

NGÔN NGỮ và PHƯƠNG PHÁP DỊCH

Phạm Đăng Hải

haipd@soict.hust.edu.vn

Chương 1: Những khái niệm cơ bản

1. Ngôn ngữ lập trình cấp cao và trình dịch
2. Đặc trưng của ngôn ngữ lập trình cấp cao
3. Các giai đoạn chính của chương trình dịch
4. Khái niệm ngôn ngữ
5. Văn phạm phi ngữ cảnh
6. Giới thiệu ngôn ngữ PL/0 mở rộng

Sự cần thiết của ngôn ngữ lập trình bậc cao



- Nhiều loại máy tính
 - Mỗi loại nhiều kiểu
 - Mỗi kiểu có ngôn ngữ máy riêng
 - Ngôn ngữ máy là dãy nhị phân
 - Dùng ngôn ngữ máy
 - Không phải dịch
 - Phức tạp
 - Không khả chuyển
- Cần ngôn ngữ
 - Độc lập với máy
 - Gần với ngữ tự nhiên
- Ví dụ: C, Pascal, basic..

Ngôn ngữ
bậc cao

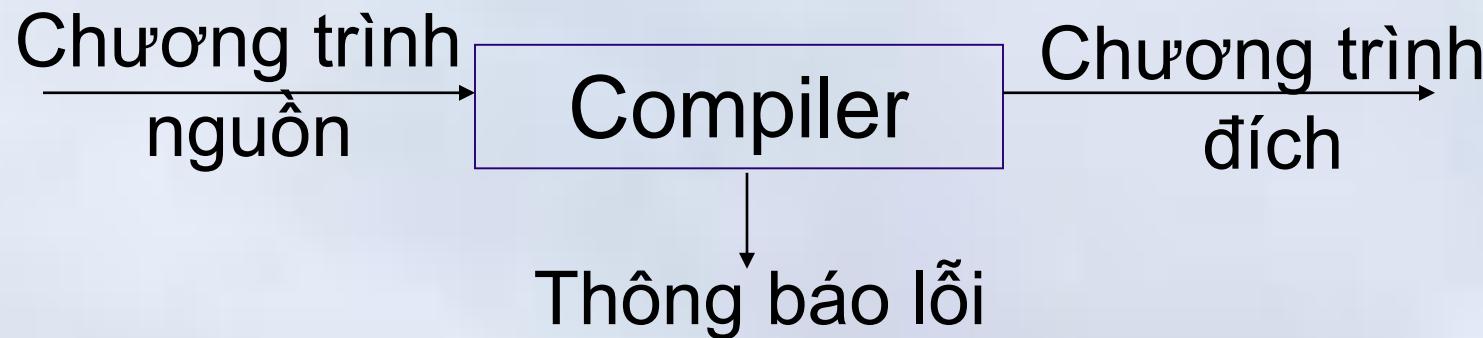
Ngôn ngữ lập trình cấp cao (NNLTCC)

Chương trình viết bằng NNLTCC

- Độc lập với máy tính
- Gần với ngôn ngữ tự nhiên
- Chương trình dễ đọc, viết và bảo trì
- Muốn thực hiện phải chuyển sang ngôn ngữ
 - Máy hiểu được (*ngôn ngữ máy*)
 - Ngôn ngữ trung gian mà máy hiểu được
Được chuyển đổi bởi **Chương trình dịch**
- Chương trình thực hiện chậm hơn

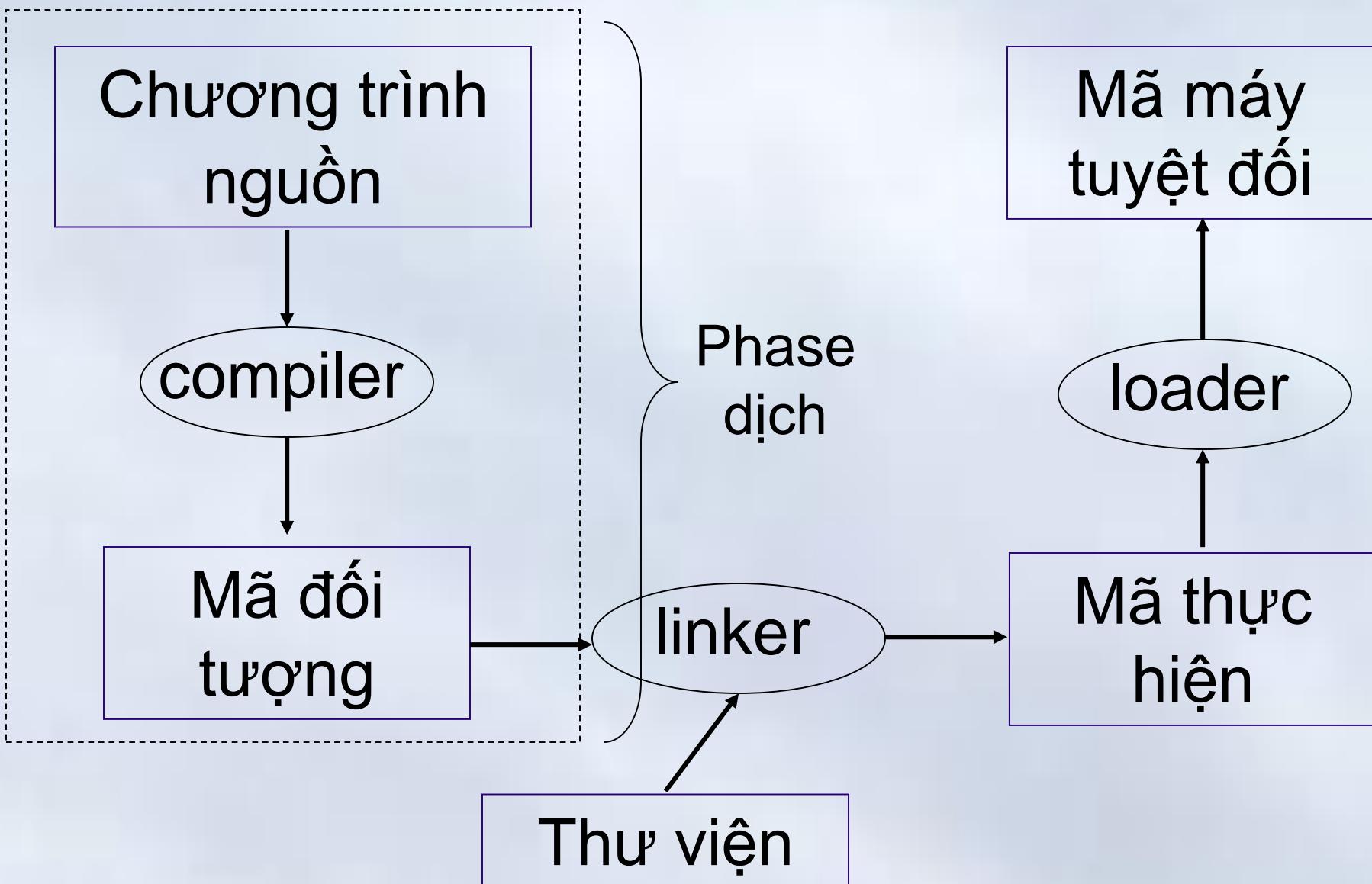
Chương trình biên dịch (compiler)

- Chương trình dịch làm nhiệm vụ dịch chương trình nguồn (*thường được viết bằng ngôn ngữ lập trình bậc cao*) sang các chương trình đối tượng (*chương trình đích*)

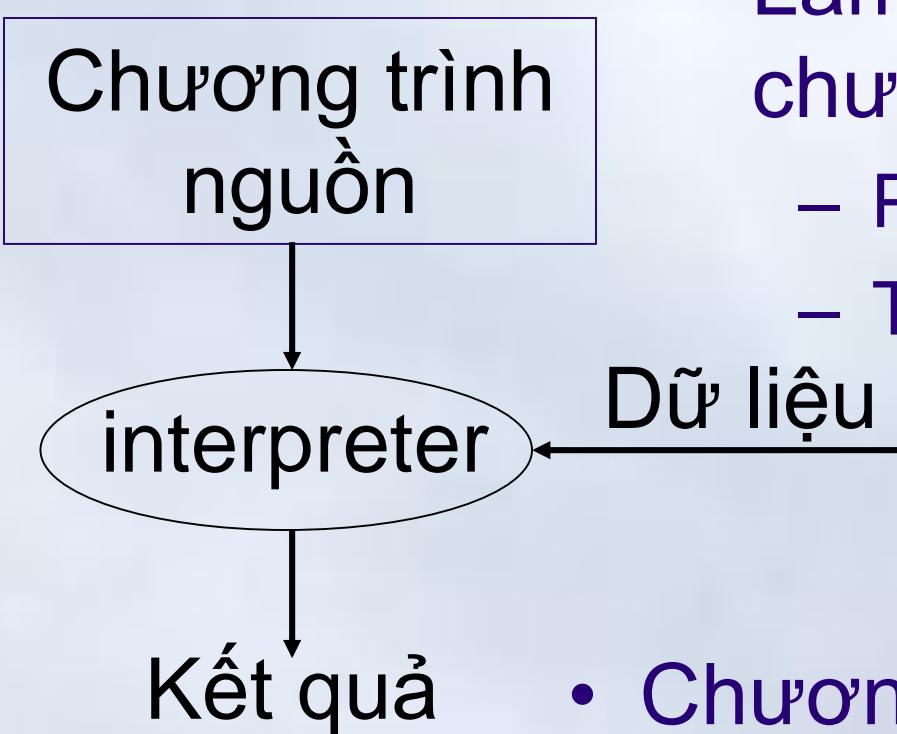


- Chương trình đích có thể không thực hiện được ngay mà cần liên kết (link) đến thư viện để được chương trình thực hiện

Các bước xử lý chương trình



Thông dịch (interpreter)

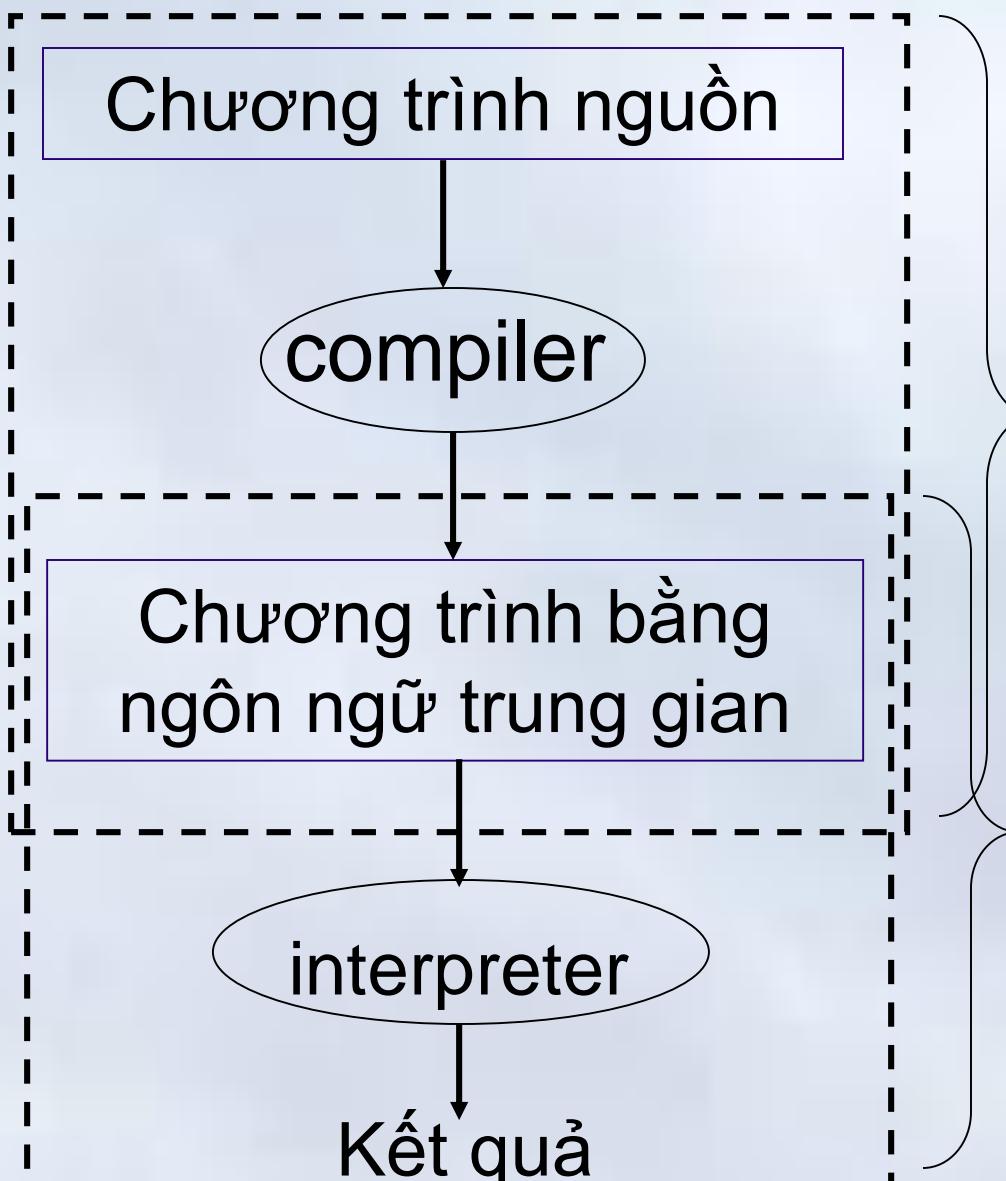


- Làm nhiệm vụ “*giải thích*” chương trình nguồn

- Phân tích câu lệnh tiếp
- Thực hiện câu lệnh

- Chương trình thông dịch có kích thước nhỏ hơn, nhưng chạy chậm hơn

Dịch và thực hiện chương trình nguồn

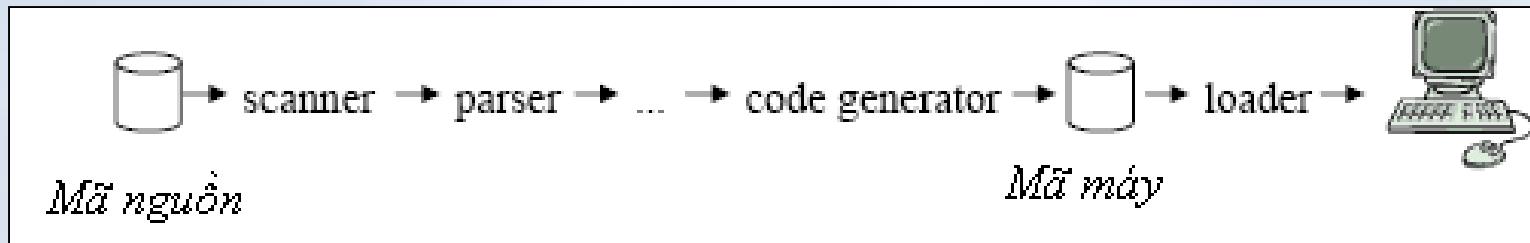


Phase 1: Chuyển từ chương trình nguồn sang NN trung gian

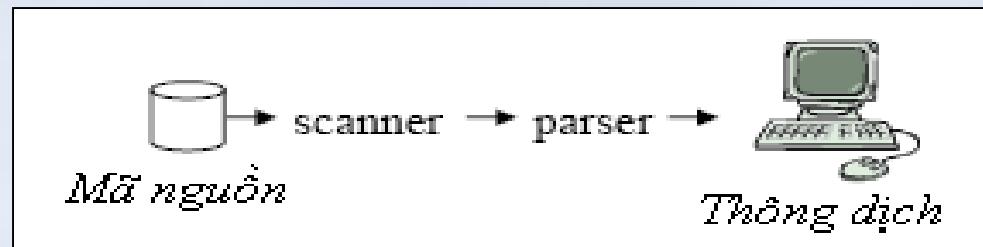
Phase 2: Thực hiện mã trung gian

Compiler >< interpreter

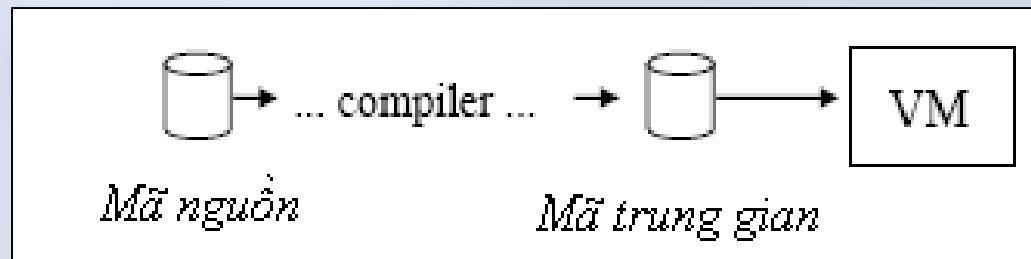
- **Compiler** : Dịch trực tiếp ra mã máy



- **Interpreter** : Trực tiếp thực hiện từng lệnh mã nguồn

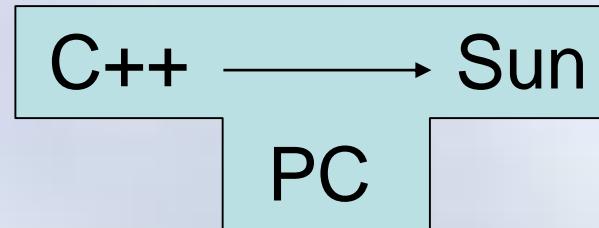
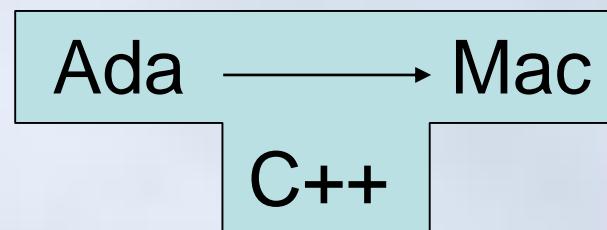
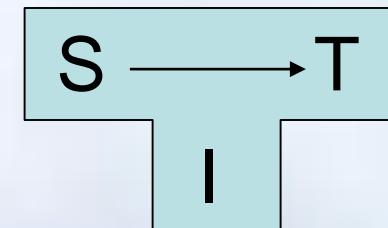


- Biến thể của Interpreter : thông dịch mã trung gian



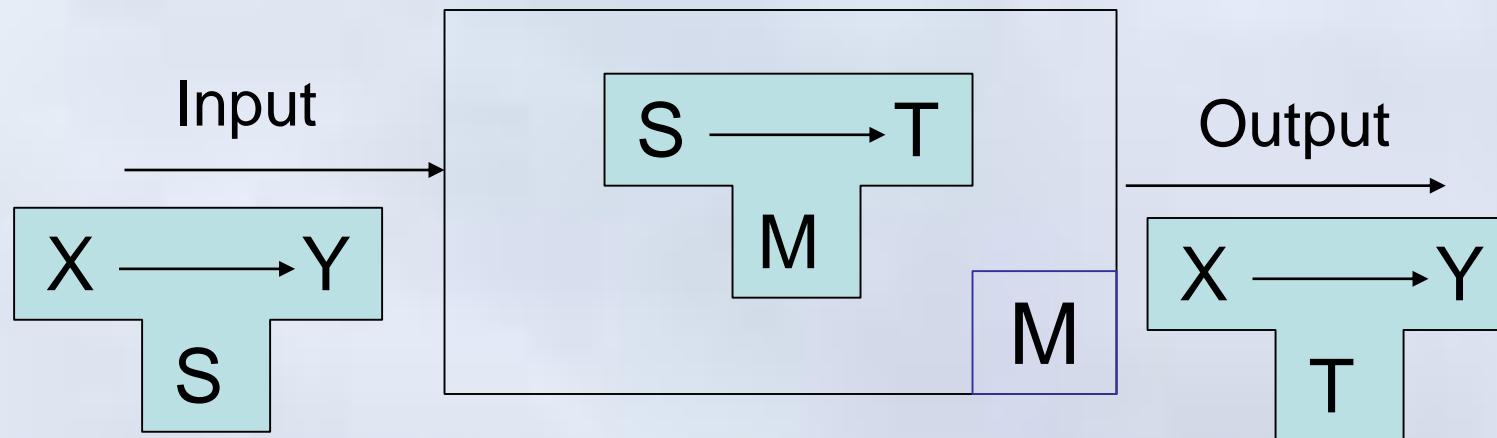
T-Diagram

- Chương dịch được đặc trưng bởi ($T_I^{S \rightarrow T}$)
 - Ngôn ngữ nguồn được dịch **S** (*input*)
 - Ngôn ngữ đích **T** (*output*)
 - Ngôn ngữ cài đặt **I** (*Đã tồn tại*)
 - Có thể là ngôn ngữ máy



T-Diagram

- M là máy tính, có ngôn ngữ máy M
- $T_M^{S \rightarrow T}$: Chương trình dịch trên M, dịch từ S \rightarrow T
 - **Input**: Chương trình viết bằng ngôn ngữ S
 - **Output**: Chương trình viết bằng ngôn ngữ T



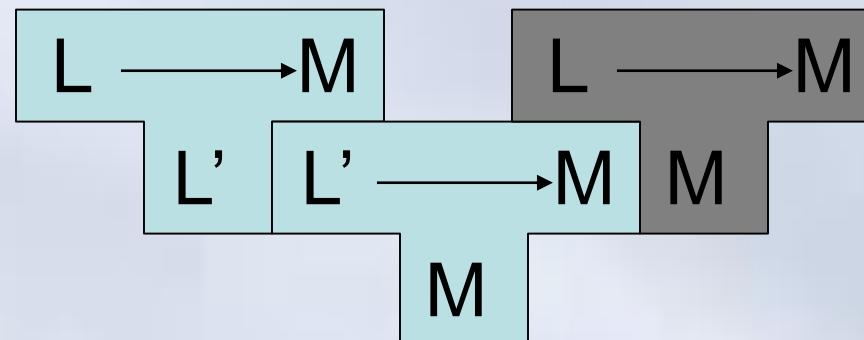
Xây dựng chương trình dịch

- Viết trực tiếp từ ngôn ngữ máy
 - Khó khăn
- Sử dụng ngôn ngữ bậc cao
 - Dễ dàng thực hiện, gõ rối,
 - Hiệu quả
 - Tăng tính khả chuyển,..
- Chương trình bậc cao đầu tiên?
 - Chiến lược mồi (*Bootstrapping*) : Đầu vào của một chương trình là chính nó

Xây dựng chương trình dịch: *Bootstrapping*

Yêu cầu: Viết $T_M^{L \rightarrow M}$

- CTD cho ngôn ngữ L , dịch ra mã máy M
- CTD thực hiện trên máy M
- Dùng mã máy M viết L' là tập con của L
 - Viết ra $T_M^{L' \rightarrow M}$
- Dùng L' để viết CTD $L \rightarrow M$ ($T_{L'}^{L \rightarrow M}$)
- Dùng $T_M^{L' \rightarrow M}$ để dịch $T_{L'}^{L \rightarrow M}$

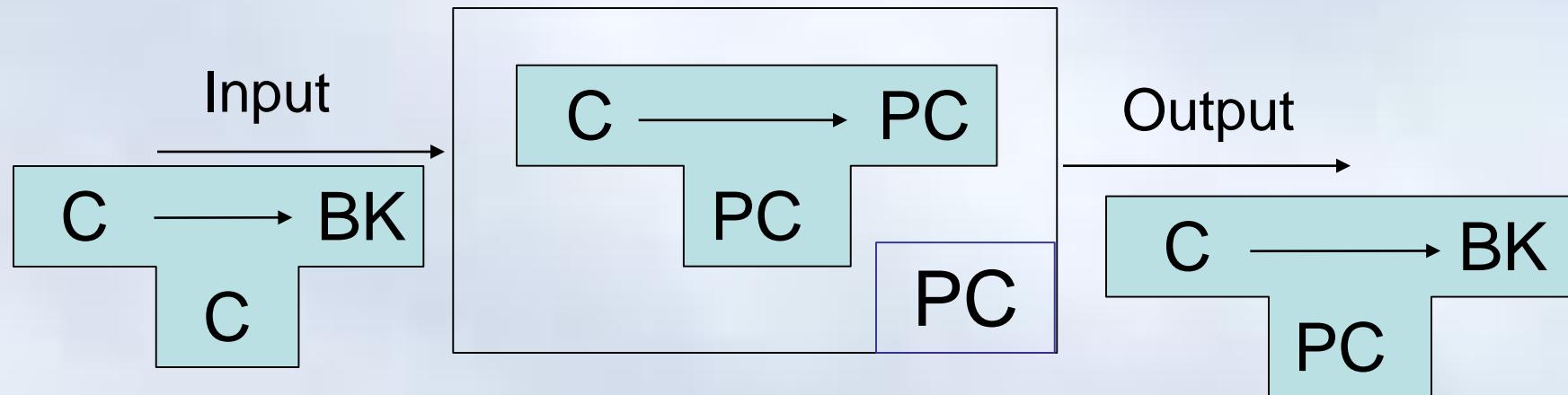


Xây dựng chương trình dịch: Cross Compiling

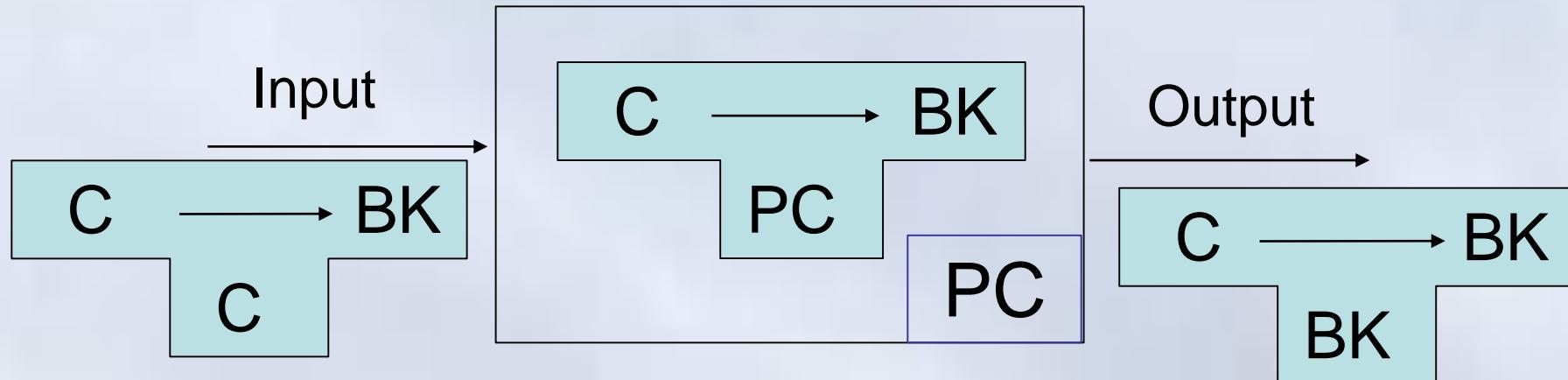
- Mục đích :
 - Xây dựng chương trình dịch cho ngôn ngữ đã tồn tại, trên một loại máy tính mới
 - Nếu máy mới đang được thiết kế: Cần phải biến kiến trúc tập lệnh, dạng lệnh, kiểu địa chỉ,..
 - Không sử dụng mã máy
- Ví dụ:
 - Yêu cầu xây dựng CTD $T_{BK}^{C \rightarrow BK}$
 - BK: là một loại máy tính mới, chưa được sản xuất
 - Đã tồn tại $T_{PC}^{C \rightarrow PC}$
 - Thực hiện xây dựng $T_{PC}^{C \rightarrow BK}$

Xây dựng chương trình dịch: Cross Compiling

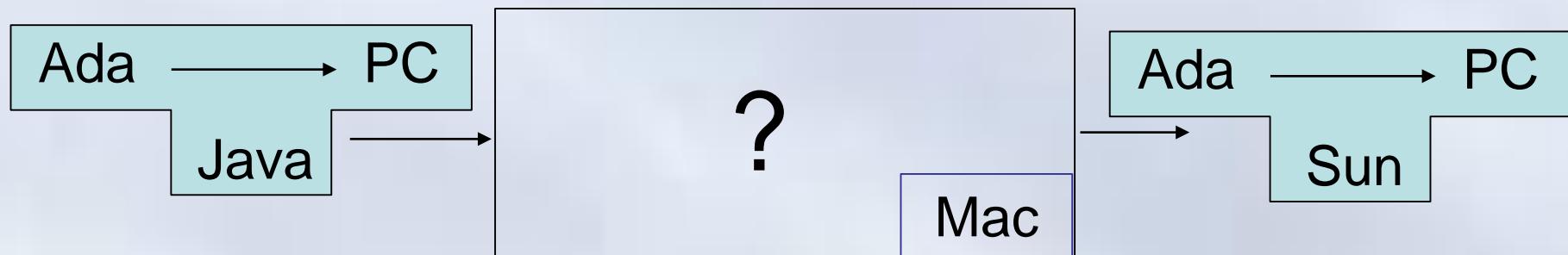
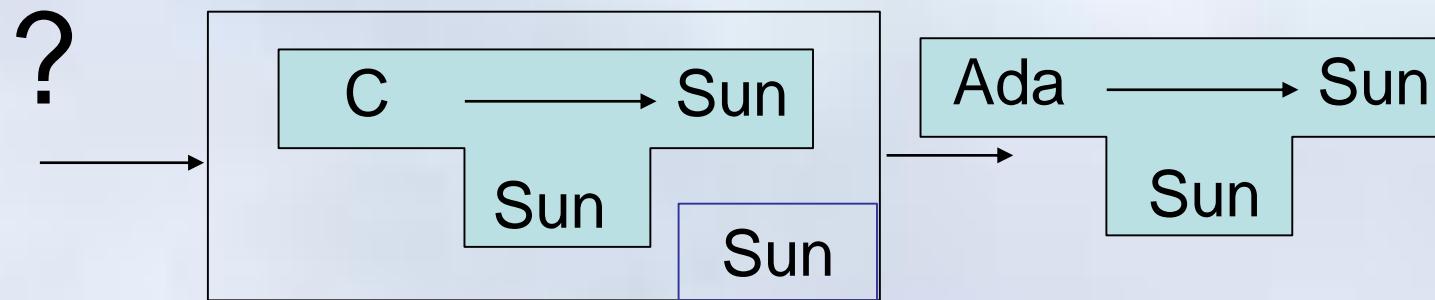
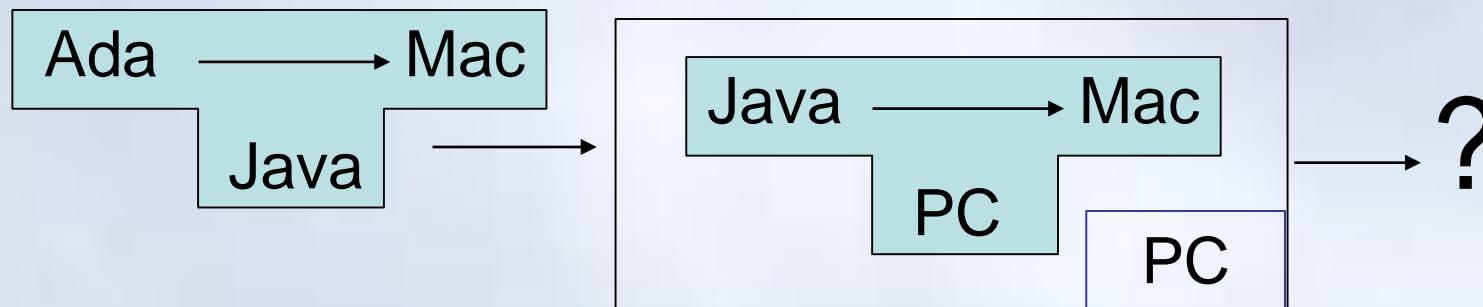
- **Bước 1**



- **Bước 2**



Xây dựng chương trình dịch: Cross Compiling



Chương 1: Những khái niệm cơ bản

1. Ngôn ngữ lập trình cấp cao và trình dịch
2. Đặc trưng của ngôn ngữ lập trình cấp cao
3. Các giai đoạn chính của chương trình dịch
4. Khái niệm ngôn ngữ
5. Văn phạm phi ngữ cảnh
6. Giới thiệu ngôn ngữ PL/0 mở rộng

Các thế hệ ngôn ngữ lập trình

- Được chia thành 5 thế hệ.
- Việc phân chia cấp cao hay thấp phụ thuộc mức độ trừu tượng của ngôn ngữ
 - Cấp thấp : gần với máy
 - Cấp cao : gần với ngôn ngữ tự nhiên

Các thế hệ ngôn ngữ lập trình

- Các ngôn ngữ lập trình bậc thấp
 - Thế hệ thứ nhất : ngôn ngữ máy
 - Thế hệ thứ hai : Assembly
- Thế hệ thứ ba ← Ngôn ngữ bậc cao
 - Dễ hiểu hơn
 - Câu lệnh gần ngôn ngữ tự nhiên
 - Cho phép thực hiện các khai báo, Ví dụ biến
 - Phần lớn các NNLT cho phép lập trình cấu trúc
 - Ví dụ: Fortran, Cobol, C, C++, Basic .

Các thế hệ ngôn ngữ lập trình

- Ngôn ngữ lập trình thế hệ thứ tư
 - Thường được sử dụng trong một lĩnh vực cụ thể
 - Dễ lập trình, xây dựng phần mềm
 - Có thể kèm công cụ tạo form, báo cáo
 - Ví dụ :SQL, Visual Basic, Oracle . . .
- Ngôn ngữ lập trình thế hệ thứ năm
 - Giải quyết bài toán dựa trên các ràng buộc đưa ra cho chương trình (*không phải giải thuật của người lập trình*)
 - Việc giải quyết bài toán do máy tính thực hiện
 - Phần lớn các ngôn ngữ dùng để lập trình logic
 - Giải quyết các bài toán trong lĩnh vực trí tuệ nhân tạo

Các thành phần của NNLTCC

1. Từ vựng và cú pháp
2. Kiểu dữ liệu
3. Các đại lượng
4. Các toán tử và biểu thức
5. Các câu lệnh
6. Chương trình con

Từ vựng và cú pháp

❖ Từ vựng

- Chữ cái: A..Z, a..z
- Chữ số: 0..9
- Dấu: dấu chức năng, dấu toán tử
 - Dấu đơn: +, -, ; {, }
 - Dấu kép: >=, <=, /*, */
- Từ khóa : từ dành riêng cho ngôn ngữ
 - Được dùng để khai báo, ra lệnh cho chương trình

❖ Cú pháp

- Là các quy tắc để
 - Viết ra các đại lượng
 - Viết các câu lệnh

Kiểu dữ liệu

❖ Các kiểu cơ bản

- Kiểu số nguyên **int** Interger
- Kiểu số thực **float** Real
- Kiểu ký tự **char** Char
- Kiểu logic **0, 1** Boolean

❖ Kiểu dữ liệu có cấu trúc

- Kiểu mảng **[]** Array
- Kiểu chuỗi ***** String
- Kiểu bản ghi **struct** Record
- Kiểu con trỏ, **&** ^
- Kiểu file **FILE** File

Các đại lượng

❖ Hằng

- Số nguyên
 - Cách biểu diễn (decimal, octal, hexadecimal,...)
- Số thực
 - Dấu phẩy tĩnh
 - Dấu phẩy động
- Hằng chuỗi
 - C: “Hello”
 - Pascal: ‘Hello’

❖ Tên

- Nguyên tắc đặt tên ?
- Độ dài?

Toán tử và biểu thức

❖ **Toán tử**

- Số học: cộng (+), chia, chia dư(%), MOD,...
- Logic
 - Quan hệ: bằng (=, ==) khác (!=, <>), lớn hơn, (>),...
 - Logic: Và (AND, &&), hoặc (OR, ||),...
- Xâu: ghép xâu ← Pascal

❖ **Biểu thức** ← Kết hợp các toán hạng bởi toán tử

- Số học: *Trả về một con số*
- Logic: *Trả về một giá trị luận lý*
- Xâu: *Trả về một chuỗi ký hiệu*

Các câu lệnh

❖ Câu lệnh tuần tự

- Câu lệnh gán: `:=, =`
- Câu lệnh vào/ra, gọi hàm
- Câu lệnh ghép, gộp: `begin..end, { }`

❖ Câu lệnh rẽ nhánh

- 1 vào 1 ra: `if...then, if()`
- 1 vào 2 ra: `if ...then...else, if()...else...`
- 1 vào, nhiều ra: `caseof, switch() {}`

❖ Câu lệnh lặp

- Số lần lặp xác định: `For ..to/downto..do, for(; ;)`
- Kiểm tra điều kiện trước: `While.. Do, while ()`
- Kiểm tra điều kiện sau: `Repeat..Until, do..while()`

Chương trình con

Các dạng chương trình con

- Thủ tục → Không trả về giá trị (void)
- Hàm → Trả về một giá trị

Vấn đề truyền tham số

- Truyền theo trị: Không thay đổi giá trị
- Truyền theo biến (địa chỉ): thay đổi giá trị

Vấn đề Địa phương /toàn cục

- Địa phương: chỉ tồn tại trong chương trình con
- Toàn cục: Tồn tại trong toàn bộ chương trình

Nhận xét

Ngôn ngữ lập trình bậc cao có
nguyên tắc giống nhau, cách thể
hiện có thể khác nhau

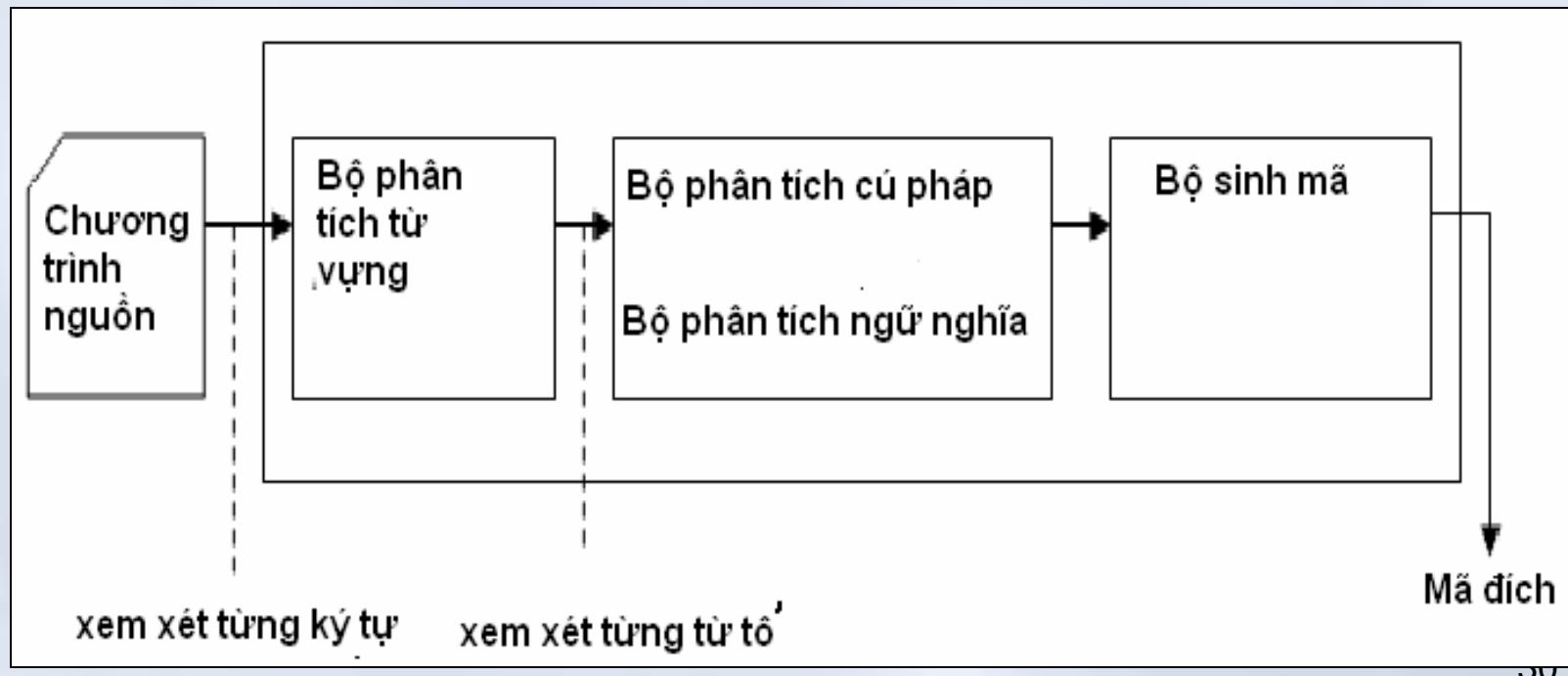
Chương 1: Những khái niệm cơ bản

1. Ngôn ngữ lập trình cấp cao và trình dịch
2. Đặc trưng của ngôn ngữ lập trình cấp cao
3. Các giai đoạn chính của chương trình dịch
4. Khái niệm ngôn ngữ
5. Văn phạm phi ngữ cảnh
6. Giới thiệu ngôn ngữ PL/0 mở rộng

Các phase của chương trình dịch

Chương trình dịch gồm 3 phase chính

1. Phân tích từ vựng
2. Phân tích cú pháp
 - Phân tích ngữ nghĩa
3. Sinh mã



Cấu trúc chương trình dịch

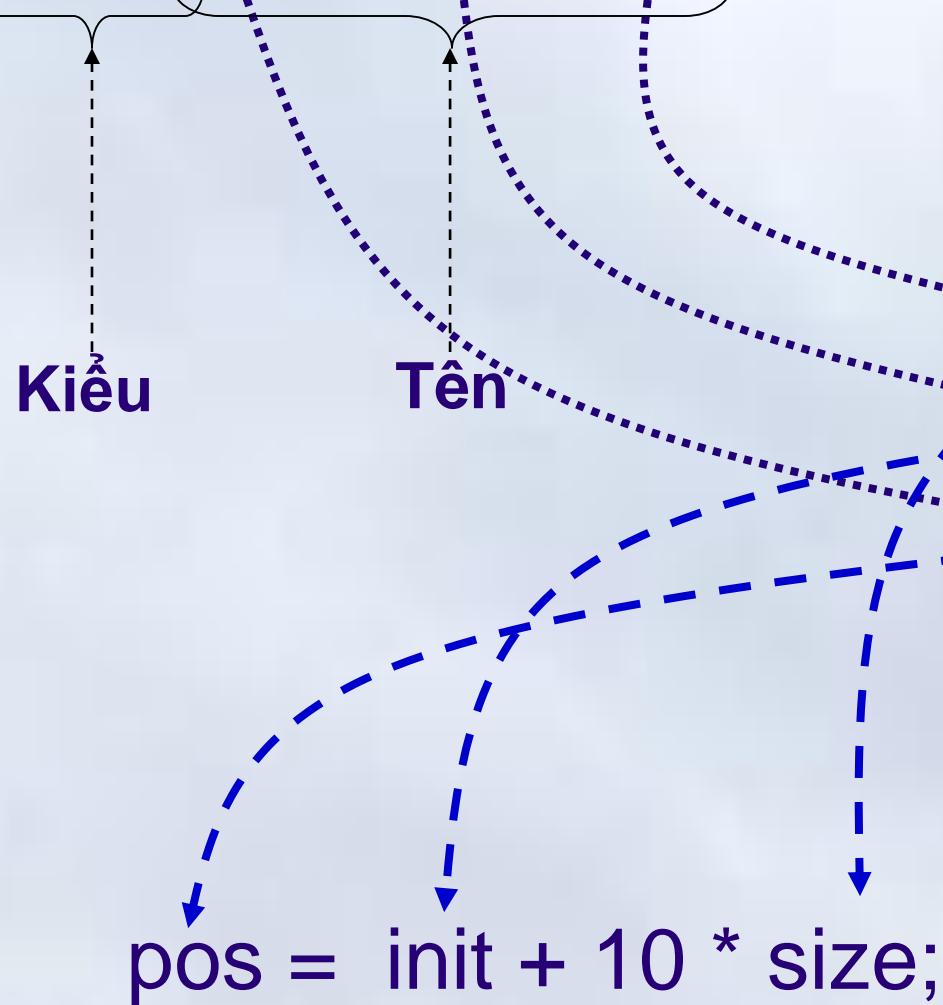


Bảng ký hiệu

- Là cấu trúc dữ liệu dùng chứa tên và thuộc tính cần thiết của chúng
 - Thuộc tính cung cấp thông tin
 - Vị trí, kiểu, phạm vi hoạt động...
 - Nếu là tên chương trình con: số tham số, kiểu trả về
 - Tên được xác định bởi bộ phân tích từ vựng
- Khi chỉ ra được một tên, tùy thuộc vào vị trí của tên trong chương trình
 - Đưa tên và thuộc tính vào bảng ký hiệu
 - Lấy thông tin của tên trong bảng ký hiệu

Bảng ký hiệu → Ví dụ

float pos, init, size; //var pos, init, size: real



Tên	Kiểu	Loại
.....	
size	real	var
init	real	var
pos	real	var
.....	

Bảng ký hiệu

Phân tích từ vựng (Lexical Analysis - Scanner)

Là pha đầu tiên của chương trình dịch

- Duyệt từng ký tự của chương trình nguồn
 - Loại bỏ các ký tự thừa
 - Dấu tab, khoảng trắng, chú thích..
- Xây dựng các từ vựng từ các ký tự đọc được
- Nhận dạng các *từ tố* từ các từ vựng
 - **Từ tố (token)** là đơn vị cú pháp được xử lý trong quá trình dịch như một thực thể không thể chia nhỏ hơn nữa
- Chuyển các từ tố cho pha tiếp

Phân tích từ vựng → Ví dụ

pos := init + 10 * size;

Bộ PTTV thực hiện

- Đọc từng ký tự: bắt đầu từ chữ cái **p**
 - Nhận dạng từ vựng thuộc dạng tên, hoặc từ khóa (vì bắt đầu bởi 1 chữ cái)
 - Đọc tiếp (**o**, **s**) tới khi gặp ký tự khác chữ cái, chữ số,
- Gặp dấu trắng → xây dựng xong từ vựng **pos**
 - Do **pos** không trùng với từ khóa. Vậy **pos** là tên (**ident**)
 - Trả lại cho bộ phân tích cú pháp từ tố **ident**
- Đọc tiếp được dấu **:** rồi dấu **=** và sau đó dấu cách
 - Nhận dạng được từ vựng **:=** và trả về từ tố gán (**assign**)

Phân tích từ vựng → Ví dụ

pos := init + 10 * size;

Kết quả thực hiện PTTV :

	Từ vựng	Tù tố	Ý nghĩa
1	pos	ident	Tên
2	:=	assign	Phép gán
3	init	ident	Tên
4	+	plus	Dấu cộng
5	10	number	Con số
6	*	times	Dấu nhân
7	size	ident	Tên
8	;	semicolon	Chấm phẩy

Trả về: ident, assign, ident, plus, number, times, ident, semicolon

Phân tích cú pháp (Syntax Analysis)

- Bộ ptcp phân tích chương trình nguồn
 - Dựa vào các từ tố nhận được từ pha pttv
- Kiểm tra những từ tố có tuân theo quy tắc cú pháp của ngôn ngữ được dịch không
 - Cú pháp thể hiện cấu trúc văn phạm của ngôn ngữ, được mô tả dạng: đệ quy, BNF, sơ đồ..
- Kết quả của bộ phân tích cú pháp:
 - Cây phân tích cú pháp (*nếu có*)
 - Có cây phân tích → chương trình đúng cú pháp
 - Thông báo lỗi nếu ngược lại

Phân tích cú pháp → Ví dụ 1

- Quy tắc định nghĩa một biểu thức
 1. Tên là một biểu thức
 2. Con số là biểu thức
 3. Nếu E , E_1 , E_2 là biểu thức thì E_1+E_2 , $E_1 * E_2$, (E) là biểu thức

Luật cơ sở

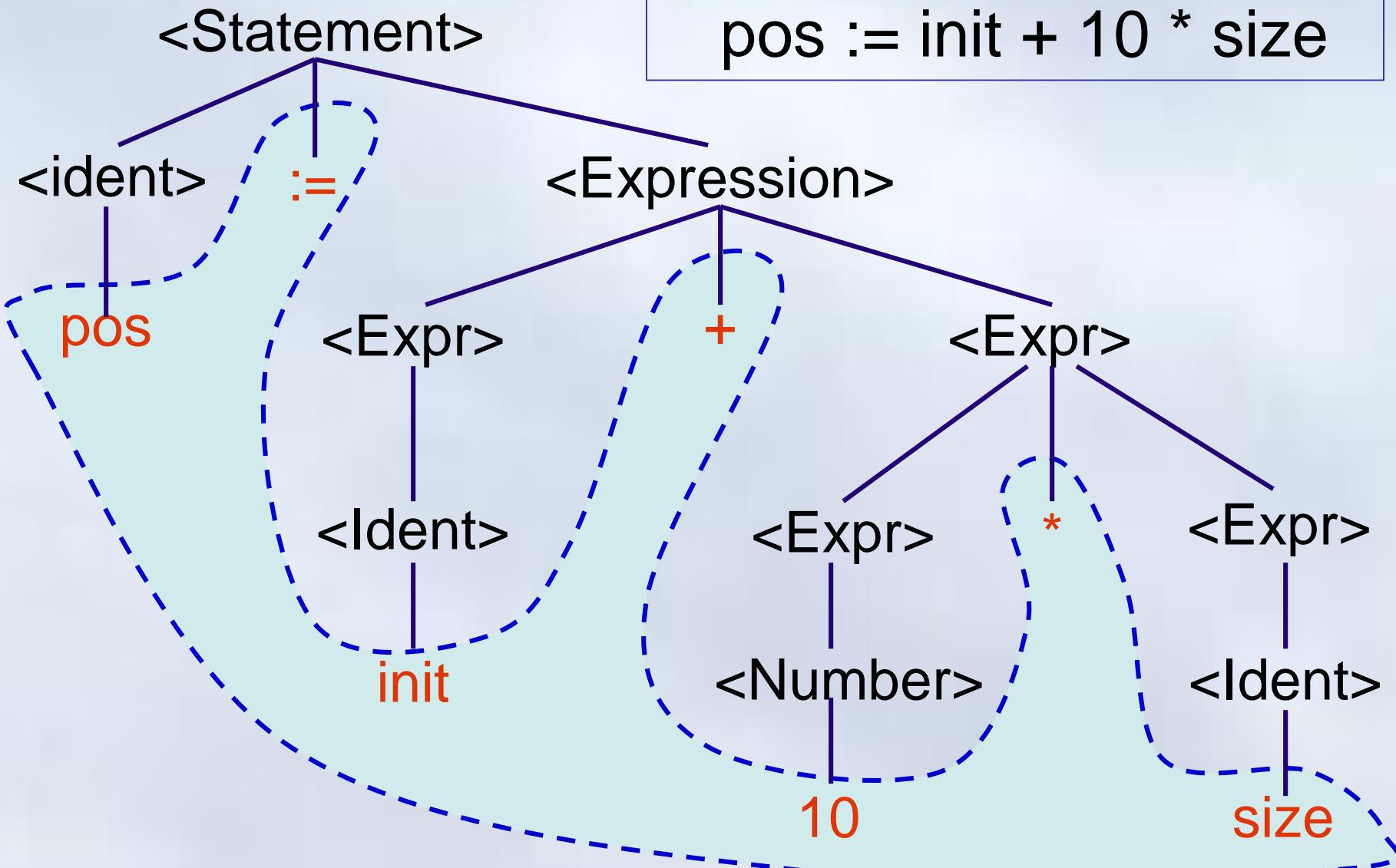
Luật đệ quy
- Kiểm định: ***init + 10 * size* là biểu thức?**
 - Theo (2): 10 là biểu thức
 - Theo (1): $size$ là biểu thức
 - Theo (3) $10 * size$ là biểu thức
 - Theo (1): $init$ là biểu thức
 - Theo (3): $init + 10 * size$ là biểu thức

Phân tích cú pháp → Ví dụ 2

- Quy tắc định nghĩa một lệnh
 1. Nếu Id là một Tên, E là một biểu thức thì
Id := E là một câu lệnh
 2. Nếu E là một biểu thức, S là một lệnh thì
If E Then S, While E Do S là câu lệnh
- Kiểm định: ***pos := init + 10 * size*** là câu lệnh?
- Theo bộ phân tích từ vựng: *pos* là một tên
 - Theo ví dụ 1: *init + 10 * size* là một biểu thức
 - Vậy theo luật (1) *pos := init + 10 * size* là một lệnh

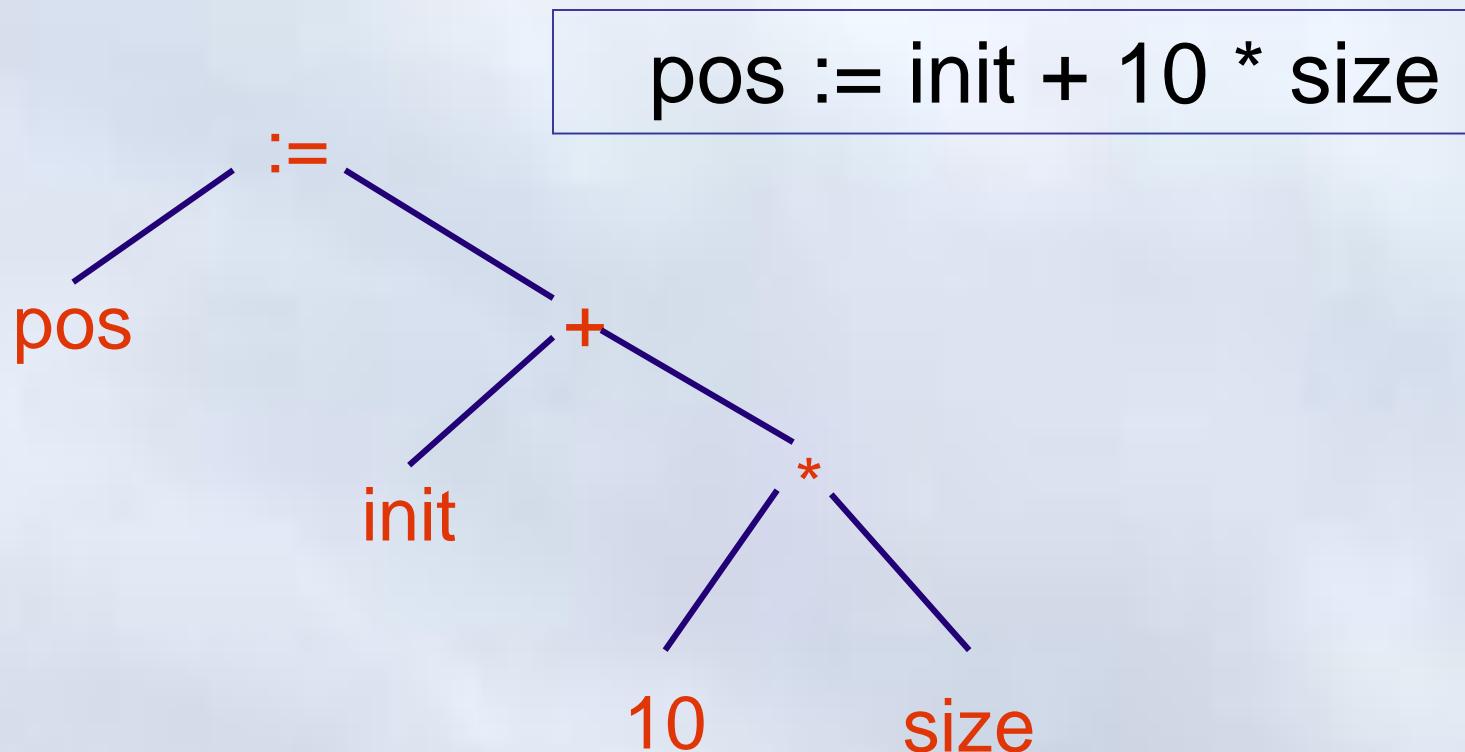
Ptcp thường được biểu diễn bởi cây phân tích/cây cú pháp

Phân tích cú pháp → Cây phân tích (parse tree)



Phân tích cú pháp → Cây cú pháp (syntax tree)

- Các nút trong là các toán tử
- Các toán hạng là các nút con của toán tử



Phân tích ngữ nghĩa (Semantic Analysis)

- Kiểm tra lỗi ngữ nghĩa của chương trình
 - Lệnh $pos := init + 10 * size$ đúng cú pháp
 - Nếu pos là hằng số
 - $pos := init + 10 * size$ sai ngữ nghĩa
- Kiểm tra kiểu toán hạng có phù hợp toán tử
 - % (MOD) đòi hỏi toán hạng nguyên
 - Một số toán tử chấp nhận toán hạng khác kiểu
 - Vấn đề chuyển kiểu tự động (*nguyên* → *thực*)
 - *Ví dụ* $pos := init + \text{int2Real}(10) * size$
- Lấy thông tin về kiểu của danh biểu (tên)
 - Thông tin dùng cho giai đoạn sinh mã

Sinh mã (Code Generation)

- Được thực hiện khi chương trình nguồn đúng cả về cú pháp và ngữ nghĩa
- Thường gồm 3 giai đoạn
 - Sinh mã trung gian
 - Tối ưu mã
 - Sinh mã đích

Sinh mã → Sinh mã trung gian

- Mã nguồn được chuyển sang chương trình tương đương trong ngôn ngữ trung gian
 - Mã trung gian là mã máy độc lập, tương tự với tập lệnh trong máy.
- Ưu điểm của mã trung gian
 - Thuận lợi khi cần thay đổi cách biểu diễn chương trình đích
 - Có thể tối ưu hóa mã độc lập với máy đích cho dạng biểu diễn trung gian.
 - Giảm thời gian thực thi chương trình đích vì mã trung gian có thể được tối ưu
- Ngôn ngữ trung gian
 - Mã 3 địa chỉ (*thường được dùng*)
 - Cây cú pháp, Ký pháp Ba Lan sau (hậu tố),..

Sinh mã \rightarrow Sinh mã trung gian \rightarrow Mã 3 địa chỉ

- Mỗi câu lệnh có nhiều nhất
 - 3 toán hạng
 - 2 toán tử, trong đó có 1 toán tử gán
- Chương trình dịch phải sinh ra biến tạm để chứa giá trị tính toán sau mỗi lệnh

Ví dụ: $pos := init + \text{int2Real}(10) * size$

Sinh ra mã 3 địa chỉ

$\text{Temp1} := \text{Int2Real}(10)$
 $\text{Temp2} := \text{Temp1} * \text{Id 3}$
 $\text{Temp3} := \text{Id2} + \text{Temp2}$
 $\text{Id1} := \text{Temp3}$

Id1, Id2, Id 3 là các
 con trỏ, trỏ tới các
 phần tử size, init, pos
 trong bảng ký hiệu

Sinh mã → Tối ưu mã trung gian

- Mục đích:

- Tối ưu về kích thước, tốc độ

Ví dụ

- *Int2Real()* được thay bằng số thực tại thời điểm dịch
 - *Temp3* chỉ dùng 1 lần làm nơi lưu tạm thời dữ liệu trước khi chuyển cho *Id1*
 - Có thể thay *Id1* trực tiếp cho *Temp3*
 - Kết quả

Temp1 := 10.0 * *Id 3*

Id1 := *Id2* + *Temp1*

Sinh mã → Sinh mã đích

- **Mục đích:** Tạo chương trình thực thi
- **Thực hiện:**
 - Các biến được cấp ô nhớ cụ thể
 - Các câu lệnh trung gian được thay bằng chuỗi mã máy tương đương
 - Các biến được ấn định cho các thanh ghi

Ví dụ

```

MOVF Id3, R2
MULF #10.0, R2 } Temp1 := 10.0 * Id 3

MOVF Id2, R1
ADDF R2, R1 } ID1:= Id2 + Temp1

MOVF R1, Id1

```

Xử lý lỗi

Lỗi có thể gặp trong mọi pha của CTD

- Phân tích từ vựng
 - Gặp ký tự lạ. Ví dụ: @, \$,.. trong NNLT C
- Phân tích cú pháp
 - Không tuân theo cú pháp: VD *while (a <> b)*
- Phân tích ngũ nghĩa
 - Gán giá trị cho hằng, tính toán với tên thủ tục
- Sinh mã
 - Kích thước quá lớn
 - Ví dụ: int Arr[1000][1000]; → Lỗi tràn ô nhớ

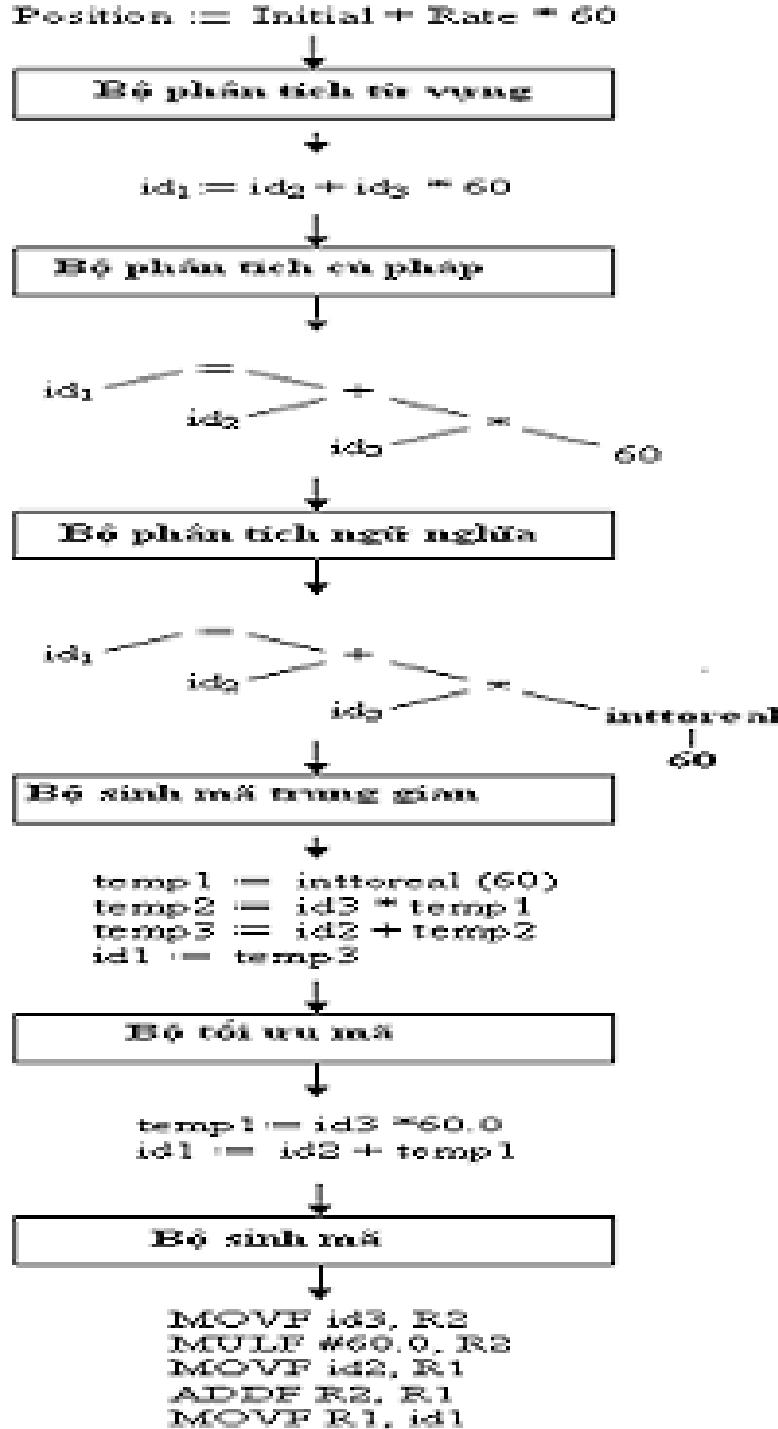
Xử lý lỗi

- Gặp lỗi, chương trình dịch cần thông báo
 - Ghi nhận kiểu lỗi
 - Ghi nhận vị trí gây ra lỗi (dòng, cột,...)
- Có thể cần hồi phục sau khi gặp lỗi
 - Mục đích: cho phép tiến hành phân tích tiếp tục, tránh lãng phí
 - Có thể thông báo không chính xác

[Phan Thi Tuoi, 1998]

Quá trình dịch một câu lệnh

Position := Initial + Rate * 60



Ghi chú

- Phân chia thành từng pha nhằm mục đích nghiên cứu để chọn giải pháp thích hợp
- Trong cài đặt, kết hợp các pha thành các modul chương trình. Thường gồm 2 phần
 - **Phần đầu:** Modul liên quan tới phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa và sinh mã trung gian
 - **Phần sau:** Tối ưu mã, sinh mã đích, xử lý lỗi
- Cơ sở đề ra ngôn ngữ lập trình là **lý thuyết ngôn ngữ và văn phạm**

Chương 1: Những khái niệm cơ bản

1. Ngôn ngữ lập trình cấp cao và trình dịch
2. Đặc trưng của ngôn ngữ lập trình cấp cao
3. Các giai đoạn chính của chương trình dịch
4. Khái niệm ngôn ngữ
5. Văn phạm phi ngữ cảnh
6. Giới thiệu ngôn ngữ PL/0 mở rộng

Giới thiệu

- Ngôn ngữ (*tự nhiên/nhân tạo*):
 - Tập hợp các câu có cấu trúc quy định
- Câu:
 - Tập hợp các từ
- Từ:
 - Ký hiệu nào đó
- Ngữ pháp/cú pháp
 - **Cấu trúc quy định** tạo câu của trong ngôn ngữ
 - Ngôn ngữ lập trình → cú pháp

Khái niệm văn phạm và ngôn ngữ

1. Kiến thức toán học liên quan
2. Bộ chữ và xâu ký hiệu
3. Văn phạm
4. Suy dẫn
5. Câu
6. Ngôn ngữ
7. Đệ quy
8. Phân loại văn phạm

Kiến thức toán học liên quan → Tập hợp

- Khái niệm:
 - Tập không có thứ tự và không lặp lại của các p/tử
 - Ví dụ: $A_5 = \{1, 3, 5\}$ tập các số nguyên dương lẻ nhỏ hơn 6
 - Tập không có phần tử nào được gọi là **tập rỗng**
 - Tập rỗng được ký hiệu: \emptyset
- Biểu diễn: $A = \{x \mid x \text{ thỏa mãn tính chất } Q\}$
 - Chỉ ra tập các phần tử thỏa mãn tính chất Q
 - Sử dụng biểu đồ Venn
 - Mỗi phần tử biểu diễn là một điểm
 - Mỗi tập là một hình bao quanh các điểm
- Lực lượng của một tập: Số phần tử của tập
 - Tập có thể hữu hạn hoặc vô hạn phần tử
 - Lực lượng của tập A, được ký hiệu $|A|$

Kiến thức toán học liên quan → Tập hợp

- Ký hiệu :
 - $a \in A$ a là phần tử trong tập A
 - $a \notin A$ a không phải phần tử trong tập A
 - $A \subset B$ A là tập con thực thụ của B
 - $A \subseteq B$ A là tập con của B
- Tập lũy thừa của A: Tập các tập con của A
 - Tập lũy thừa của A ký hiệu 2^A
 - A là một tập thì $2^A = \{S | S \subseteq A\}$
 - Ví dụ $2^{\{1,3,5\}} = \{\emptyset, \{1\}, \{3\}, \{5\}, \{1,3\}, \{1,5\}, \{5,3\}, \{1,3,5\}\}$
 - Nhận xét $|2^A| = 2^{|A|}$
 - $S \in 2^A \rightarrow \exists$ số nhị phân $e = e_1..e_n$ ($n = |A|$) thỏa mãn: $e_i = 1$ khi phần tử i của A có mặt trong S; Ngược lại $e_i = 0$

Kiến thức toán học liên quan → Tập hợp

- Các phép toán trên tập: A, B là 2 tập

– **Hợp:** $A \cup B = \{a \mid a \in A \text{ hoặc } a \in B\}$

– **Giao:** $A \cap B = \{a \mid a \in A \text{ và } a \in B\}$

- A, B được gọi là phân biệt nếu $A \cap B = \emptyset$

– **Hiệu:** $A - B = \{a \mid a \in A \text{ và } a \notin B\}$

– **Tích Đè-các:** $A \times B = \{(a, b) \mid a \in A \text{ và } b \in B\}$

- $A = \{1, 2\}$
- $B = \{a, b\}$

$$A \times B = \{(1, a), (1, b), (2, a), (2, b)\}$$

- $A_1 \times A_2 \times \dots \times A_n = \{(a_1, a_2, \dots, a_n) \mid a_i \in A_i\}$

- Tích Đè-các lũy thừa bậc n:

$$A^n = A \times A \times \dots \times A = \{(a_1, \dots, a_n) \mid a_i \in A\}$$

Kiến thức toán học liên quan → Quan hệ

- Khái niệm:
 - Tập con \mathcal{R} của tích Đè-các của các tập
- Quan hệ 2 ngôi:
 - Tập con của tích Đè-các 2 tập
- Ví dụ:
 - $a \in A$ và $b \in B \Rightarrow (a, b) \in A \times B$ là một quan hệ 2 ngôi
 - Nếu $A \equiv B \Rightarrow$ Quan hệ 2 ngôi trên A
- Nếu \mathcal{R} là một quan hệ và $(a, b) \in \mathcal{R}$, ký hiệu $a \mathcal{R} b$

Kiến thức toán học liên quan → Hàm

- Hàm là một quan hệ trên tích Đè-các $\mathcal{D} \times \mathcal{R}$ thỏa mãn
 - $\forall d \in \mathcal{D}, \exists$ nhiều nhất 1 cặp $(d, r) \in \mathcal{D} \times \mathcal{R}$
 - \mathcal{D} được gọi là **miền xác định**
 - \mathcal{R} được gọi là **miền giá trị**
- Ký hiệu $f: \mathcal{D} \rightarrow \mathcal{R}$ (f : là một hàm/ ánh xạ)
 - Nếu $(d, r) \in \mathcal{D} \times \mathcal{R}$ thì
 - Giá trị của hàm f tại d là r và ký hiệu $f(d) = r$
- Hàm xác định: Khi \mathcal{D} và \mathcal{R} là những tập xác định
 - Có thể biểu diễn bằng bảng các cặp $\{(d, r)\}$
- Hàm n biến vào, m biến ra
 - Miền xác định là tích Đè-các của n tập,
 - Miền giá trị là tích Đè-các của m tập

Kiến thức toán học liên quan → Đồ thị

Đồ thị $G = (V, E)$ bao gồm

- Tập hữu hạn V các nút - đỉnh
- Tập hữu hạn E các cặp nút – cạnh
 - $E = \{(v_1, v_2) | v_1, v_2 \in V\} \Leftrightarrow E \subseteq V \times V$
- Đồ thị vô hướng: $\exists (v_1, v_2) \in E \Rightarrow \exists (v_2, v_1) \in E$
- Đồ thị có hướng: Không phải là đồ thị vô hướng
 - Nếu $(v_1, v_2) \in E$: v_1 là đỉnh đứng trước v_2 trong đồ thị
- Đường đi trên đồ thị có hướng/vô hướng
 - Dãy đỉnh (v_1, v_2, \dots, v_n) thỏa mãn $(v_i, v_{i+1}) \in E$ ($i: 1..n-1$)
- Đồ thị có chu trình:
 - \exists đường đi đỉnh (v_1, v_2, \dots, v_n) , trong đó $v_1 \equiv v_1$

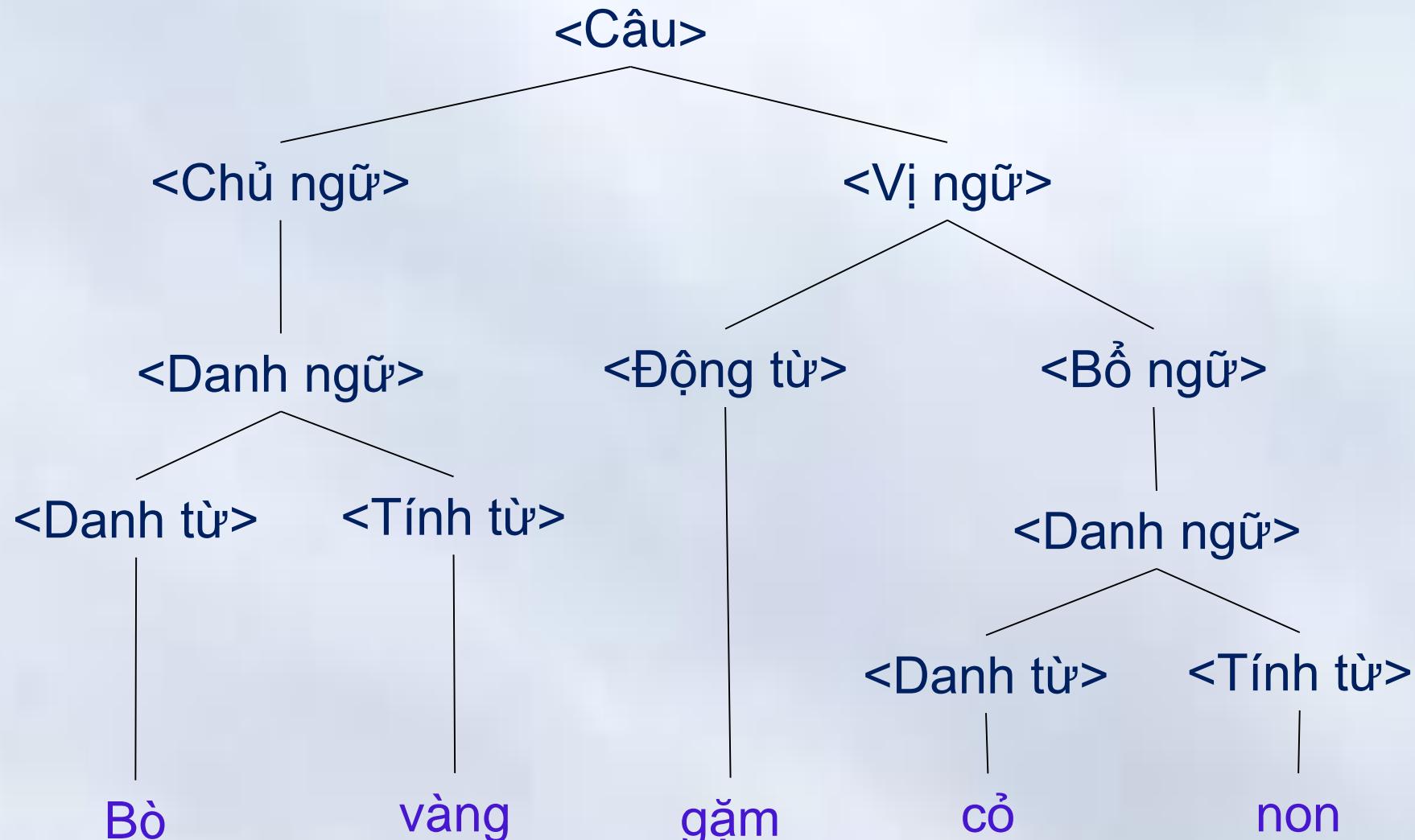
Kiến thức toán học liên quan → Cây

Là một đồ thị định hướng với các tính chất sau

- Có một nút gọi là gốc, có các tính chất
 - Không có nút đứng trước gốc
 - Có một đường đi từ gốc đến mọi nút khác
- Nút khác với nút gốc có đúng một nút đứng trước
- Các nút sau của một nút: **Sắp thứ tự** từ trái qua phải
- Quy ước biểu diễn cây
 - Vẽ nút gốc ở trên, cung hướng **xuống dưới** (bỏ mũi tên)
 - Các nút sau vẽ theo thứ tự từ trái qua phải
- Nút lá: Không có nút sau (không có con)
 - Nút trong: Mọi nút khác nút lá

Kiến thức toán học liên quan → Cây

Cấu trúc ngũ pháp của một câu tiếng việt



Bộ chữ

- Tập các thành phần được gọi là các ký hiệu
 - Trong thực tế: Tập là hữu hạn và khác rỗng.
 - Thường được ký hiệu **V** (Vocabulary) hoặc Σ

Ví dụ

- Bảng chữ cái a,b,c...:
 - Bộ chữ cái của một ngôn ngữ tự nhiên
- Bộ từ điển tiếng Anh
 - Bộ chữ cái trong ngôn ngữ tiếng Anh
- Các từ khóa, tên, ký hiệu..
 - Bộ chữ của một ngôn ngữ lập trình

Xâu ký hiệu

Khái niệm

- Là dãy liên tiếp các ký hiệu của một bộ chữ
 - Được gọi là xâu trên bộ chữ đó
 - Thường được ký hiệu $\alpha, \beta, \gamma, \dots$
- Xâu rỗng: xâu không chứa ký hiệu nào
 - Thường được ký hiệu ϵ

Ví dụ

- α : Madam \rightarrow Xâu trên bảng chữ cái a, b, c,..
- β : Hôm_nay trời đẹp \rightarrow Xâu trong tiếng Việt
- γ : while (a > b) a = a – b; \rightarrow Xâu trong NNLT

Xâu ký hiệu

- Độ dài xâu

- Số ký hiệu trong xâu đó
- Thường được ký hiệu $|\alpha|$ hay $l(\alpha)$

- Ví dụ: $l(\alpha) = 5$ $l(\beta) = 3$
- $l(\gamma) = 12$ $l(\varepsilon) = 0$

- Đảo ngược xâu

- Đảo ngược xâu α , (ký hiệu là α^R)
 - Viết các ký hiệu của xâu α theo chiều ngược lại

- Xâu con

- Xâu v là xâu con của w nếu xâu v được tạo nên từ dãy các ký hiệu liên tiếp của xâu w
 - v là tiền tố w nếu v nằm ở đầu, nếu nằm ở cuối, v là hậu tố
 - Ví dụ: ada là xâu con của xâu $madam$

Xâu ký hiệu

- Ghép xâu
 - Ghép 2 xâu α, β là việc viết các ký hiệu của xâu α rồi viết tiếp các ký hiệu của xâu β
- Xâu lũy thừa:
 - $\alpha^n = \alpha \alpha \dots \alpha$ (ghép n xâu α)
- Ví dụ: $\alpha: abc$ và $\beta: ba$
 - $\alpha\beta = abcba$ và $\beta\alpha = bacab$
 - $\alpha^3 = abcababc$
- Nhận xét:
 - $\alpha\beta \neq \beta\alpha$; $\alpha\varepsilon \equiv \varepsilon\alpha \equiv \alpha$; $\varepsilon\varepsilon \equiv \varepsilon$;
 - $I(\alpha\beta) = I(\alpha) + I(\beta)$

Xâu ký hiệu → Tính toán trên tập xâu

- *A, B là 2 tập xâu trên một bộ chữ*
 - **Hợp:** $A \cup B = \{\alpha \mid \alpha \in A \text{ hoặc } \alpha \in B\}$
 - **Giao:** $A \cap B = \{\alpha \mid \alpha \in A \text{ và } \alpha \in B\}$
 - **Tích/Ghép:** $AB = \{x = \alpha\beta \mid \alpha \in A \text{ và } \beta \in B\}$
 - **Tích Descarter:** $AxB = \{<\alpha, \beta> \mid \alpha \in A \text{ và } \beta \in B\}$
- **Ví dụ:** $A = \{0, 01\}$ $B = \{\varepsilon, 1, 01\}$
 - $A \cup B = \{\varepsilon, 0, 1, 01\}$
 - $A \cap B = \{01\}$
 - $AB = \{0, 01, 001, 011, 0101\}$
 - $AxB = \{<0, \varepsilon>, <0, 1>, <0, 01>, <01, \varepsilon>, <01, 1>, <01, 01>\}$

Xâu ký hiệu → Tính toán trên tập xâu

- A là *tập xâu trên một bộ chữ*
 - **Lũy thừa:** $A^n = \{\epsilon\}$ nếu $n = 0$
 $A^n = AA^{n-1} = A^{n-1}A$ nếu $n > 0$
 - **Bao đóng:** $A^* = \lim(A^0 \cup A^1 \cup \dots \cup A^n)$, $n \rightarrow \infty$
 - **Bao đóng dương:** $A^+ = \lim(A^1 \cup A^2 \cup \dots \cup A^n)$
- Nhận xét
 - **A^* :** Tập tất cả các xâu có thể được tạo nên từ các ký hiệu trong A, *kể cả xâu rỗng*.
 - **A^+ :** Tập tất cả các xâu có thể được tạo nên từ các ký hiệu trong A, *không kể xâu rỗng*.

Xâu ký hiệu → Tính toán trên tập xâu

- **Ví dụ :** Cho $A = \{0, 1\}$, $B = \{a, b\}$
 - $- A^0 = \{\varepsilon\}$ $A^1 = \{0, 1\}$
 - $- A^2 = \{00, 01, 10, 11\}$
 - $- A^n = \{Tập các số nhị phân độ dài n\}$
 - $- A^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \dots\}$
 - $- A^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$
 - $- BA^2 = \{a, b\}\{00, 01, 10, 11\}$
 $= \{a00, a01, a10, a11, b00, b01, b10, b11\}$
 - $- BA^+ = ?$
 - $- A^+BA^+ = ?$
 - $- ABA^* = ?$

Ngôn ngữ (Languages)

- Cho V Là một bộ chữ
 - V^* Tập tất cả các xâu trên bộ chữ, kể cả xâu ϵ
 - V^+ Tập tất cả các xâu trên bộ chữ, không kể xâu ϵ
 - V^* là đếm được: Liệt kê lần lượt các xâu của V^* theo thứ tự độ dài
Cùng độ dài, liệt kê theo thứ tự từ điển
- Cho V Là một bộ chữ,
 $L \subseteq V^*$ được gọi là một ngôn ngữ trên bộ chữ V
 - Mỗi p/tử của L được gọi là một câu/từ/xâu ngôn ngữ
 - Tập $\{\epsilon\}$ và \emptyset là ngôn ngữ trên mọi bộ chữ
- Nhận xét
 - Ngôn ngữ là một tập (các xâu) \Rightarrow Có thể tạo ngôn ngữ mới từ các ngôn ngữ đã có bởi sử dụng toán tử tập

Ngôn ngữ (Languages) → Các phép toán

- Ví dụ:
 - Nếu L, L_1, L_2 : Ngôn ngữ trên cùng một bộ chữ V
 - $L_1 \cap L_2, L_1 \cup L_2, L_1 - L_2, L_1 L_2$: Ngôn ngữ trên V
 - $V^* - L, L^+, L^*$: Ngôn ngữ trên V
- Tính chất
 - $L^+ = LL^* = L^* L$
 - $L^* = L^+ \cup \{\epsilon\}$
- Câu hỏi: Các phát biểu sau đúng hay sai?
 - L là ngôn ngữ thì $L^+ = L^* - \{\epsilon\}$
 - $|L_1 L_2| = |L_1| \times |L_2|$

Văn phạm (Grammar)

$$G = (V_T, V_N, P, S)$$

- **V_T** : Tập các ký hiệu kết thúc của một bảng chữ
 - Các phần tử của V_T thường được ký hiệu: a, b, c,..
- **V_N** : Tập các k/hiệu không kết thúc của một bảng chữ
 - Các phần tử của V_N thường được ký hiệu: A, B, C,..
 - $V_T \cap V_N = \emptyset$; $V = V_T \cup V_N$: Bộ chữ của văn phạm
- **S** : Ký hiệu bắt đầu. $S \in V_N$
- **P** : Tập hữu hạn các cặp (α, β) được gọi là các quy tắc hay các sản xuất. Thường được viết $\alpha \rightarrow \beta$
 - $\alpha \in V^* V_N V^*$ // \rightarrow Phải có ít nhất một ký hiệu không k/thúc
 - $\beta \in V^*$ // \rightarrow Có thể chứa xâu rỗng

Văn phạm → Ví dụ

Cho văn phạm

$$G_1 = (V_T, V_N, P, S)$$

Trong đó

- $V_T = \{a, b, c\}$
- $V_N = \{S, A\}$
- $P = \{S \rightarrow aSA, S \rightarrow b, A \rightarrow bA, A \rightarrow c\}$

Ghi chú

- Chỉ cần nhắc tới tập sản xuất khi giới thiệu văn phạm

Suy dẫn (Derivation)

Cho văn phạm $G = (V_T, V_N, P, S)$

- Gọi γ viết ra δ hay δ được suy dẫn trực tiếp ra từ γ , và được ký hiệu $\gamma \Rightarrow \delta$ nếu tồn tại các xâu α, β, v, w thỏa mãn các điều kiện

- $\gamma = \alpha v \beta \quad \left. \begin{array}{l} \\ \end{array} \right\}$ Do $v \rightarrow w$ là một sản xuất của P .
- $\delta = \alpha w \beta \quad \left. \begin{array}{l} \\ \end{array} \right\}$ Thay v trong γ bằng w , sẽ được δ
- $(v, w) \in P$ {có thể viết $v \rightarrow w \in P$ }
- $\alpha, \beta \in V^*, v \in V^* V_N V^*, w \in V^*$

- Ví dụ với văn phạm G_1

- $aSc \Rightarrow abc \{a=a, \beta=c, v=S, w=b, S \rightarrow b \in P\}$

Suy dẫn

Cho văn phạm $G = (V_T, V_N, P, S)$

- Gọi δ được suy dẫn ra từ γ , nếu tồn tại dãy các xâu $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_n$ thỏa mãn điều kiện

$$\gamma = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n = \delta$$

- Nếu $n \geq 1$, áp dụng ít nhất 1 bước suy dẫn, ký hiệu: $\gamma \Rightarrow^+ \delta$
- Nếu $n \geq 0$, có thể không áp dụng bước nào, ký hiệu: $\gamma \Rightarrow^* \delta$

Suy dẫn → Ví dụ

Xét văn phạm G_1

- $abA \Rightarrow abbA \{ \alpha=ab, \beta=\epsilon, \nu=A, w=bA, A \rightarrow bA \in P \}$
- $abA \Rightarrow abbA \Rightarrow abbbA \Rightarrow abbbbA \Rightarrow abbbbc$
 $\{ \alpha_0 = abA, \alpha_1 = abbA, \dots, \alpha_4 = abbbbc \}$

Vậy

$abA \Rightarrow^* abbbbc$

$abA \Rightarrow^+ abbbbc$

$abA \Rightarrow^* abA // \leftarrow$ Áp dụng 0 bước suy dẫn

$abA \Rightarrow^+ abA // \leftarrow$ Sai, do áp dụng ít nhất 1 suy dẫn

Câu

Cho văn phạm $G = (V_T, V_N, P, S)$

- Một xâu δ được suy ra từ ký hiệu khởi đầu S , được gọi là *dạng câu* hay *dạng cú pháp*
 - δ là một dạng câu nếu $S \Rightarrow^* \delta$
- Câu là một dạng cú pháp chỉ bao gồm toàn ký hiệu kết thúc
 - δ là một câu nếu $S \Rightarrow^* \delta$ và $\delta \in V_T^*$

Câu → Ví dụ

Ví dụ văn phạm G_1 $S \Rightarrow aSA \Rightarrow abA \Rightarrow abc$ $S \Rightarrow aSA \Rightarrow abA \Rightarrow abbA \Rightarrow abbc$ $S \Rightarrow aSA \Rightarrow abA \Rightarrow abbA \Rightarrow abbbA \Rightarrow abbbc$ Dạng câu: $aSA, abA, abc, abbA, abbc, \dots$ Câu: $abc, abbc, abbbc, abbbb, \dots$ **aabbcc** có phải là một câu?

Ngôn ngữ sản sinh

Cho văn phạm $G = (V_T, V_N, P, S)$

- Ngôn ngữ L được *sinh ra* từ văn phạm G , ký hiệu $L(G)$ là tập tất cả các câu của văn phạm
 - $L(G) = \{\delta \mid \delta \in V_T^* \text{ và } S \Rightarrow^* \delta\}$
- Ví dụ văn phạm G_1
 - $L(G_1) = \{b, abc, abbc, abbbc, \dots, aabbcc, \dots\}$
- $G_2 = (\{a,b\}, \{S,A\}, \{S \rightarrow aA, A \rightarrow a, A \rightarrow b\}, S)$
 - $L(G_2) = \{aa, ab\}$

Quy ước

Nếu $A \rightarrow \alpha_1$

$A \rightarrow \alpha_2$

.....

$A \rightarrow \alpha_n$

Được viết gọn lại thành $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$

Ví dụ:

$G_1 = (\{a, b, c\}, \{S, A\}, \{S \rightarrow aSA|b, A \rightarrow bA|c\}, S)$

$G_2 = (\{a, b\}, \{S, A\}, \{S \rightarrow aA, A \rightarrow a|b\}, S)$

Ngôn ngữ → Ví dụ

Cho văn phạm $G = (V_T, V_N, P, S)$

- $V_T = \{Mơ, Mận, thì, uống, nhanh, đẹp\}$
- $V_N = \{<\text{Câu}>, <\text{Danh từ}>, <\text{Động từ}>, <\text{Tính từ}>\}$
- $S : \text{Câu}$
- $P = \{ \quad <\text{Câu}> \rightarrow <\text{Danh từ}><\text{Động từ}><\text{Tính từ}>, \\ <\text{Danh từ}> \rightarrow Mơ \mid Mận, \\ <\text{Động từ}> \rightarrow uống \mid \text{thì}, \\ <\text{Tính từ}> \rightarrow nhanh \mid \text{đẹp} \}$

$L(G) = \{Mơ \text{ uống nhanh}, Mơ \text{ thì nhanh}, Mơ \text{ thì đẹp},$
 $\text{Mơ uống đẹp}, \text{Mận thì nhanh}, \dots\}$

Đệ quy

Cho văn phạm $G = (V_T, V_N, P, S)$

$A \in V_N // \leftarrow A$ là một ký hiệu không kết thúc

- A được gọi là đệ quy nếu tồn tại

$A \Rightarrow^+ \alpha A \beta$, với $\alpha, \beta \in V^*$

- Nếu $\alpha = \varepsilon$, $(A \Rightarrow^+ A \beta)$ thì gọi là *đệ quy trái*

– Nếu $A \Rightarrow A \beta$: Đệ quy **trái** trực tiếp (Sản xuất: $A \rightarrow A \beta$)

- Nếu $\beta = \varepsilon$, $(A \Rightarrow^+ \alpha A)$ thì gọi là *đệ quy phải*

– Nếu $A \Rightarrow \alpha A$: Đệ quy **phải** trực tiếp (Sản xuất: $A \rightarrow \alpha A$)

- Nếu $\alpha, \beta \neq \varepsilon$, $(A \Rightarrow^+ \alpha A \beta)$ thì gọi là *đệ quy trong*

– Nếu $A \Rightarrow \alpha A \beta$: Đệ quy trong trực tiếp (Sản xuất: $A \rightarrow \alpha A \beta$)

Ví dụ, định nghĩa Tên

$<\text{Tên}> \rightarrow <\text{Chữ cái}> | <\text{Tên}> <\text{Chữ cái}> | <\text{Tên}> <\text{Chữ số}>$

Văn phạm tương đương

Hai văn phạm là tương đương, nếu chúng cùng sinh ra một ngôn ngữ

$$G_1 = ({}^1V_T, {}^1V_N, P_1, S_1)$$

$$G_2 = ({}^2V_T, {}^2V_N, P_2, S_2)$$

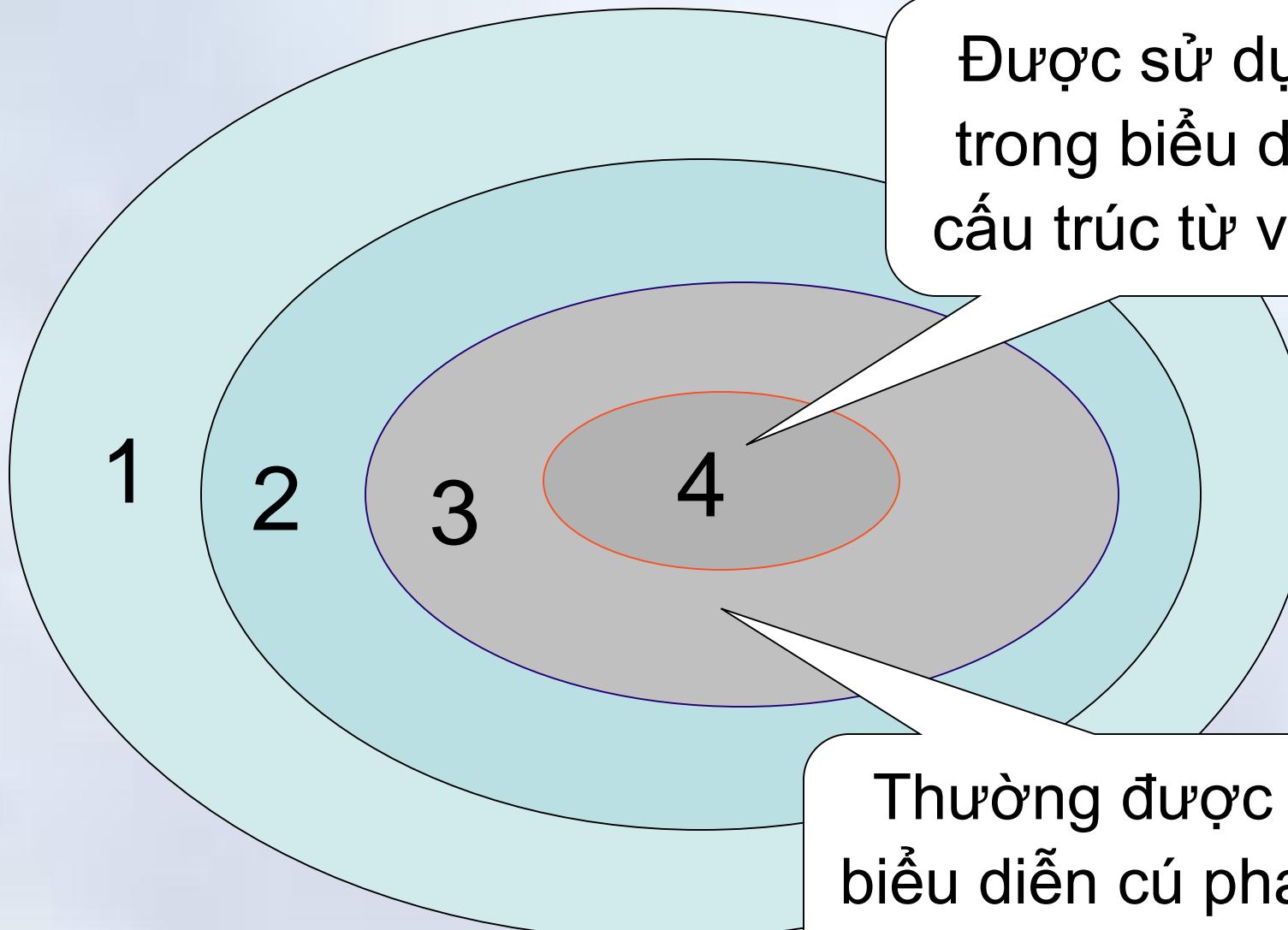
$$L(G_1) = L(G_2) \Rightarrow G_1 \Leftrightarrow G_2$$

Phân loại văn phạm

Phân thành 4 loại [Chomsky, 1957]

1. Văn phạm ngữ cầu (*phrase structure grammar*)
 1. **Ràng buộc:** Không có ràng buộc ngoài đ/nghĩa
2. Văn Phạm cảm ngữ cảnh (*context sensitive*)
 - **Ràng buộc:** Nếu $\alpha \rightarrow \beta$ thì $l(\alpha) \leq l(\beta)$; $S \rightarrow \epsilon$ được chấp nhận khi S không là vế phải của SX bất kỳ
3. Văn phạm phi ngữ cảnh (*context free grammar*)
 - **Ràng buộc:** Vế trái của các sản xuất là một ký hiệu không kết thúc
4. Văn phạm chính quy (*regular grammar*)
Ràng buộc: SX có dạng $A \rightarrow a|aB$ hoặc $A \rightarrow a|Ba$

Phân loại văn phạm



Chương 1: Những khái niệm cơ bản

1. Ngôn ngữ lập trình cấp cao và trình dịch
2. Đặc trưng của ngôn ngữ lập trình cấp cao
3. Các giai đoạn chính của chương trình dịch
4. Khái niệm ngôn ngữ
5. Văn phạm phi ngữ cảnh
6. Giới thiệu ngôn ngữ PL/0 mở rộng

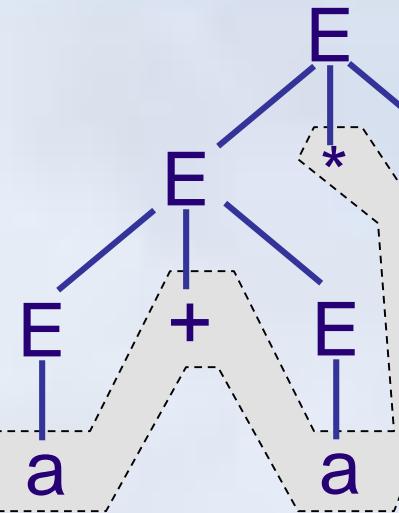
Văn phạm phi ngũ cành

- Cây suy dẫn
- Suy dẫn trái, suy dẫn phải
- Văn phạm đơn nghĩa
- Sản xuất đệ quy
- Văn đề phân tích cú pháp

Cây suy dẫn

- $G = (V_T, V_N, P, S)$ là văn phạm phi ngữ cảnh
- Cây suy dẫn D cho VP G là cây có thứ tự, trong đó
 - Mỗi một nút có nhãn là ký hiệu trong tập $V_T \cup V_N \cup \{\epsilon\}$
 - Nhãn của nút gốc là S (*Start Symbol*)
 - Các nút lá có nhãn là *một ký hiệu kết thúc* hoặc ϵ
 - Nếu A là nút trong của D và X_1, X_2, \dots, X_n là các hậu duệ trực tiếp của A theo tự trái qua phải thì
 - $A \rightarrow X_1 X_2 \dots X_n$ là một sản xuất của P
 - Nếu $n=0$ ($X=\epsilon$) thì X là hậu duệ đơn và duy nhất của A và $A \rightarrow \epsilon$ là một sản xuất trong P ,
- **Biên (kết quả)** của cây suy dẫn: Từ thu được bằng cách nối các nút lá lại theo thứ tự từ trái qua phải
 - Rõ ràng, nếu α là biên của D thì $S \Rightarrow^* \alpha$

Cây suy dẫn → Ví dụ

Văn phạm: $E \rightarrow E+E \mid E^*E \mid (E) \mid a$ Xét xâu: $\omega = a + a^*a$ Thay thế các biến **trái nhất**:

$$E \Rightarrow E^*E \Rightarrow E+E^*E \Rightarrow a+E^*E \Rightarrow a+a^*E \Rightarrow a+a^*a$$

Thay thế các biến **phải nhất**:

$$E \Rightarrow E^*E \Rightarrow E^*a \Rightarrow E+E^*a \Rightarrow E+a^*a \Rightarrow a+a^*a$$

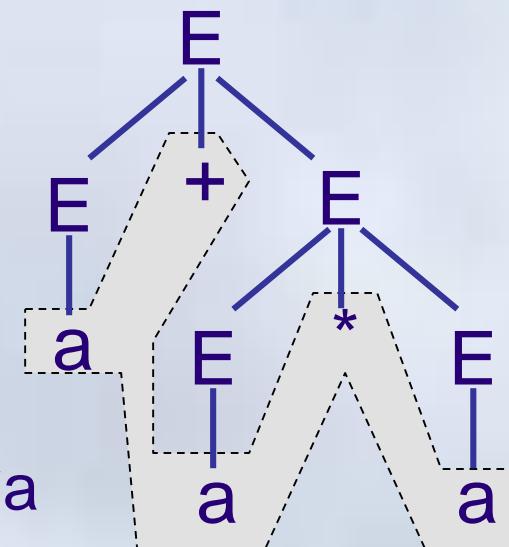
Khác nhau?

Thay thế các biến **trái nhất**:

$$E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E^*E \Rightarrow a+a^*E \Rightarrow a+a^*a$$

Thay thế các biến **phải nhất**:

$$E \Rightarrow E+E \Rightarrow E+E^*E \Rightarrow E+E^*a \Rightarrow E+a^*a \Rightarrow a+a^*a$$



Suy dẫn trái

Cho văn phạm $G = (V_T, V_N, P, S)$

- Gọi δ được suy dẫn *trực tiếp* ra từ bên trái của γ , và được ký hiệu $\gamma \Rightarrow_L \delta$ nếu tồn tại các xâu x, α, β thỏa mãn các điều kiện

- $\gamma = xA\alpha \Rightarrow x\beta\alpha = \delta$
- $A \rightarrow \beta \in P$, và
- $\alpha, \beta \in V^*$, $x \in V_T^*$

Luôn thay ký hiệu không kết thúc *bên trái nhất* của xâu

- Gọi δ được suy dẫn ra từ bên trái của γ nếu tồn tại dãy các xâu $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_n$ thỏa mãn

$$\gamma = \alpha_0 \Rightarrow_L \alpha_1 \Rightarrow_L \alpha_2 \Rightarrow_L \dots \Rightarrow_L \alpha_n = \delta$$

- Nếu $n \geq 1$, dùng ít nhất một suy dẫn, ký hiệu: $\gamma \Rightarrow_L^+ \delta$
- Nếu $n \geq 0$, có thể không suy dẫn nào, ký hiệu: $\gamma \Rightarrow_L^* \delta$

Suy dẫn phải

Cho văn phạm $G = (V_T, V_N, P, S)$

- Gọi δ được suy dẫn *trực tiếp* ra từ bên phải của γ , và được ký hiệu $\gamma \Rightarrow_R \delta$ nếu tồn tại các xâu y, α, β thỏa mãn các điều kiện

- $\gamma = \alpha A y \Rightarrow \alpha \beta y = \delta$
- $A \rightarrow \beta \in P$, và
- $\alpha, \beta \in V^*$, $y \in V_T^*$

Luôn thay ký hiệu không kết thúc *bên phải nhất* của xâu

- Gọi δ được suy dẫn ra từ bên phải của γ nếu tồn tại dãy các xâu $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_n$ thỏa mãn:

$$\gamma = \alpha_0 \Rightarrow_R \alpha_1 \Rightarrow_R \alpha_2 \Rightarrow_R \dots \Rightarrow_R \alpha_n = \delta$$

- Nếu $n \geq 1$, dùng ít nhất một suy dẫn, ký hiệu: $\gamma \Rightarrow_R^+ \delta$
- Nếu $n \geq 0$, có thể không suy dẫn nào, ký hiệu: $\gamma \Rightarrow_R^* \delta$

Văn phạm đơn nghĩa

Văn phạm PNC $G = (V_T, V_N, P, S)$ đơn nghĩa nếu

- Với mỗi câu $\omega \in L(G)$ chỉ có đúng một suy dẫn trái (*hoặc một suy dẫn phải*) \Leftrightarrow **Chỉ có đúng một cây suy dẫn**

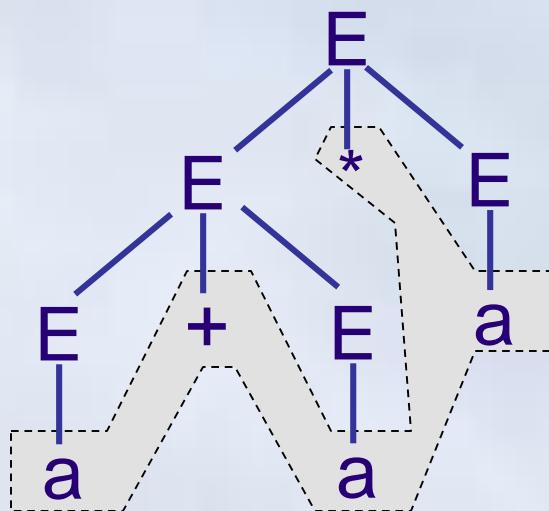
Ngược lại, nếu có nhiều hơn một suy dẫn trái, hoặc một suy dẫn phải hoặc có nhiều cây suy dẫn \Rightarrow Văn phạm nhập nhằng

Nếu $\omega \in L(G)$ có nhiều hơn 2 cây suy dẫn
 \Rightarrow Có thể phân tích cú pháp theo nhiều hơn 2 cách
 \Rightarrow Nhiều hơn 2 cách hiểu

Văn phạm đơn nghĩa → Ví dụ 1

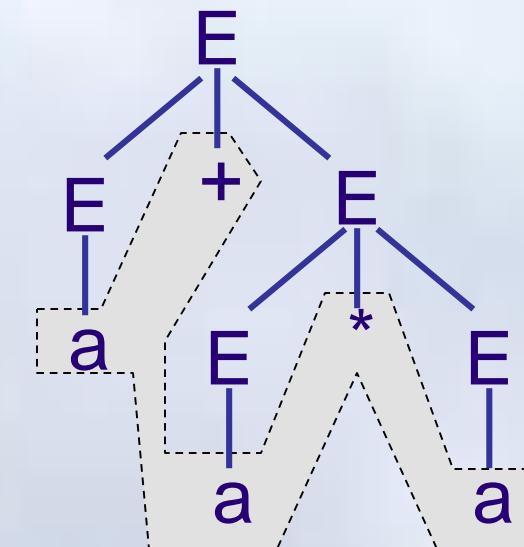
Văn phạm: $E \rightarrow E+E \mid E^*E \mid (E) \mid a$

Xâu: $\omega = a + a^* a$ là biến của 2 cây suy dẫn
 \Rightarrow Văn phạm nhập nhằng



$$a = 2$$

$$E = 8$$



$$E = 6$$

Văn phạm đơn nghĩa → Ví dụ 2

$$G = (V_T, V_N, P, S)$$

$V_T = \{\text{if, then, else}\}$

$V_N = \{\langle \text{Cond} \rangle, \langle \text{Stmt} \rangle\}$

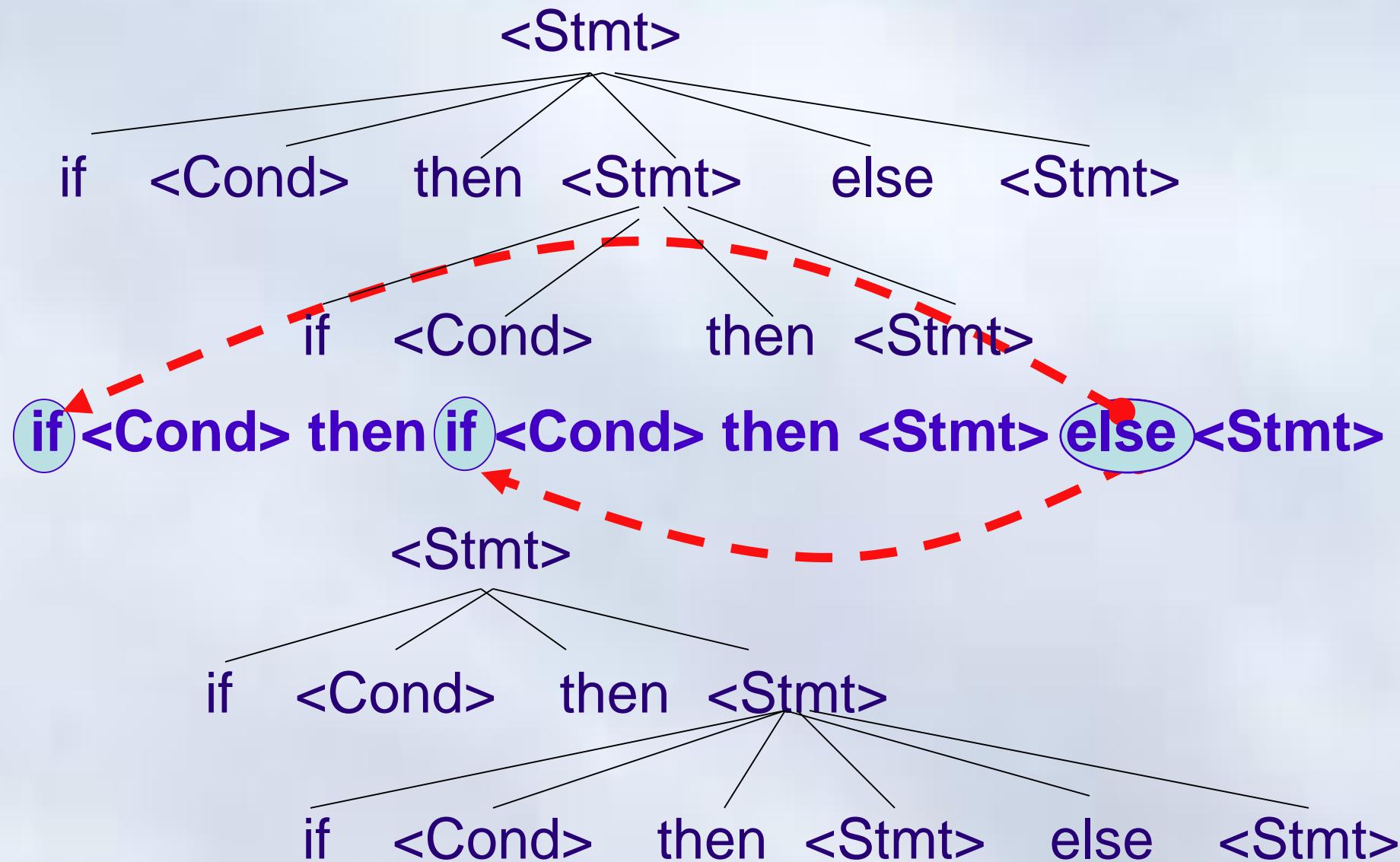
$S: \langle \text{Stmt} \rangle$

$P = \{ \langle \text{Stmt} \rangle \rightarrow \text{if } \langle \text{Cond} \rangle \text{ then } \langle \text{Stmt} \rangle$
 $\qquad \qquad \qquad \langle \text{Stmt} \rangle \rightarrow \text{if } \langle \text{Cond} \rangle \text{ then } \langle \text{Stmt} \rangle \text{ else } \langle \text{Stmt} \rangle \}$

Xét dạng câu

$\text{if } \langle \text{Cond} \rangle \text{ then if } \langle \text{Cond} \rangle \text{ then } \langle \text{Stmt} \rangle \text{ else } \langle \text{Stmt} \rangle$

Văn phạm đơn nghĩa → Ví dụ 2



Văn phạm đơn nghĩa → Khử nhập nhằng

Đưa thêm các ký hiệu không kết thúc và các sản xuất để được các văn phạm tương đương

$$E \rightarrow E+E \mid E^*E \mid (E) \mid a$$

$$E \rightarrow E+T \mid E^*T \mid T$$

$$T \rightarrow (E) \mid a$$

Luôn thực hiện từ trái qua
phải trừ khi gặp biểu thức
trong ngoặc

Phép nhân có độ ưu tiên cao
hơn phép cộng khi không
gặp biểu thức trong ngoặc

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

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

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

Sản xuất đệ quy

- Sản xuất là đệ quy có dạng:

$$A \rightarrow \alpha A \beta, \alpha, \beta \in V^*$$

- Dùng để biểu diễn các quá trình lặp hay cấu trúc lồng nhau

- Đệ quy trái: $A \rightarrow b|Aa$

$$A \Rightarrow Aa \Rightarrow Aaa \Rightarrow Aaaa \Rightarrow baaaa \dots$$

- Đệ quy phải: $A \rightarrow b|aA$

$$A \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow aaaab \dots$$

- Đệ quy giữa $A \rightarrow b|aAb$

VD: $A \rightarrow b|\{A\}$

$$A \Rightarrow \{A\} \Rightarrow \{\{A\}\} \Rightarrow \{\{\{A\}\}\} \Rightarrow \{\{\{b\}\}\}$$

Sản xuất đệ quy \rightarrow Khử đệ quy trái

Khử đệ quy trái bằng cách thêm ký hiệu không kết thúc và sản xuất mới để được văn phạm tương đương

$$\begin{aligned}
 E &\rightarrow E + T \mid T \\
 T &\rightarrow T^* F \mid F \\
 F &\rightarrow (E) \mid a
 \end{aligned}$$



$$\begin{aligned}
 E &\rightarrow T E' \\
 E' &\rightarrow + T E' \mid \epsilon \\
 T &\rightarrow F T' \\
 T' &\rightarrow * F T' \mid \epsilon \\
 F &\rightarrow (E) \mid a
 \end{aligned}$$

Vấn đề phân tích cú pháp (1/4)

- Cho văn phạm $G = (V_T, V_N, P, S)$
 $L(G) = \{\omega \mid \omega \in V_T^* \text{ và } S \Rightarrow^* \omega\}$
- Nhiệm vụ của phân tích cú pháp là xem xét
 một xâu có thuộc ngôn ngữ được sản sinh
 bởi văn phạm không
 Cho $x \in V_T^*$; $S \Rightarrow^* x$?
- Nếu xâu x được đoán nhận, cần chỉ ra các
 sản xuất đã sử dụng để sinh ra
 - Cấu trúc lén cây suy dẫn
- **Tồn tại 2 phương pháp trên xuống/dưới lên**

Vấn đề phân tích cú pháp (2/4)

- Trên xuống (Top-down)
 - Xuất phát từ ký hiệu khởi đầu S đi dần tới nút lá (xâu cần phân tích), bằng cách áp dụng liên tục các sản xuất
- Dưới lên (Bottom-up)
 - Xâu cần phân tích được xem xét từ trái qua phải và tìm cách thu gọn về ký hiệu khởi đầu S bằng cách áp dụng các sản xuất

Vấn đề phân tích cú pháp (3/4)

Xét văn phạm:

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

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

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

Xâu cần phân tích ω : $a+a^*a$

Top-Down

$$\begin{aligned} E \Rightarrow E+T &\Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+T^*F \Rightarrow \\ &a+F^*F \Rightarrow a+a^*F \Rightarrow a+a^*a \end{aligned}$$

Bottom-Up

$$\begin{aligned} a+a^*a &\Leftarrow F+a^*a \Leftarrow T+a^*a \Leftarrow E+a^*a \Leftarrow E+F^*a \Leftarrow \\ &E+T^*a \Leftarrow E+T^*F \Leftarrow E+T \Leftarrow E \end{aligned}$$

Vấn đề phân tích cú pháp (4/4)

- Phân tích quay lui

- Tại mỗi bước có nhiều sản xuất để lựa chọn
 - Nếu lựa chọn nhầm thực hiện quay lui

Phân tích quay lui tăng thời gian thực hiện

- Phân tích tất định

- Tại mỗi bước xác định duy nhất một sản xuất
 - Yêu cầu văn phạm cần có tính chất nhất định
 - Các kỹ thuật
 - Trên xuống: Phân tích LL(k), Đệ quy trên xuống,...
 - Dưới lên: Thao tác trước, Phân tích LR(k),...

Chương 1: Những khái niệm cơ bản

1. Ngôn ngữ lập trình cấp cao và trình dịch
2. Đặc trưng của ngôn ngữ lập trình cấp cao
3. Các giai đoạn chính của chương trình dịch
4. Khái niệm ngôn ngữ
5. Văn phạm phi ngữ cảnh
6. Giới thiệu ngôn ngữ PL/0 mở rộng

Dạng chuẩn BNF

- Siêu ngữ (metalanguage)
 - Ngôn ngữ sử dụng các lệnh để mô tả ngôn ngữ khác
- *Dạng chuẩn BNF*
 - *Backus-Normal-Form/Backus-Naur-Form*
 - Là dạng siêu cú pháp để mô tả các ngôn ngữ lập trình
 - BNF được sử dụng rộng rãi để mô tả văn phạm của các ngôn ngữ lập trình, tập lệnh và các giao thức truyền thông.

Ký pháp BNF

Là một tập các luật thỏa mãn:

- Về trái của mỗi luật là một cấu trúc cú pháp.
- Tên cấu trúc cú pháp là ký hiệu không kết thúc.
- Ký hiệu không kết thúc được bao trong cặp `<>`
- Ký hiệu kết thúc được phân cách bằng cặp nháy đơn hoặc nháy kép
- Mỗi ký hiệu không kết thúc được định nghĩa bằng một hay nhiều luật.
- Các luật có dạng **N ::= S**
 - N là ký hiệu không kết thúc,
 - s là một xâu gồm 0 hay nhiều ký hiệu kết thúc và không kết thúc.
- Các luật có chung về trái được phân cách bằng /

Ký pháp BNF → Ví dụ

- Mô tả cách viết một số nguyên, thực dấu phẩy tĩnh

```
<Số> ::= '-'<Số thập phân>|<số thập phân>
<Số thập phân> ::= <Dãy chữ số>|<Dãy chữ số>'.<Dãy chữ số>
<Dãy chữ số> ::= <Chữ số>|<Chữ số><Dãy chữ số>
<Chữ số> ::= '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
```

Ký pháp EBNF (Extended BNF)

Được phát triển từ BNF, có ký pháp tương tự nhưng được đơn giản hóa bằng cách dùng một số ký hiệu đặc biệt :

- Không cần dùng ‘ ’ cho ký hiệu kết thúc
- Dùng [] để chỉ phần bên trong là tùy chọn (có hoặc không)
- Dùng { } phần bên trong có thể lặp lại một số lần tùy ý hoặc không xuất hiện lần nào
 - Nếu lặp lại m hay n lần , dùng m hay n là chỉ số trên hoặc dưới
- Dùng (|) cho biết có thể lựa chọn 1 trong các ký hiệu nằm bên trong

Ký pháp EBNF → Ví dụ

- $S \rightarrow a \{b\}$ $\Leftrightarrow S \rightarrow a | ab | abb | \dots$
- $S \rightarrow [a]\alpha$ $\Leftrightarrow S \rightarrow \alpha | a \alpha$
- $S \rightarrow (a|b|c)\alpha$ $\Leftrightarrow S \rightarrow a\alpha | b\alpha | c\alpha$

Trong BNF

$\langle \text{Lệnh if} \rangle ::= \text{'IF'} \langle \text{Biểu thức} \rangle \text{'THEN'} \langle \text{Lệnh} \rangle | \text{'IF'} \langle \text{Biểu thức} \rangle \text{ THEN} \langle \text{Lệnh} \rangle \text{'ELSE'} \langle \text{Lệnh} \rangle$

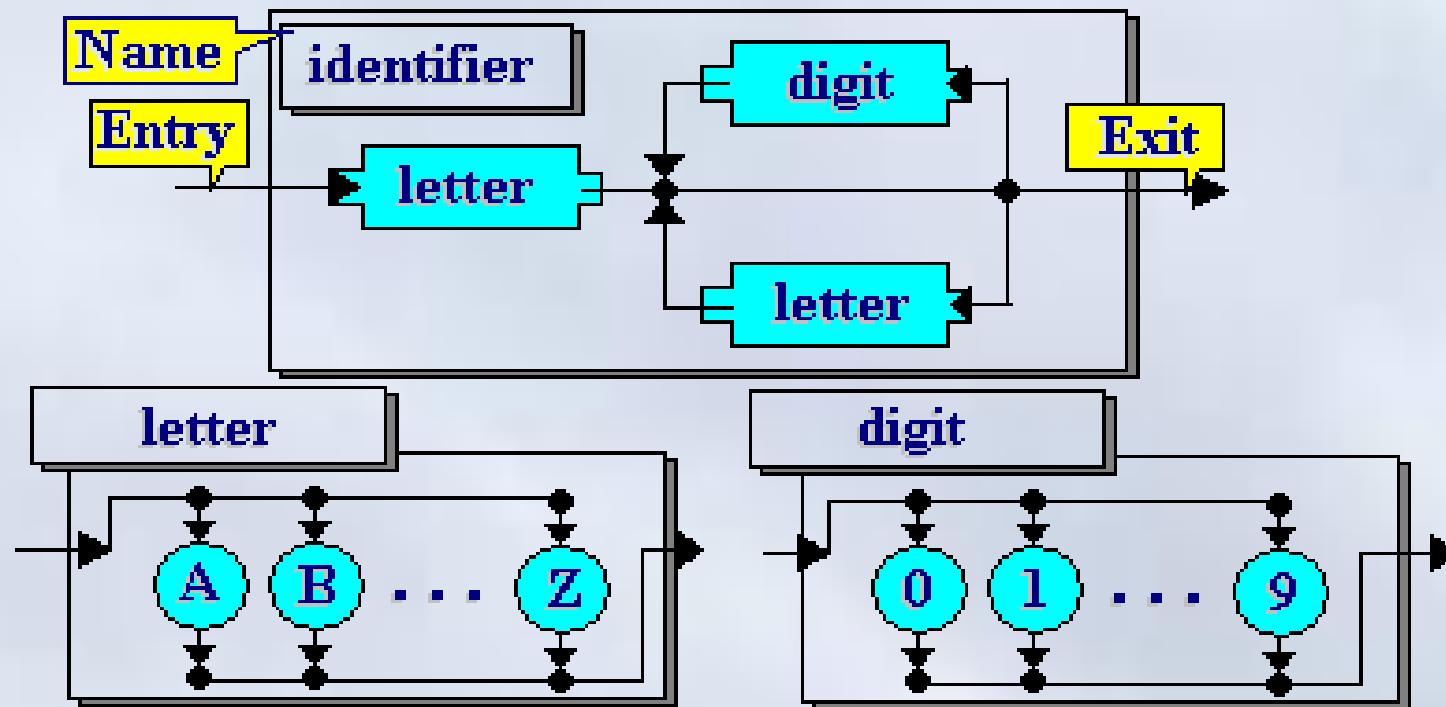
BNF >< EBNF

Trong EBNF

$\langle \text{Lệnh if} \rangle ::= \text{IF} \langle \text{Biểu thức} \rangle \text{ THEN} \langle \text{Lệnh} \rangle [\text{ELSE} \langle \text{Lệnh} \rangle]$

Sơ đồ cú pháp

- Là công cụ để mô tả cú pháp của ngôn ngữ lập trình dưới dạng đồ thị
- Mỗi sơ đồ cú pháp là một đồ thị định hướng với lối vào và lối ra xác định.
- Mỗi sơ đồ cú pháp có một tên duy nhất



Ngôn ngữ PL/0

- Được giới thiệu bởi Niklaus Wirth năm 1975
 - Trong quyển Algorithms + Data Structures = Programs
 - Là ngôn ngữ lập trình phục vụ giáo dục
 - Tựa Pascal, chứa đặc trưng của một NNLT cấp cao

Đặc trưng cơ bản

- **Từ vựng:**
 - **Chữ cái:** a..z, A..Z ; **Chữ số:** 0..9
 - Dấu đơn: + - * / % () [] > < = , ; .
 - Dấu kép: := >= <= <>
 - Từ khóa: 15 *từ khóa* : Begin, Call, Const, Do, End, Else, For, If, Odd, Procedure, Program, Then, To, Var, While
 - Tên: Bắt đầu bởi chữ cái, tiếp theo là tổ hợp chữ cái /chữ số. Độ dài tối đa 10. Nếu vượt quá sẽ bị bỏ qua

Ngôn ngữ PL/0

Đặc trưng cơ bản (tiếp)

- Hỗ trợ kiểu dữ liệu cơ bản *kiểu nguyên*
 - Nếu hằng số nguyên có hơn 6 chữ ⇒ Báo lỗi **số quá lớn**
 - PL/0 mở rộng có cấu trúc *mảng một chiều* các số nguyên
- Biểu thức số học, biểu thức so sánh
- Cho phép định nghĩa hằng (**const**), biến (**var**)
- Câu lệnh gán (***:=***) , gộp (**begin end**), gọi thủ tục (**call**), rẽ nhánh (**if..else**), lặp (**while, for**)
- Câu lệnh vào/ra (*read/readln/write/writeln*)
- Có cấu trúc khối – chương trình con dạng thủ tục
 - Các thủ tục có thể lồng nhau
 - PL/0 mở rộng cho phép truyền tham số theo trị, theo biến

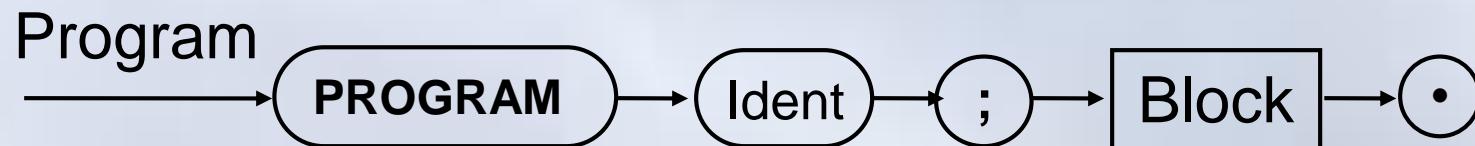
Sơ đồ cú pháp cho ngôn ngữ PL/0 mở rộng

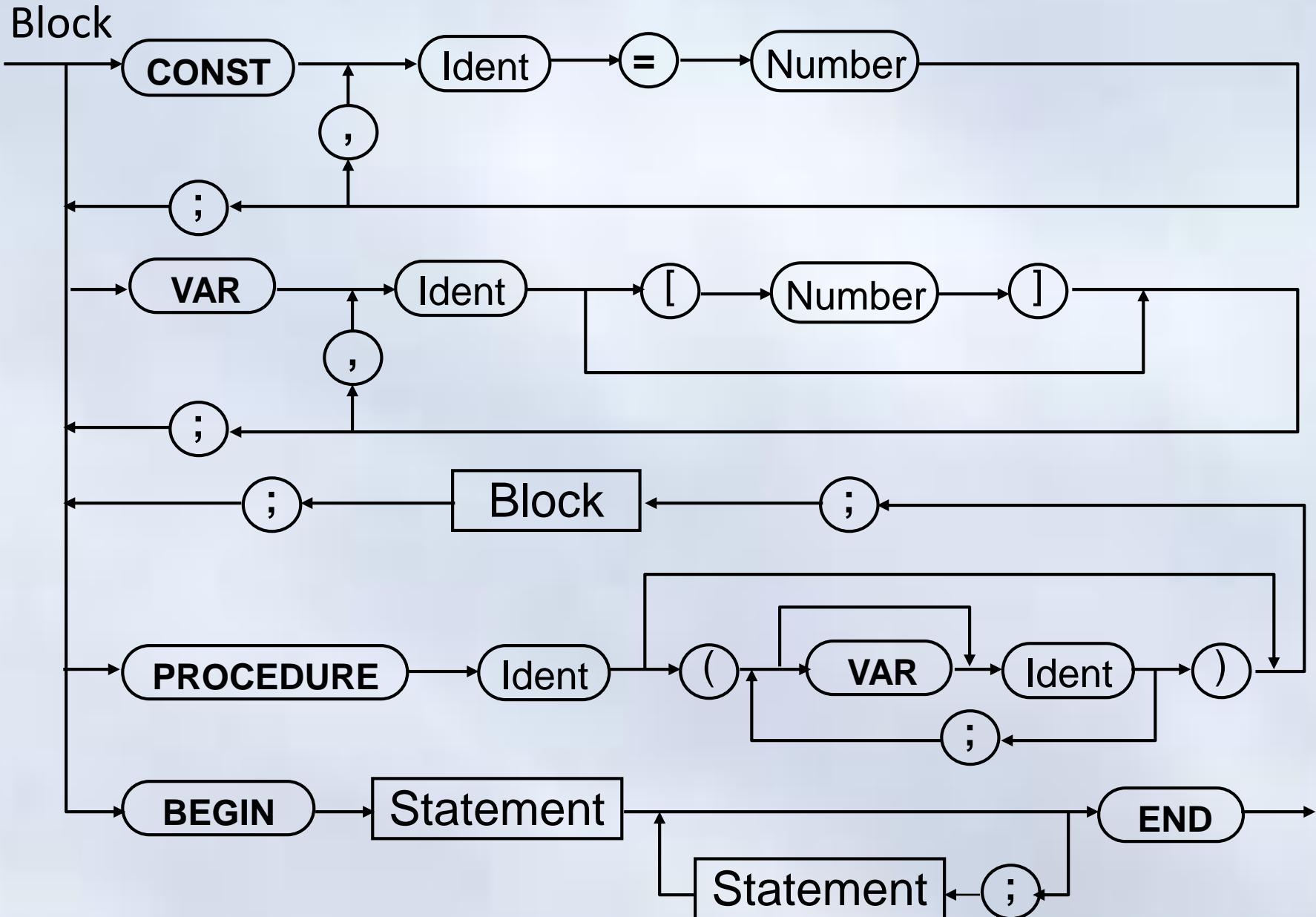


Dạng EBNF

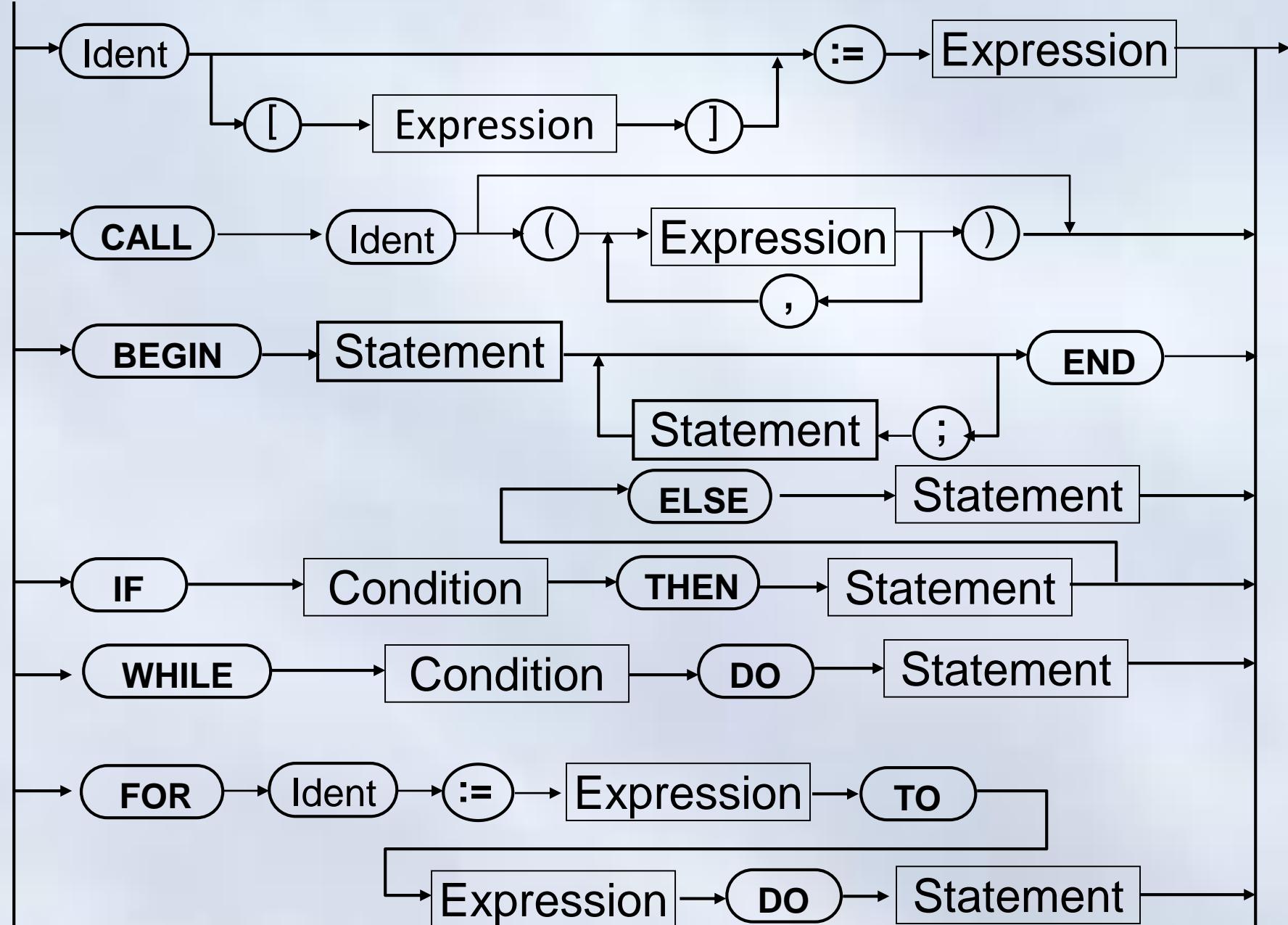
`<program> ::= PROGRAM <Identifier> ; <Block>.`

Dạng sơ đồ

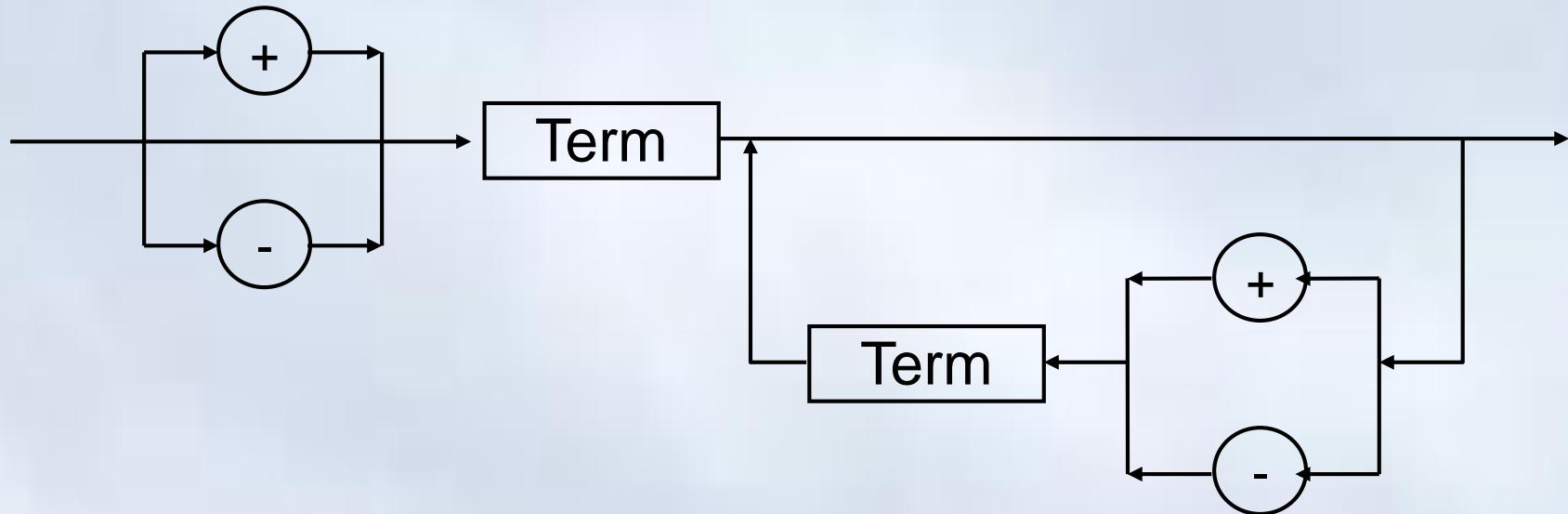




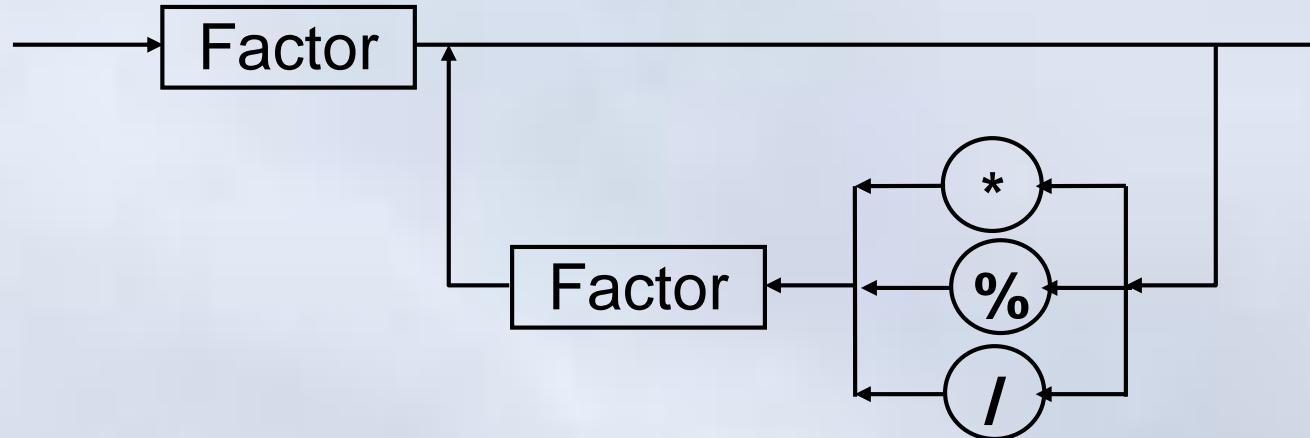
Statement



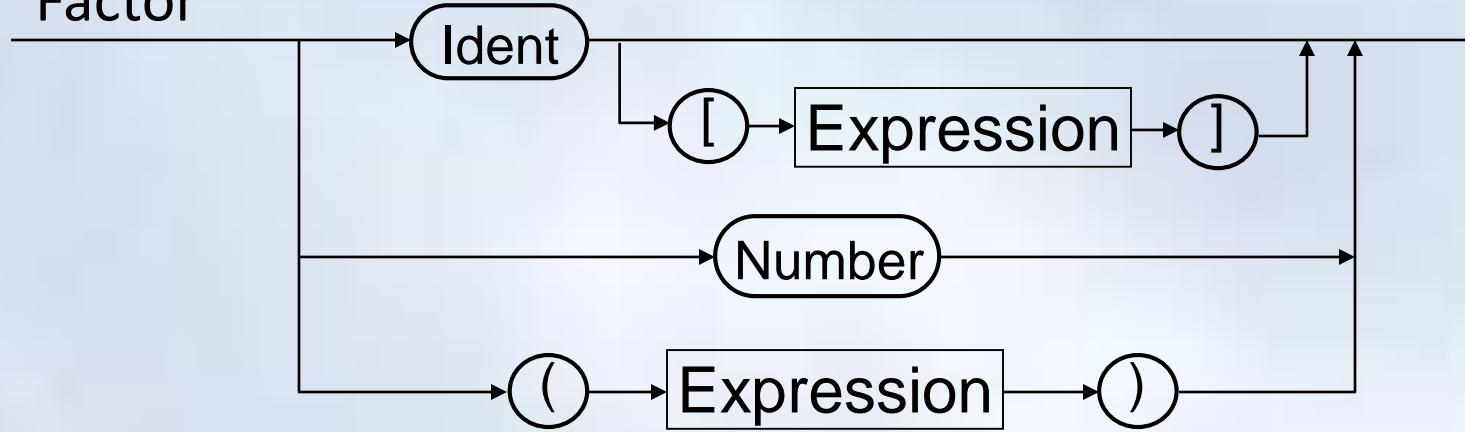
Expression



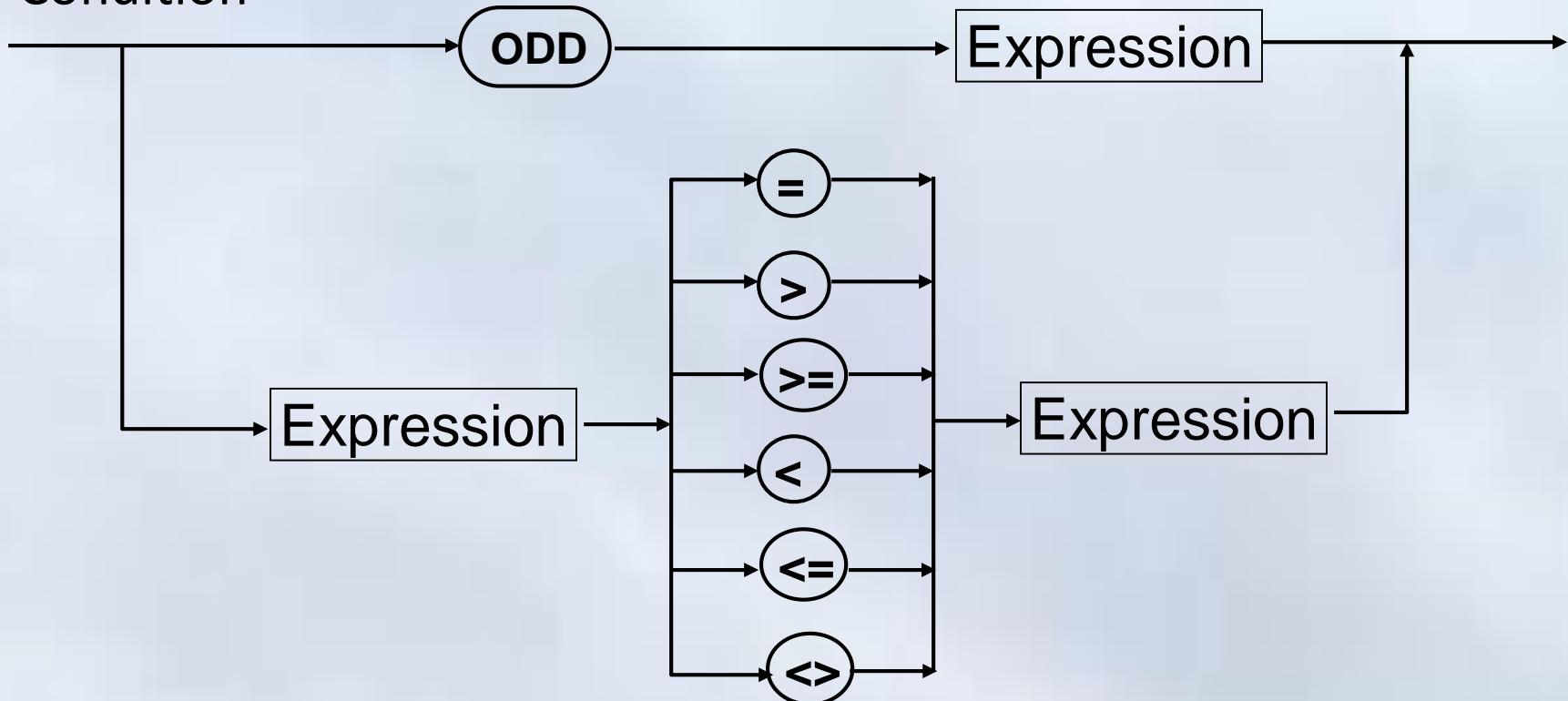
Term



Factor



Condition



Ví dụ viết trên ngôn ngữ PL/0 mở rộng

Program Example2;

Var a,b,c;

Procedure gcd(a; b; var d);

Begin

 While a <> b do

 if a > b then a := a – b

 else b := b – a;

 d := b;

End;

Begin

 Call ReadIn(a); Call ReadIn(b);

 Call gcd(a,b,c);

 Call Writeln(c);

End.

Phân tích một chương trình PL/0 mở rộng

```
Program Vidu;  
Begin  
    X := 10  
End.
```

$\omega = \text{Program Vidu ; Begin } X := 10 \text{ End .}$

$L(\omega) = 9$

$\textcolor{red}{<\text{program}>} \Rightarrow^* \omega?$

Phân tích một chương trình PL/0 mở rộng

<program>

- ⇒ **Program** Ident ; <block> .
- ⇒ **Program** Vidu ; <block>.
- ⇒ **Program** Vidu; **Begin** <Statement> **End**.
- ⇒ **Program** Vidu; **Begin** Ident := <Expression> **End**.
- ⇒ **Program** Vidu; **Begin** X := <Expression> **End**.
- ⇒ **Program** Vidu; **Begin** X := <Term> **End**.
- ⇒ **Program** Vidu; **Begin** X := <Factor> **End**.
- ⇒ **Program** Vidu; **Begin** X := Number **End**.
- ⇒ **Program** Vidu; **Begin** X := 10 **End**

Ident(Vidu), Ident (X),
Number(10) được xác định
bởi bộ phân tích từ vựng

IT4079:NGÔN NGỮ và PHƯƠNG PHÁP DỊCH

Phạm Đăng Hải

haipd@soict.hust.edu.vn

Chương 2: Phân tích từ vựng

1. Nhiệm vụ của bộ phân tích từ vựng
2. Biểu thức chính quy
3. Ô tô mát hữu hạn
4. Phân tích từ vựng của ngôn ngữ PL/0

Mục đích & Nhiệm vụ

- Mục đích:
 - Tìm chuỗi **dài nhất** các ký tự đầu vào, bắt đầu từ ký tự hiện tại tương ứng với một từ tố và trả về từ tố này
- Nhiệm vụ
 - Duyệt từng ký tự của văn bản nguồn
 - Loại bỏ các ký tự không cần thiết như dấu cách, chú thích,..
 - Xây dựng từ vựng từ những ký tự đọc được
 - Nhận dạng từ tố và gửi tới pha tiếp

Nhận biết từ tố gồm

- Nhận biết các từ khóa, tên do người dùng định nghĩa
- Nhận biết các con số, hằng chuỗi, hằng ký tự
- Nhận biết các ký tự đặc biệt (+, *, ...), ký hiệu kép (:=, !=, ...)

Từ vựng và Từ tố

- **Từ vựng (Lexeme)**

- Là đơn vị nhỏ nhất trong ngôn ngữ lập trình
 - Được coi là ký hiệu của một bảng chữ của ngôn ngữ
- Được xây dựng từ các ký tự ASCII

- **Từ tố (Token)**

- Là thuật ngữ dùng chỉ các từ vựng có cùng ý nghĩa cú pháp
 - Có thể coi từ vựng là những từ cụ thể trong từ điển: “*hôm nay*”, “*trời*”, “*đẹp*”; còn từ tố là loại từ: “*trạng từ*”, “*danh từ*”, “*tính từ*”,..

Từ tố → Ví dụ

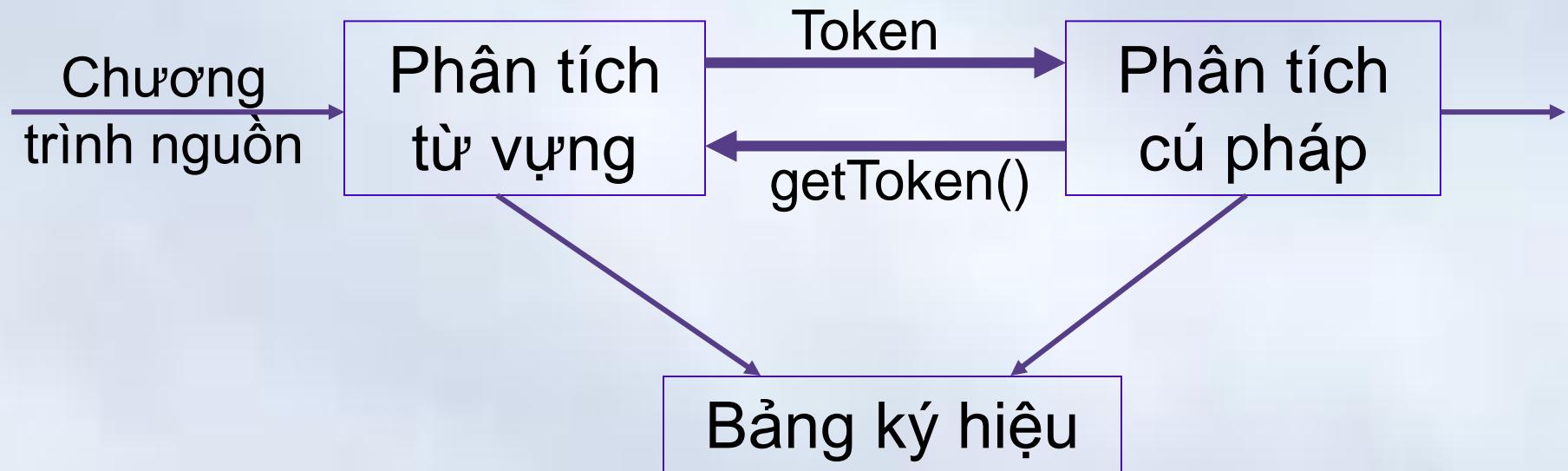
```
pos := start + 10 * size;
```

- “**pos**”, “**start**”, “**size**”, “**+**”, “**10**”, “*****”, “**:=**”, “**;**” là từ vựng
- “**pos**”, “**start**”, “**size**”, → các từ vựng thuộc lớp từ tố tên (*ident*)
- “**:=**” → từ vựng của từ tố gán (*assign*)
- “**10**” → từ vựng của từ tố số nguyên (*number*)
- “**+**” → từ vựng của từ tố cộng (*plus*)
- “*****” → từ vựng của từ tố nhân (*times*)
- “**;**” → từ vựng của từ tố chấm phẩy (*semicolon*)

Từ tố → Chú ý

- Các từ tố *Ident, number, plus, assign,...* do người viết trình dịch tự định nghĩa để dễ dàng cho việc mã hóa chương trình. Đây là việc số hóa ký hiệu
- Một từ tố có thể ứng với tập các từ vựng khác nhau nên cần thêm một số thông tin khác để biết được cụ thể đó là từ vựng nào
 - Các chuỗi “**19**”, “**365**” đều là chuỗi số, có từ tố “**number**”, nhưng khi sinh mã cần phải biết rõ giá trị là **19** hay **365**
- Bộ phân tích từ vựng không chỉ nhận dạng được các từ tố mà còn phải biết thuộc tính tương ứng
 - Từ tố tác động đến bộ phân tích cú pháp
 - Thuộc tính sử dụng trong bộ sinh mã

Thực hiện



- Thực hiện lắp dựa vào yêu cầu từ bộ ptcp
 - Bộ ptcp khi cần một từ tố sẽ gọi **getToken()**
 - Nhận được y/cầu, bộ pttv sẽ đọc các ký tự cho tới khi xây dựng xong từ vựng và nhận ra từ tố hoặc gặp lỗi
- Thường bộ pttv được chia thành 2 phần chính
 - Đọc ký tự
 - Xây dựng từ vựng và nhận dạng từ tố

Mẫu (Pattern)

- Là luật để mô tả một từ tố nào đó
 - Cơ sở phân biệt & nhận dạng các từ tố khác nhau
 - Chuỗi ký tự cùng thỏa mãn một luật \Rightarrow có cùng một từ tố
 - Từ tố là tên riêng của một luật mô tả, từ vựng là một trường hợp thỏa mãn luật
- Ví dụ
 - Luật mô tả của từ tố ***Ident***
 - Bắt đầu là một chữ cái
 - Tiếp theo là tổ hợp chữ cái, chữ số
 - Luật mô tả của từ tố ***assign***
 - Bắt đầu bởi ký tự “:”, ngay sau đó là ký tự “=”
 - Luật được mô tả bởi **biểu thức chính quy**

Chương 2: Phân tích từ vựng

1. Nhiệm vụ của bộ phân tích từ vựng
2. Biểu thức chính quy
3. Ô tô mát hữu hạn
4. Phân tích từ vựng của ngôn ngữ PL/0

Giới thiệu

- **Ngôn ngữ:** Tập hợp của các câu/ xâu (string)
- **Câu:** Dãy hữu hạn của các từ/ký hiệu (symbol)
- **Từ:** Được tạo nên từ một bộ chữ hữu hạn

Ví dụ:

- Ngôn ngữ C là tập các câu lệnh tạo nên các chương trình C hợp lệ
- Bộ chữ cho ngôn ngữ C là tập các chữ cái ASCII
- Một ngôn ngữ có thể là vô hạn, hoặc hữu hạn
- Một ngôn ngữ (*có thể vô hạn*) có thể được mô tả hữu hạn nhờ sử dụng biểu thức chính quy :
 - Mỗi biểu thức đặc trưng cho một tập câu/xâu
 - Chỉ xét xâu có thuộc ngôn ngữ không, chưa xét ý nghĩa của xâu

Biểu thức chính quy (regular expression)

Cho Σ là một bảng chữ của một ngôn ngữ.

- \emptyset là biểu thức chính quy biểu diễn ngôn ngữ \emptyset
- ε là biểu thức chính quy biểu diễn ngôn ngữ $\{\varepsilon\}$
- $\forall a \in \Sigma$, a là biểu thức chính quy biểu diễn tập $\{a\}$
- Nếu r và s là các biểu thức chính quy biểu diễn các tập xâu R và S tương ứng thì $(r + s)$, $(r.s)$, (r^*) là các biểu thức chính quy biểu diễn các tập xâu $R \cup S$, RS và R^* tương ứng.

Ngôn ngữ được xác định bởi biểu thức chính quy e , ký hiệu là $L(e)$ là **ngôn ngữ chính quy**

Biểu thức chính quy → Ghi chú

- Biểu thức $(r + s)$ có thể được viết $(r | s)$;
- Biểu thức $(r.s)$ thành (rs)
- Có thể bỏ qua ký hiệu ϵ
 - $(a |) \Leftrightarrow (a | \epsilon)$ // cũng có thể viết a ?
- Có thể bỏ ngoặc đơn bởi định nghĩa thứ tự ưu tiên
 - Phép đóng Kleene (*) ưu tiên hơn phép ghép(.)
 - Phép ghép(.) ưu tiên hơn phép hoặc (+)
 - Ví dụ: $(0+(1(0^*))) = (0+(10^*)) = (0+10^*) = 0+10^*$

Biểu thức chính quy → Ví dụ

Cho $\Sigma = \{a, b\}$ một bảng chữ.

$$- e_1 = a^* + b^* \Rightarrow L(e_1) = \{\epsilon, a, aa, aaa, \dots, b, bb, \dots\}$$

$$- e_2 = a^* b^* \Rightarrow L(e_2) = \{\epsilon, a, b, aa, ab, bb, aaa, \dots\}$$

$$- e_3 = a(a+b)^*$$

$$\Rightarrow L(e_3) = \{a, aa, ab, aaa, aab, aba, abb, \dots\}$$

- Xâu có dạng: bắt đầu là ký hiệu a, tiếp theo là tổ hợp bất kỳ của các ký hiệu a, b
- Nếu a là một chữ cái, b là chữ số

$\Rightarrow L(e_3)$ là ngôn ngữ chứa các tên

$\Rightarrow e_3$ biểu thức chính quy sinh mô tả một tên

Tính chất đại số của btcq

- 2 biểu thức chính quy là tương đương nếu cùng xác định một ngôn ngữ
- Nếu r, s, t là các biểu thức chính quy

$$- r + s = s + r$$

$$r + r = r$$

$$- (r + s) + t = r + s + t = r + (s + t)$$

$$- (r.s).t = r. s. t = r. (s. t)$$

$$- r.\varepsilon = \varepsilon.r = r$$

$$- r + \emptyset = \emptyset + r = r$$

$$- r(s+t) = rs + rt$$

$$(r+s)t = rt + st$$

$$- r + r^* = r^* ; \quad (r + \varepsilon)^* = r^* ; \quad (r^*)^* = r^*$$

$$- rr^* = r^*r = r +$$

$$- (r+s)^* = (r^*s^*)^*$$

Văn phạm chính quy và Ngôn ngữ chính quy

• Văn phạm chính quy

- Văn phạm mà mọi sản xuất có dạng

$A \rightarrow a|aB$ hoặc $A \rightarrow a|Ba$

- Dùng diễn tả từ vựng của NNLT

$<\text{Tên}> \rightarrow <\text{Chữ cái}> | <\text{Tên}> <\text{Chữ cái}> | <\text{Tên}> <\text{Chữ số}>$

$<\text{Tên}> \rightarrow "a" | "b" | "c" | \dots | "z" | "A" | "B" | \dots | "Z"$

$<\text{Chữ số}> \rightarrow "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"$

- Văn phạm chính quy sinh ra ngôn ngữ chính quy

• Ngôn ngữ chính quy

- Được biểu diễn (*mô tả*) bởi biểu thức chính quy
- Đoán nhận bởi các Otomat hữu hạn

Ngôn ngữ chính quy \rightarrow Văn phạm chính quy

r là biểu thức chính qui cần chuyển

1. Thêm ký hiệu khởi đầu S và tạo sản xuất $S \rightarrow r$
2. Loại bỏ khỏi văn phạm các siêu ký hiệu của r
 - \forall SX dạng $A \rightarrow r_1.r_2$: Thêm ký hiệu không kết thúc B và thay thành 2 SX:
$$A \rightarrow r_1 B \quad \& \quad B \rightarrow r_2$$
 - \forall SX dạng $A \rightarrow r_1+r_2$: Thay bởi $A \rightarrow r_1|r_2$
 - \forall SX dạng $A \rightarrow r_1^* r_2$: Thêm ký hiệu không kết thúc B và thay thành 4 sản xuất

$$A \rightarrow r_1 B; \quad A \rightarrow r_2$$

$$B \rightarrow r_1 B; \quad B \rightarrow r_2$$

Ví dụ : Chuyển đổi biểu thức $e = a(a+b)^*$

1. Thêm S và sản xuất S $\rightarrow a(a+b)^*$
2. Loại bỏ các ký hiệu không thuộc bộ chữ
 - Xét $r = a(a+b)^* // r_1 = a; r_2 = (a+b)^*$
 - Thêm A và các SX $S \rightarrow aA \& A \rightarrow (a+b)^*$
 - Xét $A \rightarrow (a+b)^* // r_1 = a+b ; r_2 = \epsilon$
 - Thêm B và các SX
 - $A \rightarrow (a+b)B \& A \rightarrow \epsilon \& B \rightarrow (a+b) \& B \rightarrow \epsilon$
 - Áp dụng luật phân phối phải
 - $A \rightarrow aB + bB \& A \rightarrow \epsilon \& B \rightarrow aB + bB \& B \rightarrow \epsilon$
 - $A \rightarrow aB|bB|\epsilon \& B \rightarrow aB|bB|\epsilon$

Ví dụ : Chuyển đổi biểu thức $e = a(a+b)^*$ (tiếp)

- Loại bỏ ký hiệu ϵ bởi tạo ra xâu mới

$B \rightarrow aB|bB| \epsilon$ thành $B \rightarrow aB|bB| a | b$

$A \rightarrow aB|bB| \epsilon$ thành $A \rightarrow aB|bB| a | b$

Vai trò của A và B là tương đương. Thay ký hiệu B bằng A và loại bỏ B

Kết quả: Văn phạm cuối

$S \rightarrow a | aA$

$A \rightarrow aA | bA | a | b$

Bài tập

- Tìm văn phạm chính quy tương với các biểu thức chính quy sau
 - $1(0+1)^*1$
 - $1+01^*$

Chương 2: Phân tích từ vựng

1. Nhiệm vụ của bộ phân tích từ vựng
2. Biểu thức chính quy
3. Ô tô mát hữu hạn
4. Phân tích từ vựng của ngôn ngữ PL/0

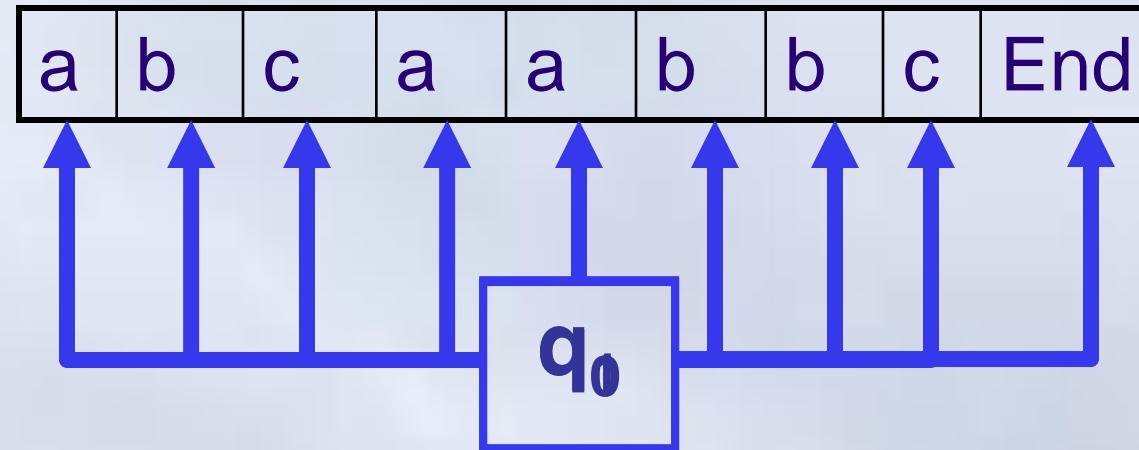
Mô tả

- Gồm một tập các trạng thái Q
 - Có một trạng thái đầu $q_0 \in Q$
 - Có một tập trạng thái kết thúc $F \subseteq Q$
 - Một bộ chữ vào Σ
 - Một tập các hàm dịch chuyển $\delta: (Q \times \Sigma) \rightarrow Q$
- Hoạt động**
- Ô-tô-mát xuất phát từ trạng thái đầu, đọc từng ký hiệu của xâu vào, chuyển trạng thái dựa trên trạng thái hiện thời và ký hiệu đọc được.
 - Sau khi đọc hết xâu vào mà ô-tô-mát ở trạng thái kết thúc, xâu được gọi là được đoán nhận bởi ô-tô-mát

Ví dụ

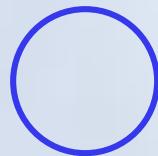
- $\Sigma = \{a, b, c\}$
- $Q = \{q_0, q_1\}$
- $q_0 = q_0$
- $F = \{q_1\}$

δ	a	b	c
q_0	q_1	q_0	q_0
q_1	q_0	q_1	q_1

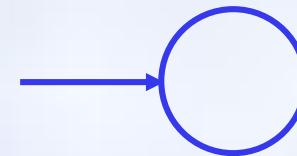


Xâu $abcaabbc$ được đoán nhận

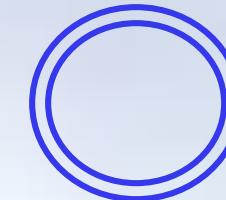
Biểu diễn ô tô mát hữu hạn



Trạng thái

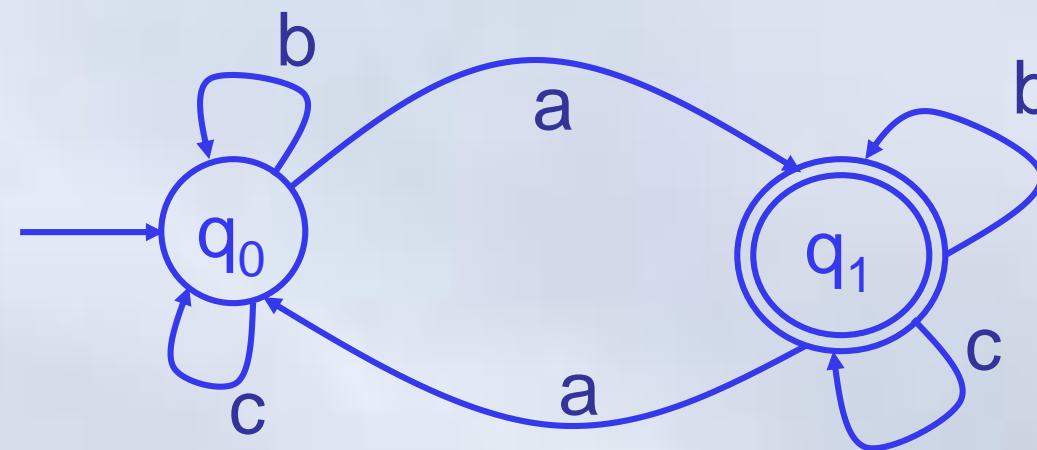


Trạng thái đầu



Trạng thái kết thúc

$$\delta(p, a) = q$$

Xâu trên bộ chữ $\{a, b, c\}$ có lẻ ký hiệu a

Ô tô mát hữu hạn đơn định (OHĐ)

OHĐ(DFA: Deterministic Finite Automata) là một hệ thống gồm

$$M = (\Sigma, Q, \delta, q_0, F)$$

- Σ : Bộ chữ vào
- Q : Tập hữu hạn các trạng thái của bộ điều khiển
 - $Q \cap \Sigma = \emptyset$
- $\delta : Q \times \Sigma \rightarrow Q$: Hàm dịch chuyển
 - Hàm dịch chuyển đơn định:
- $q_0 \in Q$: Trạng thái ban đầu
- $F \subseteq Q$ Tập các trạng thái cuối

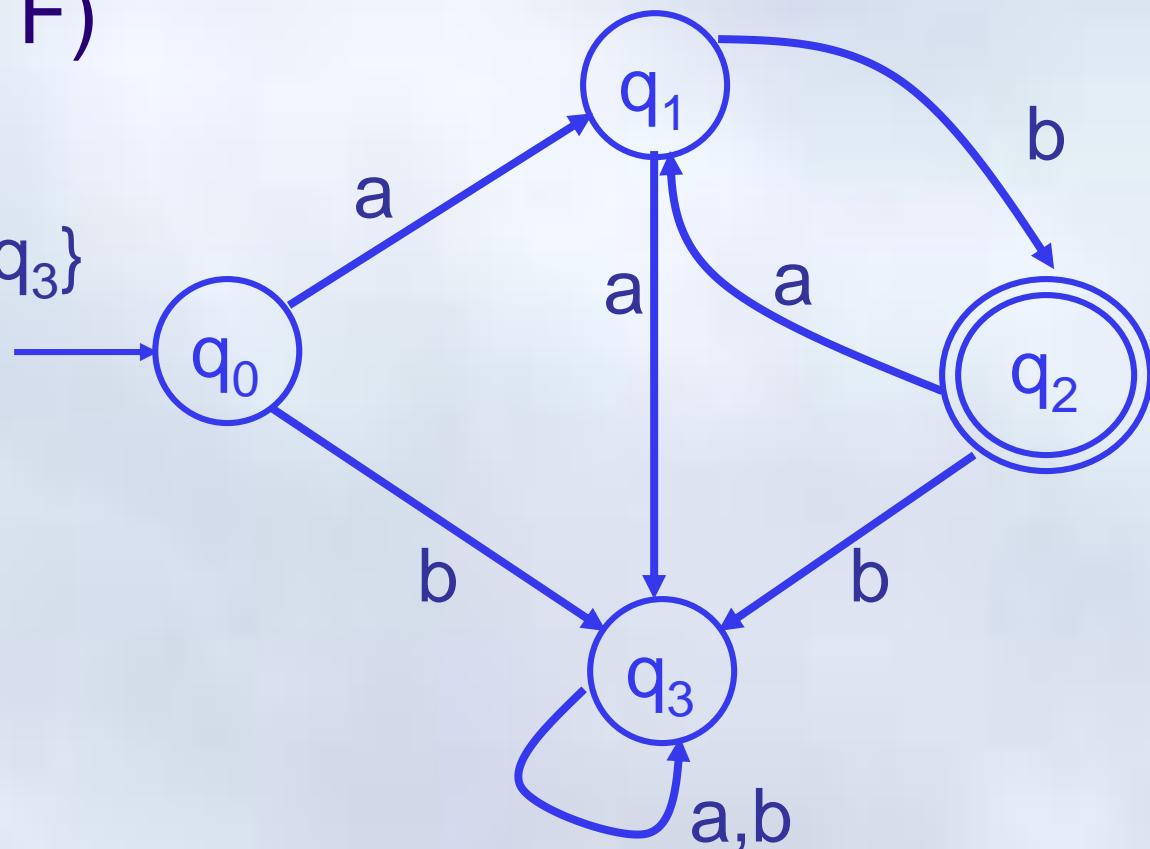
Ví dụ

$$M = (\Sigma, Q, \delta, q_0, F)$$

$$\Sigma = \{a, b\}$$

- $Q = \{q_0, q_1, q_2, q_3\}$
- $q_0 = q_0$
- $F = \{q_2\}$

δ	a	b
q_0	q_1	q_3
q_1	q_3	q_2
q_2	q_1	q_3
q_3	q_3	q_3

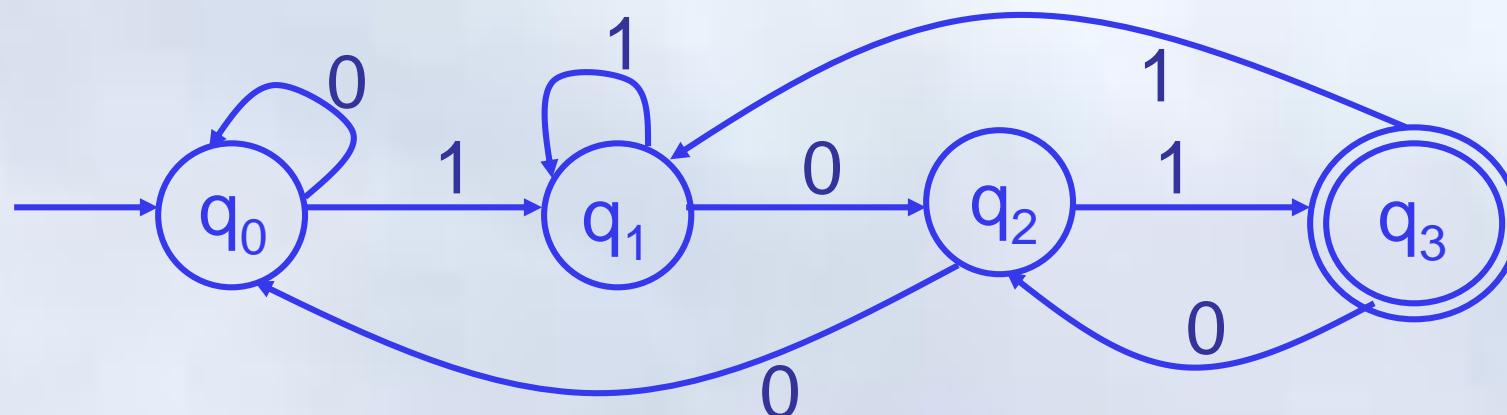


$$\begin{aligned}
 L(M) &= \{ab\}\{ab\}^{n \geq 0} = \{ab\}^{n(n>0)} \\
 &= \{ab, abab, ababab, \dots\}
 \end{aligned}$$

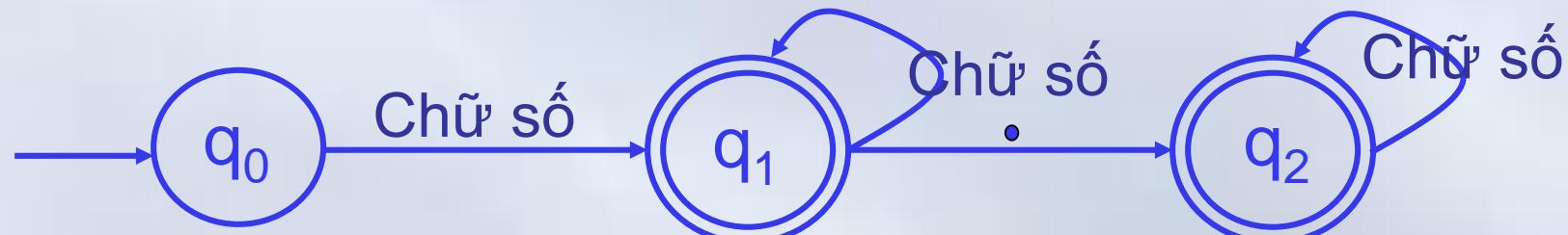
Hình trạng của FA

- Hình trạng của một FA là một xâu dạng qx
 - $q \in Q$: Trạng thái hiện tại của FA
 - $x \in \Sigma^*$: Phân chưa xét của xâu vào
 - Ký hiệu được đọc bởi đầu đọc là ký hiệu đầu của x
- Chuyển đổi hình trạng
 - Nếu $x = ay$ và $\exists \delta(q, x) = p$ thì $qx = qay \Rightarrow py$
 $qx \Rightarrow py$: Là một bước biến đổi hình trạng
 - Ví dụ: $q_0abaab \Rightarrow q_1baab \Rightarrow q_2aab \Rightarrow q_1ab \Rightarrow q_3b \Rightarrow q_3$
- Hình trạng đầu : $q_0\omega$ (ω : xâu cần đoán nhận)
 - Nếu $q_0\omega \Rightarrow^* q_{n+1} \in F$: Xâu ω được đoán nhận
- Ngôn ngữ được đoán nhận bởi DFA M là $L(M)$
 - $L(M) = \{\omega \mid \omega \in \Sigma^* \text{ và } q_0\omega \Rightarrow^* p \in F\}$

Ô tô mát hữu hạn đơn định → Ví dụ



$(0+1)^*101 \Leftrightarrow$ Xâu nhị phân có hậu tố là 101



Đoán nhận số nguyên hoặc số thực dấu phẩy tĩnh

Ô tô mát hữu hạn không đơn định (OHK)

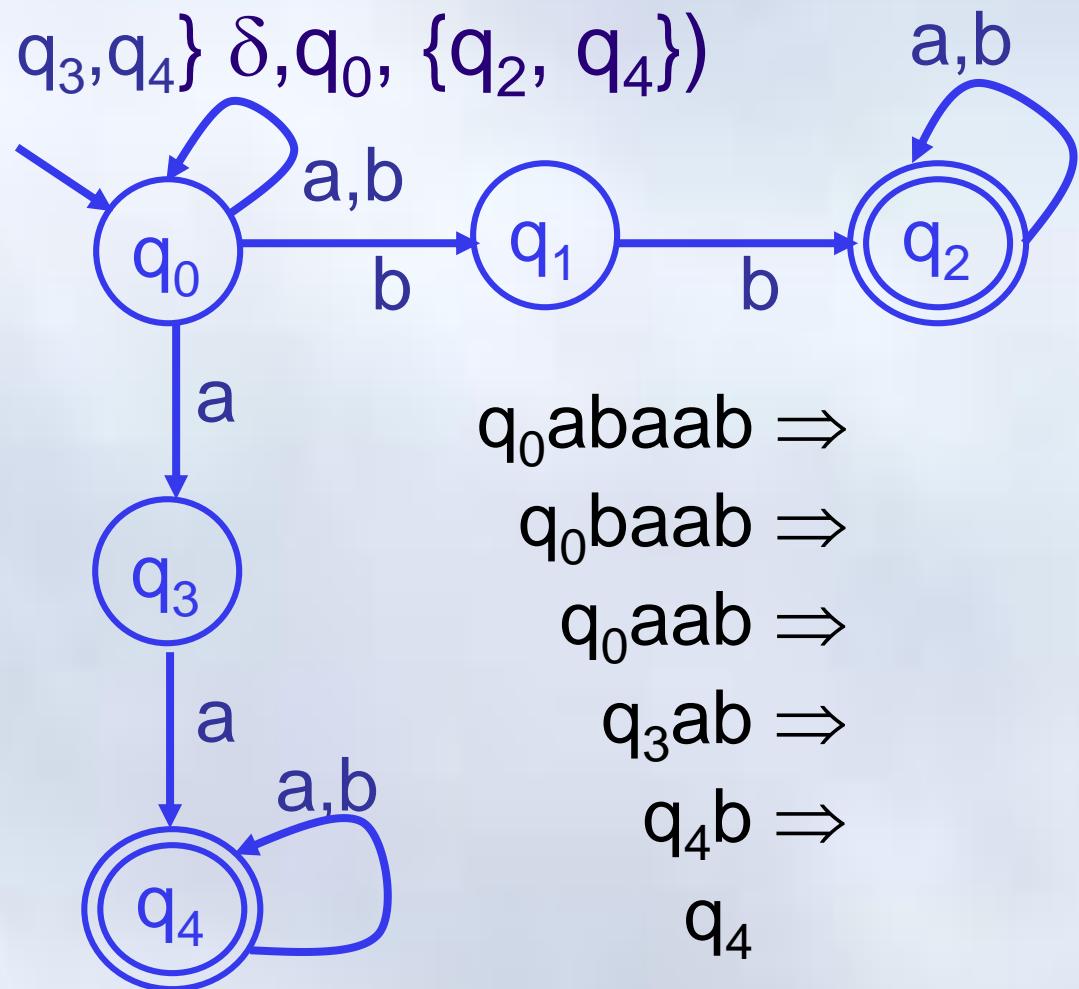
- OHK (**NFA**: Nondeterministic Finite Automata) là một hệ thống gồm

$$M = (\Sigma, Q, \delta, q_0, F)$$
 - Σ, Q, q_0, F : Định nghĩa như DFA
 - $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$
 - 2^Q : Tập các tập con của Q , kể cả tập rỗng
 - Hàm dịch chuyển không đơn định: Tại mỗi bước có thể tồn tại nhiều lựa chọn
- Đoán nhận xâu: Quá trình chuyển đổi hình trạng.
 - Quá trình đoán nhận không đơn định
- Ngôn ngữ: $L(M) = \{\omega \mid \omega \in \Sigma^* \text{ và } \exists q_0 \omega \xrightarrow{*} p \in F\}$

Ví dụ

$$M = (\{a,b\}, \{q_0, q_1, q_2, q_3, q_4\}, \delta, q_0, \{q_2, q_4\})$$

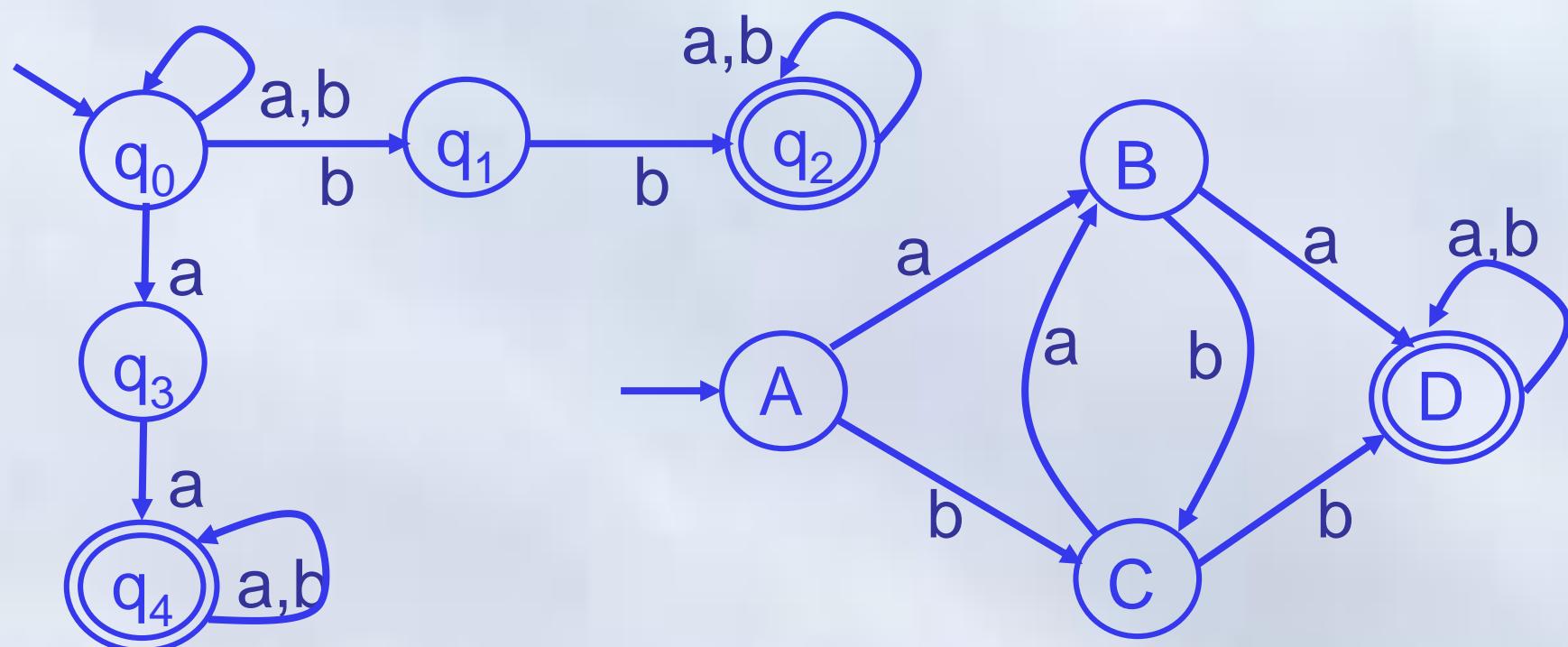
δ	a	b
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	\emptyset	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	\emptyset
q_4	$\{q_4\}$	$\{q_4\}$



Đoán nhận xâu có 2 ký tự a, hoặc 2 ký tự b liên tiếp

DFA và NFA

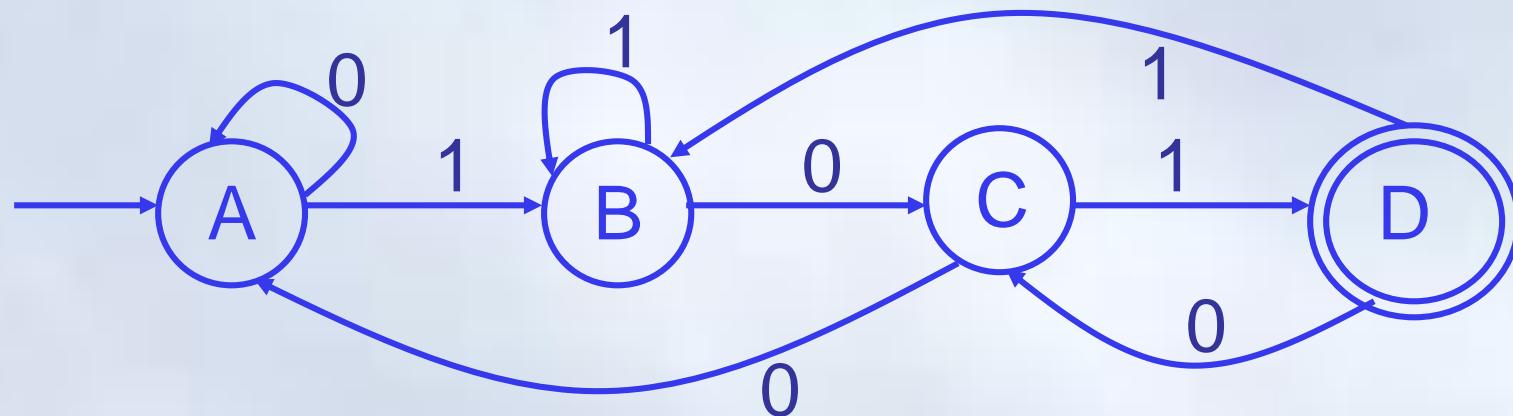
- DFA và NFA tương đương nhau
 - DFA và NFA cùng đoán nhận một lớp ngôn ngữ
 - Ngôn ngữ chính quy
- Với mỗi NFA, tồn tại DFA tương đương



Thuật toán đoán nhận xâu của OHĐ

- Phương pháp phân tích bảng
 - Dựa trên giải thuật tổng quát để đoán nhận DFA
 - Ưu điểm:
 - Chương trình độc lập với DFA
 - Dễ biến đổi, không cần sửa lại chương trình
 - Nhược điểm:
 - Khó khăn trong lập bảng
 - Kích thước bảng lớn
- Phương pháp diễn giải
 - Thực hiện như diễn giải sơ đồ
 - Dễ viết, nhưng c\trình gắn với đồ thị dịch chuyển
 - Được sử dụng để xây dựng bộ phân tích từ vựng

Phương pháp phân tích bảng



```

#include <stdio.h>
#include <string.h>
enum state {A,B,C,D};
int Delta[4][2]={A,B,C,B,A,D,C,B}; //Hàm dịch chuyển dạng bảng
char c, str[100];
int i, L;
enum state q = A; //Automat ở trạng thái đầu
  
```

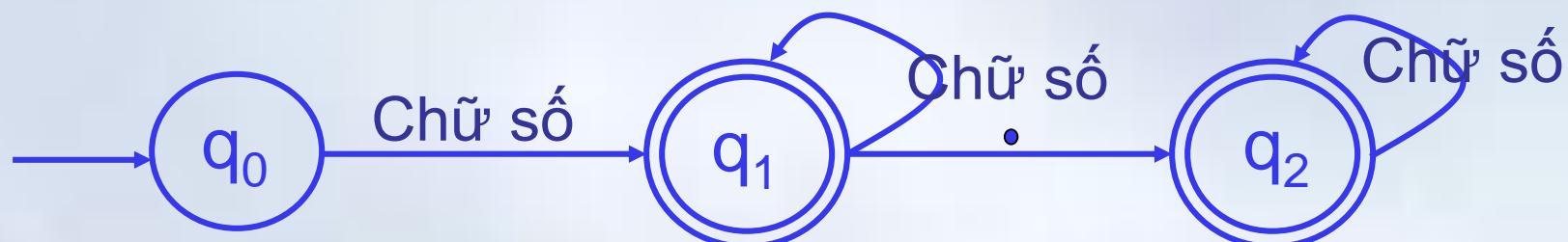
Phương pháp phân tích bảng

```

void main(){
    printf("Nhập xau.. :");    fflush(stdin); gets(str);
    i = 0; L = strlen(str);    c = str[i]-48;
    while (i < L){
        if(c == 0 || c == 1){    //Xâu vào chỉ gồm các ký hiệu 0,1
            q = Delta[q][c];    //Chuyển trạng thái mới
            i++;                 //Dịch chuyển đầu đọc
            c = str[i]-48;        //Ký hiệu đọc được
        } else {   printf("Loi xau vao...\n"); break; }
    }
    if(i==L) //Nếu đọc hết toàn bộ xâu vào
        if (q == D) printf("\n Xau %s duoc doan nhan !\n\n",str);
        else printf("\n Xau %s khong duoc doan nhan !\n\n",str);
}

```

Phương pháp diễn giải



```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void main(){
    char c, str[100];
    int i, L;
    printf("Nhập xau.. :"); fflush(stdin); gets(str);
    i= 0; L = strlen(str);
```

Phương pháp phân tích bảng

```
if(isdigit(str[i])){      //Nếu ký hiệu đọc được là một chữ số
    i=i+1;                //Đọc ký hiệu tiếp trên xâu vào
    while(isdigit(str[i])) i = i + 1; //Vẫn là chữ số, đọc tiếp
    if(i==L) printf("%s la so nguyen \n",str); //Đọc hết xâu vào
    else if(str[i]!='.') printf("%s khong duoc doan nhan (%d)\n",str,i+1);
    else {                //Đã đọc được dãy số và dấu '.'
        i = i + 1;
        while(isdigit(str[i])) i = i + 1;
        if(i==L) printf("%s la so thuc dau phay tinh \n",str);
        else printf("%s khong duoc doan nhan(%d)\n",str,i+1);
    }
} else printf("%s khong duoc doan nhan (%d)\n",str,i+1);
}//main
```

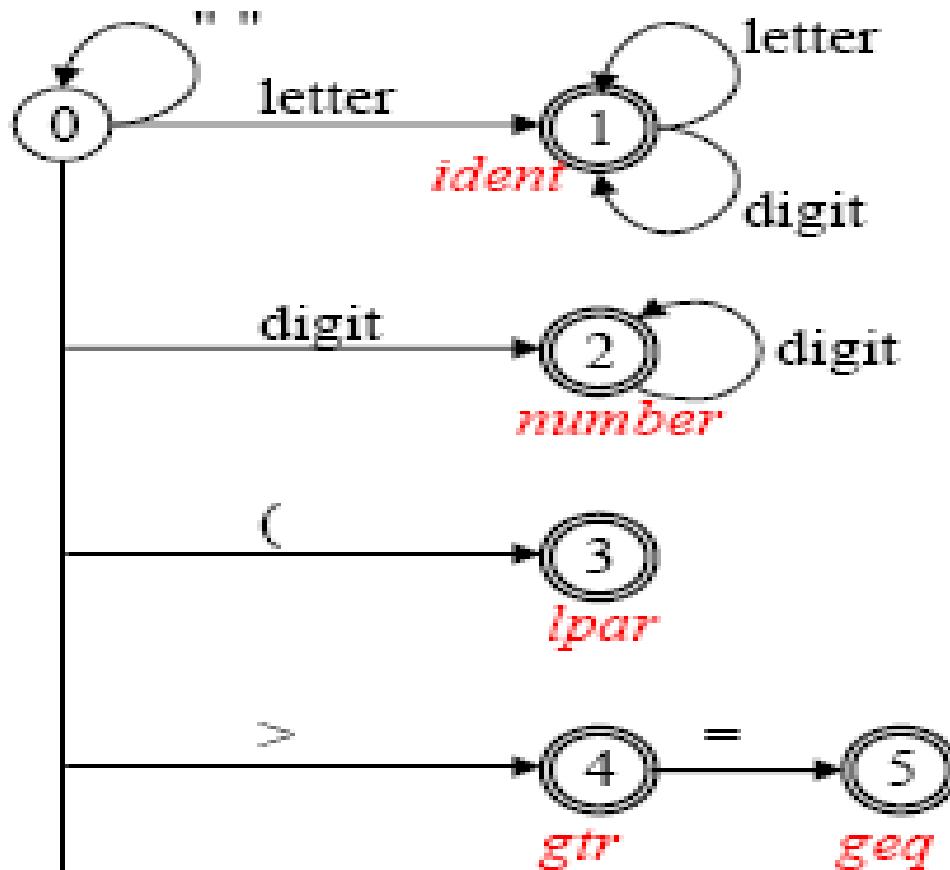
Chương 2: Phân tích từ vựng

1. Nhiệm vụ của bộ phân tích từ vựng
2. Biểu thức chính quy
3. Ô tô mát hữu hạn
4. Phân tích từ vựng của ngôn ngữ PL/0

Các từ vựng của PL/0 mở rộng

- Số nguyên
- Định danh
- Từ khóa:
**begin, end, if, then, while, do, call, odd, to
const, var, procedure, program, else, for**
- Dấu phép toán:
 - Số học: + - * /
 - So sánh: = <> < > <= >=
- Dấu phân cách: () . , ; []
- Dấu phép gán: :=

Ô-tô-mát hữu hạn của bộ phân tích từ vựng



Xâu vào: max ≥ 30

$s_0 \xrightarrow{m} s_1 \xrightarrow{a} s_1 \xrightarrow{x} s_1$

Ở s_1 đọc " " không chuyển trạng thái
Đoán nhận ident

$s_0 \xrightarrow{''} s_0 \xrightarrow{>} s_4 \xrightarrow{=} s_5$

Loại bỏ khoảng trắng
Không dừng ở s_4
Ở s_5 đọc " " không chuyển trạng thái
Đoán nhận geq

$s_0 \xrightarrow{''} s_0 \xrightarrow{3} s_2 \xrightarrow{0} s_2$

Loại bỏ khoảng trắng ở đầu
Ở s_2 đọc " " không chuyển trạng thái
Đoán nhận number

Sau khi mỗi từ tố được nhận biết, bộ phân tích từ vựng lại quay về trạng thái 0 \Rightarrow Trạng thái bắt đầu của thủ tục đoán nhận

Các từ tố của PL/0 mở rộng

- Mỗi từ tố là tên một trạng thái
 - Từ tố là **ident**, cần kiểm tra xem từ vựng tương ứng có phải là từ khóa không
- Các từ tố của PL/0
 - Số nguyên: **NUMBER**
 - Định danh: **IDENT**
 - Nếu từ vựng trùng từ khóa, từ vựng sẽ mang ý nghĩa từ tố trùng với tên từ khóa
 - Ví dụ: Từ vựng **Begin** có từ tố **BEGIN**
Từ vựng **Procedure** có từ tố **PROCEDURE**

Các từ tố của PL/0 mở rộng

Các toán tử

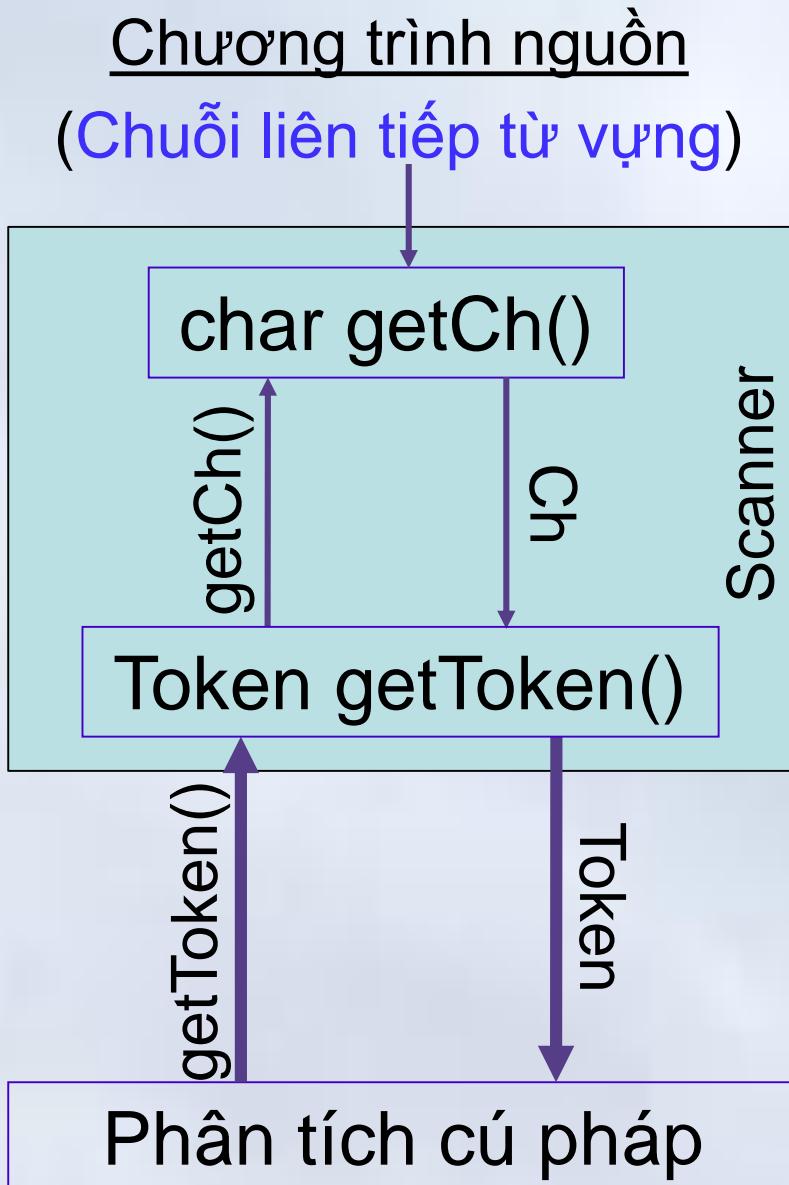
Từ vựng	Từ tố	Từ vựng	Từ tố
+	PLUS	-	MINUS
*	TIMES	/	SLASH
=	EQU	◊	NEQ
<	LSS	≤	LEQ
>	GTR	≥	GEQ
(LPARENT)	RPARENT
[LBRACK]	RBRACK
:=	ASSIGN	;	SEMICOLON
,	COMMA	.	PERIOD
%	PERCENT		NONE

Các khai báo

```
typedef enum {//Các loại Token được sử dụng trong PL/0
    NONE=0, IDENT, NUMBER,
    BEGIN, CALL, CONST, DO, ELSE, END, FOR, IF, ODD,
    PROCEDURE, PROGRAM, THEN, TO, VAR, WHILE,
    PLUS, MINUS, TIMES, SLASH, EQU, NEQ, LSS, LEQ,
    GTR, GEQ, LPARENT, RPARENT, LBRACK, RBRACK,
    PERIOD, COMMA, SEMICOLON, ASSIGN, PERCENT
} TokenType;

TokenType Token; //Token nhận dạng được
int    Num;        //Từ vựng khi Token là NUMBER
char  Id[MAX_IDENT_LEN + 1]; //Từ vựng khi Token là IDENT
```

Xây dựng từ vựng



- Scanner gồm 2 hàm
 - `getCh()`:
 - Làm việc với thiết bị lưu trữ
 - Đọc file nguồn, trả về các ký tự ASCII
 - `getToken()`
 - Được gọi từ Parser
 - Xây dựng từ vựng từ các ký tự trả về bởi `getCh()`
 - Nhận dạng từ tố
- **Hoạt động**
 - Tương tự OHD
 - Khi được gọi (*bắt đầu hoạt động*) ở trạng thái q_0

Xây dựng từ vựng

- Khi bộ pttv - *thủ tục getToken()* bắt đầu hoạt động, ô-tô-mát ở trạng thái khởi tạo (*Trạng thái 0*)
- Bộ pttv gọi liên tiếp *getCh()* để đọc các ký hiệu trên văn bản nguồn cho tới khi gặp một ký tự không thuộc luật mô tả hiện tại → Ô-to-mát không chuyển trạng thái được. Khi đó
 - Xâu đọc được là từ vựng mang ý nghĩa của từ tố đang phân tích và là trạng thái hiện tại của Ô-tô-mát
 - Đọc thửa ra một ký tự
 - Là ký tự trắng hoặc ký tự đầu của từ tố tiếp → Khi *getToken()* được gọi, sẽ làm việc ngay với một ký tự có sẵn.
 - Tại lần gọi *getToken()* đầu tiên để xác định từ tố thứ nhất, bộ pttv phải cũng cấp sẵn một ký tự → là ký tự trắng

Xây dựng từ vựng

```
//Ch chứa ký tự đọc được từ văn bản nguồn bởi hàm getCh()
switch(Ch)  {
    case SPACE : while (Ch==SPACE) getCh();
    case LETTER: getCh();
        while (Ch==LETTER || Ch==DIGIT) getCh();
        return Ident; //Cũng có thể là từ khóa →Phải kiểm tra
    case DIGIT: while (Ch==DIGIT) getCh(); return NUMBER;
    case '+': getCh(); return plus;
    case '>': getCh();
        if(Ch == '=') { getCh(); return GEQ;}
        else return GTR;
    ....
}
```

Nhận dạng từ tố

Xây dựng xong từ vựng, cần nhận dạng từ tố

- Là tên trạng thái cuối cùng của Ô-tô-mát
- Nếu Ô-tô-mát kết thúc ở trạng thái IDENT, cần kiểm tra đây có phải là từ khóa
 - Dùng kỹ thuật bảng chuyển đổi

Chú ý: Với từ tố ident và number, cần phải ghi nhận giá trị từ vựng tương ứng

BEGIN	→	BEGIN
CALL	→	CALL
.....		
VAR	→	VAR
WHILE	→	WHILE

Bài tập

1. Xây dựng hoàn chỉnh bộ phân tích từ vựng cho PL/0
2. Tìm hiểu về Lex/JLex

IT4079:NGÔN NGỮ và PHƯƠNG PHÁP DỊCH

Phạm Đăng Hải

haipd@soict.hust.edu.vn

Chương 3: Phân tích cú pháp

1. Bài toán phân tích cú pháp
2. Phương pháp phân tích cú pháp quay lui
3. Phương pháp phân tích bảng
4. Phương pháp phân tích cú pháp tất định
5. Phân tích cú pháp cho PL/0

Bài toán đặt ra

Cho

- Văn phạm phi ngũ cảnh G

$$G = (V_T, V_N, P, S)$$

- Xâu ω $\in V_T^*$

Hỏi

- $\omega \in L(G)$?

Nếu $\omega \in L(G)$

Program Vidu;

Begin

X := 10

End.

Trong chương trình dịch,
xâu ω là chuỗi các token
thu được từ giai đoạn
trước – phân tích từ vựng

PROGRAM IDENT
SEMICOLON BEGIN IDENT
ASSIGN NUMBER END
PERIOD

- Chỉ ra các sản xuất đã sử dụng để sinh ra ω
- Cấu trúc nêu cây suy dẫn

Phương pháp phân tích

- Kiểm tra xâu phân tích từ trái qua phải
 - Kiểm tra ký hiệu trái nhất của xâu cần phân tích
 - Tới ký hiệu tiếp,.. Cho tới ký hiệu cuối cùng
- Phương pháp xây dựng cây phân tích
 - Trên xuống (*Top-down*): $S \Rightarrow^* \omega?$
 - Dưới lên (*Bottom-up*): $\omega^* \Leftarrow S?$
- Phương pháp lựa chọn sản xuất ($A \rightarrow \alpha_1 | \dots | \alpha_n$)
 - Quay lui (*backtracking*)
 - Thủ lần lượt các sản xuất
 - Tất định (*deterministic*)
 - Xác định được duy nhất một sản xuất thích hợp

Phân tích trái

- **Phân tích trái** của xâu α là dãy các sản xuất được sử dụng trong *suy diễn trái* từ S ra α
- Các sản xuất được đánh số thứ tự $1,..p$
 - Phân tích là danh sách các số từ 1 đến p
- Ví dụ cho văn phạm

$$1. E \rightarrow T+E$$

$$2. E \rightarrow T$$

$$3. T \rightarrow F^* T$$

$$4. T \rightarrow F$$

$$5. F \rightarrow (E)$$

$$6. F \rightarrow a$$

Xét xâu $a^*(a+a)$

$$E \Rightarrow^2 T \Rightarrow^3 F^* T \Rightarrow^6 a^* T$$

$$\Rightarrow^4 a^* F \Rightarrow^5 a^*(E)$$

$$\Rightarrow^1 a^*(T+E) \Rightarrow^4 a^*(F+E)$$

$$\Rightarrow^6 a^*(a+E) \Rightarrow^2 a^*(a+T)$$

$$\Rightarrow^4 a^*(a+F) \Rightarrow^6 a^*(a+a)$$

Phân tích phải

- **Phân tích phải** của xâu α là dãy các sản xuất được sử dụng trong *suy diễn phải* từ S ra α
- Các sản xuất được đánh số thứ tự $1,..p$
 - Phân tích là danh sách các số từ 1 đến p
- Ví dụ cho văn phạm

$$1. E \rightarrow T+E$$

$$2. E \rightarrow T$$

$$3. T \rightarrow F^* T$$

$$4. T \rightarrow F$$

$$5. F \rightarrow (E)$$

$$6. F \rightarrow a$$

Xét xâu $a^*(a+a)$

$$E \Rightarrow^2 T \Rightarrow^3 F^* T \Rightarrow^4 F^* F$$

$$\Rightarrow^5 F^*(E) \Rightarrow^1 F^*(T+E)$$

$$\Rightarrow^2 F^*(T+T) \Rightarrow^4 F^*(T+F)$$

$$\Rightarrow^6 F^*(T+a) \Rightarrow^4 F^*(F+a)$$

$$\Rightarrow^6 F^*(a+a) \Rightarrow^6 a^*(a+a)$$

Chương 3: Phân tích cú pháp

1. Bài toán phân tích cú pháp
2. Phương pháp phân tích cú pháp quay lui
3. Phương pháp phân tích bảng
4. Phương pháp phân tích cú pháp tất định
5. Phân tích cú pháp cho PL/0

Giới thiệu

- Tư tưởng chủ yếu của giải thuật
 - Xây dựng cây phân tích cú pháp (cây suy dẫn) cho xâu ω
- **Thuật toán Top-down**
 - Đi từ nút gốc tới nút lá
- **Thuật toán Bottom –up**
 - Quá trình phân tích gạt thu gọn

Phân tích Top-down

Cho VPPNC $G = (V_T, V_N, P, S)$

\forall sản xuất $A \rightarrow \alpha_1 | \dots | \alpha_n$ được đánh số 1, 2, ..

Xây dựng cây phân tích cho xâu ω :

Bước 1: Khởi tạo

- Xây dựng cây chỉ có một nút gốc S
 - S (Start symbol): Ký hiệu khởi đầu
- Gọi **ký hiệu cần phân tích** là ký hiệu đầu tiên của xâu ω
- Gọi S là **nút hoạt động**,

Phân tích Top-down

Bước 2. Tạo các nút con của cây (một cách đệ quy)

❖ **Nút hoạt động** là ký hiệu không kết thúc A

- Chọn sản xuất đầu tiên của A chưa được áp dụng: $A \rightarrow X_1 X_2 \dots X_k$ ($k \geq 0$)
- Tạo k con trực tiếp của A với nhãn $X_1, X_2, \dots X_k$
- Nếu $k > 0$, Lấy X_1 làm nút hoạt động
- Nếu $k = 0$, (sản xuất $A \rightarrow \epsilon$), lấy nút bên phải (ngay sau) A là nút hoạt động

Quay lại thực hiện bước 2

Phân tích Top-down

Bước 2. Tạo các nút con của cây (tiếp)

- ❖ **Nút hoạt động** là ký hiệu kết thúc a
 - So sánh a với ký hiệu cần phân tích hiện tại
 - **Nếu trùng nhau**
 - Nút hoạt động là nút bên phải của a
 - Ký hiệu cần phân tích là ký hiệu tiếp theo trên xâu vào
 - *Quay lại thực hiện bước 2*
 - **Nếu không trùng nhau**
 - Quay lại bước đã sử dụng một sản xuất và thử sản xuất tiếp.
 - Nếu đã hết khả năng, quay lại bước trước

Phân tích Top-down

3. Điều kiện dừng

- Đã áp dụng hết khả năng mà không tạo được cây \Rightarrow **Xâu không được đoán nhận**
- Tạo ra cây suy dẫn cho xâu vào

Phân tích Top-down

❖ Điều kiện áp dụng thuật toán

- Văn phạm không đệ quy trái

❖ Chi phí ($n = l(\omega)$; $C, K > 1$ là các hằng số)

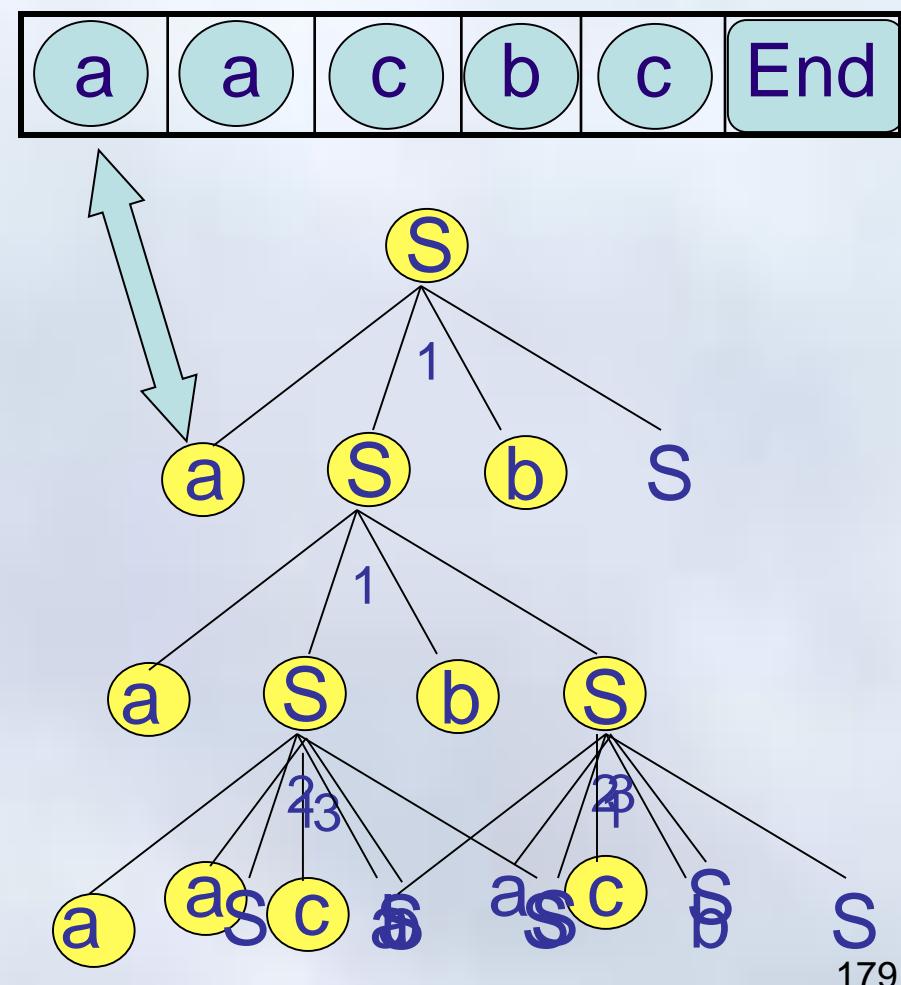
- Bộ nhớ C^*n
- Thời gian K^n

Phân tích Top-down → Ví dụ

Văn phạm $S \rightarrow aSbS|aS|c$, xâu $\omega = aacbc$

Đánh số sản xuất

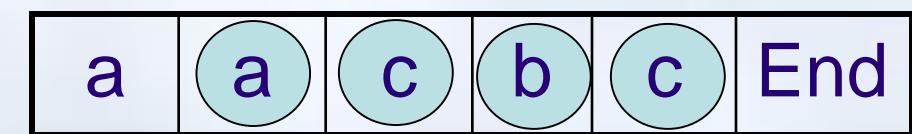
1. $S \rightarrow aSbS$
 2. $S \rightarrow aS$
 3. $S \rightarrow c$



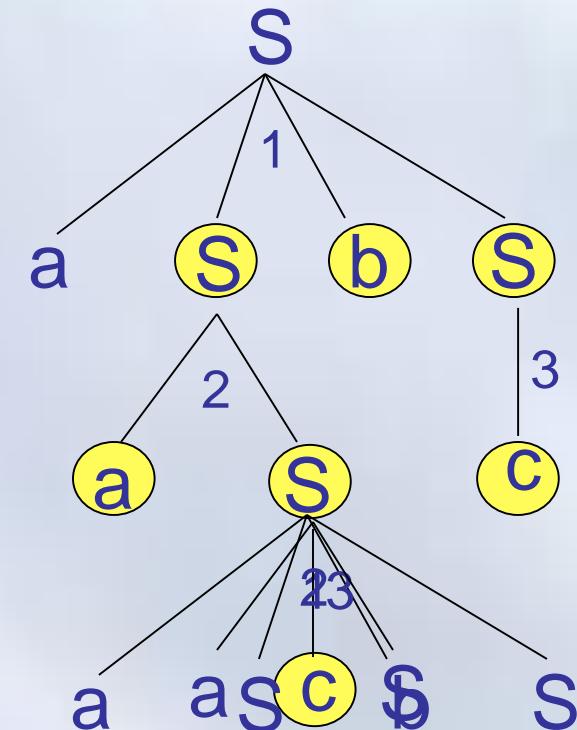
Phân tích Top-down → Ví dụ

Văn phạm $S \rightarrow aSbS|aS|c$, xâu $\omega = aacbc$

Đánh số sản xuất

1. $S \rightarrow aSbS$ 2. $S \rightarrow aS$ 3. $S \rightarrow c$ 

Đánh số sản xuất

1. $S \rightarrow c$ 2. $S \rightarrow aS$ 3. $S \rightarrow aSbS$ 

Phân tích Top-down → Bài tập

Cho văn phạm

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

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

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

Xây dựng cây suy dẫn cho xâu a+a

Giải thuật phân tích Top-down quay lui (1/8)

Vào

- Văn phạm phi ngũ cành không đệ quy trái
- xâu cần phân tích $\omega = a_1 \dots a_n$, $n \geq 0$
- Các sản xuất của G được đánh số 1, ..., q

Ra

- Một phân tích trái cho ω (nếu có)
- Thông báo lỗi nếu ngược lại

Quy ước

$\forall A \in V_N$, giả sử $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$

Coi các sản xuất trên là

$A_1 \rightarrow \alpha_1$
 \dots
 $A_n \rightarrow \alpha_n$

Để phân biệt sản xuất đã sử dụng của cùng một ký hiệu không kết thúc A

Giải thuật phân tích Top-down quay lui (2/8)

Phương pháp: Dùng 2 stack D_1 và D_2

- D_1 ghi lại những lựa chọn đã sử dụng và những ký hiệu vào đã được phân tích
 - Đầu đọc đã đổi vị trí theo ký hiệu đã phân tích
- D_2 biểu diễn dạng câu trái hiện tại có được bằng cách thay thế các ký hiệu không kết thúc bởi vế phải tương ứng.
 - Đỉnh của D_2 là nút hoạt động của cây tạo ra

Giải thuật phân tích Top-down quay lui (3/8)

Hình trạng của giải thuật

Bộ bốn (s , i , α , β)

- $s \in Q$: Trạng thái hiện thời của giải thuật
 - q : Trạng thái bình thường
 - b : Quay lui
 - t : Kết thúc
- i : Vị trí đầu đọc (*# kết thúc xâu băng vào*)
- α : Nội dung stack thứ nhất, đỉnh ở bên phải
- β : Nội dung stack thứ hai, đỉnh ở bên trái
- Hình trạng ban đầu (q , 1 , ϵ , $S\#$)

Giải thuật phân tích Top-down quay lui (4/8)

Thực hiện giải thuật

- Bắt đầu từ hình trạng đầu, tính liên tiếp các hình trạng tiếp theo cho đến khi không tính được nữa.
- Ký hiệu $|—$ chỉ một thay đổi hình trạng $(q, i, \alpha, \beta) |— (q', i', \alpha', \beta')$
- Giải thuật thực hiện theo các dịch chuyển

Giải thuật phân tích Top-down quay lui (5/8)

1. Mở rộng cây

$$(q, i, \alpha, A\beta) \vdash (q, i, \alpha A_1, \gamma_1 \beta)$$

$A \in V_N$ và $A_1 \rightarrow \gamma_1$ là lựa chọn đầu tiên của A

2. Phù hợp với ký hiệu vào

$$(q, i, \alpha, a\beta) \vdash (q, i+1, \alpha a, \beta)$$

Ký hiệu $a \in V_T$ ở đỉnh D_2 phù hợp ký hiệu vào

\Rightarrow Chuyển a sang D_1 và dịch chuyển đầu đọc

3. Không phù hợp ký hiệu vào

$$(q, i, \alpha, a\beta) \vdash (b, i, \alpha, a\beta) \quad // \text{chuyển sang}$$

$//$ trạng thái quay lui

Giải thuật phân tích Top-down quay lui (6/8)

4. Quay lui trên xâu vào

$$(b, i, \alpha a, \beta) \vdash (b, i-1, \alpha, a\beta) \quad a \in V_T$$

Chuyển ký hiệu trên đỉnh D_1 sang D_2 và dịch đầu đọc sang trái

5. Thủ lựa chọn tiếp theo

$$(b, i, \alpha A_j, \gamma_j \beta) \vdash (q, i, \alpha A_{j+1}, \gamma_{j+1} \beta)$$

- γ_{j+1} là lựa chọn tiếp của ký hiệu A
- Nếu là sản xuất cuối của A, quay lui tiếp
 - Thay γ_j bởi A và loại bỏ A_j trên đỉnh của D_1

$$(b, i, \alpha A_j, \gamma_j \beta) \vdash (b, i, \alpha, A_j \beta)$$

Giải thuật phân tích Top-down quay lui (7/8)

6. Nếu hết khả năng quay lui ($i=1, A \equiv S$):

Không đoán nhận

7. Kết thúc thành công

- $(q, n+1, \alpha, \#) \vdash (t, n+1, \alpha, \varepsilon)$
- Kết luận: Xâu đã được đoán nhận.
- Để tìm phân tích trái, thực hiện hàm $h(\alpha)$

Tìm phân tích trái

- $h(a) = \varepsilon \quad \forall a \text{ là ký hiệu kết thúc}$
- $h(A_i) = p$, Với p là số hiệu của sản xuất liên hệ với sản xuất $A \rightarrow \gamma$ với γ là lựa chọn thứ i của A

Giải thuật phân tích Top-down quay lui (8/8)

Ví dụ văn phạm

1. $S_1 \rightarrow aSb$ 2. $S_2 \rightarrow c$ $\omega: aacbb$ $h(S_1aS_1aS_2cbb)=112$

(q, 1, ϵ , $S\#$)
 \vdash (q, 1, S_1 , $aSb\#$)
 \vdash (q, 2, S_1a , $Sb\#$)
 \vdash (q, 2, S_1aS_1 , $aSbb\#$)
 \vdash (q, 3, S_1aS_1a , $Sbb\#$)
 \vdash (q, 3, $S_1aS_1aS_1$, $aSbbb\#$)
 \vdash (b, 3, $S_1aS_1aS_1$, $aSbbb\#$)
 \vdash (q, 3, $S_1aS_1aS_2$, $cbb\#$)
 \vdash (q, 4, $S_1aS_1aS_2c$, $bb\#$)
 \vdash (q, 5, $S_1aS_{1a}S_2cb$, $b\#$)
 \vdash (q, 6, $S_1aS_1aS_2cbb$, $\#$)
 \vdash (t, 6, $S_1aS_1aS_2cbb$, ϵ)
 \vdash Đoán nhận

Phân tích Bottom-up

- Sử dụng chu trình phân tích phải, thông qua tất cả các suy dẫn phải có thể theo chiều ngược lại phù hợp với xâu vào
 - Là quá trình gạt-thu gọn (**shift – reduce**)
- Thuật toán sử dụng stack S, dùng chứa các ký hiệu của văn phạm đã sinh ra một tiền tố nào đó trên xâu vào

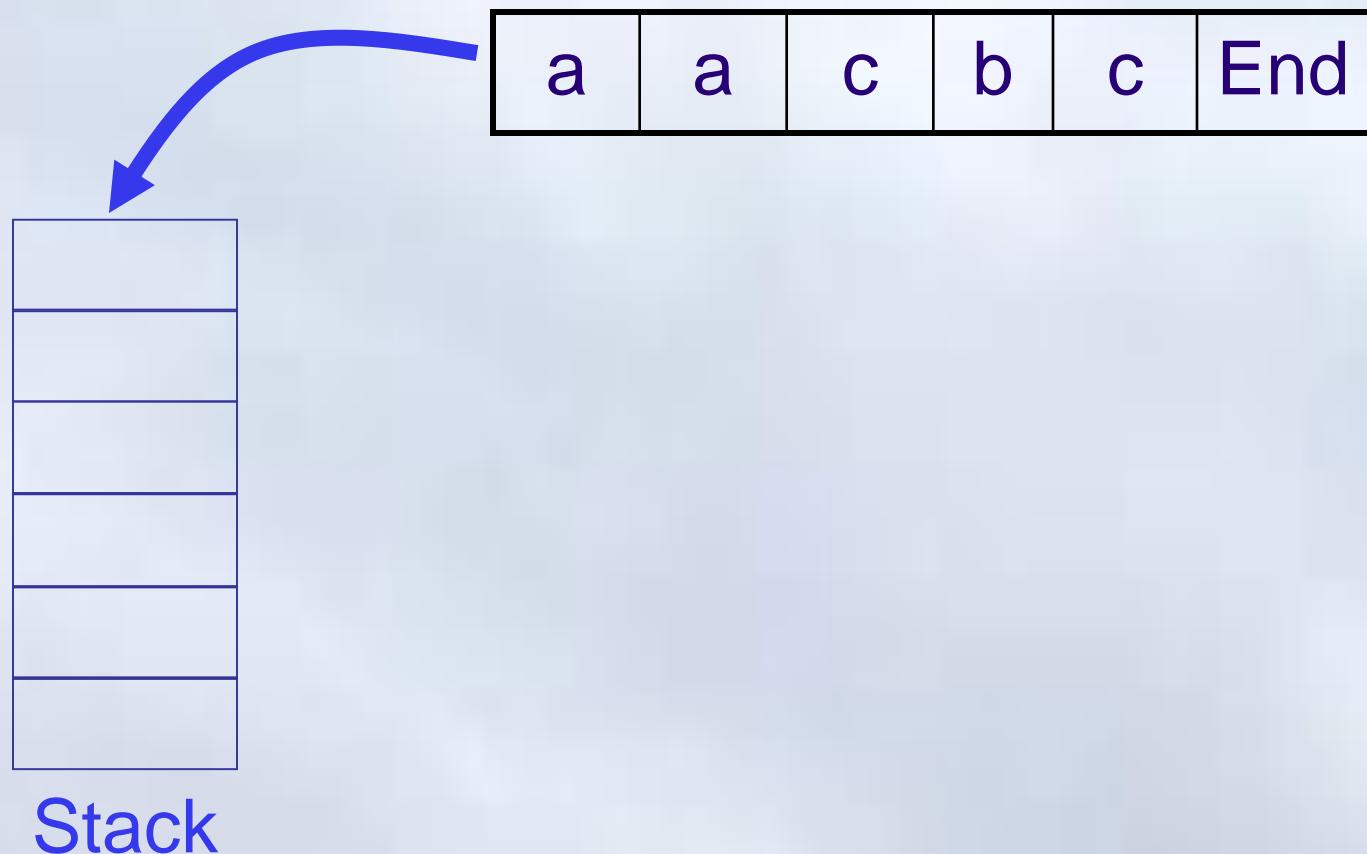
Phân tích Bottom-up

Hoạt động

- Xét tất cả các xâu ω có thể trên đỉnh Stack S
 - Nếu tồn tại một sản xuất $A \rightarrow \omega \in P$, thu gọn xâu ω về A
 - Nếu có nhiều lựa chọn \rightarrow đánh số để thử lần lượt
 - Nếu không thể thu gọn được, gạt ký hiệu tiếp theo của ω vào Stack
 - Nếu đi hết xâu mà không thể thu gọn \rightarrow quay lui lại bước thu gọn sau cùng để thử thu gọn khác
 - Thuật toán dừng khi
 - Đã gạt hết các ký hiệu và thu gọn về S
 - Đã thử hết các trường hợp những vẫn không thu gọn

Phân tích Bottom-up → Ví dụ

Văn phạm $S \rightarrow aSbS|aS|c$, xâu $\omega = aacbc$



Phân tích Bottom-up

❖ Điều kiện áp dụng thuật toán

- Văn phạm không chứa sản xuất dạng

$$A \rightarrow \epsilon \text{ hoặc } A \Rightarrow^+ A$$

❖ Chi phí ($n = l(\omega)$; $C, K > 1$ là các hằng số)

- Bộ nhớ C^*n
- Thời gian K^n

Phân tích Bottom-up → Ví dụ

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

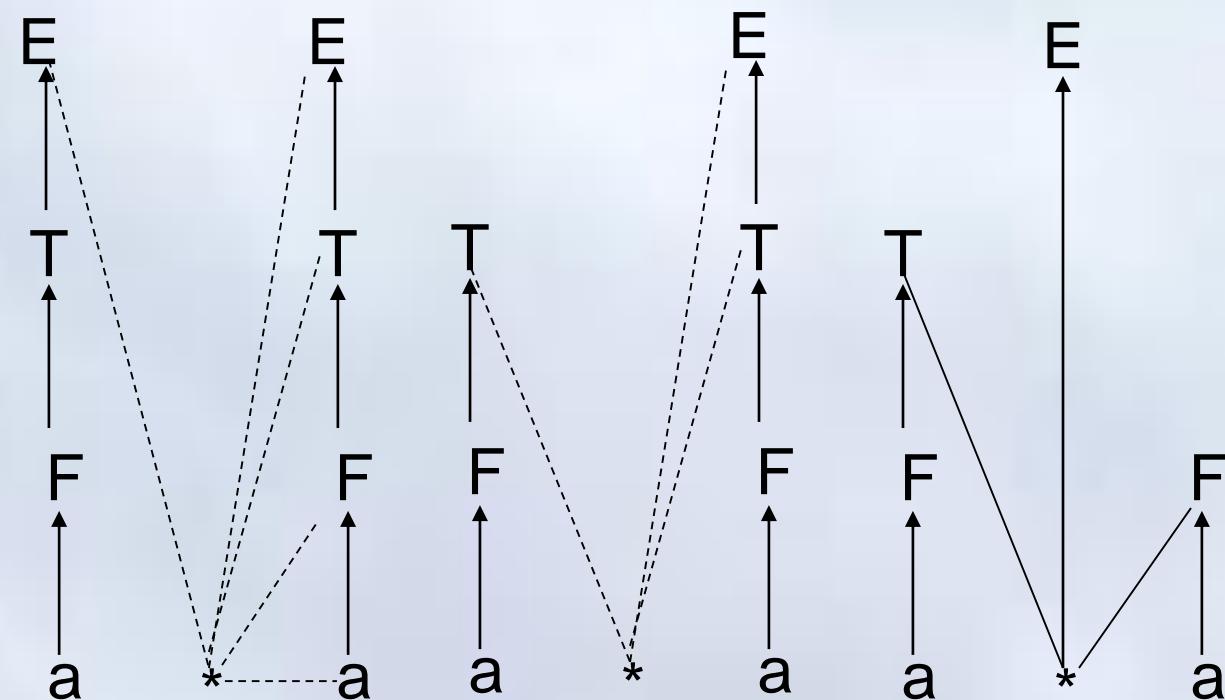
$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow a$$

$$\text{xâu } \omega = a * a$$

Dấu --- thể hiện các xâu trên đỉnh
Stack không rút gọn được



Giải thuật phân tích Bottom-Up quay lui

Vào : Văn phạm không chứa sản xuất dạng

$$A \rightarrow \epsilon \text{ hoặc } A \Rightarrow^+ A$$

Các sản xuất của G được đánh số 1,...,q

Ra: - Phân tích phải cho ω (nếu có)

- Thông báo lỗi nếu ngược lại

Hình trạng của giải thuật: Bộ bốn (s, i, α, β)

- Hình trạng ban đầu ($q, 1, \#, \epsilon$)
- s : trạng thái thuật toán gồm: $q, b, t; \alpha, \beta$ nội dung các stack
- **Stack D_1** : chứa ký hiệu của văn phạm đã sinh ra tiền tố của xâu vào (*đến vị trí $i-1$*)
- **Stack D_2** : Ghi nhận quá trình gạt (s) thu gọn cần thiết để sinh được D_1 từ xâu vào

Giải thuật phân tích Bottom-Up quay lui

1. Thu gọn

$$(q, i, \alpha\beta, \gamma) \vdash (q, i, \alpha A, j\gamma)$$

- Nếu $A \rightarrow \beta$ là sản xuất thứ j của P
- β là xâu trên đỉnh stack D_1 (được thu gọn về A)

2. Gạt (shift)

$$(q, i, \alpha, \gamma) \vdash (q, i+1, \alpha a_i, s\gamma)$$

- Gạt ký hiệu $i+1$ lên đỉnh D_1 , dịch đầu dọc. Đặt s lên D_2 để chỉ ra đã thực hiện thao tác gạt
- Nếu $i < n+1$, thực hiện thu gọn
- Nếu $i = n+1$, kiểm tra chấp nhận hoặc thực hiện quay lui

Giải thuật phân tích Bottom-Up quay lui

3. Chấp nhận

$$(q, n+1, \#S, \gamma) \vdash (t, n+1, \#S, \gamma)$$

- Đã đọc hết xâu vào và thu gọn về S.
- Tính phân tích phải bởi thực hiện $h(\alpha)$
 - $h(s) = \varepsilon$
 - $h(j) = j$ với j là số thứ tự sản xuất

4. Quay lui trên xâu vào

Nếu α khác $\#S$ chuyển trạng thái quay lui

$$(q, n+1, \alpha, \gamma) \vdash (b, n+1, \alpha, \gamma)$$

Giải thuật phân tích Bottom-Up quay lui

- **Quay lui 1**

$$(\mathbf{b}, i, \alpha A, j\gamma) \vdash (\mathbf{q}, i, \alpha' B, k\gamma)$$

- Nếu $A \rightarrow \beta$ là sản xuất thứ j của P
- xâu $\alpha\beta = \alpha'\beta'$ và $B \rightarrow \beta'$ là sản xuất thứ k của P

- **Quay lui 2**

$$(\mathbf{b}, n+1, \alpha A, j\gamma) \vdash (\mathbf{b}, n+1, \alpha\beta, \gamma)$$

- Nếu $A \rightarrow \beta$ là sản xuất thứ j của P và không có lựa chọn nào khác của $\alpha\beta$ để thu gọn (*như quay lui 1*), thực hiện **tự hủy** thu gọn đã thực hiện
- Nếu đã đọc đến cuối xâu \rightarrow tiếp tục quay lui

Giải thuật phân tích Bottom-Up quay lui

- **Quay lui 3**

$$(\mathbf{b}, i, \alpha A, j\gamma) \vdash (\mathbf{q}, i+1, \alpha\beta a_i, s\gamma)$$

- Nếu $i < n+1$, $A \rightarrow \beta$ là sản xuất thứ j của P và không còn lựa chọn nào khác của xâu $\alpha\beta$ thì
 - Hủy thu gọn j ($A \rightarrow \beta$),
 - Gạt a_i vào D_1 và đặt s vào D_2 rồi tiếp tục thu gọn

- **Quay lui 4**

$$(\mathbf{b}, i, \alpha a, s\gamma) \vdash (\mathbf{b}, i-1, \alpha, \gamma)$$

- Nếu đỉnh D_2 là ký hiệu gạt và mọi lựa chọn ở vị trí i đã hết, hoạt động *gạt bị tự hủy*
 - Đầu đọc dịch trái 1 đơn vị
 - Loại bỏ ký hiệu a khỏi D_1 và s khỏi D_2

Phân tích Bottom-Up quay lui → Ví dụ

Ví dụ văn phạm

1. $S \rightarrow aSbS$

2. $S \rightarrow aS$

3. $S \rightarrow c$

xâu $\omega = aacbc$

$h(13ss23sss) =$

1323

(q, 1, #, ε)
 |— (q, 2, #a, s) |— (q, 3, #aa, ss)
 |— (q, 4, #aac, sss) |— (q, 4, #aaS, 3sss)
 |— (q, 4, #aS, 23sss)
 |— (q, 4, #S, 223sss)
 |— (q, 5, #Sb, s223sss)
 |— (q, 6, #Sbc, ss223sss)
 |— (q, 6, #SbS, 3ss223sss)
 |— (b, 6, #SbS, 3ss223sss) //quay lui
 |— (b, 6, #Sbc, ss223sss) //hủy thu gọn
 |— (b, 5, #Sb, s223sss) //gạt tự hủy
 |— (b, 4, #S, 223sss) //gạt tự hủy
 |— (q, 5, #aSb, s23sss) //quay lui 3
 |— (q, 6, #aSbc, ss23sss)
 |— (q, 6, #aSbS, 3ss23sss)
 |— (q, 6, #S, 13ss23sss) |— **Chấp nhận** 200

Phân tích quay lui với PL/0

- Cài đặt phức tạp
- Chi phí thời gian quá lớn nếu chương trình phải phân tích gồm nhiều ký hiệu (tùy tố)
- Không thể thông báo lỗi chi tiết

Chương 3: Phân tích cú pháp

1. Giới thiệu
2. Phương pháp phân tích cú pháp quay lui
3. Phương pháp phân tích bảng
4. Phương pháp phân tích cú pháp tất định
5. Xây dựng bộ phân tích cú pháp cho PL/0

Giới thiệu

- Áp dụng với lớp văn phạm phi ngũ cảnh
- Chi phí: (n :độ dài xâu cần phân tích)
 - n^3 về thời gian
 - n^2 về mặt bộ nhớ
 - Nếu văn phạm đơn nghĩa: n^2 về thời gian
- Phương pháp
 1. Giải thuật CYK (Cocke-Younger-Kasami)
 2. Giải thuật Earley

Giải thuật CYK

- Điều kiện áp dụng
 - Văn phạm phi ngũ cảnh ở dạng chuẩn Chomsky (CNF: Chomsky Normal Form)
 - Văn phạm không chứa ϵ -sản xuất (Sản xuất dạng $A \rightarrow \epsilon$) và sản xuất vô ích
- Nguyên tắc
 - Tạo bảng phân tích tam giác theo xâu cần phân tích

Dạng chuẩn Chomsky

- Khái niệm
 - Văn phạm phi ngũ cảnh được gọi là ở dạng chuẩn Chomsky nếu mọi sản xuất đều có dạng $A \rightarrow BC$ hoặc $A \rightarrow a$ ($A, B, C \in V_N$, $a \in V_T$)
- Mọi văn phạm không chứa ϵ -sản xuất đều có thể chuyển về dạng chuẩn Chomsky

Chuyển VPPNC sang dạng chuẩn Chomsky

1. Với mọi sản xuất dạng $A \rightarrow X_1 X_2 \dots X_n$ ($n \geq 2$)

- Nếu $X_i \in V_T$ là ký hiệu kết thúc ($X_i = a$)
 - Thêm ký hiệu $C_a \in V_N$ và sản xuất $C_a \rightarrow a$
 - Thay X_i bởi C_a trong các sản xuất của văn phạm

2. Với mọi sản xuất dạng $A \rightarrow B_1 B_2 \dots B_n$ ($n \geq 3$)

- Thêm các ký hiệu không kết thúc D_1, D_2, \dots, D_{n-2}
- Thay sản xuất $A \rightarrow B_1 B_2 \dots B_n$ bởi tập
 - $A \rightarrow B_1 D_1$
 - $D_1 \rightarrow B_2 D_2$
 - $D_{n-2} \rightarrow B_{n-1} B_n$

Dạng chuẩn Chomsky → Ví dụ

$$S \rightarrow bA \mid aB$$

$$A \rightarrow bAA \mid aS \mid a$$

$$B \rightarrow aBB \mid bS \mid b$$

$$S \rightarrow C_bA \mid C_aB$$

$$A \rightarrow C_bD_1 \mid C_aS \mid a$$

$$B \rightarrow C_aD_2 \mid C_bS \mid b$$

$$D_1 \rightarrow AA \quad C_a \rightarrow a$$

$$D_2 \rightarrow BB \quad C_b \rightarrow b$$

Thêm 2 sản xuất

$$C_b \rightarrow b \quad \text{và} \quad C_a \rightarrow a$$

$$S \rightarrow bA : \quad S \rightarrow C_bA$$

$$S \rightarrow aB : \quad S \rightarrow C_aB$$

$$A \rightarrow bAA : \quad A \rightarrow C_bAA$$

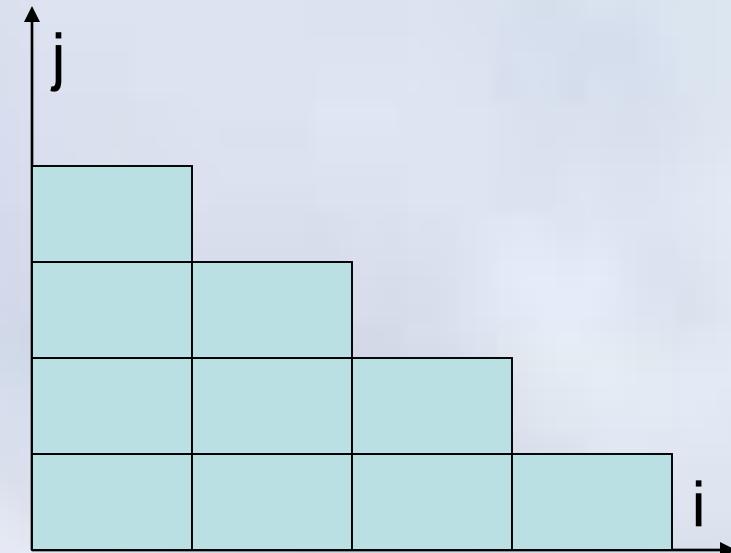
$$A \rightarrow aS : \quad A \rightarrow C_aS$$

$$B \rightarrow aBB : \quad B \rightarrow C_aBB$$

$$B \rightarrow bS : \quad B \rightarrow C_bS$$

Giải thuật CYK → Mô tả bảng

- Bảng dữ liệu T dạng ma trận tam giác dưới, phụ thuộc xâu cần phân tích
- Các phần tử là T_{ij} , i chỉ số cột, j chỉ số hàng
- Mỗi phần tử T_{ij} chứa tập con V_N
- $A \in T_{ij} \Leftrightarrow A \Rightarrow^+ a_i a_{i+1} \dots a_{i+j-1}$
 - j ký hiệu kể từ vị trí i
- Nếu $S \in T_{1n}$
 - $S \Rightarrow^+ a_1 a_2 \dots a_{1+n-1} = \omega$
 - Vậy $\omega \in L(G) \Leftrightarrow S \in T_{1n}$



Giải thuật CYK → Xây dựng bảng

1. Xây dựng các phần tử cho hàng thứ nhất

$$T_{i1} = \{A \mid A \in V_N \text{ và } A \rightarrow a_i\}$$

2. Giả thiết đã xây dựng xong cho $j-1$ hàng

$$T_{ij} = \{A \mid \exists k \ 1 \leq k < j \text{ thỏa mãn}$$

$$B \in T_{ik}, C \in T_{i+k, j-k} \ A \rightarrow BC \in P\}$$

Rõ ràng: $A \in T_{ij}$ thì

$$A \rightarrow BC \Rightarrow^+ a_i a_{i+1} \dots a_{i+k-1} C \ (\text{do } B \in T_{ik})$$

$$\Rightarrow^+ a_i a_{i+1} \dots a_{i+k-1} a_{i+k}, a_{i+k-1} \dots A_{i+k+j-k-1} = a_i \dots a_{i+j-1}$$

3. Lặp lại bước 2 cho tới khi tính xong các phần tử của T

Giải thuật CYK → Xây dựng bảng → Ví dụ

$$S \rightarrow AA \mid AS \mid b$$

$$A \rightarrow SA \mid AS \mid a$$

$$\omega: abaab$$

		j				
		i				
		A, S				
S, A		A, S				
A, S		S	A, S			
S, A	A		S	S, A		
A	S	A	A	A	S	

Below the table, the input string $\omega: abaab$ is shown with labels: a, b, a, a, b under the respective characters.

Giải thuật CYK → Tìm cây phân tích

- Là quá trình ngược với xây dựng bảng
- Sử dụng hàm **Gen(i, j, A)** sinh ra dãy phân tích trái ứng với suy diễn $A \Rightarrow^*_L a_i a_{i+1} \dots a_{i+j-1}$
 - **Gen(1, n, S)** để sinh ra dãy suy diễn trái của xâu cần phân tích
- **Gen(i, j, A)**
 - $j = 1$: $A \rightarrow a_i$ là sản xuất m. Viết ra giá trị m
 - $j > 1$: Nếu **k** là một số nguyên (nhỏ nhất) thỏa mãn: $B \in T_{ik}, C \in T_{i+k, j-k}, A \rightarrow BC \in P$ là s.xuất m
 - Viết ra giá trị **m**
 - Gọi Gen(i, k, B) và Gen (i+k, j-k, C)

Giải thuật CYK → Tìm cây phân tích → Ví dụ

1. $S \rightarrow AA$
2. $S \rightarrow AS$
3. $S \rightarrow b$
4. $A \rightarrow SA$
5. $A \rightarrow AS$
6. $A \rightarrow a$

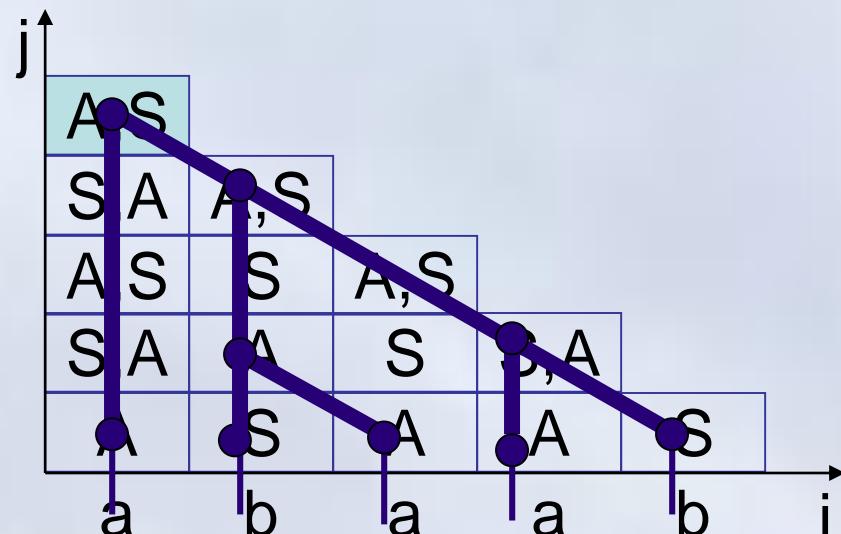
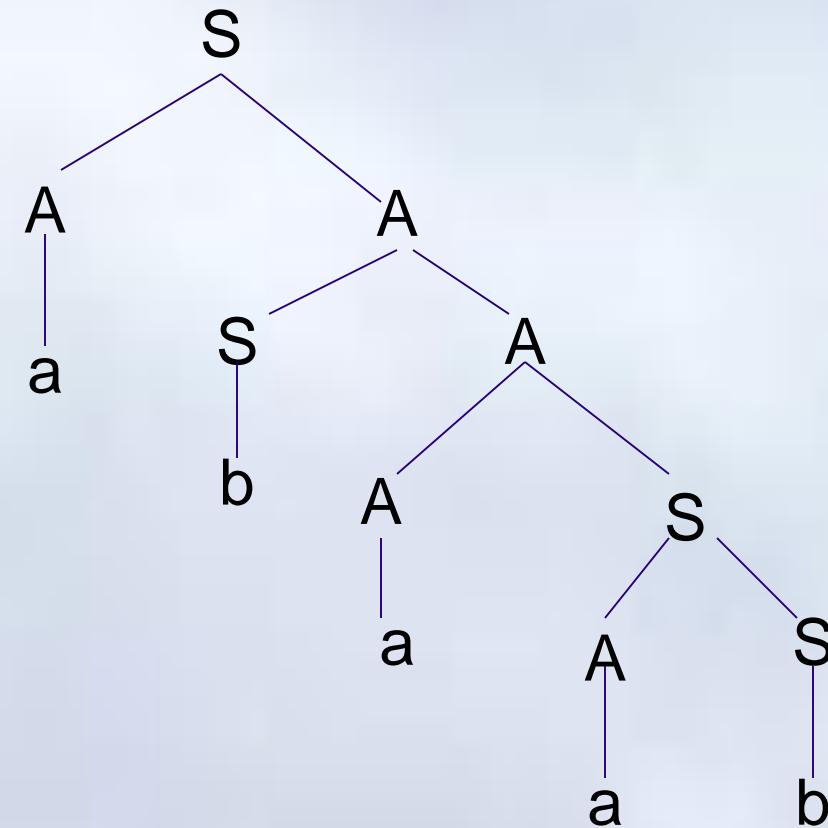
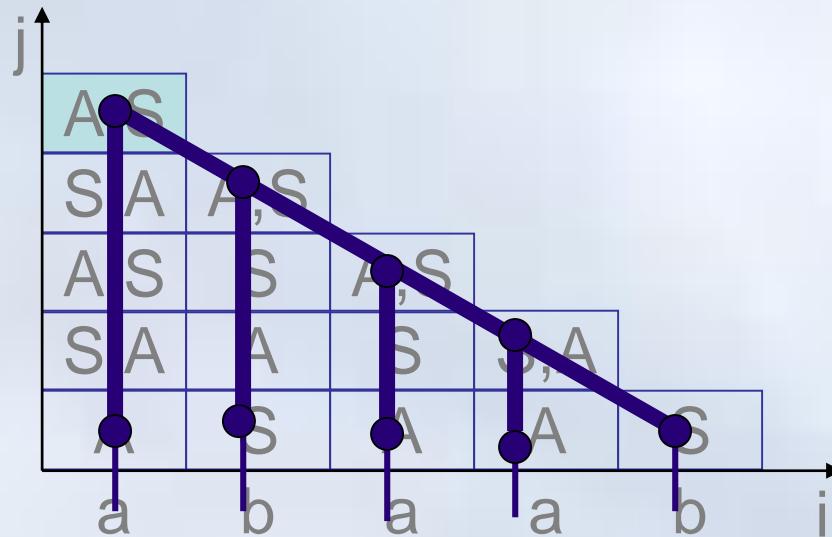
ω : abaab

Gen (1, 5, S)

164356263

262436263

Giải thuật CYK → Tìm cây phân tích



Giải thuật CYK → Bài tập

Cho văn phạm $S \rightarrow aSbS \mid aS \mid c$

Dùng phương pháp CYK phân tích $\omega : aacbc$

Chương 3: Phân tích cú pháp

1. Giới thiệu
2. Phương pháp phân tích cú pháp quay lui
3. Phương pháp phân tích bảng
4. Phương pháp phân tích cú pháp tất định
5. Xây dựng bộ phân tích cú pháp cho PL/0

Nội dung

1. Giới thiệu

2. Phân tích Top-Down

3. Phân tích Bottom-Up

Giới thiệu

Phân tích tất định xem xét một lượt xâu vào từ trái qua phải và tại mỗi bước sẽ xác định được **duy nhất một sản xuất** phù hợp với trạng thái hiện tại

Ví dụ 1

Văn phạm: $S \rightarrow aA$, $A \rightarrow c|bA$

Xâu cần phân tích: abbc

K/hiệu	Xâu phân tích	Sản xuất
S	abbc	$S \rightarrow aA$
aA	abbc	So sánh
A	bbc	$A \rightarrow bA$
A	bc	$A \rightarrow bA$
A	c	$A \rightarrow c$
c	c	So sánh
-	-	

Sản xuất luôn được lựa
chọn duy nhất nhờ xem
trước một ký hiệu

Ví dụ 2

Văn phạm: $S \rightarrow A|B$, $A \rightarrow aA|c$, $B \rightarrow aB|b$

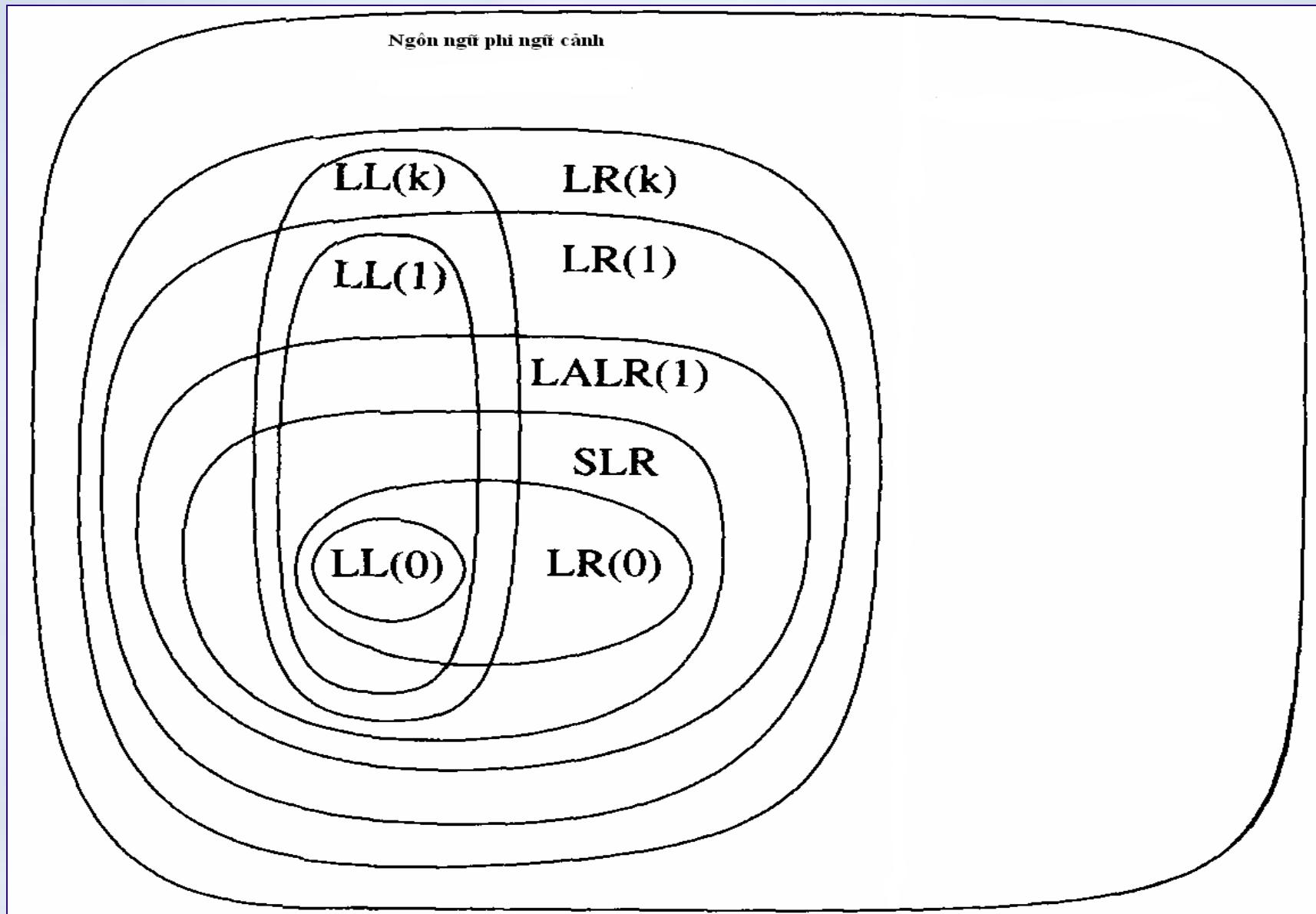
Xâu phân tích: aab

Khiếu	Xâu phân tích	Sản xuất
S	aab	$S \rightarrow A$
A	aab	$A \rightarrow aA$
aA	aab	So sánh
A	ab	$A \rightarrow aA$
aA	ab	So sánh
A	b	?

Nhìn trước một ký hiệu chưa đủ để lựa chọn đúng sản xuất \Rightarrow Phải thực hiện quay lui

Để tránh quay lui, thuật toán phân tích tất định giới hạn trong một số lớp ngôn ngữ: LL(k), LR(k), ...

Phân cấp các ngôn ngữ phi ngữ cảnh



Phân tích Top-Down

- Văn phạm LL(k)
- Văn phạm LL(1)
- Phân tích xem trước
- Phân tích đệ quy trên xuống

Ngôn ngữ LL(k)

- Được sinh ra từ văn phạm LL(k)
- Khi thực hiện phân tích văn phạm LL(k), bộ phân tích cần nhìn trước k ký hiệu để quyết định sản xuất nào sẽ được sử dụng
- Để đạt được tính chất này, văn phạm LL(k) cần thỏa mãn một số điều kiện nào đó

LL(k)

→ k : Số ký hiệu cần nhìn trước

→ **Left**: Thực hiện các suy dẫn trái nhất

→ **Left**: Câu được phân tích từ trái qua phải

FIRST_k(α)

- Cho văn phạm phi ngũ cản G=(V_T, V_N, P, S)
- k là một số nguyên dương, α ∈ V*
- Định nghĩa

$$\begin{aligned}
 \text{FIRST}_k(\alpha) &= \{x \mid x \in V_T^*, |(x)| = k, \alpha \Rightarrow_L^* x\beta\} \\
 &= \{x \mid x \in V_T^*, |(x)| < k, \alpha \Rightarrow_L^* x\}
 \end{aligned}$$

FIRST_k(α) là tập các xâu x gồm k ký hiệu kết thúc trái nhất của các xâu được suy dẫn ra từ α

– Chấp nhận trường hợp xâu x không có đủ k ký hiệu nhưng α suy dẫn ra x và không còn ký hiệu nào sau x

FOLLOW_k(α)

- Cho văn phạm phi ngũ cǎnh $G=(V_T, V_N, P, S)$
- k là một số nguyên dương, $\alpha \in V^*$
- Định nghĩa

$$\text{FOLLOW}_k(\alpha) = \{x \mid S \Rightarrow_L^* \beta\alpha\delta, x \in \text{FIRST}_k(\delta)\}$$

FOLLOW_k(α) là tập tất cả các xâu chứa k ký hiệu kết thúc, **đúng ngay đằng sau α** trong bất kỳ dạng câu nào

– Nếu $\alpha \equiv A \in V_N$ và $S \Rightarrow_L^* \beta A$, $\text{FOLLOW}_1(\alpha) \ni \{\epsilon\}$

Ví dụ

Cho văn phạm

$$S \rightarrow aAb \mid c$$

$$A \rightarrow hS \mid g$$

- $\text{First}_1(S) = \{a, c\}$ $\text{First}_1(A) = \{h, g\}$
- $\text{Follow}_1(S) = \{b\}$ $\text{Follow}_1(A) = \{b\}$
- $\text{First}_2(S) = \{ah, ag, c\}$ $\text{First}_2(A) = \{ha, hc, g\}$
- $\text{Follow}_2(S) = \{b, bb\}$ $\text{Follow}_1(A) = \{b, bb\}$
- $\text{First}_3(S) = \{aha, ahc, agb, c\}$
- $\text{First}_3(A) = \{hah, hag, hc, g\}$
- $\text{Follow}_3(S) = \{b, bb, bbb\}$ $\text{Follow}_3(A) = \{b, bb, bbb\}$

Sản xuất LL(k)

Cho văn phạm phi ngũ cảnh $G = (V_T, V_N, P, S)$,
một sản xuất từ $A \in V_N$ là sản xuất LL(k) nếu
thỏa mãn các điều kiện

1. $S \Rightarrow_L^* xAa \Rightarrow_L x\beta_1 a \Rightarrow_L^* xZ_1 \quad (x \in V_T^*)$
2. $S \Rightarrow_L^* xAa \Rightarrow_L x\beta_2 a \Rightarrow_L^* xZ_2$
3. Nếu $\text{FIRST}_k(Z_1) = \text{FIRST}_k(Z_2)$ thì $\beta_1 = \beta_2$

Văn phạm LL(k)

- Văn phạm phi ngũ cảnh $G = (V_T, V_N, P, S)$, là LL(k) nếu mọi sản xuất của G đều là sản xuất LL(k)
- Ý nghĩa
 - Nếu G là LL(k) và $xA\alpha$ là một dạng câu, khi đó nếu biết được k ký hiệu kết thúc được suy dẫn ra từ $A\alpha$ thì chỉ *tồn tại duy nhất* một sản xuất từ A thỏa mãn

Ví dụ

Văn phạm $S \rightarrow aAS \mid b$, $A \rightarrow bSa \mid a$
 là văn phạm LL(1)

- Xét các sản xuất từ S : $S \rightarrow aAS \mid b$
 - Xét các cặp suy dẫn trái

$$S \Rightarrow_L^* xSa \Rightarrow_L x\beta_1\alpha \Rightarrow_L^* xZ_1$$

$$S \Rightarrow_L^* xSa \Rightarrow_L x\beta_2\alpha \Rightarrow_L^* xZ_2$$
 - Nếu $\text{FIRST}_1(Z_1) = \text{FIRST}_1(Z_2) = \{a\}$ thì từ S phải dùng sản xuất có ký hiệu **a** ở đầu: $S \rightarrow aAS$
 - Nếu $\text{FIRST}_1(Z_1) = \text{FIRST}_1(Z_2) = \{b\}$ dùng: $S \rightarrow b$
- Tương tự với các sản xuất từ A : $A \rightarrow bSa \mid a$

Ví dụ

Văn phạm $S \rightarrow aAa \mid bAba$, $A \rightarrow b \mid \epsilon$

Không phải văn phạm **LL(1)**

Xét sản xuất $S \rightarrow bAba$, nếu đang ở ký hiệu không kết thúc A và nhìn trước **b**, sẽ không biết cần sử dụng sản xuất $A \rightarrow b$ hay $A \rightarrow \epsilon$

Là văn phạm **LL(2)**

- Nếu đang ở ngũ cảnh **aAa**, và nhìn trước được ba, phải áp dụng $A \rightarrow b$, còn nếu nhìn trước được **a#**, phải áp dụng $A \rightarrow \epsilon$
- Nếu đang ở ngũ cảnh **bAba**, và nhìn trước được **bb**, phải áp dụng $A \rightarrow b$, còn nếu nhìn trước được **ba**, phải áp dụng $A \rightarrow \epsilon$

Điều kiện để văn phạm là LL(k)

Cho văn phạm phi ngũ cảnh $G = (V_T, V_N, P, S)$,

G là văn phạm LL(k) khi và chỉ khi thỏa mãn

- Nếu $xA\alpha$ là một dạng câu ($S \Rightarrow_L^* xA\alpha$)
 $A \rightarrow \beta_1 \in P$ và $A \rightarrow \beta_2 \in P$ ($\beta_1 \neq \beta_2$)
- Thì $\text{FIRST}_k(\beta_1\alpha) \cap \text{FIRST}_k(\beta_2\alpha) = \emptyset$

Văn phạm LL(1)

- Văn phạm LL(k): nhắc tới trong lý thuyết
 - Sử dụng thực tế, $k = 1$.
- Quy ước
 - $\text{FIRST}_1(\alpha) \equiv \text{FIRST}(\alpha)$
 - $\text{FOLLOW}_1(\alpha) \equiv \text{FOLLOW}(\alpha)$
- Định nghĩa lại tập FIRST và FOLLOW
 - Đễ dễ dàng tính toán
 - Thường quan tâm tới FOLLOW của một ký hiệu không kết thúc

FIRST(α)

$$\begin{aligned} \text{FIRST}(\alpha) &= \{ a \mid a \in V_T, \alpha \Rightarrow_L^* a\beta \} \text{ hoặc} \\ &= \{ \epsilon \mid \alpha \Rightarrow_L^* \epsilon \} \end{aligned}$$

Tập tất cả các ký hiệu kết thúc, đứng đầu một chuỗi được suy dẫn ra từ α

FIRST(a) → Tính chất

- $a \in V_T^*$ $\quad \text{FIRST}(a) = \{a\}$
 $\quad \text{FIRST}(\varepsilon) = \{\varepsilon\}$
- $A \rightarrow a\gamma \quad \text{FIRST}(A) \supseteq \{a\}$
- $A \rightarrow B\gamma \quad \text{FIRST}(A) \supseteq \text{FIRST}(B) - \{\varepsilon\}$
 - Nếu $b \in \text{FIRST}(B)$ thì $B \Rightarrow_L^* b\beta$
Vậy $A \rightarrow B\gamma \Rightarrow_L^* b\beta\gamma$. Do đó $\text{FIRST}(A) \supseteq \{b\}$
 - Xét trường hợp $A \rightarrow Ba$ và $B \Rightarrow^* \varepsilon$
 $\text{FIRST}(B) \supseteq \{\varepsilon\}$ nhưng $\{\varepsilon\} \not\subset \text{FIRST}(A)$
- $A \rightarrow B\gamma$ và $B \Rightarrow^* \varepsilon$
 $\text{FIRST}(A) \supseteq (\text{FIRST}(B) - \{\varepsilon\}) \cup \text{FIRST}(\gamma)$

FOLLOW(A)

$$\text{FOLLOW}(A) = \{ a \mid a \in V_T, S \Rightarrow_L^* \alpha A a \beta \}$$

- Tập tất cả các ký hiệu kết thúc có thể đứng ngay sau ký hiệu không kết thúc A trong một dạng câu nào đó
- Trường hợp đặc biệt, nếu A là ký hiệu bên phải nhất của một dạng câu ($S \Rightarrow_L^* \alpha A$) thì $\text{FOLLOW}(A) \supseteq \{\epsilon\} // \text{FOLLOW}(A) \supseteq \{\#\}$
 - Đánh dấu không có ký hiệu nào ở bên phải A

FOLLOW(A) → Tính chất

- $A \rightarrow \gamma B \alpha$ $\text{FOLLOW}(B) \supseteq \text{FIRST}(\alpha) - \{\epsilon\}$
 - Nếu $\text{FIRST}(\alpha) \supseteq \{a\}$, $\alpha \Rightarrow_L^* a\beta$.
Vì $A \rightarrow \gamma B \alpha$ nên $A \Rightarrow_L^* \gamma B a\beta$
Vậy $\text{FOLLOW}(B) \supseteq \{a\}$
 - Xét $S \rightarrow Aa$, $A \rightarrow \gamma B \alpha$ và $\alpha \Rightarrow_L^* \epsilon$
 $\{\epsilon\} \subseteq \text{FIRST}(\alpha)$ nhưng $\{\epsilon\} \not\subseteq \text{FOLLOW}(B)$
- $A \rightarrow \gamma B$ hoặc $A \rightarrow \gamma B \beta$ và $\beta \Rightarrow_L^* \epsilon$ $\text{FOLLOW}(B) \supseteq \text{FOLLOW}(A)$
 - $\{a\} \subseteq \text{FOLLOW}(A)$ thì $S \Rightarrow_L^* aAa\beta$
Do $A \rightarrow \gamma B$ nên $S \Rightarrow_L^* aAa\beta \Rightarrow a\gamma B a\beta$

Ví dụ

Cho văn phạm: $E \rightarrow T \mid E+T$ $T \rightarrow F \mid T^*F$ $F \rightarrow a \mid (E)$ $\text{FIRST}(F) = \{a, (\}$ $\text{FIRST}(T) = \text{FIRST}(F) = \{a, (\}$ $\text{FIRST}(E) = \text{FIRST}(T) = \{a, (\}$ $\text{FOLLOW}(E) = \{ +,) \ , \ \# \}$ $\text{FOLLOW}(T) = \text{FOLLOW}(T) \cup \{ * \} = \{ +,), * \ , \ \# \}$ $\text{FOLLOW}(F) = \text{FOLLOW}(T) = \{ +,), * \ , \ \# \}$

Toán tử \oplus

Cho V_1 và V_2 là 2 tập ký hiệu.

Định nghĩa toán tử \oplus

$$\begin{aligned} V_1 \oplus V_2 &= V_1 \text{ nếu } \varepsilon \notin V_1 \\ &= (V_1 - \{\varepsilon\}) \cup V_2 \text{ nếu } \varepsilon \in V_1 \end{aligned}$$

Ví dụ

$$\{a, b\} \oplus \{c, d\} = \{a, b\}$$

$$\{\varepsilon, a, b\} \oplus \{c, d\} = \{a, b, c, d\}$$

$$\{ \} \oplus \{a, b\} = \{ \} \quad // Do \text{ tập rỗng không chứa } \varepsilon$$

Bổ đề

$$\text{FIRST}(\alpha\beta) = \text{FIRST}(\alpha) \oplus \text{FIRST}(\beta)$$

$\forall a \neq \epsilon \text{ và } a \in \text{FIRST}(\alpha)$

- Định nghĩa \oplus : $\text{FIRST}(\alpha) \oplus \text{FIRST}(\beta) \ni a$
- Định nghĩa FIRST: $\alpha \Rightarrow_L^* a\gamma$

Vậy $\text{FIRST}(\alpha\beta) = \text{FIRST}(a\gamma\beta) \ni a$

$\epsilon \in \text{FIRST}(\alpha)$

- Định nghĩa \oplus :

$$\text{FIRST}(\alpha) \oplus \text{FIRST}(\beta) = (\text{FIRST}(\alpha) - \{\epsilon\}) \cup \text{FIRST}(\beta)$$
- Định nghĩa FIRST, $\alpha \Rightarrow_L^* \epsilon$.
 - Dựa trên tính chất FIRST ($A \rightarrow B\gamma$ và $B \Rightarrow^* \epsilon$)
$$\text{FIRST}(\alpha\beta) = (\text{FIRST}(\alpha) - \{\epsilon\}) \cup \text{FIRST}(\beta)$$

Tính toán FIRST(α)

Nếu $\alpha = X_1 X_2 \dots X_n$ với $X_i \in V$

$$\begin{aligned} \text{FIRST}(\alpha) = \text{FIRST}(X_1) \oplus \\ \text{FIRST}(X_2) \oplus \dots \oplus \text{FIRST}(X_n) \end{aligned}$$

Thuật toán tính FIRST(A), $A \in V_N$ Xây dựng các tập $F_i(X)$ với $X \in V$

1. $a \in V_T$ $F_i(a) = \{a\} \forall i$
2. $F_0(A) = \{a | a \in V_T \text{ và } A \rightarrow a \in P\} \cup \{\epsilon | A \rightarrow \epsilon \in P\}$
3. Giả thiết đã tính được tập F_0, F_1, \dots, F_{i-1} , tính F_i
 - $F_i(A) = \{a | a \in V_T, \text{ Nếu } A \rightarrow X_1 X_2 \dots X_n \in P, \text{ thì } a \in F_{i-1}(X_1) \oplus F_{i-1}(X_2) \oplus \dots \oplus F_{i-1}(X_n)\}$
4. Thuật toán dừng khi các F_i không đổi (luôn đúng vì $F_{i-1}(X) \subseteq F_i(X) \subseteq V_T$).

Ví dụ tính FIRST

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

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

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

X	$F_0(X)$	$F_1(X)$	$F_2(X)$	$F_3(X)$	
E	\emptyset	\emptyset	a, (a, (Không đổi
T	\emptyset	a, (a, (Không đổi
F	a, (a, (Không đổi

Thuật toán tính FOLLOW(A)

Thực hiện các quy tắc sau cho tới khi không xuất hiện các thay đổi trong tập FOLLOW

1. Đặt # vào tập FOLLOW(S)

- S là ký hiệu khởi đầu
- # $\notin V$, dùng đánh dấu kết thúc chuỗi cần phân tích

2. Với mọi sản xuất có dạng $A \rightarrow \alpha B \beta$, thêm FIRST(β) - $\{\epsilon\}$ vào FOLLOW(B)

3. Nếu tồn tại sản xuất $A \rightarrow \alpha B$ hoặc có $A \rightarrow \alpha B \beta$ mà $\{\epsilon\} \subseteq \text{FIRST}(\beta)$ thì thêm FOLLOW(A) vào FOLLOW(B)

Ví dụ tính FOLLOW

VP: $E \rightarrow T \mid E+T, \quad T \rightarrow F \mid T^*F, \quad F \rightarrow a \mid (E)$

Quy tắc	Sản xuất áp dụng	FOLLOW		
		E	T	F
1	Khởi tạo	#	\emptyset	\emptyset
2	$E \rightarrow E+T$	+ , #		
	$T \rightarrow T^*F$		*	
	$F \rightarrow (E)$) , + , #		
3	$E \rightarrow E+T$		* ,) , + , #	
	$E \rightarrow T$			
	$T \rightarrow T^*F$			* ,) , + , #
	$T \rightarrow F$			
FOLLOW (X)) , + , #	* ,) , + , #	* ,) , + , #

Ví dụ

$$S \rightarrow BA$$

$$A \rightarrow + BA \mid \epsilon$$

$$B \rightarrow DC$$

$$C \rightarrow * DC \mid \epsilon$$

$$D \rightarrow (S) \mid a$$

Tính FIRST(X) và FOLLOW(X) với $X \in V_N$

Ví dụ tính FIRST

X	$F_0(X)$	$F_1(X)$	$F_2(X)$	$F_3(X)$	FIRST
S	\emptyset	\emptyset	(, a	(, a	{ (, a }
A	+, ϵ	+, ϵ			{ +, ϵ }
B	\emptyset	(, a	(, a		{ (, a }
C	*, ϵ	*, ϵ			{ *, ϵ }
D	(, a	(, a			{ (, a }

Ví dụ tính FOLLOW

Quy tắc	Sản xuất áp dụng	FOLLOW				
		S	A	B	C	D
1	Khởi tạo	#				
2	$S \rightarrow BA, A \rightarrow +BA$			+		
	$B \rightarrow DC, C \rightarrow *DC$					*
	$D \rightarrow (S)$)				
3	$S \rightarrow BA$		#,)			
	$A \rightarrow +BA, A \rightarrow \epsilon$			#,)		
	$B \rightarrow DC$				#, +,)	
	$C \rightarrow *DC, C \rightarrow \epsilon$					#, +,)
FOLLOW(X)		#,)	#,)	#, +,)	#, +,)	* , #, +,)

Điều kiện để văn phạm là LL(1)

Văn phạm phi ngũ cản G = (V_T, V_N, P, S) là LL(1) nếu thỏa mãn các điều kiện sau

- Mọi sản xuất dạng $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$, $n \geq 2$ thỏa mãn $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \emptyset$, $i \neq j$

Ý nghĩa: Nếu $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ thì các α_i phải bắt đầu bởi các ký hiệu kết thúc khác nhau

- $\forall A \in V_N, A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ và $\exists \alpha_j \Rightarrow^* \varepsilon$ thì $\text{FIRST}(\alpha_i) \cap \text{FOLLOW}(A) = \emptyset \forall i \neq j$

Ý nghĩa: Nếu $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n | \varepsilon$ và $b \in \text{FOLLOW}(A)$ thì không có nhánh α_i nào được bắt đầu bởi ký hiệu b

Văn phạm là LL(1) đơn giản

Văn phạm phi ngũ cành $G = (V_T, V_N, P, S)$ là LL(1) đơn giản nếu mọi sản xuất của văn phạm có dạng

$$A \rightarrow a_1 \alpha_1 \mid a_2 \alpha_2 \mid \dots \mid a_n \alpha_n$$

Trong đó $a_i \in V_T$ và $a_i \neq a_j$ với $i \neq j$

Ví dụ về văn phạm LL(1)

1. $E \rightarrow T \mid E+T$
2. $T \rightarrow F \mid T^*F$
3. $F \rightarrow a \mid (E)$

- Sản xuất (3) là sản xuất LL(1) đơn giản
 - Xét sản xuất (1)
 - $\text{FIRST}(T) = \{ a, (\}$
 - $\text{FIRST}(E) = \{ a, (\}$
 - Vậy $\text{FIRST}(T) \cap \text{FIRST}(E) = \{ a, (\} \neq \emptyset$
- \Rightarrow Văn phạm không phải là LL(1)

Ví dụ về văn phạm LL(1)

1. $S \rightarrow BA$
2. $A \rightarrow + BA \mid \epsilon$
3. $B \rightarrow DC$
4. $C \rightarrow * DC \mid \epsilon$
5. $D \rightarrow (S) \mid a$

- Sản xuất (1) &(3) là các sản xuất đơn
- Sản xuất (5) là sản xuất LL(1) đơn giản
- Sản xuất (2)
 - $\text{FIRST}(+BA) \cap \text{FOLLOW}(A) = \{+\} \cap \{\#, \)\} = \emptyset$
- Sản xuất (4)
 - $\text{FIRST}(*DC) \cap \text{FOLLOW}(C) = \{*\} \cap \{+, \#, \)\} = \emptyset$

Khử đệ quy trái

- Các văn phạm LL(1) không đệ quy trái
 - Có thể khử đệ quy trái của VPPNC
- Đệ quy trái (*trực tiếp*)
 - Sản xuất: $A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \dots \mid \beta_m$ với $\beta_i[1] \neq A$
 - Đệ qui trái dùng khi áp dụng SX $A \rightarrow \beta_1 \mid \dots \mid \beta_m$
 - Từ A sinh ra các xâu dạng

$$(\beta_1 \mid \dots \mid \beta_m) (\alpha_1 \mid \dots \mid \alpha_n)^*$$
 - Xâu trên được sinh ra tự văn phạm
 - $A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_m A'$
 - $A' \rightarrow \varepsilon \mid \alpha_1 A' \mid \dots \mid \alpha_n A'$

Khử đệ quy trái

- Ví dụ:
 - Văn phạm $A \rightarrow Aa | b$ đệ quy trái trực tiếp
 - Tương đương với văn phạm không đệ quy trái $A \rightarrow bB; B \rightarrow \epsilon | aB$
 - Ngôn ngữ được sinh ra bởi 2 VP: ba^*
- Ghi chú
 - Loại bỏ đệ quy trái \Rightarrow cây phân tích bị thay đổi
 - Cần phải cấu trúc lại cây phân tích để có được cấu trúc gốc

Khử đệ quy trái → Ví dụ

$$1. \ E \rightarrow T \mid E+T$$

$$2. \ T \rightarrow F \mid T^*F$$

$$3. \ F \rightarrow a \mid (E)$$

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

$$E \rightarrow TE'$$

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

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

$$T \rightarrow FT'$$

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

$$S \rightarrow BA$$

$$A \rightarrow + BA \mid \epsilon$$

$$B \rightarrow DC$$

$$C \rightarrow * DC \mid \epsilon$$

$$D \rightarrow (S) \mid a$$

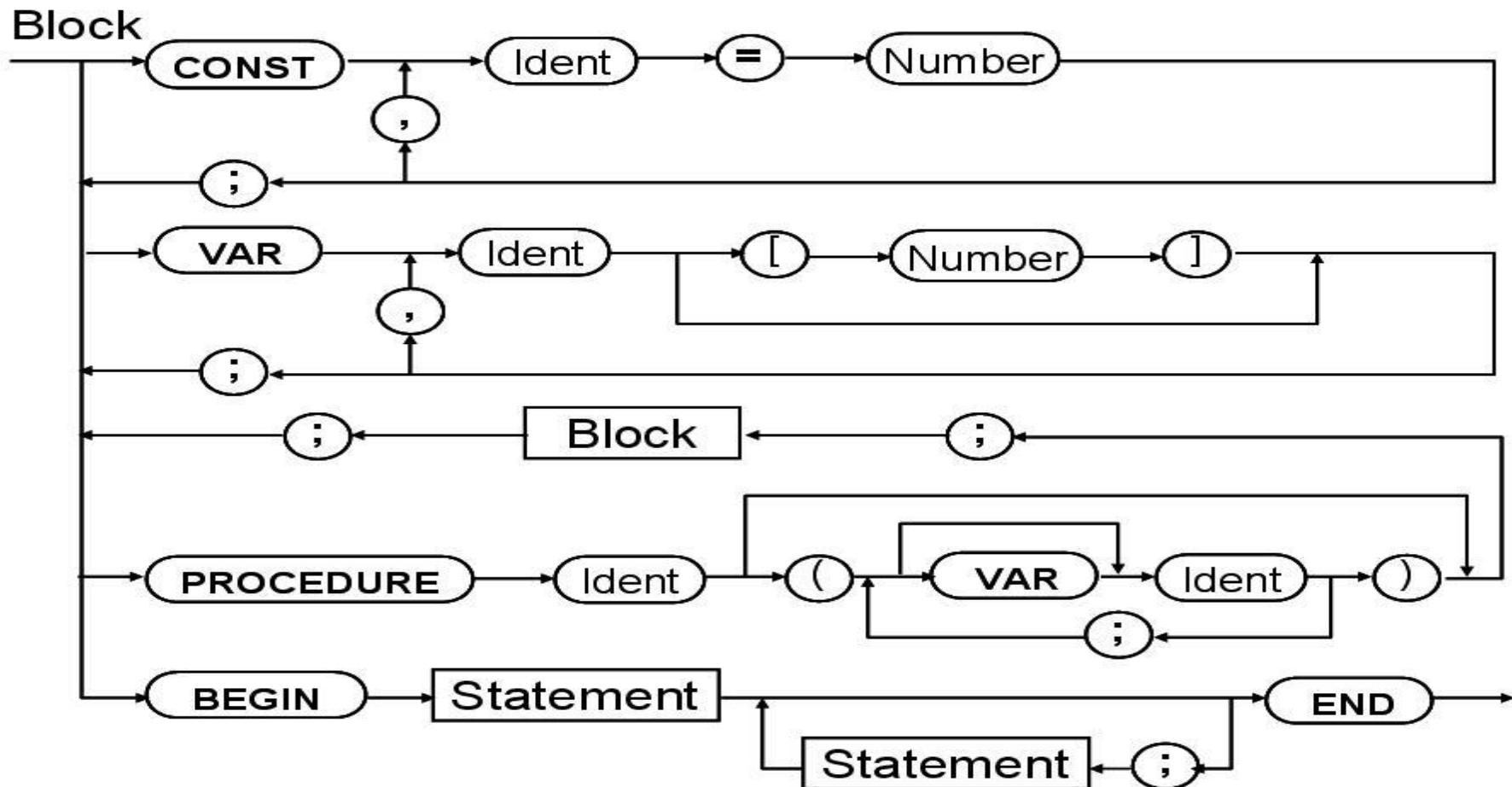
Văn phạm là LL(1) trên sơ đồ cú pháp

- Ở mỗi lối rẽ, các nhánh phải bắt đầu bằng các ký hiệu khác nhau
- Nếu sơ đồ có chứa một đường rỗng (*statement*) thì mọi ký hiệu đứng sau ký hiệu được biểu diễn bởi sơ đồ phải khác các ký hiệu đứng đầu các nhánh của sơ đồ

Văn phạm PL/0 là LL(1) ?

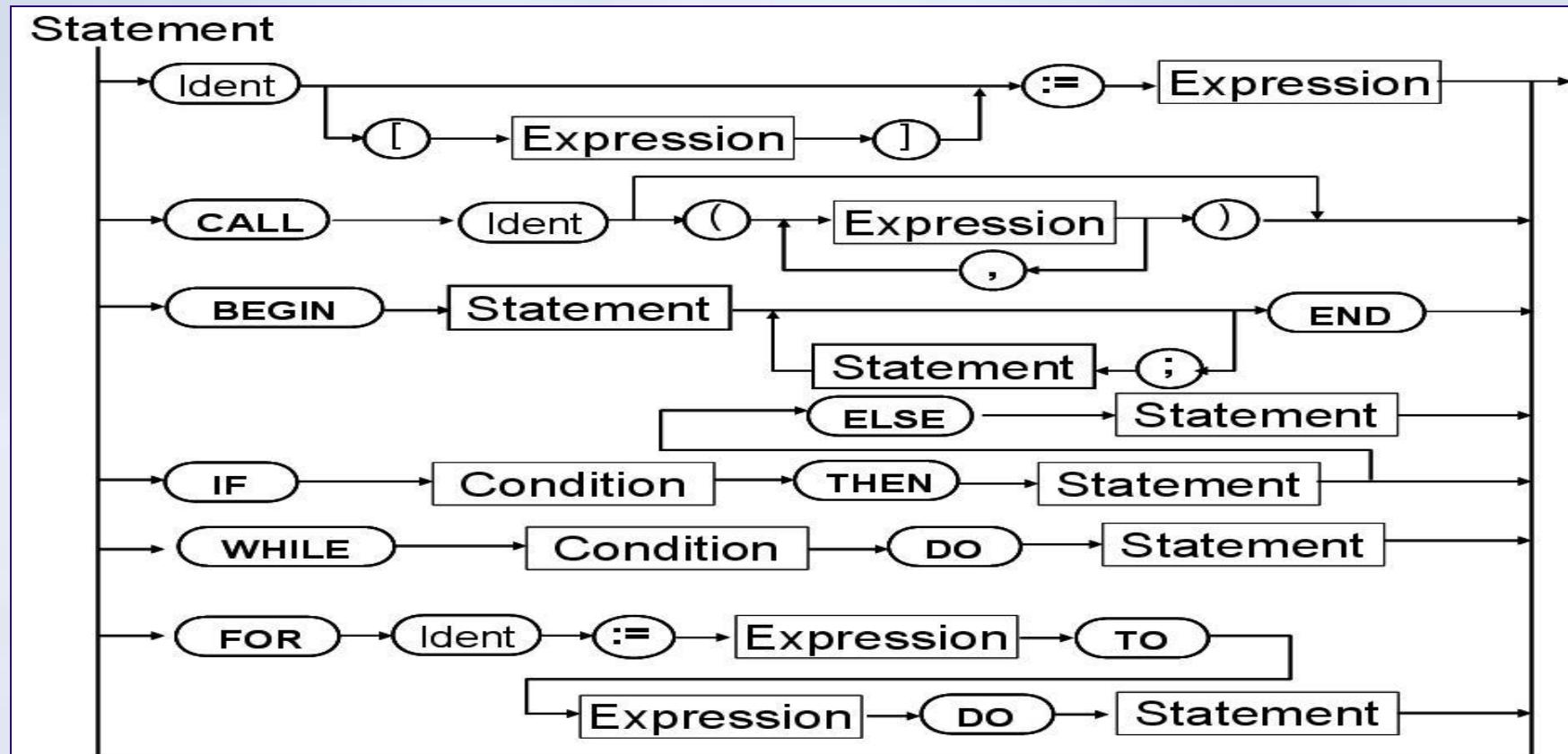
A	FIRST(A)	FOLLOW(A)
Block	CONST VAR PROCEDURE BEGIN	· ;
Statement	Ident CALL BEGIN IF WHILE FOR, ε	; END
Expression	+ - (ident number	; END TO THEN DO) -] < <= > = = !=
Condition	+ - (ident number, ODD	THEN DO
Term	Ident number (; END TO THEN DO) - + < <= > = = !=]
Factor	Ident number (; END TO THEN DO + - * /) < <= > = = !=]

Văn phạm PL/0 là LL(1) ?



Các nhánh của sơ đồ bắt đầu bởi một từ khóa (từ tố) khác nhau (CONST, VAR, PROCEDURE, BEGIN). Vậy **<Block>** là một sản xuất LL(1) đơn giản

Văn phạm PL/0 là LL(1) ?



Điều kiện 1: Các nhánh bắt đầu bởi các từ tô khác nhau

Điều kiện 2: (*Thỏa mãn*)

FOLLOW(<Statement>) = {END, ., ; }

FIRST(<Statement>)= {Ident, CALL, BEGIN, IF, WHILE, FOR}

Thuật toán phân tích xem trước

- Chỉ nghiên cứu với văn phạm LL(1)
 - Cho phép phân tích đúng nhờ đọc trước 1 ký hiệu trên xâu vào
- Bao gồm 2 giai đoạn
 - Xây dựng ma trận phân tích
 - Thực hiện phân tích xem trước

Xây dựng ma trận phân tích

Xây dựng ma trận phân tích M trên tập:

$$(V \cup \{\#\}) \times (V_T \cup \{\#\})$$

1. Nếu $A \rightarrow \alpha$ là sản xuất thứ k của P

- Nếu $\forall a \neq \epsilon$ và $a \in \text{FIRST}(\alpha)$ thì $M[A, a] = (\alpha, k)$
- Nếu $\epsilon \in \text{FIRST}(\alpha)$ thì

$$\forall b \in \text{FOLLOW}(A), M[A, b] = (\alpha, k)$$

2. $M[a, a] = \text{Đây}$

3. $M[\#, \#] = \text{Nhận}$

4. Mọi ô khác có giá trị sai

Xây dựng ma trận phân tích → Ví dụ

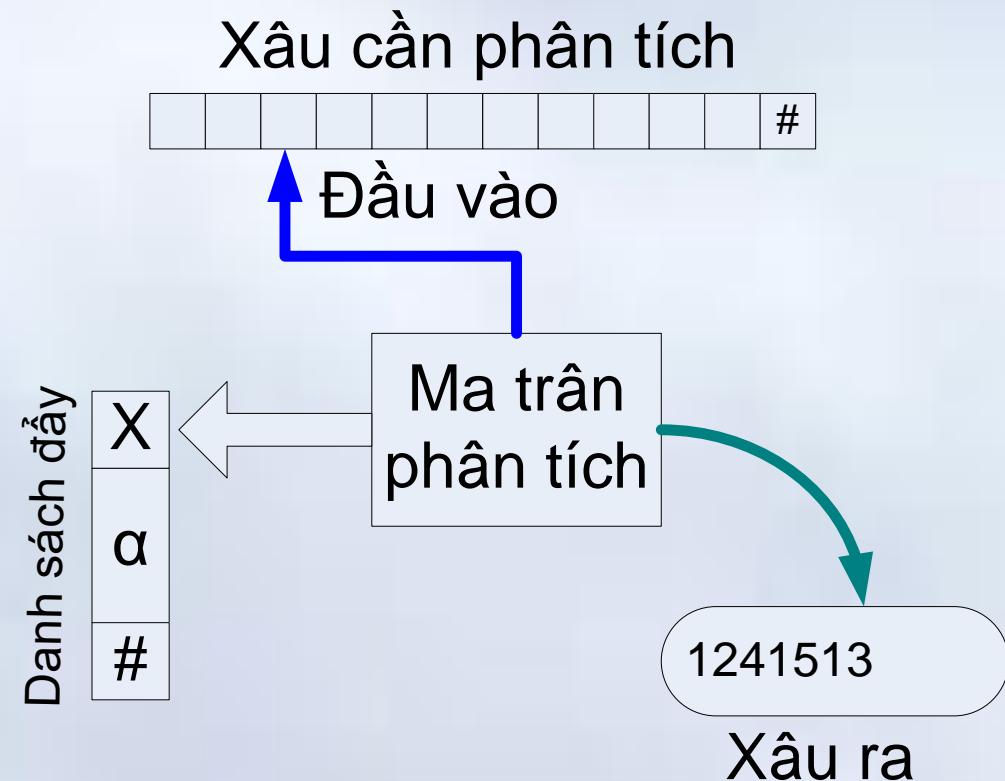
1. $S \rightarrow aAS$
2. $S \rightarrow b$
3. $A \rightarrow a$
4. $A \rightarrow bSA$

M	a	b	#
S	aAS, 1	b, 2	Sai
A	a, 3	bSA, 4	Sai
a	Đẩy	Sai	Sai
b	Sai	Đẩy	Sai
#	Sai	Sai	Nhận

Phân tích xem trước → Mô tả

Thuật toán gồm

- Xâu cần phân tích
 - Đọc bởi một đầu vào
- Ma trận phân tích M
- Danh sách đầy
 - Chứa xâu $X\alpha\#$
 - $\#$ Là đáy danh sách
 - $X\alpha \in V^*$
- Xâu ra
 - Viết bởi một đầu ghi
 - Chứa số thứ tự các SX

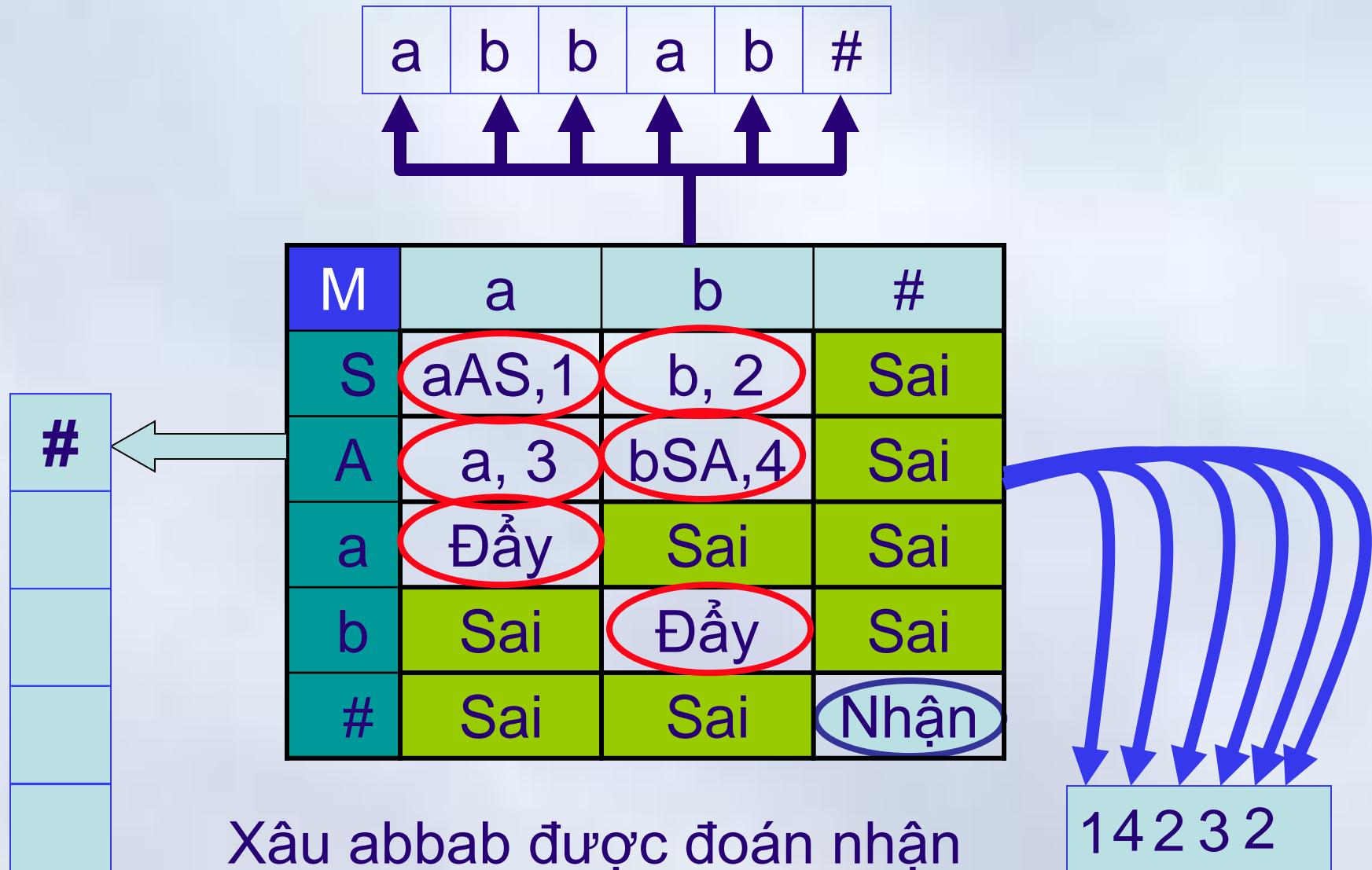


Phân tích xem trước → Hoạt động

Thuật toán dựa vào ma trận phân tích M, ký hiệu đọc được bởi đầu đọc và ký hiệu trên đỉnh Stack để quyết định công việc

- **M[X,a] = {β, k}**: X trên đỉnh DS được thay bằng xâu β (ký hiệu đầu của β ở trên); Viết k ra xâu ra
- **M[X,a] = “đẩy”**: Đầu đọc dịch phải một đơn vị, X bị loại bỏ khỏi danh sách
- **M[X,a] = “sai”**: Dừng lại và thông báo xâu không được đoán nhận
- **M[X,a] = “nhận”**: Thuật toán dừng, xâu được đoán nhận và xâu ra chứa DS các SX đã dùng

Phân tích xem trước → Ví dụ



Phân tích xem trước → Hình trạng thuật toán

- Hình trạng của thuật toán là bộ 3 ($x\#, a\#, \pi$)
 - $x \in V_T^*$ là phần chưa xét của xâu vào
 - $\#$ là ký hiệu đánh dấu kết thúc xâu cần phân tích
 - $a \in V$ là nội dung danh sách đầy
 - $\#$ là ký hiệu nằm dưới của danh sách đầy. $\# \notin V$
 - π nội dung của xâu ra
- Hình trạng ban đầu ($\omega\#, S\#, \epsilon$)
 - ω là xâu cần phân tích
- Hình trạng cuối nếu thành công ($\#, \#, \pi$)
 - π là danh sách các sản xuất đã sử dụng

Phân tích xem trước → Ví dụ

Xét xâu $\omega \equiv \text{abbab}$

(abbab#, S#, ε)

$\rightarrow (\text{abbab#}, \text{aAS}\#, 1) \rightarrow^d (\text{bbab#}, \text{AS}\#, 1)$

$\rightarrow (\text{bab#}, \text{bSAS}\#, 14) \rightarrow^d (\text{bab#}, \text{SAS}\#, 14)$

$\rightarrow (\text{bab#}, \text{bAS}\#, 142) \rightarrow^d (\text{ab}\#, \text{AS}\#, 142)$

$\rightarrow (\text{ab}\#, \text{aS}\#, 1423) \rightarrow^d (\text{b}\#, \text{S}\#, 1423)$

$\rightarrow (\text{b}\#, \text{b}\#, 14232) \rightarrow^d (\#\#, \#\#, 14232)$

\rightarrow Nhận

Phân tích xem trước → Ví dụ 2

Cho văn phạm

- | | |
|-----------------------------|--|
| 1. $S \rightarrow BA$ | $\text{FIRST}(S) = \text{FIRST}(B) = \text{FIRST}(D) = \{ (, a \}$ |
| 2. $A \rightarrow +BA$ | $\text{FIRST}(A) = \{ +, \epsilon \}$ |
| 3. $A \rightarrow \epsilon$ | $\text{FIRST}(C) = \{ *, \epsilon \}$ |
| 4. $B \rightarrow DC$ | $\text{FOLLOW}(S) = \text{FOLLOW}(A) = \{ \#,) \}$ |
| 5. $C \rightarrow *DC$ | $\text{FOLLOW}(B) = \text{FOLLOW}(C) = \{ +, \#,) \}$ |
| 6. $C \rightarrow \epsilon$ | $\text{FOLLOW}(D) = \{ *, +, \#,) \}$ |
| 7. $D \rightarrow (S)$ | Văn phạm đã cho là LL(1) |
| 8. $D \rightarrow a$ | |

Ví dụ 2 → Ma trận phân tích

M	a	()	+	*	#
S	BA, 1	BA, 1				
A			$\varepsilon, 3$	+BA, 2		$\varepsilon, 3$
B	DC, 4	DC, 4				
C			$\varepsilon, 6$	$\varepsilon, 6$	*DC, 5	$\varepsilon, 6$
D	a, 8	(S), 7				
a	Đẩy					
(Đẩy				
)			Đẩy			
+				Đẩy		
*					Đẩy	
#						Nhận

Ví dụ 2 → Phân tích xâu (a*a)

[(a*a)#[,S#,ε]

\rightarrow [(a*a)#[,BA#,1] \rightarrow [(a*a)#[,DCA#,14]
 \rightarrow [(a*a)#[,(S)CA#,147] \rightarrow^d [a*a)#[,S)CA#,147]
 \rightarrow [a*a)#[,BA)CA#,1471] \rightarrow [a*a)#[,DCA)CA#,14714]
 \rightarrow [a*a)#[,aCA)CA#,147148]
 \rightarrow^d [*a)#[,CA)CA#,147148]
 \rightarrow [*a)#[,*DCA)CA#,1471485]
 \rightarrow^d [a)#[,DCA)CA#,1471485]
 \rightarrow [a)#[,aCA)CA#,14714858]
 \rightarrow^d [)#[,CA)CA#,14714858] \rightarrow [)#[,A)CA#,147148586]
 \rightarrow [)#[,)CA#,1471485863] \rightarrow^d [#[,CA#,1471485863]
 \rightarrow [#[,A#,14714858636] \rightarrow [#, #,147148586363]
 \rightarrow Nhận

Phương pháp đệ quy trên xuống

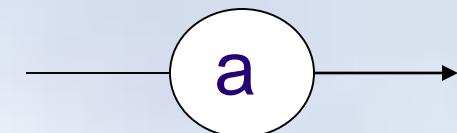
- Sử dụng để phân tích cú pháp cho các văn phạm LL(1)
- Có thể mở rộng cho văn phạm LL(k), nhưng việc tính toán phức tạp
- Sử dụng để phân tích văn phạm khác có thể dẫn đến lặp vô hạn
- Bộ phân tích gồm một tập thủ tục, mỗi thủ tục ứng với một sơ đồ cú pháp (*một ký hiệu không kết thúc*)

Phương pháp đệ quy trên xuống

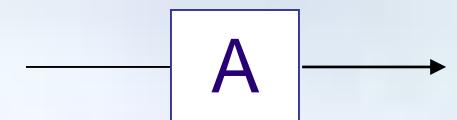
- Với mỗi ký hiệu không kết thúc, lập một sơ đồ cú pháp tương ứng
 - Tên sơ đồ, là tên ký hiệu không kết thúc
- Xây dựng chương trình bởi diễn dịch mỗi sơ đồ cú pháp thành một thủ tục tương ứng
 - Tên thủ tục là tên sơ đồ
 - Mỗi thủ tục có nhiệm vụ triển khai một đoạn nào đó trên văn bản nguồn
- Quá trình phân tích bắt đầu từ thủ tục S và gọi tới các thủ tục khác một cách đệ quy

Xây dựng sơ đồ cú pháp cho các sản xuất EBNF

$\forall a \in V_T$, lập sơ đồ cú pháp



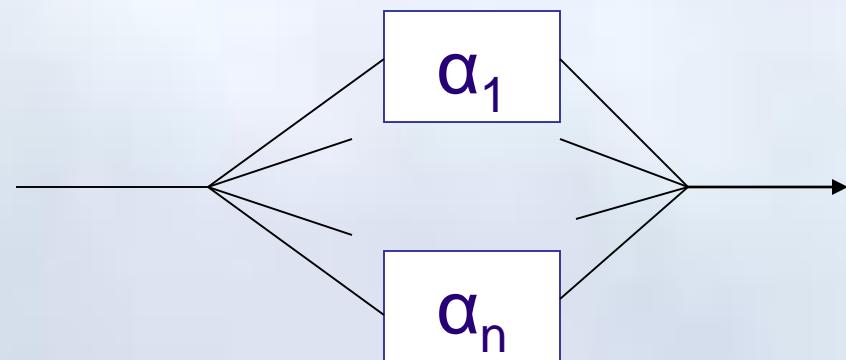
$\forall A \in V_N$, lập sơ đồ cú pháp



\forall sản xuất dạng

$A \rightarrow a_1 | a_2 | \dots | a_n$

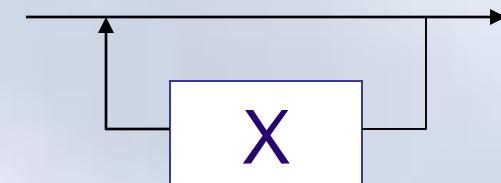
Lập sơ đồ có pháp



Nếu a có dạng $X_1 X_2 \dots X_N$, lập sơ đồ cú pháp

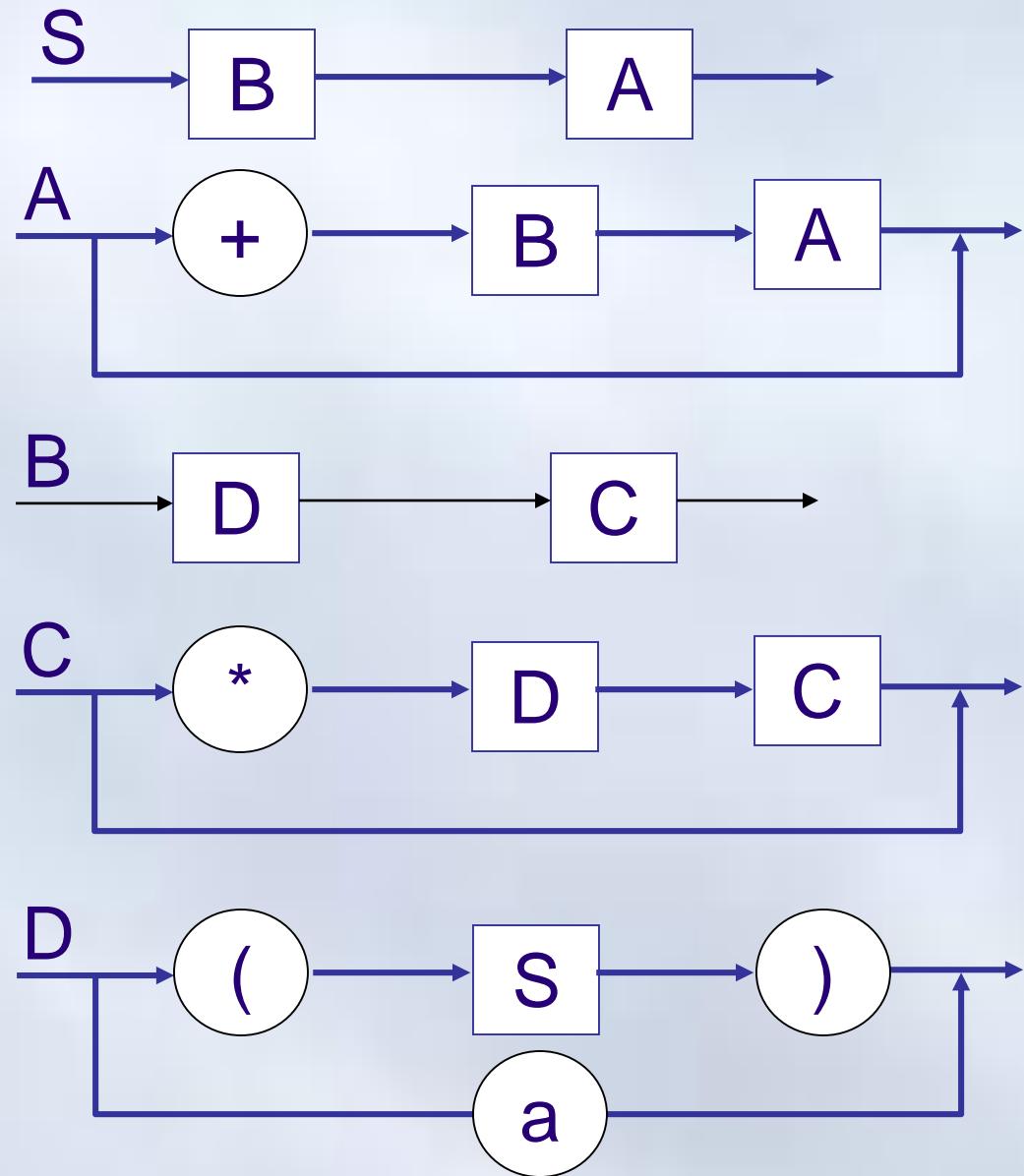


Nếu a có dạng lặp $a \in \{X\}$
Sơ đồ cú pháp có dạng



Xây dựng sơ đồ cú pháp → Ví dụ

$S \rightarrow BA$
 $A \rightarrow +BA|\varepsilon$
 $B \rightarrow DC$
 $C \rightarrow *DC|\varepsilon$
 $D \rightarrow (S)|a$

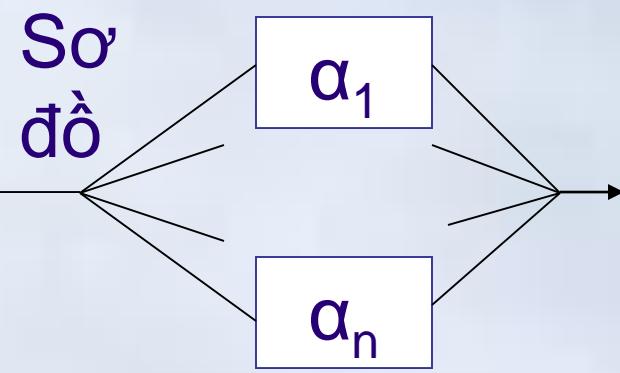


Xây dựng chương trình

- $T(S)$ là câu lệnh diễn dịch cho sơ đồ S
- Ch là ký hiệu đọc trước trên xâu vào bởi $nextCh$

Sơ đồ dạng: $\rightarrow [X_1] \rightarrow [X_2] \rightarrow \dots \rightarrow [X_N] \rightarrow$

Diễn dịch thành: $\{T(X_1); T(X_2); \dots; T(X_N); \}$



If Ch in FIRST(α_1) Then $T(\alpha_1)$
 Else If Ch in FIRST(α_1) Then $T(\alpha_2)$
 Else If...
 Else If Ch in FIRST(α_n) Then $T(\alpha_n)$

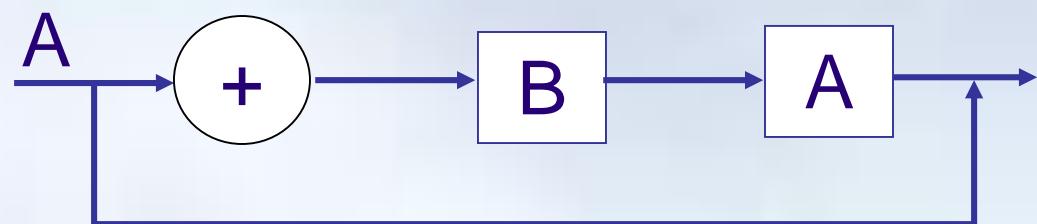
Sơ đồ cú pháp:

$\rightarrow [A] \rightarrow$ diễn dịch: Call A

Sơ đồ cú pháp

If Ch = a Then $nextCh$
 Else Error (Đang đợi ký hiệu a)

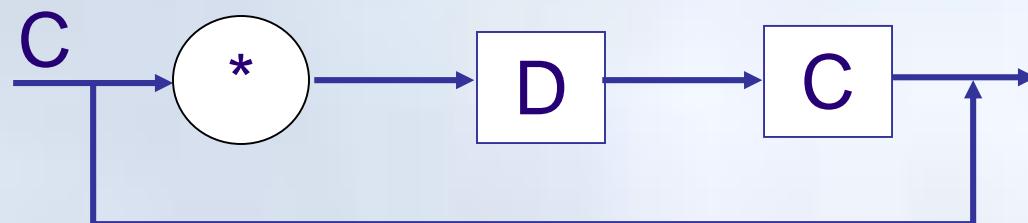
Xây dựng chương trình → Ví dụ



```
void S(){  
    B();  
    A();  
}
```

```
void A(){  
    if( Ch=='+' ) {  
        nextCh();  
        B();  
        A();  
    }  
}
```

Xây dựng chương trình → Ví dụ



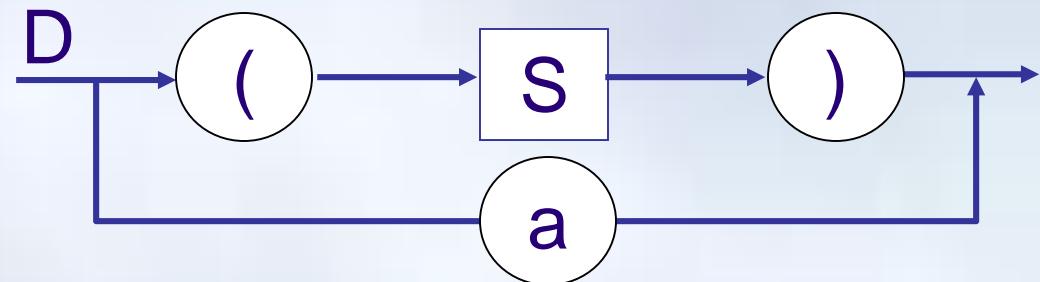
```
void C(){  
    if( Ch=='*') {  
        nextCh();  
        D();  
        C();  
    }  
}
```



```
void B(){  
    D();  
    C();  
}
```

Xây dựng chương trình → Ví dụ

```
void D(){  
    if(Ch=='('){  
        nextCh();  
        S();  
        if(Ch==')') nextCh();  
        else Error("Thieu dong ngoac");  
    } else if(Ch=='a') nextCh();  
    else Error("Thieu toan hang");  
}
```



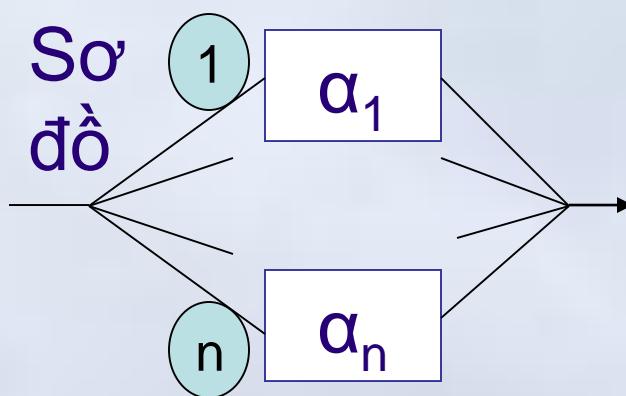
Xây dựng chương trình → Ví dụ

```
void main(){  
    gets(Str);    //nhập xâu cần phân tích  
    nextCh();    //đọc trước một ký hiệu  
    S();          //bắt đầu phân tích  
    printf("Bieu thuc hop le...\n");  
}
```

$(a+a)^*a$	→ Bieu thuc hop le
a^*a^+	→ Thieu toan hang
$a+^*a$	→ Thieu toan hang
$a^*(a+a)$	→ Thieu dong ngoac
$a+aa$	→ Bieu thuc hop le

Tìm danh sách sản xuất sử dụng

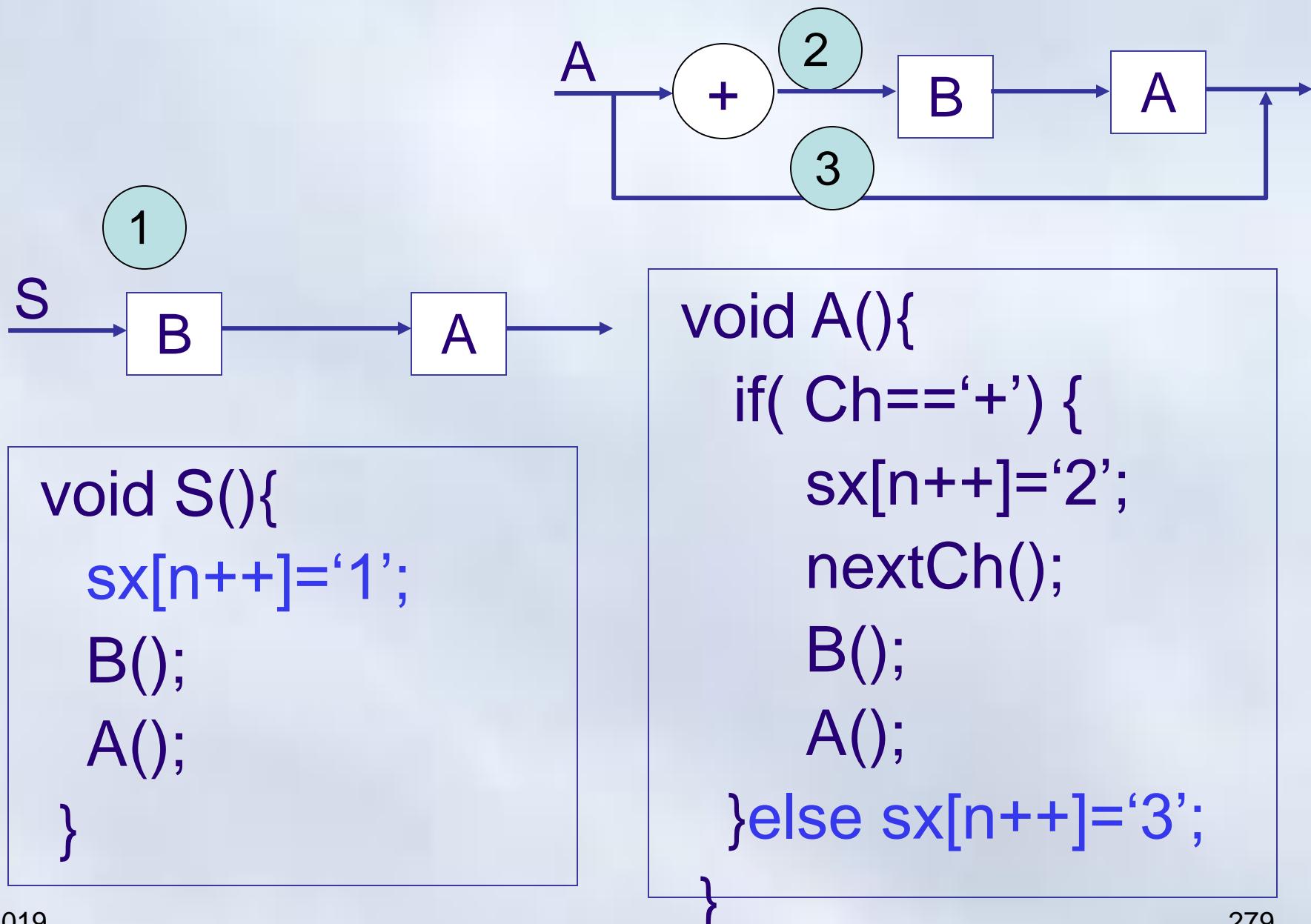
- Các nhánh trên sơ đồ được đánh số
 - Nhánh sơ đồ ứng với số hiệu sản xuất
- Khi đi theo một nhánh, cần viết ra số hiệu nhánh trước khi thực hiện câu lệnh diễn giải



```

If Ch in FIRST( $\alpha_1$ ) Then
  Print("1"); T( $\alpha_1$ )
Else If...
Else If Ch in FIRST( $\alpha_n$ ) Then
  Print("n"); T( $\alpha_n$ )
  
```

Tìm danh sách sản xuất sử dụng → Ví dụ



Chương 3: Phân tích cú pháp

1. Giới thiệu
2. Phương pháp phân tích cú pháp quay lui
3. Phương pháp phân tích bảng
4. Phương pháp phân tích cú pháp tất định
5. Xây dựng bộ phân tích cú pháp cho PL/0

Nguyên tắc

- Văn phạm của PL/0 là LL(1)
 - Sử dụng phương pháp phân tích xem trước
 - Bảng phân tích lớn
 - Dùng phương pháp đệ quy trên xuống
- Bộ phân tích cú pháp
 - Gồm tập thủ tục, mỗi thủ tục ứng với một sơ đồ
 - Luôn đọc trước một ký hiệu/từ tố
 - Xem trước một từ tố sẽ cho phép chọn đúng đường đi khi gặp điểm rẽ nhánh trên sơ đồ cú pháp
 - Khi thoát khỏi thủ tục triển khai một đích, luôn có một từ tố đã được đọc dôi ra

Các thủ tục

- Token * getToken() //phân tích từ vựng
- void error (const char msg[]); //Báo lỗi
- void factor(void); //phân tích nhân tử
- void term(void); //phân tích số hạng
- void expression(void); // phân tích biểu thức
- void condition(void); // phân tích điều kiện
- void statement(void); // phân tích câu lệnh
- void block(void); // phân tích các khối câu lệnh
- void program(); //Phân tích chương trình

Term

```
void term()
```

```
{
```

```
    factor();
```

```
    while(Token == TIMES || Token == SLASH)
```

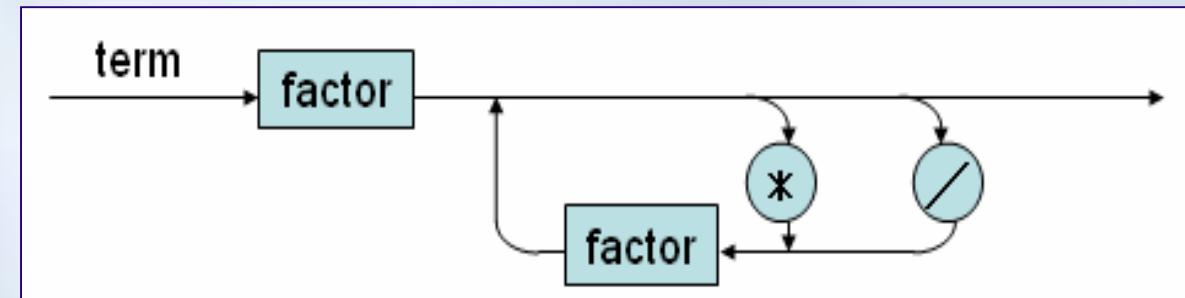
```
{
```

```
    Token = getToken();
```

```
    factor();
```

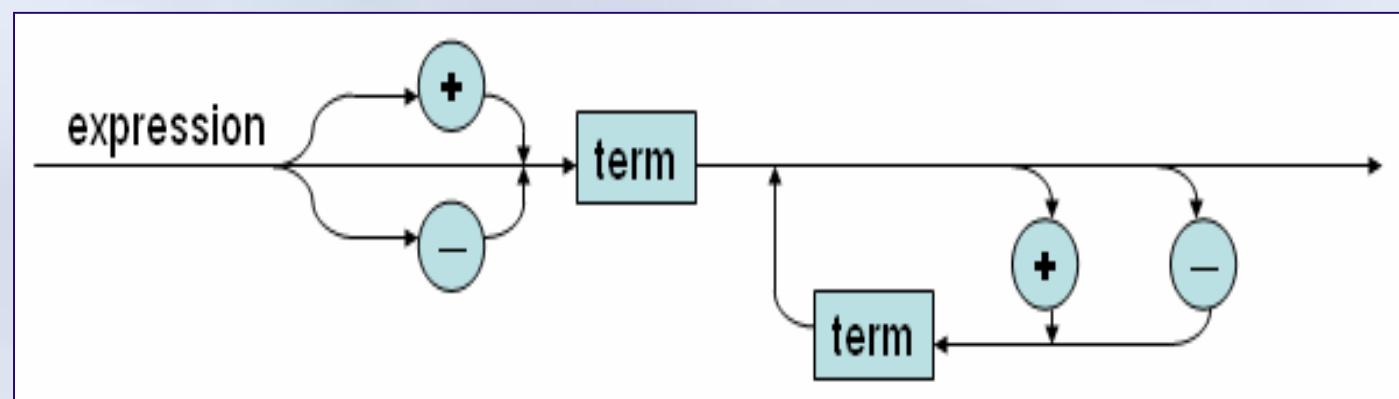
```
}
```

```
}
```



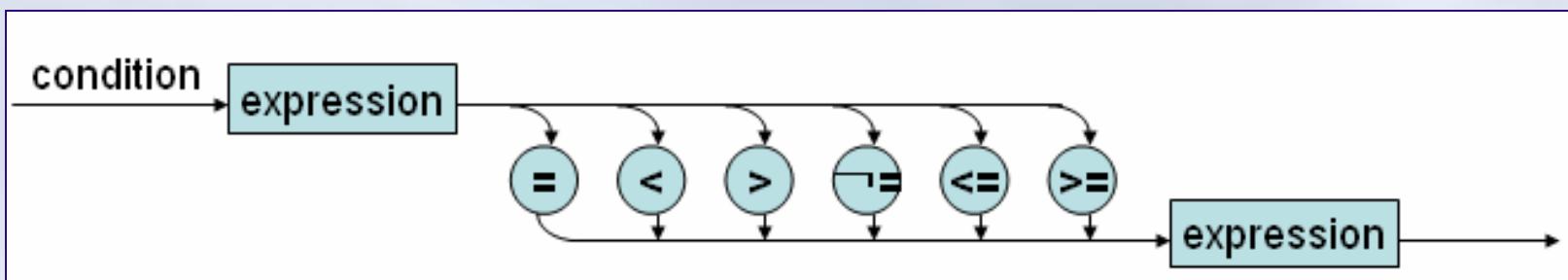
Expression

```
void expression() {  
    if(Token== plus || Token == minus)  
        Token = getToken();  
    term();  
    while(Token == plus || Token == minus){  
        Token = getToken();  
        term();  
    }  
}
```

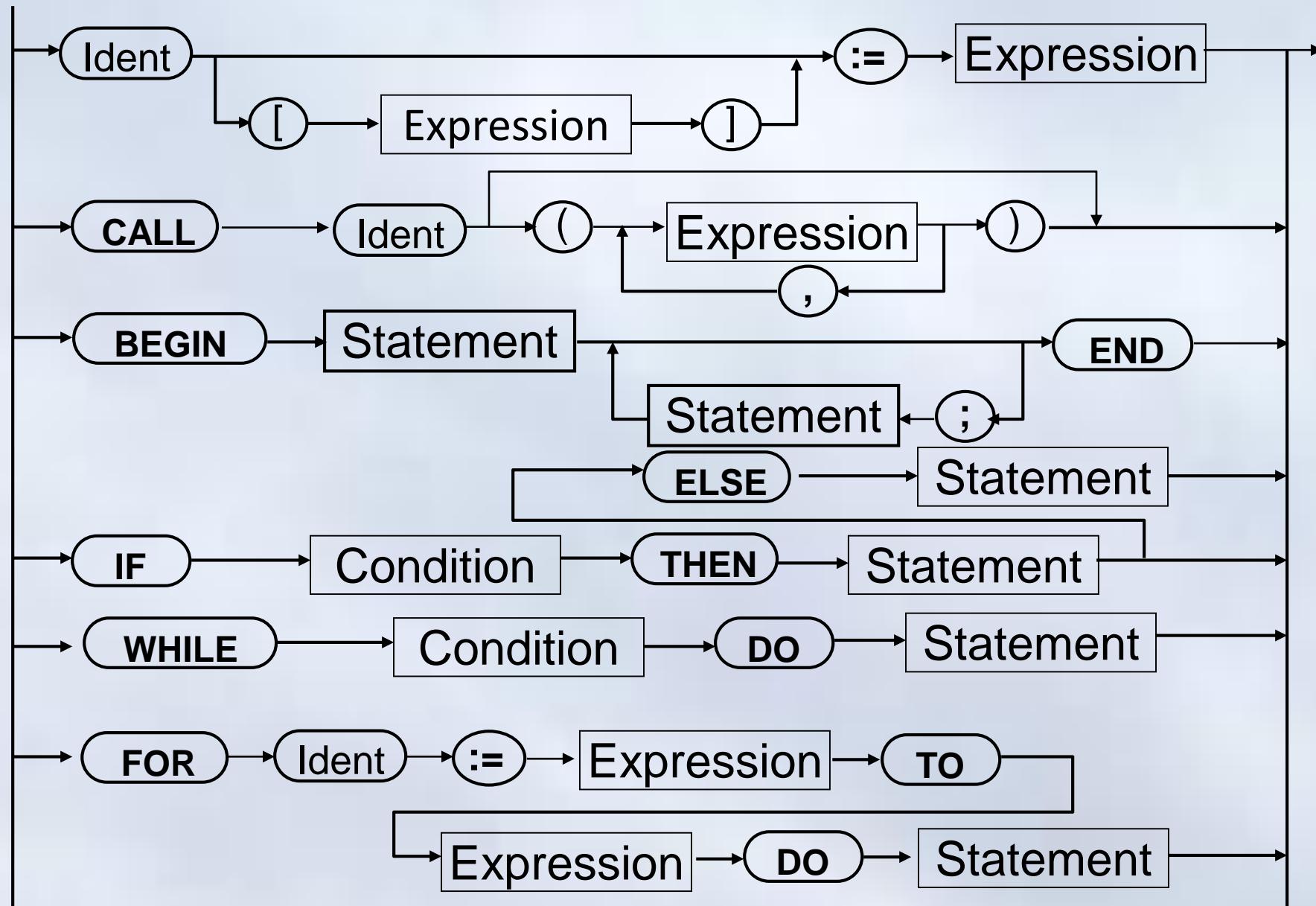


Condition

```
void condition() {  
    expression();  
    if(Token == eql || Token == neq || Token == lss ||  
        Token == leq || Token == grt || Token == geq){  
        Token = getToken();  
        expression();  
    }else {  
        error("condition: syntax error");  
    }  
}
```



Statement



Statement

```
void statement() {  
    if(Token==Ident){    //variable := <expression>  
        Token =getToken();    //array variable ?  
        if(Token==assign){  
            getToken();  
            Expresion();  
        }else Error("Thieu toan tu gan")  
    }  
    else.....  
}
```

Statement

```
if (Token == CALL){  
    Token= getToken();  
    if(Token == IDENT){  
        if(Token == LPAREN){  
            Token = getToken();  
            Expression();  
            while(Token==COMMA){  
                Token = getToken();  
                Expression();  
            }  
            if(Token = RPAREN) getToken()  
            else Error("Thiếu dấu đóng ngoặc");  
        }  
        }else Error("Thiếu tên thủ tục/hàm");  
    }else...
```

Statement

```
if (Token == BEGIN){  
    Token = getToken()  
    Statement();  
    while (Token == SEMICOLON) {  
        Token = getToken();  
        Statement();  
    }  
    if(Token==END) Token = getToken()  
    else Error("Thiếu từ khóa End");  
}else...
```

Program

```
void Program(){
    if(Token==PROGRAM){
        Token = getToken();
        if (Token==IDENT){
            getToken();
            if(Token==SEMICOLON){
                getToken();
                Block();
                if(Token==PERIOD)
                    printf("Thành công");
                else Error("Thiếu dấu .");
            }else Error("Thiếu dấu chấm phẩy");
        }else Error("Thiếu tên chương trình");
    }else Error("Thiếu từ khóa Program")
```

IT4079:NGÔN NGỮ và PHƯƠNG PHÁP DỊCH

Phạm Đăng Hải

haipd@soict.hust.edu.vn

Chương 4: Phân tích ngũ nghĩa

1. Giới thiệu
2. Bảng ký hiệu
3. Chương trình dịch định hướng cú pháp
4. Kiểm tra kiểu
5. Xử lý sai sót

Ví dụ 1

Cho văn phạm $G = (V_T, V_N, P, S)$

$P: \{ \begin{aligned} <\text{Câu}> &\rightarrow <\text{Chủ ngữ}> <\text{Vị ngữ}> \\ <\text{Chủ ngữ}> &\rightarrow <\text{Danh ngữ}> | <\text{Danh từ}> \\ <\text{Chủ ngữ}> &\rightarrow <\text{Danh ngữ}> | <\text{Danh từ}> \\ <\text{Danh ngữ}> &\rightarrow <\text{Danh từ}> <\text{Tính từ}> \\ <\text{Vị ngữ}> &\rightarrow <\text{Động từ}> | <\text{Động từ}> <\text{Bổ ngữ}> \\ <\text{Bổ ngữ}> &\rightarrow <\text{Danh ngữ}> \\ <\text{Danh từ}> &\rightarrow « \text{Bò} » | « \text{Cỏ} » | \\ <\text{Tính từ}> &\rightarrow « \text{Vàng} » | « \text{Non} » \\ <\text{Động từ}> &\rightarrow « \text{gặt} » \} \end{aligned}$

Ví dụ 1

$L(G) =$

- « Bò vàng găm cỏ non »
 - « Bò vàng găm cỏ vàng »
 - « Bò non găm cỏ non »
 - « Bò vàng găm bò non »
 - « Cỏ non găm bò vàng »
-

Các câu đều
đúng ngữ pháp,
nhưng không
phải câu nào
cũng đúng ngữ
nghĩa (có ý
nghĩa)

Ví dụ 2

```
Program Toto;  
Const N = 0;  
Begin  
    N :=10;  
End.
```

<Statement>

⇒ <Variable> := <Expression>

⇒ <VariableIdentifier> := <Expression>

⇒ N := <Expression>

⇒ N := <Term>

⇒ N := <Factor>

⇒ N := <Unsignedconstant>

⇒ N := <unsignedinteger>

⇒ N := 10

Hoàn toàn đúng cú pháp của KPL

Sử dụng sai ý nghĩa ban đầu (Hằng số)

Nhận xét

- Không phải mọi câu văn (**NNLT**: câu lệnh) đúng ngũ pháp (**NNLT**: cú pháp) đều có giá trị sử dụng (**NNLT**: thực hiện được)
- Bộ phân tích ngũ nghĩa nhằm mục đích kiểm tra tính đúng đắn về mặt ngũ nghĩa của câu văn (**NNLT**: câu lệnh)

Nhiệm vụ bộ phân tích ngũ nghĩa trong NNLT

- Quản lý thông tin về các định danh (tên)
 - Hằng, biến, kiểu tự định nghĩa, chương trình con
- Kiểm tra việc sử dụng các định danh
 - Phải được khai báo trước khi dùng
 - Phải được sử dụng đúng mục đích
 - Gán giá trị cho hằng, tính toán trên kiểu, thủ tục...
 - Đảm bảo tính nhất quán
 - Tên được khai báo chỉ một lần trong phạm vi
 - Các phần tử trong kiểu liệt kê (*enum*) là duy nhất

Bảng ký hiệu

Nhiệm vụ bộ phân tích ngũ nghĩa trong NNLT

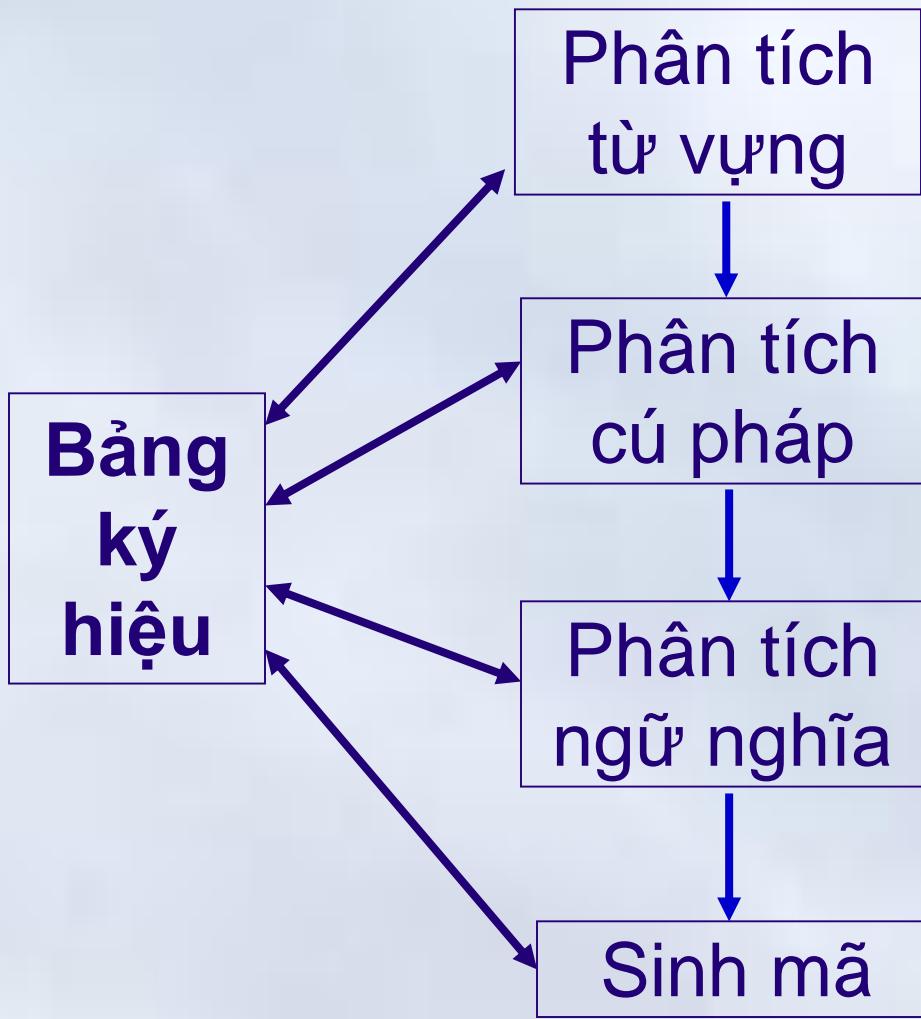
- Kiểm tra kiểu dữ liệu cho toán tử
 - Toán tử % của C đòi hỏi toán hạng kiểu nguyên
 - Có thể yêu cầu chuyển kiểu bắt buộc (*int2real*)
 - Chỉ số của mảng phải nguyên
- Kiểm tra sự tương ứng giữa tham số thực sự và hình thức
 - Số lượng tham số, tương ứng kiểu
- Kiểm tra kiểu trả về của hàm..

Các biểu thức kiểu của ngôn ngữ
Bộ luật để định kiểu cho các cấu trúc

Chương 4: Phân tích ngũ nghĩa

1. Giới thiệu
2. Bảng ký hiệu
3. Chương trình dịch định hướng cú pháp
4. Kiểm tra kiểu
5. Xử lý sai sót

Mục đích



- Bảng dữ liệu mà các pha của CTD đều s/dụng
- Dùng chứa thông tin về các danh biếu (tên) xuất hiện trong chương trình nguồn
- Mỗi phần tử ứng với một tên, gồm 2 trường
 - Trường tên
 - Khóa của bảng
 - Trường thuộc tính
 - Thuộc tính của tên
 - Biến (kiểu), hằng (giá trị)..

Mục đích

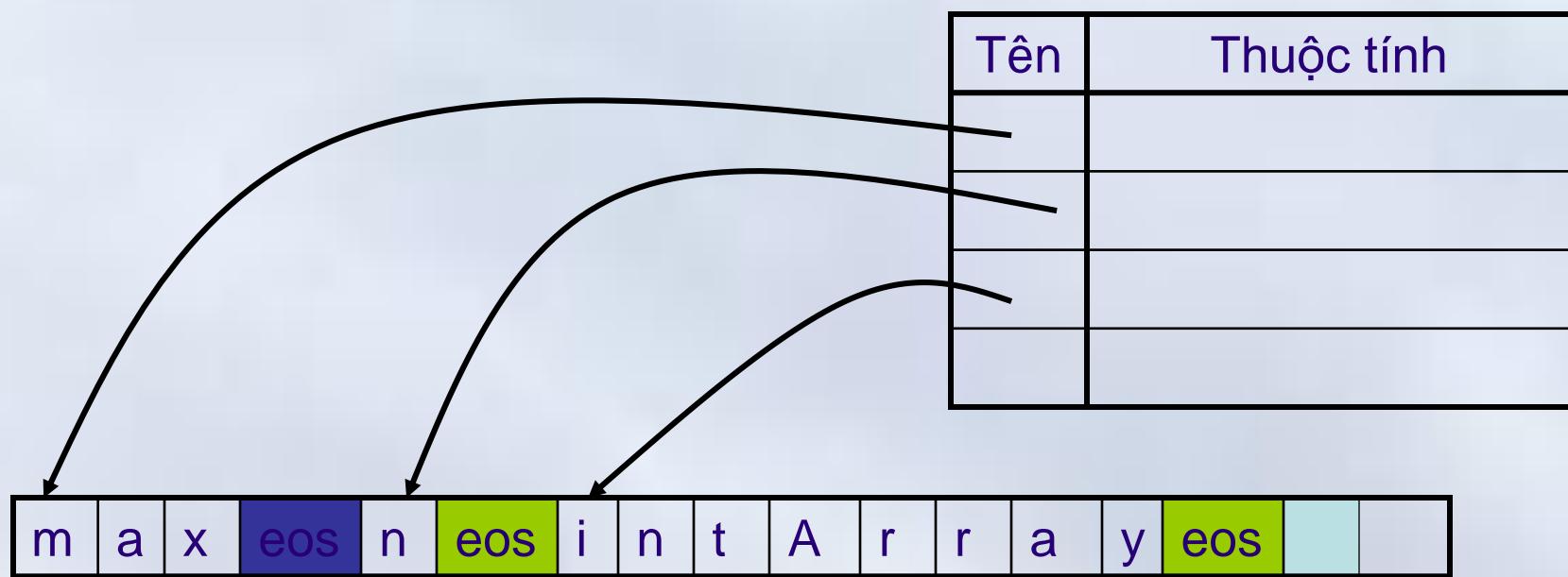
Khi gấp một danh biểu trong chương trình

- **Gấp trong giai đoạn khai báo**
 - Đưa tên và các thông tin tương ứng vào bảng
 - Ví dụ: **Const Max = 10;**
 - Đưa **Max** vào bảng, với kiểu là **constant**, giá trị là **10**;
- **Gấp trong câu lệnh**
 - Đọc thông tin ra để sử dụng
 - Phân tích ngữ nghĩa: Sử dụng đúng mục đích không?
 - Ví dụ: Max := 20; ← Sai mục đích
 - Sinh mã: Kích thước bộ nhớ cấp phát cho tên
 - Ví dụ: int → 2 bytes, float → 4 byte

Lưu trữ tên

Tên								Thuộc tính
m	a	x						
n								
i	n	t	A	r	r	a	y	

- Đơn giản
- Nhanh
- Độ dài tên bị giới hạn
- Lãng phí nhớ ($\approx 20\%$)



Hiệu quả hơn

Các yêu cầu phải có của bảng ký hiệu

- Phát hiện một tên cho trước có trong bảng hay không
- Thêm một tên vào bảng
- Lấy thuộc tính tương ứng với một tên
- Thêm các thông tin mới vào một tên
- Xóa một tên, nhóm tên ra khỏi bảng

Các cấu trúc dữ liệu cho bảng ký hiệu

Nhiều cách tổ chức bảng ký hiệu khác nhau

- Danh sách tuyến tính
 - Đơn giản, chậm
- Bảng băm
 - Nhanh, phức tạp
- Cây nhị phân tìm kiếm
 - Mức độ phức tạp và tốc độ vừa phải

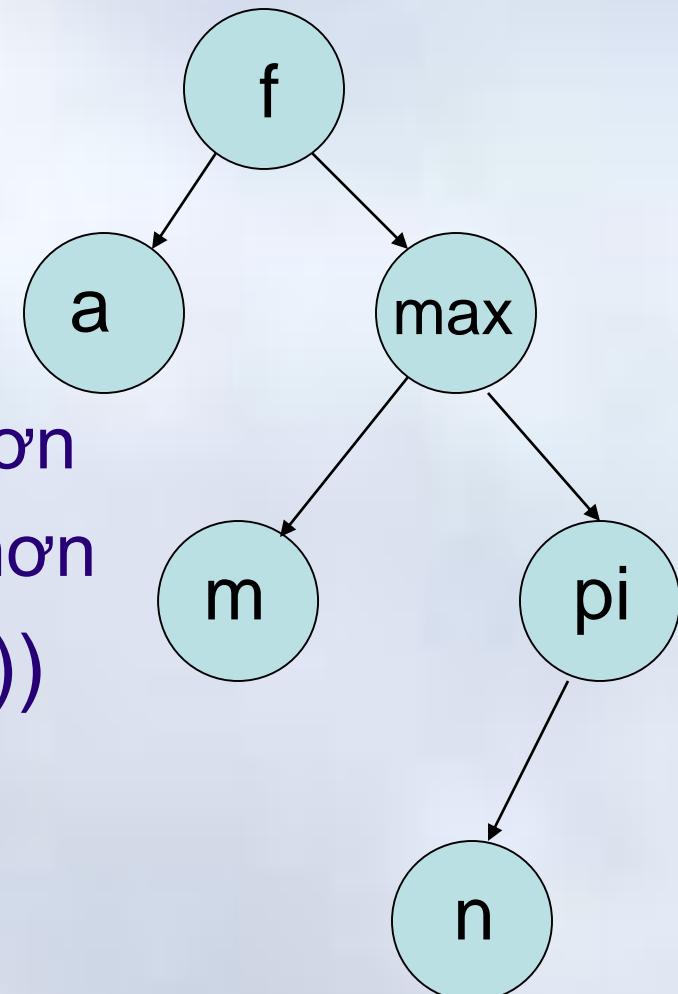
Bảng ký hiệu ảnh hưởng tới chất lượng của chương trình dịch vì CTD làm việc liên tục với bảng ký hiệu

Các CSDL cho bảng ký hiệu → Danh sách

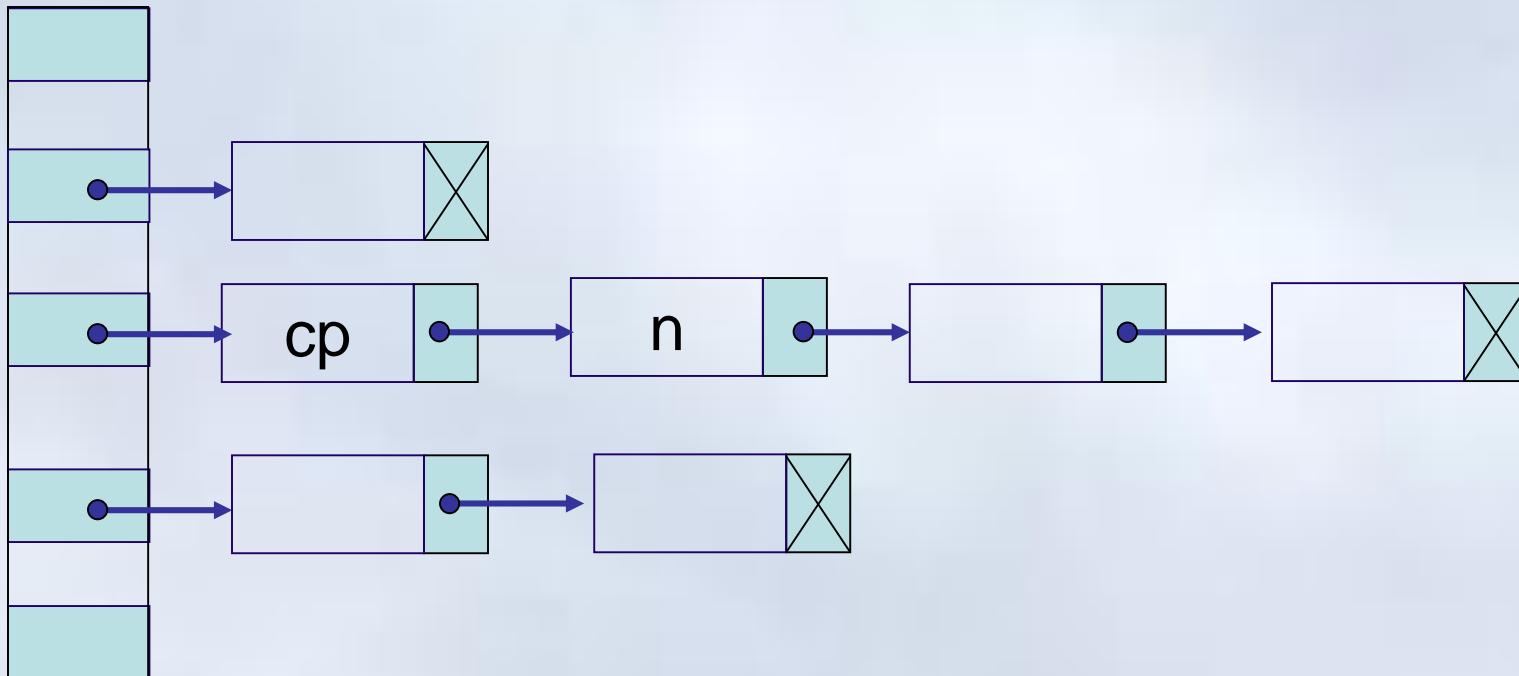
- Dùng một (*nhiều*) mảng lưu các tên trong c/trình
 - Kích thước cố định (phải lớn → lãng phí)
 - Giải quyết: danh sách liên kết (hai) chiều
 - Tiết kiệm bộ nhớ
- Danh sách không sắp xếp
 - Thêm tên vào bảng theo thứ tự gấp trong c/ trình nguồn
 - Tốc độ tìm kiếm chậm (*Độ phức tạp: $O(n)$*)
 - Thường dùng với các ngôn ngữ nhỏ, có ít danh biểu
- Danh sách sắp xếp
 - Phức tạp khi thêm tên vào bảng
 - Tốc độ tìm kiếm: $O(\log(n))$
 - Dùng khi tập danh biểu xác định (*VD tập từ khóa*)

Các CSDL cho bảng ký hiệu → Cây tìm kiếm

- Các nút của cây nhị phân có
 - Khóa là tên của bản ghi
 - 2 con trỏ Left và Right
- Mọi nút phải thỏa mãn
 - Khóa của con trái phải nhỏ hơn
 - Khóa của con phải phải lớn hơn
- Thời gian tìm kiếm $O(\log_2(n))$
 - Gốc rỗng → không tồn tại
 - Trùng khóa gốc → thỏa mãn
 - Nhỏ hơn → Tìm tại con trái
 - Lớn hơn → Tìm tại con phải



Các CSDL cho bảng ký hiệu → Bảng băm



Bảng là danh sách N phần tử

- Số phần tử cố định
 - Mỗi phần tử chứa tên và các thuộc tính của tên
 - Có thể là con trỏ, trỏ tới danh sách liên kết
- Đòi hỏi một hàm băm

Các CSDL cho bảng ký hiệu → Bảng băm

- H là bảng băm gồm N phần tử
 - h : là hàm băm
 - α : là một tên
- $h(\alpha) \in [0..N-1]$

Hoạt động

Giả thiết $\beta = H[h(\alpha)]$

- $\beta = \alpha \rightarrow$ Trùng tên: Tên đã khai báo
- $\beta = \varepsilon \rightarrow$ Tên chưa khai báo
- $\beta \neq \varepsilon \text{ & } \beta \neq \alpha \rightarrow$ Xung đột. Một ô chứa 2 tên
 - Đi theo DSLK tương ứng để ra được α

Bảng ký hiệu cho cấu trúc khối

- Ngôn ngữ là có cấu trúc khối nếu
 - Một khối có thể được nằm trong khối khác
 - Ví dụ: Trong *Procedure* có *Procedure* khác...
 - Phạm vi của khai báo trong mỗi khối là chính khối đó và các khối nằm trong nó
- Luật lồng nhau gần nhất
 - Cho phép nhiều khai báo của cùng một tên
 - Các tên phải nằm trong các khối khác nhau
 - Khai báo có hiệu lực thuộc khối gần nhất
 - Khi một thủ tục nằm trong thủ tục khác được gọi, các khai báo của thủ tục bên ngoài tạm dừng hoạt động

Luật về phạm vi sử dụng

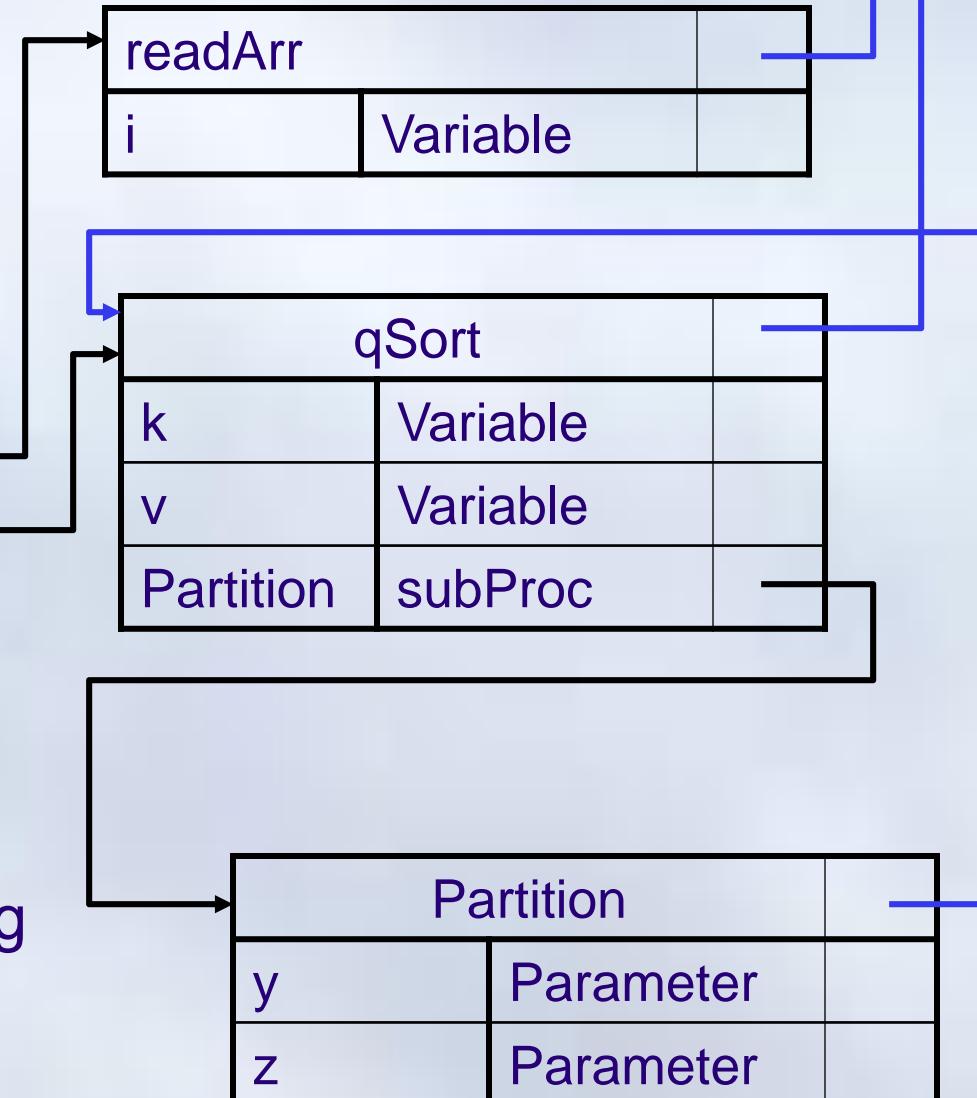
- Toán tử thêm (**Insert**) vào bảng ký hiệu không được ghi đè những khai báo trước
- Toán tử tìm kiếm (**lookup**) trong bảng ký hiệu luôn luôn tham chiếu luật phạm vi gần nhất
- Toán tử xóa (**delete**) chỉ được xóa khai báo gần nhất
- Giải pháp
 - Dùng stack để lưu trữ dấu vết của các thủ tục lồng nhau
 - Tạo bảng ký hiệu mới cho mỗi thủ tục
 - Thêm một định danh mới, cần chỉ rõ bảng ký hiệu cần thêm

Phạm vi → Ví dụ

```
1) Program sort; //Chương trình Quicksort
2) Const Max = 100;
3) Var Arr: array[0..Max] of integer;
4) Procedure readarray;
5)     Var i: integer;
6)     Begin ..... end {readarray};
7) Procedure exchange(i, j: integer);
8)     Begin {exchange} end;
9) Procedure quicksort(m, n: integer);
10)    Var k, v: integer;
11)    Function partition(y,z: integer): integer;
12)    Begin .....exchange(i,j) end; {partition}
13)    Begin ... end; {quicksort}
14) Begin ... end; {sort}
```

Dùng nhiều bảng ký hiệu

Program Sort	Null	
Tên	Thuộc tính	Ptr
Max	Constant	
Arr	Variable	
readArr	subProc	
qSort	subProc	



- Khi đoán nhận một chương trình con mới (*gọi tới `compileBlock()` đệ quy*), cần tạo một bảng ký hiệu cho chương trình con tương ứng
- Khi kết thúc (*ra khỏi thủ tục `compileblock()`*), bảng ký hiệu sẽ bị xóa

Dùng Stack

j	Variable
i	Variable
Partition	subProc
v	Variable
k	Variable
qSort	subProc
readArr	subProc
Max	constant
Arr	Variable
Đây Stack	

Vị trí thêm vào



Chiều tìm kiếm

Khi gặp danh biếu
trong khai báo đưa
vào đỉnh của stack

Gặp danh biếu
trong câu lệnh, tìm
từ đỉnh trở xuống

Ví dụ đơn giản

- Dùng stack làm bảng ký hiệu
- Dùng giá trị **Tx** cho biết vị trí của đỉnh stack
 - Tx là **tham trị** của `compileBlock()//compileBlock(int Tx)`
 - Giá trị của Tx không thay đổi trong `compileBlock()`
 - Khi phân tích Block, thủ tục `compileBlock()` có thể gọi đến chính nó, Tx sẽ được kế thừa
 - Khi đoán nhận thủ tục con, Tx sẽ tăng dần lên khi gấp các danh biểu của chương trình con
 - Khi đoán nhận xong - thoát ra khỏi `compileBlock()`, Tx khôi phục lại giá trị cũ (*giá trị trước khi gọi*)
 - Nhưng khai báo bên trong một khối sẽ bị xóa đi
 - Ban đầu, danh sách rỗng → `compileBlock(0)`

Ví dụ đơn giản

Const M=10;

Var X, Y: integer;

Procedure A;

Var X, Z: Integer;

Procedure AA;

Var X: integer;

Begin

$Z = X + Y * M$

End;

Begin

End;

Procedure B;

Var X, Y;

Begin

End;

Begin

End.

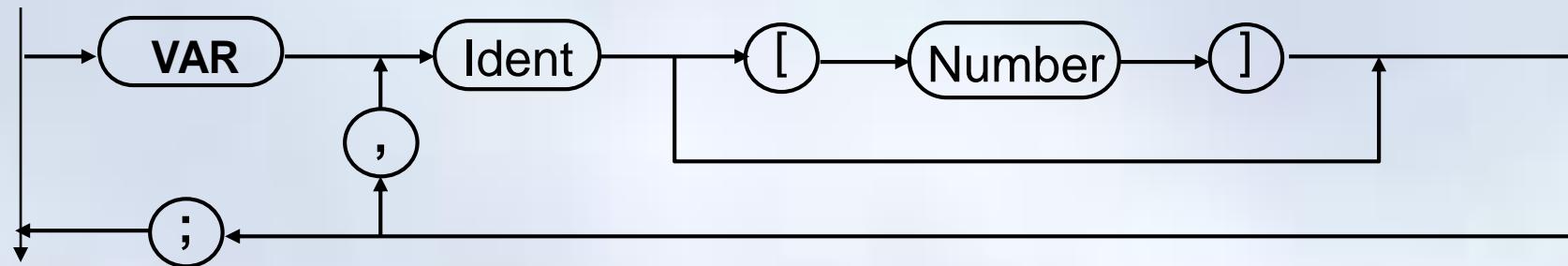
X	Variable	
AA	subProc	←
Z	Variable	
X	Variable	
A	subProc	←
Y	constant	←
X	constant	
M	Constant	
Đáy Stack		←

Ví dụ đơn giản

Cần các thủ tục

- **Procedure Enter(char * Id, Object Type);**
 - Đưa một danh biểu vào bảng ký hiệu
 - Giá trị của danh biểu
 - Kiểu danh biểu (constant, type, variable, procedure, function)
- **Function Location(char * Id) : int;**
 - Chỉ ra vị trí của danh biểu trong bảng ký hiệu. Nếu không thấy, trả về giá trị 0
- **Function checkIdent(char * Id) : Boolean;**
 - Kiểm tra tên (Id) đã được khai báo trong phạm vi chưa
- **Function getKind(int idx) : Object**
 - Trả về kiểu của danh biểu tại vị trí idx

Ví dụ đơn giản → Khai báo



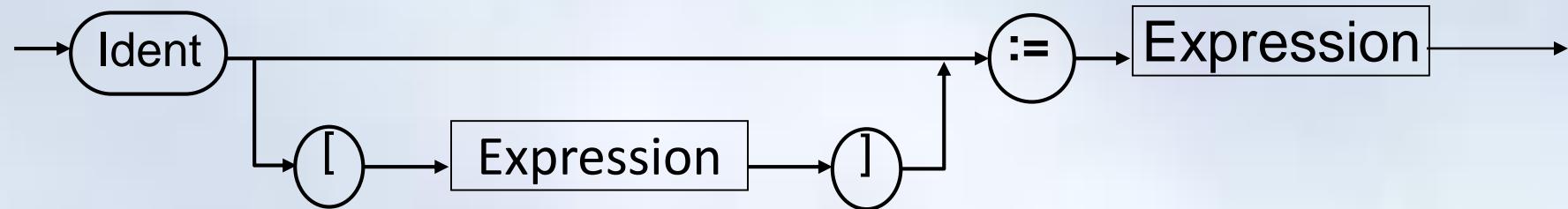
```
void compileBlock( ){
```

```
....  
if(Token==VAR){  
    Token = getToken();  
    compileDeclareVariable ();  
    while(Token == COMMA) {  
        Token = getToken();  
        compileDeclareVariable ();  
    }  
}  
....
```

Ví dụ đơn giản → Khai báo

```
void compileDeclareVariable(void){  
    if(Token==IDENT){  
        if(CheckIdent(Id) == 0) Enter(Id,Variable);  
        Else Error();//Trung ten  
        Token = getToken();  
        if(Token == LBRACK){  
            .....  
        }  
        }else Error(19);//Thieu ten bien  
    }  
}
```

Ví dụ đơn giản → Sử dụng



```

void compileStatement(){
    int p;
    if(Token == IDENT){
        p = Location(Id);
        if(p < 0) Error(52); //Ten chua khai bao
        if(getKind(p) != Variable)
            Error( );//Ve trai phai la mot bien
        Token = getToken();
        if(Token == LBRACK){
            .....
        }
    }
}
  
```

Chương 4: Phân tích ngũ nghĩa

1. Giới thiệu
2. Bảng ký hiệu
3. Chương trình dịch định hướng cú pháp
4. Kiểm tra kiểu
5. Xử lý sai sót

Định nghĩa hướng cú pháp (Syntax directed definition)

- Là dạng tổng quát của VPPNC
- Mỗi ký hiệu VP liên kết với một tập thuộc tính
 - Hai loại thuộc tính: Tổng hợp và kế thừa
 - Thuộc tính là khái niệm trừu tượng, biểu diễn một đại lượng bất kỳ: số, xâu, vị trí trong bộ nhớ..
- **Thuộc tính tổng hợp**
 - Giá trị của thuộc tính tại một nút trong cây được xác định từ giá trị của các nút con của nó.
- **Thuộc tính kế thừa**
 - Giá trị của thuộc tính được định nghĩa dựa vào giá trị nút cha và/hoặc các nút anh em của nó.

Dạng của định nghĩa hướng cú pháp

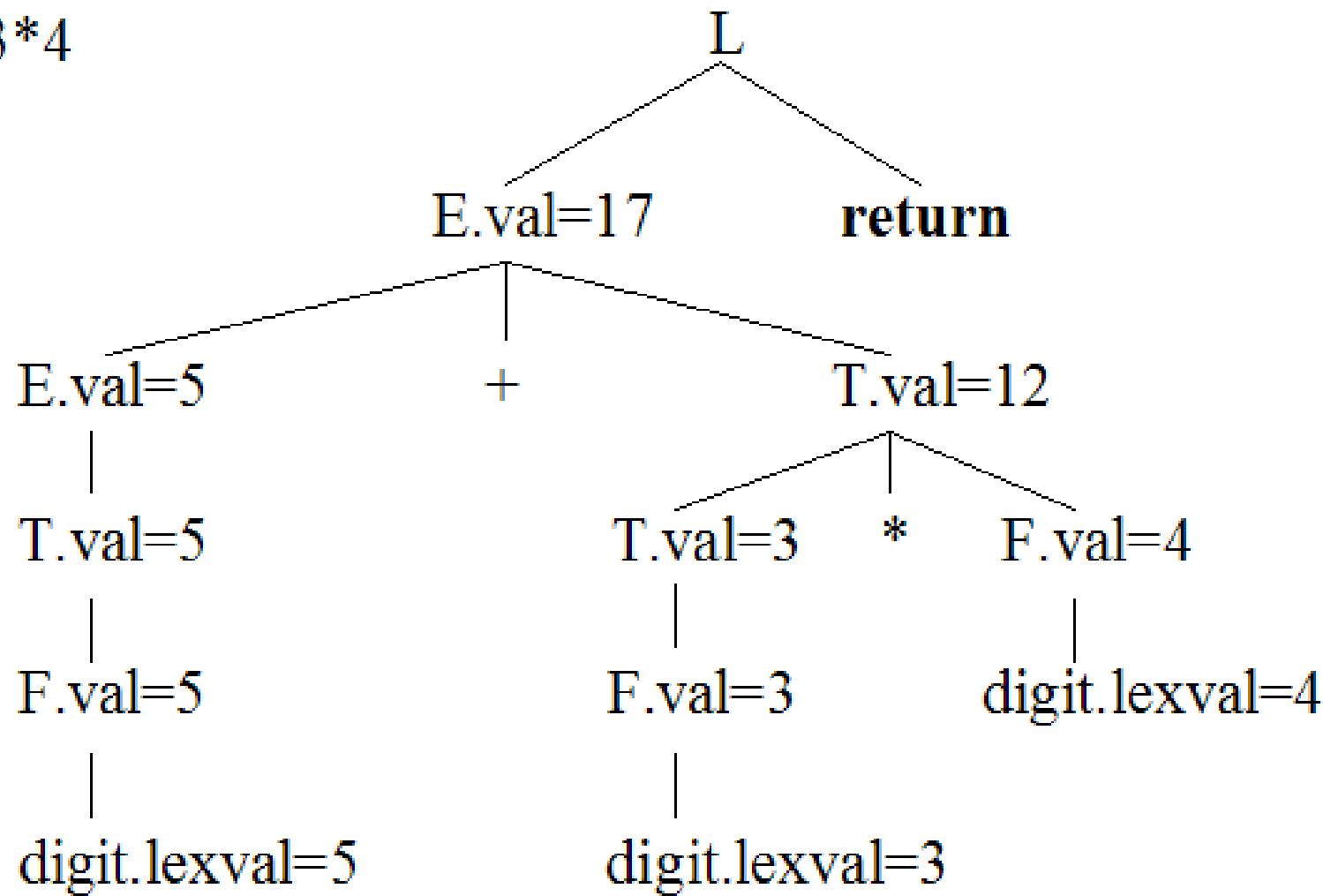
- Mỗi sản xuất dạng $A \rightarrow \alpha$ liên hệ với một tập luật ngữ nghĩa có dạng $b = f(c_1, c_2, \dots, c_n)$ trong đó f là một hàm và b thoả một trong hai yêu cầu sau:
 - b là **thuộc tính tổng hợp** của A ; c_1, \dots, c_n là các thuộc tính của các ký hiệu trong α phải sản xuất $A \rightarrow \alpha$
 - b là **thuộc tính kế thừa** của một trong những ký hiệu trong α ; c_1, \dots, c_n là thuộc tính của các ký hiệu trong sản xuất $A \rightarrow \alpha$
- Tập luật ngữ nghĩa dùng để tính giá trị thuộc tính của ký hiệu A

Định nghĩa hướng cú pháp → Ví dụ

Sản xuất	Quy tắc ngữ nghĩa
$L \rightarrow E \text{ return}$	$\text{Print}(E.\text{val})$
$E \rightarrow E_1 + T$	$E.\text{val} = E_1.\text{val} + T.\text{val}$
$E \rightarrow T$	$E.\text{val} = T.\text{val}$
$T \rightarrow T_1 * F$	$T.\text{val} = T_1.\text{val} * F.\text{val}$
$T \rightarrow F$	$T.\text{val} = F.\text{val}$
$F \rightarrow (E)$	$F.\text{val} = E.\text{val}$
$F \rightarrow \text{digit}$	$F.\text{val} = \text{digit}.\text{lexval}$
<ul style="list-style-type: none"> Các ký hiệu E, T, F có thuộc tính tổng hợp val Từ tố digit có thuộc tính tổng hợp lexval (Được bộ phân tích từ vựng đưa ra) 	

Cây phân tích cú pháp có chú giải

Cây có chỉ ra giá trị các thuộc tính tại mỗi nút

Input: $5+3*4$ 

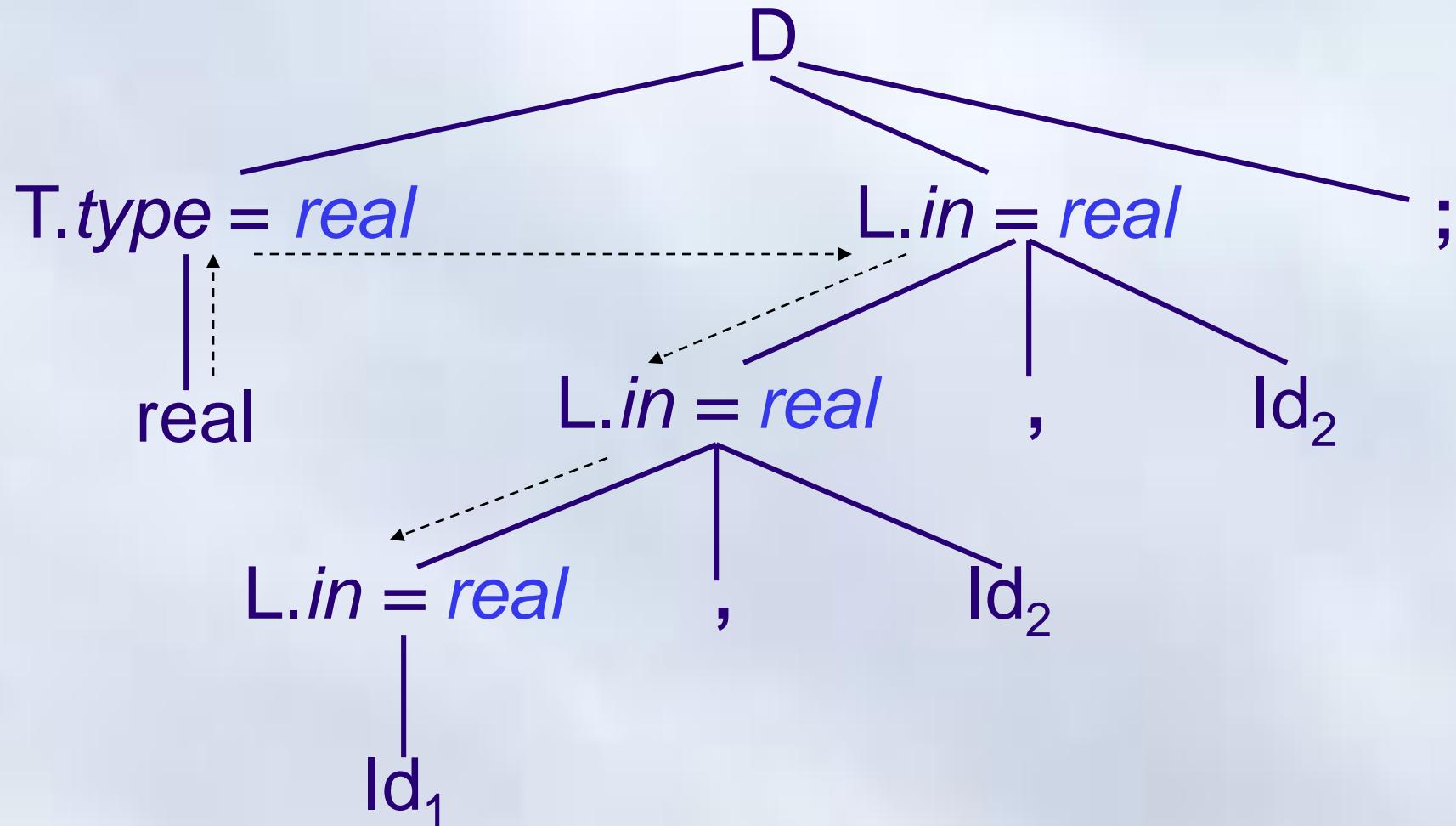
Thuộc tính kế thừa → Ví dụ

Sản xuất	Quy tắc ngữ nghĩa
$D \rightarrow T \ L ;$	$L.in := T.type$
$T \rightarrow int$	$T.type = integer$
$T \rightarrow float$	$T.type := real$
$L \rightarrow L_1 , \text{Id} \ // L_1 \equiv L$	$L_1.in := L.in$ $\text{addType}(\text{Id.entry}, L.in)$
$L \rightarrow \text{Id}$	$\text{addType}(\text{Id.entry}, L.in)$

- Ký hiệu không kết thúc **T** có thuộc tính tổng hợp *type* có giá trị được xác định bởi từ khóa *int/float* trong khai báo
- Luật ngữ nghĩa $L.in := T.type$ gắn với sản xuất $D \rightarrow T \ L ;$ ấn định thuộc tính kế thừa của L ($L.in$) tới kiểu khai báo
- Luật ngữ nghĩa gắn với sản xuất từ L , sẽ truyền kiểu của L với các khai báo tiếp và dùng thủ tục $\text{addType}()$ để lưu thuộc tính **kiểu** của danh biểu vào bảng ký hiệu tại vị trí Id.entry

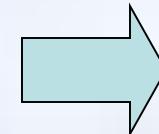
Cây phân tích cú pháp có chú giải

Ví dụ với khai báo: float x, y, z ;

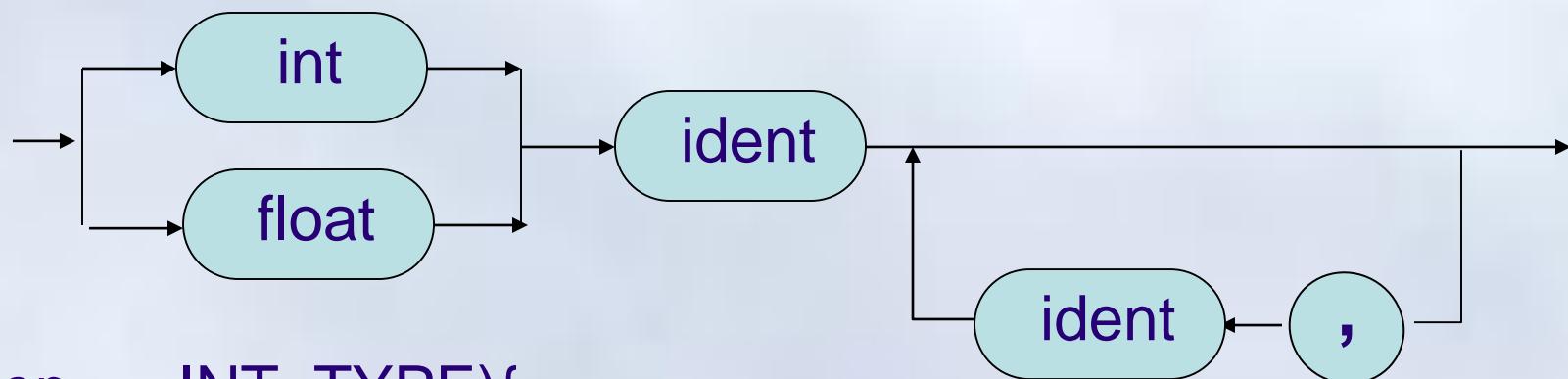


Thuộc tính kế thừa → Ví dụ

- $D \rightarrow T \ L$
- $T \rightarrow \text{int} \ |\text{float}$
- $L \rightarrow L \ , \ \text{Ident} \ |\text{Ident}$



- $D \rightarrow T \ L$
- $T \rightarrow \text{int} \ |\text{float}$
- $L \rightarrow \text{Ident} \ , \ \text{Ident}$



```

if(Token == INT_TYPE){
    Type = INT_TYPE;
    Token= getToken();
    if (Token = IDENT)
        if(CheckIdent(Id) == 0) Enter(Id,Variable,INT_TYPE);
}
  
```

Lưu trữ thông tin về phạm vi

- Văn phạm cho phép các chương trình con bao nhau
 - Khi bắt đầu phân tích chương trình con, phần khai báo của chương trình bao tạm dừng
 - Dùng một bảng ký hiệu riêng cho mỗi chương trình con
- Văn phạm của khai báo này:
$$P \rightarrow D ; S \quad // D: \text{Description}, S: \text{Statement}$$
$$D \rightarrow D; D \mid id : T \mid \text{proc id} ; D_1 ; S \quad // D_1 \equiv D$$
- Khi khai báo chương trình con $D \rightarrow \text{proc id} D_1; S$ được phân tích thì các khai báo trong D_1 được lưu trong bảng ký hiệu mới.

Lưu trữ thông tin về phạm vi → Ví dụ

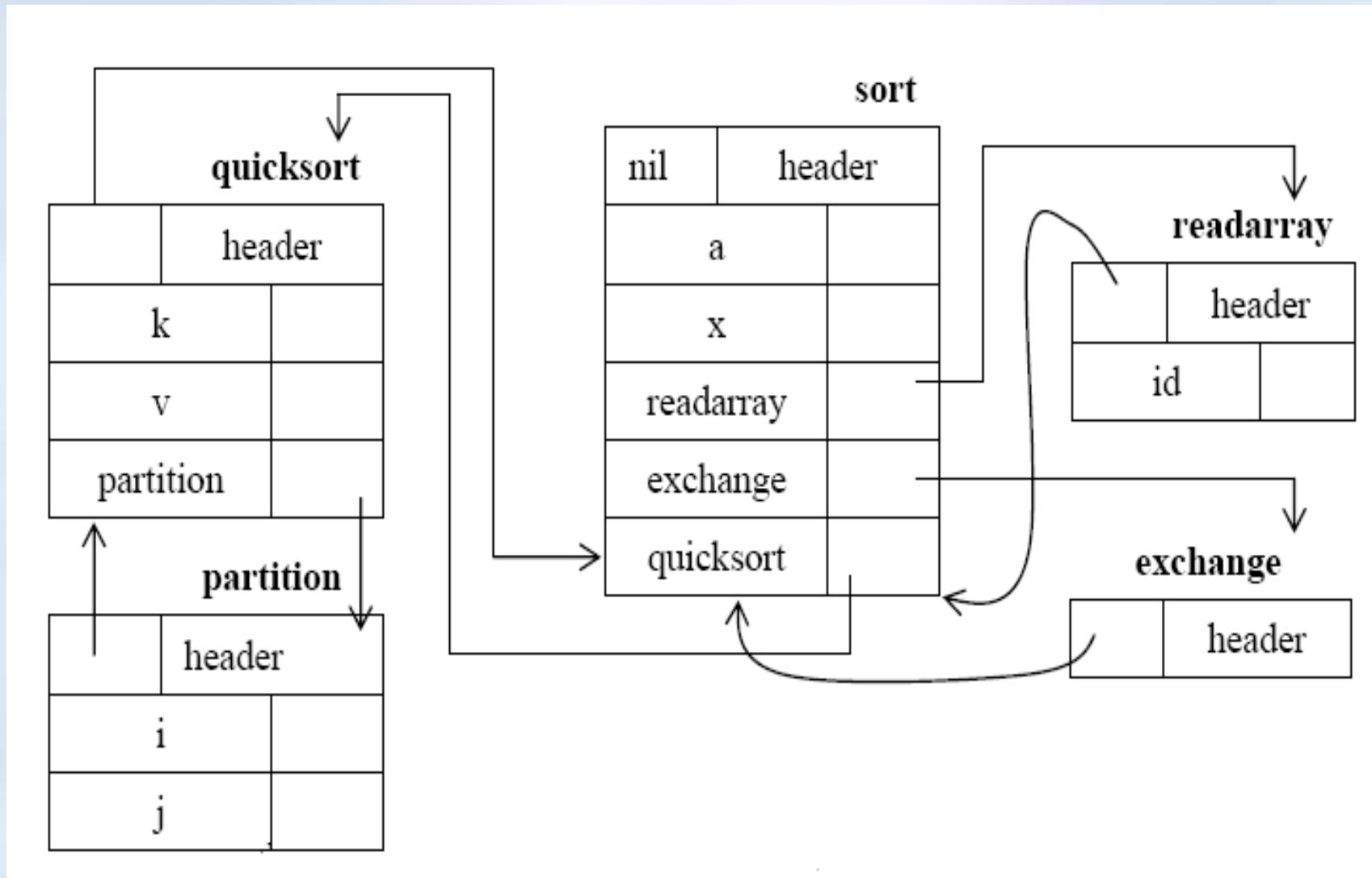
```

1) Program sort; //Chương trình Quicksort
2) Var a: array[0..10] of integer;
3)      x: integer;
4) Procedure readarray;
5)      Var i: integer;
6)      Begin ..... end {readarray};
7) Procedure exchange(i, j: integer);
8)      Begin {exchange} end;
9) Procedure quicksort(m, n: integer);
10)     Var k, v: integer;
11)     Function partition(y,z: integer): integer;
12)     Var l, j : integer;
13)     Begin .....exchange(i,j) end; {partition}
14)     Begin ... end; {quicksort}
15)Begin ... end; {sort}

```

Lưu trữ thông tin về phạm vi → Ví dụ

- Các bảng ký hiệu của chương trình sort



Quy tắc ngũ nghĩa → Các thao tác

- **mktable(previous)** – tạo một bảng kí hiệu mới, bảng này có *previous* chỉ đến bảng cha của nó.
- **enter(table, name, type, offset)** – thêm một đối tượng mới có tên *name* vào bảng kí hiệu được chỉ ra bởi *table* và đặt kiểu là *type* và địa chỉ tương đối là *offset* vào các trường tương ứng.
- **enterproc(table, name, newbtable)** – tạo một phần tử mới trong bảng *table* cho chương trình con *name*, *newbtable* trả tới bảng kí hiệu của CTC này.
- **addwidth(table, width)** – ghi tổng kích thước của tất cả các p/tử trong bảng kí hiệu vào header của bảng.

Khai báo trong chương trình con

Sản xuất	Quy tắc ngữ nghĩa
$P \rightarrow MD$	$addwidth(top(tblptr), top(offset)); pop(tblptr);$ $pop(offset)$
$M \rightarrow \epsilon$	$t := mkttable(null);$ $push(t, tblptr); push(0, offset)$
$D \rightarrow D ; D$	
$D \rightarrow \text{proc } id;$ $ND_1; S$	$t := top(tblptr); addwidth(t, top(offset));$ $pop(tblptr); pop(offset);$ $enterproc(top(tblptr), id.name, t)$
$N \rightarrow \epsilon$	$t := mkttable(top(tblptr));$ $push(t, tblptr); push(0, offset);$
$D \rightarrow id : T$	$enter(top(tblptr), id.name, T.type, top(offset));$ $top(offset) := top(offset) + T.width$

Tblptr: là Stack dùng chứa các con trỏ trả về tới bảng ký hiệu

Offset: Là Stack dùng lưu trữ các Offset

Xử lý các khai báo trong chương trình con

- Sản xuất: **P→MD** :
 - Hoạt động của cây con M được thực hiện trước
- Sản xuất: **M → ε**:
 - Tạo bảng ký hiệu cho phạm vi ngoài cùng (*chương trình sort*) bằng lệnh **mktable(nil)** //Không có SymTab cha
 - Khởi tạo stack **tblptr** với bảng ký hiệu vừa tạo ra
 - Đặt offset = 0.
- Sản xuất: **N → ε**:
 - Tạo ra một bảng mới **mktable(top(tblptr))**
 - Tham số **top(tblptr)** cho giá trị con trả tới bảng cha
 - Thêm bảng mới vào đỉnh stack **tblptr** //push(t,tblptr)
 - 0 được đẩy vào stack **offset** //push(0,Offset)

N đóng vai trò tương tự M khi một khai báo CTC xuất hiện

Xử lý các khai báo trong chương trình con

- Với mỗi khai báo **id: T**
 - một phần tử mới được tạo ra cho id trong bảng kí hiệu hiện hành (*top(tblptr)*)
 - Stack *tblptr* không đổi,
 - Giá trị **top(offset)** được tăng lên bởi *T.width*.
- Khi **D → proc id ; N D₁ ; S** diễn ra
 - Kích thước của tất cả các đối tượng dữ liệu khai báo trong *D₁* sẽ nằm trên đỉnh stack offset.
 - Kích thước này được lưu trữ bằng cách dùng *Addwidth()*,
 - Các stack *tblptr* và offset bị lấy mất phần tử trên cùng (*pop()*)
 - Thao tác thực hiện trên các khai báo của chương trình con.

Ví dụ

X:int;

Y:int

Proc A;

X:int

N : int;;

Proc B;

X:int;

S_B

S_A

S

Chương 4: Phân tích ngũ nghĩa

1. Giới thiệu
2. Bảng ký hiệu
3. Chương trình dịch định hướng cú pháp
4. Kiểm tra kiểu
5. Xử lý sai sót

Kiểm tra kiểu

- Có hai phương pháp kiểm tra kiểu
- Kiểm tra tĩnh : áp dụng trong thời gian dịch
 - Trong nhiều ngôn ngữ (C hay Pascal,...) kiểm tra kiểu là tĩnh và được dùng để kiểm tra tính đúng đắn của chương trình trước khi nó được thực hiện
- Kiểm tra động : thực hiện trong thời gian thực thi
 - Một số phép kiểm tra chỉ có thể thực hiện động
 - `int Tab[255], i;` → khi thực hiện `Tab[i]` có thể nằm ngoài phạm vi
- Bộ kiểm tra kiểu được xây dựng dựa trên
 - Các **biểu thức kiểu** của ngôn ngữ
 - Bộ **luật để định kiểu** cho các cấu trúc

Biểu thức kiểu (type expression)

Là một kiểu dữ liệu chuẩn hoặc được xây dựng từ các kiểu dữ liệu khác bởi cấu trúc kiểu

1. Kiểu cơ sở (*boolean, char, integer, real*) là biểu thức kiểu.

Chấp nhận các kiểu cơ sở đặc biệt:

- **type_error**: chỉ ra một lỗi trong quá trình kiểm tra kiểu
- **void**: cho phép kiểm tra kiểu đối với lệnh.

2. Do biểu thức kiểu có thể được đặt tên \Rightarrow Tên là biểu thức kiểu

3. Áp dụng toán tử xây dựng kiểu (*cấu trúc kiểu*) vào các biểu thức kiểu tạo ra biểu thức kiểu

Cấu trúc kiểu

- **Mảng (Array):**

- Nếu T là biểu thức kiểu thì $array(I, T)$ là biểu thức kiểu biểu diễn một mảng với các phần tử kiểu T và chỉ số trong miền I
 - Khai báo: $array [10] of integer$
 - có kiểu: $array(1..10,int); //array(10,int)$
 - Ví dụ: $Array[10] of Array[20] of integer$
 - $Array(10,Array(20,int))$

- **Tích Descarter:**

- Nếu T_1 và T_2 là các biểu thức kiểu thì tích Descarter $T_1 \times T_2$ là biểu thức kiểu

Cấu trúc kiểu

- **Bản ghi (Record):**

- Là tích descarte các kiểu của các trường bên trong

- Ví dụ:

```
struct{  
    double r;  
    int i;  
}
```

Có kiểu ((r x double) x (i x int))

Cấu trúc kiểu

- **Con trỏ:**
 - Nếu T là biểu thức kiểu thì $\text{pointer}(T)$ là biểu thức kiểu
- **Hàm**
 - Nếu D là miền xác định và R là miền giá trị của hàm thì kiểu của nó được biểu diễn là $D \rightarrow R$.

Ví dụ

- $\text{int } f(\text{char } a, \text{char } b) \rightarrow$ Có kiểu: $\text{char} \times \text{char} \rightarrow \text{int}$.
- $\text{int } * f(\text{int } a, \text{int } b) \rightarrow$ có kiểu $\text{int} \times \text{int} \rightarrow \text{pointer}(\text{int})$

Hệ thống kiểu

- Tập các luật để ấn định các biểu thức kiểu vào các phần khác nhau của chương trình
- Bộ kiểm tra kiểu thực hiện một hệ thống kiểu
- Được định nghĩa thông qua định nghĩa tựa cú pháp

Bộ kiểm tra kiểu đơn giản

Xét văn phạm

$$P \rightarrow D; E$$
$$D \rightarrow D; D \mid id : T$$
$$T \rightarrow \text{char} \mid \text{int} \mid \text{array}[num] \text{ of } T \mid ^T$$
$$E \rightarrow \text{literal} \mid \text{num} \mid id \mid E \text{ mod } E \mid E[E] \mid E^$$

Văn phạm sinh ra chương trình bắt đầu bởi các khai báo D và sau đó là một biểu thức E

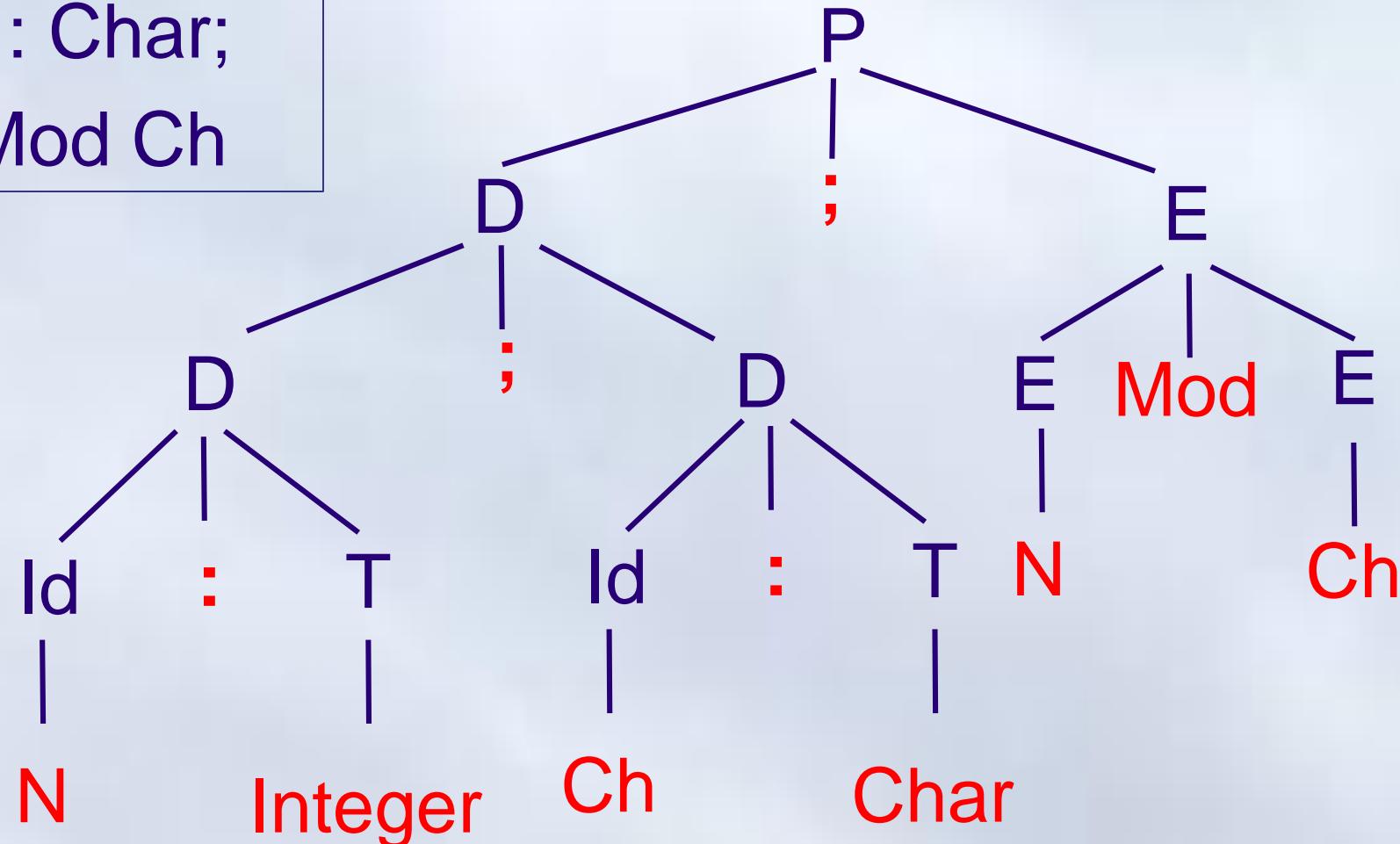
- Giả thiết chỉ số mảng bắt đầu từ 1

Bộ kiểm tra kiểu của định danh → Ví dụ

N : integer;

Ch : Char;

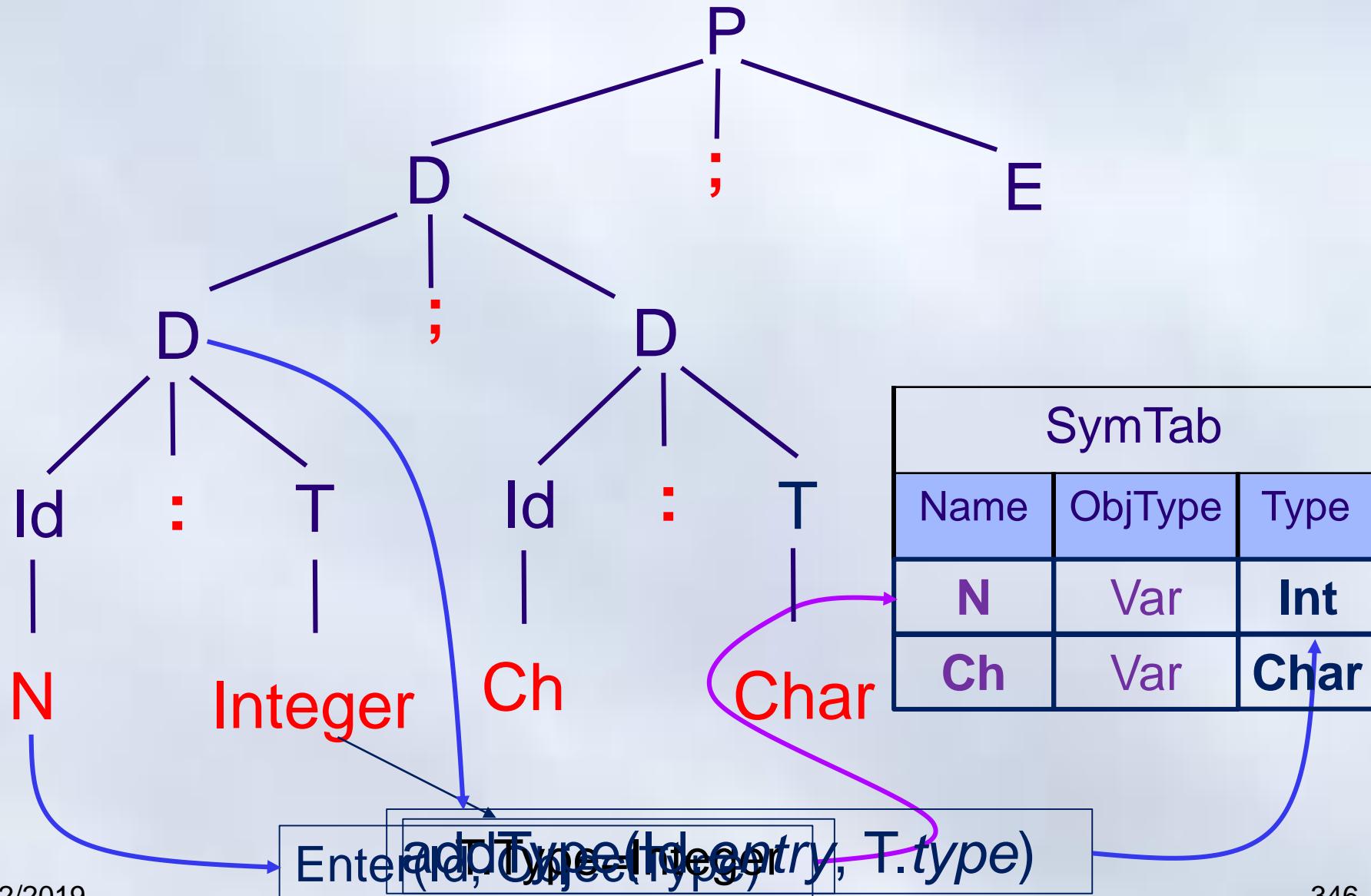
N Mod Ch



Bộ kiểm tra kiểu của định danh

Sản xuất	Quy tắc ngữ nghĩa
$P \rightarrow D ; E$	
$D \rightarrow D ; D$	
$D \rightarrow Id : T$	<code>addType(Id.entry, T.type)</code>
$T \rightarrow \text{char}$	$T.type = \text{char}$
$T \rightarrow \text{integer}$	$T.type = \text{integer}$
$T \rightarrow {}^{\wedge}T_1$	$T.type = \text{pointer}(T_1.type)$
$T \rightarrow \text{array}[num] \text{ of } T_1$	$T.Type = \text{array}(1..num.val, T_1.type)$
<p>Do $P \rightarrow D ; E$ nên kiểu của các định danh được khai báo đều được lưu trong bảng ký hiệu trước khi biểu thức E được kiểm tra kiểu</p>	

Bộ kiểm tra kiểu của định danh → Ví dụ



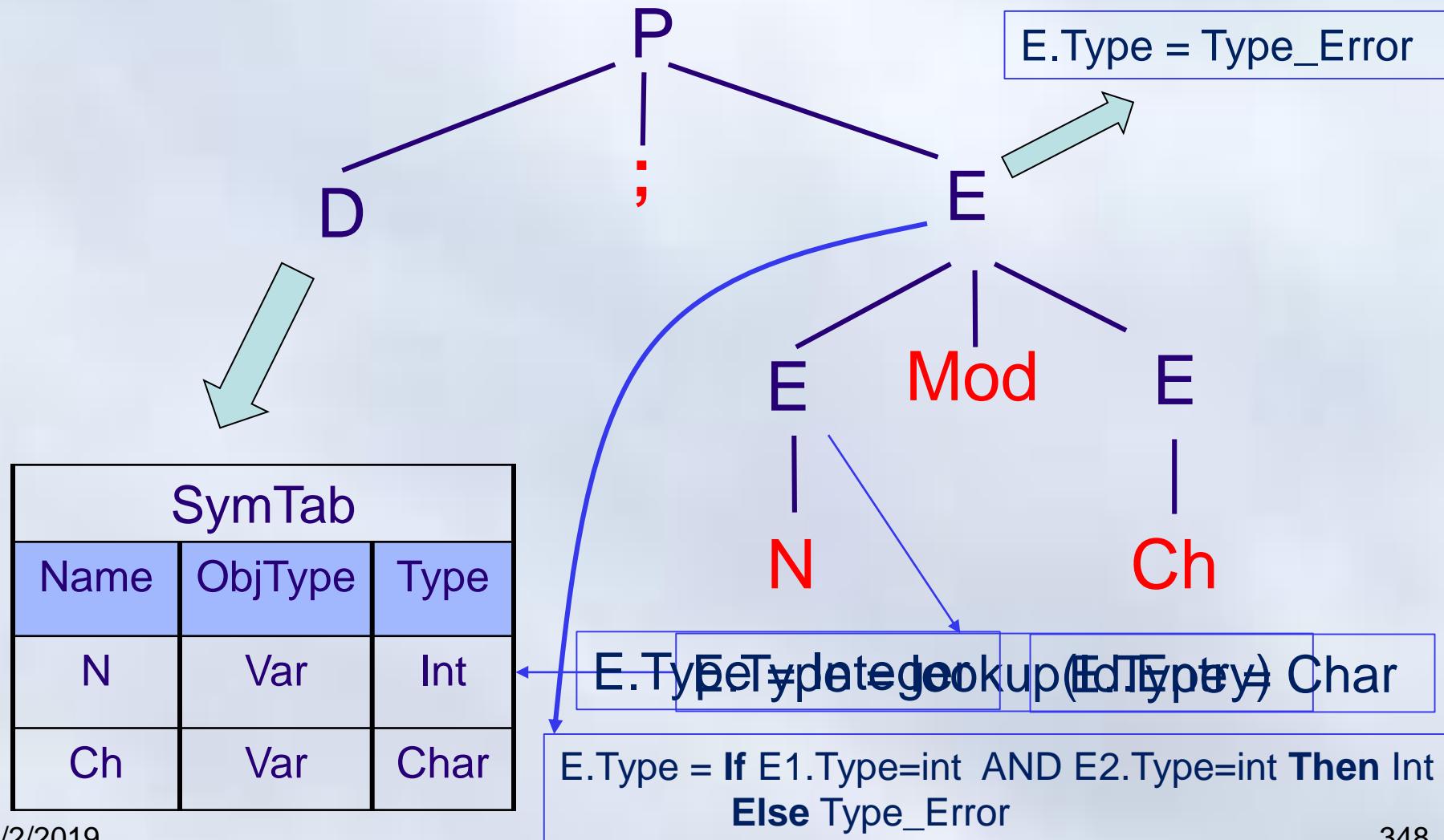
Bộ kiểm tra kiểu của biểu thức

SẢN XUẤT	QUY TẮC NGỮ NGHĨA
$E \rightarrow \text{literal}$	$E.type := \text{char}$
$E \rightarrow \text{num}$	$E.type := \text{int}$
$E \rightarrow \text{id}$	$E.type := \text{lookup}(\text{id.entry})$
$E \rightarrow E_1 \text{ mod } E_2$	$E.type := \text{if } E_1.type = \text{int} \text{ and } E_2.type = \text{int}$ then int else type_error
$E \rightarrow E_1[E_2]$	$E.type := \text{if } E_2.type = \text{int} \text{ and } E_1.type = \text{array}(s,t)$ then t else type_error
$E \rightarrow E_1 \uparrow$	$E.type := \text{if } E_1.type = \text{pointer}(t) \text{ then } t$ else type_error

Hàm $\text{lookup}(\text{idx})$ trả về kiểu trong bảng ký hiệu tại vị trí idx

Bộ kiểm tra kiểu của định danh → Ví dụ

N : integer; Ch : Char; N Mod Ch

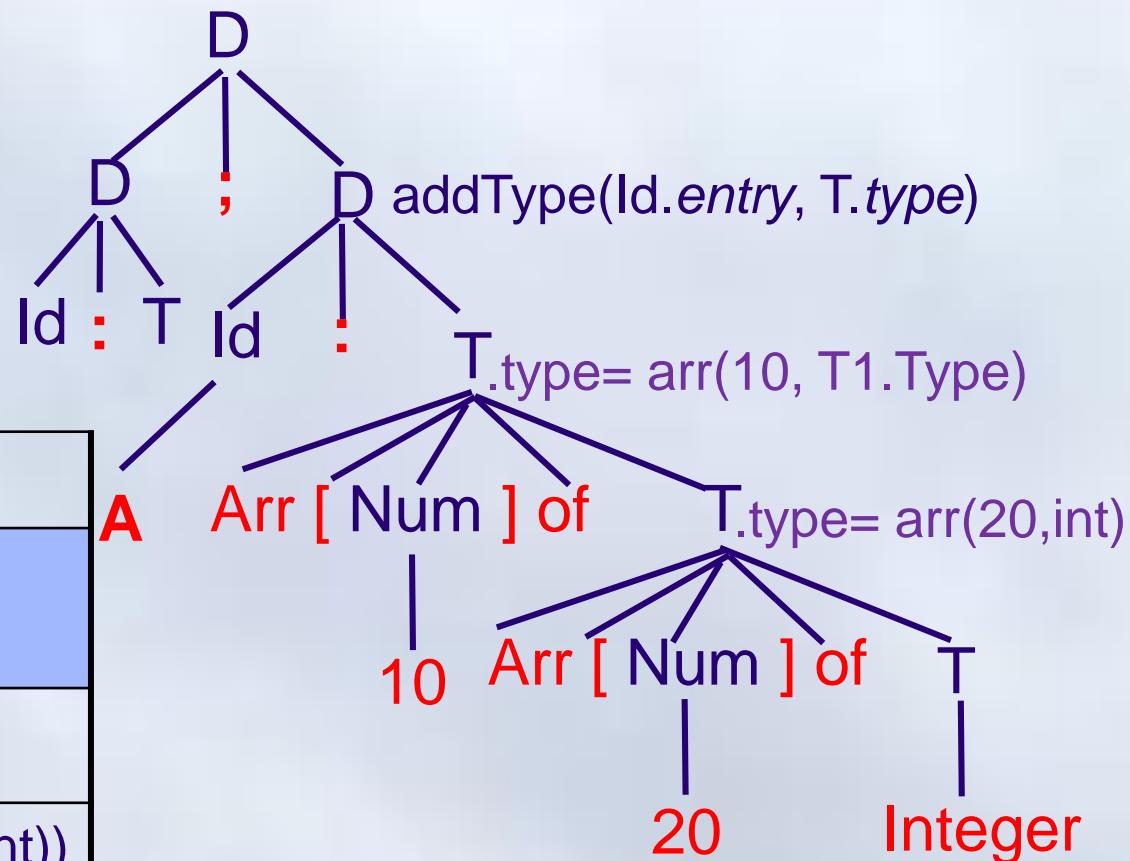


Bộ kiểm tra kiểu của định danh → Ví dụ 2

N : integer;

A : Array[10] of Array[20] of Integer;

A[10][10] Mod 100



SymTab		
Name	Obj Type	Type
N	Var	Int
A	Var	Arr (10,Arr(20,int))

Bộ kiểm tra kiểu của định danh → Ví dụ 2

N : integer;

A : Array[10] of Array[20] of Integer;

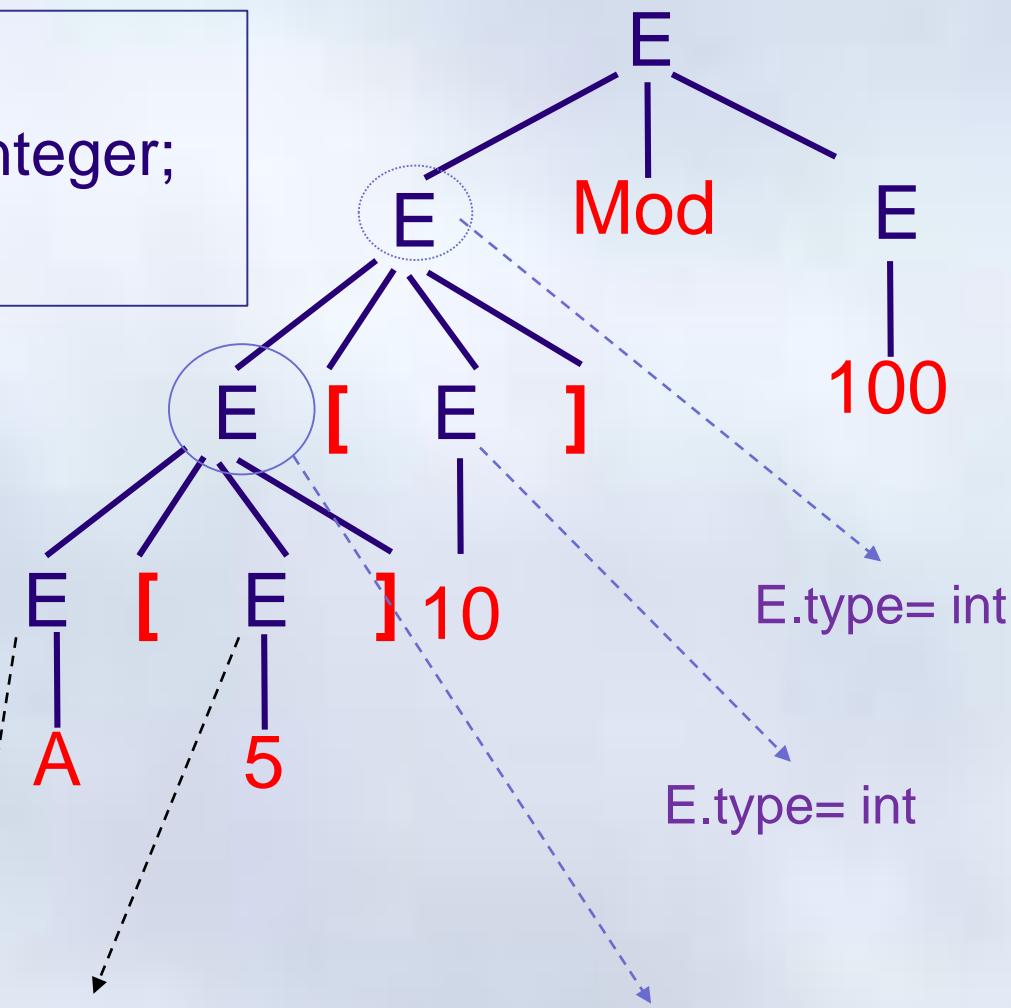
A[5][10] Mod 100

SymTab		
Name	Obj Type	Type
N	Var	Int
A	Var	Arr (10,Arr(20,int))

E.type= A.type=arr(10,Arr(20,int))

E.type= int

E.type= Arr(20,int)



Bộ kiểm tra kiểu của câu lệnh

- Với các câu lệnh không có giá trị (*lệnh gán, lệnh gộp,..*) sẽ có kiểu cơ sở **void**
- Lỗi trong câu lệnh thì kiểu của lệnh là: **type_error**
- Một chương trình hoàn chỉnh có thêm luật **P → D; S**

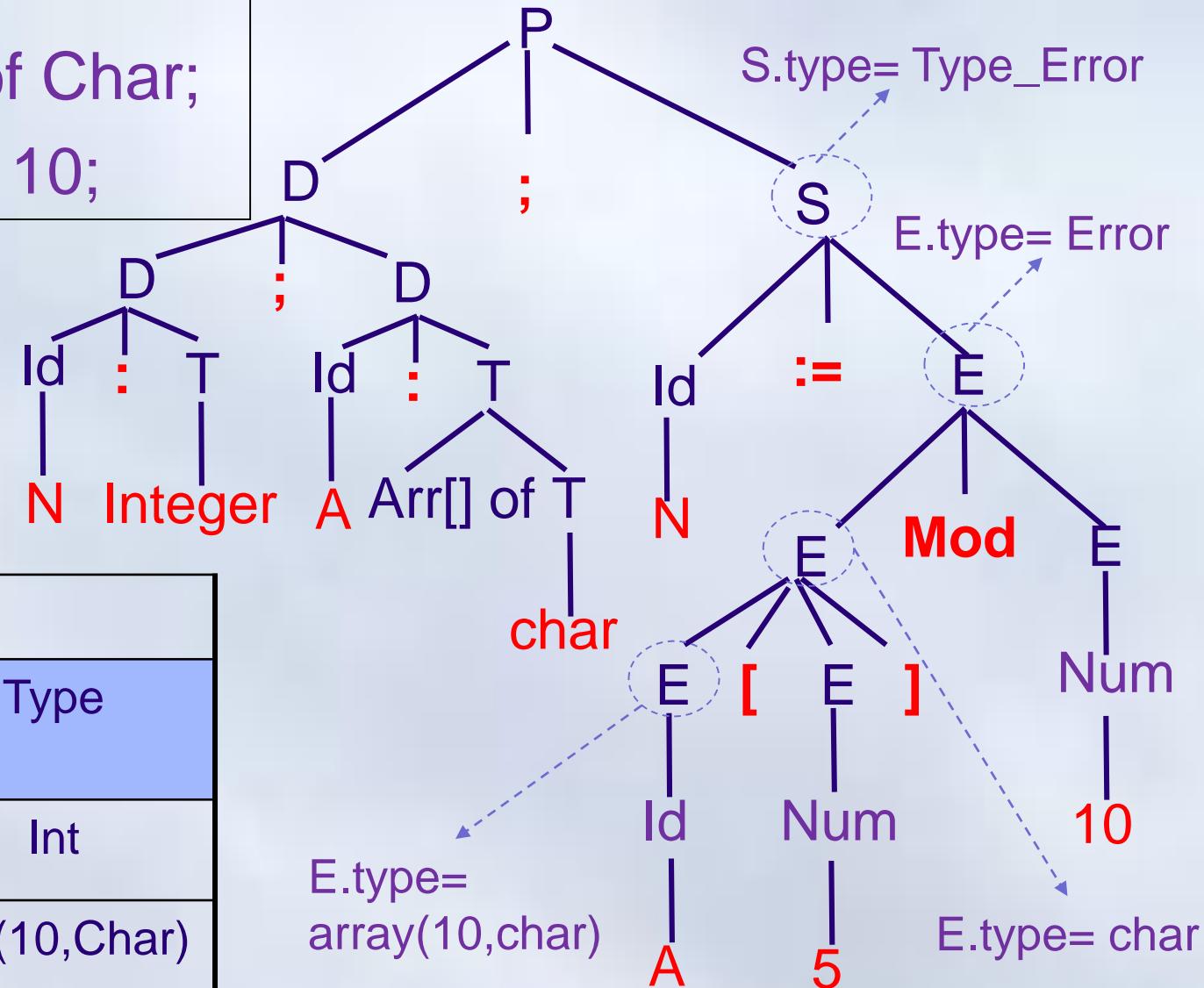
SẢN XUẤT	QUY TẮC NGỮ NGHĨA
$S \rightarrow \text{id} := E$	$S.type := \text{if } \text{id.type} = E.type \text{ then void}$ else type_error
$S \rightarrow \text{if } E \text{ then } S_1$	$S.type := \text{if } E.type = \text{boolean} \text{ then } S_1.type$ else type_error
$S \rightarrow \text{while } E \text{ do } S_1$	$S.type := \text{if } E.type = \text{boolean} \text{ then } S_1.type$ else type_error
$S \rightarrow S_1; S_2$	$S.type := \text{if } S_1.type = \text{void} \text{ and } S_2.type = \text{void}$ then void else type_error

Bộ kiểm tra kiểu của câu lệnh → Ví dụ

N : integer;

A : Array[10] of Char;

N := A[5] Mod 10;



SymTab		
Name	Obj Type	Type
N	Var	Int
A	Var	Arr (10,Char)

Bộ kiểm tra kiểu của hàm

Sản xuất	Quy tắc ngữ nghĩa
$D \rightarrow Id : T$	$\text{addType}(Id.\text{entry}, T.\text{type})$ $D.\text{type} = T.\text{type}$
$D \rightarrow D_1 ; D_2$	$D.\text{type} = D_1.\text{type} \times D_2.\text{type}$
$D \rightarrow \text{fun } id(D):T; B$	$\text{addType}(id.\text{entry}, D.\text{type} \rightarrow T.\text{type})$
$B \rightarrow \{ S \}$	
$S \rightarrow Id(EList)$ //gọi hàm	$S.\text{type} =$ if lookup ($Id.\text{entry}$) = $t_1 \rightarrow t_2$ AND $EList.\text{type} = t_1$ then t_2 else type_error
$EList \rightarrow E$	$EList.\text{type} = E.\text{type}$
$EList \rightarrow EList_1, E$	$EList.\text{type} = EList_1.\text{type} \times E.\text{type}$

Kiểm tra biểu thức kiểu tương đương

```
//s và t là các biểu thức kiểu
function sequiv(s, t): boolean;
begin
```

VAR A, B : MATRIX;
A và B có cùng kiểu?

if s và t có cùng kiểu dữ liệu chuẩn then return true;

else if s = array(s1, s2) and t = array(t1, t2) then

 return sequiv(s1, t1) and sequiv(s2, t2)

else if s = s1 x s2 and t = t1 x t2 then

 return sequiv(s1, t1) and sequiv(s2, t2)

else if s = pointer(s1) and t = pointer(t1) then

 return sequiv(s1, t1)

else if s = s1 → s2 and t = t1 → t2 then

 return sequiv(s1, t1) and sequiv(s2, t2)

else return false;

end;

Chuyển kiểu

float x; int n;

$x + n \rightarrow$ Quy đổi về một kiểu

SẢN XUẤT	QUY TẮC NGỮ NGHĨA
$E \rightarrow \text{num}$	$E.type := \text{int}$
$E \rightarrow \text{num.num}$	$E.type := \text{real}$
$E \rightarrow \text{id}$	$E.type := \text{lookup}(\text{id.entry})$
$E \rightarrow E_1 \text{ op } E_2$	$E.type := \begin{cases} \text{if } E_1.type = \text{int} \text{ and } E_2.type = \text{int} \\ \quad \text{then } \text{int} \\ \text{else if } E_1.type = \text{int} \text{ and } E_2.type = \text{real} \\ \quad \text{then } \text{real} \\ \text{else if } E_1.type = \text{real} \text{ and } E_2.type = \text{int} \\ \quad \text{then } \text{real} \\ \text{else if } E_1.type = \text{real} \text{ and } E_2.type = \text{real} \\ \quad \text{then } \text{real} \\ \text{else } type_error \end{cases}$

Chương 4: Phân tích ngũ nghĩa

1. Giới thiệu
2. Bảng ký hiệu
3. Kiểm tra kiểu
4. Chương trình dịch định hướng cú pháp
5. Xử lý sai sót

Giới thiệu

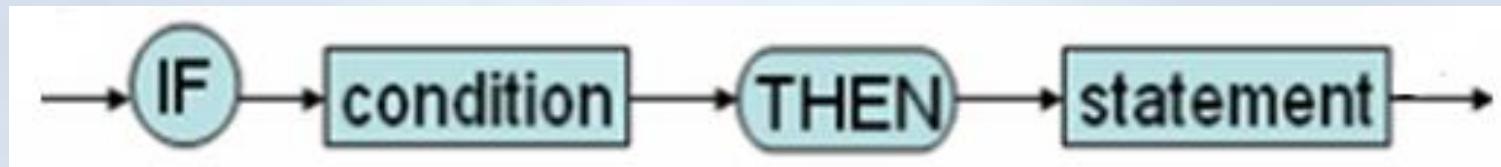
Phân tích cú pháp nếu gặp lỗi

- Gọi tới thủ tục **Error(...)** để báo lỗi
 - Thông báo kiểu của lỗi và vị trí của lỗi
- Sau khi báo lỗi ngừng quá trình dịch lại
 - Bởi văn bản nguồn không khớp với dự đoán → Chương trình dịch không biết đi theo hướng nào
 - VD: gặp statement **(X):=10**; sẽ không biết đi nhánh nào
 - Chương trình dịch có **n** lỗi → cần **n+1** lần dịch
 - Tốn kém
 - Cần điều chỉnh để CTD có thể hồi phục sau khi gặp lỗi

Phương pháp

- Vượt bỏ một đoạn trên văn bản nguồn cho tới khi tìm được một từ tố cho phép nắm bắt lại được cấu trúc chương trình và có thể tiếp tục làm việc theo dự kiến
- Các từ tố cho phép nắm bắt được cấu trúc là các **ký hiệu ngừng**. Có thể là
 - Các từ tố theo sau cấu trúc đang phân tích
 - Khi gấp, cho biết cấu trúc đã bị bỏ qua
 - Là **FOLLOW()** của cấu trúc
 - Các từ tố trong cấu trúc hiện tại, ít khi bị bỏ xót
 - Được gọi là **Các ký hiệu chốt**
 - Thường là từ khóa của cấu trúc hiện tại

Phương pháp → Ví dụ

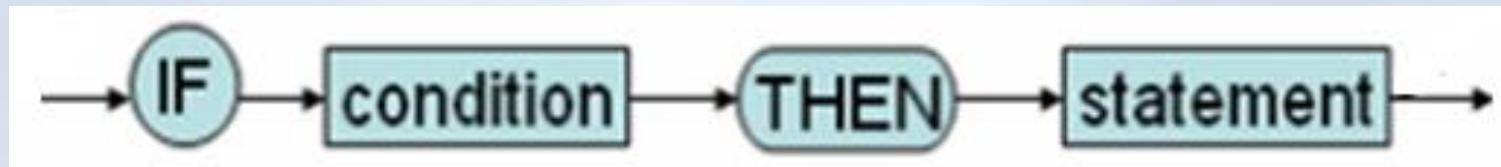


if max > a + *10 then

Lỗi ở biểu thức (Term) (thiếu toán hạng hoặc dấu '(')

- Sau Condition (Expression) là Then, Do,..
- Khi gặp KW_THEN → đã bỏ qua hết **Condition**
 - Ghi nhận lỗi (coi như đã triển khai xong Condition)
 - Có thể tiếp tục triển khai **Statement**

Phương pháp → Ví dụ



if **max > a + 10** Min then

Condition

Lỗi ở Statement, nhánh
if : (thiếu từ khóa **Then**)

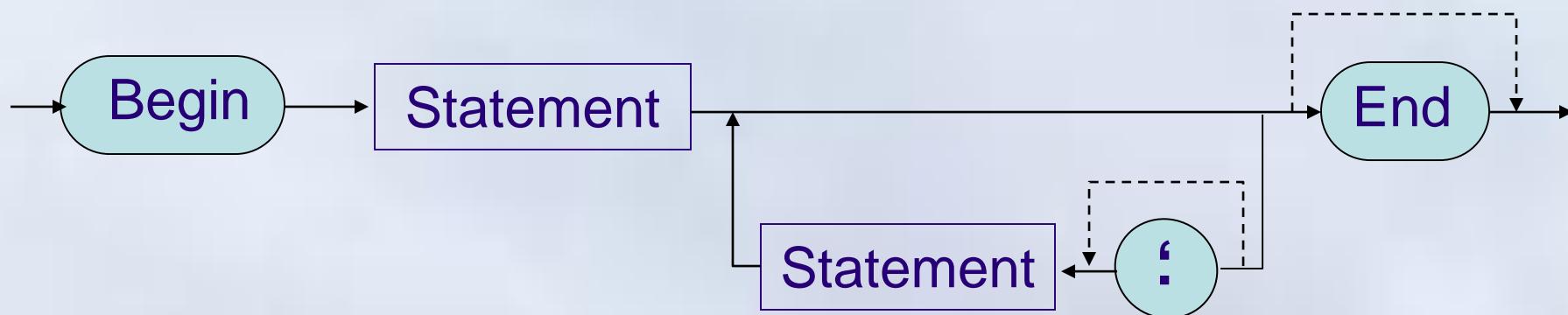
- Sau Statement là **End ; .**
- Không nhất thiết phải bỏ cả statement cho tới khi gấp **End ; .**
- Khi đến **Then** là có thể bắt đầu phân tích tiếp

Phương pháp → Thực hiện

- Để thực hiện, cần cho biết các thủ tục phân tích cú pháp (*dùng để triển khai một đích, ví dụ compileBlock(), compileStatement(),...*) có thể hồi phục ở những ký hiệu ngừng nào
 - Các ký hiệu chốt thuộc chính cấu trúc mà thủ tục đang phân tích
 - Các ký hiệu theo sau cấu trúc (phụ thuộc chỗ gọi)
Ví dụ: Đang phân tích Condition, nếu Condition được gọi trong nhánh **IF**, ký hiệu theo sau là **THEN**, còn trong nhánh **WHILE**, ký hiệu theo sau là **DO**
 - Các ký hiệu ngừng được kế thừa từ các cấu trúc (thủ tục) bên trên

Phương pháp ghi nhận lỗi

- Khi lỗi có thể đoán biết được không cần vượt quá một đoạn trong chương trình nguồn
 - Ví dụ, lỗi thiếu dấu ; hoặc **End** trong statement
- Có thể sửa lại sơ đồ và viết lại theo sơ đồ mới



Phương pháp ghi nhận lỗi

```

if (token == BEGIN){
    getToken();
    compileStatement();
    while (token ∈ [Semicolon] ∪ FIRST(Statement) do{
        if(Token == Semicolon)
            getToken();
        else
            addError(« Thiếu dấu ';' »);
            compileStatement();
    }
    if (token == END) getToken();
    else addError(« Thiếu từ khóa End »);
}

```

addError() chỉ
ghi nhận một
thông báo lỗi,
không dừng
chương trình

IT4079:NGÔN NGỮ và PHƯƠNG PHÁP DỊCH

Phạm Đăng Hải

haipd@soict.hust.edu.vn

Chương 5: Sinh mã

1. Sinh mã trung gian

2. Sinh mã đích

3. Tối ưu mã

Giới thiệu

- Bộ sinh mã trung gian chuyển chương trình nguồn sang chương trình tương đương trong ngôn ngữ trung gian
 - Chương trình trung gian là một chương trình cho một máy trùu tượng
- Ngôn ngữ trung gian được người thiết kế trình biên dịch quyết định, có thể là:
 - Cây cú pháp
 - Ký pháp Ba Lan sau (hậu tố)
 - Mã 3 địa chỉ ...

Nội dung

- Chương trình dịch định hướng cú pháp
- Cây cú pháp
- Ký pháp Ba lan sau
- Mã 3 địa chỉ
 - Các dạng mã
 - Dịch trực tiếp cú pháp thành mã 3 địa chỉ
 - Sinh mã cho khai báo
 - Sinh mã cho lệnh gán
 - Sinh mã cho các biểu thức logic
 - Sinh mã cho các cấu trúc lập trình

Chương trình dịch định hướng cú pháp

Mỗi ký hiệu VP liên kết với một tập thuộc tính:

- **Thuộc tính tổng hợp:**

- Giá trị của thuộc tính tại một nút trong cây được xác định từ giá trị của các nút con của nó.

- **Thuộc tính kế thừa:**

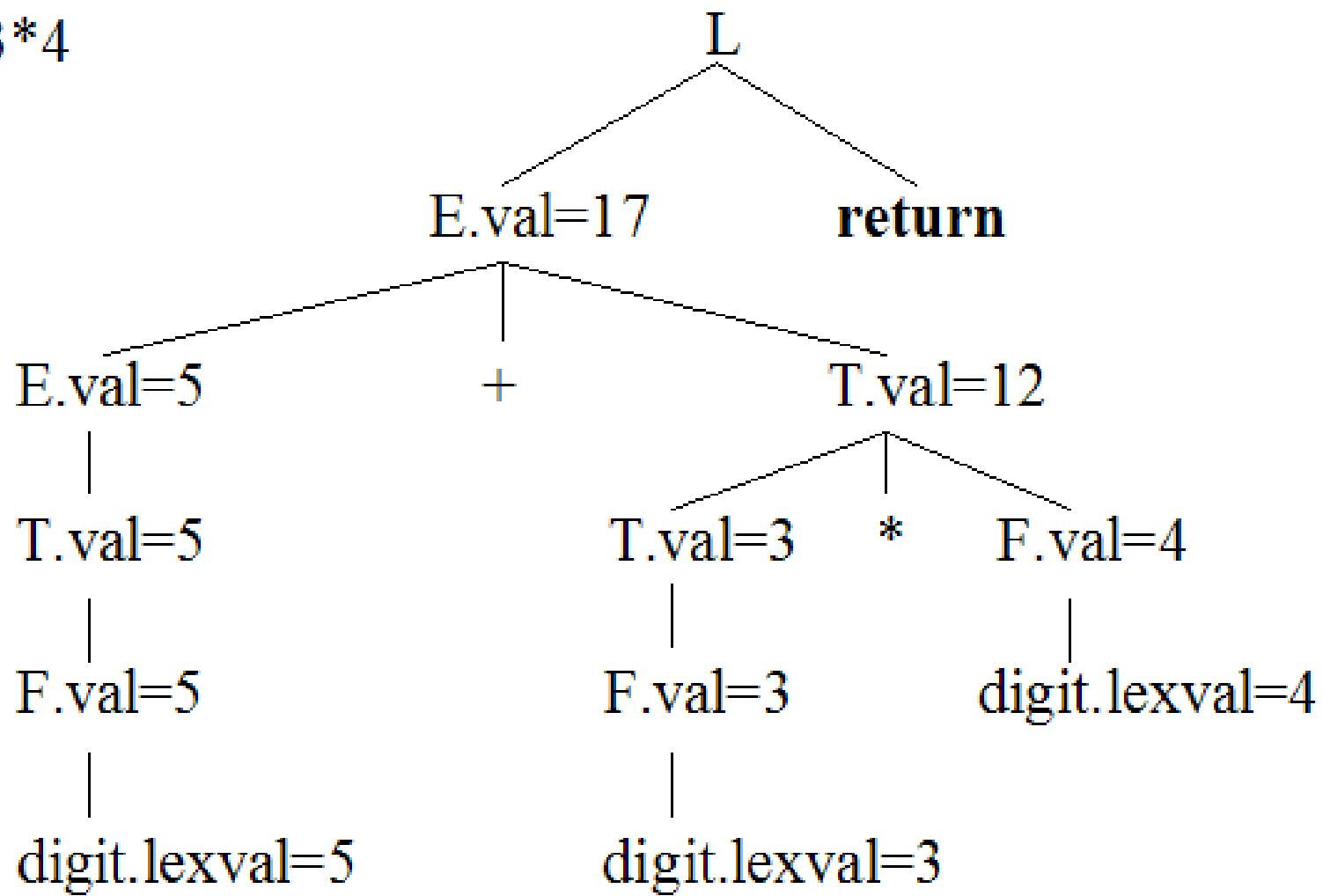
- Giá trị của thuộc tính được định nghĩa dựa vào giá trị nút cha và/hoặc các nút anh em của nó.

- Tồn tại một tập luật ngữ nghĩa dùng để tính giá trị thuộc tính

Ví dụ

Sản xuất	Quy tắc ngũ nghĩa
$L \rightarrow E \text{ return}$	$\text{Print}(E.\text{val})$
$E \rightarrow E_1 + T$	$E.\text{val} = E_1.\text{val} + T.\text{val}$
$E \rightarrow T$	$E.\text{val} = T.\text{val}$
$T \rightarrow T_1 * F$	$T.\text{val} = T_1.\text{val} * F.\text{val}$
$T \rightarrow F$	$T.\text{val} = F.\text{val}$
$F \rightarrow (E)$	$F.\text{val} = E.\text{val}$
$F \rightarrow \text{digit}$	$F.\text{val} = \text{digit}.\text{lexval}$
<ul style="list-style-type: none"> Các ký hiệu E, T, F có thuộc tính tổng hợp val Từ tố digit có thuộc tính tổng hợp lexval (Được bộ phân tích từ vựng đưa ra) 	

Chú giải cây suy dẫn

Input: $5+3*4$ 

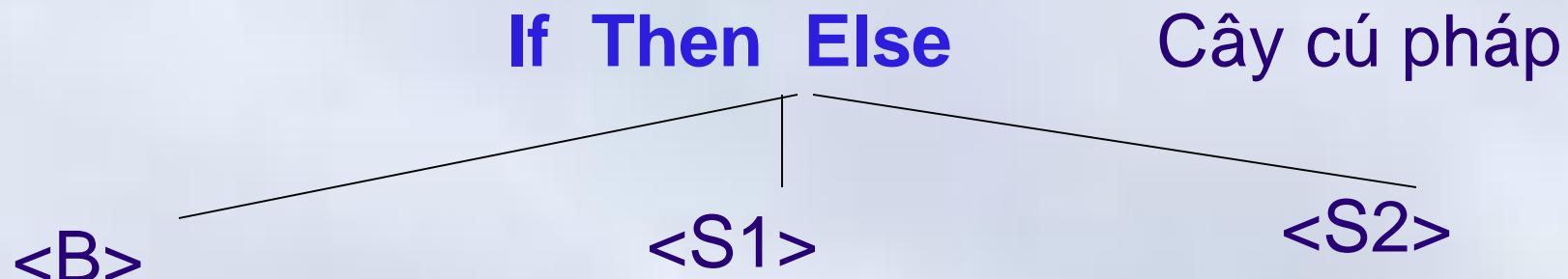
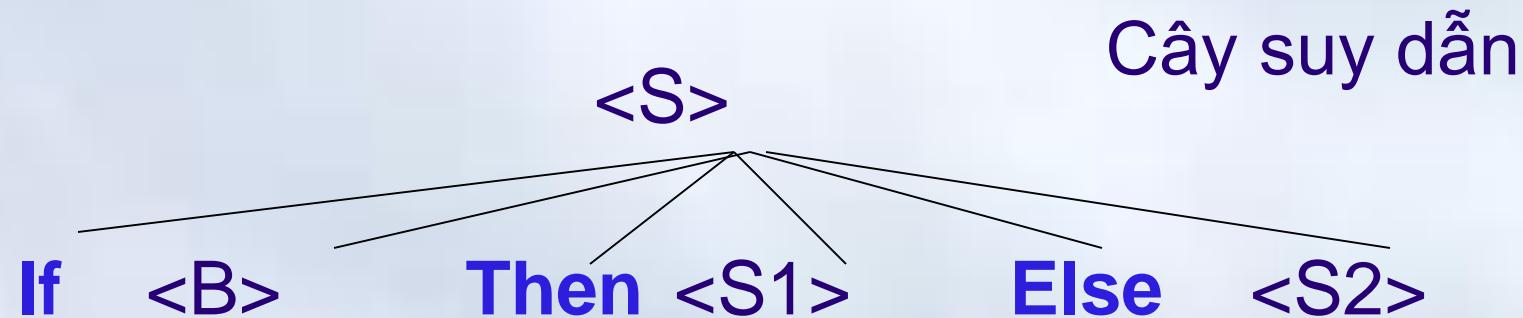
Nội dung

- Chương trình dịch định hướng cú pháp
- Cây cú pháp
- Ký pháp Ba lan sau
- Mã 3 địa chỉ
 - Các dạng mã
 - Dịch trực tiếp cú pháp thành mã 3 địa chỉ
 - Sinh mã cho khai báo
 - Sinh mã cho lệnh gán
 - Sinh mã cho các biểu thức logic
 - Sinh mã cho các cấu trúc lập trình

Cây cú pháp (Syntax tree)

- Cây cú pháp (*syntax tree*) là dạng thu gọn của cây phân tích (*parse tree*) dùng để biểu diễn cấu trúc của ngôn ngữ
- Trong cây cú pháp các toán tử và từ khóa không xuất hiện ở các nút lá mà đưa vào các nút trong.
 - Cha của các nút lá là các toán hạng tương ứng
- Cây cú pháp có ý nghĩa dụng trong cài đặt
 - Cây phân tích (cú pháp) chỉ ý nghĩa về mặt logic

Cây cú pháp → Ví dụ 1

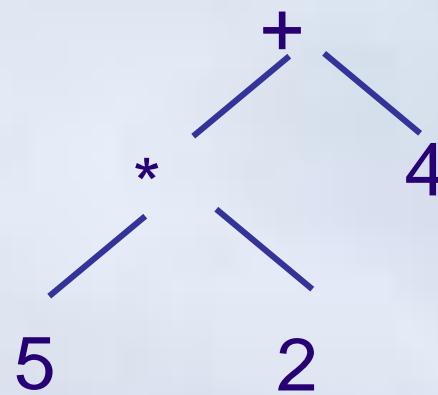
Ví dụ: $S \rightarrow \mathbf{If} \ B \ \mathbf{Then} \ S1 \ \mathbf{Else} \ S2$ 

Xây dựng cây cú pháp → Ví dụ 2

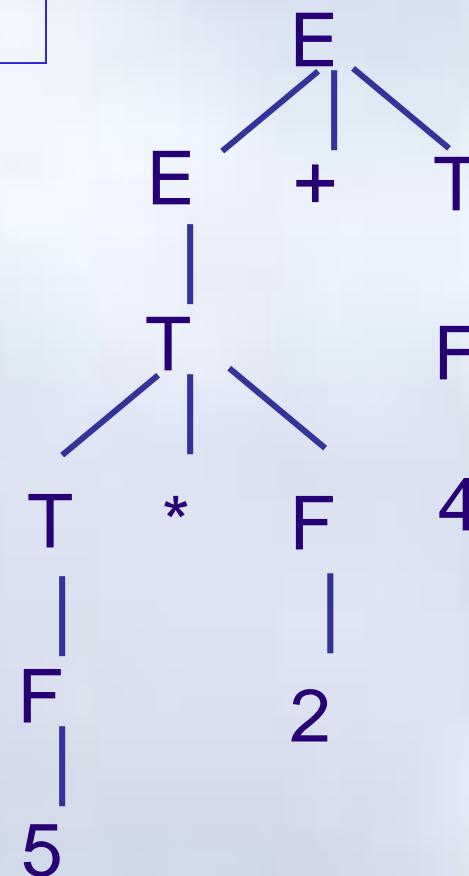
$$E \rightarrow E + T \mid T$$

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

$$F \rightarrow (E) \mid \text{num}$$

$$5 * 2 + 4$$


Cây cú pháp

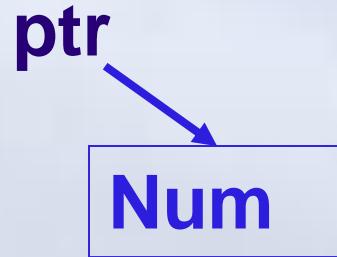


Cây phân tích

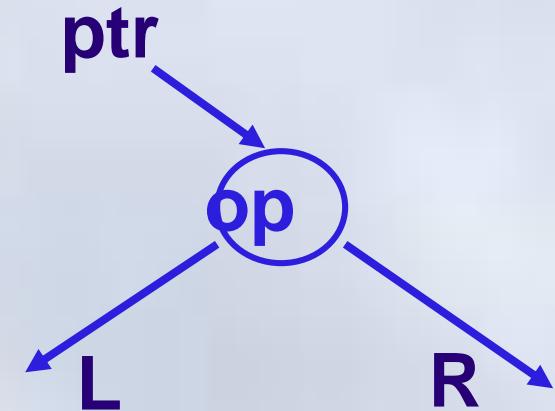
Xây dựng cây cú pháp cho biểu thức

- Các ký hiệu không kết thúc có thuộc tính tổng hợp **link** để lưu con trỏ, trỏ tới một nút trên cây cú pháp
- Sử dụng các hàm

`ptr = mkLeaf(Num)`



`ptr = mkNode(op, L, R)`



Cây cú pháp (Syntax tree)

Sản xuất	Luật ngữ nghĩa
$E \rightarrow E_1 + T$	$E.\text{link} := \text{mkNode}(+, E_1.\text{link}, T.\text{link})$
$E \rightarrow T$	$E.\text{link} := T.\text{link}$
$T \rightarrow T_1 * F$	$T.\text{link} := \text{mkNode}(*, T_1.\text{link}, F.\text{link})$
$T \rightarrow F$	$T.\text{link} := F.\text{link}$
$F \rightarrow (E)$	$F.\text{link} := E.\text{link}$
$F \rightarrow \text{num}$	$F.\text{link} := \text{mkLeaf}(\text{num})$

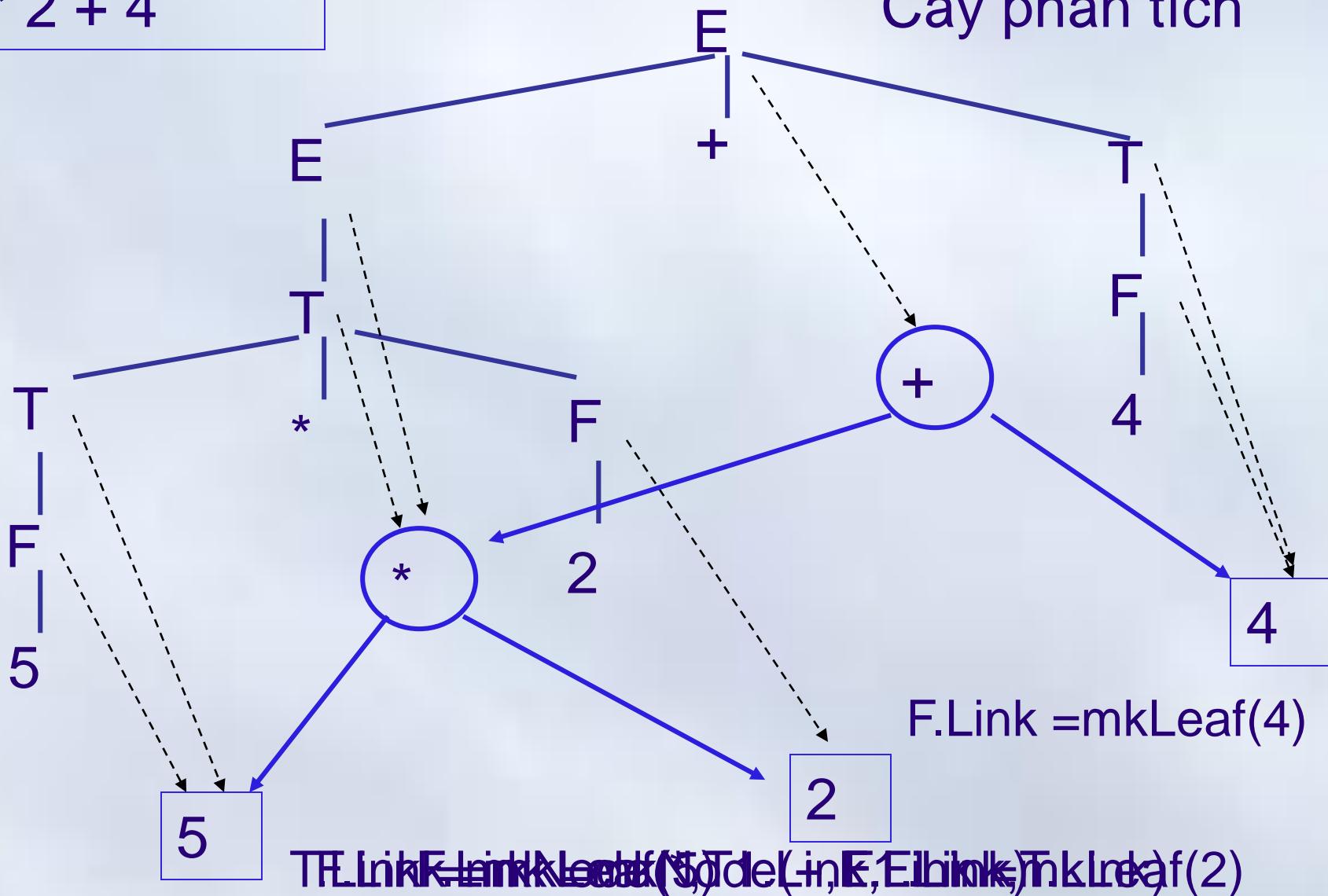
Cây cú pháp được xây dựng từ dưới lên trên

- Sau khi phân tích xong một sản xuất mới gọi luật ngữ nghĩa tương ứng (*duyệt thứ tự sau*)

Xây dựng cây cú pháp → Ví dụ

5 * 2 + 4

Cây phân tích



Nội dung

- Chương trình dịch định hướng cú pháp
- Cây cú pháp
- Ký pháp Ba lan sau
- Mã 3 địa chỉ
 - Các dạng mã
 - Dịch trực tiếp cú pháp thành mã 3 địa chỉ
 - Sinh mã cho khai báo
 - Sinh mã cho lệnh gán
 - Sinh mã cho các biểu thức logic
 - Sinh mã cho các cấu trúc lập trình

Ký pháp Ba lan sau (Reverse Polish notation)

Ký pháp Ba lan: Là một ký hiệu toán học trong đó dấu đặt trước/ sau toán hạng

- Ký pháp thông thường (giữa): $5 + 6$
- Ký pháp Ba lan trước: $+ 5 6$
- Ký pháp Ba lan sau (ngược): $5 6 +$

Mục đích:

- Giảm thiểu bộ nhớ và dùng stack để tính toán

Sử dụng:

- Trong một số loại máy tính tay

Quy tắc dịch dạng trung tố → dạng hậu tố

Sản xuất	Ký pháp hậu tố	Ký pháp tiền tố
$E \rightarrow E + T$	$E \rightarrow E T +$	$E \rightarrow + E T$
$E \rightarrow T$	$E \rightarrow T$	$E \rightarrow T$
$T \rightarrow T * F$	$T \rightarrow T F *$	$T \rightarrow * T F$
$T \rightarrow F$	$T \rightarrow F$	$T \rightarrow F$
$F \rightarrow (E)$	$F \rightarrow E$	$F \rightarrow E$
$F \rightarrow \text{digit}$	$F \rightarrow \text{digit}$	$F \rightarrow \text{digit}$

Ký pháp Ba lan sau \rightarrow Ví dụ 1 $a+b^*c+d$

$E \Rightarrow E + T$	\Leftrightarrow	$E \Rightarrow ET +$
$\Rightarrow E + T + T$	\Leftrightarrow	$\Rightarrow ET + T +$
$\Rightarrow T + T + T$	\Leftrightarrow	$\Rightarrow TT + T +$
$\Rightarrow F + T + T$	\Leftrightarrow	$\Rightarrow FT + T +$
$\Rightarrow a + T + T$	\Leftrightarrow	$\Rightarrow aT + T +$
$\Rightarrow a + T^* F + T$	\Leftrightarrow	$\Rightarrow aTF^* + T +$
$\Rightarrow a + F^* F + T$	\Leftrightarrow	$\Rightarrow aFF^* + T +$
$\Rightarrow a + b^* F + T$	\Leftrightarrow	$\Rightarrow abF^* + T +$
$\Rightarrow a + b^* c + T$	\Leftrightarrow	$\Rightarrow abc^* + T +$
$\Rightarrow a + b^* c + F$	\Leftrightarrow	$\Rightarrow abc^* + F +$
$\Rightarrow a + b^* c + d$	\Leftrightarrow	$\Rightarrow abc^* + d +$

Ký pháp Ba lan sau \rightarrow Ví dụ 2
$$(a+b)^* (c+d)$$

$E \Rightarrow T$	\Leftrightarrow	$E \Rightarrow T$
$\Rightarrow T^* F$	\Leftrightarrow	$\Rightarrow T F^*$
$\Rightarrow F^* F$	\Leftrightarrow	$\Rightarrow F F^*$
$\Rightarrow (E)^* F$	\Leftrightarrow	$\Rightarrow E F^*$
$\Rightarrow (T + F)^* F$	\Leftrightarrow	$\Rightarrow T F + F^*$
$\Rightarrow (F + F)^* F$	\Leftrightarrow	$\Rightarrow F F + F^*$
$\Rightarrow (a + F)^* F$	\Leftrightarrow	$\Rightarrow a F + F^*$
$\Rightarrow (a + b)^* F$	\Leftrightarrow	$\Rightarrow a b + F^*$
$\Rightarrow (a + b)^* (E)$	\Leftrightarrow	$\Rightarrow a b + E^*$
$\Rightarrow (a + b)^* (T + F)$	\Leftrightarrow	$\Rightarrow a b + T F + ^*$
$\Rightarrow (a + b)^* (F + F)$	\Leftrightarrow	$\Rightarrow a b + F F + ^*$
$\Rightarrow (a + b)^* (c + F)$	\Leftrightarrow	$\Rightarrow a b + c F + ^*$
$\Rightarrow (a + b)^* (c + d)$	\Leftrightarrow	$\Rightarrow a b + c d + ^*$

Nội dung

- Chương trình dịch định hướng cú pháp
- Cây cú pháp
- Ký pháp Ba lan sau
- Mã 3 địa chỉ
 - Các dạng mã
 - Dịch trực tiếp cú pháp thành mã 3 địa chỉ
 - Sinh mã cho khai báo
 - Sinh mã cho lệnh gán
 - Sinh mã cho các biểu thức logic
 - Sinh mã cho các cấu trúc lập trình

Mã 3 địa chỉ

- Là loại mã trung gian thường dùng, tương tự mã assembly
- Chương trình trung gian là một dãy các lệnh thuộc kiểu mã 3 địa chỉ
 - Mỗi lệnh gồm tối đa 3 toán hạng
 - Tồn tại nhiều nhất một toán tử ở về phải cộng thêm một toán tử gán
- x,y,z là các địa chỉ, tức là tên, hằng hay các tên trung gian do trình biên dịch sinh ra
 - Tên trung gian phải được sinh để thực hiện các phép toán trung gian
 - Các địa chỉ được thực hiện như con trỏ tới phần tử tương ứng của nó trong bảng ký hiệu

Mã 3 địa chỉ → Ví dụ

- Câu lệnh

- $A = x + y * z$

- Chuyển thành mã 3 địa chỉ

$$T = y * z$$

$$A = x + T$$

T là tên trung gian

- Được bộ sinh mã trung gian sinh ra cho các toán tử trung gian

Mã 3 địa chỉ → Các dạng phổ biến

- Mã 3 địa chỉ tương tự mã Assembly:
 - Lệnh có thể có nhãn,
 - Tồn tại những lệnh chuyển điều khiển cho các cấu trúc lập trình.
- Các dạng lệnh
 - *Lệnh gán $x := y \ op \ z$.*
 - *Lệnh gán với phép toán 1 ngôi : $x := op \ y$.*
 - *Lệnh sao chép: $x := y$.*
 - *Lệnh gán có chỉ số $X := y[i]$ hoặc $x[i] = y$*

Mã 3 địa chỉ → Các dạng phổ biến

- *Lệnh gán địa chỉ và con trỏ*
 $x = \&y;$; $x = *y;$ $*x = y$
- *Lệnh nhảy không điều kiện: goto L,*
 - L là nhãn của một lệnh
- *Lệnh nhảy có điều kiện IF x relop y goto L.*
 - Nếu thỏa mãn quan hệ relop ($>$, \geq , $<$, ..) thì thực hiện lệnh tại nhãn L,
 - Nếu không thỏa mãn, thực hiện câu lệnh ngay tiếp theo lệnh IF

Mã 3 địa chỉ → Các dạng phổ biến

➤ Gọi thủ tục với n tham số **call p, n**.

Khai báo tham số

param x

Trả về giá trị

return y

Thường dung với chuỗi lệnh 3 địa chỉ

– Lời gọi chương trình con **Call p(X_1, X_2, \dots, X_n)** sinh ra

param x_1

param x_2

param x_n

Call p, n

Dịch trực tiếp cú pháp thành mã 3 địa chỉ

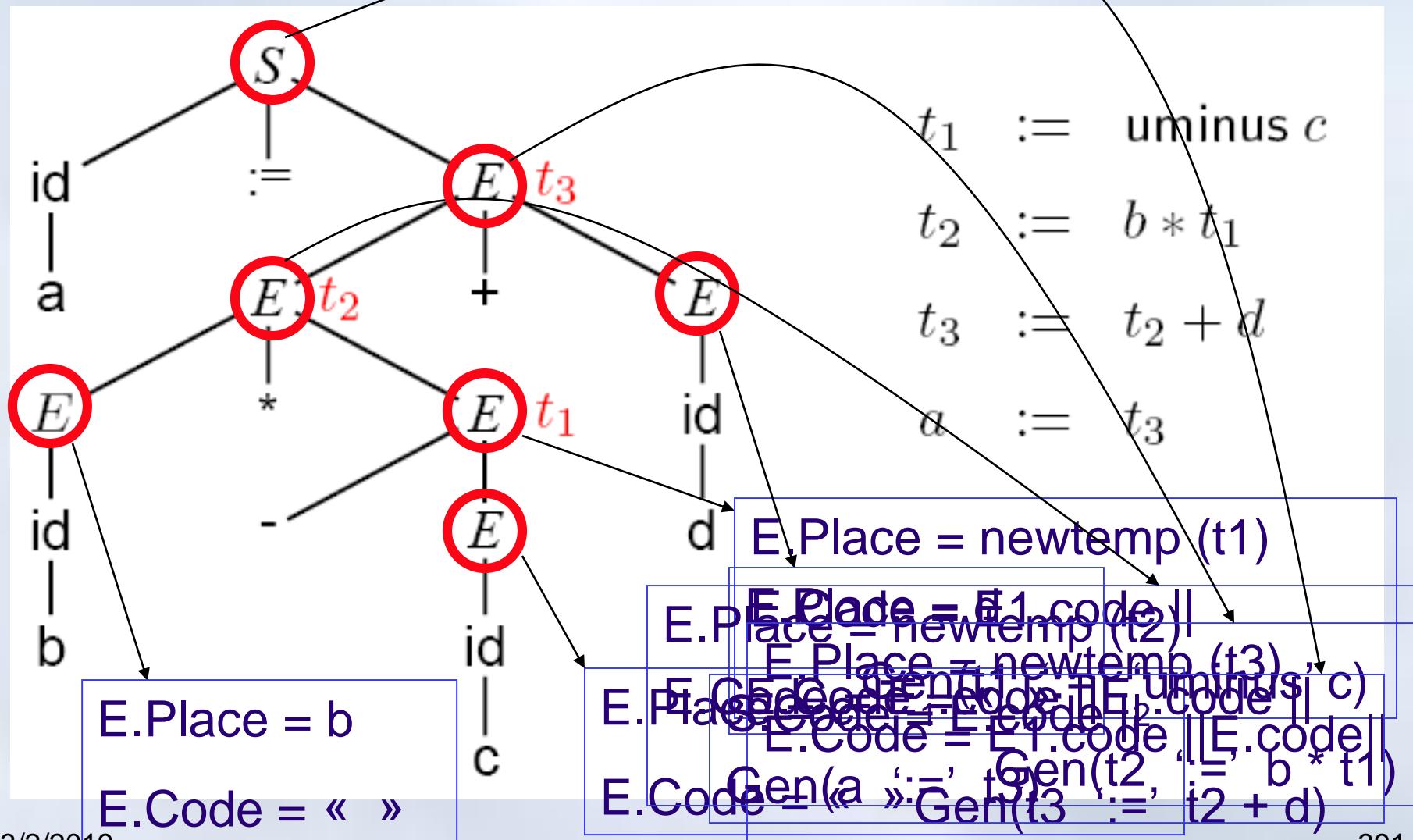
- Thuộc tính tổng hợp **S.code** biểu diễn mã ba địa chỉ của lệnh **S**
- Các tên trung gian được sinh ra cho các tính toán trung gian
- Các ký hiệu không kết thúc E có 2 thuộc tính
 - E.place: Thuộc tính địa chỉ/tên chứa giá trị của ký hiệu E
 - E.code: Chứa chuỗi mã lệnh địa chỉ để đánh giá E
- Hàm **newtemp()** sinh ra các tên trung gian t1, t2,..
- Sử dụng hàm **gen(x ':= y '+' z)** thể hiện mã 3 địa chỉ câu lệnh $x := y + z$
 - Các biểu thức ở các vị trí của x, y, z được đánh giá khi truyền vào hàm gen()

Dịch trực tiếp cú pháp thành mã 3 địa chỉ

Sản xuất	Quy tắc ngữ nghĩa
$S \rightarrow \text{Id} := E$	$S.\text{Code} = E.\text{code} \parallel \text{gen}(\text{id.place} ' := ' E.\text{place})$
$E \rightarrow E_1 + E_2$	$E.\text{Place} = \text{newTemp}()$ $E.\text{Code} = E_1.\text{code} \parallel E_2.\text{code} \parallel$ $\text{gen}(E.\text{place} ' := ' E_1.\text{place} ' + ' E_2.\text{place})$
$E \rightarrow E_1 * E_2$	$E.\text{Place} = \text{newTemp}()$ $E.\text{Code} = E_1.\text{code} \parallel E_2.\text{code} \parallel$ $\text{gen}(E.\text{place} ' := ' E_1.\text{place} ' * ' E_2.\text{place})$
$E \rightarrow -E_1$	$E.\text{place} = \text{newtemp}();$ $E.\text{code} = E_1.\text{code} \parallel$ $\text{gen}(E.\text{place} ' := ' 'uminus' E_1.\text{place})$
$E \rightarrow (E)$	$E.\text{place} = E_1.\text{place}; E.\text{code} = E_1.\text{code}$
$E \rightarrow \text{Id}$	$E.\text{place} = \text{id.place}; E.\text{code} = "$

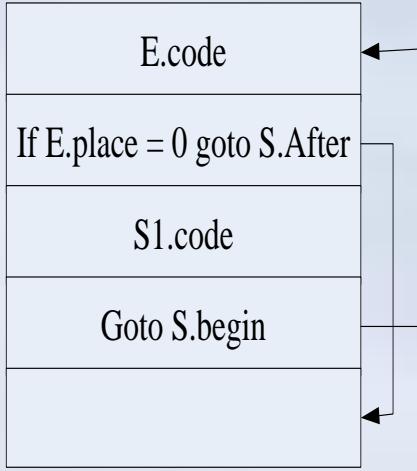
Dịch trực tiếp cú pháp thành mã 3 địa chỉ

Câu lệnh gán: ~~a := b * -c + d~~



Dịch trực tiếp cú pháp thành mã 3 địa chỉ

Ví dụ: Câu lệnh lặp while

Sản xuất	Quy tắc ngữ nghĩa
<p>$S \rightarrow \text{while } E \text{ do } S1$</p> <p>S.Begin</p>  <p>S.After</p>	<p>S.Begin = newLabel()</p> <p>S.After = newLabel()</p> <p>S.Code = /*Sinh mã cho lệnh while gồm*/ gen(S.begin ':') E.code /*Sinh mã cho lệnh đánh giá E*/ Gen('if' E.place '=' 0 'goto' S.After) S1.code /*Sinh mã cho lệnh S1*/ gen('goto' S.Begin) /*Sinh mã cho goto*/ Gen(S.After ':') /*Sinh mã cho nhãn mới*/</p>

Hàm newLabel(): Sinh ra một nhãn mới

Ví dụ:

While a + b Do a := a * b

S.Begin =

newLabel() =

L1

S.After =

newLabel() =

L2

S.Code =

L1: t1 := a + b

If t1 = 0 goto L2

t2 = a * b

a := t2

goto L1

L2:

Cài đặt lệnh 3 địa chỉ → Biểu diễn bộ nhớ

- Sử dụng cấu trúc gồm 4 trường: Op, Arg1, Arg2, Result
 - Op: Chứa mã nội bộ của toán tử
 - Các trường Arg1, Arg2, Result trả tới các ô trong bảng ký hiệu, ứng với các tên tương ứng
- Câu lệnh dạng **a:= b Op c**
 - Đặt b vào Arg1, c vào Arg2 và a vào Result
- Câu lệnh một ngôi: **a:= b;** **a:=-b**
 - Không sử dụng Arg2

Cài đặt lệnh 3 địa chỉ → Biểu diễn bộ bốn

Ví dụ lệnh $a = -b * (c+d)$

Lệnh 3 địa chỉ

$t1 := -b$

$t2 := c + d;$

$t3 := t1 * t2;$

$a := t3$

Biểu diễn bởi dãy các bộ 4

	Op	Arg1	Arg2	Result
0	uminus	b		t1
1	+	c	d	t2
2	*	t1	t2	t3
3	:=	t3		a

Các tên tạm phải được đưa vào bảng ký hiệu

Cài đặt lệnh 3 địa chỉ → Biểu diễn bộ ba

- Mục đích để trách đưa tên tạm vào bảng ký hiệu
- Tham khảo tới giá trị tạm thời bằng vị trí lệnh sử dụng tính ra giá trị này
- Bỏ trường **Result**, Các trường **Arg1**, **Arg2** trả tới phần tử tương ứng trong bảng ký hiệu hoặc câu lệnh tương ứng

	Op	Arg1	Arg2
0	uminus	b	
1	+	c	d
2	*	(0)	(2)
3	:=	a	(2)

Sinh mã cho khai báo

- Sử dụng biến toàn cục offset
 - Trước khi bắt đầu khai báo: offset = 0
 - Với mỗi khai báo biến sẽ đưa tên đối tượng, kiểu và giá trị của offset vào bảng ký hiệu
 - Tăng offset lên bằng kích thước của dữ liệu
- Các tên trong chương trình con được truy xuất thông qua địa chỉ tương đối offset
 - Khi gấp tên đối tượng (biến), dựa vào trường offset để biết vị trí trong vùng dữ liệu

Sinh mã cho khai báo

Sản xuất	Quy tắc ngũ nghĩa
$P \rightarrow MD$	$\{\}$
$M \rightarrow \epsilon$	$\{Offset = 0\}$
$D \rightarrow D ; D$	
$D \rightarrow Id : T$	$enter(id.name, T.type, offset)$ $Offset = Offset + T.Width$
$T \rightarrow interger$	$T.type = Interger; T.width = 2$
$T \rightarrow real$	$T.type = real; T.width = 4$
$T \rightarrow array[num] of T_1$	$T.type = array(1..num.val, T_1.type)$ $T.width = num.val * T_1.width$

Hàm **Enter(name, type, offset)** thêm một đối tượng vào bảng ký hiệu với tên (name), kiểu(type) và địa chỉ tương đối (offset) của vùng dữ liệu của nó.

Sinh mã cho khai báo → Ví dụ

A: Integer;
 B: Integer;
 C: Integer;

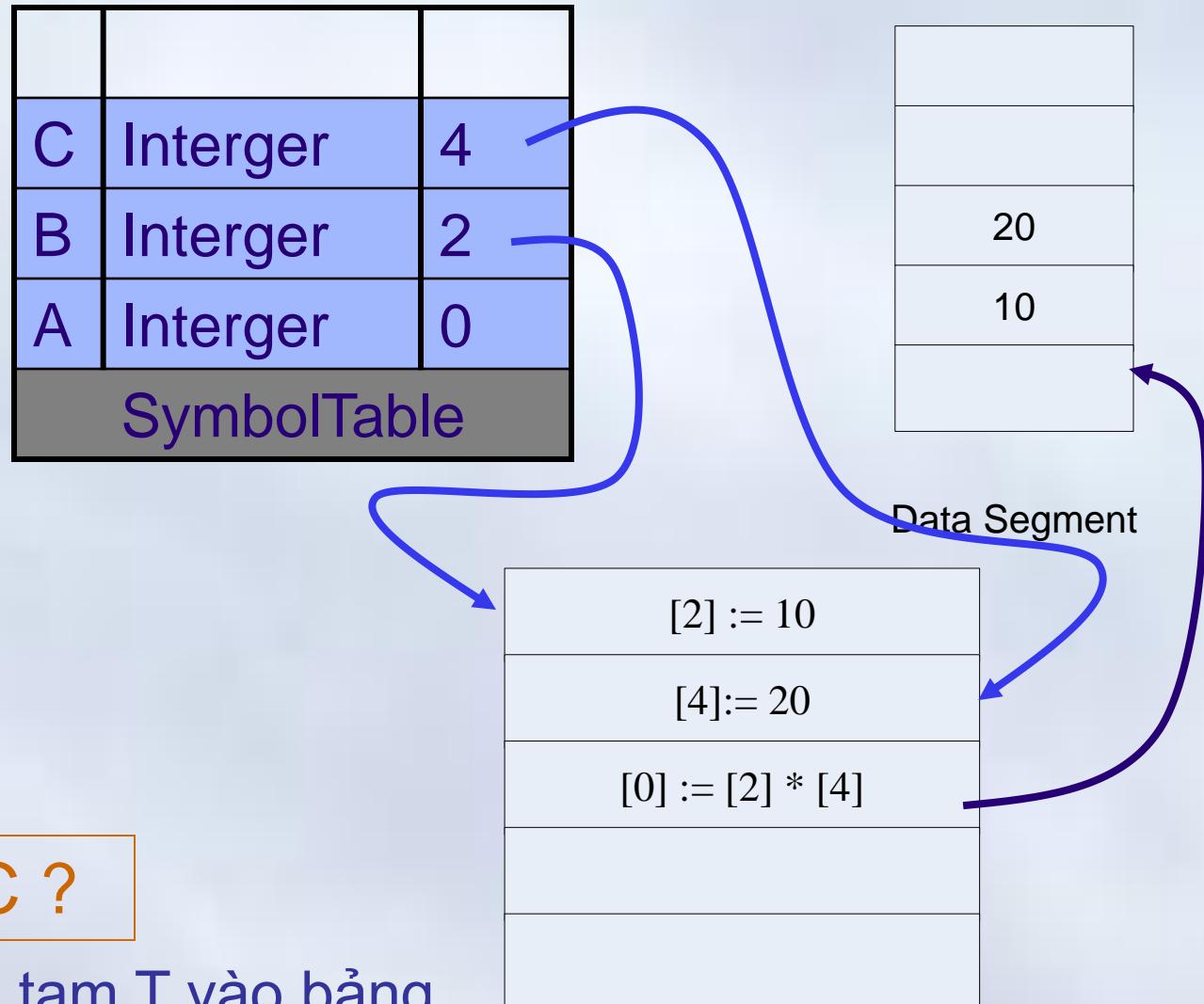
B := 10

C := 20

A := B * C

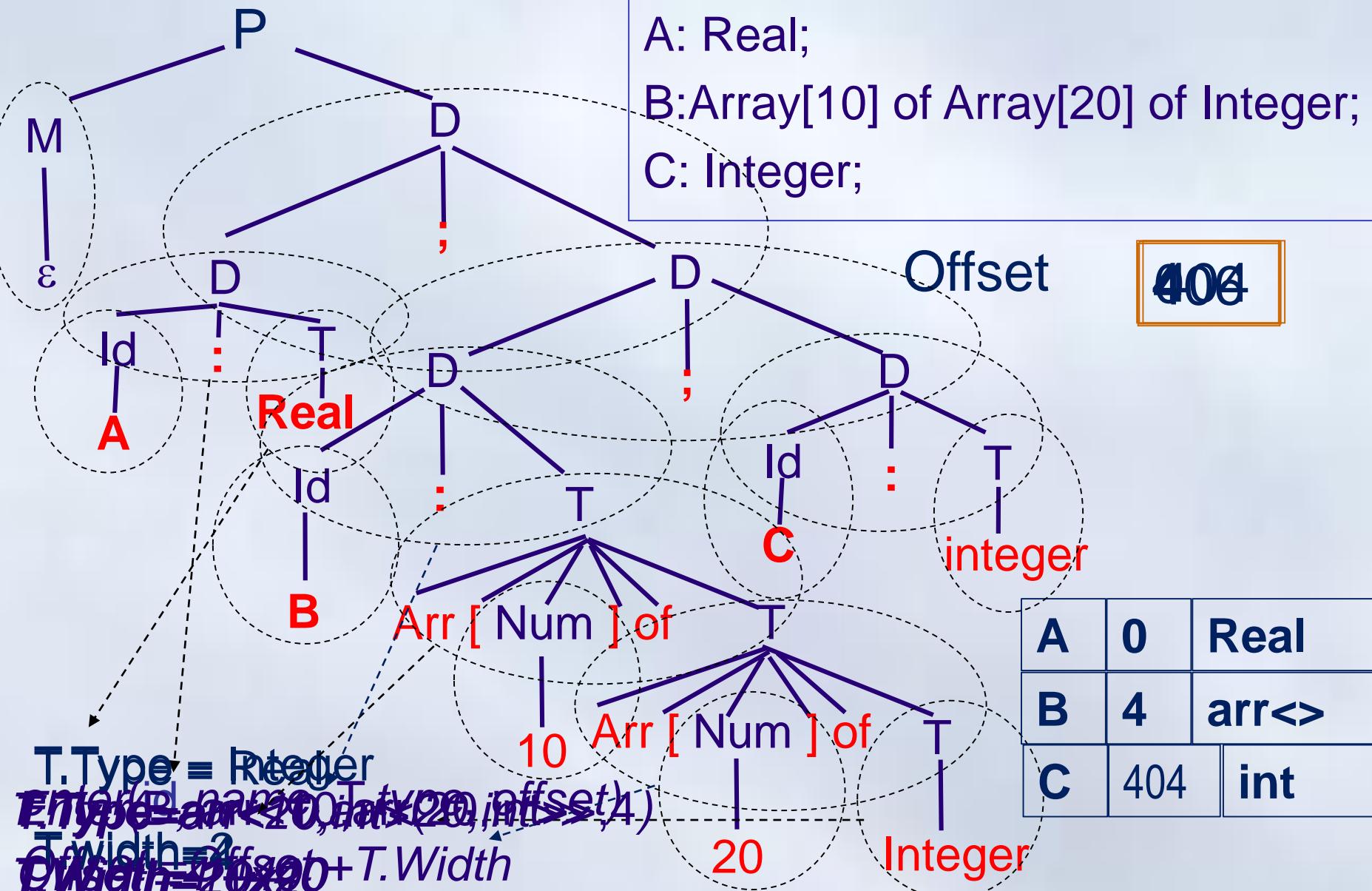
A := A * B + C ?

Thêm biến tạm T vào bảng
 ký hiệu, T có offset là : 6



Code Segment

Sinh mã cho khai báo → Ví dụ



Lưu trữ thông tin về phạm vi

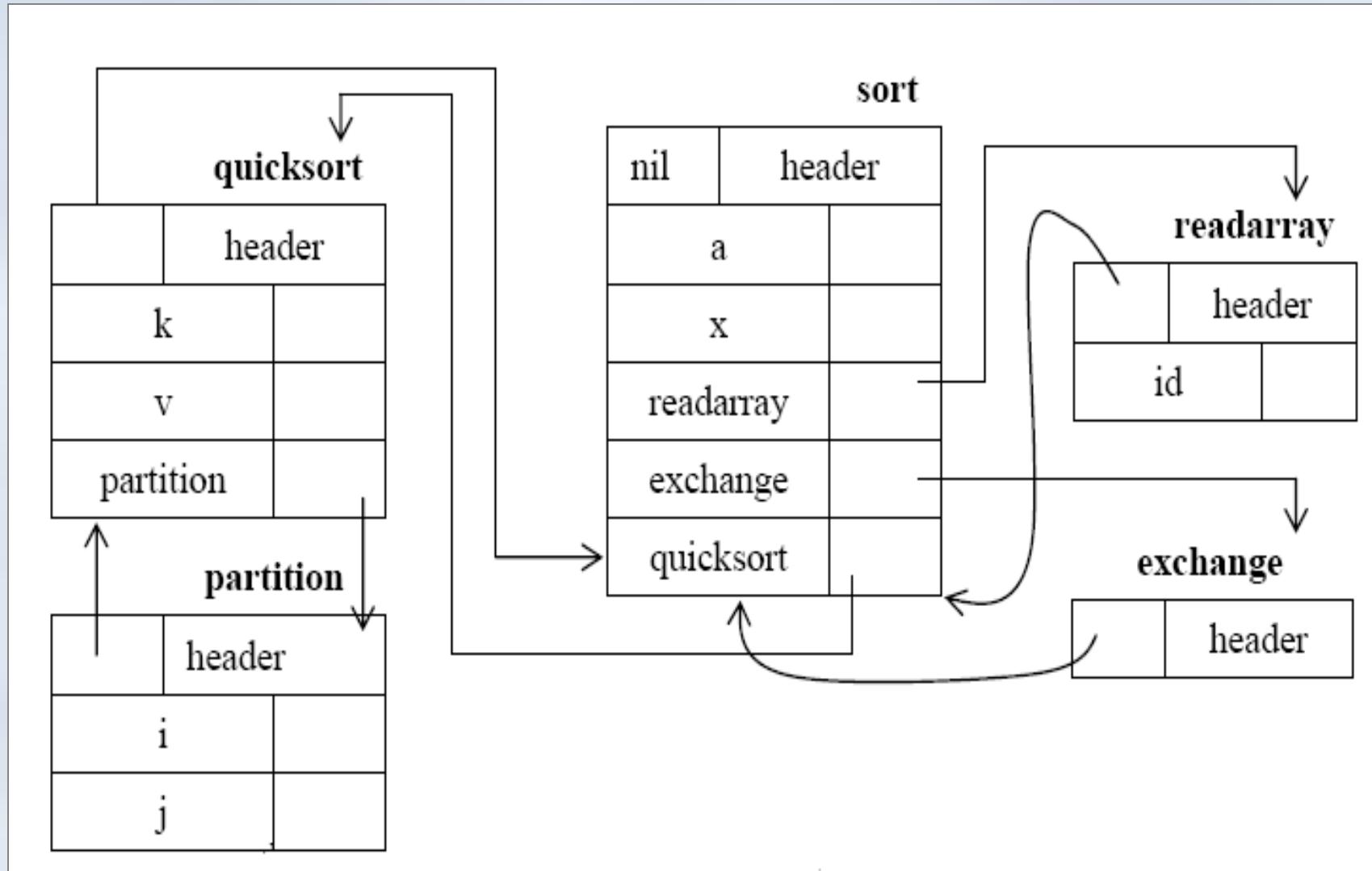
- Văn phạm cho phép các chương trình con bao nhau
 - Khi bắt đầu phân tích chương trình con, phần khai báo của chương trình bao tạm dừng
 - Dùng một bảng ký hiệu riêng cho mỗi chương trình con
- Văn phạm của khai báo này:
$$P \rightarrow D$$
$$D \rightarrow D; D \mid id : T \mid proc \ id \ ; \ D \ ; \ S$$
- Khi khai báo chương trình con $D \rightarrow proc \ id \ D1; S$ được phân tích thì các khai báo trong $D1$ được lưu trong bảng ký hiệu mới.

Lưu trữ thông tin về phạm vi → Ví dụ

```
1) Program sort; //Chương trình Quicksort
2) Var a: array[0..10] of integer;
3)       x: integer;
4) Procedure readarray;
5)       Var i: integer;
6)       Begin ..... end {readarray};
7) Procedure exchange(i, j: integer);
8)       Begin {exchange} end;
9) Procedure quicksort(m, n: integer);
10)       Var k, v: integer;
11) Function partition(y,z: integer): integer;
12)       Var i, j : integer;
13)       Begin .....exchange(i,j) end; {partition}
14)       Begin ... end; {quicksort}
15) Begin ... end; {sort}
```

Lưu trữ thông tin về phạm vi → Ví dụ

- Các bảng ký hiệu của chương trình sort



Quy tắc ngũ nghĩa → Các thao tác

- **mktable(previous)** – tạo một bảng kí hiệu mới, bảng này có *previous* chỉ đến bảng cha của nó.
- **enter(table, name, type, offset)** – thêm một đối tượng mới có tên *name* vào bảng kí hiệu được chỉ ra bởi *table* và đặt kiểu là *type* và địa chỉ tương đối là *offset* vào các trường tương ứng.
- **enterproc(table, name, newbtable)** – tạo một phần tử mới trong bảng *table* cho chương trình con *name*, *newbtable* trả tới bảng kí hiệu của CTC này.
- **addwidth(table, width)** – ghi tổng kích thước của tất cả các p/tử trong bảng kí hiệu vào header của bảng.

Khai báo trong chương trình con

Sản xuất	Quy tắc ngữ nghĩa
$P \rightarrow MD$	$addwidth(top(tblptr), top(offset)); pop(tblptr);$ $pop(offset)$
$M \rightarrow \epsilon$	$t := mkttable(null); push(t, tblptr);$ $push(0, offset)$
$D \rightarrow D ; D$	
$D \rightarrow \text{proc } id;$ $ND_1; S$	$t := top(tblptr); addwidth(t, top(offset));$ $pop(tblptr); pop(offset);$ $enterproc(top(tblptr), id.name, t)$
$N \rightarrow \epsilon$	$t := mkttable(top(tblptr)); push(t, tblptr);$ $push(0, offset);$
$D \rightarrow id : T$	$enter(top(tblptr), id.name, T.type, top(offset));$ $top(offset) := top(offset) + T.width$

Tblptr: là Stack dùng chứa các con trỏ trả về tới bảng ký hiệu

Offset: Là Stack dùng lưu trữ các Offset

Xử lý các khai báo trong chương trình con

- Sản xuất: **P→MD** :
 - Hoạt động của cây con M được thực hiện trước
- Sản xuất: **M → ε**:
 - Tạo bảng ký hiệu cho phạm vi ngoài cùng (*chương trình sort*) bằng lệnh **mktable(nil)** //Không có SymTab cha
 - Khởi tạo stack **tblptr** với bảng ký hiệu vừa tạo ra
 - Đặt offset = 0.
- Sản xuất: **N → ε**:
 - Tạo ra một bảng mới **mktable(top(tblptr))**
 - Tham số **top(tblptr)** cho giá trị con trả tới bảng cha
 - Thêm bảng mới vào đỉnh stack **tblptr** //push(t,tblptr)
 - 0 được đẩy vào stack **offset** //push(0,Offset)

N đóng vai trò tương tự M khi một khai báo CTC xuất hiện

Xử lý các khai báo trong chương trình con

- Với mỗi khai báo **id: T**
 - một phần tử mới được tạo ra cho id trong bảng kí hiệu hiện hành (*top(tblptr)*)
 - Stack *tblptr* không đổi,
 - Giá trị **top(offset)** được tăng lên bởi *T.width*.
- Khi **D → proc id ; N D₁ ; S** diễn ra
 - Kích thước của tất cả các đối tượng dữ liệu khai báo trong *D₁* sẽ nằm trên đỉnh stack offset.
 - Kích thước này được lưu trữ bằng cách dùng *Addwidth()*,
 - Các stack *tblptr* và offset bị lấy mất phần tử trên cùng (*pop()*)
 - Thao tác thực hiện trên các khai báo của chương trình con.

Xử lý các khai báo trong chương trình con

X:int;

M:Array[10] of int;

Proc A;

X:int;

N : int;

Proc B;

X:int;

S_B

S_A

S

Sinh mã cho lệnh gán → Các hàm

- **Hàm lookup()**
 - Tìm trong bảng kí hiệu xem một tên (*id.name*) đã tồn tại
 - Tìm trong bảng ký hiệu hiện thời (`top(tblptr)`)
 - Nếu không có, tìm trong các bảng ký mức cha (*con trỏ trong phần header của bảng ký hiệu*)
 - Nếu tồn tại, trả về con trỏ tới vị trí; ngược lại, trả về nil.
 - **Vị trí Id:** Vị trí trong bảng và độ lệch mức
- **Thủ tục emit()**
 - Ghi mã 3 địa chỉ vào một tập tin output
 - `gen()` xây dựng thuộc tính code cho các kí hiệu chưa kết thúc
 - Khi thuộc tính code của kí hiệu không kết thúc trong về trái sản xuất được tạo ra bằng cách nối thuộc tính code của kí hiệu không kết thúc trong về phải theo đúng thứ tự xuất hiện, sẽ ghi ra tập tin bên ngoài

Sinh mã cho lệnh gán

Sản xuất	Quy tắc ngữ nghĩa
$S \rightarrow \text{Id} := E$	$p := \text{lookup}(\text{id.name})$ if $p \neq \text{nil}$ then $\text{emit}(p := E.\text{place})$ else $\text{error}()$
$E \rightarrow E_1 + E_2$	$E.\text{Place} = \text{newTemp}()$ $\text{emit}(E.\text{place} := E_1.\text{place} + E_2.\text{place})$
$E \rightarrow E_1 * E_2$	$E.\text{Place} = \text{newTemp}()$ $\text{emit}(E.\text{place} := E_1.\text{place} * E_2.\text{place})$
$E \rightarrow -E_1$	$E.\text{place} = \text{newtemp}();$ $\text{emit}(E.\text{place} := \text{'uminus'} E_1.\text{place})$
$E \rightarrow (E)$	$E.\text{place} = E_1.\text{place} ;$
$E \rightarrow \text{Id}$	$p := \text{lookup}(\text{id.name})$ if $p \neq \text{nil}$ then $E.\text{place} := p$ else $\text{error}()$

Địa chỉ hóa các phần tử của mảng

- Các phần tử của mảng được lưu trữ trong một khối ô nhớ kế tiếp nhau để truy xuất nhanh
- **Mảng một chiều:** nếu kích thước một phần tử là w \Rightarrow địa chỉ tương đối phần tử thứ i của mảng A là

$$A[i] = \text{base} + (i - \text{low}) * w$$

$$A[i] = i * w + (\text{base} - \text{low} * w)$$

- *Low*: cận dưới tập chỉ số. Một số ngôn ngữ, $\text{low} = 0 // 1$
- *Base*: địa chỉ tương đối của vùng nhớ cấp phát cho mảng (địa chỉ tương đối của phần tử $A[\text{low}]$)
- $c = \text{base} - \text{low} * w$ có thể được tính tại thời gian dịch và lưu trong bảng kí hiệu. Vậy $A[i] = i * w + c$

- **Mảng 2 chiều:** mảng của mảng 1 chiều

Sinh mã cho biểu thức logic

- Biểu thức logic được sinh bởi văn phạm sau:

$$E \rightarrow E \text{ or } E \mid E \text{ and } E \mid \text{not } E$$
$$\mid (E) \mid \text{id} \mid \text{relop} \mid \text{id} \mid \text{true} \mid \text{false}$$

- Trong đó:

- **Or** và **And** kết hợp trái

- Or có độ ưu tiên thấp nhất tiếp theo là And, và Not (Văn phạm trên nhập nhằng)

- Mã hóa giá trị logic true/false

- Mã hóa bằng số; đánh giá một biểu thức logic như một biểu thức số học

- Biểu diễn số 1: true, 0: false

Sinh mã cho biểu thức logic → Ví dụ

- Biểu thức **a or b and not c**
 - Mã 3 địa chỉ:
 - $t1 = \text{not } c$
 - $t2 = b \text{ and } t1$
 - $t3 = a \text{ or } t2$
- Biểu thức **a < b**
 - Tương đương lệnh **if a<b then 1 else 0.**
 - Mã 3 địa chỉ tương ứng (g/thiết lệnh bắt đầu 100)
 - 100: if a<b goto 103
 - 101: t:=0
 - 102: goto 104
 - 103: t:= 1
 - 104:

Sinh mã cho biểu thức logic: biểu diễn số

Sản xuất	Quy tắc ngữ nghĩa
$E \rightarrow E_1 \text{ or } E_2$	<code>E.Place =newTemp();</code> <code>Emit(E.place '==' E1.place 'or' E2.place)</code>
$E \rightarrow E_1 \text{ and } E_2$	<code>E.Place =newTemp();</code> <code>Emit(E.place '==' E1.place 'and' E2.place)</code>
$E \rightarrow \text{not } E_1$	<code>E.Place =newTemp();</code> <code>Emit(E.place '==' 'not' E1.place)</code>
$E \rightarrow \text{Id}_1 \text{ relop } \text{Id}_2$	<code>E.Place =newTemp();</code> <code>Emit('if' id1.place relop id2.place 'goto' nextstat+3')</code> <code>Emit(E.place '==' '0'); Emit('goto' nextstat+2);</code> <code>Emit(E.place '==' '1');</code>
$E \rightarrow \text{True}$	<code>E.Place =newTemp();</code> <code>Emit(E.place '==' '1')</code>
$E \rightarrow \text{False}$	<code>E.Place =newTemp();</code> <code>Emit(E.place '==' '0')</code>

Nextstat (*next statement*) cho biết chỉ số của câu lệnh 3 địa chỉ tiếp theo

Sinh mã cho biểu thức logic → Ví dụ

- Biểu thức $a < b \text{ AND } c > d$
 - $E \rightarrow E \text{ and } E \rightarrow \text{Id} < \text{Id} \text{ and } E \rightarrow \text{Id} < \text{Id} \text{ and } \text{Id} > \text{Id}$

100: if a < b goto 103

101: t1 := 0

102: goto 104

103: t1 := 1

104: if c > d goto 107

105: t2 := 0

106: goto 108

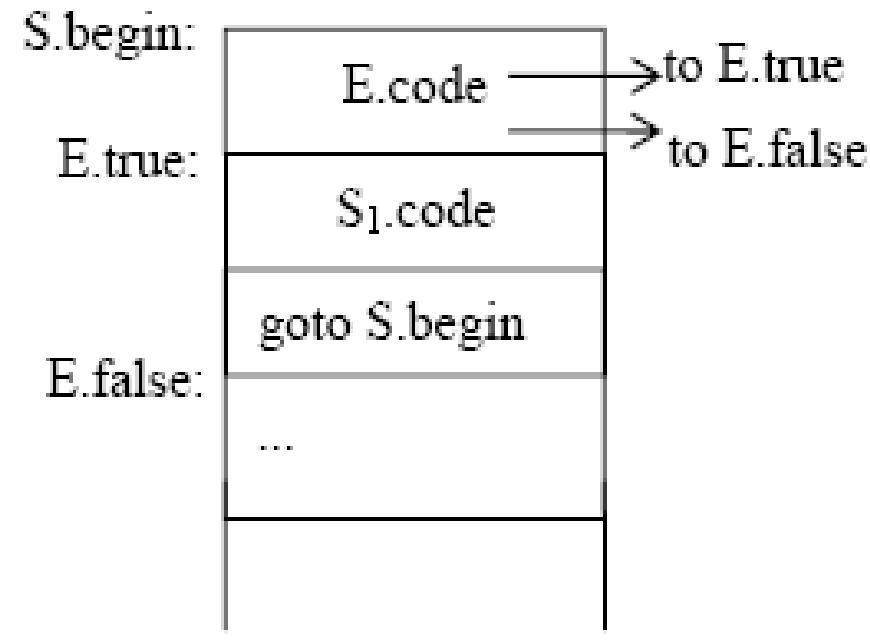
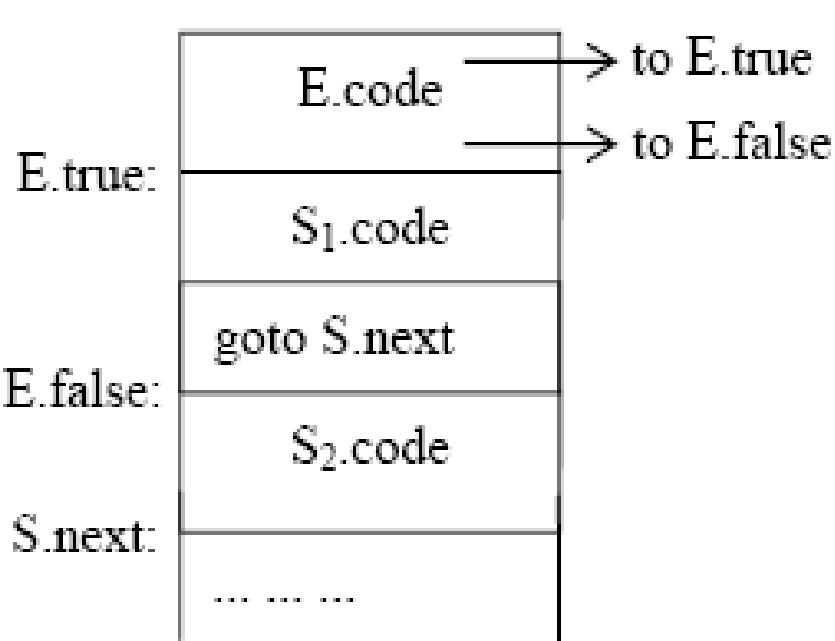
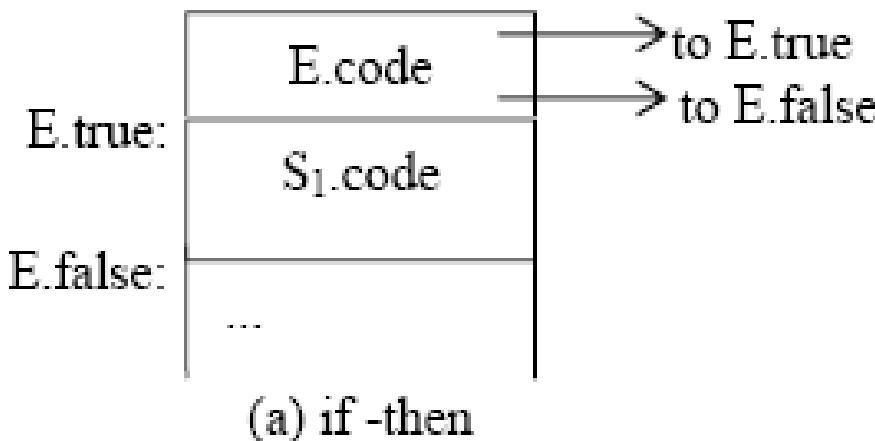
107: t2 := 1;

108: t3 := t1 and t2

Sinh mã cho các cấu trúc lập trình

- Cấu trúc: $S \rightarrow \begin{array}{l} \text{if } E \text{ then } S_1 \mid \\ \text{if } E \text{ then } S_1 \text{ else } S_2 \mid \\ \text{while } E \text{ do } S_1 \end{array}$
- **E** là biểu thức logic. E có 2 nhãn
 - **E.true**: nhãn của dòng điều khiển nếu E là true
 - **E.false**: nhãn của dòng điều khiển nếu E là false
- **S.code**:dãy lệnh 3 địa chỉ được sinh ra bởi S
- **S.next**: là nhãn địa chỉ của lệnh 3 địa chỉ đầu tiên sẽ thực hiện sau mã lệnh của cấu trúc S
- **S.begin**: nhãn địa chỉ của lệnh đầu tiên được sinh ra cho cấu trúc S

Sinh mã cho các cấu trúc lập trình



Dịch trực tiếp cú pháp cho các cấu trúc lập trình

Sản xuất	Quy tắc ngữ nghĩa
S \rightarrow if E then S ₁	$E.\text{True} = \text{newLabel}();$ $E.\text{False} = S.\text{next}; \quad S_1.\text{next} = S.\text{next}$ $S.\text{Code} = E.\text{code} \parallel \text{gen}(E.\text{true} ' : ') \parallel S_1.\text{code}$
S \rightarrow if E then S ₁ else S ₂	$E.\text{True} = \text{newLabel}(); \quad E.\text{False} = \text{newLabel}();$ $S_1.\text{next} = S.\text{next}; \quad S_2.\text{next} = S.\text{next}$ $S.\text{Code} = E.\text{code} \parallel \text{gen}(E.\text{true} ' : ') \parallel S_1.\text{code} \parallel$ $\text{gen}(\text{'goto'} \ S.\text{next}) \parallel$ $\text{gen}(E.\text{false} ' : ') \parallel S_2.\text{code}$
S \rightarrow while E do S ₁	$S.\text{Begin} = \text{newLabel}(); \quad E.\text{True} = \text{newLabel}();$ $E.\text{False} = S.\text{next}; \quad S_1.\text{next} = S.\text{Begin}$ $S.\text{Code} = \text{gen}(S.\text{begin} ' : ') \parallel E.\text{code} \parallel \text{gen}(E.\text{true} ' : ')$ $\parallel S_1.\text{code} \parallel \text{gen}(\text{'goto'} ' S.\text{Begin});$

Sinh mã cho biểu thức logic trong cấu trúc lập trình

- Nếu E có dạng: **$a < b$**
 - Mã lệnh sinh ra có dạng
If $a < b$ then goto E.true
goto E.false
- Nếu E có dạng: **$E_1 \text{ or } E_2$** thì
 - Nếu E_1 là true thì E cũng là true
 - Nếu E_1 là false thì phải đánh giá E_2 ; E sẽ là true hay false phụ thuộc E_2
- Tương tự với **$E_1 \text{ and } E_2$**
- Nếu E có dạng **$\text{not } E_1$**
 - Nếu E_1 là true, E là false và ngược lại

**E.Code sinh ra
như thế nào?**

Sinh mã cho biểu thức logic trong cấu trúc lập trình

Sản xuất	Quy tắc ngữ nghĩa
$E \rightarrow E_1 \text{ or } E_2$	$E_1.\text{true} := E.\text{true}$ $E_1.\text{false} := \text{newLabel}()$ $E_2.\text{true} := E.\text{true}$ $E_2.\text{false} := E.\text{false}$ $E.\text{Code} = E_1.\text{code} \parallel \text{gen}(E_1.\text{false} ': ') \parallel E_2.\text{code}$
$E \rightarrow E_1 \text{ and } E_2$	$E_1.\text{true} := \text{newLabel}()$ $E_1.\text{false} := E.\text{false}$ $E_2.\text{true} := E.\text{true}$ $E_2.\text{false} := E.\text{false}$ $E.\text{Code} = E_1.\text{code} \parallel \text{gen}(E_1.\text{true} ': ') \parallel E_2.\text{code}$
Chú ý: $E.\text{True}$ và $E.\text{false}$ là các thuộc tính kế thừa	

Sinh mã cho biểu thức logic trong cấu trúc lập trình

Sản xuất	Quy tắc ngữ nghĩa
$E \rightarrow \text{not } E_1$	$E_1.\text{true} := E.\text{false}$ $E_1.\text{false} := E.\text{true}$ $E.\text{Code} = E_1.\text{code}$
$E \rightarrow (E_1)$	$E_1.\text{True} := E.\text{true}$ $E_1.\text{false} := E.\text{false}$ $E.\text{Code} := E_1.\text{code}$
$E \rightarrow \text{id}_1 \text{ relop } \text{id}_2$	$E.\text{Code} := \text{gen}(\text{'if'} \text{ id}_1.\text{place} \text{ relop } \text{id}_2.\text{place} \text{ 'goto'} E.\text{true})$ $\text{gen}(\text{'goto'} E.\text{false});$
$E \rightarrow \text{True}$	$E.\text{Code} := \text{gen}(\text{'goto'} E.\text{true})$
$E \rightarrow \text{False}$	$E.\text{Code} := \text{gen}(\text{'goto'} E.\text{false})$

Sinh mã cho biểu thức logic \rightarrow Ví dụ 1
$$a < b \text{ or } c < d \text{ and } e < f$$

- Giả thiết Ltrue và Lfalse là nhãn đi đến ứng với các giá trị true và false của biểu thức
- Dựa trên quy tắc ngũ nghĩa, sinh ra

if $a < b$ goto Ltrue

goto L1:

L1: if $c < d$ goto L2

goto Lfalse

L2: if $e < f$ goto Ltrue

goto Lfalse

Sinh mã cho biểu thức logic \rightarrow Ví dụ 1

- $E \rightarrow a < b$

$E.\text{code} = \text{if } a < b \text{ goto } E.\text{true} \text{ goto } E.\text{false}$

- $E \rightarrow E_1 \text{ or } E_2$

– $E_1.\text{true} := E.\text{true} \Rightarrow E_1.\text{true} = \text{Ltrue}$

- Ltrue là nhãn đi tới nếu biểu thức là true

- Lfalse là nhãn đi tới nếu biểu thức là false

– $E_1.\text{false} := \text{L1}; // E1.\text{False} = \text{newLabel}()$

– $E_2.\text{true} := E.\text{true} = \text{Ltrue}; E_2.\text{false} := E.\text{false} = \text{Lfalse}$

– $E.\text{Code} = E_1.\text{code} \parallel \text{gen}(E_1.\text{false} ': ') \parallel E_2.\text{code}$

$\text{if } a < b \text{ goto } E_1.\text{true} \text{ goto } E_1.\text{false} \parallel E_1.\text{false} \parallel E_2.\text{code}$

if a < b goto Ltrue goto L1 L1: || E₂.code (1)

Sinh mã cho biểu thức logic \rightarrow Ví dụ 1

- $E \rightarrow c < d$ $E.\text{code} = \text{if } c < d \text{ goto } E.\text{true goto } E.\text{false}$
- $E \rightarrow e < f$ $E.\text{code} = \text{if } e < f \text{ goto } E.\text{true goto } E.\text{false}$
- $E \rightarrow E_1 \text{ and } E_2$
 - $E_1.\text{true} := L2$
 - $E_1.\text{false} := E.\text{false} = L\text{False};$
 - $E_2.\text{true} := E.\text{true} = L\text{true}; \quad E_2.\text{false} := E.\text{false} = L\text{false}$
 - $E.\text{Code} = E_1.\text{code} \parallel \text{gen}(E_1.\text{true} ':\ ') \parallel E_2.\text{code}$
 $\text{if } c < d \text{ goto } E_1.\text{true goto } E_1.\text{false} \parallel E_1.\text{true} : \parallel E_2.\text{code}$
 $\text{if } c < d \text{ goto } L2 \text{ goto } L\text{false} \ L2: \text{if } e < f \text{ goto } E_2.\text{true goto } E_2.\text{false}$
 $\text{if } c < d \text{ goto } L2 \text{ goto } L\text{false} \ L2: \text{if } e < f \text{ goto } L\text{true goto } L\text{false} \ (2)$

Sinh mã cho biểu thức logic \rightarrow Ví dụ 2

While $a \neq b$ **do**

If $a > b$ **Then**

$a := a - b$ // (S_1)

Else

$b := b - a$ // (S_2)

$S \Rightarrow \text{Id} := E \Rightarrow \text{Id} := E_1 - E_2 \Rightarrow \text{Id} := \text{Id} - \text{Id}$

S_1 .Code

$t1 := a - b$
 $a := t1$

(A)

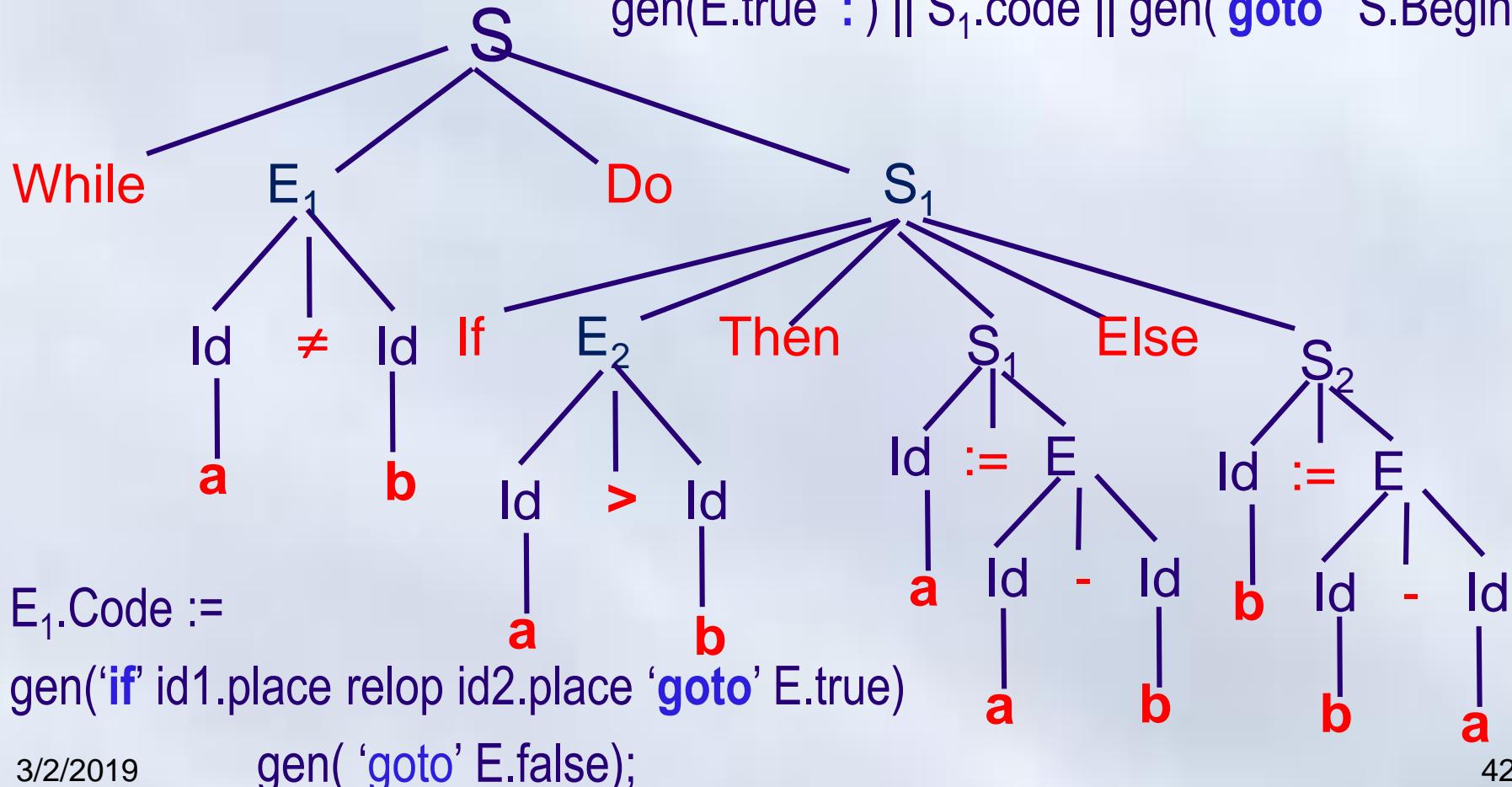
S_2 .Code

$t2 := b - a$
 $a := t2$

(B)

Sinh mã cho biểu thức logic → Ví dụ 2

S.Begin=newLabel(); E₁.True = newLabel();
 E₁.False = S.next; S₁.next = S.Begin
 S.Code = gen(S.begin ':') || E.code ||
 gen(E.true ':') || S₁.code || gen('goto' ' S.Begin);



Sinh mã cho biểu thức logic → Ví dụ 2

S → While E do S₁

S.Begin = L1 // S.Begin=newLabel();

E.True = L2 // E.True = newLabel();

E.False = Next // E.False = S.next();

S₁.next = L1 // S₁.next = S.Begin

S.Code = L1: || E.code || L2 || S₁.code || goto L1 (1)

E → a ≠ b

E.Code := if a ≠ b goto L2 goto Next (2)

E.Code:=gen('if' id1.place relop id2.place 'goto' E.true)
gen('goto' E.false);

Sinh mã cho biểu thức logic \rightarrow Ví dụ 2

$S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

$E.\text{True} = L3$ // $E.\text{True} = \text{newLabel}();$

$E.\text{False} = L4$ // $E.\text{False} = \text{newLabel}();$

$S_1.\text{next} = L1$ // $S_1.\text{next} = S.\text{next};$

$S_2.\text{next} = L1$ // $S_2.\text{next} = S.\text{next}$ (3)

$S.\text{Code} = E.\text{code} \parallel L3 : \parallel S1.\text{code} \parallel \text{goto } L1 \ L4 : \parallel S2.\text{code}$

$E \rightarrow a > b$

$E.\text{Code} := \text{if } a > b \text{ goto } L3 \text{ goto } L4$ (4)

$E.\text{Code} := \text{gen('if' } id1.\text{place relop } id2.\text{place 'goto' } E.\text{true})$
 $\text{gen('goto' } E.\text{false});$

Sinh mã cho biểu thức logic \rightarrow Ví dụ 2

1 + 2 + + 4 + A + B

L1 : **if** a \neq b **goto** L2
 goto Next

L2 : **if** a > b **goto** L3
 goto L4

L3 : t1 := a – b
 a := t1
 goto L1

L4: t2 := b – a
 a := t2
 goto L1

Next:

Biểu thức hỗn hợp

- Thực tế, các biểu thức logic thường chứa các biểu thức số học
 - $(a+b) < c$
- Xét văn phạm
 - $E \rightarrow E + E \mid E \text{ and } E \mid E \text{ relop } E \mid \text{Id}$
 $E \text{ and } E$ đòi hỏi 2 đối số phải là logic
 $E + E, E \text{ relop } E$: Các đối số là biểu thức toán học
- Để sinh mã trong trường hợp phức hợp
 - Dùng thuộc tính tổng hợp $E.\text{Type}$ cho biết kiểu là *arith* hay *logic*.

Chương 5: Sinh mã

1. Sinh mã trung gian

2. Sinh mã đích

3. Tối ưu mã

Nội dung

1. Giới thiệu
2. Môi trường thực hiện: Máy ngăn xếp
 - Bộ thông dịch cho máy ngăn xếp
3. Sinh mã đích từ mã trung gian
 - Mã trung gian là cây cú pháp
 - Mã trung gian là ký pháp Ba Lan sau
 - Mã trung gian là mã 3 địa chỉ
4. Sinh mã đích từ mã nguồn
 - Xây dựng bảng ký hiệu
 - Sinh mã cho câu lệnh

1. Giới thiệu

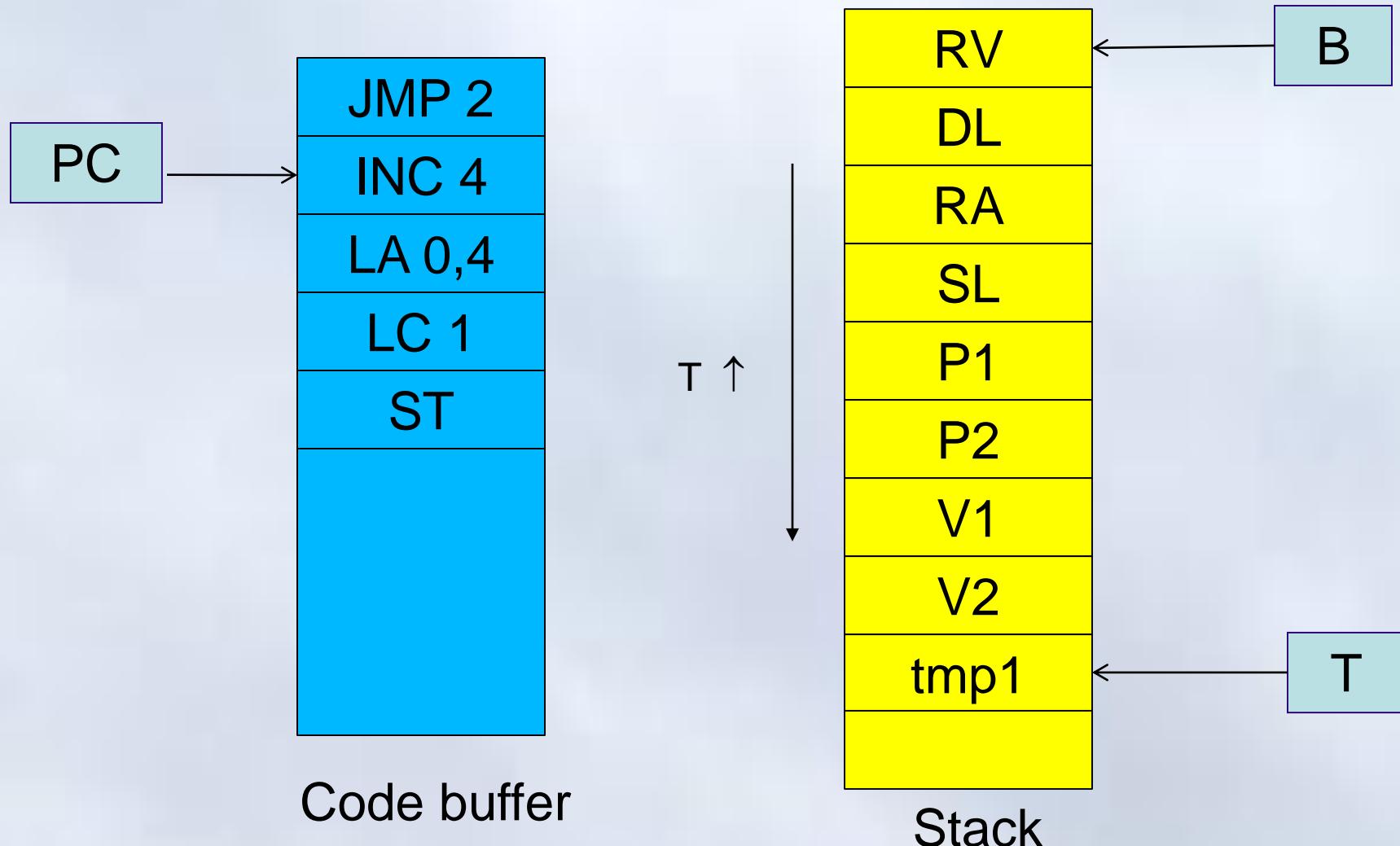
- Chương trình đích viết trên một ngôn ngữ trung gian
- Là dạng Assembly của máy giả định
 - Máy ảo (*Máy ảo làm việc với bộ nhớ stack*)
- Việc thực hiện chương trình thông qua một bộ thông dịch *interpreter*
 - *Interpreter* mô phỏng hành động của máy ảo
 - Thực hiện tập lệnh assembly của nó
- Chương trình đích được dịch từ
 - Mã trung gian
 - Mã nguồn

2. Môi trường thực hiện

- Sử dụng máy ảo là máy ngăn xếp
- Máy ngăn xếp là một hệ thống tính toán
 - Sử dụng ngăn xếp để lưu trữ các kết quả trung gian của quá trình tính toán
 - Kiến trúc đơn giản
 - Bộ lệnh đơn giản
- Máy ngăn xếp có hai vùng bộ nhớ chính
 - **Khối lệnh:**
 - Chứa mã thực thi của chương trình
 - **Ngăn xếp:**
 - Lưu trữ các kết quả trung gian

2. Máy ngăn xếp

PC, B, T là các thanh ghi của máy



2. Máy ngăn xếp → Thanh ghi

- **PC (program counter):**
 - Con trỏ lệnh trỏ tới lệnh hiện tại đang thực thi trên bộ đệm chương trình
- **B (base):**
 - Con trỏ trỏ tới địa chỉ cơ sở của **vùng nhớ cục bộ**. Các biến cục bộ được truy xuất gián tiếp qua con trỏ này
- **T (top);**
 - Con trỏ, trỏ tới đỉnh của ngăn xếp

2. Máy ngăn xếp → Bản hoạt động

- Không gian nhớ cấp phát cho mỗi chương trình con (*hàm/thủ tục/chương trình chính*) khi chúng được kích hoạt
 - Lưu giá trị tham số
 - Lưu giá trị biến cục bộ
 - Lưu các thông tin quan trọng khác:
 - RV, DL, RA, SL
- Một chương trình con có thể có nhiều bản hoạt động

2. Máy ngăn xếp → Bản hoạt động (stack frame)

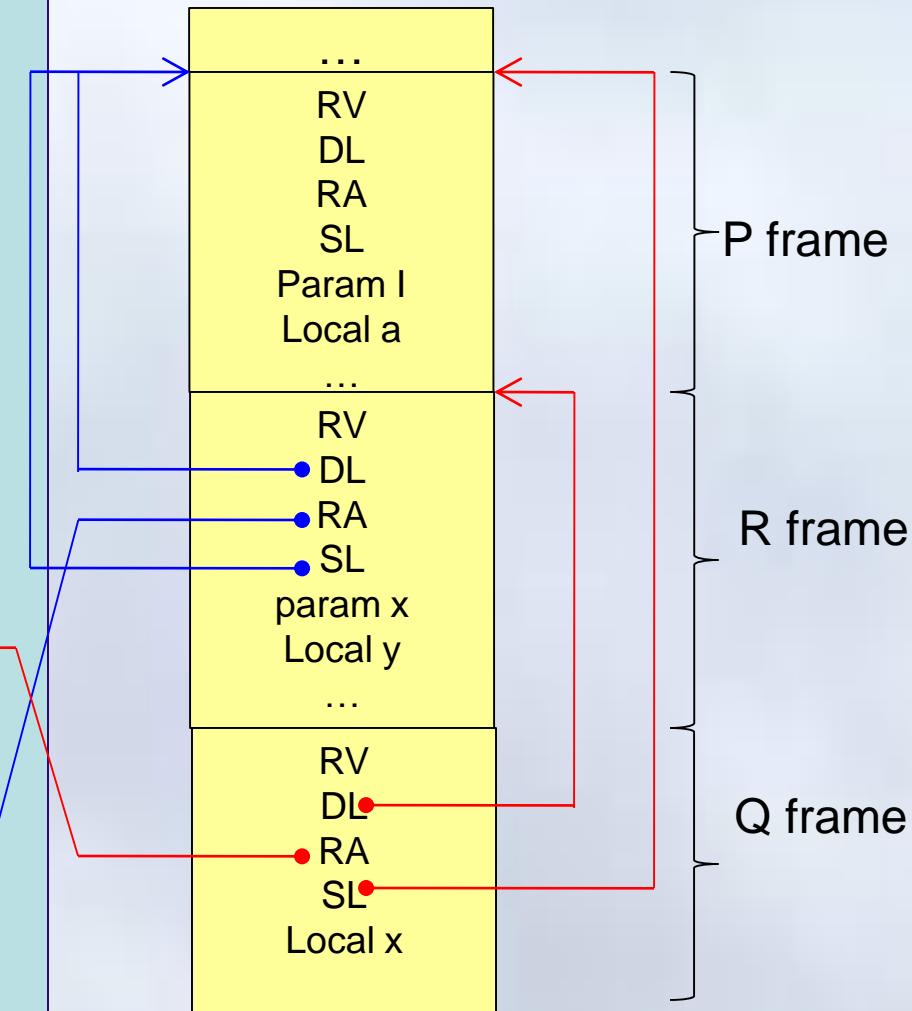
- RV (*return value*):
 - Lưu trữ giá trị trả về cho mỗi hàm
- DL (*dynamic link*):
 - Địa chỉ cơ sở của bản hoạt động của chương trình con gọi tới nó (caller).
 - Được sử dụng để hồi phục ngữ cảnh của chương trình gọi (caller) khi chương trình được gọi (called) kết thúc
- RA (*return address*):
 - Địa chỉ lệnh quay về khi kết thúc chương trình con
 - Sử dụng để tìm tới lệnh tiếp theo của caller khi called kết thúc
- SL (*static link*):
 - Địa chỉ cơ sở của bản hoạt động của chương trình con bao ngoài chương trình được gọi
 - Sử dụng để truy nhập các biến phi cục bộ

2. Máy ngăn xếp → Bản hoạt động → Ví dụ

```

Procedure P(I : integer);
  Var a : integer;
  Function Q;
    Var x : char;
    Begin
      ...
      return
    End;
  Procedure R(X: integer);
    Var y : char;
    Begin
      ...
      y = Call Q;
    End;
  Begin
    ...
    Call R(1);
  End;

```



2. Máy ngăn xếp → Lệnh

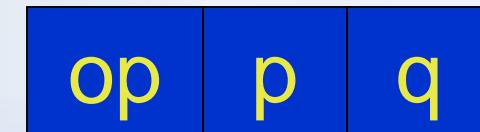
- Lệnh máy có dạng : **Op p q**
 - Op : Mã lệnh
 - p, q : Các toán hạng.
 - Các toán hạng có thể tồn tại đầy đủ, có thể chỉ có 1 toán hạng, có thể không tồn tại
 - Ví dụ

J 1 % Nhảy đến địa chỉ 1

LA 0, 4 % Nạp địa chỉ từ số 0+4 lên đỉnh stack

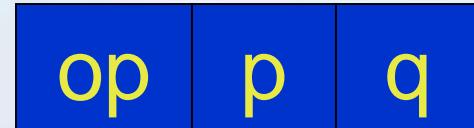
HLT %Kết thúc chương trình

2. Máy ngăn xếp → Bộ lệnh (1/5)



LA	Load Address	$t:=t+1; \ s[t]:=base(p)+q;$
LV	Load Value	$t:=t+1; \ s[t]:=s[base(p)+q];$
LC	Load Constant	$t:=t+1; \ s[t]:=q;$
LI	Load Indirect	$s[t]:=s[s[t]]; \ t:=t+1;$
INT	Increment T	$t:=t+q;$
DCT	Decrement T	$t:=t-q;$

2. Máy ngăn xếp → Bộ lệnh (2/5)



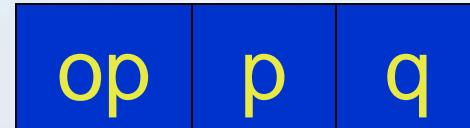
J	Jump	pc:=q;
FJ	False Jump	if s[t]=0 then pc:=q; t:=t-1;
HLT	Halt	Halt
ST	Store	s[s[t-1]]:=s[t]; t:=t-2;
CALL	Call	s[t+2]:=b; s[t+3]:=pc; s[t+4]:=base(p); b:=t+1; pc:=q;
EP	Exit Procedure	t:=b-1; pc:=s[b+2]; b:=s[b+1];
EF	Exit Function	t:=b; pc:=s[b+2]; b:=s[b+1];

2. Máy ngăn xếp → Bộ lệnh (3/5)



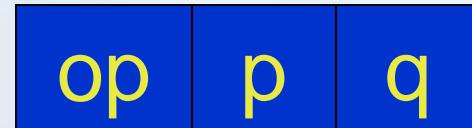
RC	Read Character	read one character into $s[s[t]]$; $t:=t-1$;
RI	Read Integer	read integer to $s[s[t]]$; $t:=t-1$;
WRC	Write Character	write one character from $s[t]$; $t:=t-1$;
WRI	Write Integer	write integer from $s[t]$; $t:=t-1$;
WLN	New Line	CR & LF

2. Máy ngăn xếp → Bộ lệnh (4/5)



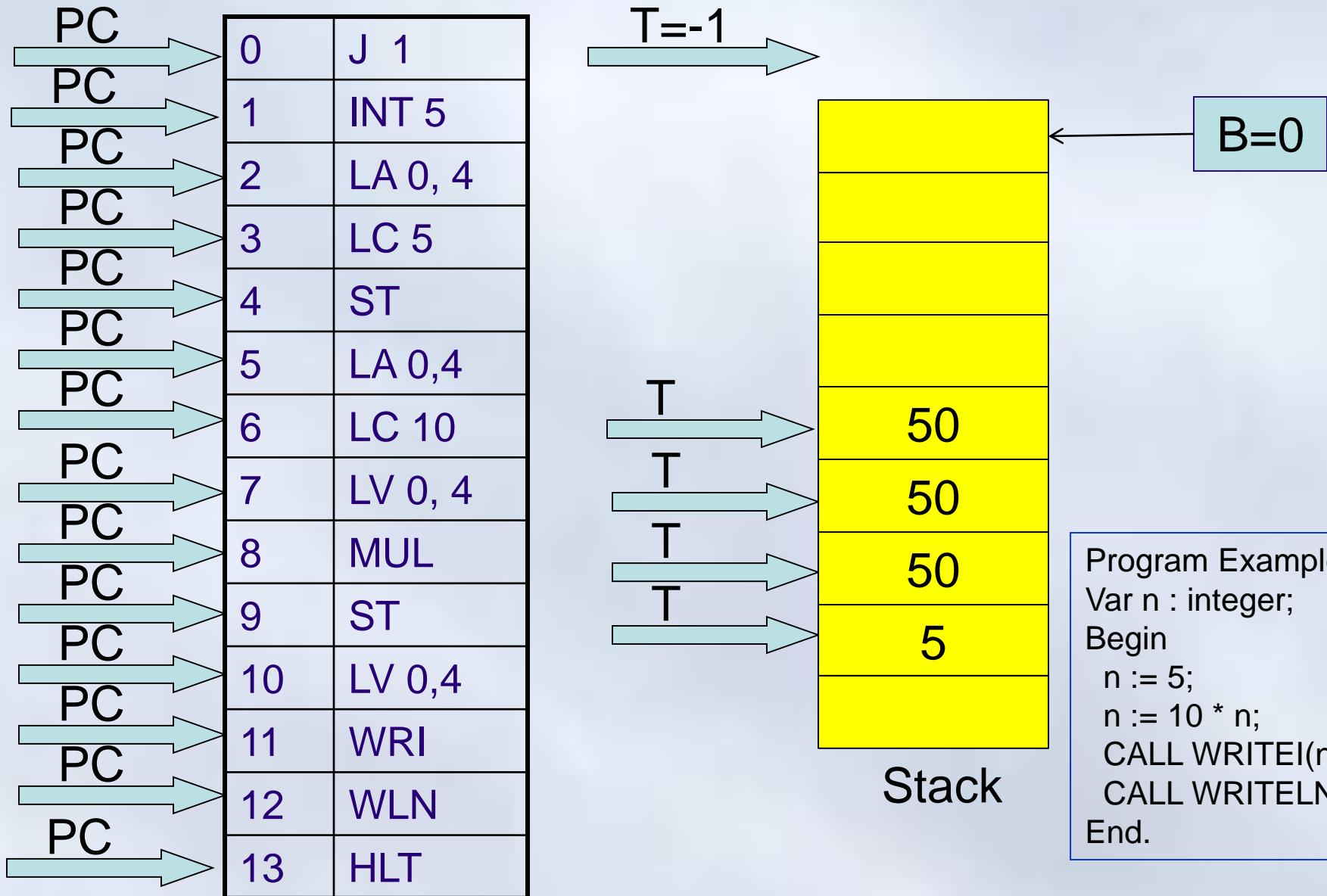
ADD	Add	$t := t - 1; s[t] := s[t] + s[t+1];$
SUB	Subtract	$t := t - 1; s[t] := s[t] - s[t+1];$
MUL	Multiply	$t := t - 1; s[t] := s[t] * s[t+1];$
DIV	Divide	$t := t - 1; s[t] := s[t] / s[t+1];$
NEG	Negative	$s[t] := -s[t];$
CV	Copy Top of Stack	$s[t+1] := s[t]; t := t + 1;$

2. Máy ngăn xếp → Bộ lệnh (5/5)



EQ	Equal	$t := t - 1; \quad \text{if } s[t] = s[t+1] \text{ then}$ $s[t] := 1 \text{ else } s[t] := 0;$
NE	Not Equal	$t := t - 1; \quad \text{if } s[t] \neq s[t+1] \text{ then}$ $s[t] := 1 \text{ else } s[t] := 0;$
GT	Greater Than	$t := t - 1; \quad \text{if } s[t] > s[t+1] \text{ then}$ $s[t] := 1 \text{ else } s[t] := 0;$
LT	Less Than	$t := t - 1; \quad \text{if } s[t] < s[t+1] \text{ then}$ $s[t] := 1 \text{ else } s[t] := 0;$
GE	Greater or Equal	$t := t - 1; \quad \text{if } s[t] \geq s[t+1] \text{ then}$ $s[t] := 1 \text{ else } s[t] := 0;$
LE	Less or Equal	$t := t - 1; \quad \text{if } s[t] \leq s[t+1] \text{ then}$ $s[t] := 1 \text{ else } s[t] := 0;$

2. Máy ngăn xếp → Ví dụ



2. Máy ngăn xếp → Xây dựng máy ảo

Mã lệnh máy

```
typedef enum {
    OP_LA,      // Load Address:
    OP_LV,      // Load Value:
    OP_LC,      // load Constant
    OP_LI,      // Load Indirect
    OP_INT,     // Increment t
    OP_DCT,     // Decrement t
    OP_J,       // Jump
    OP_FJ,      // False Jump
    OP_HL,      // Halt
    OP_ST,      // Store
    OP_CALL,    // Call
    OP_EP,      // Exit Procedure
    OP_EF,      // Exit Function
}
```

```
OP_RC,      // Read Char
OP_RI,      // Read Integer
OP_WRC,     // Write Char
OP_WRI,     // Write Int
OP_WLN,     // WriteLN
OP_ADD,     // Add
OP_SUB,     // Substract
OP_MUL,     // Multiple
OP_DIV,     // Divide
OP_NEG,     // Negative
OP_CV,      // Copy Top
OP_EQ,      // Equal
OP_NE,      // Not Equal
OP_GT,      // Greater
OP_LT,      // Less
OP_GE,      // Greater or Equal
OP_LE,      // Less or Equal
OP_BP,      // Break point.
} OpCode;
```

2. Máy ngăn xếp → Xây dựng máy ảo

Cấu trúc một lệnh

```
typedef struct{
    OpCode Op;
    int p;
    int q;
} Instruction;
```

Cấu trúc đoạn mã lệnh

```
Instruction Code [MAX_CODE];
```

Cấu trúc đoạn dữ liệu (stack)

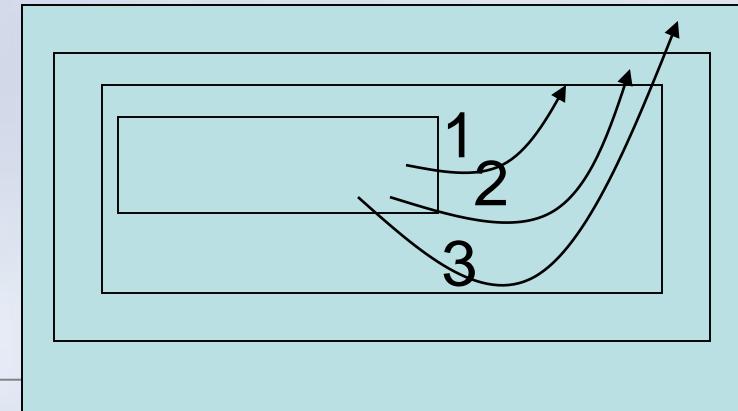
```
int Stack[MAX_DATA];
```

2. Máy ngăn xếp → Xây dựng máy ảo

Bộ thực hiện lệnh

```
void interpreter(){
    int pc = 0, t = -1, b = 0;
    do {
        switch (Code[pc].Op) {
            case OP_LA:  t++;
                Stack[t] = base(Code[pc].p) + Code[pc].q;
                break;
            case OP_LV:  t++;
                Stack[t] = Stack[base(Code[pc].p) + Code[pc].q];
                break;
            case OP_INT:
                t += Code[pc].q;
                break;
            case OP_DCT: ...
        }
    }
}
```

Hàm **base(int L)** dùng để tính địa chỉ cơ sở của chương trình con bao bên ngoài L mức



2. Máy ngăn xếp → Xây dựng máy ảo

```

.....
case OP_CALL:
    Stack[t+2] = b;           // Dynamic Link
    Stack[t+3] = pc;          // Return Address
    Stack[t+4] = base(Code[pc].p); // Static Link
    b = t + 1;                // Base & Result
    pc = Code[pc].q - 1;
    break;
case OP_J:
    pc = Code[pc].q - 1;
    break;
} //switch
pc++;
}while(Exit == 0);
}//interpreter

```

```

int base(int L) {
    int c = b;
    while (L > 0) {
        c = Stack[c + 3];
        L--;
    }
    return c;
}

```

3. Sinh mã đích từ mã trung gian

Mã trung gian là ký pháp Ba lan sau

- **Gặp các toán hạng:**
 - Sinh mã lệnh đưa giá trị toán hạng lên stack
 - Nếu toán hạng là hằng số: LC
 - Toán hạng là một địa chỉ : LV
- **Gặp các toán tử:**
 - Sinh mã của toán tử tương ứng

3. Sinh mã đích từ mã trung gian

Ví dụ: biểu thức $1 + 2 * 3 + 4 \Rightarrow 1\ 2\ 3\ * + 4 +$

Sinh mã:

LC 1

LC 2

LC 3

MUL

ADD

LC 4

AD

$(a + b) * (c + d) \Rightarrow a\ b + c\ d + *$
//a, b, c, d là các hằng số

LC a

LC b

ADD

LC c

LC d

ADD

MUL

3. Sinh mã đích từ mã trung gian

Mã trung gian là mã 3 địa chỉ

- **Nguyên tắc:**
 - Ánh xạ mỗi câu lệnh mã 3 địa chỉ bội thành dãy các lệnh mã máy tương ứng
- **Giả thiết**
 - A là một đối tượng, A.addr là địa chỉ của A trong bảng ký hiệu
- **Chú ý:**
 - Các câu lệnh gán 3 địa chỉ có thể chuyển đổi thành dạng hậu tố
 - Ví dụ: $a := b + c \Rightarrow a\ b\ c\ + :=$

3. Sinh mã đích từ mã trung gian

a := b + c

LA a.Addr

LV b.Addr //LC b

LV c.Addr //LC c

ADD

ST

a := b * c

LA a.Addr

LV b.Addr //LC b

LV c.Addr //LC c

MUL

ST

a.Addr : Địa chỉ của a trong đoạn dữ liệu (stack)

- Tính toán dựa vào trường offset trong bảng ký hiệu

3. Sinh mã đích từ mã trung gian

goto L

J f(L)

f(L): Là hàm ánh xạ từ nhãn L thành một địa chỉ

if a goto L

LV a.Addr

LC 0

EQ

FJ f(L)

if a > 10 goto L

LV a.Addr

LC 10

LE

FJ f(L)

4. Sinh mã đích trực tiếp từ mã nguồn

- Xây dựng lại bảng ký hiệu
 - Bổ sung thông tin cho biến
 - Bổ sung thông tin cho tham số
 - Bổ sung thông tin cho hàm/thủ tục/chương trình
- Sinh mã cho các câu lệnh
 - Câu lệnh gán
 - Câu lệnh rẽ nhánh
 - Câu lệnh lặp
 - Câu lệnh lấy địa chỉ
 - Gọi thủ tục

4. Sinh mã đích trực tiếp từ mã nguồn

Xây dựng lại bảng ký hiệu

- **Với biến**
 - Vị trí trên bản hoạt động của biến
- **Với tham số**
 - Vị trí trên bản hoạt động của tham số
 - Vị trí tính từ base của Frame
- **Với thủ tục**
 - Số lượng tham số

4. Sinh mã cho các lệnh → Biểu thức

Sản xuất	Ba lan sau	Quy tắc sinh mã
$E \rightarrow T\{+T\}$	$E \rightarrow T\{T+\}$	GenT() {genT() ADD}
$T \rightarrow F\{*F\}$	$F \rightarrow F\{F^*\}$	GenF() {genF() ML}
$F \rightarrow \text{Id}$	$F \rightarrow \text{Id}$	LV Id.Addr
$F \rightarrow \text{Num}$	$F \rightarrow \text{Num}$	LC num
$F \rightarrow (E)$	$F \rightarrow E$	genE()

genE(), genT(), genF(): hàm sinh mã cho ký hiệu E, T, F

- genE(), genT(): Là lời gọi tới thủ tục phân tích E, T
- genF(): Tùy thuộc vào sản xuất được sử dụng

$F \rightarrow \text{Id}[E] ?$

LA Id.Addr; LC Id.widthType; genE(); MUL; ADD; LI;

4. Sinh mã cho các lệnh → Biểu thức

```
void term(){
    factor();
```

```
    while(Token == TIMES || Token == SLASH){
```

```
        Op = Token;
```

```
        Token = getToken();
```

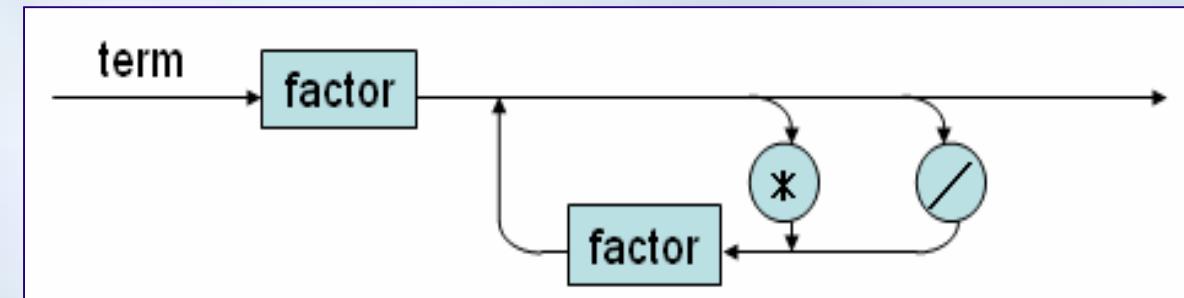
```
        factor();
```

```
        if (Op == TIMES) genCode(OP_ML);
```

```
        else genCode(OP_DV);
```

```
}
```

```
}
```



4. Sinh mã cho các lệnh → Biểu thức

$$a+10 * (b+5)$$

$$\left\{ \begin{array}{l} a.\text{Addr} = 15 \\ b.\text{Addr} = 20 \end{array} \right.$$

- LV 15
- LC 10
- LV 20
- LC 5
- ADD
- MUL
- ADD

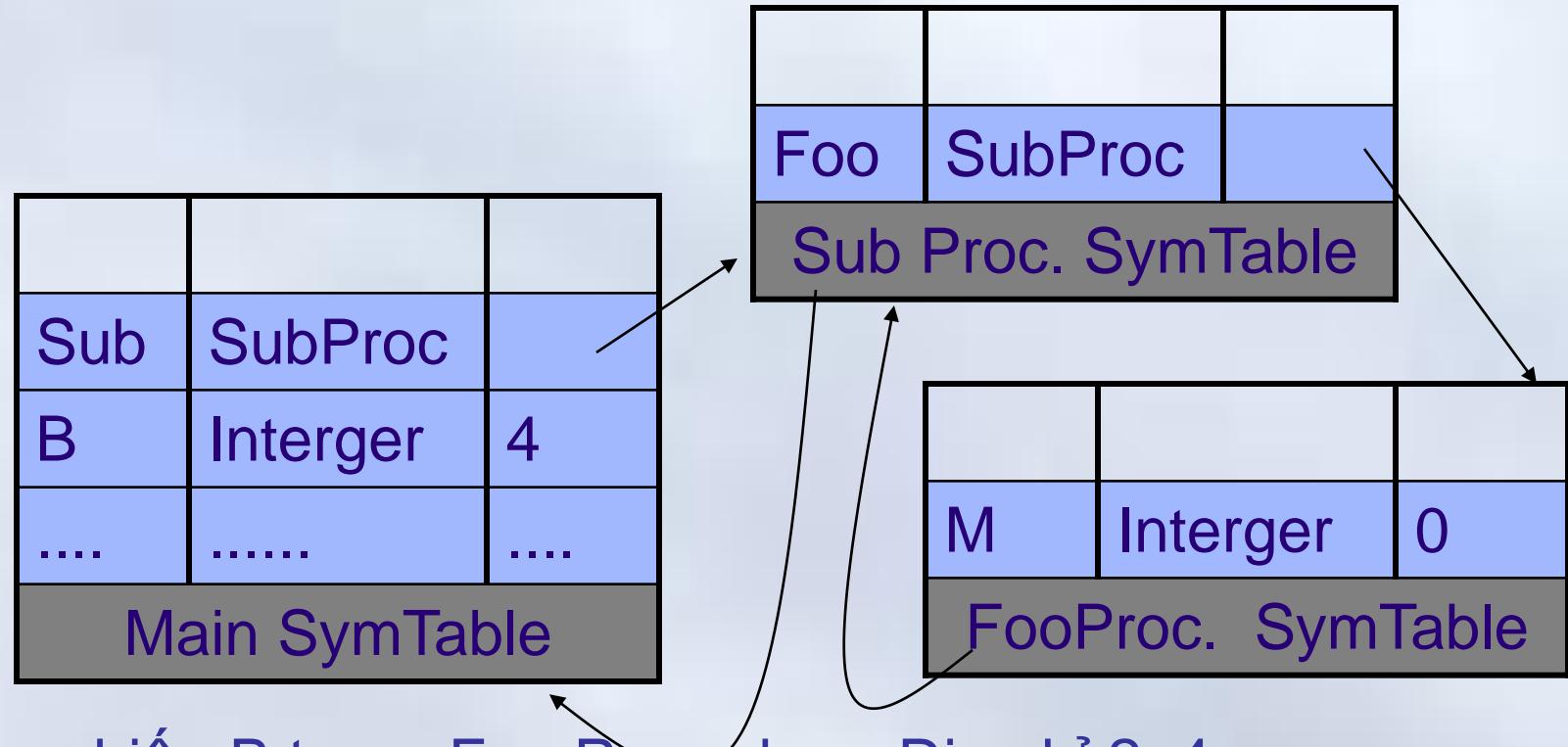
Ba lan sau

a 10 b 5 + * +

4. Sinh mã cho các lệnh → Khái niệm Lvalue

- **Lvalue**

- Danh biểu, có địa chỉ: Biến, tham biến
 - Tính toán địa chỉ theo phạm vi và vị trí trong đó



Dùng biến B trong Foo Procedure: Địa chỉ 2, 4

Dùng biến B trong Foo Procedure: Địa chỉ 1, 4

4. Sinh mã cho các lệnh → Lệnh gán

- **Lệnh gán V := Exp**

<code of Lvalue v> // đẩy địa chỉ của v lên stack

<code of exp> // đẩy giá trị của exp lên stack

ST // $S[S[t-1]] = S[t]$

Ví dụ trong thủ tục Foo: **B := M *10**

LA 2, 4

LV 0, 0

LC 10

MUL

ST

4. Sinh mã cho các lệnh → Lệnh rẽ nhánh

If <dk> Then statement

<code of dk> // đẩy giá trị điều kiện dk lên stack

FJ L //Nhảy có điều kiện, L = 0

<code of statement> //Sinh mã cho statement

L: //Cập nhật lại nhãn L bằng kích thước đoạn mã
//được sinh ra bởi statement

If <dk> Then st1 Else st2

<code of dk> // đẩy giá trị điều kiện dk lên stack

FJ L1 //L1 hiện giờ bằng 0

<code of st1>

J L2 //L2 = 0

L1: //Cập nhật lại nhãn cho L1

<code of st2>

L2: //Cập nhật lại nhãn cho L2

4. Sinh mã cho các lệnh → Lệnh lặp while

While <dk> Do statement

L1: <code of dk> // Nhãn L1 xác định

FJ L2 //genFJ(0)

<code of statement>

J L1

L2: //Cập nhật giá trị cho nhãn L2

begin = getCurrentCodeAddress() //Xác định L1

compileCondition() //sinh mã cho Condition

Jmp = genFJ(0) //Sinh ra mã lệnh nhay toi dia chi 0

compileStatement()

GenJ(begin)

Jmp.q = getCurrentCodeAddress() //sua la dia chi nhay toi

4. Sinh mã cho các lệnh → Lệnh lặp for

For v := exp1 to exp2 do statement

<code of Lvalue v> //Đặt địa chỉ v lên stack
 genCV // Sinh mã CV: nhân đôi địa chỉ của v =Copy địa chỉ V lên stack
 <code of exp1> //Đặt giá trị của biểu thức exp1 lên stack
 genST() // Sinh mã lệnh ST: lưu giá trị của exp1 vào giá trị của v

L1: //Sinh địa chỉ cho nhãn L1: getCurrentCodeAddress()
 genCV()
 genLI() // lấy giá trị của v & lưu vào đỉnh stack. Đỉnh stack là giá trị v
 <code of exp2> //Sinh mã để đặt giá trị của exp2 lên stack
 genLE () //Lệnh LE để so sánh 2 giá trị trên đỉnh stack: v và Exp2
 genFJ(L2) //sinh mã lệnh FJ, nhay tới nhãn L2 chưa xác định
 <code of statement> //Sinh mã lệnh cho phần segment
 genCV() || genCV(); //Đỉnh stack là địa chỉ của v, Sao chép 2 lần
 genLI(); //Đỉnh stack là giá trị của v
 genLC(1) || genAD(); //Đặt lên stack 1 rồi cộng với giá trị của v
 genST(); // Lưu giá trị mới của v (v+1) vào vị trí của v
 genJ(L1) //nhảy tới nhãn L1 đã tính
 L2: //Tính toán nhãn L2 và cập nhật L2 tương ứng
 DCT 1 //Giảm thanh ghi T1 đơn vị do đỉnh stack là v

Lấy địa chỉ/giá trị biến

- Khi lấy địa chỉ/giá trị một biến cần tính đến phạm vi của biến
 - Biến cục bộ được lấy từ frame hiện tại
 - Biến phi cục bộ được lấy theo các **StaticLink** với cấp độ lấy theo “độ sâu” của phạm vi hiện tại so với phạm vi hiện thời

Lấy giá trị của tham số hình thức

- Khi tính toán giá trị của **Factor**
- Cũng cần tính độ sâu như biến
 - Nếu là tham trị: giá trị của tham trị chính là giá trị cần lấy.
 - Nếu là tham biến: giá trị của tham số là địa chỉ của giá trị cần lấy.

Sinh lời gọi hàm/thủ tục

- Lời gọi
 - Thủ tục gấp trong sinh mã lệnh **CallSt**
- Trước khi sinh lời gọi thủ tục cần phải nạp giá trị cho các tham số hình thức bằng cách
 - Tăng giá trị T lên 4 (bỏ qua RV,DL,RA,SL)
 - Sinh mã cho k tham số thực tế
 - Tham trị: compileExpression()
 - Tham biến: compileLValue()
 - Giảm giá trị T đi 4 + k
 - Sinh lệnh CALL

Sinh lời gọi hàm/thủ tục → Ví dụ

Procedure Foo(VAR A, B)

CALL Foo(N, A*10)

Lvalue(Ident) trả về địa chỉ của Ident gồm

- Độ lệch frame
- Offset trong frame

INT 4
LA Lvalue(N)
LV Lvalue(A)
LC 10
MUL
DCT 6
CALL Lvalue(Foo)

Sinh mã cho lệnh CALL (p, q)

Giả sử cần sinh lệnh CALL cho hàm/thủ tục A

Lệnh CALL có hai tham số:

- p: Độ sâu của lệnh CALL, chưa static link.
Base(p) = base của frame chương trình con
chưa khai báo của A.
- q: Địa chỉ lệnh mới
q + 1 = địa chỉ đầu tiên của dãy lệnh cần thực hiện khi gọi A.

CALL (p, q)

```

s[t+2]:=b;          // Lưu lại dynamic link
s[t+3]:=pc;         // Lưu lại return address
s[t+4]:=base(p);   // Lưu lại static link
b:=t+1;            // Base mới và return value
pc:=q;              // địa chỉ lệnh mới

```

Hoạt động khi thực hiện lệnh CALL (p, q)

- Thực hiện các lệnh và stack biến đổi tương ứng.
- Khi kết thúc
 - Thủ tục (lệnh **EP**): toàn bộ frame được giải phóng, con trỏ **T** đặt lên đỉnh frame cũ.
 - Hàm (lệnh **EF**): frame được giải phóng, chỉ chứa giá trị trả về tại offset 0, con trỏ **T** đặt lên đầu frame hiện thời (offset 0).

Chương 5: Sinh mã

1. Sinh mã trung gian

2. Sinh mã đích

3. Tối ưu mã

Giới thiệu

Yêu cầu

- Chương trình sau khi tối ưu phải tương đương
- Tốc độ thực hiện trung bình tăng
- Hiệu quả đạt được tương xứng với công sức

Có thể tối ưu mã vào lúc nào

- Mã nguồn
 - Do người lập trình (giải thuật)
- Mã trung gian
- Mã đích

Tối ưu mã cục bộ

- Nguyên tắc
 - Xem xét một dãy lệnh trong mã đích và thay thế chúng bằng những đoạn mã ngắn hơn và hiệu quả hơn
- Xu hướng chính
 - Loại bỏ lệnh dư thừa
 - Thông tin dòng điều khiển
 - Loại bỏ biểu thức con chung
 - Tính toán giá trị hằng
 - Giảm chi phí tính toán
 - Lan truyền biến gán.....

Loại bỏ lệnh dư thừa

- **Mã không đến được**

goto L2

$x := x + 1$ \leftarrow Không cần

L2:....

- **Mã chết**

$x := 32$

$y := x + y$

Nếu x không được dùng trong những lệnh tiếp theo, có thể chuyển thành:

$y := 32 + y$

Thông tin dòng điều khiển

goto L1

...

L1: goto L2 ← Không cần

chuyển thành

goto L2

Loại bỏ biểu thức con chung

- Ví dụ câu lệnh $a[i+1] = b[i+1]$
- Sinh ra mã

$t1 = i+1$

$t2 = b[t1]$

$t3 = t1 + t2$

$a[t3] = t2$

Không cần

Tính giá trị hằng

 $x := 32$

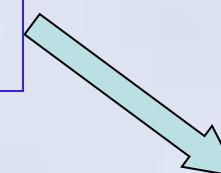
trở thành

 $x := 64$ $x := x + 32$

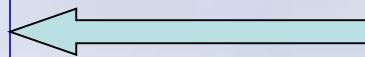
```
i = 4  
t1 = i+1  
t2 = b[t1]  
a[t1] = t2
```



```
i = 4  
t1 = 5  
t2 = b[t1]  
a[t1] = t2
```



```
i = 4  
t2 = b[5]  
a[5] = t2
```



```
i = 4  
t1 = 5  
t2 = b[5]  
a[5] = t2
```

Giảm chi phí tính toán

- **Tối ưu vòng lặp**

Chuyển những đoạn mã bất biến ra ngoài vòng lặp:

while ($i \leq \text{limit} - 2$)

\Rightarrow

$t := \text{limit} - 2$

while ($i \leq t$)

- **Tối ưu toán học**

$x := x + 0 \quad \leftarrow$ Không cần

$A := \text{sqrt}(x) \quad \rightarrow A := x * x$

$x := x * 2 \quad \rightarrow x := x + x$

$x := x * 8 \quad \rightarrow x := x \ll 3$

Lan truyền biến gán

```
tmp2 = tmp1 ;  
tmp3 = tmp2 * tmp1;      tmp3 = tmp1 * tmp1;  
tmp4 = tmp3 ;           tmp5 = tmp3 * tmp1 ;  
tmp5 = tmp3 * tmp2 ;      c = tmp3 + tmp5 ;  
c = tmp5 + tmp4 ;
```



Khối cơ bản (basic block)

- Khái niệm
 - Chuỗi các lệnh kế tiếp nhau trong đó dòng điều khiển đi vào lệnh đầu tiên của khối và ra ở lệnh cuối cùng của khối mà không bị dừng hoặc rẽ nhánh.
- Ví dụ

$t1 := a * a$

$t2 := a * b$

$t3 := 2 * t2$

$t4 := t1 + t2$

$t5 := b * b$

$t6 := t4 + t5$

Giải thuật phân chia các khối cơ bản

- **Input:**
 - Dãy lệnh ba địa chỉ.
- **Output:**
 - DS các khối cơ bản với mã ba địa chỉ của từng khối
- **Phương pháp:**
 - Xác định tập các lệnh đầu, của từng khối cơ bản
 - i. Lệnh đầu tiên của chương trình là lệnh đầu.
 - ii. Bất kỳ lệnh nào là đích nhảy đến của các lệnh GOTO có hoặc không có điều kiện là lệnh đầu
 - iii. Bất kỳ lệnh nào đi sau lệnh GOTO có hoặc không có điều kiện là lệnh đầu
 - Với mỗi lệnh đầu, khối cơ bản bao gồm nó và tất cả các lệnh tiếp theo không phải là lệnh đầu hay lệnh kết thúc chương trình

Giải thuật phân chia các khối cơ bản → Ví dụ

1. prod := 0
2. i := 1
3. t1 := 4 * i
4. t2 := a[t1]
5. t3 := 4 * i
6. t4 := b[t3]
7. t5 := t2 * t4
8. t6 := prod + t5
9. prod := t6
10. t7 := i + 1
11. i := t7
12. if i<=20 goto 3

- Lệnh (1) là lệnh đầu theo quy tắc i,
- Lệnh (3) là lệnh đầu theo quy tắc ii
- Lệnh sau lệnh (12) là lệnh đầu theo quy tắc iii.
- Các lệnh (1)và (2) tạo nên khối cơ bản thứ nhất.
- Lệnh (3) đến (12) tạo nên khối cơ bản thứ hai.

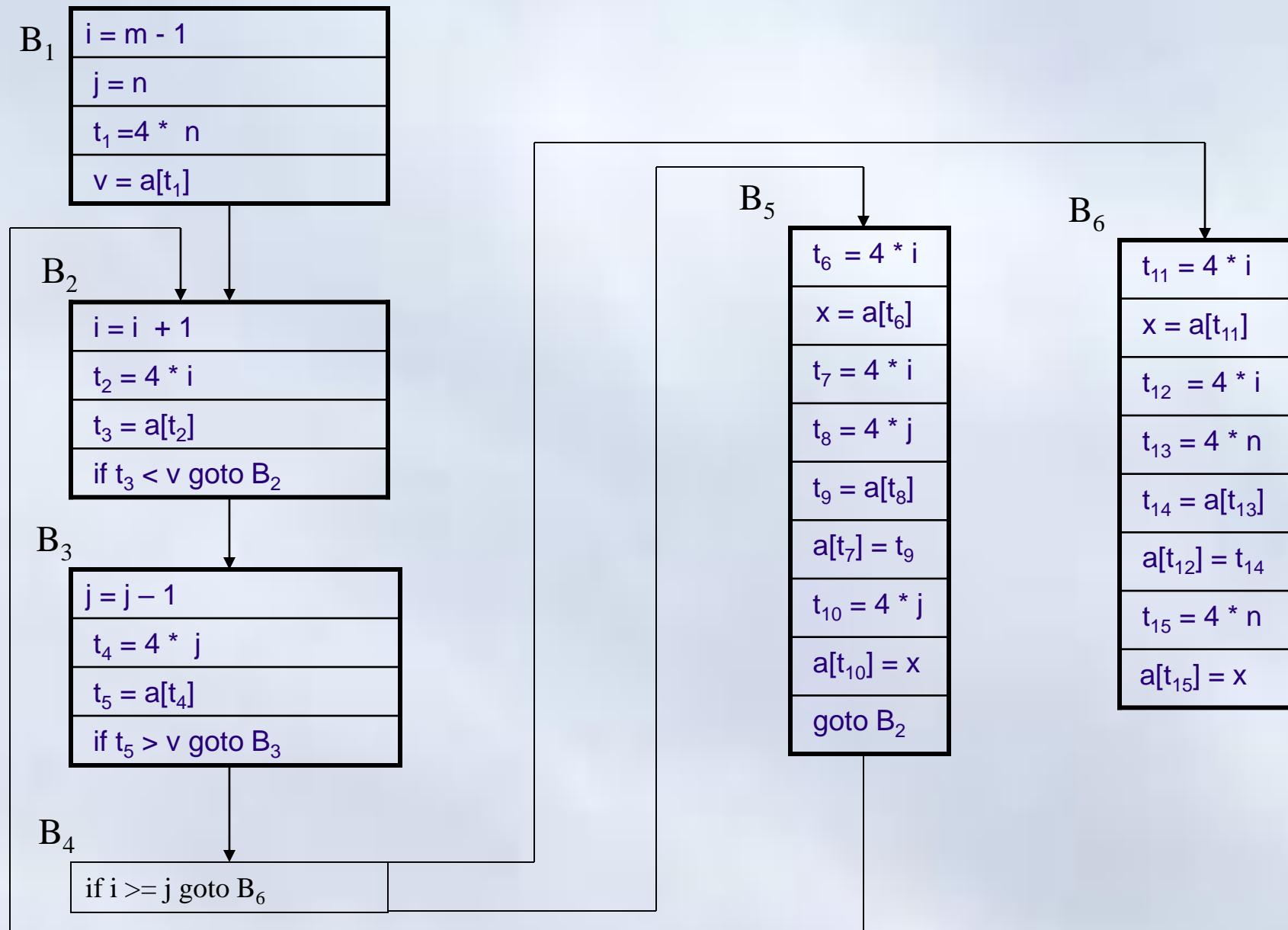
Mã ba địa chỉ của Quick Sort

1	$i = m - 1$
2	$j = n$
3	$t_1 = 4 * n$
4	$v = a[t_1]$
5	$i = i + 1$
6	$t_2 = 4 * i$
7	$t_3 = a[t_2]$
8	$\text{if } t_3 < v \text{ goto (5)}$
9	$j = j - 1$
10	$t_4 = 4 * j$
11	$t_5 = a[t_4]$
12	$\text{if } t_5 > v \text{ goto (9)}$
13	$\text{if } i \geq j \text{ goto (23)}$
14	$t_6 = 4 * i$
15	$x = a[t_6]$

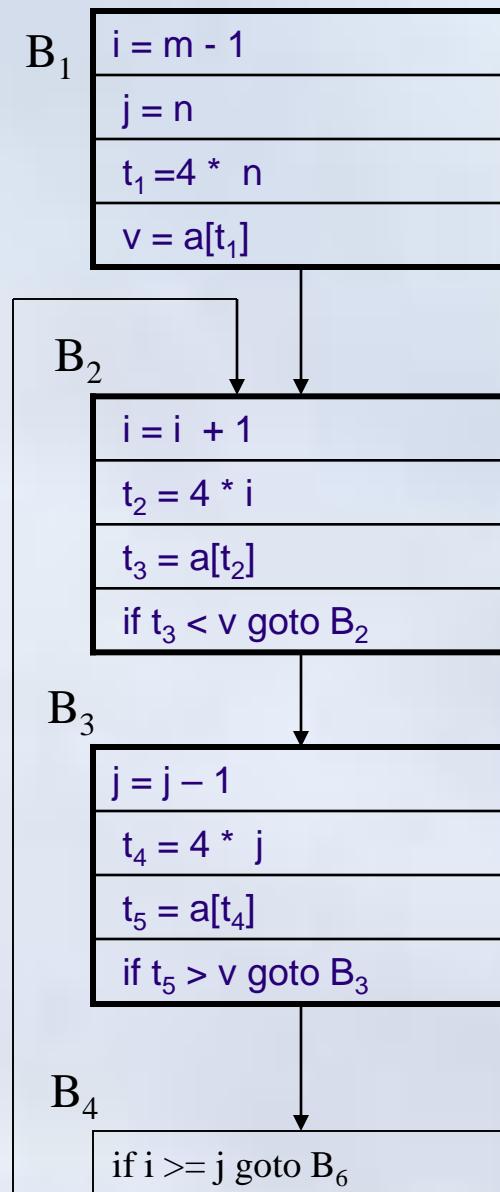
Xác
định
khối cơ
bản

16	$t_7 = 4 * i$
17	$t_8 = 4 * j$
18	$t_9 = a[t_8]$
19	$a[t_7] = t_9$
20	$t_{10} = 4 * j$
21	$a[t_{10}] = x$
22	goto (5)
23	$t_{11} = 4 * i$
24	$x = a[t_{11}]$
25	$t_{12} = 4 * i$
26	$t_{13} = 4 * n$
27	$t_{14} = a[t_{13}]$
28	$a[t_{12}] = t_{14}$
29	$t_{15} = 4 * n$
30	$a[t_{15}] = x$

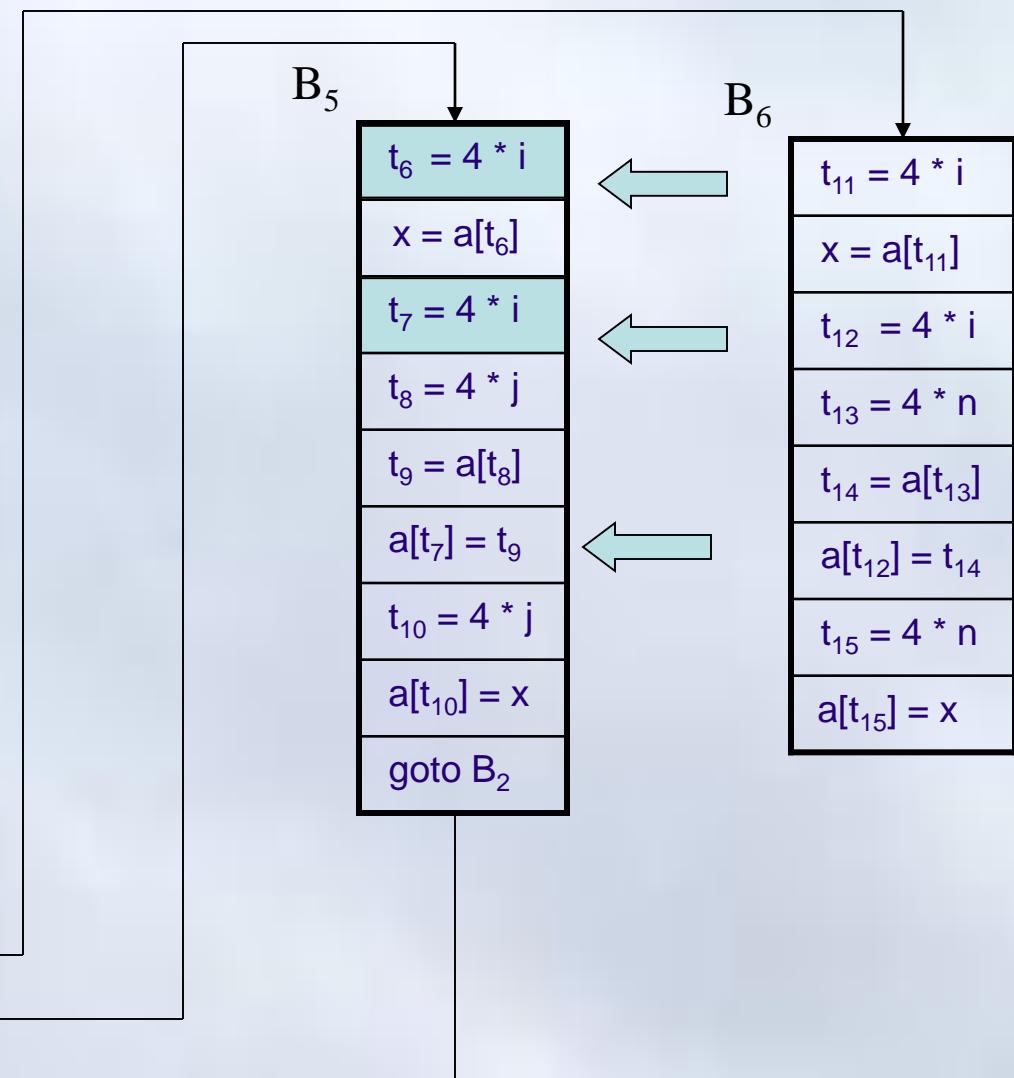
Đồ thị quan hệ các khối



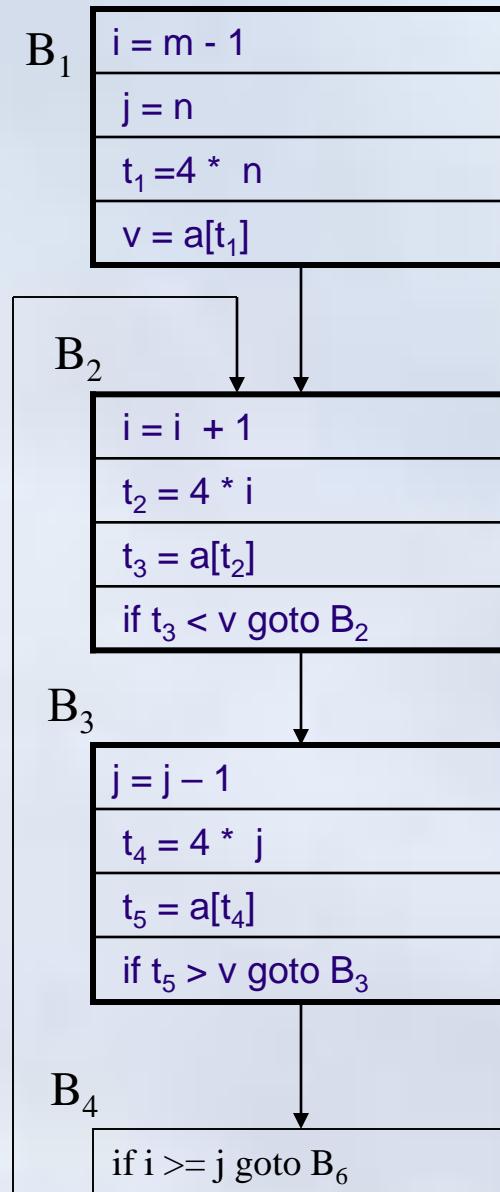
Tối ưu mã → Ví dụ



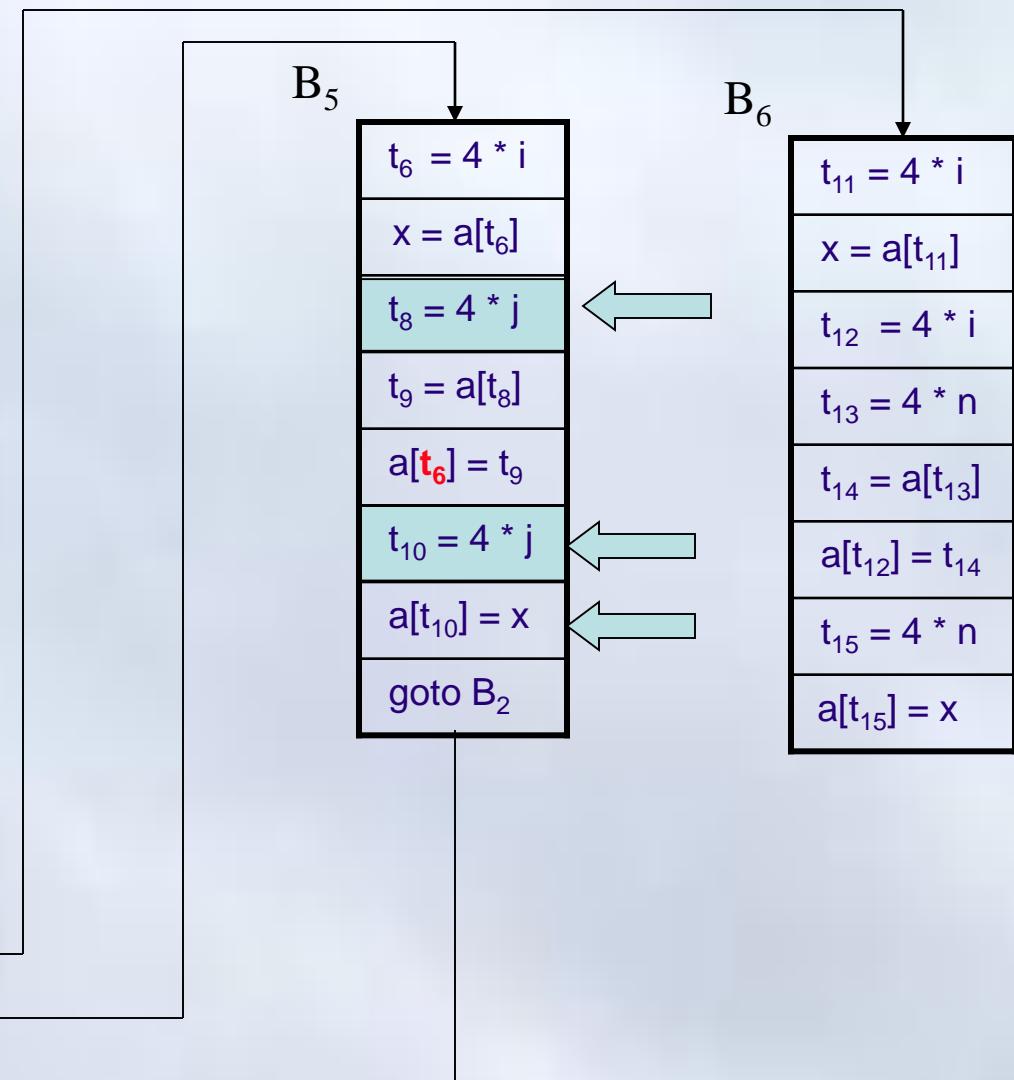
Loại biểu thức con chung



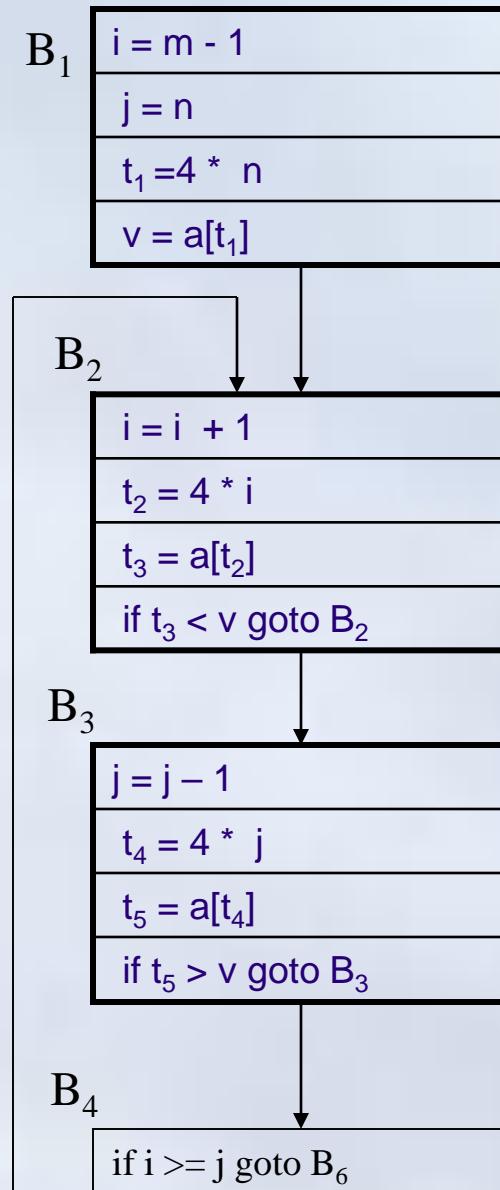
Tối ưu mã → Ví dụ



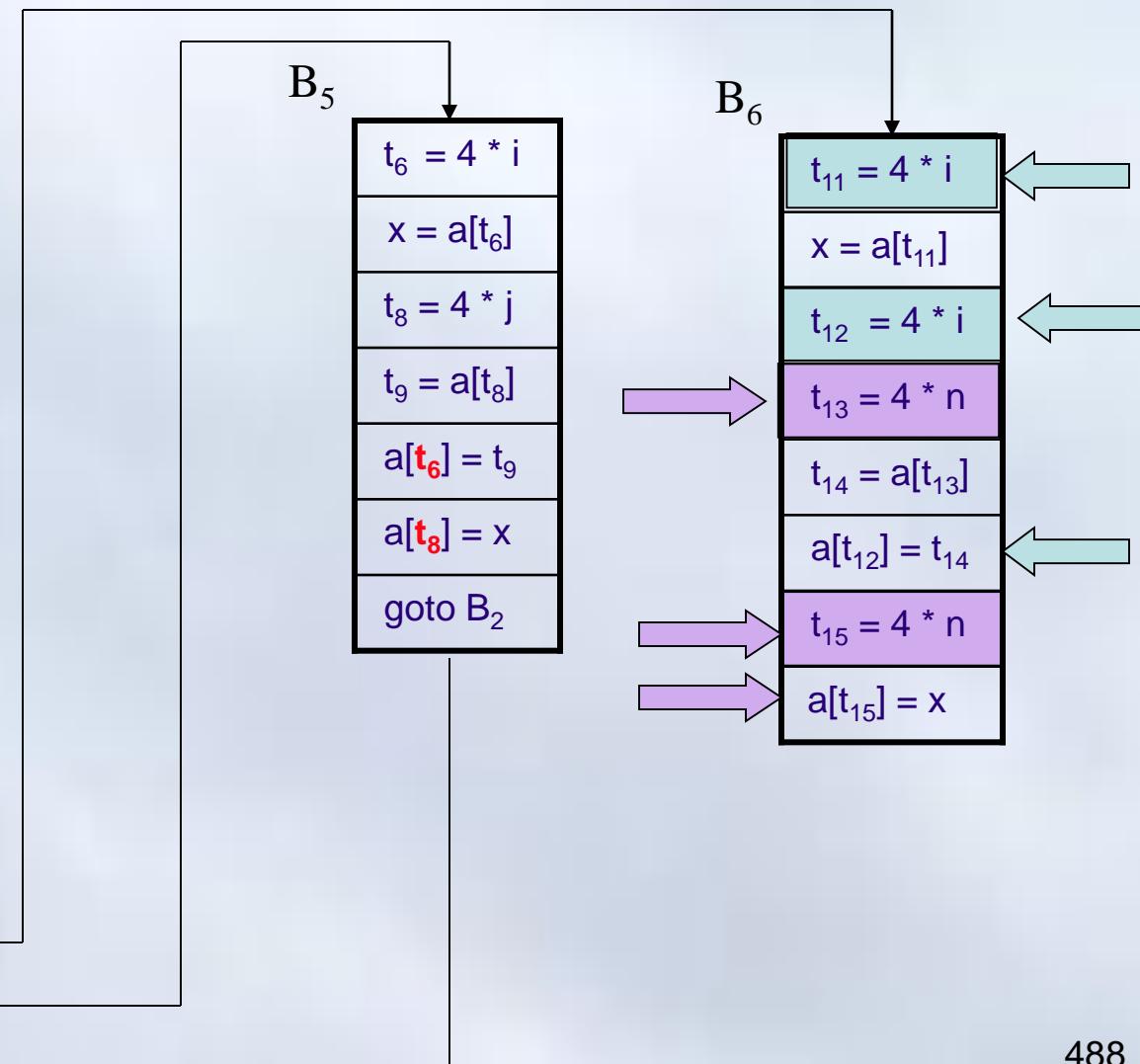
Loại biểu thức con chung



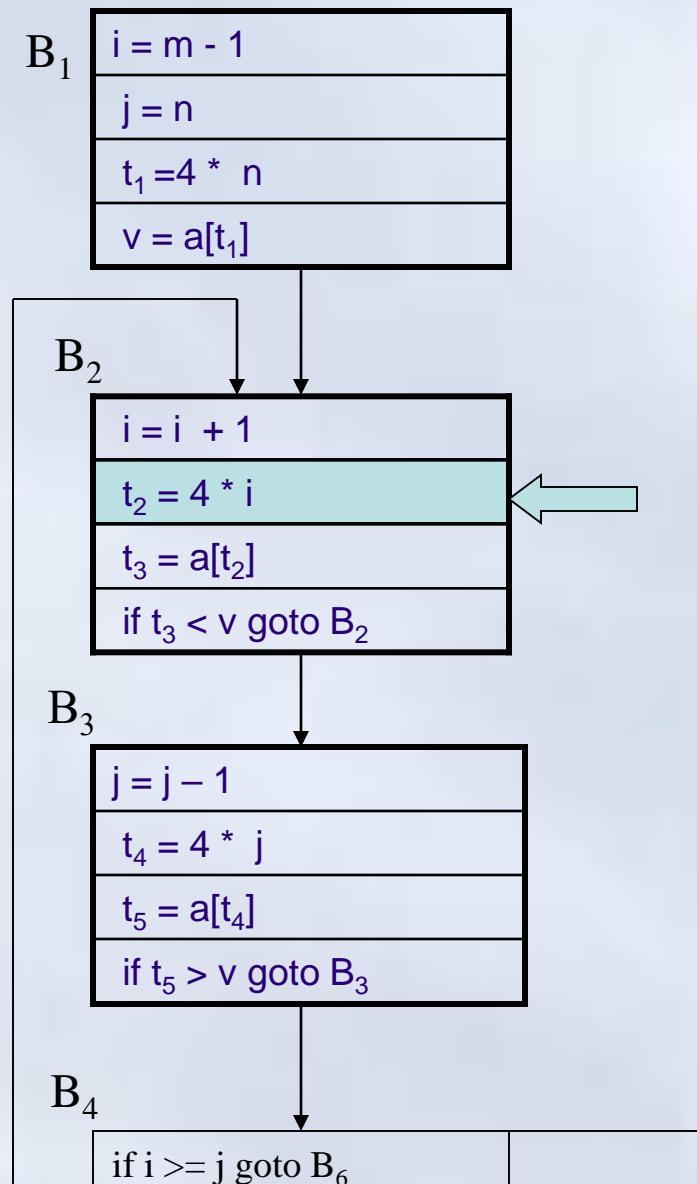
Tối ưu mã → Ví dụ



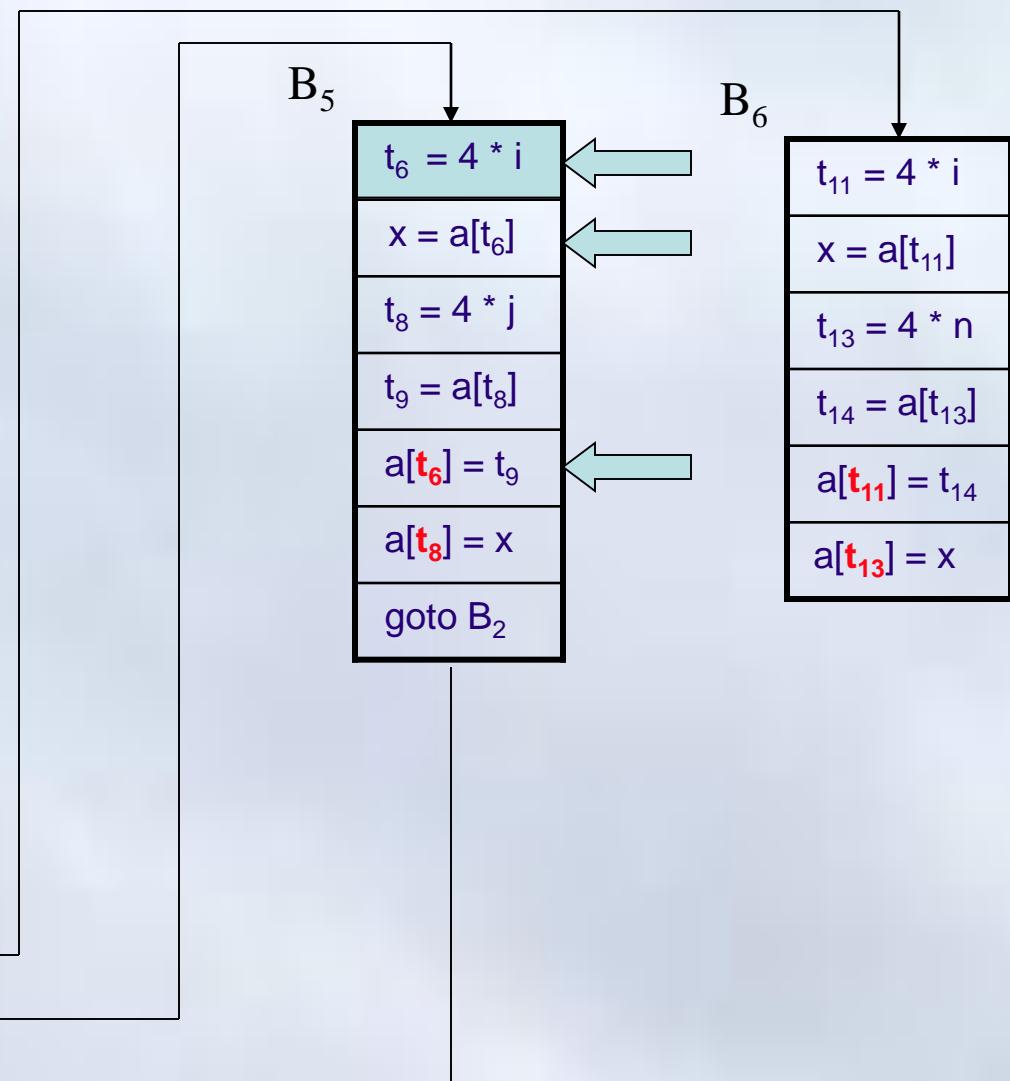
Loại biểu thức con chung



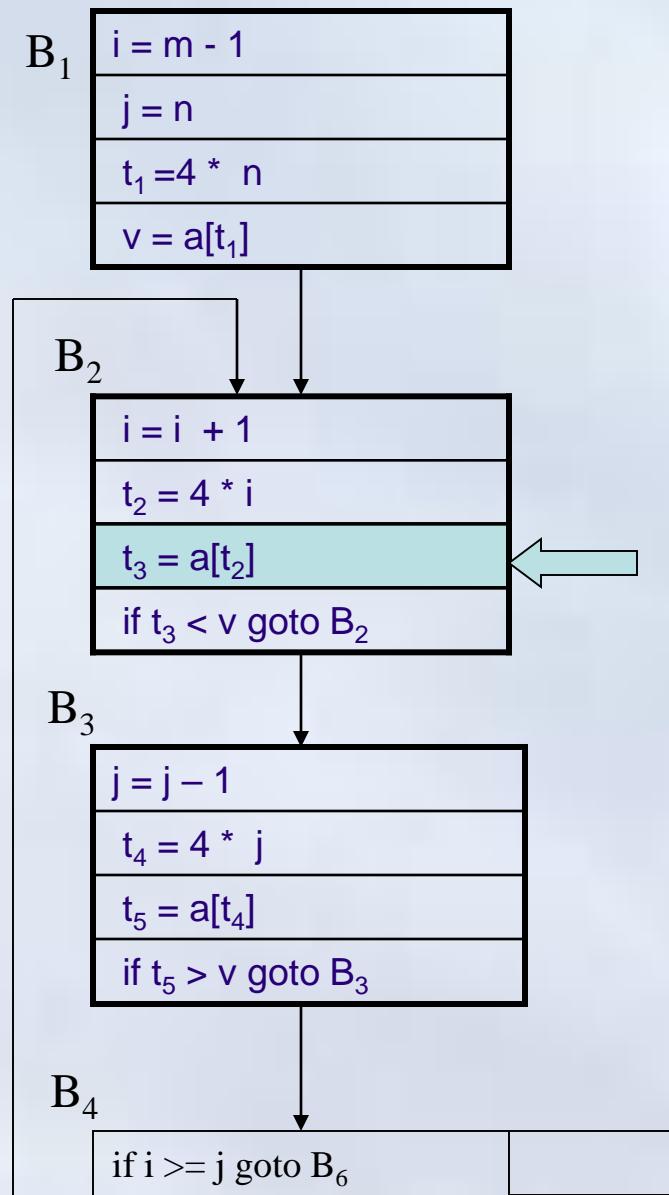
Tối ưu mã → Ví dụ



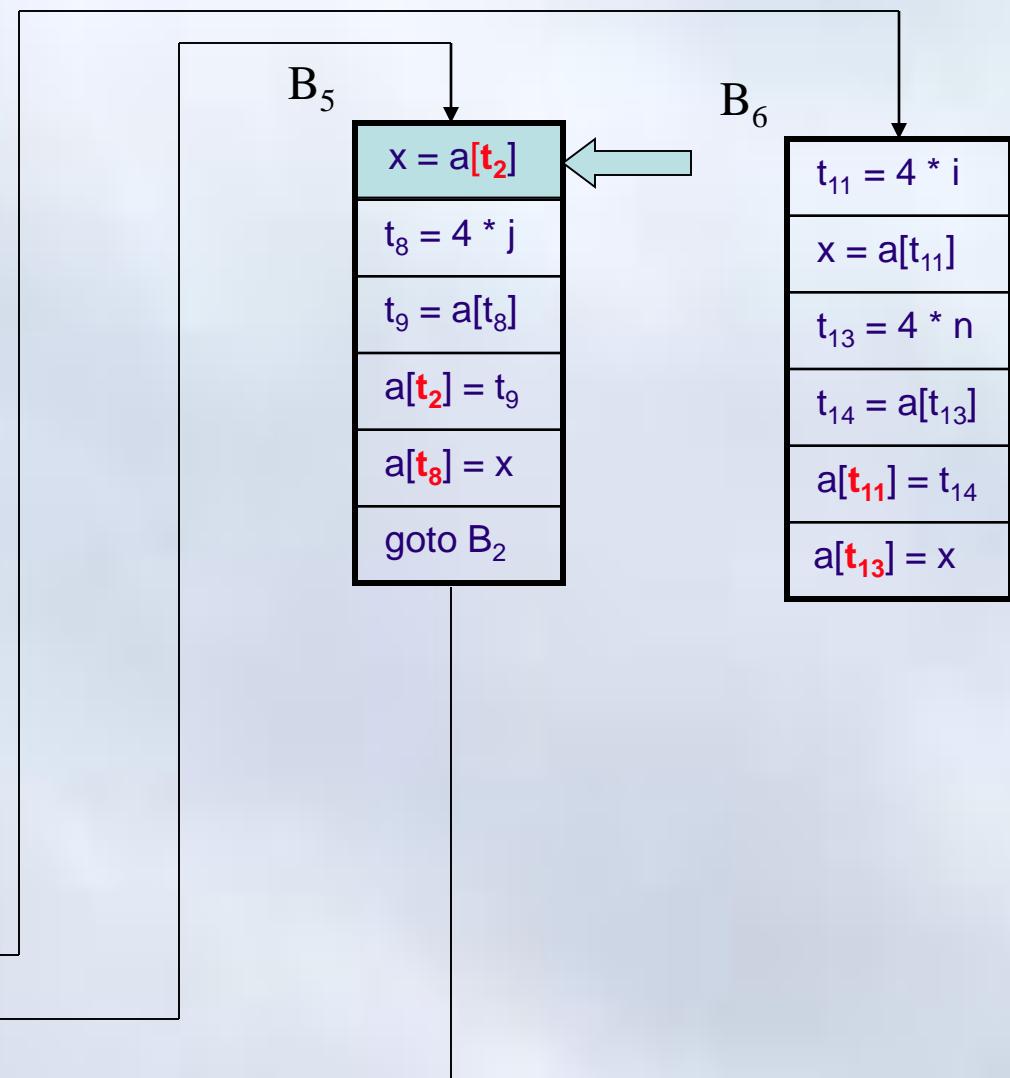
Loại biểu thức con chung



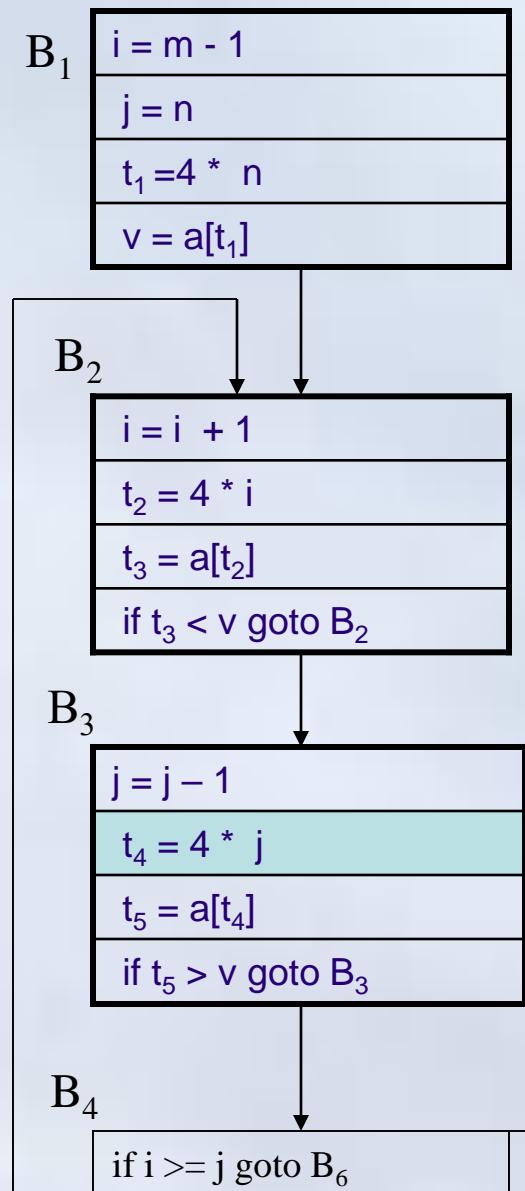
Tối ưu mã → Ví dụ



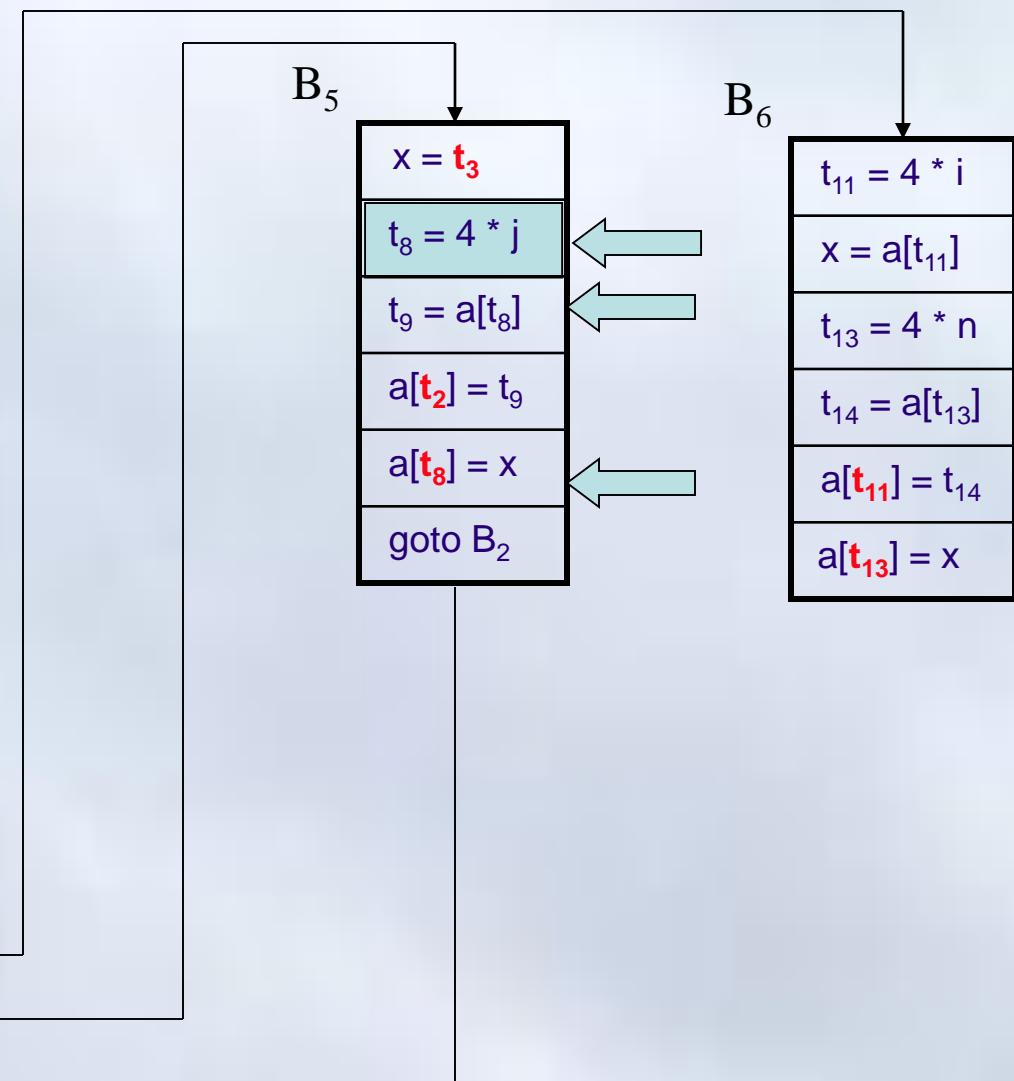
Loại biểu thức con chung



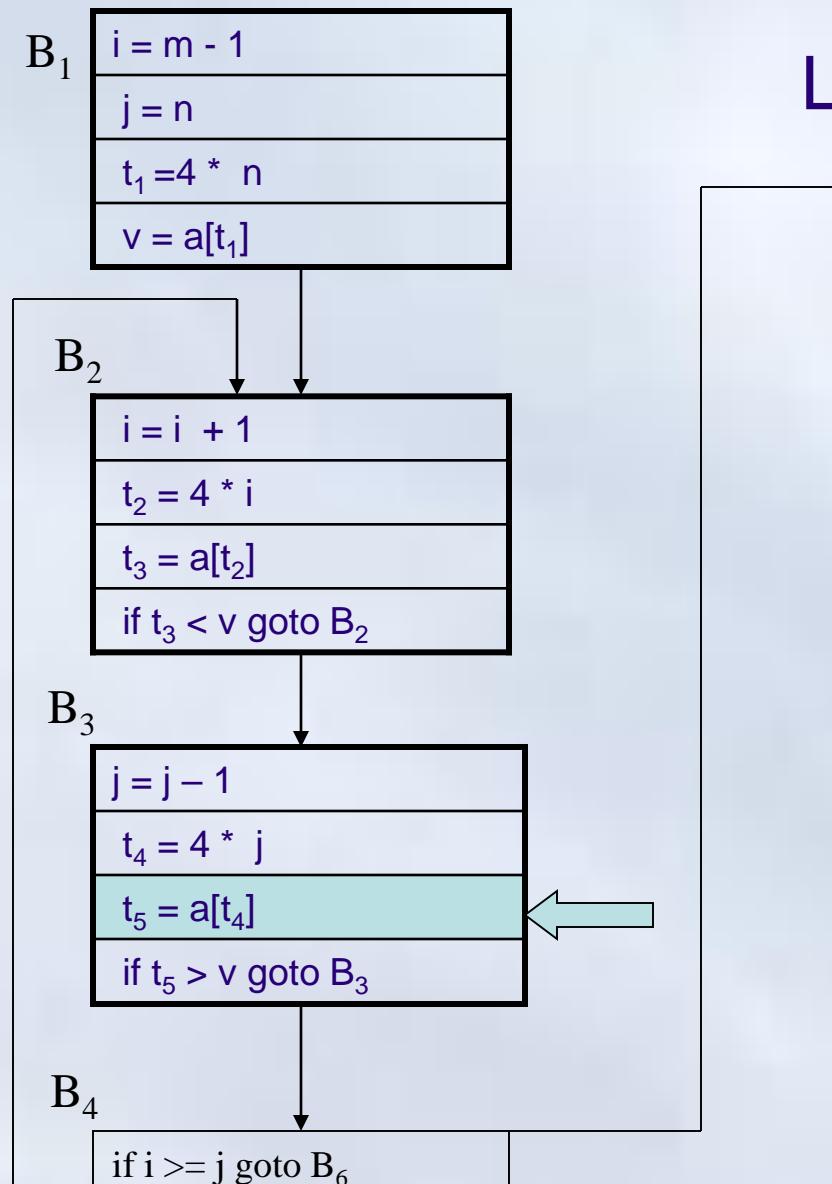
Tối ưu mã → Ví dụ



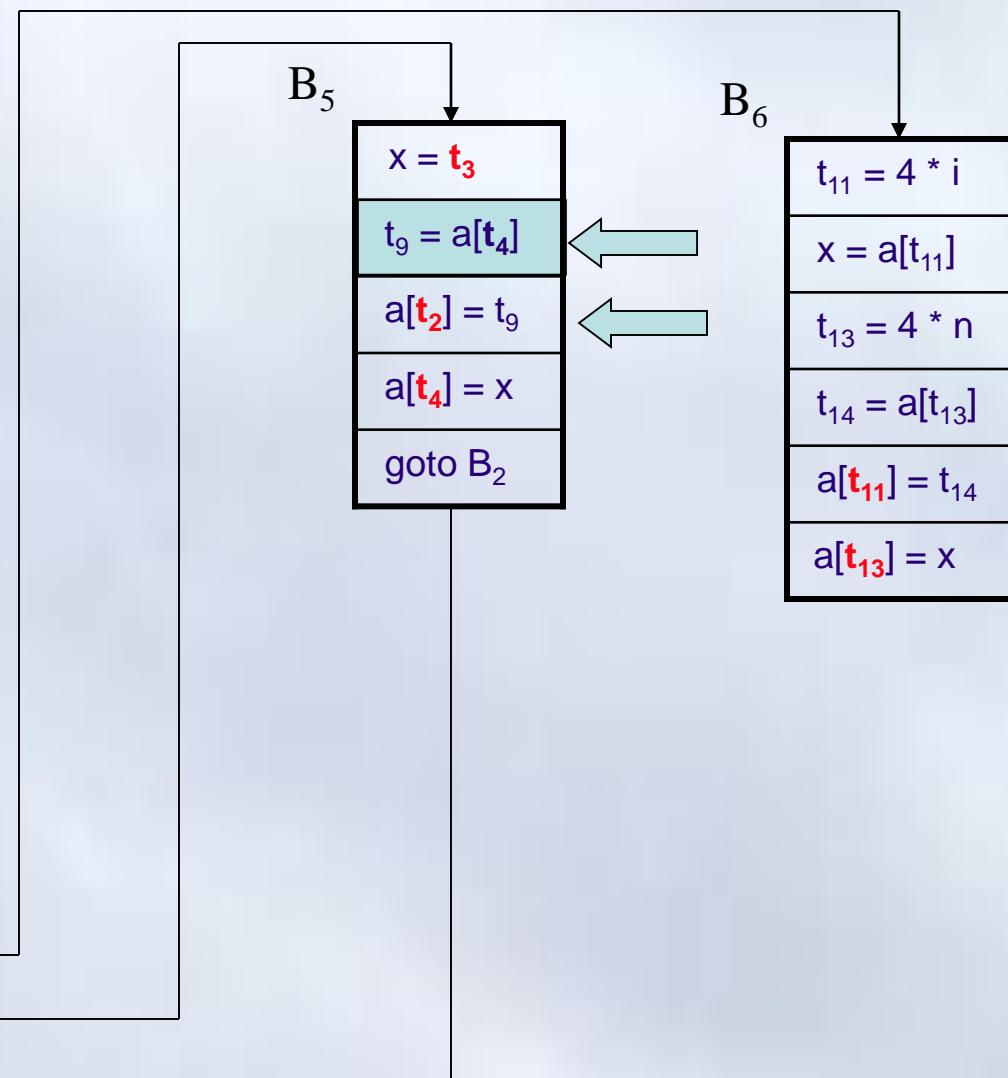
Loại biểu thức con chung



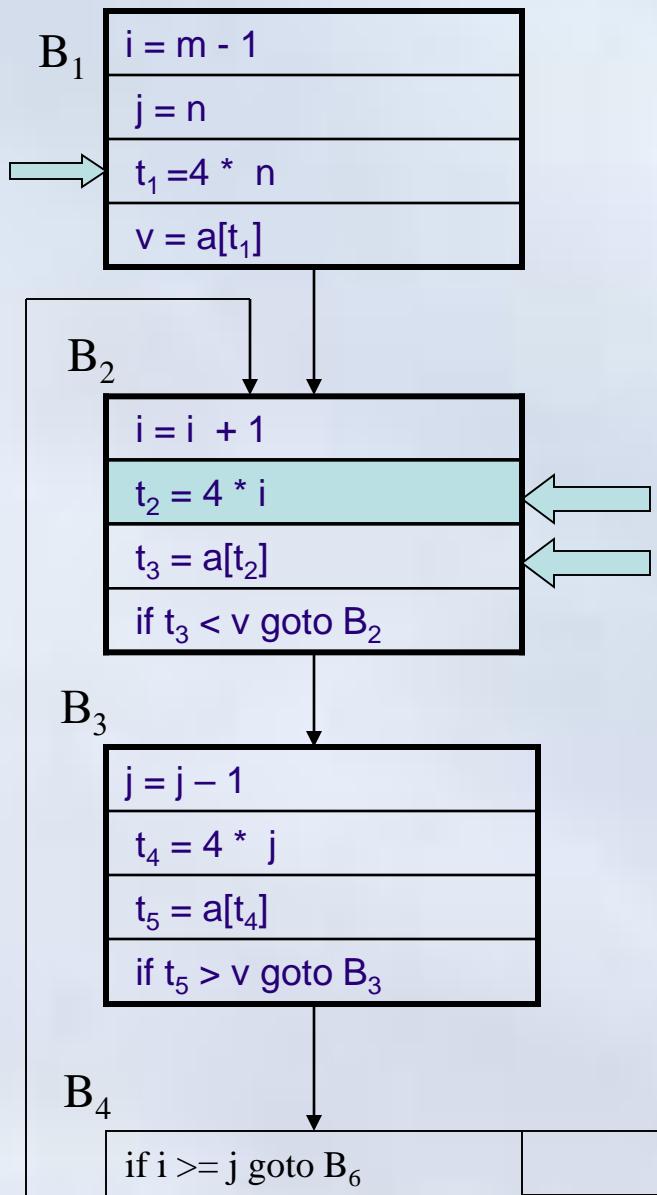
Tối ưu mã → Ví dụ



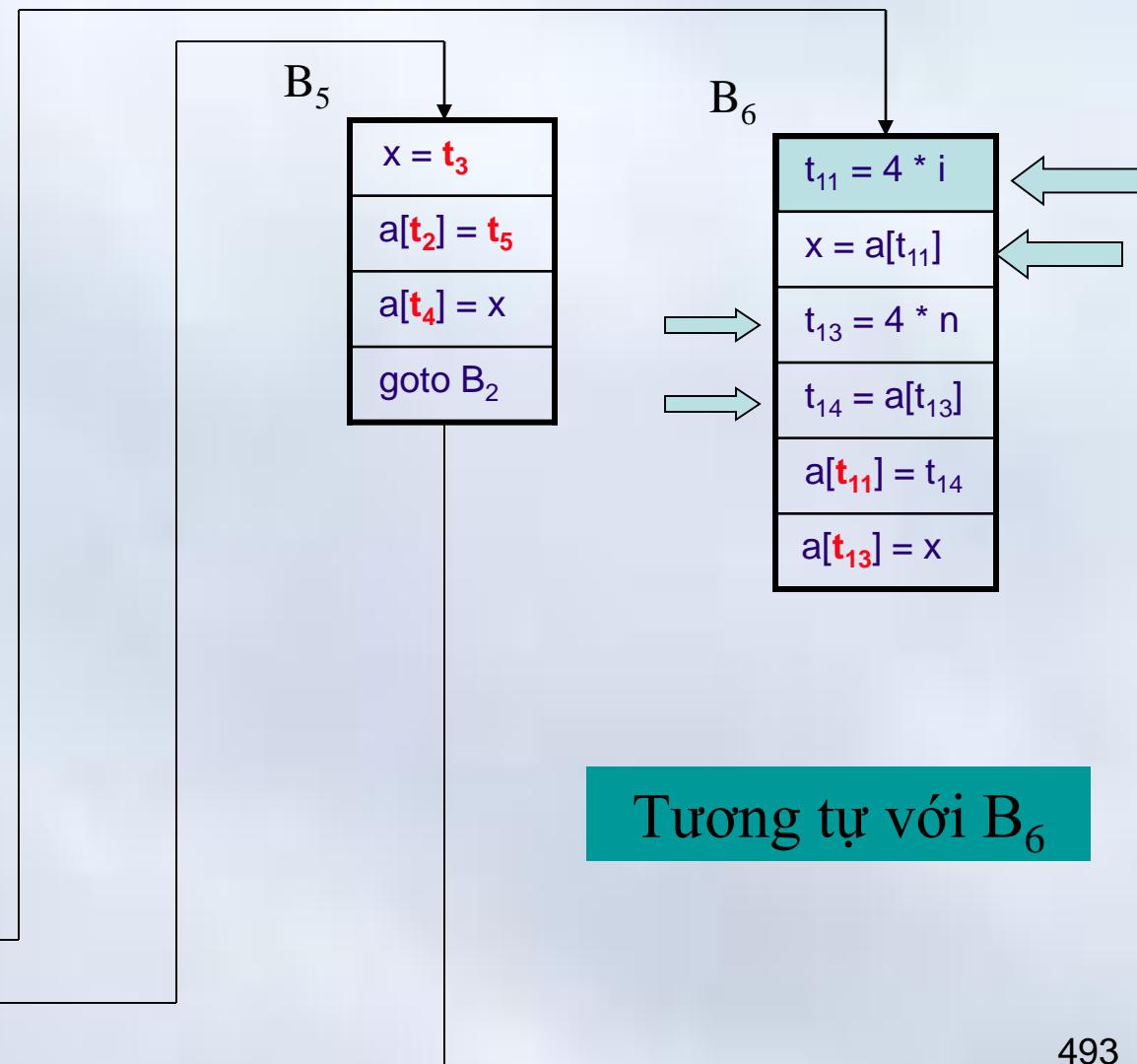
Loại biểu thức con chung



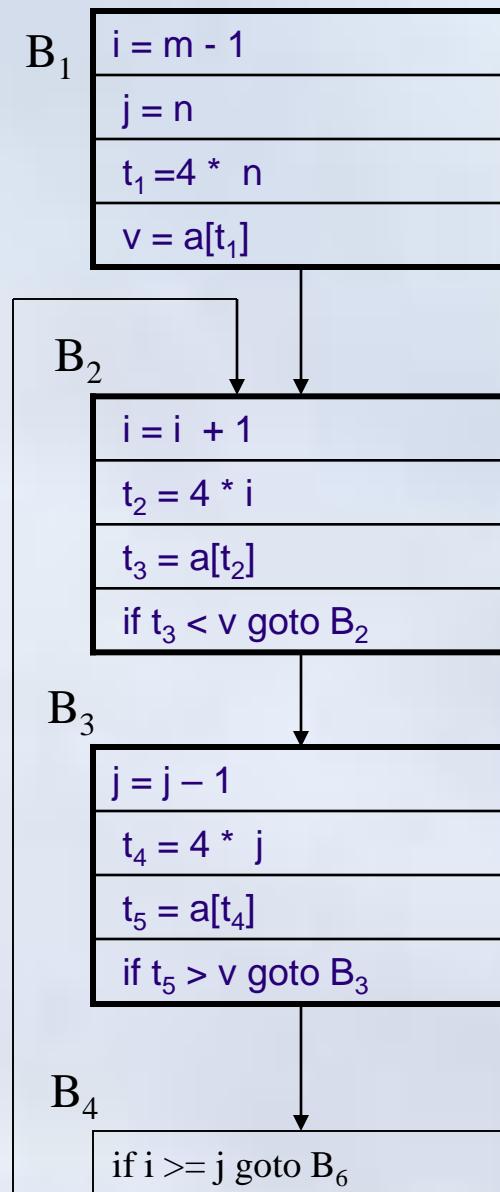
Tối ưu mã → Ví dụ



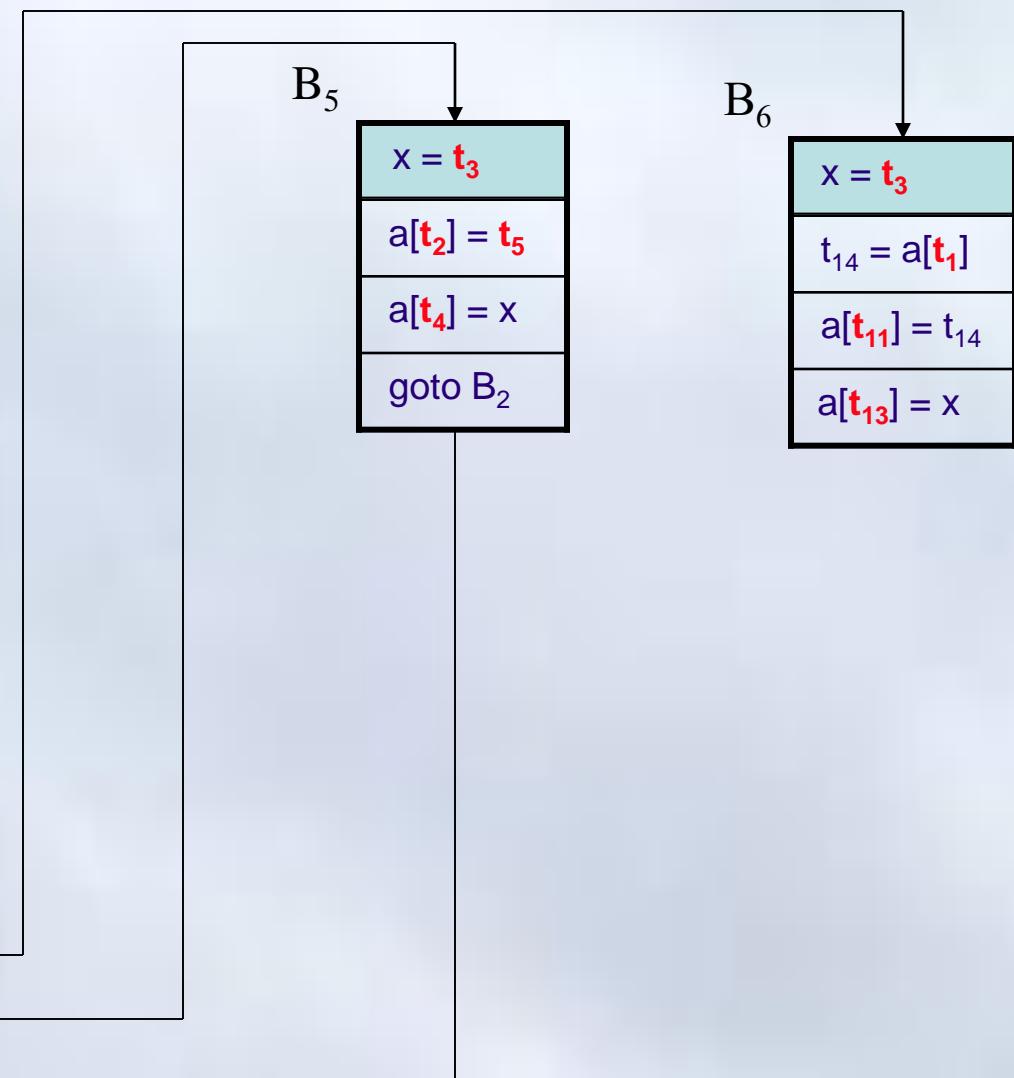
Loại biểu thức con chung



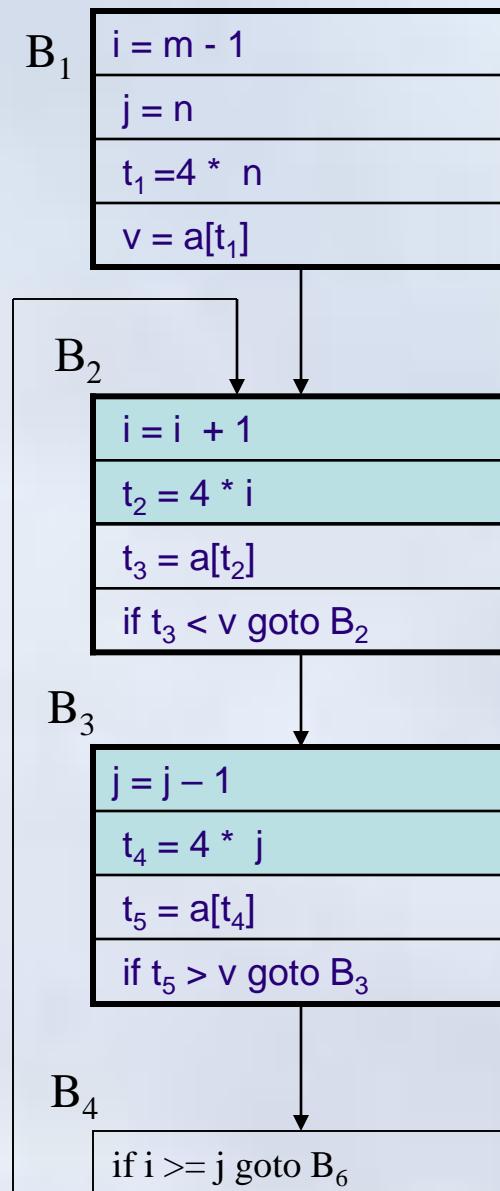
Tối ưu mã → Ví dụ



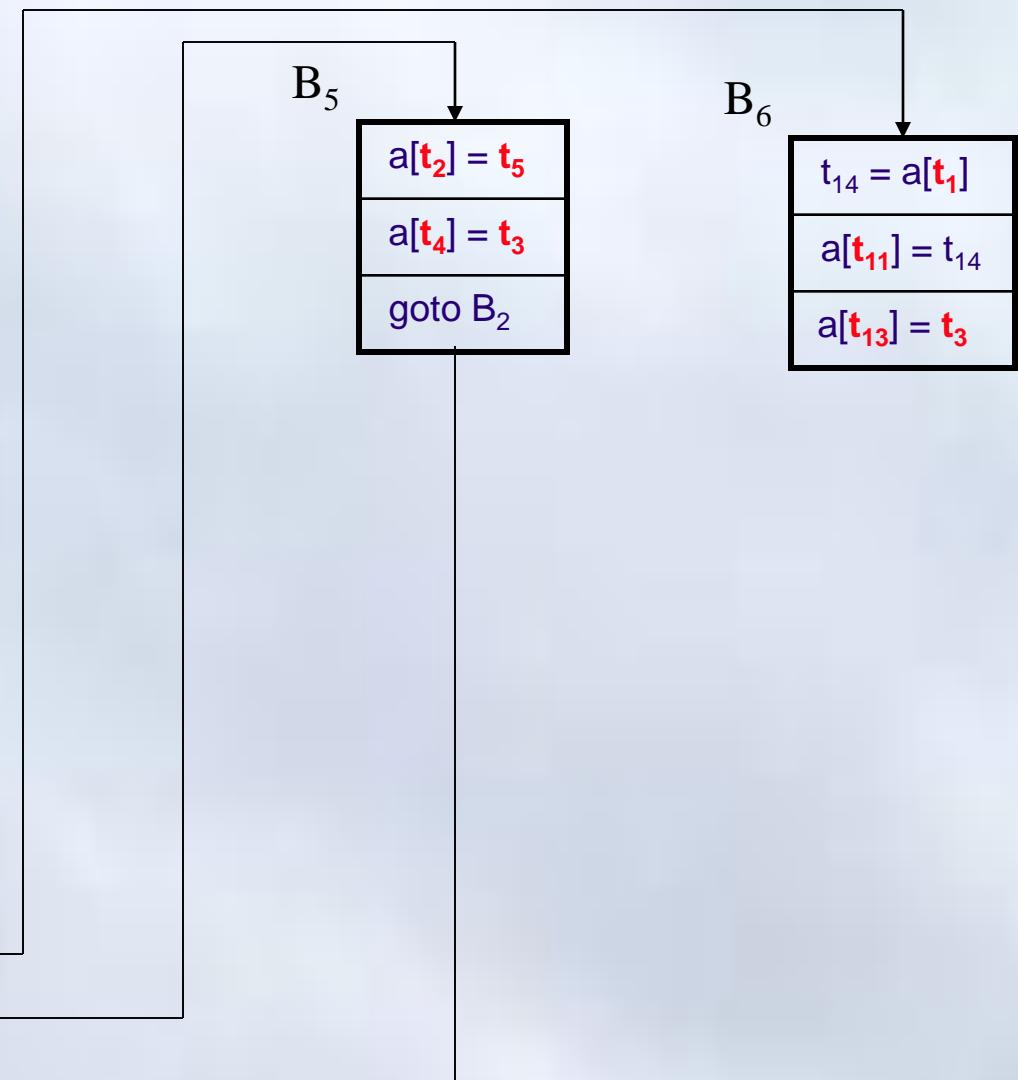
Lan truyền biến gán



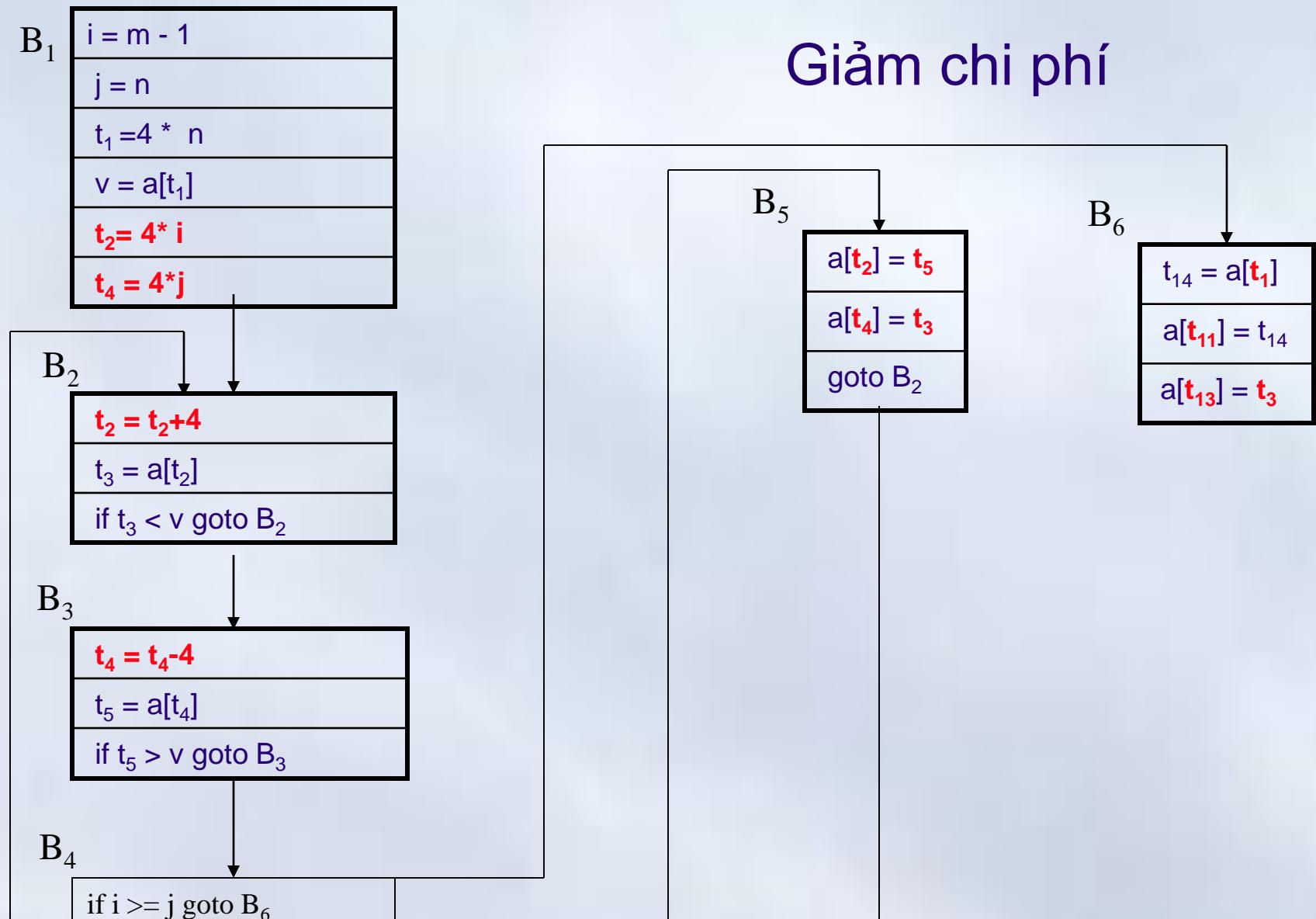
Tối ưu mã → Ví dụ



Giảm chi phí



Tối ưu mã → Ví dụ



Tối ưu mã đích

- Không tồn tại một giải thuật tổng quát để tìm ra chương trình tối ưu nhất
 - Dùng một số thuật toán đơn giản để tối ưu mã đích
 - Xét ví dụ máy có 1 thanh ghi **R** và tồn tại các lệnh
 - **Load m** → Nạp vào thanh ghi R giá trị m ($R=m$)
 - **Add m** → Cộng vào thanh ghi R giá trị m ($R += m$)
 - **Mul m** → Nhân vào thanh ghi R giá trị m ($R *= m$)
 - **Store m** → Ghi giá trị thanh ghi R vào địa chỉ m
- Trong đó, m có thể là hằng số, biến

Tối ưu mã đích → Phép biến đổi đơn giản

- Phép giao hoán
 - Hai lệnh liên tiếp **Load** α và **Add** β có thể được thay thế bằng **Load** β và **Add** α
 - Hai lệnh liên tiếp **Load** α và **Mul** β có thể được thay thế bằng **Load** β và **Mul** α
- Dãy lệnh **store** α **Load** α sẽ bị hủy bỏ nếu
 - α không được sử dụng hoặc được lưu giá trị mới trước khi dùng (store α)
- Dãy lệnh **Load** α **store** β sẽ bị hủy bỏ nếu
 - Sau đó là lệnh load khác và không có sự thay đổi β từ đó về sau, đồng thời sự sử dụng α được thay bằng β

Tối ưu mã đích → Phép biến đổi đơn giản → Ví dụ

- Load c
 - Store #3
 - Load b
 - Store #1
 - Load a
 - Add #1
 - Store #2
 - Load 10
 - Mul #2
 - Add # 3
 - Store V
- Load c
 - Stroe #3
 - Load b
 - Store #1
 - Load # 1
 - Add a
 - Strore #2
 - Load #2
 - Mul 10
 - Add # 3
 - Store V
- Load c
 - Stroe #3
 - Load b
 - Load b
 - Load c
 - Add a
 - Add a
 - Mul 10
 - Add c
 - Store V
-