CHƯƠNG III
PHÂN TÍCH TỪ VỰNG

Nội dung chính:
Chương này trình bày các kỹ thuật xác định và cài đặt *bộ phân tích từ vựng*. Kỹ thuật đơn giản để xây dựng một bộ phân tích từ vựng là xây dựng các *lược đồ* - automata hữu hạn xác định (Deterministic Finite Automata - DFA) hoặc không xác định (Nondeterministic Finite Automata - NFA) – mô tả cấu trúc của các *thẻ từ* (token) của ngôn ngữ nguồn và sau đó dịch "thủ công" chúng sang chương trình nhận dạng các token. Một kỹ thuật khác nhằm tạo ra bộ phân tích từ vựng là sử dụng *Lex* – ngôn ngữ hành động theo *mẫu* (pattern). Trước tiên, người thiết kế trình biên dịch phải mô tả các mẫu được xác định bằng các biểu thức chính quy, sau đó sử dụng trình biên dịch của Lex để tự động tạo ra một bộ định dạng automata hữu hạn hiệu quả (bộ phân tích từ vựng). Các mô tả và cách thức hoạt động chi tiết của công cụ Lex được trình bày rõ hơn trong phần phụ lục A.

Mục tiêu cần đạt:
Sau khi học xong chương này, sinh viên phải nắm được các kỹ thuật tạo ra bộ phân tích từ vựng. Cụ thể,

- Xây dựng các lược đồ cho các biểu thức chính quy mô tả ngôn ngữ cần được viết trình biên dịch. Sau đó chuyển đổi chúng sang một chương trình phân tích từ vựng.
- Sử dụng công cụ có sẵn Lex để sinh ra bộ phân tích từ vựng.

Kiến thức cơ bản:
Sinh viên phải có các kiến thức về:

- DFA và NFA. Các automata hữu hạn xác định và không xác định này được sử dụng để nhận dạng chính xác ngôn ngữ mà các biểu thức chính quy có thể biểu diễn.
- Cách chuyển đổi từ NFA sang DFA nhằm làm đơn giản hóa quá trình cài đặt bộ phân tích từ vựng.

Tài liệu tham khảo:

[1] **Automata and Formal Language. An Introduction** – Dean Kelley – Prentice Hall, Englewood Cliffs, New Jersey 07632.

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

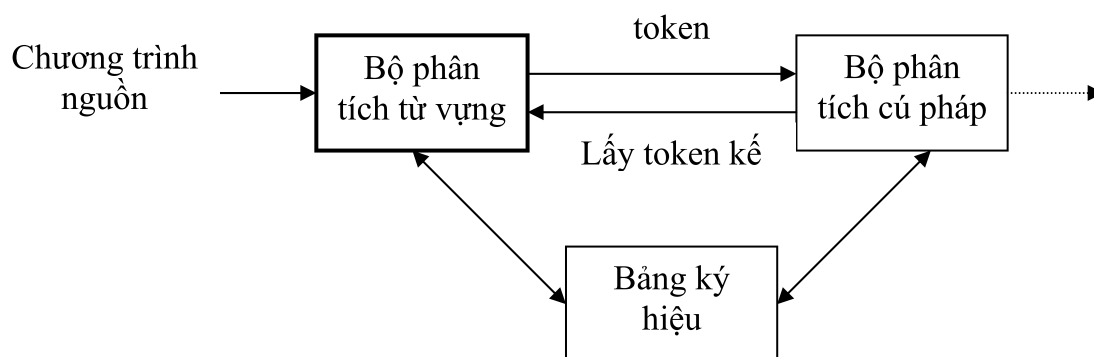
[3] **Compiler Design** – Reinhard Wilhelm, Dieter Maurer - Addison - Wesley Publishing Company, 1996.

[4] **Design of Compilers : Techniques of Programming Language Translation** - Karen A. Lemone - CRC Press, Inc, 1992.

[5] **Modern Compiler Implementation in C -** Andrew W. Appel - Cambridge University Press, 1997.

I. VAI TRÒ CỦA BỘ PHÂN TÍCH TỪ VỰNG

Phân tích từ vựng là giai đoạn đầu tiên của mọi trình biên dịch. Nhiệm vụ chủ yếu của nó là đọc các ký hiệu nhập rồi tạo ra một chuỗi các token được sử dụng bởi bộ phân tích cú pháp. Sự tương tác này được thể hiện như hình sau, trong đó bộ phân tích từ vựng được thiết kế như một thủ tục được gọi bởi bộ phân tích cú pháp, trả về một token khi được gọi.



Hình 3.1 - *Giao diện của bộ phân tích từ vựng*

1. Các vấn đề của giai đoạn phân tích từ vựng

Có nhiều lý do để tách riêng giai đoạn phân tích từ vựng với giai đoạn phân tích cú pháp:

1. Thứ nhất, nó làm cho việc thiết kế đơn giản và dễ hiểu hơn. Chẳng hạn, bộ phân tích cú pháp sẽ không phải xử lý các khoảng trắng hay các lời chú thích nữa vì chúng đã được bộ phân tích từ vựng loại bỏ.

2. Hiệu quả của trình biên dịch cũng sẽ được cải thiện, nhờ vào một số chương trình xử lý chuyên dụng sẽ làm giảm đáng kể thời gian đọc dữ liệu từ chương trình nguồn và nhóm các token.

3. Tính đa tương thích (mang đi dễ dàng) của trình biên dịch cũng được cải thiện. Đặc tính của bộ ký tự nhập và những khác biệt của từng loại thiết bị có thể được giới hạn trong bước phân tích từ vựng. Dạng biểu diễn của các ký hiệu đặc biệt hoặc là những ký hiệu không chuẩn, chẳng hạn như ký hiệu (trong Pascal có thể được cô lập trong bộ phân tích từ vựng.

2. Token, mẫu từ vựng và trị từ vựng

Khi nói đến bộ phân tích từ vựng, ta sẽ sử dụng các thuật ngữ *từ tố* (thẻ từ, token), *mẫu từ vựng* (pattern) và *trị từ vựng* (lexeme) với nghĩa cụ thể như sau:

- Từ tố (token) là các ký hiệu kết thúc trong văn phạm đối với một ngôn ngữ nguồn, chẳng hạn như: từ khóa, danh biểu, toán tử, dấu câu, hằng, chuỗi, ...

- Trị từ vựng (lexeme) của một token là một chuỗi ký tự biểu diễn cho token đó.

- Mẫu từ vựng (pattern) là qui luật mô tả một tập các trị từ vựng kết hợp với một token nào đó.

Một số ví dụ về cách dùng của các thuật ngữ này được trình bày trong bảng sau:

Token	Trị từ vựng minh họa	Mô tả của mẫu từ vựng
const	const	const
if	if	if
relation	<, <=, =, < >, >, >=	< hoặc <= hoặc = hoặc <> hoặc > hoặc >=
id	pi, count, d2	Mở đầu là chữ cái theo sau là chữ cái, chữ số
num	3.1416, 0, 5	Bất kỳ hằng số nào
literal	“ hello ”	Mọi chữ cái nằm giữa “ và “ ngoại trừ “

Hình 3.2 - *Các ví dụ về token*

3. Thuộc tính của token

Khi có nhiều mẫu từ vựng khớp với một trị từ vựng, bộ phân tích từ vựng trong trường hợp này phải cung cấp thêm một số thông tin khác cho các bước biên dịch sau đó. Do đó đối với mỗi token, bộ phân tích từ vựng sẽ đưa thông tin về các token vào các thuộc tính đi kèm của chúng. Các token có ảnh hưởng đến các quyết định phân tích cú pháp; các thuộc tính ảnh hưởng đến việc phiên dịch các thẻ từ. Token kết hợp với thuộc tính của nó tạo thành một bộ <*token, tokenval*>.

Ví dụ 3.1: Token và giá trị thuộc tính đi kèm của câu lệnh Fortran : E = M * C ** 2 được viết như một dãy các bộ sau:

< **id**, con trỏ trong bảng ký hiệu của E >

< **assign_op**, >

< **id**, con trỏ trong bảng ký hiệu của M >

< **mult_op**, >

< **id**, con trỏ trong bảng ký hiệu của C>

< **exp_op**, >

< **num**, giá trị nguyên 2 >

Chú ý rằng một số bộ không cần giá trị thuộc tính, thành phần đầu tiên là đủ để nhận dạng trị từ vựng.

4. Lỗi từ vựng

Chỉ một số ít lỗi được phát hiện tại bước phân tích từ vựng, bởi vì bộ phân tích từ vựng có nhiều cách nhìn nhận chương trình nguồn. Ví dụ chuỗi **fi** được nhìn thấy lần đầu tiên trong một chương trình C với ngữ cảnh : **fi (a == f (x))** ... Bộ phân tích từ vựng không thể biết đây là lỗi không viết đúng từ khóa **if** hay một danh biểu chưa được khai báo. Vì fi là một danh biểu hợp lệ nên bộ phân tích từ vựng phải trả về một token và để một giai đoạn khác sau đó xác định lỗi. Tuy nhiên, trong một vài tình huống phải khắc phục lỗi để phân tích tiếp. Chiến lược đơn giản nhất là "*phương thức hoảng sợ*" (panic mode): Các ký tự tiếp theo sẽ được xóa ra khỏi chuỗi nhập còn lại

cho đến khi tìm ra một token hoàn chỉnh. Kỹ thuật này đôi khi cũng gây ra sự nhầm lẫn cho giai đoạn phân tích cú pháp, nhưng nói chung là vẫn có thể sử dụng được.

Một số chiến lược khắc phục lỗi khác là:

1. Xóa đi một ký tự dư.
2. Xen thêm một ký tự bị mất.
3. Thay thế một ký tự không đúng bằng một ký tự đúng.
4. Chuyển đổi hai ký tự kế tiếp nhau.

II. LƯU TRỮ TẠM CHƯƠNG TRÌNH NGUỒN

Việc đọc từng ký tự trong chương trình nguồn có thể tiêu hao một số thời gian đáng kể do đó ảnh hưởng đến tốc độ dịch. Để giải quyết vấn đề này người ta đọc một lúc một chuỗi ký tự, lưu trữ vào trong vùng nhớ tạm - gọi là *bộ đệm* input (buffer). Tuy nhiên, việc đọc như vậy cũng gặp một số trở ngại do không thể xác định một chuỗi như thế nào thì chứa trọn vẹn một token? Phần này giới thiệu vài phương pháp đọc bộ đệm hiệu quả:

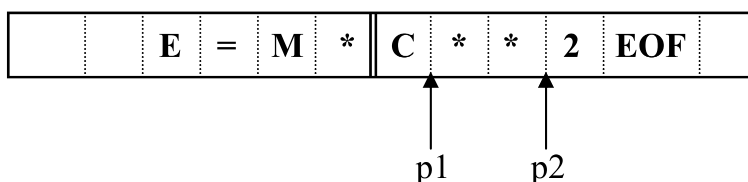
1. Cặp bộ đệm (Buffer Pairs)

Đối với nhiều ngôn ngữ nguồn, có một vài trường hợp bộ phân tích từ vựng phải đọc thêm một số ký tự trong chương trình nguồn vượt quá trị từ vựng cho một mẫu trước khi có thể thông báo đã so trùng được một token.

Trong phương pháp cặp bộ đệm, vùng đệm sẽ được chia thành hai nửa với kích thước bằng nhau, mỗi nửa chứa được N ký tự. Thông thường, N là số ký tự trên một khối đĩa, N bằng 1024 hoặc 4096.

Mỗi lần đọc, N ký tự từ chương trình nguồn sẽ được đọc vào mỗi nửa bộ đệm bằng một lệnh đọc (read) của hệ thống. Nếu số ký tự còn lại trong chương trình nguồn ít hơn N thì một ký tự đặc biệt eof được đưa vào buffer sau các ký tự vừa đọc để báo hiệu chương trình nguồn đã được đọc hết.

Sử dụng hai con trỏ dò tìm trong buffer. Chuỗi ký tự nằm giữa hai con trỏ luôn luôn là trị từ vựng hiện hành. Khởi đầu, cả hai con trỏ đặt trùng nhau tại vị trí bắt đầu của mỗi trị từ vựng. Con trỏ p1 (lexeme_beginning) - con trỏ bắt đầu trị từ vựng - sẽ giữ cố định tại vị trí này cho đến khi con trỏ p2 (forwar) - con trỏ tới - di chuyển qua từng ký tự trong buffer để xác định một token. Khi một trị từ vựng cho một token đã được xác định, con trỏ p1 dời lên trùng với p2 và bắt đầu dò tìm một trị từ vựng mới.



Hình 3.3 - *Cặp hai nửa vùng đệm*

Khi con trỏ p2 tới ranh giới giữa 2 vùng đệm, nửa bên phải được lấp đầy bởi N ký tự tiếp theo trong chương trình nguồn. Khi con trỏ p2 tới vị trí cuối bộ đệm, nửa bên trái sẽ được lấp đầy bởi N ký tự mới và p2 sẽ được dời về vị trí bắt đầu bộ đệm.

Phương pháp cặp bộ đệm này thường hoạt động rất tốt nhưng khi đó số lượng ký tự đọc trước bị giới hạn và trong một số trường hợp nó có thể không nhận dạng được token khi con trỏ p2 phải vượt qua một khoảng cách lớn hơn chiều dài vùng đệm.

Giải thuật hình thức cho hoạt động của con trỏ p2 trong bộ đệm :

```
if   p2 ở cuối nửa đầu   then
        begin
            Đọc vào nửa cuối;
            p2 := p2 + 1;
        end
else   if  p2 ở cuối của nửa cuối   then
            begin
                Đọc vào nửa đầu;
                Dời p2 về đầu bộ đệm ;
          end
        else   p2 := p2 + 1
```

2. Khóa cầm canh (Sentinel)

Phương pháp cặp bộ đệm đòi hỏi mỗi lần di chuyển p2 đều phải kiểm tra xem có phải đã hết một nửa buffer chưa nên kém hiệu quả vì phải hai lần kiểm tra. Để khắc phục điều này, mỗi lần chỉ đọc N-1 ký tự vào mỗi nửa buffer còn ký tự thứ N là một ký tự đặc biệt, thường là eof. Như vậy chúng ta đã rút ngắn một lần kiểm tra.

			E	=	M	*	eof	C	*	*	2	eof	

p1 (↑ tại C)　　p2 (↑ tại * thứ hai)

Hình 3.4 - *Khóa cầm canh eof tại cuối mỗi vùng đệm*

Giải thuật hình thức cho hoạt động của con trỏ p2 trong bộ đệm :

```
p2 := p2 + 1;
if   p2↑ = eof   then
        begin
            if   p2 ở cuối của nửa đầu   then
                begin
                        Đọc vào nửa cuối;
                        p2 := p2 + 1;
                end
            else   if   p2 ở cuối của nửa sau   then
```

```
            begin
                    Đọc vào nửa đầu;
                    Dời p2 vào đầu của nửa đầu;
            end
        else        /* EOF ở giữa vùng đệm chỉ hết chương trình nguồn */
            kết thúc phân tích từ vựng;
    end
```

III. ĐẶC TẢ TOKEN (Specification of Token)

1. Chuỗi và ngôn ngữ

Chuỗi là một tập hợp hữu hạn các ký tự. Độ dài chuỗi là số các ký tự trong chuỗi. Chuỗi rỗng ε là chuỗi có độ dài 0.

Ngôn ngữ là tập hợp các chuỗi. Ngôn ngữ có thể chỉ bao gồm một chuỗi rỗng ký hiệu là ∅.

2. Các phép toán trên ngôn ngữ

Cho 2 ngôn ngữ L và M :

- **Hợp** của L và M : $L \cup M = \{ s \mid s \in L \text{ hoặc } s \in M \}$
- **Ghép** (concatenation) của L và M: $LM = \{ st \mid s \in L \text{ và } t \in M \}$
- **Bao đóng Kleen** của L: $L^* = \cup_{i=0}^{\infty} L^i$

  (Ghép của 0 hoặc nhiều L)
- **Bao đóng dương** (positive closure) của L: $L^+ = \cup_{i=1}^{\infty} L^i$

  (Ghép của 1 hoặc nhiều L)

Ví dụ 3.2: L = {A, B, ..., Z, a, b, ..., z }

D = { 0, 1, , ..., 9 }

1. $L \cup D$ là tập hợp các chữ cái và số.
2. LD là tập hợp các chuỗi bao gồm một chữ cái và một chữ số.
3. L^4 là tập hợp tất cả các chuỗi 4 chữ cái.
4. L^* là tập hợp tất cả các chuỗi của các chữ cái bao gồm cả chuỗi rỗng.
5. $L(L \cup D)^*$ là tập hợp tất cả các chuỗi mở đầu bằng một chữ cái theo sau là chữ cái hay chữ số
6. D^+ là tập hợp tất cả các chuỗi gồm một hoặc nhiều chữ số.

3. Biểu thức chính quy (Regular Expression)

Trong Pascal, một danh biểu là một phần tử của tập hợp $L (L \cup D)^*$. Chúng ta có thể viết: danhbiểu = letter (letter | digit)* - Đây là một biểu thức chính quy.

Biểu thức chính quy được xây dựng trên một tập hợp các luật xác định. Mỗi biểu thức chính quy r đặc tả một ngôn ngữ L(r).

Sau đây là các luật xác định biểu thức chính quy trên tập Alphabet ∑.

1. ε là một biểu thức chính quy đặc tả cho một chuỗi rỗng {ε }.
2. Nếu a ∈ ∑ thì a là biểu thức chính quy r đặc tả tập hợp các chuỗi {a}
3. Giả sử r và s là các biểu thức chính quy đặc tả các ngôn ngữ L(r) và L(s) ta có:
   a. (r) | (s) là một biểu thức chính quy đặc tả L(r) ∪ L(s)
   b. (r) (s) là một biểu thức chính quy đặc tả L(r)L(s).
   c. (r)* là một biểu thức chính quy đặc tả (L(r))*

   Quy ước:

   Toán tử bao đóng * có độ ưu tiên cao nhất và kết hợp trái.

   Toán tử ghép có độ ưu tiên thứ hai và kết hợp trái.

   Toán tử hợp | có độ ưu tiên thấp nhất và kết hợp trái.

Ví dụ 3.3: Cho ∑ = { a, b}

1. Biểu thức chính quy a | b đặc tả {a, b}
2. Biểu thức chính quy (a | b) (a | b) đặc tả tập hợp {aa, ab, ba, bb}.Tập hợp này có thể được đặc tả bởi biểu thức chính quy tương đương sau: aa | ab | ba | bb.
3. Biểu thức chính quy a* đặc tả { ε, a, aa, aaa, ... }
4. Biểu thức chính quy (a | b)* đặc tả {(, a, b, aa,bb, ...}. Tập này có thể đặc tả bởi (a*b*)*.
5. Biểu thức chính quy a | a* b đặc tả {a, b, ab, aab,... }

Hai biểu thức chính quy cùng đặc tả một tập hợp ta nói rằng chúng tương đương và viết r = s.

4. Các tính chất đại số của biểu thức chính quy

Biểu thức chính quy cũng tuân theo một số luật đại số và có thể dùng các luật này để biến đổi biểu thức thành những dạng tương đương. Bảng sau trình bày một số luật đại số cho các biểu thức chính quy r, s và t.

Tính chất	Mô tả
r | s = s | r	| có tính chất giao hoán
r | (s | t) = (r | s) | t	| có tính chất kết hợp
(rs) t = r (st)	Phép ghép có tính chất kết hợp
r (s | t) = rs | rt (s | t) r = sr | tr	Phép ghép phân phối đối với phép |
εr = r	ε là phần tử đơn vị của phép ghép

$r\varepsilon = r$	
$r^* = (r \mid \varepsilon)^*$	Quan hệ giữa r và ε
$r^{**} = r^*$	* có hiệu lực như nhau

Hình 3.5 - *Một số tính chất đại số của biểu thức chính quy*

5. Định nghĩa chính quy (Regular Definitions)

Định nghĩa chính quy là một chuỗi các định nghĩa có dạng :

d1 → r1   di là một tên,

d2 → r2   ri là một biểu thức chính quy.

...

dn → rn

Ví dụ 3.4: Tập hợp các danh biểu trong Pascal là một tập hợp các chuỗi chữ cái và số, mở đầu bằng một chữ cái. Định nghĩa chính quy của tập đó là:

letter → A | B | ...| Z | a | b |...| z

digit → 0 | 1 | ...| 9

id → letter (letter | digit)*

Ví dụ 3.5 : Các số không dấu trong Pascal là các chuỗi 5280, 39.37, 6.336E4 hoặc 1.894E-4. Định nghĩa chính quy sau đặc tả tập các số này là :

digit → 0 | 1 |...| 9

digits → digit digit*

optional_fraction → . digits | ε

optional_exponent → (E (+ | - | ε) digits) | ε

num → digits optional_fraction optional_exponent

6. Ký hiệu viết tắt

Người ta quy định các ký hiệu viết tắt cho thuận tiện trong việc biểu diễn như sau:

1. Một hoặc nhiều: dùng dấu +
2. Không hoặc một: dùng dấu ?

Ví dụ 3.6: r | ε được viết tắt là r ?

Ví dụ 3.7: Viết tắt cho định nghĩa chính quy tập hợp số **num** trong ví dụ 3.5

digit → 0 | 1 |... | 9

digits → digit +

optional_fraction → (. digits) ?

optional_exponent → (E (+ | -) ? digits) ?

num → digits optional_fraction optional_exponent

3. Lớp ký tự

[abc] = a | b | c

[a - z] = a | b |... | z

Sử dụng lớp ký hiệu chúng ta có thể mô tả danh biểu như là một chuỗi sinh ra bởi biểu thức chính quy :

[A - Z a - z] [A - Z a - z 0 - 9]*

IV. NHẬN DẠNG TOKEN

Trong suốt phần này, chúng ta sẽ dùng ngôn ngữ được tạo ra bởi văn phạm dưới đây làm thí dụ minh họa :

stmt → **if** expr **then** stmt

| **if** expr **then** stmt **else** stmt

| ε

expr → term **relop** term

| term

term → **id**

| **num**

Trong đó các ký hiệu kết thúc if, then, else, relop, id, num được cho bởi định nghĩa chính quy sau:

if → if

then → then

else → else

relop → < | <= | = | <> | > | >=

id → **letter (letter | digit)** *

num → **digit** + (**. digit** +) ? (**E** (+ | **-**) ? **digit** +) ?

Định nghĩa chính quy của các khoảng trắng ws (white space)

delim → **blank | tab | newline**

ws → **delim**+

Mục đích của chúng ta là xây dựng một bộ phân tích từ vựng có thể định vị được từ tố cho các token kế tiếp trong vùng đệm và tạo ra output là một cặp token thích hợp và giá trị thuộc tính của nó bằng cách dùng mẫu biểu thức chính quy cho các token như sau:

Biểu thức chính quy	Token	Trị thuộc tính
ws	**-**	-
if	**if**	-
then	**then**	-

else	**else**	-
id	**id**	con trỏ trong bảng ký hiệu
num	**num**	giá trị số
<	**relop**	LT (Less Than)
<=	**relop**	LE (Less Or Equal)
=	**relop**	EQ (Equal)
< >	**relop**	NE (Not Equal)
>	**relop**	GT (Greater Than)
>=	**relop**	GE (Greater Or Equal)

Hình 3.6 - *Mẫu biểu thức chính quy cho một số token*

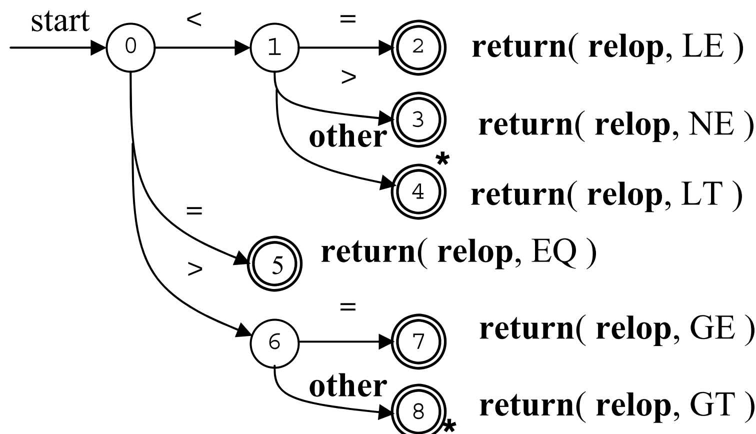
1. Sơ đồ dịch

Để dễ dàng nhận dạng token, chúng ta xây dựng cho mỗi token một *sơ đồ dịch* (translation diagram). Sơ đồ dịch bao gồm các trạng thái (state) ký hiệu bởi vòng tròn và các cạnh mũi tên nối các trạng thái.

Nói chung thường có nhiều sơ đồ dịch, mỗi sơ đồ đặc tả một nhóm token. Nếu xảy ra thất bại khi chúng ta đang đi theo một sơ đồ dịch thì chúng ta dịch lui con trỏ tới về nơi nó đã ở trong trạng thái khởi đầu của sơ đồ này rồi kích hoạt sơ đồ dịch tiếp theo. Do con trỏ đầu trị từ vựng và con trỏ tới cùng chỉ đến một vị trí trong trạng thái khởi đầu của sơ đồ, con trỏ tới sẽ được dịch lui lại để chỉ đến vị trí được con trỏ đầu trị từ vựng chỉ tới. Nếu xảy ra thất bại trong tất cả mọi sơ đồ dịch thì xem như một lỗi từ vựng đã được phát hiện và chúng ta sẽ khởi động một thủ tục khắc phục lỗi.

Phần dưới đây trình bày một số sơ đồ dịch nhận dạng các token trong văn phạm ví dụ trên.

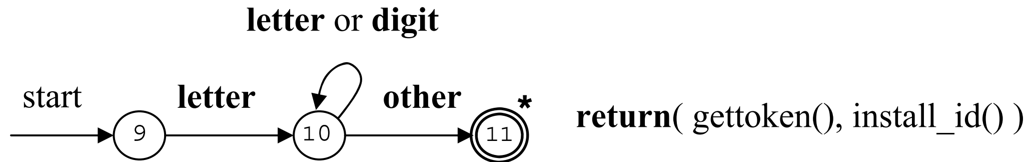
Sơ đồ dịch nhận dạng cho token **relop**:



Hình 3.7 - *Sơ đồ dịch cho các toán tử quan hệ*

Chúng ta dùng ký hiệu * để chỉ ra những trạng thái mà chúng ta đã đọc quá một ký tự, cần phải quay lui con trỏ lại.

Sơ đồ dịch nhận dạng token **id**:

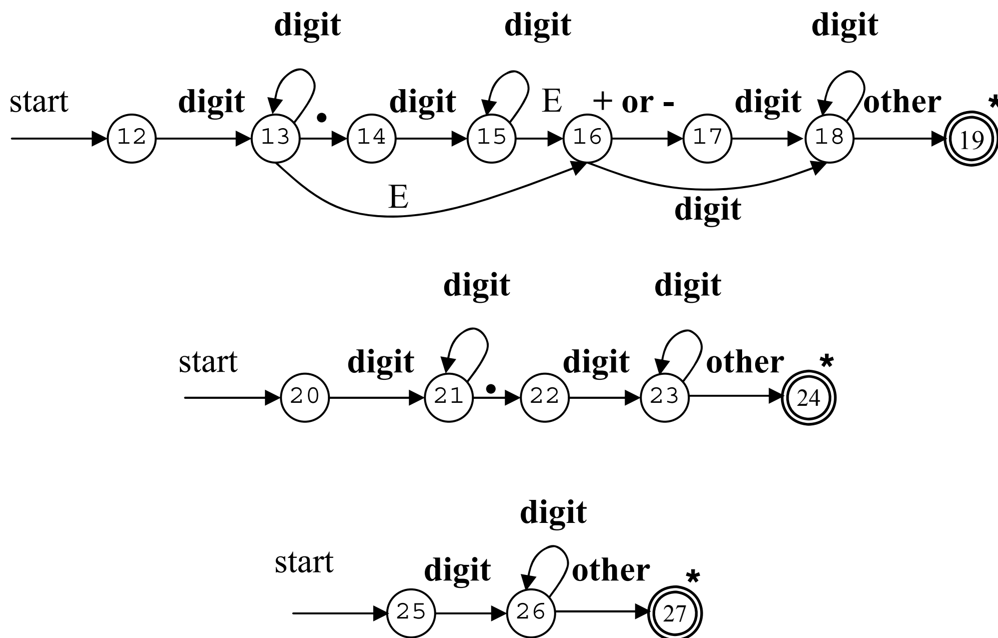


Hình 3.8 - *Sơ đồ dịch cho các danh biểu và từ khóa*

Một kỹ thuật đơn giản để tách từ khóa ra khỏi các danh biểu là khởi tạo bảng ký hiệu lưu trữ thông tin về danh biểu một cách thích hợp. Đối với các token cần nhận dạng trong văn phạm này, chúng ta cần nhập các chuỗi **if, then** và **else** vào bảng ký hiệu trước khi đọc các ký hiệu trong bộ đệm nguyên liệu. Đồng thời ghi chú trong bảng ký hiệu để trả về token đó khi một trong các chuỗi này được nhận ra. Sử dụng các hàm gettoken( ) và install_id( ) tương ứng để nhận token và các thuộc tính trả về.

Sơ đồ dịch nhận dạng token **num**:

Một số vấn đề sẽ nảy sinh khi chúng ta xây dựng bộ nhận dạng cho các số không dấu. Trị từ vựng cho một token num phải là trị từ vựng dài nhất có thể được. Do đó, việc thử nhận dạng số trên các sơ đồ dịch phải theo thứ tự từ sơ đồ nhận dạng số dài nhất.



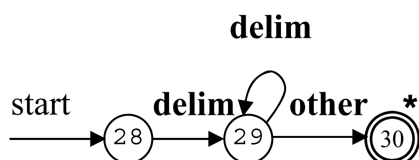
Hình 3.9 - *Sơ đồ dịch cho các số không dấu trong Pascal*

Có nhiều cách để tránh các đối sánh dư thừa trong các sơ đồ dịch trên. Một cách là viết lại các sơ đồ dịch bằng cách tổ hợp chúng thành một - một công việc nói chung là không đơn giản lắm. Một cách khác là thay đổi cách đáp ứng với thất bại trong qua trình duyệt qua một sơ đồ. Phương pháp được sử dụng ở đây là cho phép ta vượt qua nhiều trạng thái kiểm nhận và quay trở lại trạng thái kiểm nhận cuối cùng đã đi qua khi thất bại xảy ra.

Sơ đồ dịch nhận dạng khoảng trắng **ws (white space)**:

Việc xử lý các khoảng trắng ws không hoàn toàn giống như các mẫu nói trên bởi vì không có gì để trả về cho bộ phân tích cú pháp khi tìm thấy các khoảng trắng trong

chuỗi nhập. Và do đó, thao tác đơn giản cho việc dò tìm trên sơ đồ dịch khi phát hiện khoảng trắng là trở lại trạng thái bắt đầu của sơ đồ dịch đầu tiên để tìm một mẫu khác.



Hình 3.10 - *Sơ đồ dịch cho các khoảng trắng*

2. Cài đặt một sơ đồ dịch

Dãy các sơ đồ dịch có thể được chuyển thành một chương trình để tìm kiếm token được đặc tả bằng các sơ đồ. Mỗi trạng thái tương ứng với một đoạn mã chương trình. Nếu có các cạnh đi ra từ trạng thái thì đọc một ký tự và tùy thuộc vào ký tự đó mà đi đến trạng thái khác. Ta dùng hàm **nextchar( )** đọc một ký tự từ trong bộ đệm input và con trỏ p2 di chuyển sang phải một ký tự. Nếu không có một cạnh đi ra từ trạng thái hiện hành phù hợp với ký tự vừa đọc thì con trỏ p2 phải quay lại vị trí của p1 để chuyển sang sơ đồ dịch kế tiếp. Hàm **fail( )** sẽ làm nhiệm vụ này. Nếu không có sơ đồ nào khác để thử, fail( ) sẽ gọi một thủ tục khắc phục lỗi.

Để trả về các token, chúng ta dùng một biến tòan cục **lexical_value**. Nó được gán cho các con trỏ được các hàm **install_id( )** và **install_num( )** trả về, tương ứng khi tìm ra một danh biểu hoặc một số. Lớp token được trả về bởi thủ tục chính của bộ phân tích từ vựng có tên là **nexttoken( )**.

```
int state = 0, start = 0;
int lexical_value;          /* để "trả về" thành phần thứ hai của token */


int fail ( )
{
   forward = token_beginning;
   switch  (start)  {
            case 0 : start = 9; break;
            case 9 : start = 12; break;
            case 12 : start = 20; break;
            case 20 : start = 25; break;
            case 25 : recover ( ); break;
            default : / * lỗi trình biên dịch */
  }
return     start;
}


token      nexttoken ( )
```

```
{   while (1)   {
    switch   (state)    {
    case 0 : c = nextchar ( ) ;     / * c là ký hiệu đọc trước */
      if   ( c = = blank  || c = = tab  || c = = newline )  {
            state = 0;
            lexeme_beginning ++ ;        / * dịch con trỏ đến đầu trị từ vựng */
      }
    else if  (c = = ‘ < ’)   state = 1;
    else if  (c = = ‘ = ’)   state = 5;
    else if  (c = = ‘ > ’)   state = 6;
    else   state = fail ( ) ; break ;

    . . .      / * các trường hợp 1- 8 ở đây */

[     case 9 : c = nextchar ( ) ;
    if  (isletter (c))          state=10;
    else      state = fail ( ) ; break ;
  case 10 : c = nextchar ( ) ;
    if  (isletter (c))          state=10;
    else      if  (isdigit(c))            state = 10 ;
              else     state = 11 ; break ;
  case 11 : retract (1) ; install_id ( ) ;
              return  (gettoken ( ));

    . . .      / * các trường hợp 12 - 24 ở đây */

  case 25 : c = nextchar ( ) ;
    if  (isdigit (c))           state=26;
    else      state = fail ( ) ; break ;
  case 26 : c = nextchar ( ) ;
    if  (isdigit (c))           state=26;
    else      state = 27 ; break ;
  case 27 : retract (1) ; install_num ( ) ;
              return  (NUM);
```

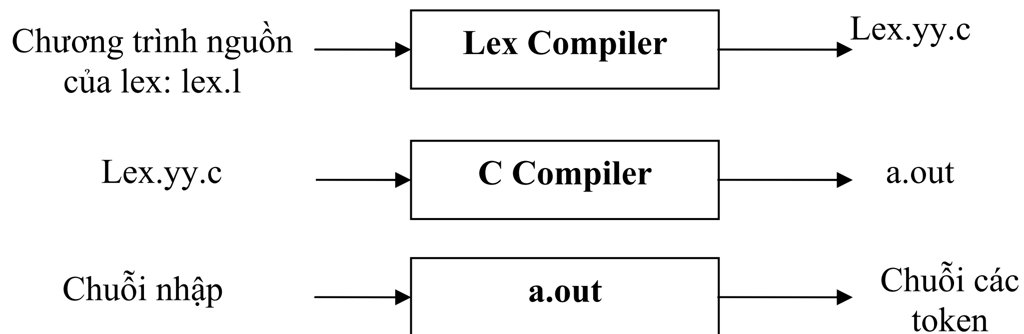
```
      }
    }
}
```

V. NGÔN NGỮ ĐẶC TẢ CHO BỘ PHÂN TÍCH TỪ VỰNG

1. Bộ sinh bộ phân tích từ vựng

Có nhiều công cụ để xây dựng bộ phân tích từ vựng dựa vào các biểu thức chính quy. **Lex** là một công cụ được sử dụng rộng rãi để tạo bộ phân tích từ vựng.

Trước hết đặc tả cho một bộ phân tích từ vựng được chuẩn bị bằng cách tạo ra một chương trình **lex.l** trong ngôn ngữ lex. Trình biên dịch Lex sẽ dịch lex.l thành một chương trình C là **lex.yy.c**. Chương trình này bao gồm các đặc tả về sơ đồ dịch được xây dựng từ các biểu thức chính quy của **lex.l**, kết hợp với các thủ tục chuẩn nhận dạng trị từ vựng. Các hành vi kết hợp với biểu thức chính quy trong **lex.l** là các đoạn chương trình C được chuyển sang **lex.yy.c**. Cuối cùng trình biên dịch C sẽ dịch lex.yy.c thành chương trình đối tượng **a.out**, đó là bộ phân tích từ vựng có thể chuyển dòng nhập thành chuỗi các token.



Hình 3.11 - *Tạo ra bộ phân tích từ vựng bằng Lex*

Chú ý: Những điều ta nói trên là nói về lex trong UNIX. Ngày nay có nhiều version của lex như Lex cho Pascal hoặc Javalex.

2. Đặc tả lex

Một chương trình lex bao gồm 3 thành phần:

Khai báo

%%

Quy tắc dịch

%%

Các thủ tục phụ

Phần khai báo bao gồm khai báo biến, hằng và các định nghĩa chính quy.

Phần quy tắc dịch cho các lệnh có dạng:

p1 {action 1 }

p2 {action 2 }

. . .

pn {action n }

Trong đó pi là các biểu thức chính quy, action i là đoạn chương trình mô tả hành động của bộ phân tích từ vựng thực hiện khi pi tương ứng phù hợp với trị từ vựng. Trong lex các đoạn chương trình này được viết bằng C nhưng nói chung có thể viết bằng bất cứ ngôn ngữ nào.

Các thủ tục phụ là sự cài đặt các hành động trong phần 2.

Ví dụ 3.8: Sau đây trình bày một chương trình Lex nhận dạng các token của văn phạm đã nêu ở phần trước và trả về token được tìm thấy.

```
%{
/* định nghĩa các hằng
LT, LE, EQ, NE, GT, GE, IF, THEN, ELSE, ID, NUMBER, RELOP */
}%
/* định nghĩa chính quy */
delim       [\t\n]
ws          {delim}+
letter      [A - Za -  z]
digit       [0 - 9]
id          {letter}({letter}| {digit})*
number      {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?
%%
{ws}        {/* Không có action, không có return */}
if          {return(IF); }
then        {return(THEN); }
else        {return(ELSE); }
{id}        {yylval = install_id( ); return(ID) }
{number}    {yylval = install_num( ); return(NUMBER) }
“< ”        {yylval = LT; return(RELOP) }
“<= “       {yylval = LE; return(RELOP) }
“= “        {yylval = EQ; return(RELOP) }
“<> “       {yylval = NE; return(RELOP) }
“> “        {yylval = GT; return(RELOP) }
“>= “       {yylval = GE; return(RELOP) }
%%
```

```
install_id ( ) {
        /* Thủ tục phụ cài id vào trong bảng ký hiệu */
}
install_num ( ) {
        /* Thủ tục phụ cài một số vào trong bảng ký hiệu */
}
```

BÀI TẬP CHƯƠNG III

3.1. Xác định bộ chữ cái của các ngôn ngữ sau:

a) Pascal

b) C

c) LISP

3.2. Hãy xác định các trị từ vựng có thể hình thành các token trong các đoạn chương trình sau:

a) **PASCAL**

```
function max (i, j :integer) : integer;
{ Trả về  số nguyên lớn hơn trong 2 số i và j }
    begin
        i > j then max : = i
                    else max  : = j;
    end;
```

b) **C**

```
int max (i, j) int i, j;       /* Trả về  số nguyên lớn hơn trong 2 số i và j */
{ return i > j ? i : j
}
```

c) **FORTRAN 77**

```
FUNCTION MAX (i, j)
C   Trả về  số nguyên lớn hơn trong 2 số i và j
IF ( I .GT. J) THEN
        MAX = I
ELSE
        MAX = J
END IF
RETURN
```

3.3. Viết một chương trình Lex sao chép một tập tin, thay các chuỗi khoảng trắng thành một khoảng trắng duy nhất.

3.4. Viết một đặc tả Lex cho các token của ngôn ngữ Pascal và dùng trình biên dịch Lex để xây dựng một bộ phân tích từ vựng cho Pascal.