

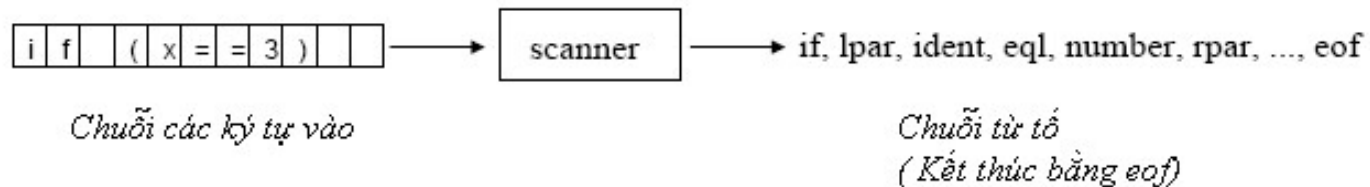


Bài 5

Bộ phân tích từ vựng

ONE LOVE. ONE FUTURE.

- Phát hiện các từ tố



- Bỏ qua các ký tự không cần thiết
 - Khoảng trống
 - Dấu tab
 - Ký tự xuống dòng (CR,LF)
 - Chú thích

- Có thể dùng luật cú pháp để mô tả các từ tổ

```
ident =    letter {letter | digit}.  
number =  digit {digit}.  
if =      "if" "f".  
eq =      "=" "=".  
...
```

- Tại sao không xử lý các luật này trong giai đoạn phân tích cú pháp ?

Xử lý các luật từ vựng trong bộ phân tích cú pháp ?

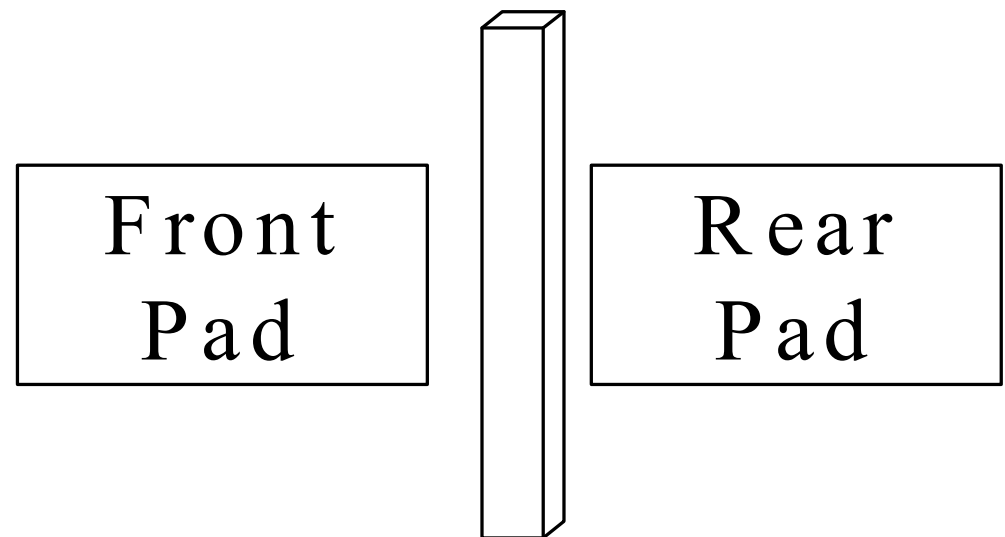
- Làm cho bộ phân tích cú pháp trở nên quá phức tạp
 - Phân biệt định danh và từ khoá
 - Phải có những luật phức tạp để xử lý chuỗi các ký tự không cần thiết (khoảng trống, tab, chú thích)

- Số nguyên
- Định danh
- Từ khóa: begin, end, if, then, while, do, call, const, var, procedure, program, type, function, of, integer, char, else, for, to, array
- Hằng ký tự
- Dấu phép toán:
 - số học
+ - */
 - so sánh
= != < > <= >=
- Dấu phân cách
() . : ; (.)
- Dấu phép gán :=

- Ôtômat hữu hạn (máy hữu hạn trạng thái là mô hình được ứng dụng trong nhiều lĩnh vực như kỹ thuật, mạng, xử lý ngôn ngữ
- Nguyên tắc làm việc: Thay đổi trạng thái căn cứ vào trạng thái hiện hành và tín hiệu điều khiển
- Trong kỹ thuật, ôôtômat hữu hạn được dùng để điều khiển máy bán hàng tự động, thang máy, biểu diễn mạch, mô tả giao thức truyền thông...
- Trong compiler, ôôtômat hữu hạn là mô hình cho bộ phân tích từ vựng
- Bộ sinh phân tích từ vựng là một phần của bộ sinh compiler. Đầu vào là các biểu thức chính quy mô tả các luật từ vựng. Đầu ra là chương trình phân tích từ vựng. Quá trình biến đổi từ biểu thức chính quy sang code đòi hỏi qua nhiều bước trung gian. Tại mỗi bước trung gian, mô hình ôôtômat hữu hạn được dùng để biến đổi

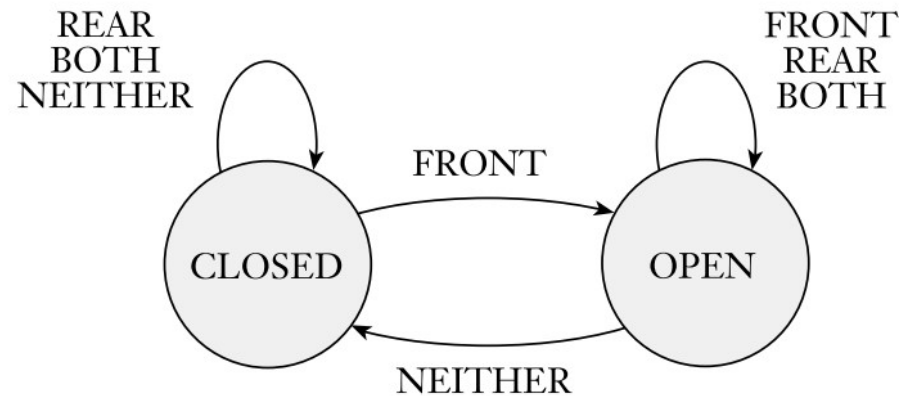
Xét một ví dụ: cửa một chiều

- Giả sử cần thiết kế cánh cửa tự động chỉ cho phép đi theo một chiều
- Cửa sẽ có hai vùng được kiểm soát bởi các sensor. Một vùng ở đằng trước và một vùng ở đằng sau cửa. Các vùng đó cho phép nhận biết có người đứng trước hay sau cửa
- Yêu cầu đặt ra là mọi người chỉ được phép đi từ vùng trước cửa sang vùng sau cửa khi cửa mở, không cho phép đi theo hướng ngược lại
- Mô hình ô tômat hữu hạn có thể cho phép giải quyết yêu cầu này



Cửa một chiều

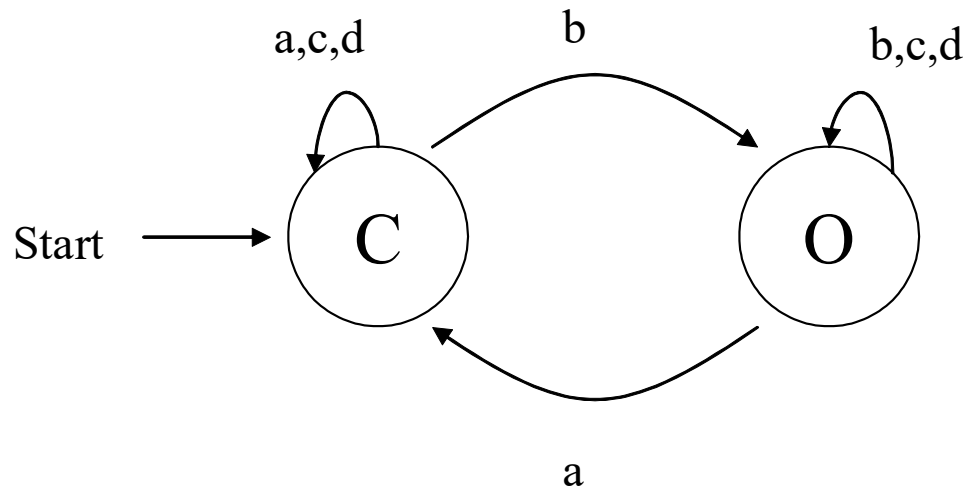
- Trạng thái của cửa Mở/Đóng. Trạng thái thay đổi nhờ các tín hiệu điều khiển sau:
 - Không có ai trên vùng trước và vùng sau cửa (NEITHER)
 - Chỉ có người ở vùng trước (FRONT)
 - Chỉ có người ở vùng sau (REAR)
 - Có người cả ở vùng trước và vùng sau (BOTH)
- Cửa không được mở khi có người đứng ở vùng sau vì sẽ đập vào họ.
- Ôtômat có thể biểu diễn bằng sơ đồ sau:



- Hai cách biểu diễn:
 - Biểu diễn phi hình thức:
 - Trực quan,
 - Thích hợp cho việc viết code do con người thực hiện
 - Biểu diễn hình thức:
 - Là dạng máy đọc được
 - Cần một mô hình toán học
 - Thích hợp cho việc tự động sinh code

Biểu diễn phi hình thức của ôtômat hữu hạn: sơ đồ trạng thái

- Đồ thị định hướng với các đỉnh là các trạng thái, các cung có nhãn là các ký hiệu vào
- Trạng thái đầu có mũi tên với nhãn “đầu”
- Các trạng thái kết thúc được ký hiệu bằng vòng tròn kép.
- Ôtômat hữu hạn đơn định (ÔHĐ): Với mọi ký hiệu $a \in \Sigma$ tồn tại nhiều nhất một cung nhãn a xuất phát từ mỗi trạng thái.



Nhận biết các từ tố của KPL

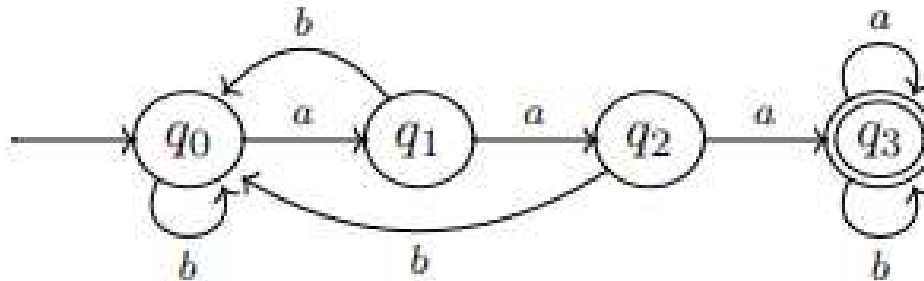
- Ngôn ngữ chứa các từ tố là ngôn ngữ chính quy. Nó có thể được đoán nhận bởi một ôtômat hữu hạn
- Mô hình ôtômat hữu hạn có nhiều loại: đơn định, không đơn định, có dịch chuyển ϵ ...
- Mô hình ôtômat hữu hạn để code bộ phân tích từ vựng là ôtômat hữu hạn đơn định (ÔHĐ)

Định nghĩa hình thức của ôtômat hữu hạn đơn định

- Xuất phát từ 1 trạng thái có tối đa 1 cung ứng với mỗi ký hiệu được xét.
- Khi phải xét hoạt động của một ôtômat hữu hạn trên máy tính, cần phải mô tả nó ở dạng máy đọc được. Mô tả hình thức chính là dạng đó.
- Ôtômat hữu hạn đơn định (ÔHĐ) là bộ 5
 $M = (Q, \Sigma, \delta, q_0, F)$, trong đó :
 - Σ : Bảng chữ hữu hạn - bảng chữ của xâu vào
 - Q : Tập hữu hạn trạng thái
 - $q_0 \in Q$: Trạng thái đầu
 - $F \subseteq Q$: Tập trạng thái kết thúc
 - δ : hàm $\Sigma \times Q \rightarrow Q$ gọi là hàm chuyển trạng thái

Ví dụ về ôtômat hữu hạn đơn định

- Đoán nhận tập các xâu trên $\{a,b\}$ chứa ba ký hiệu a liên tiếp



- Mô tả hình thức

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

$$Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{a,b\}, F = \{q_3\}$$

- Hàm chuyển δ được mô tả trong bảng bên

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

Đầu vào, đầu ra của bộ phân tích từ vựng

- Vào (CT nguồn)

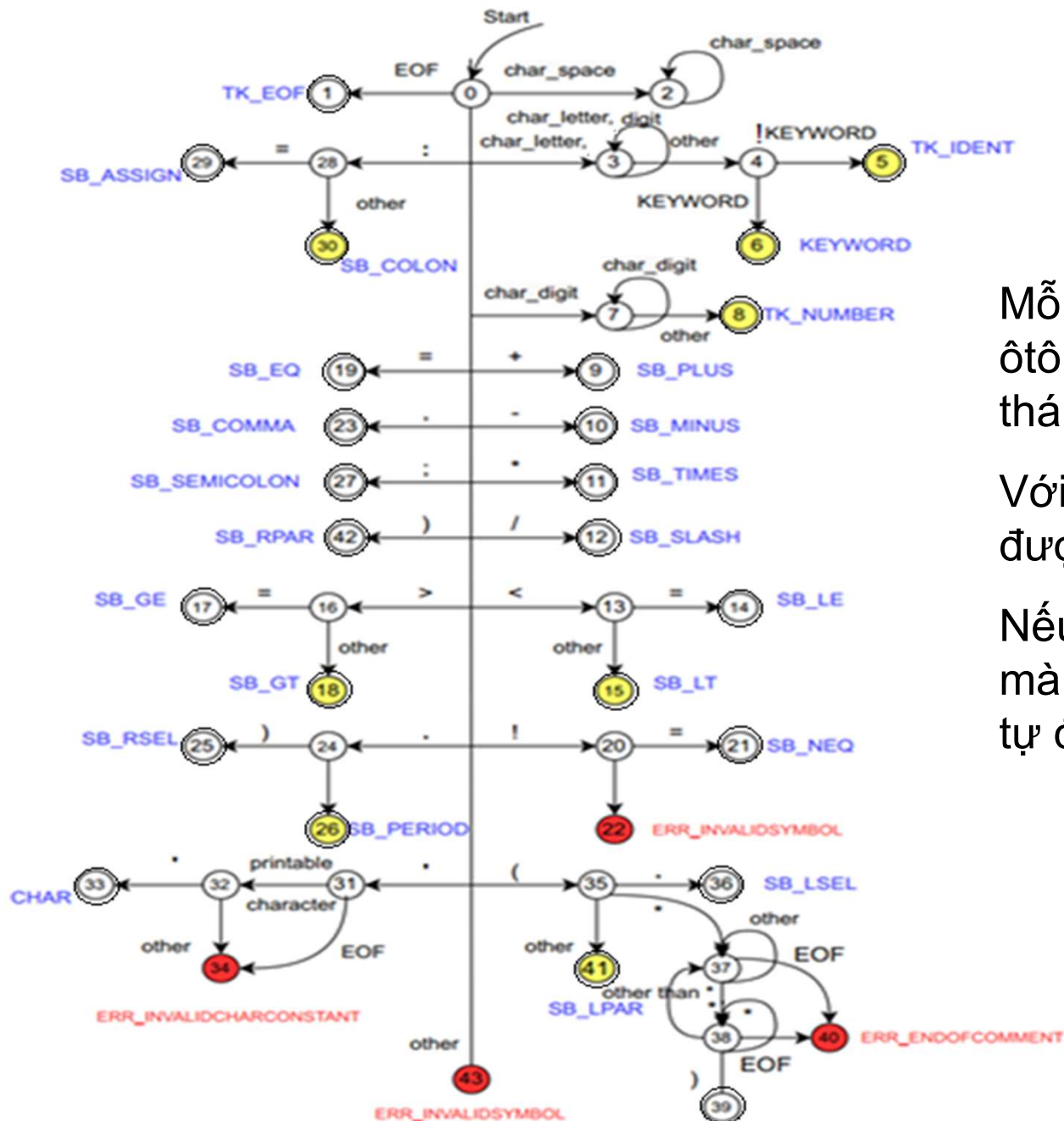
```
Program Example1; (* Example 1*)  
Begin  
End. (* Example1*)
```

- Ra (Dãy từ tố)

```
1-1:KW_PROGRAM  
1-9:TK_IDENT(Example1)  
1-17:SB_SEMICOLON  
2-1:KW_BEGIN  
3-1:KW_END  
3-4:SB_PERIOD
```

- Chỉ cho phép số nguyên không dấu, giá trị không vượt quá hằng INT_MAX (limits.h)
- Định danh bao gồm chữ cái (hoa + thường), chữ số, bắt đầu bằng chữ cái, độ dài không vượt quá 15. Phân biệt chữ hoa, chữ thường
- Từ khóa không phân biệt chữ hoa, chữ thường
- Chỉ cho phép hằng ký tự, là một ký tự in được, bao trong cặp nháy đơn
- Ngôn ngữ không sử dụng hằng xâu ký tự
- Dấu – chỉ được dùng cho phép trừ, không phải phép đổi dấu
- Phép so sánh khác biểu thị bằng !=

Ôtômat hữu hạn của bộ phân tích từ vựng KPL



Mỗi khi đoán nhận được 1 từ tố, ô tômat hữu hạn lại quay về trạng thái 0.

Với những ký tự không đoán nhận được, cần thông báo lỗi.

Nếu ô tô mat đến những trạng thái màu vàng, ký tự hiện hành đã là ký tự đầu của từ tổ tiếp theo

- Phân loại ký tự
- Chọn cấu trúc dữ liệu phù hợp cho token
- Nhận biết các từ tố bằng mô hình ô tômat hữu hạn đơn định

Phân loại các ký tự của CT nguồn

```
typedef enum {  
    CHAR_SPACE,           // Khoảng trống  
    CHAR_LETTER,          // Chữ cái  
    CHAR_DIGIT,           // Chữ số  
    CHAR_PLUS,            // '+'  
    CHAR_MINUS,           // '-'  
    CHAR_TIMES,           // '*'  
    CHAR_SLASH,           // '/'  
    CHAR_LT,              // '<'  
    CHAR_GT,              // '>'  
    CHAR_EXCLAMATION,     // '!'  
    CHAR_EQ,              // '='  
    CHAR_COMMA,           // ','  
    CHAR_PERIOD,          // '.'  
    CHAR_COLON,           // ':'  
    CHAR_SEMICOLON,       // ';'   
    CHAR_SINGLEQUOTE,     // '\''  
    CHAR_LPAR,            // '('  
    CHAR_RPAR,            // ')'   
    CHAR_UNKNOWN          // Ký tự không được phép có mặt trong chương trình  
} CharCode;  
  
CharCode charCodes[256] = {...};
```



```
enum {  
    TK_NONE, TK_IDENT, TK_NUMBER, TK_CHAR, TK_EOF,  
  
    KW_PROGRAM, KW_CONST, KW_TYPE, KW_VAR,  
    KW_INTEGER, KW_CHAR, KW_ARRAY, KW_OF,  
    KW_FUNCTION, KW_PROCEDURE,  
    KW_BEGIN, KW_END, KW_CALL,  
    KW_IF, KW_THEN, KW_ELSE,  
    KW_WHILE, KW_DO, KW_FOR, KW_TO,  
  
    SB_SEMICOLON, SB_COLON, SB_PERIOD, SB_COMMA,  
    SB_ASSIGN, SB_EQ, SB_NEQ, SB_LT, SB_LE, SB_GT, SB_GE,  
    SB_PLUS, SB_MINUS, SB_TIMES, SB_SLASH,  
    SB_LPAR, SB_RPAR, SB_LSEL, SB_RSEL  
};
```

```
state = 0;
currentChar = readChar();
token = getToken();
while (token!=EOF)
{
    state =0;
    token = getToken();
}
```

Nhận biết từ tổ

```
switch (state)
{
case 0 :
    switch (currentChar)
    {
        case space
            state = 2;
        case lpar
            state = 38;
        case letter
            state = 3;
        case digit
            state = 7;
        case plus
            state = 9;
        case lt
            state = 13
        .....
    }
}
```



Case 9:

```
readChar() ;  
return SB_PLUS;
```

case 13:

```
readChar() ;  
if (currentChar = EQ) state = 14 else state =  
15;
```

case 14:

```
readChar() ;  
return SB_LE;
```

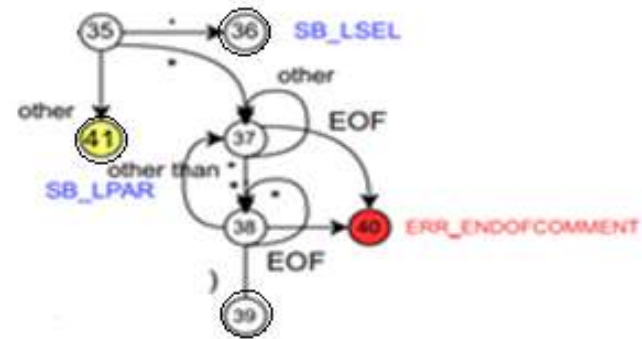
case 15:

```
return SB_LT;
```

Nhận biết từ tổ (tiếp)

Case 2:

```
while (currentChar= space) // skip blanks
    readChar();
return getToken();
case 35:
    readChar();
    if (currentChar= EOF) state =41;
    else
        switch (currentChar)
        {
        case period
            state = 36; // token lsel
        case times
            state =37; //skip comment
        default
            state =41; // token lpar
        }
        return getToken();
    }
```



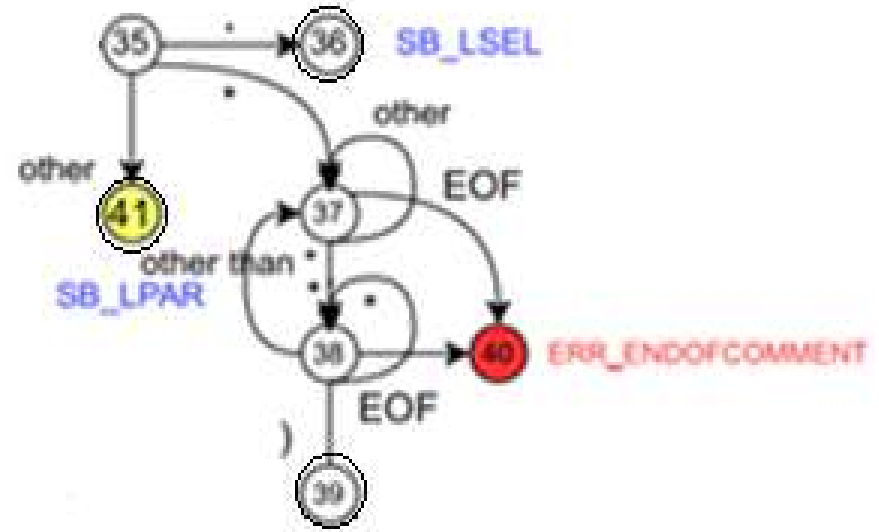
Nhận biết từ tổ (tiếp)

```
case 37: // skip comment
    readChar();
    while (currentChar != times)
    {
        state = 37;
        readChar();
    }
```

```
state = 38;
```

```
case 38:
    readChar();
    while (currentChar == times)
    {
        state = 38;
        currentChar = readChar();
    }
```

```
If (currentChar == lpar) state = 39; else state = 40;
```



- Lập danh mục từ khóa, có thể dùng mảng
- Nếu số lượng từ khóa nhiều có thể phân phối bộ nhớ động
- Lập một hàm trả ra một từ khóa hoặc định danh
- Biến ch nhận ký tự đầu tiên của từ vị.
- Đọc tất cả các chữ cái và chữ số tiếp theo đến khi nhận được một ký tự không phải chữ cái hoặc chữ số.
- Sử dụng thuật toán tìm kiếm nhị phân để tìm xem có từ khóa nào trùng với t.
- Nếu tìm được $t.kind =$ thứ tự của từ khóa trong bảng từ khóa
- Ngược lại, $t.kind = ident$
- Kết thúc quá trình đoán nhận, biến ch chứa ký tự đầu của từ tố tiếp theo.

case 4:

```
    if (checkKeyword (token) == TK_NONE) state = 5;  
    else state = 6;
```

case 5:

```
    install_ident(); // save to symbol table
```

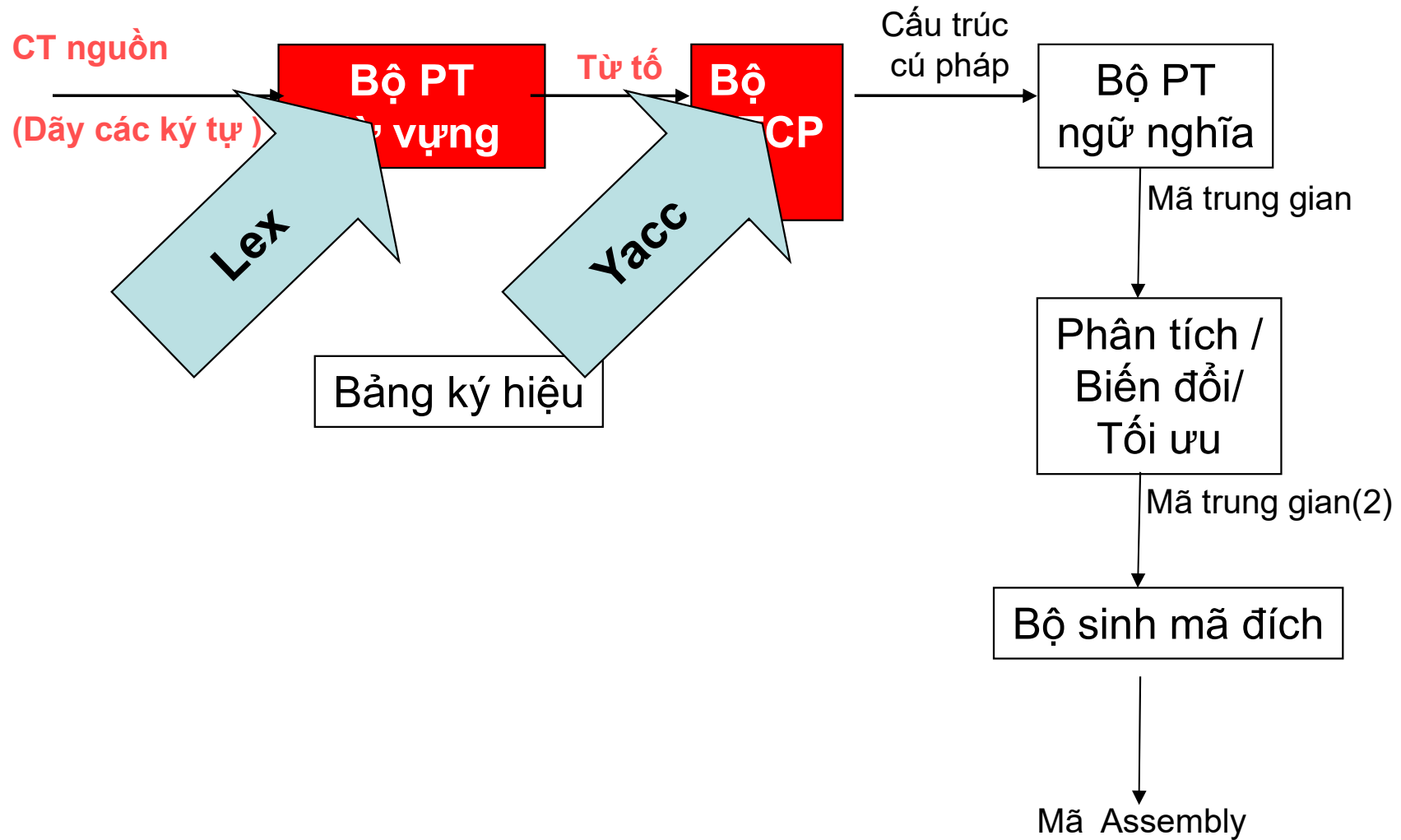
case 6

```
    return checkKeyword(token);
```

.....

- Công cụ để đặc tả bộ phân tích từ vựng cho nhiều ngôn ngữ
- Một số bộ sinh phân tích từ vựng
 - Lex và Yacc của AT & T
 - **Lexer trong ANTLR** (ANother Tool for Language Recognition) ĐH San Francisco
 - **Flex của Berkeley Lab**

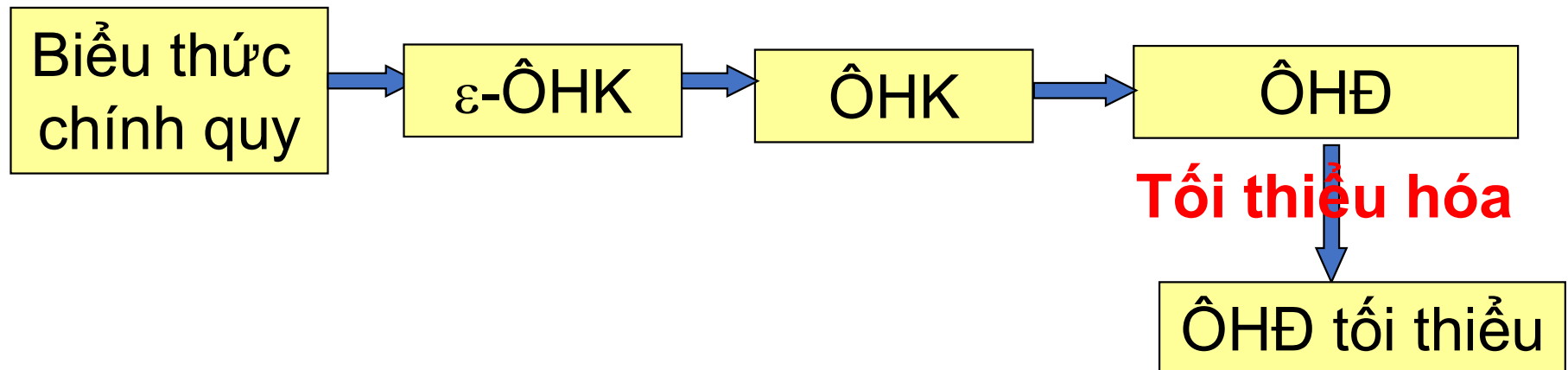
Nguyên lý



- Vào
Đặc tả từ vựng của ngôn ngữ lập trình
(Biểu thức chính quy)
- Ra
Chương trình phân tích từ vựng
(chương trình C)

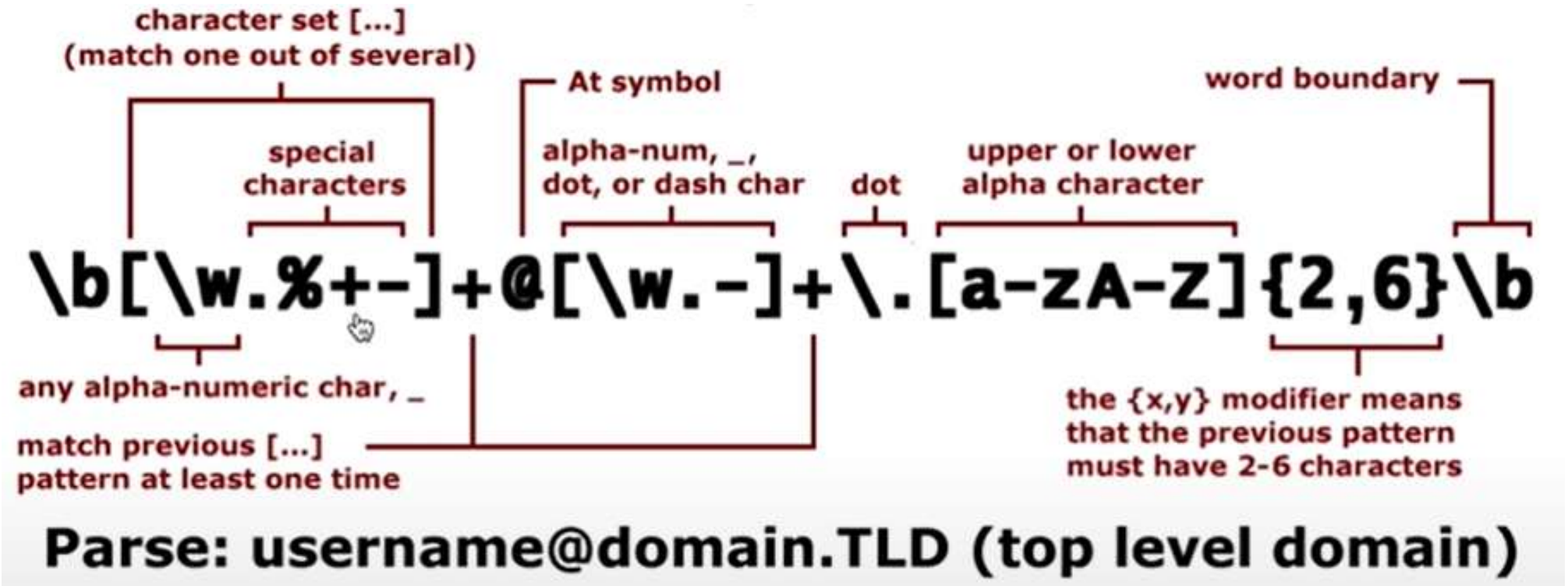
Quá trình biến đổi để xây dựng bộ sinh phân tích từ vựng

- Luật từ vựng được thể hiện bằng các Biểu thức chính quy
- Từ biểu thức chính quy biến đổi tương đương qua các dạng khác nhau của ôtômat hữu hạn.
- Mô tả hình thức của ôtômat hữu hạn đơn định được dùng để sinh ra mã nguồn của bộ phân tích từ vựng

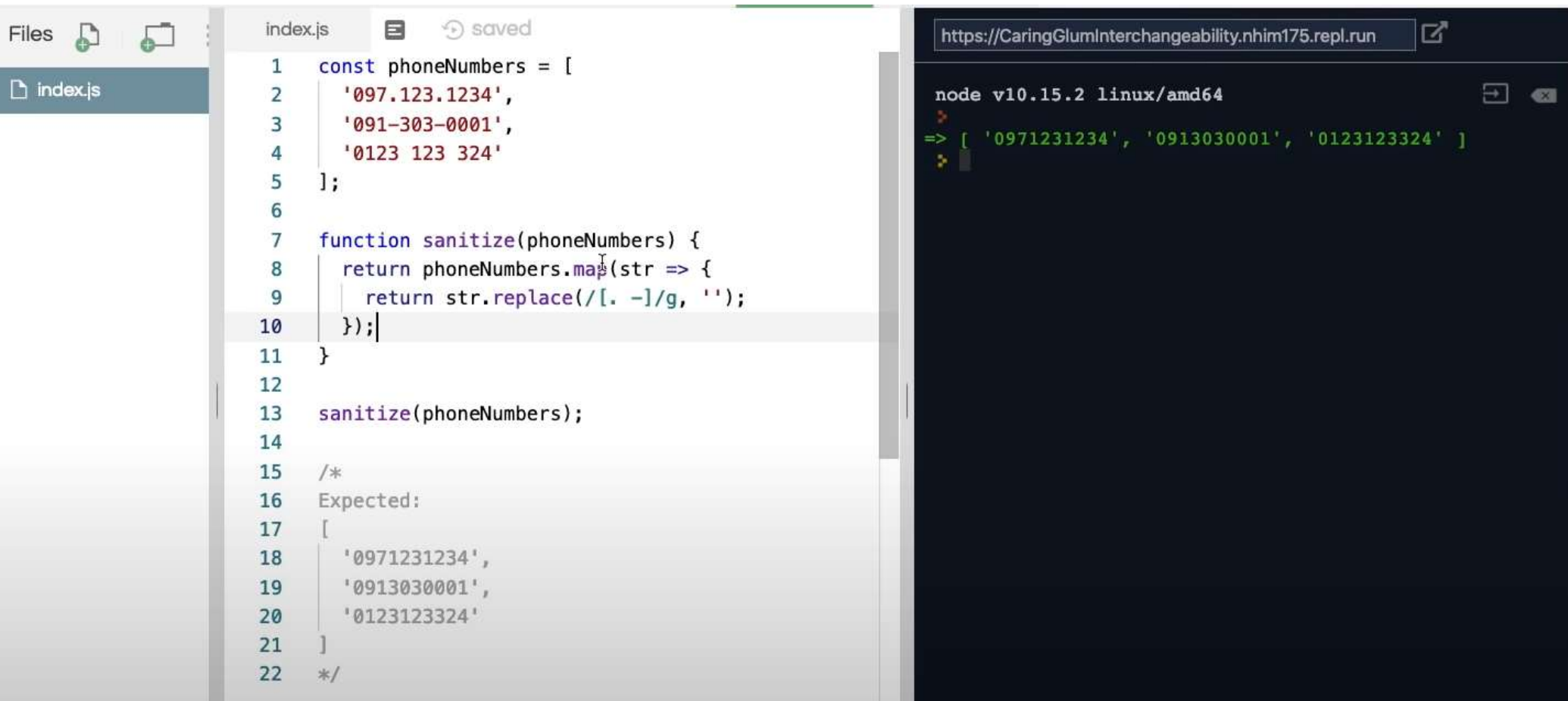


Biểu thức chính quy

- Biểu thức chính quy mô tả tên miền



Biểu thức chính quy trong Javascript



The image shows a code editor on the left and a terminal on the right. The code editor displays a JavaScript file named `index.js` with the following content:

```
1  const phoneNumbers = [  
2    '097.123.1234',  
3    '091-303-0001',  
4    '0123 123 324'  
5  ];  
6  
7  function sanitize(phoneNumbers) {  
8    return phoneNumbers.map(str => {  
9      return str.replace(/[. -]/g, '');  
10   });  
11  }  
12  
13  sanitize(phoneNumbers);  
14  
15  /*  
16  Expected:  
17  [  
18    '0971231234',  
19    '0913030001',  
20    '0123123324'  
21  ]  
22  */
```

The terminal on the right shows the output of the code execution. The URL `https://CaringGlumInterchangeability.nhim175.repl.run` is visible at the top. The terminal output is:

```
node v10.15.2 linux/amd64  
=> [ '0971231234', '0913030001', '0123123324' ]
```


Biểu thức chính quy (regular expression)

Định nghĩa hình thức của biểu thức chính quy

Cho Σ là một bảng chữ.

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

Biểu thức chính quy là công cụ tương đương với ô tô mat hữu hạn

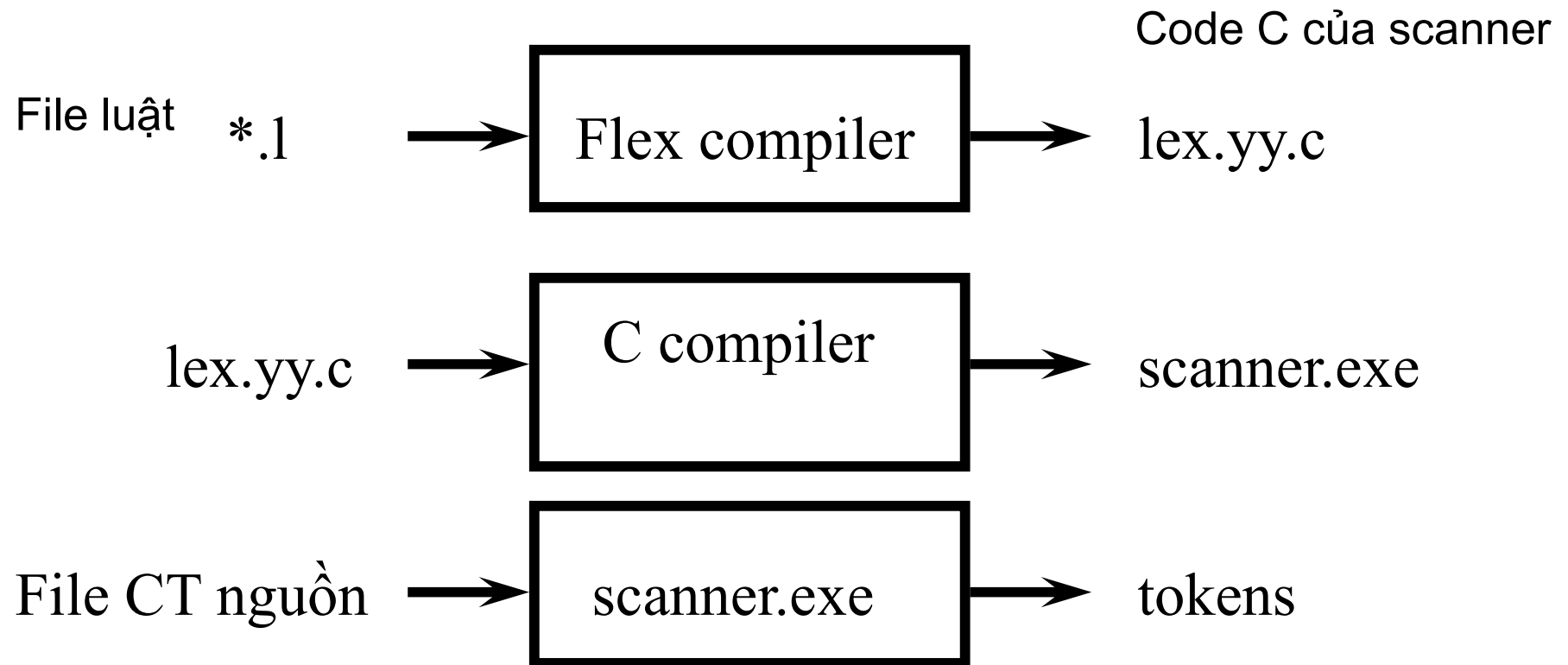
- Tập các xâu trên $\{0, 1\}$ bắt đầu bằng 1
 $(1((0+1)^*))$
- Tập các xâu trên $\{a,b\}$ có chứa 3 ký hiệu a liên tiếp
 $(((((a + b)^*)a)a)a)((a+b)^*))$
- Tập các định danh của KPL? Cần sự biến đổi biểu thức chính quy thành định nghĩa chính quy với những định nghĩa cho tập chữ cái, chữ số...

FLEX: Bộ sinh phân tích từ vựng

- Công cụ để **sinh** ra bộ phân tích từ vựng.
- Đọc file đầu vào **là mô tả từ vựng của một ngôn ngữ**.
- Mô tả là một tập các cặp hai thành phần **Biểu thức chính quy (định nghĩa chính quy)** và **code C**, gọi là *luật*.
- Flex sinh ra code trên C, lưu trữ trong file `lex.yy.c` có chứa thủ tục `yylex()`.
- File này được dịch và kết nối với các thư viện của FLEX để tạo ra file thực thi.
- Khi chạy file thực thi, nó sẽ phân tích các biểu thức chính quy, rồi thực hiện đoạn code C tương ứng với luật đó.



Quy trình làm việc của bộ sinh PTTV



Các bước sinh bộ phân tích từ vựng từ FLEX

1. Tạo file luật (*.l)
2. Chạy FLEX với file luật tạo ra file `lex.yy.c` với hàm phân tích `yylex()`.
3. Chạy C compiler để tạo ra bộ phân tích từ vựng.



Đặc tả LEX

Đầu vào cho FLEX là file đặc tả bao gồm 3 phần, phân cách bởi một dòng %%

%{

khai báo bổ trợ

%}

định nghĩa chính quy

%%

luật dịch

%%

thủ tục bổ trợ

p_1 {hành động 1}

p_2 {hành động 2}

.....

p_3 {hành động 3}

Ví dụ: Dùng Flex sinh scanner cho KPL

- Dựa trên các đặc điểm từ vựng của KPL
 - Số nguyên: không dấu
 - Định danh : dùng chữ cái, chữ số, bắt đầu bằng chữ cái
 - Hằng ký tự
 - Bỏ qua ký tự trắng, chú thích
- Xây dựng mô tả từ vựng scanner.l
- Dùng Flex sinh ra tệp lex.yy.c
- Thực hiện với ví dụ

Phần1: định nghĩa và hàm thư viện :

Tạo ra hàm yyloc() tính số dòng, số cột của từ tố

Tham số truyền vào là text , x, y.Trong đó :

text: từ tố cần phân tích

x số của dòng chứa từ tố

y: số của cột chứa từ tố.

Phần 1: định nghĩa và hàm thư viện.

Định nghĩa chính quy:

- NUMBER : các số từ 0 → 9 (có thể mở rộng ra gồm nhiều chữ số) .Biểu thức chính quy : $[0-9]^+$
- DELIMITER : Các dấu trắng , khoảng cách , xuống dòng. Biểu thức chính quy : $[\backslash n \backslash t \backslash r]$
- CHAR : Nhận diện các kí tự bắt đầu bởi dấu ' , kết thúc bởi dấu ' và in ra phần bên trong 2 dấu nháy ([:print:] tương đương với hành động in ra) . Biểu thức chính quy : $\backslash'[:print:]'\backslash'$
- IDENT: các biến được định nghĩa . Cho phép các kí tự hoặc số. Biểu thức chính quy : $[a-zA-Z][a-zA-Z0-9]^*$
- COMMENT: khớp với các kí tự được bắt đầu bằng "(" và kết thúc bởi ")". Biểu thức chính quy : $\backslash(\backslash^*([\backslash^*][\backslash^*+\backslash^*]))^*\backslash^*+\backslash)$
- ERROR : khi các kí tự không nằm trong khoảng cho phép (automat kpl) Biểu thức chính quy : $[\wedge+\backslash-^*/,;.:)=a-zA-Z0-9<>]$

- **Phần quy tắc dịch :**

Do cấu trúc của flex dịch từ trên xuống , do đó ta sẽ ưu tiên thứ tự cho các biểu thức cho phù hợp. Ví dụ như EOF hay COMMENT sẽ duyệt đầu tiên và in ra .

Hàm phải được thể hiện trên 1 dòng , in ra chữ tương ứng với cú pháp được đưa vào từ input , cộng thêm địa chỉ của từ được phân tích bằng cách sử dụng hàm lấy vị trí đã nêu ở trên.

- Chương trình sẽ in ra bao gồm vị trí của từ “x-y” , từ tố.

- **Phần thủ tục hỗ trợ**
- Cho phép file đọc input và in ra file output .
Khi run file sẽ gồm 3 phần: file exe , tên file input , tên output (tùy chọn) .
Nếu sai cú pháp thoát chương trình

Khai báo hỗ trợ và định nghĩa chính quy

```
1 %option nowwrap
2 %{
3
4     void yyloc(char* text, int *x, int *y){
5         while (*text != '\0'){
6             if (*text == '\n'){
7                 *y += 1;
8                 *x = 1;
9             } else {
10                 *x += 1;
11             }
12             text++;
13         }
14     }
15
16     int x = 1;
17     int y = 1;
18
19 %}
20
21 NUMBER          [0-9]+
22 DELIMITER       [ \n\t\r]
23 CHAR            \'[[[:print:]]\'
24 IDENT           [a-zA-Z][a-zA-Z0-9]*
25 COMMENT         \(\([^*]|(\*+[^*]))*\*+\\\)
26 ERROR           [^+\\-*/,;.:()=a-zA-Z0-9<>]
27
28 %%
```

Các luật dịch

```
DELIMITER      [ \n\t\r]
CHAR            \'[:print:]]\'
IDENT          [a-zA-Z][a-zA-Z0-9]*
COMMENT        \(\*(\[^\*]|(\*[^\*]))*\*+\)
ERROR          [^+\-*/,;.:()=a-zA-Z0-9<>]

%%

<<EOF>>        {return 0;}
{COMMENT}      {yyloc(yytext, &x, &y);}
{DELIMITER}    {yyloc(yytext, &x, &y);}

"+"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_PLUS"); yyloc(yytext, &x, &y);}
"-"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_MINUS"); yyloc(yytext, &x, &y);}
"*"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_TIMES"); yyloc(yytext, &x, &y);}
"/"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_SLASH"); yyloc(yytext, &x, &y);}
";"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_SEMICOLON"); yyloc(yytext, &x, &y);}
","           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_COMMA"); yyloc(yytext, &x, &y);}
"."           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_PERIOD"); yyloc(yytext, &x, &y);}
":="          {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_ASSIGN"); yyloc(yytext, &x, &y);}
"("           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_LSEL"); yyloc(yytext, &x, &y);}
")"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_RSEL"); yyloc(yytext, &x, &y);}
"!="          {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_NEQ"); yyloc(yytext, &x, &y);}
">="          {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_GE"); yyloc(yytext, &x, &y);}
"<="          {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_LE"); yyloc(yytext, &x, &y);}
"="           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_EQ"); yyloc(yytext, &x, &y);}
"<"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_LT"); yyloc(yytext, &x, &y);}
">"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_GT"); yyloc(yytext, &x, &y);}
":"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_COLON"); yyloc(yytext, &x, &y);}
"("           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_LPAR"); yyloc(yytext, &x, &y);}
")"           {fprintf(yyout, "%d-%d: %s\n", y, x, "SB_RPAR"); yyloc(yytext, &x, &y);}

"PROGRAM"     {fprintf(yyout, "%d-%d: %s\n", y, x, "KW_PROGRAM"); yyloc(yytext, &x, &y);}
"CONST"       {fprintf(yyout, "%d-%d: %s\n", y, x, "KW_CONST"); yyloc(yytext, &x, &y);}
"TYPE"        {fprintf(yyout, "%d-%d: %s\n", y, x, "KW_TYPE"); yyloc(yytext, &x, &y);}
"VAR"         {fprintf(yyout, "%d-%d: %s\n", y, x, "KW_VAR"); yyloc(yytext, &x, &y);}
"INTEGER"     {fprintf(yyout, "%d-%d: %s\n", y, x, "KW_INTEGER"); yyloc(yytext, &x, &y);}
"CHAR"        {fprintf(yyout, "%d-%d: %s\n", y, x, "KW_CHAR"); yyloc(yytext, &x, &y);}
```

Thủ tục hỗ trợ

```
83 int main(argc, argv)
84 int argc;
85 char **argv;
86 {
87     extern FILE* yyin;
88     extern FILE* yyout;
89
90     if (argc == 3) {
91         yyin = fopen(argv[1], "r");
92         yyout = fopen(argv[2], "w");
93     } else {
94         printf("Usage: scanner.exe [input_file] [output_file]");
95         exit(1);
96     }
97     yylex();
98     return 0;
99 }
```