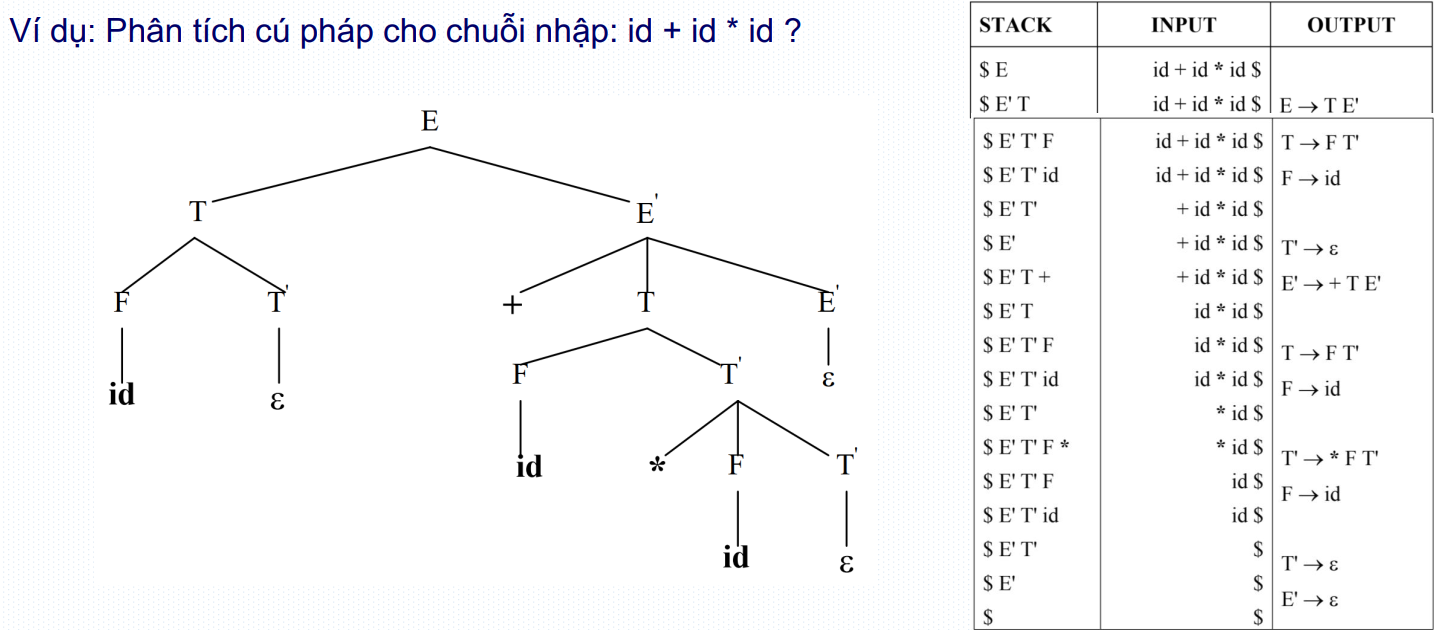
#### Xây dựng cây PTCP

❑ Xét ký hiệu X trên đỉnh Stack và ký hiệu nhập hiện hành a.

❑ Bộ phân tích cú pháp hoạt động như sau:

1. Nếu X = a = $ thì chương trình phân tích cú pháp kết thúc thành công.
2. Nếu X = a ≠ $, *pop X* ra khỏi Stack và *đọc ký hiệu Input* tiếp theo.
3. Nếu X là ký hiệu chưa kết thúc thì chương trình truy xuất đến phần tử M[X,a] trong bảng phân tích M:

* Nếu M[X,a] là 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 thêm X → UVW vào Output.
* Nếu M[X,a] = error, gọi chương trình phục hồi lỗi.



### Thuật toán LR

❑Có 03 phương pháp xây dựng một bảng PTCP:

* LR đơn giản (SLR)
* LR chính tắc
* LALR (Lookahead-LR)

#### Ví dụ phương pháp SLR

Xét văn phạm G:

E → E + T | T

T → T \* F | F

F → (E) | id

Phân tích cú pháp cho chuỗi nhập: id \* id + id

#### Mục (Item)

Cho một văn phạm G, mục LR(0) văn phạm là một luật sinh của G

với một dấu chấm mục tại vị trí nào đó trong vế phải.

Ví dụ: Luật sinh A → XYZ có 4 mục như sau :

* A → •XYZ
* A → X• YZ
* A → XY• Z
* A → XYZ•

Luật sinh A → ε chỉ tạo ra một mục A → •

#### Văn phạm tăng cường (Augmented Grammar)

Giả sử G là một văn phạm với ký hiệu bắt đầu S, ta thêm một ký hiệu bắt đầu mới S' và luật sinh S' → S để được văn phạm mới G' gọi là văn phạm tăng cường.

Xét văn phạm tăng cường của biểu thức, ta được:

E' → E

E → E + T | T

T → T \* F | F

F → (E) | id

#### 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 theo quy tắc sau:

* Tất cả các mục của I được thêm cho closure(I).
* Nếu 𝐴 → 𝛼 • 𝐵𝛽 ∈ 𝑐𝑙𝑜𝑠𝑢𝑟𝑒(𝐼) và B → γ là một luật sinh thì thêm B → • γ vào

𝑐𝑙𝑜𝑠𝑢𝑟𝑒(𝐼) nếu nó chưa có trong đó.

* Lặp lại bước này cho đến khi không thể thêm vào closure(I) được nữa.

Nếu I là tập hợp chỉ gồm văn phạm { E'→ • E } thì closure(I) bao gồm:

E' → • E (Luật 1)

E → • E + T (Luật 2)

E → • T (Luật 2)

T → • T \* F (Luật 2)

T → • F (Luật 2)

F → • (E) (Luật 2)

F → • id (Luật 2)

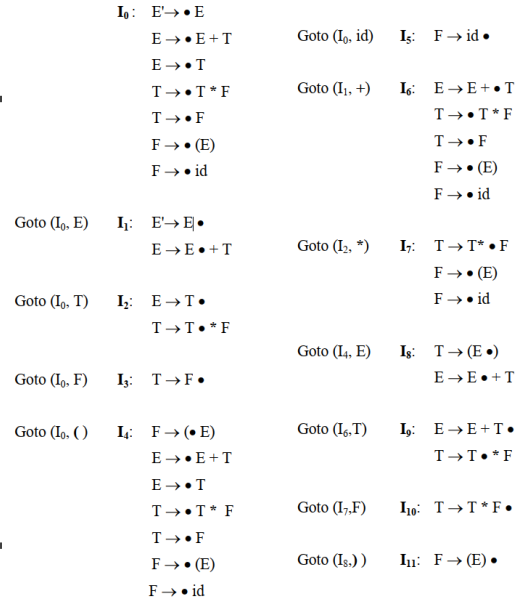
#### Phép toán Goto

* Goto(I, X), trong đó I là một tập các mục và X là một ký hiệu văn phạm là bao đóng của tập hợp các mục A → αX•β sao cho A → α•Xβ ∈ I.
* Cách tính goto(I, X):

1. Tạo một tập I' = ∅.
2. Nếu A → α•Xβ ∈ I thì đưa A→ αX•β vào I', tiếp tục quá trình này cho đến khi xét hết tập I.
3. Goto(I, X) = closure(I')

Giả sử I = { E' → E•, E → E • + T }. Tính goto (I, +) ?

* Ta có I' = { E→ E + • T }
* ( goto (I, +) = closure(I') bao gồm các
* mục :
* E → E + • T (Luật 1)
* T → • T \* F (Luật 2)
* T → • F (Luật 2)
* F → • (E) (Luật 2)
* F → • id (Luật 2)



#### Xây dựng bảng phân tích SLR

❑Thuật toán xây dựng bảng phân tích SLR

Input: Một văn phạm tăng cường G'

Output: Bảng phân tích SLR với hàm action và goto

Phương pháp:

1. Xây dựng C = { I0, I1, ..., In }, họ tập hợp các mục LR(0) của G'.
2. Trạng thái i được xây dựng từ Ii .Các action tương ứng trạng thái i được xác định như sau:

* Nếu A → α • aβ ∈ Ii và goto (Ii, a) = Ij thì action[i, a] = "shift j". Ở đây a là ký hiệu kết thúc.
* Nếu A → α• ∈ Ii thì action[i, a] = "reduce (A → α)", ∀a ∈ FOLLOW(A). Ở đây A không phải là S'
* Nếu S' → S• ∈ Ii thì action[i, $] = "accept".

Nếu một action đụng độ được sinh ra bởi các luật trên, ta nói văn phạm không phải là SLR(1). Giải thuật sinh ra bộ phân tích cú pháp sẽ thất bại trong trường hợp này.

1. Với mọi ký hiệu chưa kết thúc A, nếu goto (Ii,A) = Ij thì goto [i, A] = j
2. Tất cả các ô không xác định được bởi 2 và 3 đều là “error”
3. Trạng thái khởi đầu của bộ phân tích cú pháp được xây dựng từ tập các mục chứa S’→ • S

Ví dụ: Ta xây dựng bảng phân tích cú pháp SLR cho văn phạm tăng cường G’

E' → E

E → E + T | T

T → T \* F | F

F → (E) | id

(0) E'→ E

(1) E → E + T

(2) E → T

(3) T → T \* F

(4) T → F

(5) F → (E)

(6) F → id

1. C = { I0, I1, ... I11 }

2. FOLLOW(E) = {+, ), $}

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

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

Dựa vào họ tập hợp mục C đã được xây dựng trong ví trước

Trước tiên xét tập mục I0 : Mục F → • (E) cho ra action[0, (] = "shift 4", và mục F → • id cho action[0, id] = "shift 5".

Các mục khác trong I0 không sinh được hành động nào.

Bây giờ xét I1 : Mục E'→ E • cho action[1, $] = "accept", mục E → E • + T

cho action[1, +] = "shift 6".

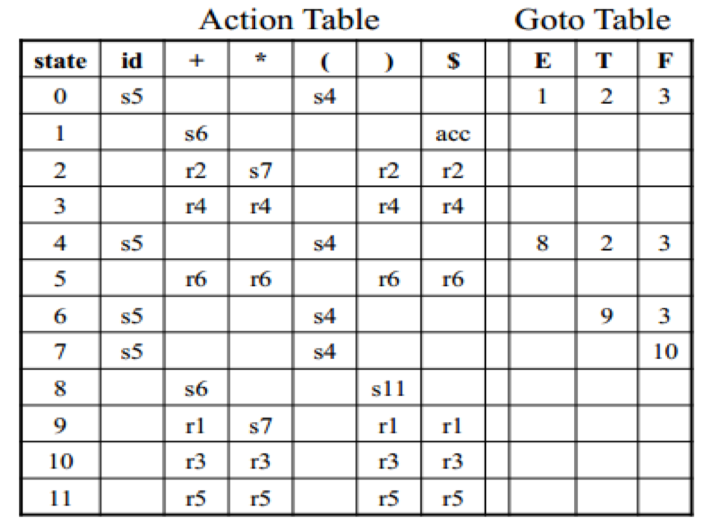
Kế đến xét I2 :

E → T •

T → T • \* F

Vì FOLLOW(E) = {+, ), $}, mục đầu tiên làm cho action[2, )] =action[2, $] = action[2,+] ="reduce (E → T)". Mục thứ hai làm cho action[2,\*] = "shift 7"

…



#### Máy phân tích cú pháp LR

1. Nếu action[𝒔𝒎, 𝒂𝒊] = Shift s: Thực hiện phép đẩy để được cấu hình mới:

(𝑠0𝑋1𝑠1𝑋2𝑠2 … 𝑋𝑚𝑠𝑚 𝑎𝑖𝑠, 𝑎𝑖+1 … 𝑎𝑛 $)

Phép đẩy làm cho s nằm trên đỉnh Stack, 𝑎𝑖+1 trở thành ký hiệu hiện hành.

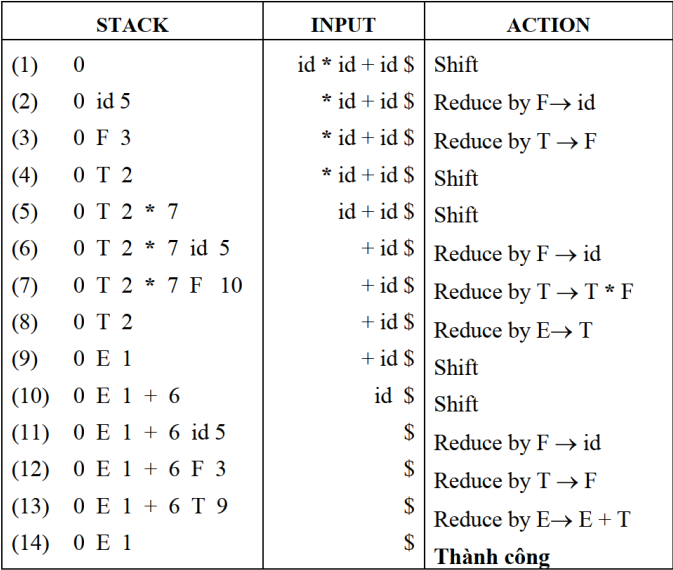
1. Nếu action [𝒔𝒎, 𝒂𝒊] = Reduce(A → β) thì thực hiện phép thu gọn để được cấu

hình: (𝑠0𝑋1𝑠1𝑋2𝑠2 … 𝑋𝑚−𝑖 𝑠𝑚−𝑖 𝐴𝑠, 𝑎𝑖 𝑎𝑖+1 … . 𝑎𝑛 $)

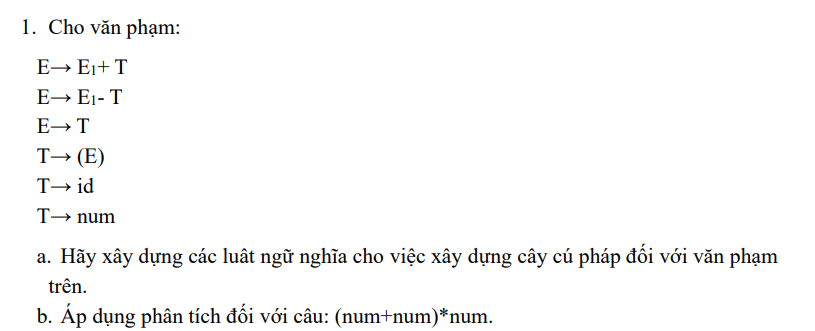
Trong đó, s = goto[𝑠𝑚−𝑖, 𝐴] và r là chiều dài số lượng các ký hiệu của β. Ở đây,

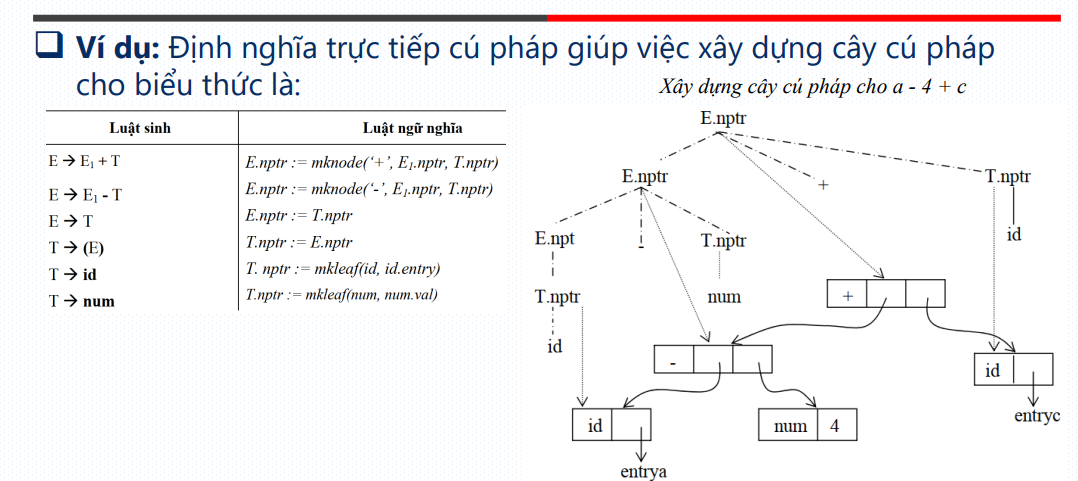
trước hết 2r phần tử của Stack sẽ bị lấy ra, sau đó đẩy vào A và s.

1. Nếu action[𝒔𝒎, 𝒂𝒊] = accept: quá trình phân tích kết thúc.
2. Nếu action[𝒔𝒎, 𝒂𝒊] = error: gọi thủ tục phục hồi lỗi.



# Phần 5: Biên dịch dựa trên cú pháp





Phần 6: