# Bài 2.
# Chương Trình Dịch Đầu Tiên

Hoàng Anh Việt
Viện CNTT&TT - ĐHBKHN

# Mục đích

- Sau khi học xong chương này, sinh viên sẽ nắm được:
  - Các thành phần cấu tạo nên chương trình dịch đơn giản
  - Hoạt động và cài đặt các giai đoạn của kỳ đầu của trình biên dịch đơn giản: Phân tích từ vựng, phân tích cú pháp và sinh mã trung gian.
  - Sử dụng máy ảo kiểu stack.

# Điều kiện

- Kiến thức cần có:
  - Sử dụng 1 trong các ngôn ngữ: C, Pascal để hiểu cách cài đặt trình Biên dịch
  - Cấu trúc dữ liệu và giải thuật để hiểu cách tổ chức dữ liệu khi cài đặt

# Tài liệu tham khảo

[1] Slide bài giảng

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

[3] Trình Biên Dịch - Phan Thị Tươi (Trường Đại học kỹ thuật Tp.HCM) – NXB Giáo dục, 1998.

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

# Nội dung

1. Định nghĩa cú pháp
2. Dịch trực tiếp cú pháp
3. Phân tích cú pháp
4. Một chương trình dịch biểu thức đơn giản
5. Phân tích từ vựng
6. Xây dựng bảng ký hiệu
7. Máy ảo kiểu stack
8. Kết nối các kỹ thuật

# 1. Định nghĩa cú pháp

1.1 Định nghĩa ngôn ngữ hình thức

1.2Văn phạm phi ngữ cảnh

1.3 Cây phân tích cú pháp

1.4 Sự nhập nhằng của văn phạm

1.5 Sự kết hợp của các toán tử

1.6 Thứ tự ưu tiên của các toán tử

# 1.1 Định nghĩa ngôn ngữ hình thức

- Bảng chữ cái
- Xâu kí tự
- Ngôn ngữ

# 1.1 Định nghĩa ngôn ngữ hình thức

**Bảng chữ cái:**

- Cho $\sum$ là một tập hữu hạn, khác rỗng các ký hiệu nào đó mà ta gọi là bảng chữ cái. Mỗi phần tử trong $\sum$ được gọi là một ký tự
- Ví dụ $\sum$={a,b,c,d,….,y}

  $\sum$={1,2,3} ;

# 1.1 Định nghĩa ngôn ngữ hình thức

**Xâu ký tự:**

- Là một dãy các ký tự trong bảng chữ cái $\sum$ được viết liền nhau
- Độ dài xâu: là số ký tự trong xâu đó
- Ví dụ $\sum$={a,b,c} . s= "baccba" là một xâu trên bảng chữ cái $\sum$. Xâu s có độ dài bằng 6
- Xâu rỗng: là xâu không có ký tự nào, độ dài bằng 0. Ký hiệu: $\lambda$

# 1.1 Định nghĩa ngôn ngữ hình thức

**Ngôn ngữ**

- Mỗi tập từ trên bảng chữ cái $\sum$ được gọi là ngôn ngữ trên bảng chữ cái đó.
- $\sum^*$: là tập tất cả các từ trên bảng chữ cái kể cả xâu rỗng
- $\sum^+ = \sum^* - \{\lambda\}$

# 1.2 Văn phạm phi ngữ cảnh

- Định nghĩa văn phạm:
  - Định nghĩa 1: văn phạm G là một bộ sắp thứ tự gồm 4 thành phần < ∑,Δ,I,R >, trong đó:
    - ∑: Bảng chữ cái, tập các ký hiệu kết thúc.
    - Δ: tập các chữ cái hỗ trợ, các phần tử (chữ cái hỗ trợ) được gọi là các ký hiệu không kết thúc.
      - » V= (∑UΔ)* được gọi là từ điển đầy đủ
    - I € Δ được gọi là ký hiệu ban đầu.
    - R là tập các quy tắc mà mỗi phần tủ của nó có dạng a→b, a, b là các từ trên từ điển đầy đủ

# 1.2 Văn phạm phi ngữ cảnh

- Định nghĩa văn phạm:
  - Định nghĩa 2: Cho G= < ∑,Δ,I,R > là một văn phạm, một xâu x= αaβ. S = αbβ  được gọi là dẫn xuất trực tiếp từ xâu x nếu ta áp dụng quy tắc (luật) a→b. Ký hiệu là x ⊨ s.
  - Định nghĩa 3: Dãy các xâu D = ($w_0$,$w_1$,….,$w_k$) được gọi là một dẫn xuất của xâu $w_k$từ $w_0$ nếu $w_i$ ⊨ $w_{i+1}$ với i=0...k-1. Số k được gọi là độ dài của dẫn xuất. Ký hiệu là $w_0$ |- $w_k$.

# 1.2 Văn phạm phi ngữ cảnh

- CFG- Context Free Grammar
- Để xác định cú pháp của một ngôn ngữ.
- Bao gồm 4 thành phần: G=< ∑,Δ,I,R >
  - ∑ : Tập các Token- ký hiệu kết thúc (terminal symbols). Ví dụ: các từ khóa, các dấu,…
  - Δ: Tập các ký hiệu chưa kết thúc (nonterminal symbols). Ví dụ: câu lệnh, biểu thức.
  - I: Là 1 ký hiệu chưa kết thúc trong Δ được chọn làm ký hiệu bắt đầu của văn phạm.
  - R: Tập các luật sinh, với mọi quy tắc r ∈ R đều có dạng r= A→β, trong đó A ∈ Δ,β ∈ V*.

# 1.2 Văn phạm phi ngữ cảnh

Ví dụ 1: Cho G=< ∑,Δ,I,R > trong đó ∑={a,b}, Δ={I}, I là ký hiệu xuất phát và R={I→λ,I→aIa,I→bIb,I→aa,I→bb} là một văn phạm phi ngữ cảnh.

# 1.2 Văn phạm phi ngữ cảnh

- Một số quy ước:
  - Mô tả văn phạm bằng cách liệt kê luật sinh
  - Luật sinh chứa ký hiệu bắt đầu sẽ được liệt kê đầu tiên
  - Nếu có nhiều luật sinh có cùng về trái thì nhóm lại thành 1 luật sinh duy nhất, trong đó các vế phải cách nhau bởi ký hiệu "|" đọc là "hoặc"

# 1.2 Văn phạm phi ngữ cảnh

- Ví dụ 1: Giả sử biểu thức là 1 danh sách của các số phân biệt nhau bởi dấu + và dấu –

list → list + digit
list → list - digit
list → digit
digit → 0 | 1 | 2 | ...| 9

⇔

list → list + digit | list - digit | digit
digit → 0 | 1 | 2 ...| 9

- VP ở đây được mô tả:
  - Tập ký hiệu kết thúc: 0,1,2..9, +, -
  - Tập không kết thúc: list, digit
  - Các luật sinh bên trên
  - Ký hiệu bắt đầu: list

# 1.2 Văn phạm phi ngữ cảnh

- Ví dụ 2: với list là 1 chuỗi các lệnh phân cách bởi dấu ; của khối begin-end trong pascal.

Luật sinh:

block -> **Begin** whole_stmt **End**

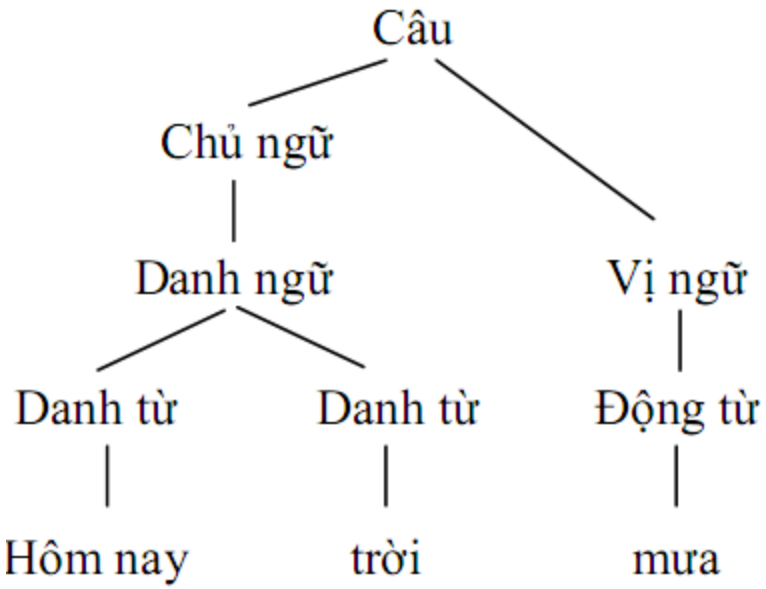
whole_stmt ->stmt_list | €

stmt_list -> stmt_list **;** stmt | stml

# 1.3 Cây phân tích cú pháp

Ví dụ: Bộ luật cú pháp của CFG:

1. <Câu>→<Chủ ngữ><Vị ngữ>
2. <Câu>→<Trạng ngữ><Chủ ngữ><Vị ngữ>
3. <Trạng ngữ>→<Phó từ>
4. <Danh ngữ>→<Danh từ><Danh từ>
5. <Chủ ngữ>→<Danh ngữ>
6. <Chủ ngữ>→<Danh từ>
7. <Vị ngữ>→<Động ngữ>
8. <Vị ngữ>→<Động từ>



# 1.3 Cây phân tích cú pháp

- Tính chất cây phân tích cú pháp:
  - Nút gốc có nhãn là ký hiệu bắt đầu
  - Mỗi một lá có nhãn là một ký hiệu kết thúc hoặc là 1 ký hiệu rỗng €
  - Mỗi 1 nút (có nhãn) là một ký hiệu chưa kết thúc
  - Nếu A là nhãn của nút không phải là nút cuối, X1, X2, …Xn là nhãn các con của nút có nhãn Atừ trái sang phải thì A-> X1X2…Xn là luật sinh thuộc tập luật sinh

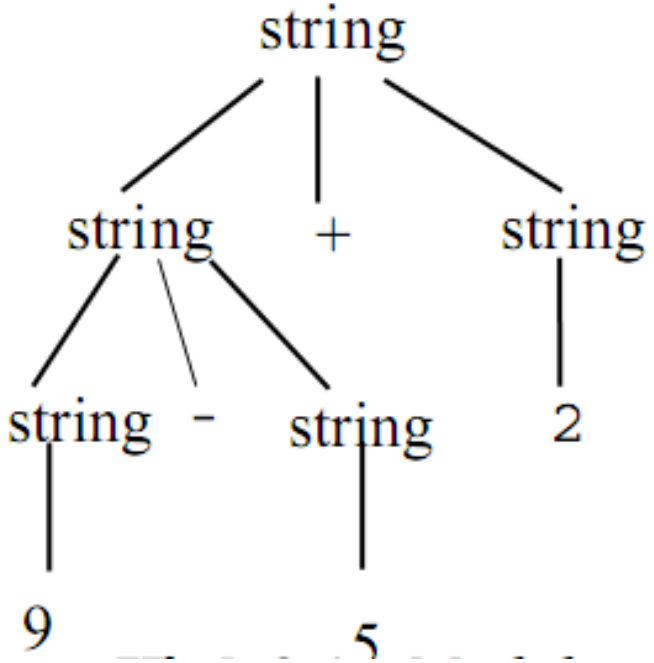
# 1.4 Sự nhập nhằng của văn phạm

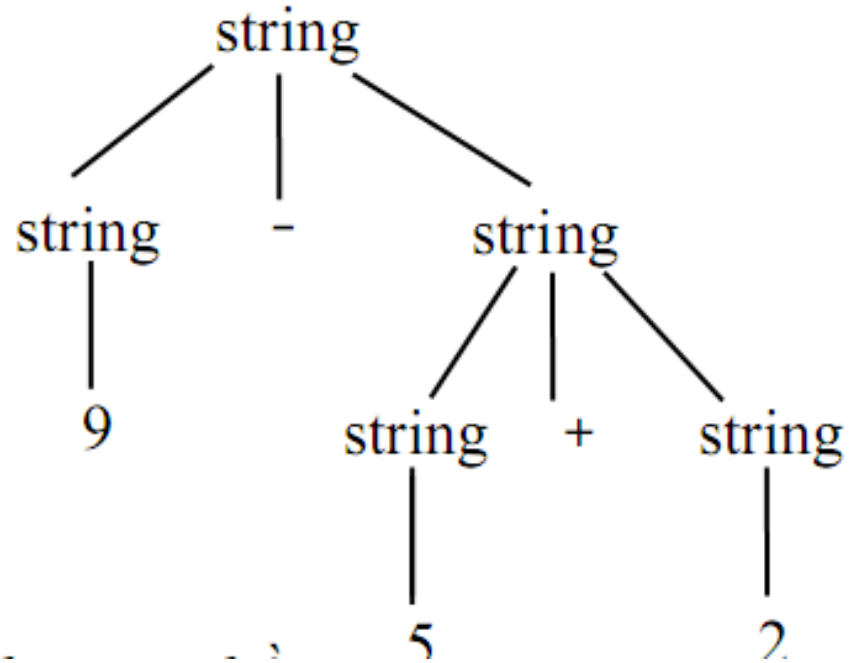
- 1 Văn phạm sinh ra nhiều hơn 1 cây phân tích cú pháp cho cùng 1 chuỗi nhập thì gọi là văn phạm nhập nhằng.
- Ví dụ văn phạm G sau đây là không tường minh:

P : string -> string + string |string –string |0 |1 |... |9

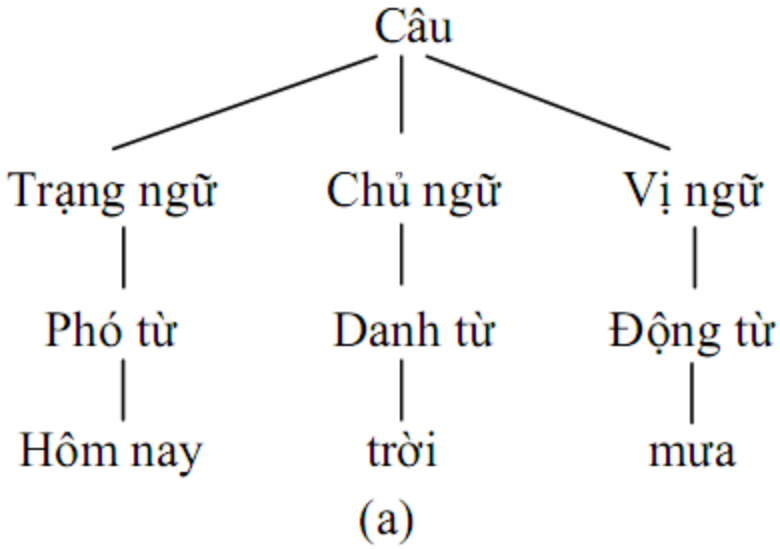
Câu 9 – 5 + 2 cho hai cây phân tích:

# 1.4 Sự nhập nhằng của văn phạm

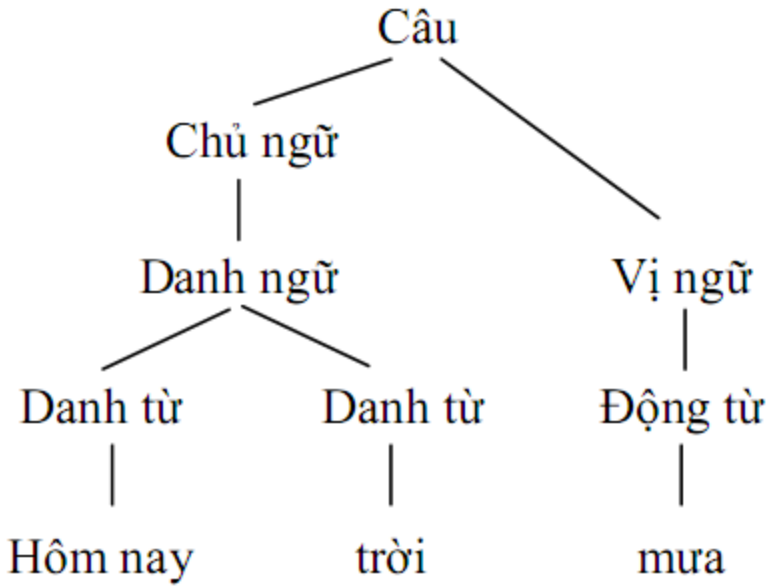


# 1.4 Sự nhập nhằng của văn phạm



(a)



(b)

# 1.5 Sự kết hợp của các toán tử

- Biểu thức a + b +c tương đương với (a+b)+c.
- Toán tử bên trái được thực hiện trước thì gọi là **kết hợp trái,** ngược lại là **kết hợp phải.**
- Các phép toán số học: +, -, *,/ : kết hợp trái
- Các pháp toán số mũ, gán bằng = có tính kết hợp phải.

# 1.5 Sự kết hợp của các toán tử

- Mức ưu tiên của các toán tử: * và / có mức ưu tiên hơn + , -. Dựa vào nguyên tắc trên chúng ta xây dựng cú pháp cho biểu thức số học:

  exp -> exp + term |exp – term |term

  term -> term * factor |term / factor |factor

  factor -> digit |( exp )

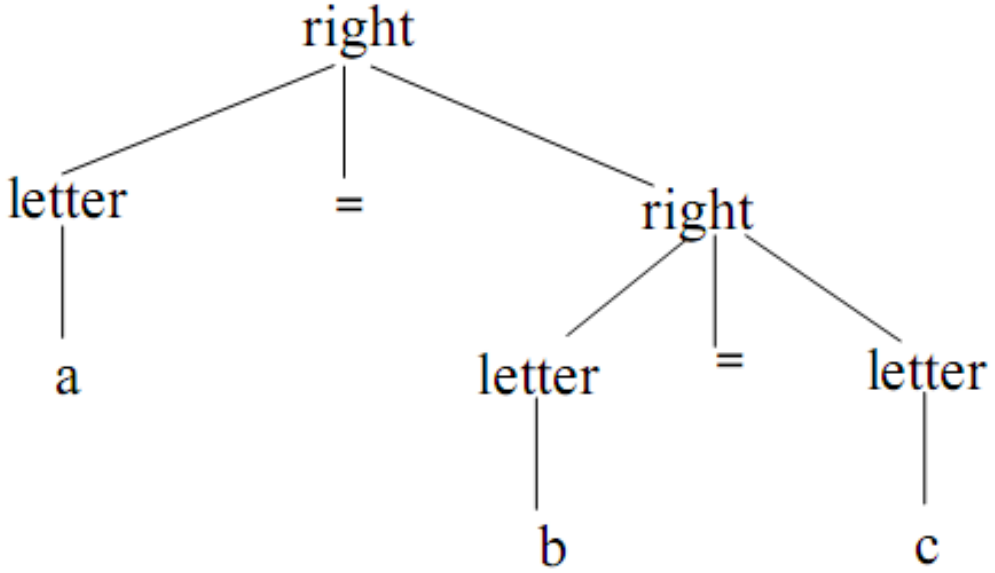
- Lưu ý: phép toán lũy thừa và phép gán trong C là phép toán kết hợp phải. Văn phạm cho phép gán như sau:

  right -> letter = right |letter

  letter -> a |b |…|z

# 1.5 Sự kết hợp của các toán tử

- Ví dụ: Xét biểu thức a=b=c, tương đương với a=(b=c)




***Chú ý***: *hướng nghiêng của cây*

# 1.6 Thứ tự ưu tiên của các toán tử

- Biểu thức: x*y+t

Có 2 cách diễn giải: (x*y)+t hoặc x*(y+t)

=> nhập nhằng. Giải quyết bằng độ ưu tiên

- Trong toán học, toán tử * và / có độ ưu tiên cao hơn + và -

Kết hợp trái +, -

Kết hợp trái *, /

Thứ tự ưu tiên từ thấp đến cao

⇒VP cho biểu thức số học: Bảng kết hợp và độ ưu tiên

# 1.6 Thứ tự ưu tiên các toán tử

- **Cú pháp cho biểu thức:**
  - Văn phạm này xem biểu thức như là một danh sách các **term** được phân cách nhau bởi dấu + hoặc -.
  - Term là một **list** các **factor** phân cách nhau bởi * hoặc /. Bất kỳ một biểu thức nào trong ngoặc đều là **factor**, vì thế với các dấu ngoặc chúng ta có thể xây dựng các biểu thức lồng sâu nhiều cấp tuỳ ý.

$$expr \rightarrow expr + term \mid expr - term \mid term$$

$$term \rightarrow term * factor \mid term / factor \mid factor$$

$$factor \rightarrow digit \mid (expr)$$

# 1.6 Thứ tự ưu tiên các toán tử

- **Cú pháp các câu lệnh:**
  - Từ khóa (keyword) cho phép chúng ta nhận ra câu lệnh trong hầu hết các ngôn ngữ.
  - Hầu hết các lệnh đều bắt đầu bởi một từ khóa ngoại trừ lệnh gán.

```
stmt → id := expr
     | if expr then stmt
     | if expr then stmt else stmt
     | while expr do stmt
     | begin opt_stmts end
```

Trong đó:

- id chỉ một danh biểu (tên biến).

-Ký hiệu chưa kết thúc **opt_stmts** sinh ra một danh sách (có thể rỗng) các lệnh phân cách nhau bởi dấu chấm phẩy (;)

# 2.Dịch Trực tiếp cú pháp

- 2.1 Ký pháp hậu tố
- 2.2 Định nghĩa trực tiếp cú pháp
- 2.3 Thuộc tính tổng hợp
- 2.4 Duyệt theo chiều sâu
- 2.5 Lược đồ dịch

# 2.1 Ký pháp hậu tố

- Ký pháp hậu tố của biểu thức E định nghĩa:
  1. Nếu E là 1 biến hay hằng thì ký pháp hậu tố của E là chính E.
  2. Nếu E là biểu thức dạng **E1 op E2** thì ký pháp hậu tố của E là **E1'E2'op.**
  3. Nếu E là biểu thức dạng **(E1)** thì ký pháp hậu tố của E là ký pháp hậu tố của **E1**

Ví dụ: Dạng hậu tố của (5-3) +4 là 53-4+

Dạng hậu tố của 6-(3+5) là 635+-

# 2.2 Định nghĩa trực tiếp cú pháp

- Định nghĩa trực tiếp cú pháp (syntax- directed definition) là sự tổng quát hóa một văn phạm phi ngữ cảnh, trong đó mỗi ký hiệu văn phạm kết hợp với một tập các thuộc tính (attribute)
- Các thuộc tính có thể là một xâu, một số, một kiểu dữ liệu, một địa chỉ trong bộ nhớ...
- Giá trị các thuộc tính được tính bởi các luật ngữ nghĩa (semantic rule) đi kèm. Mỗi luật ngữ nghĩa được viết như lời gọi các thủ tục hoặc một đoạn chương trình
- Cây phân tích cú pháp có trình bày giá trị các thuộc tính tại mỗi nút gọi là cây chú thích

# 2.2 Định nghĩa trực tiếp cú pháp

- Trong một định nghĩa trực tiếp cú pháp, mỗi luật sinh $A \rightarrow \alpha$ kết hợp một tập luật ngữ nghĩa có dạng b:= f ($c_1$, $c_2$,..., $c_k$) trong đó f là một hàm và:

1) b là một thuộc tính tổng hợp (synthesized attribute) của A và $c_1$, $c_2$,..., $c_k$ là các thuộc tính của các ký hiệu văn phạm của luật sinh. Hoặc
2) b là một thuộc tính kế thừa (inherited attribute) của một trong các ký hiệu văn phạm trong vế phải của luật sinh và $c_1$, $c_2$,..., $c_k$ là các thuộc tính của các ký hiệu văn phạm của luật sinh

**Ví dụ:** Định nghĩa trực tiếp cú pháp (ĐNTTCP) cho một máy tính đơn giản

| PRODUCTION | SYMANTIC RULES |
|---|---|
| L → E**n** | *print(E.val)* |
| E → $E_1$ + T | *E.val* := $E_1$*.val* + *T.val* |
| E → T | *E.val* := *T.val* |
| T → $T_1$ * F | *T.val* := $T_1$*.val* * *F.val* |
| T → F | *T.val* := *F.val* |
| F → (E) | *F.val* := *E.val* |
| F → **digit** | *F.val* := *digit.lexval* |

- Token digit có thuộc tính tổng hợp *lexval* mà giá trị được cung cấp bởi bộ phân tích từ vựng

# 2.3 Thuộc tính tổng hợp

- Thuộc tính tổng hợp là thuộc tính mà giá trị của nó tại mỗi nút trên cây phân tích cú pháp được tính từ giá trị thuộc tính tại các nút con của nó
- Định nghĩa trực tiếp cú pháp chỉ sử dụng các thuộc tính tổng hợp gọi là định nghĩa S- thuộc tính (S-attributed definition)
- Trong cây phân tích cú pháp của định nghĩa S- thuộc tính, các luật ngữ nghĩa tính giá trị các thuộc tính cho các nút từ dưới lên, từ lá đến gốc

# 2.3 Thuộc tính tổng hợp

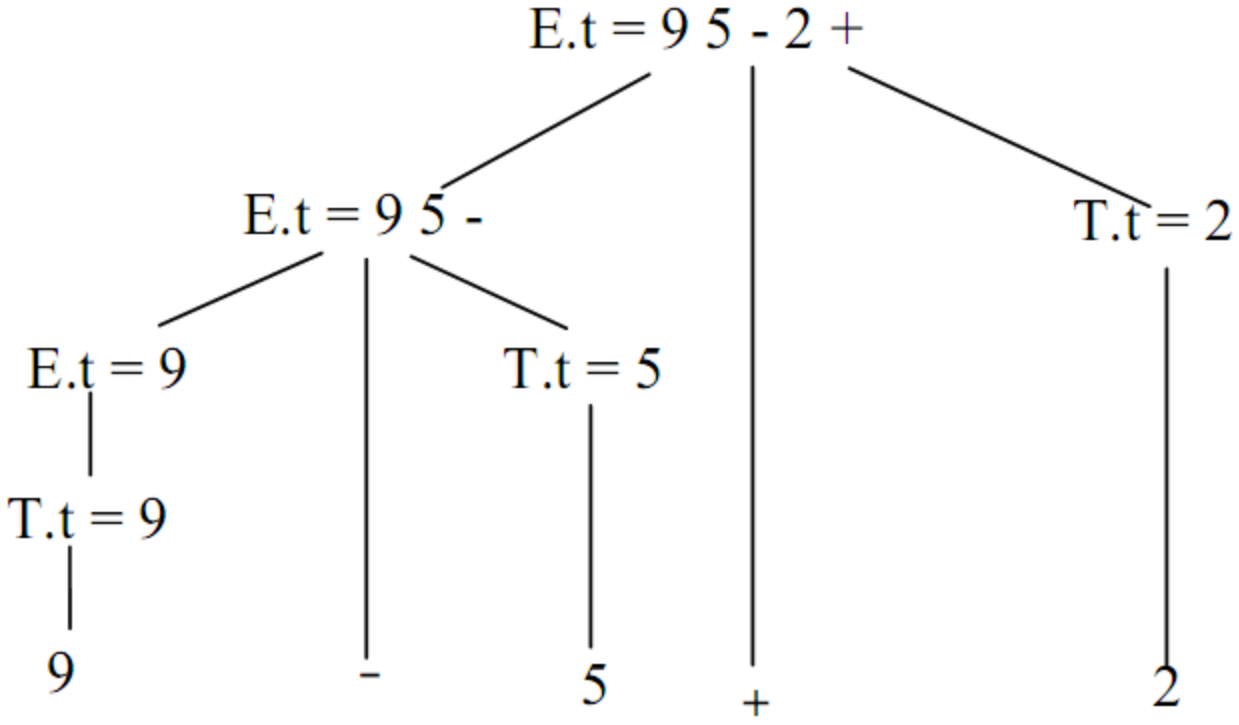
- Một thuộc tính được gọi là tổng hợp nếu giá trị của nó tại một nút trên cây cú pháp được xác định từ các giá trị của các thuộc tính tại các nút con của nút đó.

Ví dụ: DNTTCP cho việc dịch biểu thức các số cách nhau bởi + và – thành ký pháp hậu tố

| **Luật sinh** | **Quy tắc ngữ nghĩa** |
|---|---|
| $E \rightarrow E1 + T$ | E.t := E1.t \|\| T.t \|\| '+' |
| $E \rightarrow E1 - T$ | E.t := E1.t \|\| T.t \|\| '-' |
| $E \rightarrow T$ | E.t := T.t |
| $T \rightarrow 0$ | T.t := '0' |
| ... | ... |
| $T \rightarrow 9$ | T.t := '9' |

# 2.3 Thuộc tính tổng hợp

- Cây PTCP cho biểu thức **9-5+2:**




*Cây phân tích cú pháp chú thích*

# 2.4 Duyệt theo chiều sâu (Depth - First Traversal)

**Procedure** visit (n: node);

**begin**

For với mỗi con m của n, từ trái sang phải do

visit (m);

Đánh giá quy tắc ngữ nghĩa tại nút n *(tính trị ngữ nghiã tại nút n)*

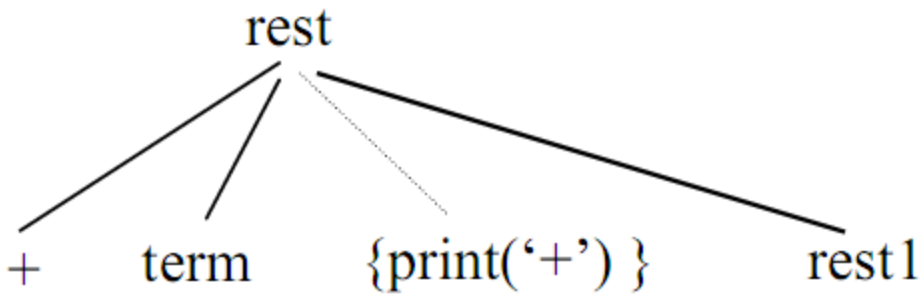
**end;**

# 2.5 Lược đồ dịch

- Một lược đồ dịch là một VPPNC, trong đó các đoạn chương trình gọi là hành vi ngữ nghĩa (semantic actions) được gán vào vế phải của luật sinh.
- Giống DNTTCP nhưng thứ tự đánh giá các quy tắc ngữ nghĩa được trình bày một cách rõ ràng.
- Vị trí mà tại đó một hành vi được thực hiện được trình bày trong cặp dấu ngoặc nhọn { } và viết vào vế phải luật sinh.

# 2.5 Lược đồ dịch

- Ví dụ: rest → + term {print ('+')} rest1.

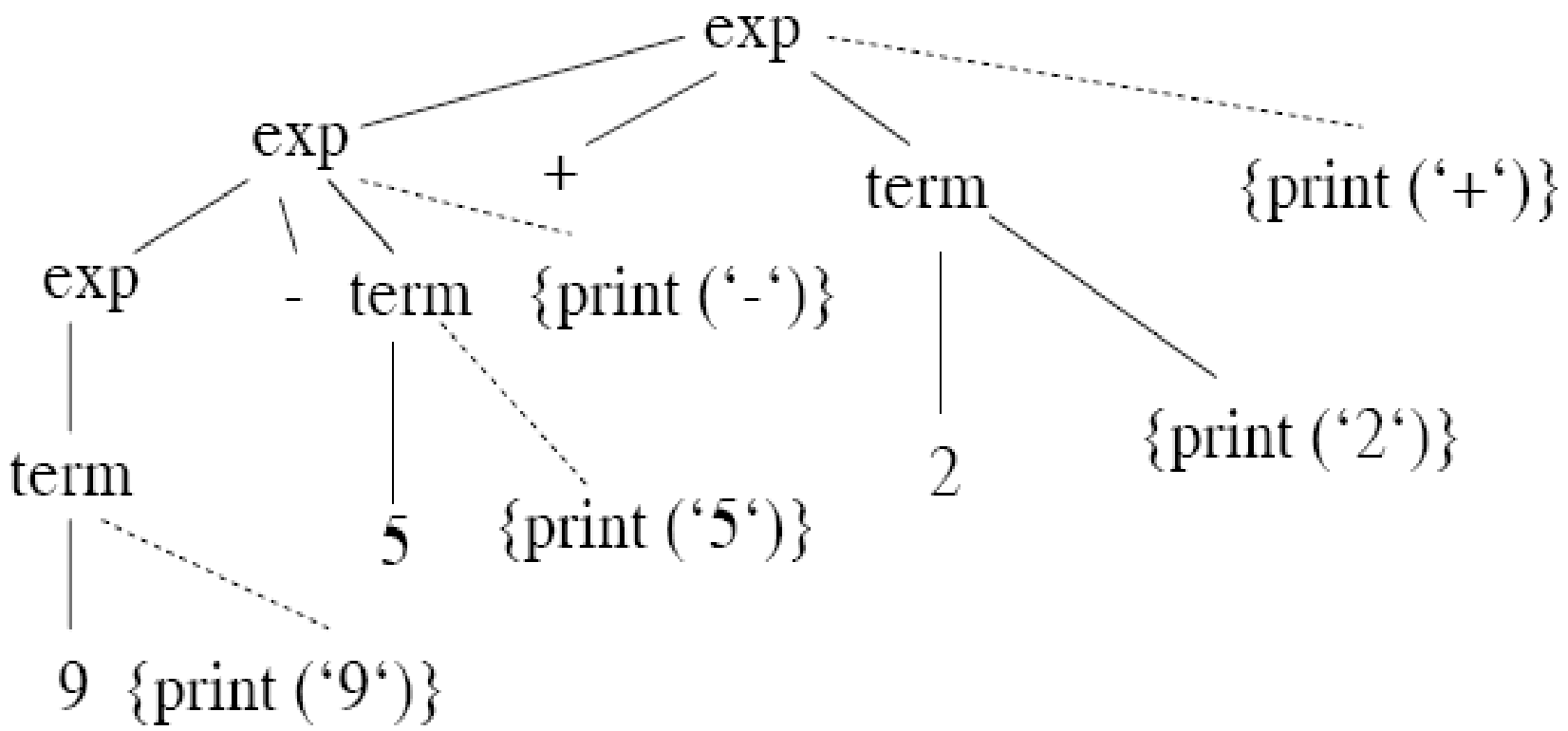


# 2.5 Lược đồ dịch

- Ví dụ: Lược đồ dịch của văn phạm G:

| Tập luật sinh | Tập luật ngữ nghĩa |
|---|---|
| exp → exp + term | exp → exp + term { print ('+')} |
| exp → exp – term | exp → exp – term {print ('-')} |
| exp → term | exp → term |
| term → 0 | term → 0 {print ('0')} |
| ..... | |
| term → 9 | term → 9 {print {'9')} |

# 2.5 Lược đồ dịch




*Hình 2.4. Lược đồ dịch của câu 9 – 5 + 2*

# 3. Phân tích cú pháp

3.1 Phân tích cú pháp từ trên xuống (Top - Down Parsing)

3.2 Phân tích cú pháp dự đoán

3.3 Loại bỏ đệ quy trái

# 3. Phân tích cú pháp

- Là quá trình xác định xem chuỗi ký hiệu kết thúc có thể được sinh ra từ 1 văn phạm?
- 2 lớp: PT từ dưới lên và từ trên xuống (thứ tự xây dựng nút).
- PT trên xuống:
  - Tiến hành từ gốc hướng đến lá
  - Thông dụng nhờ tính hiệu quả
- PT dưới lên:
  - Tiến hành từ lá hướng đến gốc
  - Xử lý được lớp văn phạm và lược đồ dịch phong phú

# 3.1 Phân tích cú pháp từ trên xuống (Top - Down Parsing)

- Ví dụ: Cho văn phạm G sinh ra một tập các kiểu dữ liệu trong Pascal:

  type -> simple |^id| **array** [simple] of **type**

  simple -> integer|char|num .. num

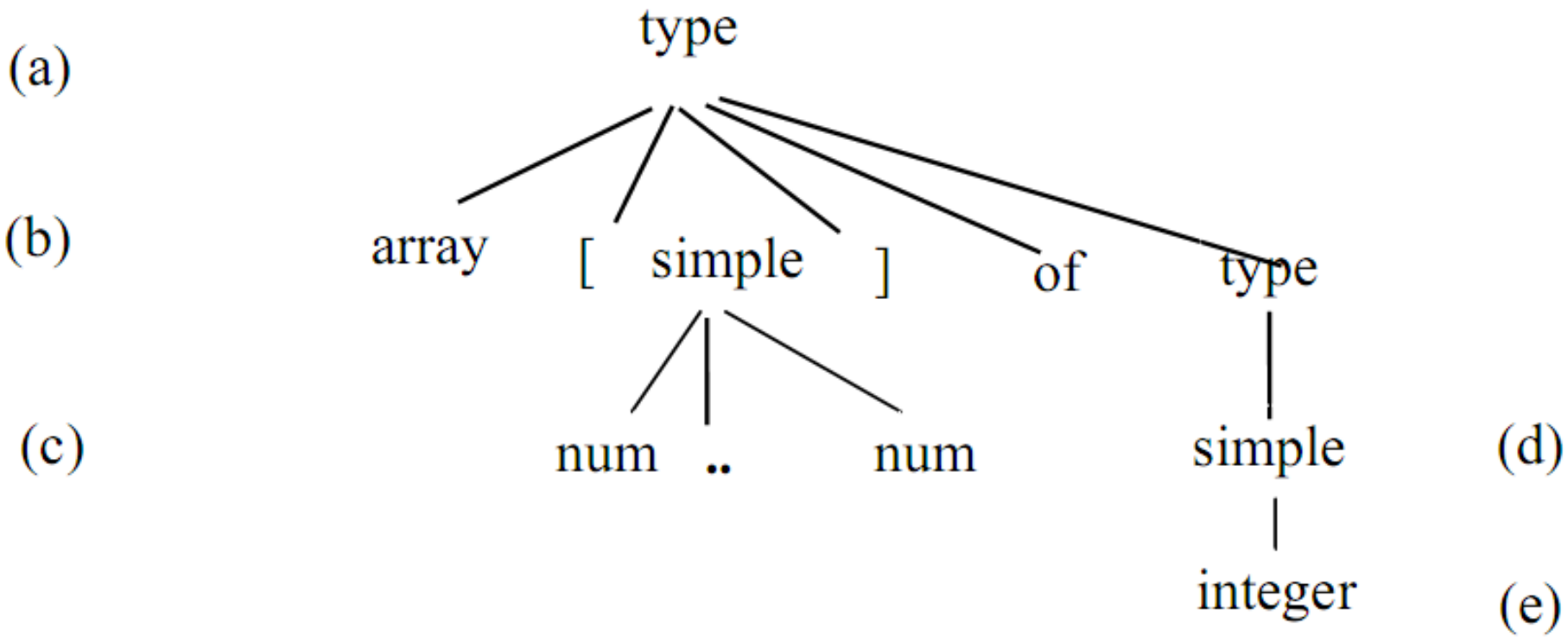
- Hãy xây dựng cây phân tích cho câu:

  array [num .. num] of integer

# 3.1 Phân tích cú pháp từ trên xuống (Top - Down Parsing)

- Phân tích trên xuống bắt đầu bởi nút gốc, nhãn là ký hiệu chưa kết thúc bắt đầu và lặp lại việc thực hiện hai bước sau đây:
  - 1. Tại nút n, nhãn là ký hiệu chưa kết thúc A, chọn một trong những luật sinh của A và xây dựng các con của n cho các ký hiệu trong vế phải của luật sinh.
  - 2. Tìm nút kế tiếp mà tại đó một cây con sẽ được xây dựng.

# 3.1 Phân tích cú pháp từ trên xuống (Top - Down Parsing)




*Minh họa quá trình phân tích cú pháp từ trên xuống*

# 3.2 Phân tích cú pháp dự đoán (Predictive Parsing)

- Dạng đặc biệt của phân tích cú pháp từ trên xuống là phương pháp đoán nhận trước. Phương pháp này sẽ nhìn trước một ký hiệu nhập để quyết định chọn thủ tục cho ký hiệu không kết thúc tương ứng.
- **Ví dụ**. Cho văn phạm G:

  P: S -> xA

  A -> z |yA
- Dùng văn phạm G để phân tích câu nhập xyyz

# 3.2 Phân tích cú pháp dự đoán (Predictive Parsing)

**Bảng 2.1.** *Các bước phân tích cú pháp của câu xyyz*

| Luật áp dụng | Chuỗi nhập |
|---|---|
| S | xyyz |
| xA | xyyz |
| yA | yyz |
| A | yz |
| yA | yz |
| A | z |
| z | z |
| - | - |

S -> xA

A -> z |yA

# 3.2 Phân tích cú pháp dự đoán (Predictive Parsing)

- **Ví dụ**. Cho văn phạm với các luật sinh như sau:

  S -> A |B

  A -> xA|y

  B -> xB|z

# 3.2 Phân tích cú pháp dự đoán (Predictive Parsing)

**Bảng 2.2.** *Phân tích cú pháp cho câu xxxz không thành công*

| Luật áp dụng | Chuỗi nhập |
|---|---|
| S | xxxz |
| A | xxxz |
| xA | xxxz |
| A | xxz |
| xA | xxz |
| A | xz |
| xA | xz |
| A | z |

S -> A |B

A -> xA|y

B -> xB|z

# 3.2 Phân tích cú pháp dự đoán (Predictive Parsing)

- *Điều kiện 1* : $A \rightarrow \xi_1 \mid \xi_2 \mid \ldots \mid \xi_n$
- *Định nghĩa:*

$first(\xi_i) = \{s \mid s \text{ là ký hiệu kết thúc và } \xi \Rightarrow s...\}$

Điều kiện 1 được phát biểu như sau :

$A \rightarrow \xi_1 \mid \xi_2 \mid \ldots \mid \xi_n$

$first(\xi_i) \cap first(\xi_j) = \varnothing$ với $i \neq j$

*Lưu ý*: 1. $first(a\xi) = \{a\}$

2. Nếu $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$; thì

$first(A\xi) = first(\alpha_1) \cup first(\alpha_2) \ldots \cup first(\alpha_n)$

# 3.3 Loại bỏ đệ quy trái

- Một bộ phân tích cú pháp đệ quy xuống có thể sẽ dẫn đến một vòng lặp vô tận nếu gặp một luật sinh đệ qui trái dạng **E** → **E** + T.

⇒Thêm vào một ký hiệu chưa kết thúc mới

- Ví dụ: A → Aα | β, thêm R:
  - A → β R
  - R → α R | ε

# 3.3 Loại bỏ đệ quy trái

- **Ví dụ**: Xét luật sinh đệ quy trái : $E \rightarrow E + T \mid T$
- Sử dụng quy tắc khử đệ quy trái nói trên với : $A \cong E$, $\alpha \cong + T$, $\beta \cong T$ . Luật sinh trên có thể biến đổi tương đương thành tập luật sinh :
  - $E \rightarrow T\ R$
  - $R \rightarrow + T\ R \mid \varepsilon$

# 4. Một Chương trình dịch Biểu thức đơn giản

- Xây dựng một bộ dịch trực tiếp cú pháp mà nó dịch một biểu thức số học đơn giản từ trung tố sang hậu tố.
- Biểu thức xét là các chữ số viết cách bởi + và –

expr → expr + term { print ('+') }

expr → expr - term { print ('-') }

expr → term

term → 0 { print ('0') }

...

term → 9 { print ('9') }

# 4. Một Chương trình dịch Biểu thức đơn giản

- Khử đệ quy trái: ký hiệu chưa kết thúc rest

expr → term rest

rest → + term { print('+') } rest | - term {print('-') rest | ε

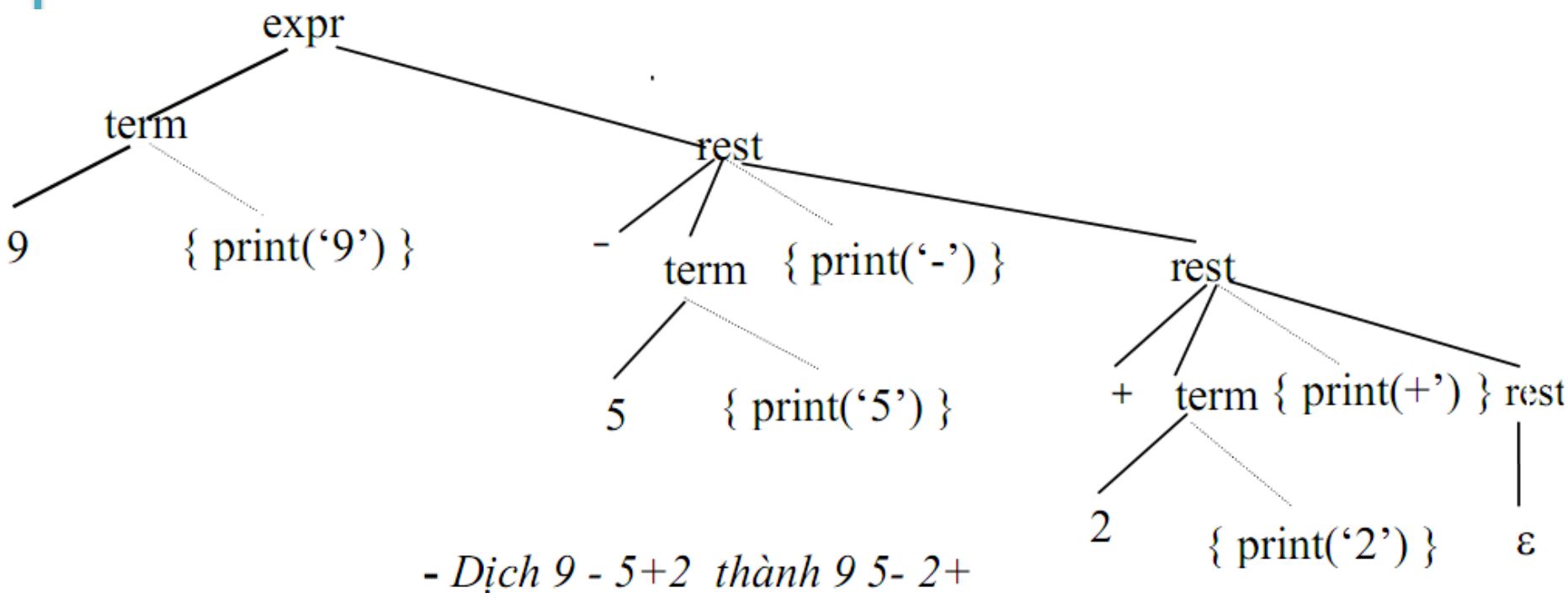
term → 0 { print('0') }

term → 1 { print('1') }

...

term → 9 { print('9') }

# 4. Một Chương trình dịch Biểu thức đơn giản



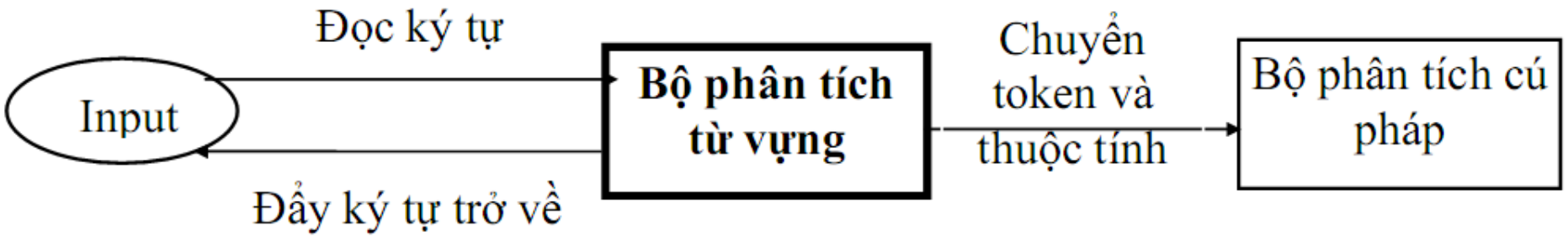

- *Dịch 9 - 5+2 thành 9 5- 2+*

# 5. Phân tích từ vựng

- Loại bỏ các khoảng trắng và dòng chú thích
- Nhận biết các hằng
- Nhận dạng các danh biểu và từ khóa

# 5. Phân tích từ vựng

- Giao diện của bộ phân tích từ vựng



# 5. Phân tích từ vựng