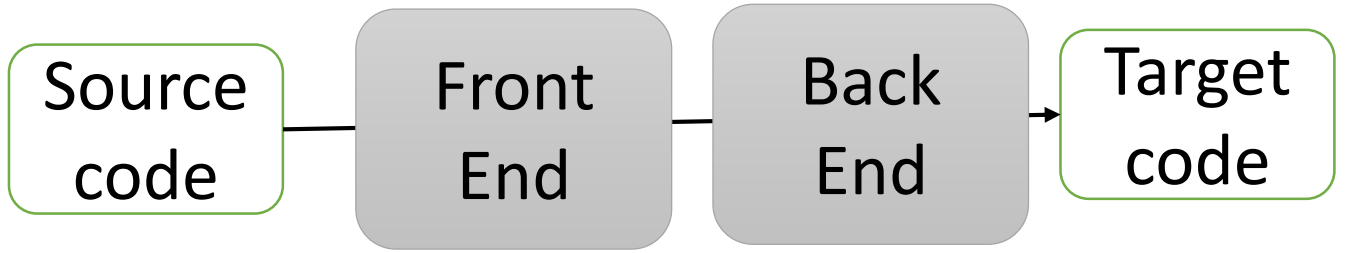
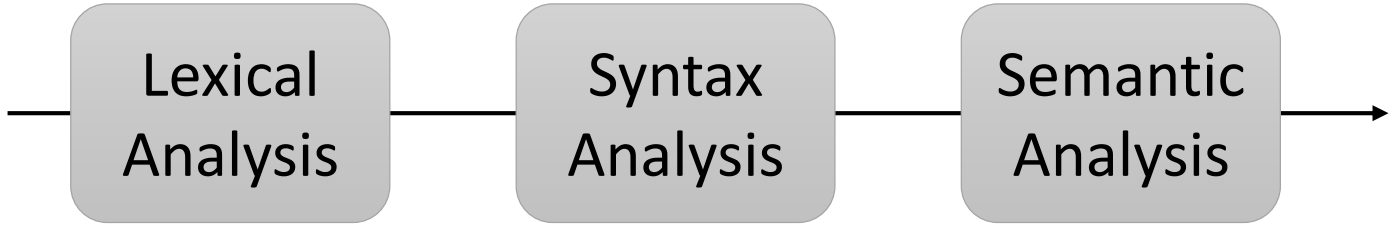
ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

Chương 2
Phân tích từ vựng (Lexical Analysis)

Phân tích từ vựng

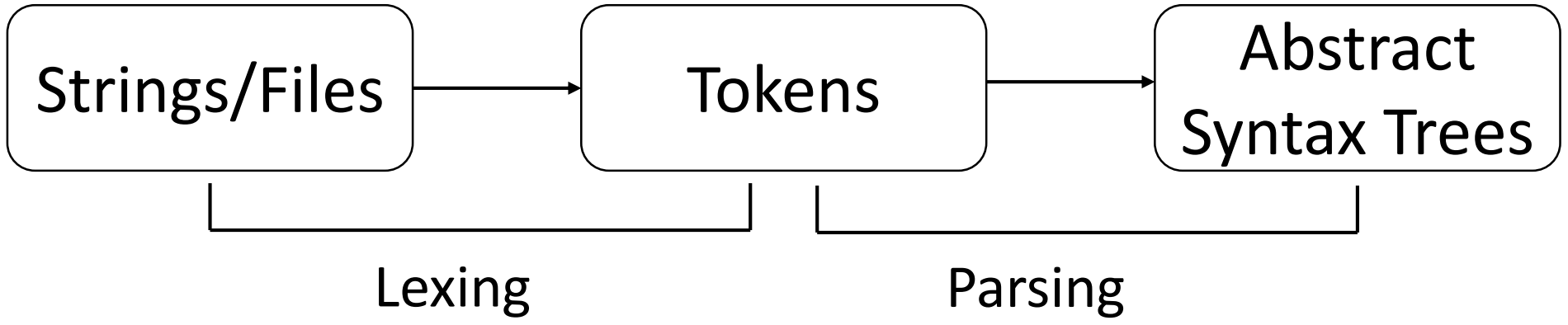
- Front End (Analysis)
  - Phân tích từ vựng
  - Phân tích cú pháp
  - Phân tích ngữ nghĩa
  - Sinh mã trung gian
  - Tối ưu mã trung gian
  - Tạo và quản lý bảng danh biểu
  - Thông báo lỗi
- Back End (Synthesis)
  - Sinh mã đối tượng
  - Tối ưu mã
  - Quản lý bảng danh biểu
  - Thông báo lỗi





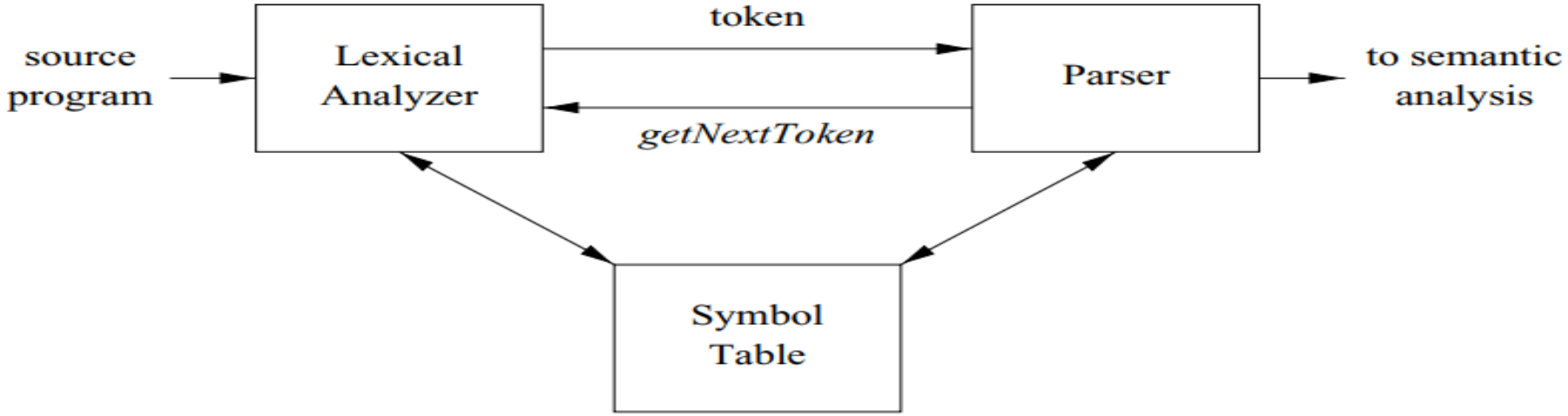
Vai trò của phân tích từ vựng

- Phân tích từ vựng là giai đoạn đầu của quá trình biên dịch
- Nhiệm vụ cơ bản của phân tích từ vựng
  - Đọc các ký tự nhập vào của chương trình nguồn (các ký tự thuộc bảng chữ cái của ngôn ngữ nguồn) và nhóm chúng lại thành các lexeme
  - Sinh ra chuỗi các token đầu ra đối với mỗi lexeme của chương trình nguồn



Vai trò của phân tích từ vựng

❑ Sự giao tiếp giữa bộ phân tích từ vựng và bộ phân tích cú pháp



- Tương tác được thực hiện bằng cách bộ phân tích cú pháp gọi bộ phân tích từ vựng
- Lệnh gọi *getNextToken* yêu cầu bộ phân tích từ vựng đọc các ký tự nhập đầu vào cho đến khi có thể xác định lexeme tiếp theo và tạo ra token tiếp theo
- Token tiếp theo này được bộ phân tích từ vựng trả về (gửi cho) cho bộ phân tích cú pháp.

Vai trò của phân tích từ vựng

- Khi đọc từng ký tự nhập đầu vào, bộ phân tích từ vựng phải biết:
  - Nhóm các ký tự thích hợp lại thành các lexeme
  - Loại bỏ những ký tự không cần thiết cho các giai đoạn sau như chú thích, khoảng trắng (whitespace): ký trự trống (blank), nhiều khoảng trống (tab), ký tự xuống hàng mới (newline)
- Ví dụ: Chuỗi ký tự đầu vào trong ngôn ngữ lập trình C

  if (a>= b) a = x +y;

  else a= y-x;

  - Bộ phân tích từ vựng sẽ nhóm các ký tự thành các lexeme:

    if (a >= b) a = x + y ; else a = y – x ;
  - Bộ phân tích từ vựng sẽ gán cho mỗi lexeme bằng token tương ứng
- Các token cùng với các lexeme sẽ được lưu vào bảng danh biểu

Vai trò của phân tích từ vựng

- Có nhiều lý do để tách riêng giai đoạn phân tích từ vựng với giai đoạn phân tích cú pháp:
  - 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,…)
  - 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.
  - Tính đa tương thích của trình biên dịch cũng được cải thiện (chẳng hạn, 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,….)

Token, Pattern, Lexeme

- Khi nói đến bộ phân tích từ vựng, ta sử dụng các thuật ngữ: token, pattern, lexeme
  - Token (từ tố, thực thể cú pháp) là các ký hiệu kết thúc trong văn phạm đối với một chương trình nguồn. Token là một cặp gồm *token name* và *attribute value*, trong đó *token name* gồm: từ khóa (keywords), định danh (identifiers), toán tử (operators), dấu câu (separators), hằng (constants), số (numbers), chuỗi (literals),…
  - Lexeme (trị từ vựng) của một token là một chuỗi ký tự biểu diễn cho token đó.
  - Pattern (mẫu từ vựng) là quy tắc mô tả một tập hợp các trị từ vựng của token nào đó (tập hợp các quy tắc định nghĩa một token).

Token, Pattern, Lexeme

- Ví dụ:

Token	Informal Description	Sample Lexemes
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

Token, Pattern, Lexeme

- Ví dụ: Xét biểu thức sau trong ngôn ngữ lập trình C

  sum=3+2;

  - Kết quả:

lexeme	token name
sum	identifier
=	operator
3	number
+	operator
2	number
;	seperator

Token, Pattern, Lexeme

- Ví dụ: if (y <= t) x = z - 3;

lexeme	token name
if	keyword
(	open parenthesis
y	identifier
<=	comparison operator
t	identifier
)	close parenthesis
x	identifier
=	assignment operator
z	identifier
-	arithmatic operator
3	number
;	semicolon

Token, Pattern, Lexeme

- Ví dụ:

```
while (i <= n) i= i*2;
```

Token, Pattern, Lexeme

- Ví dụ:

while (i <= n) i= i*2;

lexeme	*token name*
while	keyword
(	open parenthesis
i	identifier
<=	comparison operator
n	identifier
)	close parenthesis
i	identifier
=	assignment operator
i	identifier
*	arithmatic operator
2	number
;	semicolon

Token, Pattern, Lexeme

- Ví dụ:

```
count = count + 2*i  ;
sum = sum + count;
```

Token, Pattern, Lexeme

❑ Ví dụ:

count = count + 2*i ;

sum = sum + count;

lexeme	*token name*
count	identifier
=	assignment operator
count	identifier
+	arithmatic operator
2	number
*	arithmatic operator
i	identifier
;	semicolon
sum	identifier
=	assignment operator
sum	identifier
+	arithmatic operator
count	identifier
t	identifier
;	semicolon

Tính chất của token

- Ta thấy cùng một pattern có thể có nhiều lexeme so khớp trùng nhau
  - ab, t, sum,… đều là các token định danh (identifier)
  - 123, 52,… đều là các token số (number)
- Khi cùng một pattern có thể có nhiều lexeme so khớp trùng nhau, bộ phân tích từ vựng phải cung cấp thêm một số thông tin khác liên quan đến token và lưu vào bảng danh biểu để phục vụ các giai đoạn tiếp theo của biên dịch.
- 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.
- Token kết hợp với thuộc tính của nó tạo thành một cặp có dạng

<token name, attribute value>

Tính chất của token

- Ví dụ, với chuỗi ký tự là phát biểu A = B* C - 8, qua giai đoạn phân tích từ vựng ta có bảng danh biểu như sau

attribute value	*lexeme*	*token name*
1	A	identifier
2	=	assignment operator
3	B	identifier
4	*	arithmatic operator
5	C	identifier
6	-	arithmatic operator
7	8	number

Tính chất của token

- Ví dụ, với chuỗi ký tự là phát biểu A = B* C - 8, qua giai đoạn phân tích từ vựng ta có các token và thuộc tính của token như sau
  - <identifier, 1> : *token name* của A là identifier, *attribute value 1* là vị trí của A trong bảng danh biểu (con trỏ trỏ đến vị trí của A trong bảng danh biểu)
  - < assignment operator, >: *token name* của = là assignment operator, *attribute value* không cần
  - < identifier, 3> : *token name* của B là identifier, *attribute value* 3 là vị trí của B trong bảng danh biểu (con trỏ trỏ đến vị trí của B trong bảnh danh biểu)
  - < arithmatic operator, >: *token name* của * là arithmatic operator, *attribute value k*hông cần
  - < identifier, 5> : *token name* của C là identifier, *attribute value* 5 là vị trí của C trong bảng danh biểu (con trỏ trỏ đến vị trí của C trong bảng danh biểu)
  - < arithmatic operator, >: *token name* của - là arithmatic operator, *attribute value* không cần
  - <number, 8> : *token name* của 8 là number, *attribute value* 8 là giá trị của hằng số nguyên

Tính chất của token

- Qua biểu diễn của token theo cặp gồm 2 thành phần như vậy, ta xác định được lexeme của mỗi token
- Với ví dụ trên, ta có chuỗi token là kết quả:
  - < identifier,‘A’> : *token name* của A là identifier, *lexeme là* ‘A’
  - < assignment operator, ‘=‘ >: *token name* của = là assignment operator, *lexeme là* ‘=’
  - < identifier,‘B’> : *token name* của B là identifier, *lexeme là* ‘B’
  - < arithmatic operator,‘*’>: *token name* của * là arithmatic operator, *lexeme là* ‘*’
  - < identifier, ‘C’> : *token name* của C là identifier, *lexeme là* ‘C’
  - < arithmatic operator,’-’ >: *token name* của - là arithmatic operator, *lexeme là* ‘-’
  - <number, 8> : *token name* của 8 là number, *lexeme là* ‘8’

Tính chất của token

- Với ví dụ trên, ta có kết quả bảng chuỗi các token đầu ra:

lexeme	The following tokens are returned by scanner to parser in specified order
A	<identifier,'A'>
=	<assignment operator,'='> hay <operator,'='>
B	<identifier,'B'>
*	<arithmatic operator,'*'> hay <operator,'*'>
C	<identifier,'C'>
-	<arithmatic operator,'-'> hay <operator,'-'>
8	<number,'8'>

Tính chất của token

- Ví dụ: cout << 3+2+3;
  - Bảng danh biểu

Tính chất của token

- Ví dụ: cout << 3+2+3;
  - Bảng danh biểu

attribute value	*lexeme*	*token name*
1	cout	identifier
2	<<	operator
3	3	literal
4	+	operator
5	2	literal
6	+	operator
7	3	literal
8	;	punctuator

Tính chất của token

- Ví dụ: cout << 3+2+3;
  - Bảng chuỗi các token đầu ra

Tính chất của token

- Ví dụ: cout << 3+2+3;
  - Bảng chuỗi các token đầu ra

Lexeme	The following tokens are returned by scanner to parser in specified order
cout	<identifier, ‘cout’>
<<	<operator, ‘<<‘>
3	<literal, ‘3’>
+	<operator, ‘+’>
2	<literal, ‘2’>
+	<operator, ‘+’>
3	<literal, ‘3’>
;	<punctuator, ‘;’>

Tính chất của token

- Ví dụ: a=b*3;

  if (a>10) a=c;

  - Bảng danh biểu

Tính chất của token

- Ví dụ: a=b*3;

  if (a>10) a=c;

  - Bảng danh biểu

attribute value	*lexeme*	*token name*
1	a	identifier
2	=	operator
3	b	identifier
4	*	operator
5	3	number
6	;	punctuator
7	if	keyword
8	(	open parenthesis
9	a	identifier
10	>	operator
11	10	number
12	)	close parenthesis
13	a	identifier
14	=	operator
15	c	identifier

Tính chất của token

- Ví dụ: a=b*3;

  if (a>10) a=c;

  - Bảng chuỗi các token đầu ra

Tính chất của token

- Ví dụ: a=b*3;

  if (a>10) a=c;

  - Bảng chuỗi các token đầu ra

Lexeme	The following tokens are returned by scanner to parser in specified order
a	<identifier,'a'>
=	<operator,'='>
b	<identifier,'b'>
*	<operator,'*'>
3	<number,'3'>
;	<punctuator,';'>
if	<keyword,'if'>
(	<open parenthesis,'('>
a	<identifier,'a'>
>	<operator,'>'>
10	<number,'10'>
)	<close parenthesis,')'>
a	<identifier,'a'>
=	<operator,'='>
c	<identifier,'c'>

Tính chất của token

- Ví dụ: if (num1 = = num2)
  result = 1;
  else
  result = 0;
  - Bảng danh biểu

Tính chất của token

- Ví dụ: if (num1 = = num2)
  result = 1;
  else
  result = 0;
  - Bảng danh biểu

attribute value	*lexeme*	*token name*
1	if	keyword
2	(	open parenthesis
3	num1	identifier
4	==	comparison operator
5	num2	identifier
6	)	close parenthesis
7	result	identifier
8	=	assignment operator
9	1	number
10	;	punctuator
11	else	keyword
12	result	identifier
13	=	assignment operator
14	0	number
15	;	punctuator

Tính chất của token

- Ví dụ: `if (num1 = = num2)`

```
        result = 1;
    else
        result = 0;
```

  - Bảng chuỗi các token đầu ra

Tính chất của token

- Ví dụ: if (num1 = = num2)
  result = 1;
  else
  result = 0;
  - Bảng chuỗi các token đầu ra

Lexeme	The following tokens are returned by scanner to parser in specified order
if	<keyword,'if'>
(	<open parenthesis,'('>
num1	<identifier,'num1'>
==	<comparison operator,'=='>
num2	<identifier,'num2'>
)	<close parenthesis,')'>
result	<identifier,'result'>
=	<assignment operator,'='>
1	<number,'1'>
;	<punctuator,';'>
else	<keyword,'else'>
result	<identifier,'result'>
=	<assignment operator,'='>
0	<number,'0'>
;	<punctuator,';'>

Tính chất của token

- Xử lý chuỗi đầu vào:

```
if (num1 == num2)
    result = 1;
else
    result = 0;
```

Tính chất của token

- Xử lý chuỗi đầu vào:

```
if (num1 == num2)
    result = 1;
else
    result = 0;
```

```
\tif(num1==num2)\n\t\tresult=1;\n\telse\n\t\tresult=0;
```

Lỗi từ vựng (Lexical errors)

- Chỉ một số ít lỗi được phát hiện tại giai đoạn 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 trong phát biểu **fi (a == f (x))** ...
  - Bộ phân tích từ vựng không thể biết đây là lỗi không viết đúng từ khóa **if** hay một định danh chưa được khai báo.
  - Vì **fi** là một định danh 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 cần phải khắc phục lỗi để phân tích tiếp.

Lỗi từ vựng (Lexical errors)

- 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 khác khắc phục lỗi:
  - Xóa đi một ký tự thừa.
  - Chèn thêm một ký tự bị mất.
  - Thay thế một ký tự không đúng bằng một ký tự đúng.
  - Chuyển đổi hai ký tự kế tiếp nhau.

Lưu trữ tạm thời chương trình nguồn

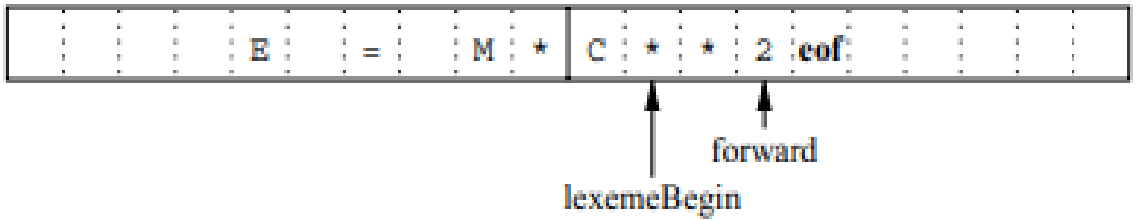
- Việc đọc từng ký tự trong chương trình nguồn có thể tốn thời gian do đó ảnh hưởng đến tốc độ dịch.
- Giải quyết vấn đề này bằng cách đọc mỗi lần một chuỗi ký tự, lưu trữ vào trong vùng nhớ tạm - gọi là bộ đệm input (input buffer).
- Tuy nhiên, việc đọc như vậy cũng gặp một số trở ngại do không thể xác định một chuỗi như thế nào thì chứa trọn vẹn một token?
- Có một số phương pháp đọc bộ đệm hiệu quả:
  - Phương pháp Cặp bộ đệm (Buffer Pairs)
  - Phương pháp Khóa cầm canh (Sentinel)

Lưu trữ tạm thời chương trình nguồn

- Phương pháp Cặp bộ đệm (Buffer Pairs)
  - Đối với nhiều ngôn ngữ nguồn, bộ phân tích từ vựng cần phải "nhìn trước" một số ký tự nhập để đoán biết ranh giới trị từ vựng của một mẫu từ vựng trước khi nó được so trùng.
  - Thực hiện bằng cách 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 này, 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).

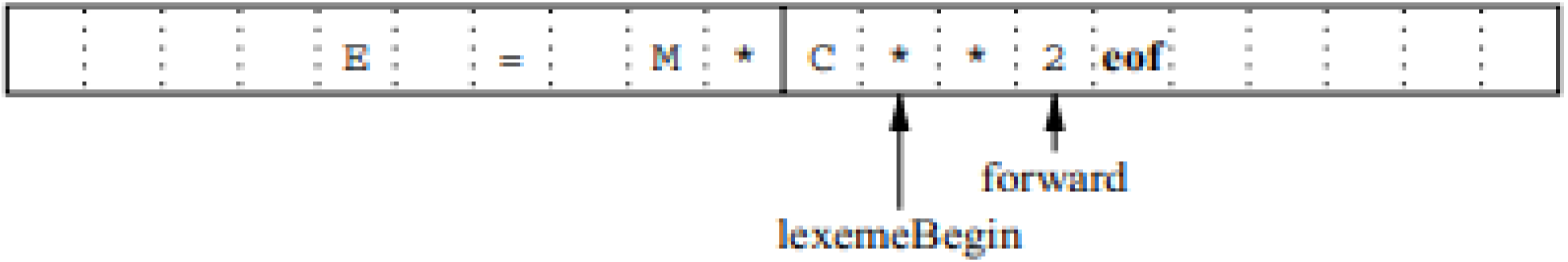
Lưu trữ tạm thời chương trình nguồn

- 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): 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



Lưu trữ tạm thời chương trình nguồn

- Sử dụng hai con trỏ dò tìm trong buffer:
  - 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.
  - Khi con trỏ p2 trỏ 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 trỏ 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.



Lưu trữ tạm thời chương trình nguồn

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

if (p2 trỏ tới ranh giới giữa 2 vùng đệm)

{lấp đầy nửa bên phải bởi N ký tự tiếp theo; p2 trỏ đến đầu nửa cuối};

else if (p2 trỏ tới vị trí cuối bộ đệm)

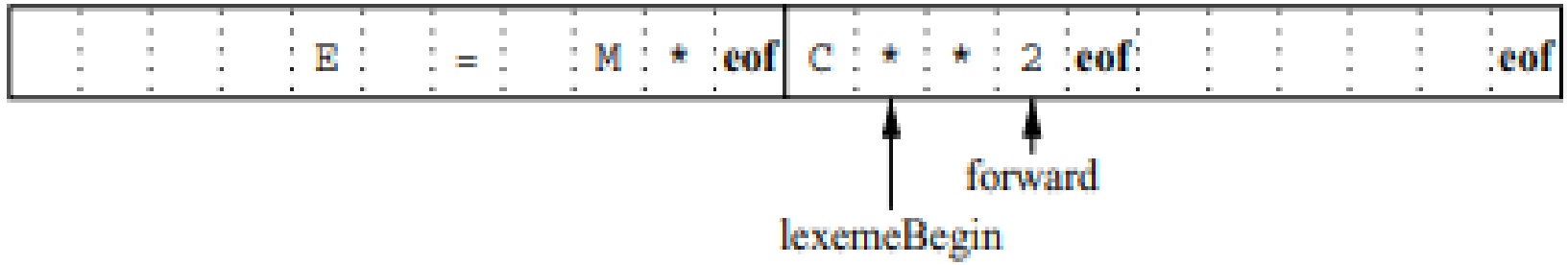
{lấp đầy nửa bên trái bởi N ký tự tiếp theo; p2 trỏ đến đầu bộ đệm};

else p2 dời sang phải;

- Phương pháp này thường hoạt động rất tốt, tuy nhiên:
  - Số lượng ký tự đọc trước bị giới hạn
  - 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.

Lưu trữ tạm thời chương trình nguồn

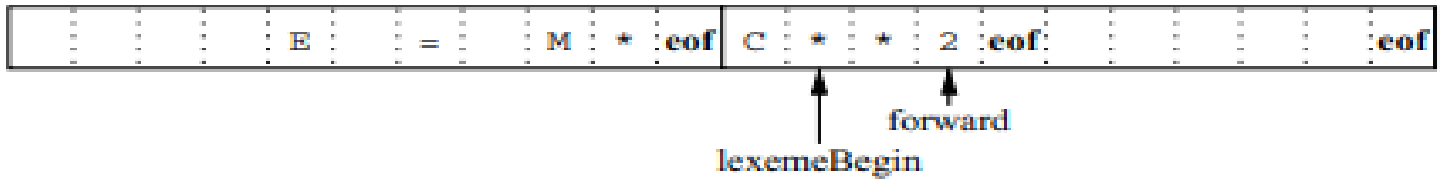
- Phương pháp Khóa cầm canh (Sentinel)
  - Phương pháp Cặp bộ đệm đòi hỏi mỗi lần di chuyển p2 đều phải kiểm tra xem đã hết một nửa buffer chưa (như vậy phải hai lần kiểm tra)
  - Phương pháp Khóa cầm canh: mỗi lần chỉ đọc N-1 ký tự vào mỗi nửa buffer còn ký tự thứ N là một ký tự đặc biệt (thường là eof). Như vậy chúng ta đã rút ngắn một lần kiểm tra.



Lưu trữ tạm thời chương trình nguồn

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

p2 dời sang phải;



if (p2 trỏ tới eof)

if (p2 trỏ tới cuối nửa đầu)

{lấp đầy nửa bên phải bởi N ký tự tiếp theo; p2 trỏ đến đầu nửa cuối};

else if (p2 trỏ tới cuối nửa sau)

{lấp đầy nửa bên trái bởi N ký tự tiếp theo; p2 trỏ đến đầu bộ đệm};

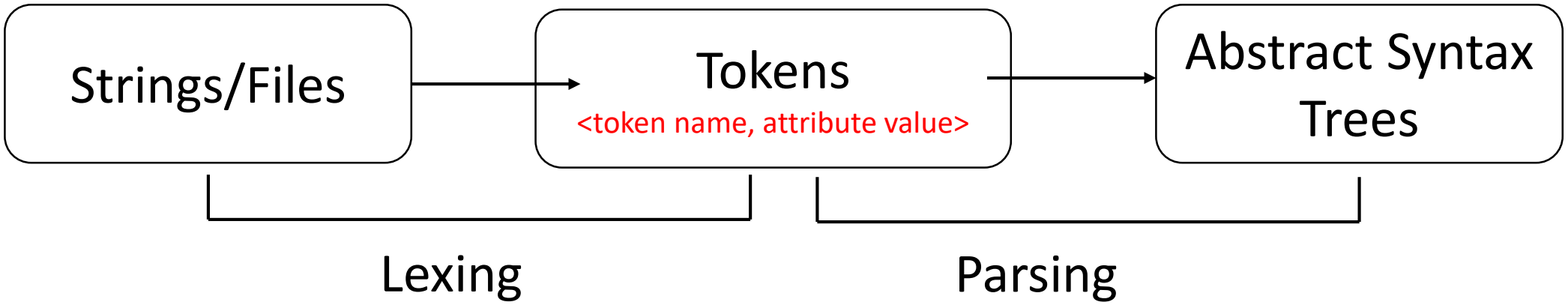
else / eof ở giữa vùng đệm chỉ hết chương trình nguồn */*

kết thúc phân tích từ vựng;

Đặc tả token (Specification of tokens)

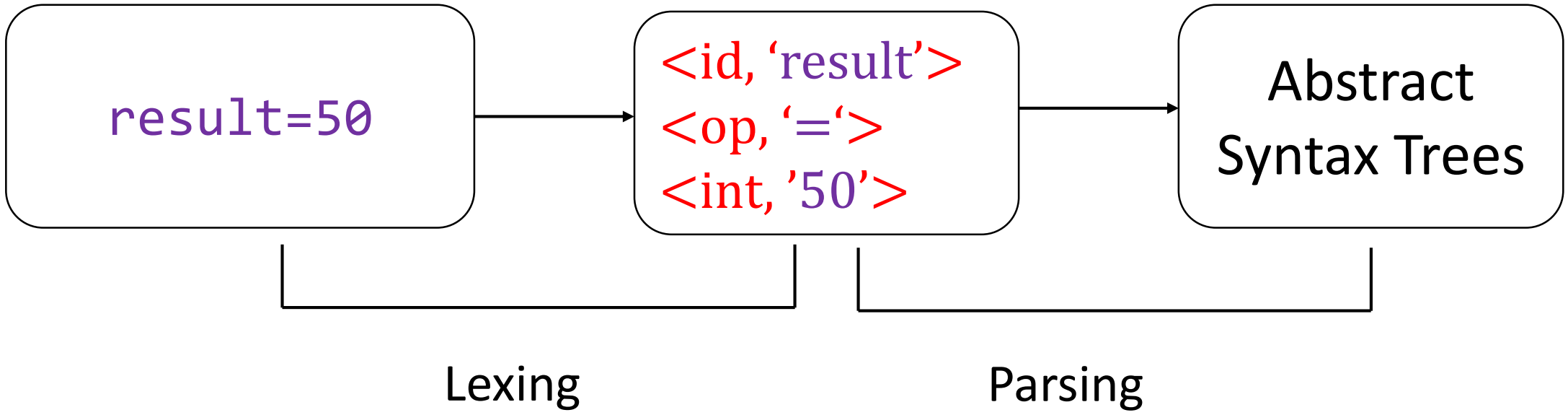
- Lớp các token (token class)
  - In English: noun, verb, adjective, …..
  - In a programming language: identifier, keyword, number, operator, open parenthesis, close parenthesis, …
- Lớp các token tương ứng với tập hợp các chuỗi ký tự
  - Identifier: Identifiers are strings of letters, digits, and underscores, starting with a letter or an underscore (num1, result, name20, _result, …..)
  - Integer: A non-empty string of digits (10, 89, 001, 00, …….)
  - Keyword: A fix set of reserved words (if, else, for, while, ….)
  - Whitespace: A non-empty sequence of blanks, newlines, and tabs

Đặc tả token (Specification of tokens)



Đặc tả token (Specification of tokens)

- Ví dụ



Đặc tả token (Specification of tokens)

- Với chuỗi đầu vào \tif(num1==num2)\n\t\tresult=1;\n\telse\n\t\tresult=0;
- Làm sao xác định được các token là các chuỗi con của chuỗi trên?
- Chúng ta cần xác định:
  - Whitespace
  - Keywords
  - Identifiers
  - Numbers
  - Operators
  - Open Parenthesis
  - Close Parenthesis
  - Semicolon
- Cần có sự đặc tả các token: sử dụng biểu thức chính quy (regular expression)

Biểu thức chính quy (Regular expression)

- Biểu thức chính quy đóng vai trò rất quan trọng cho việc biểu diễn mẫu từ vựng nhằm đặc tả token
- Mỗi mẫu từ vựng có thể so khớp với tập các chuỗi ký tự nên tên biểu thức chính quy có thể dùng làm tên cho tập các chuỗi ký tự
- Một số khái niệm và định nghĩa hình thức về ngôn ngữ:
  - Bảng chữ cái
  - Chuỗi ký tự là một dãy liên tiếp các ký tự thuộc bảng chữ cái đã cho
  - Độ dài chuỗi ký tự là số lượng các ký tự trong chuỗi.
  - Chuỗi rỗng ε là chuỗi không có ký tự nào (có độ dài 0)
  - Ngôn ngữ là tập hợp các chuỗi ký tự (ngôn ngữ rỗng ký hiệu là Ø).

Biểu thức chính quy (Regular expression)

- Các toán tử trên ngôn ngữ:

  Cho 2 ngôn ngữ L và M:

  - Hợp (Union) của L và M: $L + M = \{s \mid s \in L \text{ hoặc } s \in M\}$
  - Ghép (Concatenation) của L và M: $LM = \{st \mid s \in L \text{ và } t \in M\}$
  - Bao đóng Kleen của L: $L^* = L^0 + L^1 + L^2 + \ldots + \ldots$ (ghép của 0 hoặc nhiều L)
  - Bao đóng dương (positive closure) của L: $L^+ = L^1 + L^2 + \ldots + \ldots$ (ghép của 1 hoặc nhiều L)

Biểu thức chính quy (Regular expression)

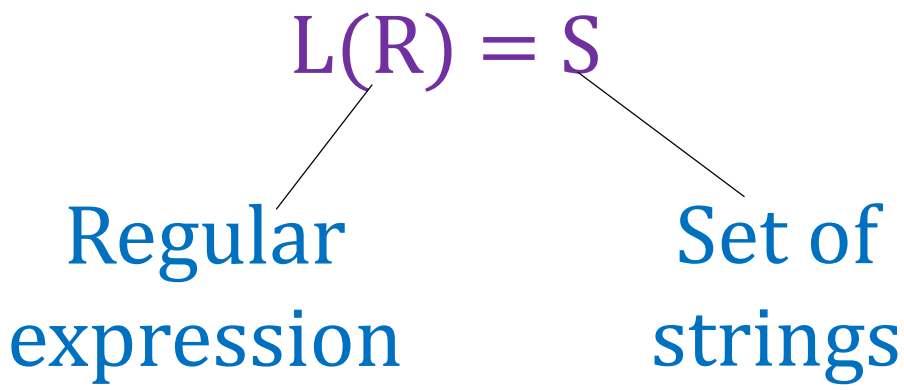
- Ví dụ: Cho L = {A, B, ..., Z, a, b, ..., z} và D = {0, 1, , ..., 9}
  - L + D là tập hợp các chữ cái và số, hay L + D ={A, B, ..., Z, a, b, ..., z, 0, 1, ..., 9}
  - LD là tập hợp các chuỗi bắt đầu bằng chữ cái và tiếp theo là một chữ số, hay

LD={A0, A1, …,A9, B0, B1,…B9, …, Z9, a0, a1,…, a9,…,z0, z1,…,z9}

  - L^3 là tập hợp tất cả các chuỗi 3 chữ cái, hay L^3={ABA, aBb, Bab,..}
  - L* là tập hợp tất cả các chuỗi gồm các chữ cái bao gồm cả chuỗi rỗng ε
  - L(L + D)* là tập hợp tất cả các chuỗi bắt đầu bằng một chữ cái theo sau là chữ cái hay chữ số, hay L(L+D)*={A, B,…,Z, a,…z, AB, A12b,…,}
  - D^+ là tập hợp tất cả các chuỗi gồm một hoặc nhiều chữ số, hay D^+={0, 1,…,9, 123, 90291758,…}

Biểu thức chính quy (Regular expression)

❑ Biểu thức chính quy R chỉ định (biểu diễn) ngôn ngữ, ký hiệu L(R)



- Single character (chuỗi gồm 1 ký tự):

  Biểu thức chính quy ‘a’ chỉ định ngôn ngữ L(‘a’)= {“a”}
- Epsilon (chuỗi rỗng):

  Biểu thức chính quy ε chỉ định ngôn ngữ L(ε) = {“”} = {ε}

Biểu thức chính quy (Regular expression)

❑ Biểu thức chính quy R chỉ định (biểu diễn) ngôn ngữ, ký hiệu L(R)

- Union (hợp):

Với A, B là hai biểu thức chính quy thì ngôn ngữ L(A)+L(B) được chỉ định bởi biểu thức chính quy A+B, tức là L(A+B) = L(A)+L(B) ={a | a∈L(A) hoặc a∈L(B)}

- Concatenation (ghép):

Với A, B là hai biểu thức chính quy thì ngôn ngữ L(A)L(B) được chỉ định bởi biểu thức chính quy AB, tức là L(AB) = L(A)L(B) = {ab | a∈L(A) và b ∈L(B)}

- Iteration (lặp):

- Với A là biểu thức chính quy thì ngôn ngữ L(A*) được chỉ định bởi biểu thức chính quy A*, tức là L(A*) = $\bigcup_{i\geq 0} L(A^i)$, A^i= A……A (*i times*) và $A^0 = \varepsilon$

- Với A là biểu thức chính quy thì ngôn ngữ L(A^+) được chỉ định bởi biểu thức chính quy A^+, tức là L(A^+) = $\bigcup_{i>0} L(A^i)$, A^i= A……A (*i times*)

Biểu thức chính quy (Regular expression)

- Định nghĩa: Một biểu thức chính quy R trên bảng chữ Σ được định nghĩa như sau (đệ quy):

R = ε

a	với $a \in \Sigma$
A + B	với A, B là các biểu thức chính quy trên Σ
AB	với A, B là các biểu thức chính quy trên Σ
A*	với A là các biểu thức chính quy trên Σ
A^+	với A là các biểu thức chính quy trên Σ

Biểu thức chính quy (Regular expression)

- Cho bảng chữ Σ = {0,1}. Lúc đó ta có:

1+0 =1 + 0 và L(1+0)={1,0}

(1+0)1 =11 + 01

và L((1+0)1)={11,01}

1* = ε + 1 + 11 + 111 + 1111 + ……..

và L(1*)={ε,1,11,111,1111,...}

0* + 1* = ε + 0 + 00 + 000 + 0000 + … + ε + 1 + 11 + 111 + 1111 + ……..

và L(0* + 1*)={ε,0,00,000,0000, …,1,11,111,1111,…..}

(0+1)* = ε + (0+1) + (0+1) (0+1) + (0+1) …… (0+1)+…

và L((0+1)*)= Σ*

L(regular_expression)

L(regular_expression) -> set of strings

a	=> L(a) = {a}
ε	=> L(ε) = {“”} = {ε}
A + B hay A | B	=> L(A + B) = L(A) ∪ L(B)
AB	=> L(AB) = {ab | a∈L(A) và b ∈L(B)}
A*	=> $L(A^*) = \bigcup_{i \geq 0} L(A^i)$
A^+	=> $L(A^+) = \bigcup_{i > 0} L(A^i)$

- Quy ước thứ tự ưu tiên thực hiện các phép toán:
  - Toán tử bao đóng *
  - Toán tử bao đóng $^+$
  - Toán tử ghép
  - Toán tử hợp + hay |

Biểu thức chính quy (Regular expression)

- Ví dụ: Cho $\sum = \{a, b\}$
  - Biểu thức chính quy a | b hay a+b chỉ định ngôn ngữ {a, b}
  - Biểu thức chính quy (a | b) (a | b) hay (a+b) (a+b) chỉ định ngôn ngữ {aa, ab, ba, bb} (ngôn ngữ này cũng có thể được chỉ định bởi biểu thức chính quy aa | ab | ba | bb)
  - Biểu thức chính quy a* chỉ định ngôn ngữ {ε, a, aa, aaa, ... }
  - Biểu thức chính quy (a | b)* hay (a+b)* chỉ định ngôn ngữ {ε, a, b, aa,bb, ...} (ngôn ngữ cũng này có thể đặc tả bởi (a*b*)*)
  - Biểu thức chính quy a | a*b hay a+a*b chỉ định ngôn ngữ {a, b, ab, aab, aaab, aaaab,... }
- Hai biểu thức chính quy r và s cùng chỉ định một ngôn ngữ được gọi là 2 biểu thức chính quy tương đương, ký hiệu r = s.

Biểu thức chính quy (Regular expression)

❑ Các tính chất đại số của biểu thức chính quy

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

Biểu thức chính quy (Regular expression)

❑ Cho bảng chữ Σ = {a,b,c}. Lúc đó ta có:

a^* = ____ , và L(a^*) ={?}

$(b+a)c$ = ____ , và L((b+a)c)={ ? }

ab^* = ____ , và L(ab^*)={ ? }

$(c+b)^*c$ = ____ , và L($(c+b)^*c$)={ ? }

$(c+b)^+a$ = ____ , và L($(c+b)^+a$)={ ? }

Biểu thức chính quy (Regular expression)

- Cho bảng chữ Σ = {a,b,0,1,2}. Lúc đó ta có:

$a(0+1+2)^*$ = ? , và L($a(0+1+2)^*$) ={?}

$(ba)^*(a+1)$ = ? , và L($(ba)^*(a+1)$) ={?}

$(a^+ + b^*)2$ =? , và L($(a^+ + b^*)2$) ={?}

$(1+2)^*a^+$ =? , và L($(1+2)^*a^+$) ={?}

$(a+b)^+(01+02)$ =? , và L($(a+b)^+(01+02)$) ={?}

Biểu thức chính quy (Regular expression)

- Cho bảng chữ Σ = {a,b,0,1,2}. Lúc đó ta có:

$a(0+1+2)^*$ = a + a0+a1+a2+a11110+a1212011+...

$(ba)^*(a+1)$ = a+1+baa+ba1+babaa+baba1+...

$(a^+ + b^*)2$ =a2+aa2+aaa2+...+2+b2+bb2+bbb2+....

$(1+2)^*a^+$
=a+aa+aaa+...+1a+1aa+1aaa+...+2a+2aa+2aaa+...+121222a+...

$(a+b)^+(01+02)$ =a01+a02+b01+b02+ababb01+aaaaab02+...

Biểu thức chính quy (Regular expression)

- Ví dụ: Cho $\sum = \{a, b, c, 1, 2\}$
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng chữ cái R=? L(R)={?}
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự kết thúc bằng chữ số R=? L(R)={?}
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng ít nhất 2 chữ số R=? L(R)={?}
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự gồm 4 ký tự và kết thúc bằng a hoặc b R=? L(R)={?}
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng chữ cái và kết thúc bằng chữ số R=? L(R)={?}

Biểu thức chính quy (Regular expression)

- Ví dụ: Xây dựng các biểu thức chính quy
  - keyword: A fix set of reserved words (“if” or “else” or “for” or …..)
  - Trên bảng chữ gồm các chữ cái
  - L={“if”,“else”, “for”,…..}

Biểu thức chính quy (Regular expression)

- Ví dụ: Xây dựng các biểu thức chính quy
  - keyword: A fix set of reserved words (“if” or “else” or “for” or …..)

Regular expression for if: ‘i”f’

Regular expression for else: ‘e”l”s”e’

Regular expression for for: ‘f”o”r’

….

Regular expression for keyword:

‘i”f’ + ‘e”l”s”e’ + ‘f”o”r’ + ……..

keyword = ‘if’ + ‘else’ + ‘for’ + ……..

Biểu thức chính quy (Regular expression)

❑Ví dụ: Xây dựng các biểu thức chính quy

- integer: a non-empty string of digits
- Trên bảng chữ gồm các chữ số
- L={0,1,2..,9,00,01234,….}

Biểu thức chính quy (Regular expression)

❑Ví dụ: Xây dựng các biểu thức chính quy

- integer: a non-empty string of digits

Regular expression for the set of strings corresponding to all the single digits

$$digit = \text{‘0’} + \text{‘1’} + \text{‘2’} + \text{‘3’} + \text{‘4’} + \text{‘5’} + \text{‘6’} + \text{‘7’} + \text{‘8’} + \text{‘9’}$$

$$integer = (digit)(digit)^* = (digit)^+$$

Biểu thức chính quy (Regular expression)

❑Ví dụ: Xây dựng các biểu thức chính quy

- identifier: strings of letters, digits, and underscores, starting with a letter or an underscore.
- Trên bảng chữ gồm các chữ cái, chữ số, '_'

Biểu thức chính quy (Regular expression)

❑Ví dụ: Xây dựng các biểu thức chính quy

- identifier: strings of letters, digits, and underscores, starting with a letter or an underscore.

digit = ‘0’ + ‘1’ + ‘2’ + ‘3’ + ‘4’ + ‘5’ + ‘6’ + ‘7’ + ‘8’ + ‘9’

letter = ‘A’+’B’+ …+’Z’+’a’+’b’+ …+’z’

identifier = (letter + ‘_’)(letter + digit + ‘_’)*

Biểu thức chính quy (Regular expression)

❑Ví dụ: Xây dựng các biểu thức chính quy

- whitespace: a non-empty sequence of blanks (‘ ‘), newlines (‘\n’), and tabs (‘\t’)

Biểu thức chính quy (Regular expression)

❑Ví dụ: Xây dựng các biểu thức chính quy

- whitespace: a non-empty sequence of blanks (‘ ‘), newlines (\n’), and tabs (‘\t’)

whitespace $=(‘\ ‘ + ‘\n’ + ‘\t’)^{+}$

Biểu thức chính quy (Regular expression)

❑Ví dụ: Xây dựng các biểu thức chính quy

- email: Make regular expression for this email address:

student@vku.udn.vn

Biểu thức chính quy (Regular expression)

❑Ví dụ: Xây dựng các biểu thức chính quy

- email: Make regular expression for this email address:

student@vku.udn.vn

letter = ‘A’+’B’+ ...+’Z’+’a’+’b’+ ...+’z’

email = (letter)$^{+}$’@’(letter)$^{+}$’.’(letter)$^{+}$’.’(letter)$^{+}$

Biểu thức chính quy (Regular expression)

❑Một số ký hiệu khác:

- At least one: AA* ≡ A^+
- Union: A | B ≡ A + B
- Option: A + ε ≡ A?
- Range: ‘a’ + ‘b’ + …+ ‘z’ ≡ [a-z]
- Excluded range: complement of [a-z] ≡ [^a-z]

Biểu thức chính quy (Regular expression)

- Ví dụ: Cho ∑ = {a, b, c,…, z, A, B, C,…, Z, 0, 1, 2,…, 9}. Tìm các biểu thức chính quy R:
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng chữ cái R=?
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự kết thúc bằng chữ số R=?
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng ít nhất 2 chữ số và kết thúc bằng chữ cái hoa R=?
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự khác rỗng mà trong đó chỉ có các chữ cái hoặc chỉ có các chữ số lẻ R=?
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự mà trong đó không có các chữ cái hoa và bắt đầu bằng chữ số R=?
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự mà trong đó có 4 ký tự chữ số và bắt đầu bằng chữ cái thường R=?
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng chữ cái hoặc kết thúc bằng chữ số R=?
  - Biểu thức chính quy R chỉ định ngôn ngữ gồm các chuỗi ký tự gồm các chữ cái mà trong đó chỉ có 3 chữ cái thường và kết thúc bằng chữ cái hoa R=?

Biểu thức chính quy (Regular expression)

❑Ví dụ:

- Number in Pascal (num): A floating point number can have some digits, an optional fraction and an optional exponent (3.15E+10, 8E-3, 1.00e+2, 2.0111e5, 15.6, 35,…)

Biểu thức chính quy (Regular expression)

❑Ví dụ:

- Number in Pascal (num): A floating point number can have some digits, an optional fraction and an optional exponent (3.15E+10, 8E-3, 15.6, 3.2e-1 …)

digit = ‘0’+’1’+’2’+’3’+’4’+’5’+’6’+’7’+’8’+’9’

digits = (digit)$^+$

opt_fraction = (‘.’digits) + ε = (‘.’digits)?

opt_exponent = ((‘E’+’e’)(‘+’ + ’-’ + ε)digits) + ε = ((‘E’+’e’)(‘+’ + ’-’)?digits)?

num = (digits)(opt_fraction)(opt_exponent)

Biểu thức chính quy (Regular expression)

- Biểu thức chính quy được sử dụng để mô tả nhiều ngôn ngữ hữu ích
- Ngôn ngữ chính quy (regular languages) được sử dụng để đặc tả ngôn ngữ
- Vấn đề đặt ra: Làm thế nào để nhận dạng các token?
- Chúng ta sử dụng biểu thức chính quy để chỉ định các lớp token và sử sụng các otomat hữu hạn (đơn định) để cài đặt việc đoán nhận.

Finite Automata

- Regular expressions = specification
- Finite automata = implementation

- A finite automata consists of
  - An input alphabet $\sum$
  - A finite set of states S
  - A start state q_0
  - A set of accepting states $F \subseteq S$
  - A set of transitions δ

Finite Automata

- Transition

$$s_1 \rightarrow^a s_2$$

- Is read

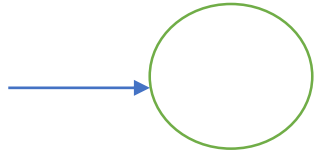
  In state s_1 on input a go to state s_2

- If end of input and in accepting state => accept
- Otherwise => reject
  - Terminates in state $s \notin F$
  - Get stuck

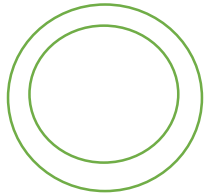
Finite Automata

- A state

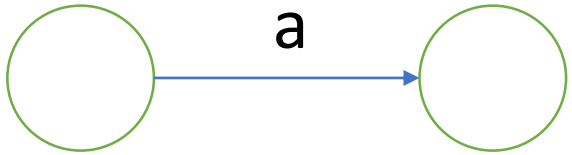
- The start state



- An accepting state

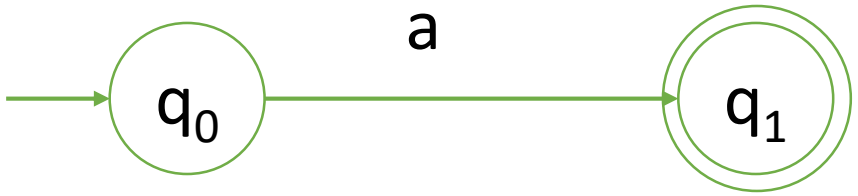


- A transition



Finite Automata

- A finite automata that accepts only “a”

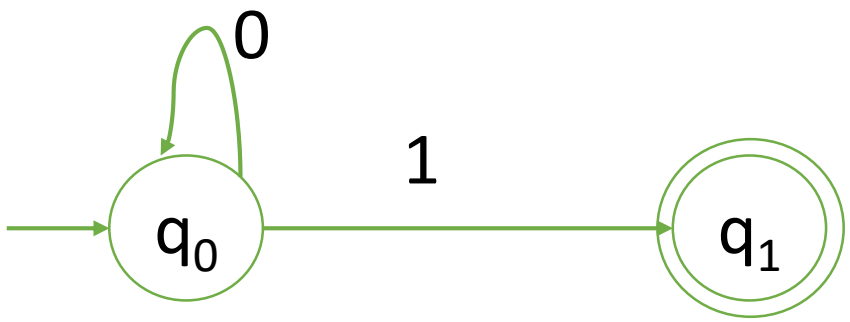


- What happen if input strings are:
  - “a”
  - “b”
  - “ab”

- Language of a finite automata is set of accepted strings.

Finite Automata

- A finite automata accepting any number of 0's followed by a single 1.



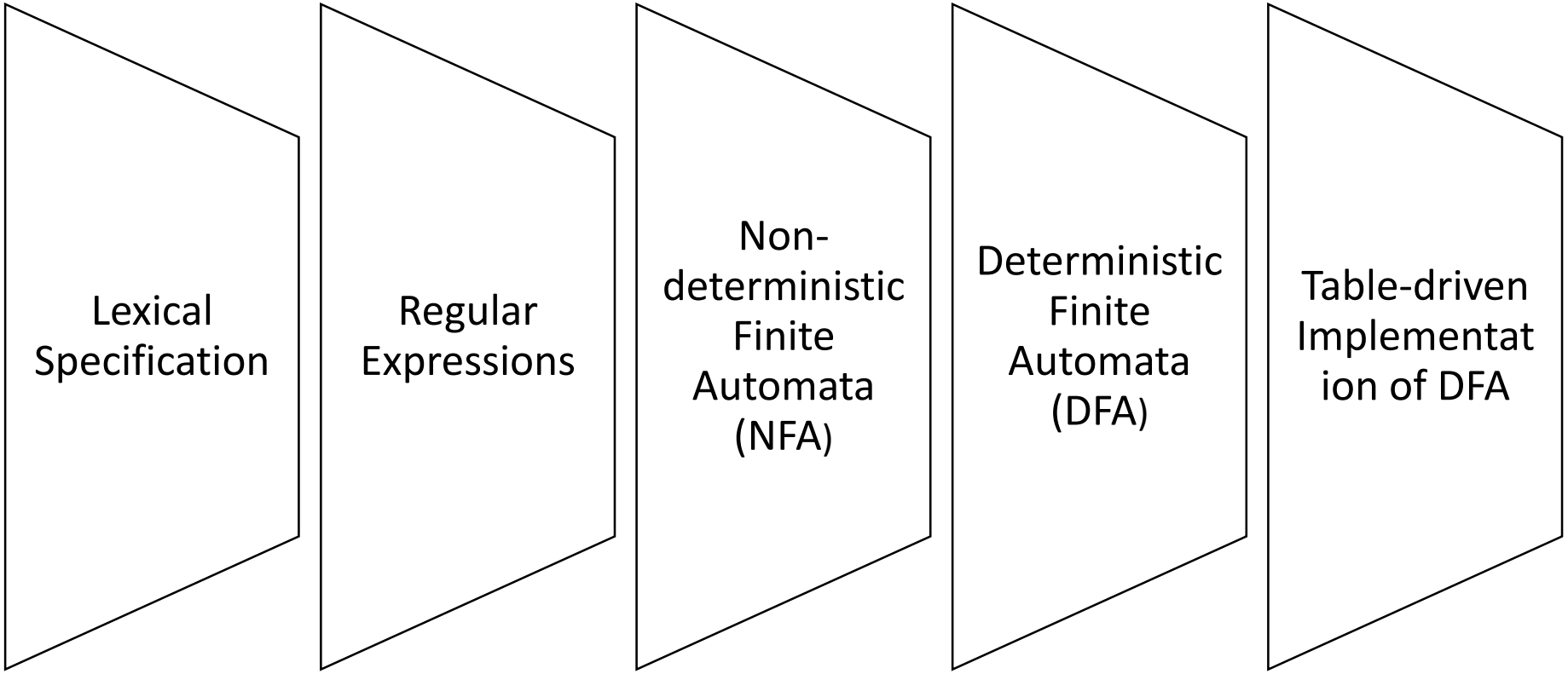
STATE	INPUT
q_0	0 0 1
q_0	0 0 1
q_0	0 0 1
q_1	0 0 1

Accept

STATE	INPUT
q_0	0 1 1
q_0	0 1 1
q_1	0 1 1

Reject

Regular Expressions to non-deterministic finite automata (NFA)

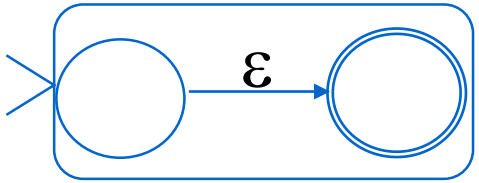


Regular Expressions to NFA

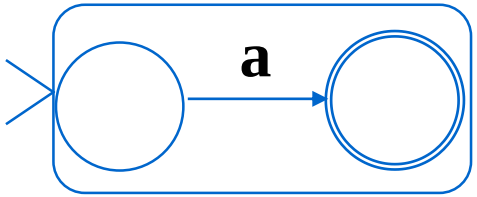
- For each kind of regular expression, define an equivalent NFA that accepts exactly the same language as the language of a regular expression.

  ⇒NFA for regular expression M

- For ε

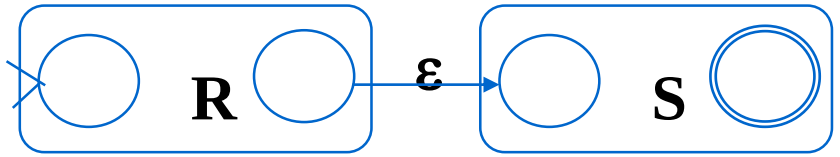


- For input ‘a’

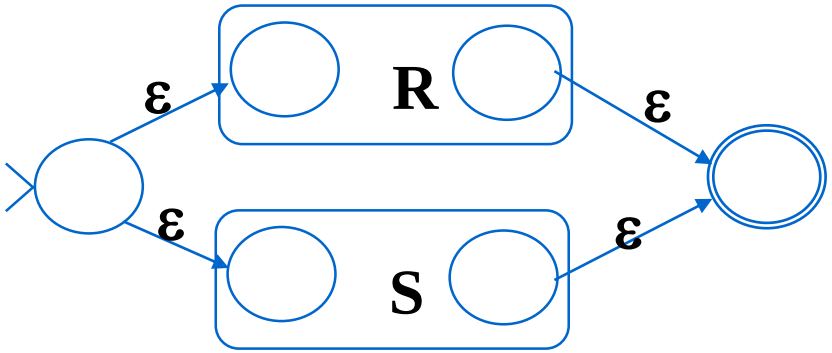


Regular Expressions to NFA

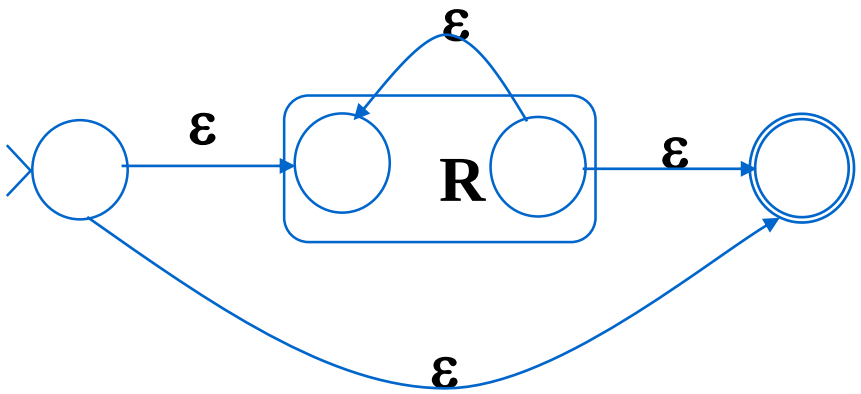
- Concatenation
  - For RS



- Union
  - For R + S

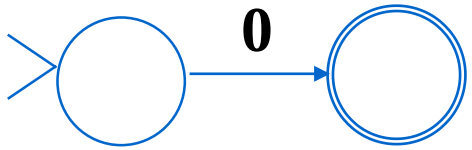


- Iteration
  - For R*

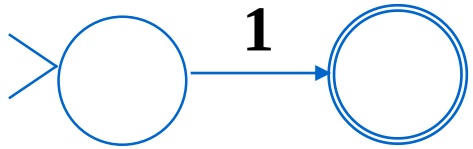


Regular Expressions to NFA

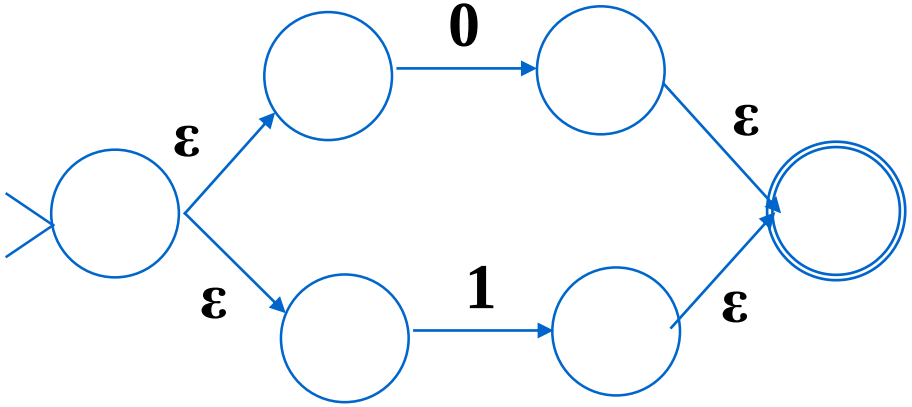
- Consider the regular expression (0+1)(01)*
  - For 0



  - For 1

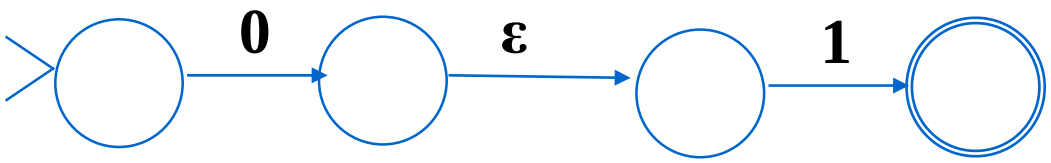


  - For 0 + 1

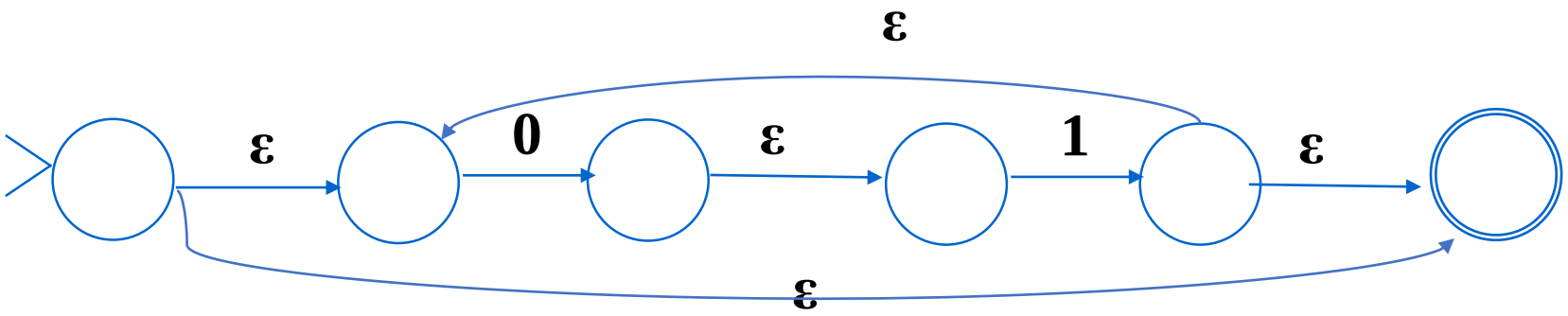


Regular Expressions to NFA

- Consider the regular expression (0+1)(01)*
  - For 01

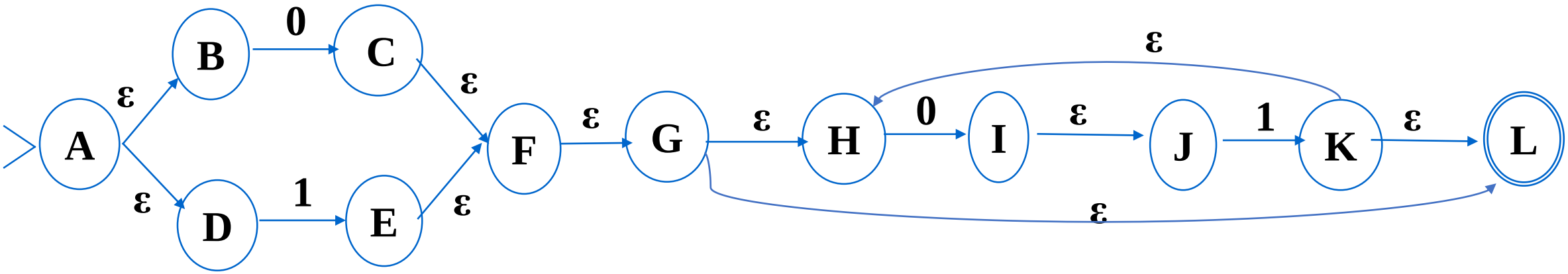


  - For (01)*

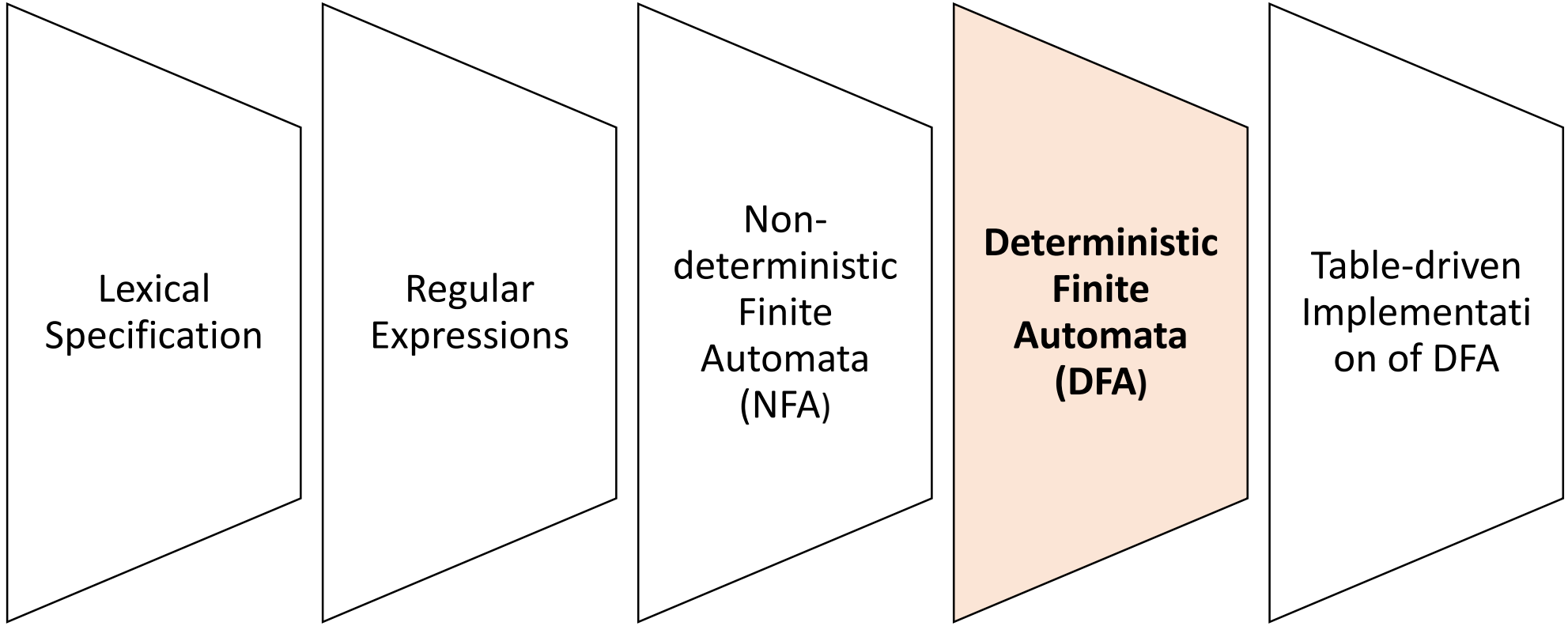


Regular Expressions to NFA

- Consider the regular expression (0+1)(01)*



Regular Expressions to non-deterministic finite automata (NFA)



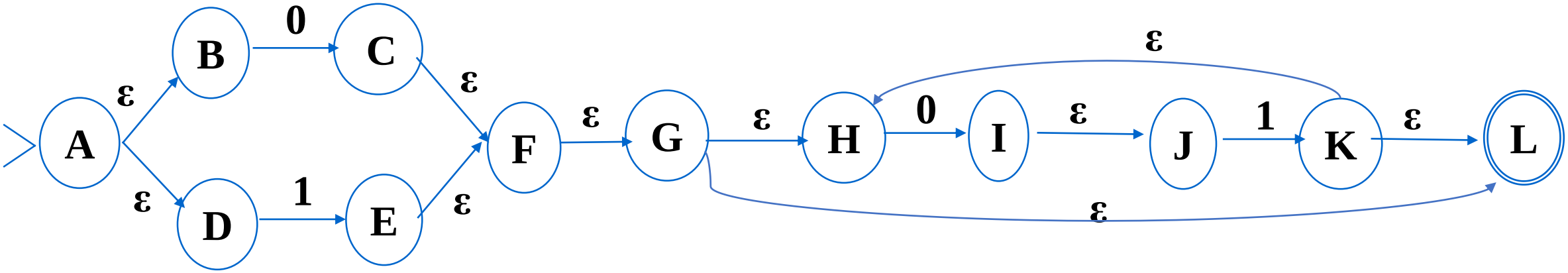
NFA to DFA

- Simulate the NFA
- Each state of DFA
  = a non-empty subset of states of the NFA
- Start state of DFA
  = the set of NFA states reachable through ε-moves from NFA start state
- Add a transition S $\xrightarrow{a}$ S' to DFA if
  – S' is the set of NFA states reachable from any
  state in S after seeing the input a, considering ε-moves as well
- Final state of DFA

  = the set includes the final state of the NFA

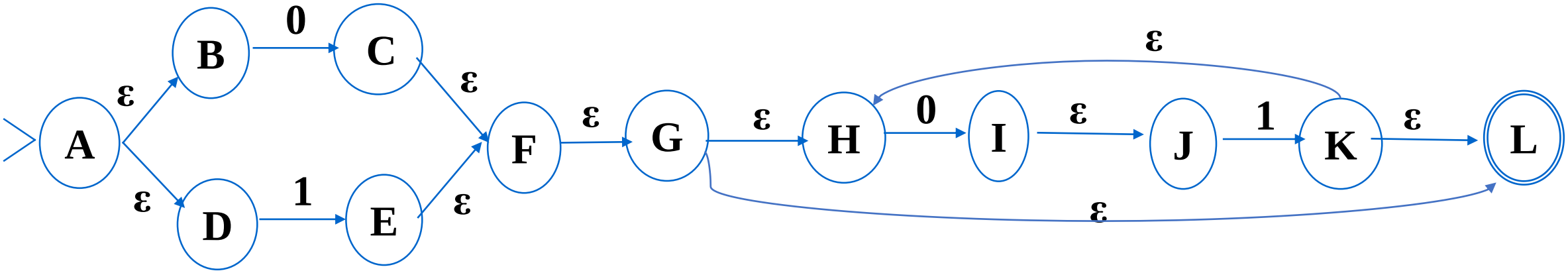
NFA to DFA

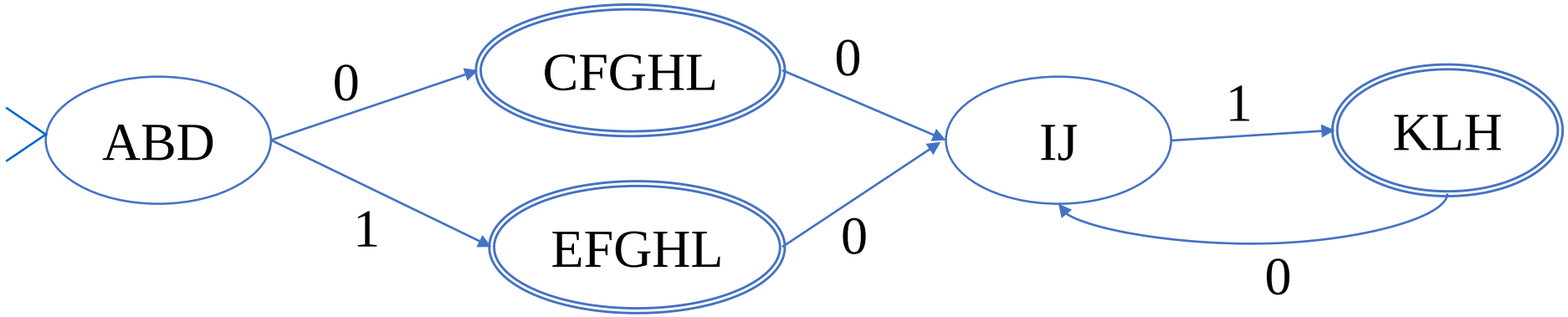
- NFA for (0+1)(01)*



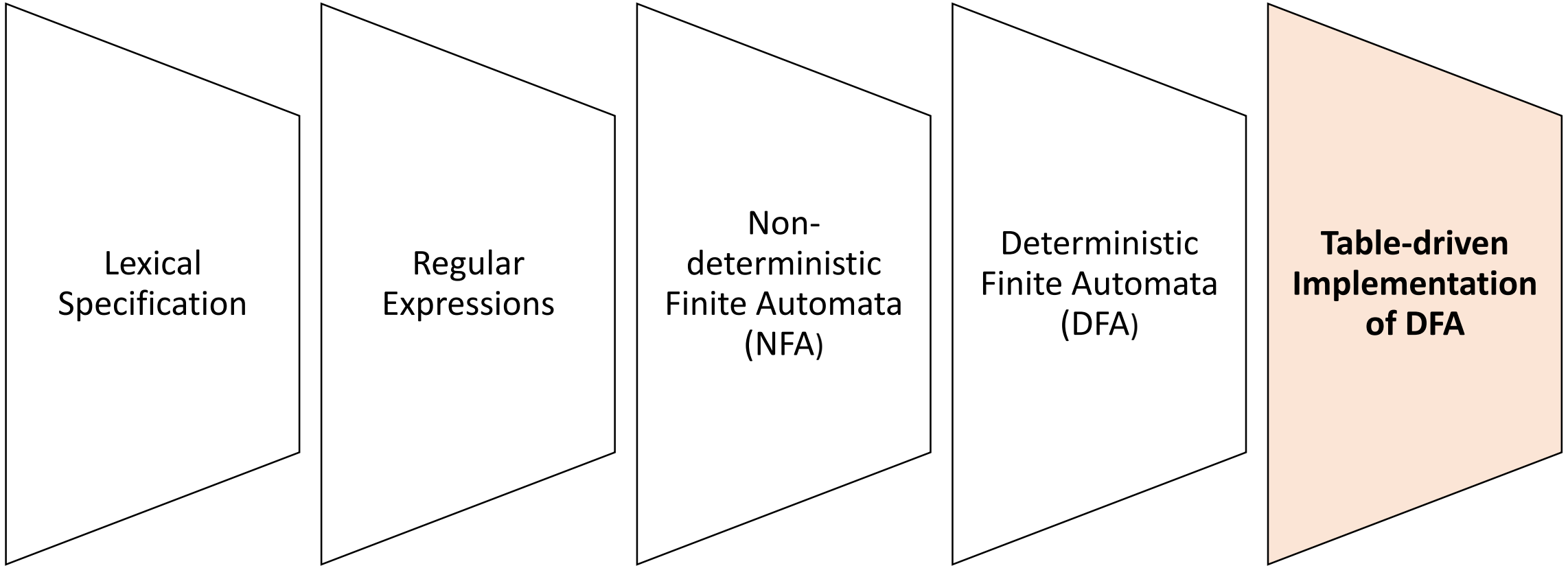
NFA to DFA

- NFA for (0+1)(01)*





Regular Expressions to non-deterministic finite automata (NFA)



Implementation of DFA

- A DFA can be implemented by a 2D table T
  - One dimension is "states"
  - Other dimension is "input symbol"

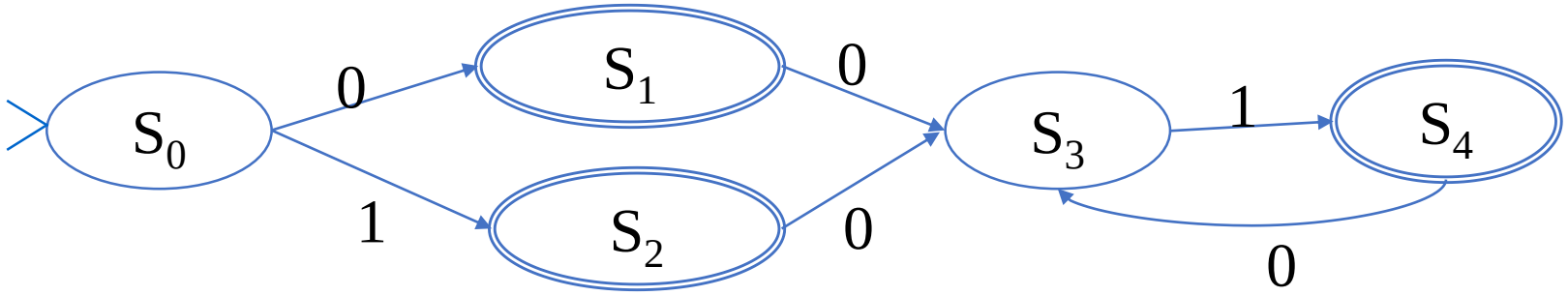
- For every transition $S_i \xrightarrow{a} S_k$ define T[i,a] = k

	Input symbols	
states		

	a	b
i	k	
j		
k		
l		

Implementation of DFA

- DFA for (0+1)(01)*



	0	1
S_0	S_1	S_2
S_1	S_3	
S_2	S_3	
S_3		S_4
S_4	S_3	

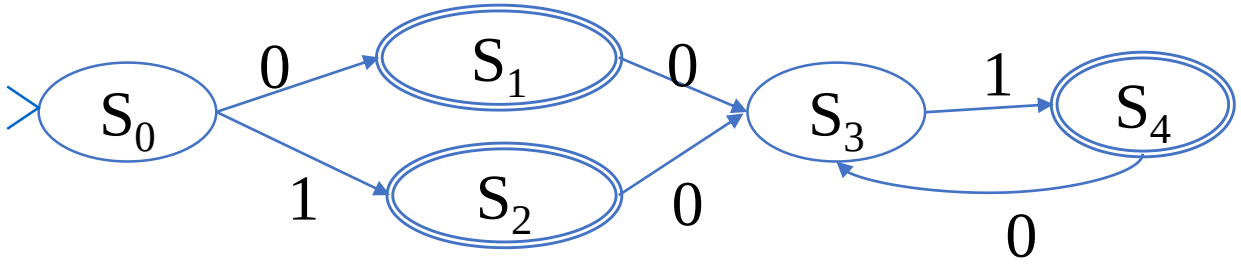
Implementation of DFA

```
i = 0;
state = 0;
while (input[i]){
   state = T[state, input[i++]];
}
```

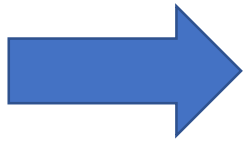
	0	1
S_0	S_1	S_2
S_1	S_3	
S_2	S_3	
S_3		S_4
S_4	S_3	

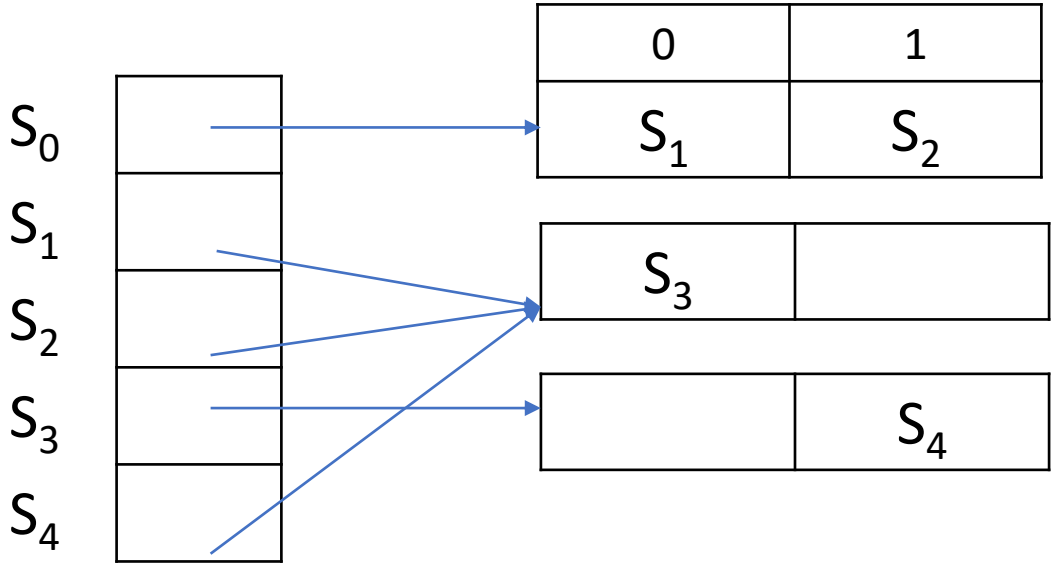
Implementation of DFA

- DFA for (0+1)(01)*



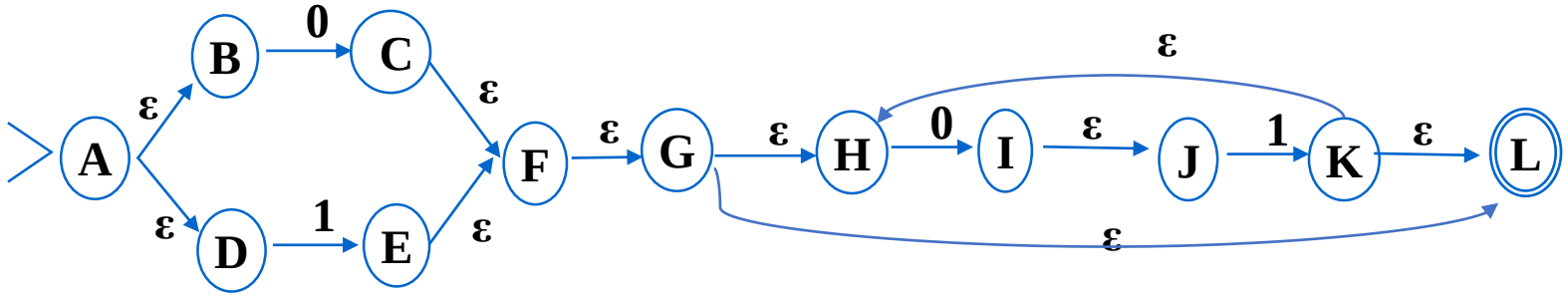
	0	1
S_0	S_1	S_2
S_1	S_3	
S_2	S_3	
S_3		S_4
S_4	S_3	





Implementation of NFA

	0	1	ε
A			{B, D}
B	{C}		
C			{F}
D		{E}	
E			{F}
F			{G}
G			{H, L}
H	{I}		
I			{J}
J		{K}	
K			{L, H}



Bài tập

1. Cho bảng chữ Σ = {1,2, b, c}. Biểu diễn R và tìm L(R):
   - R= b*(1+2)
   - R= (bc+1)c*
   - R=(1^++b*)(c+2)b
   - R= 1*(c+b)2^+
   - R= b^+(1bc+1c2)1*
2. Cho bảng chữ ∑ = {a, b, c, 0, 1}. Tìm biểu thức chính quy R:
   - R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng chữ số.
   - R chỉ định ngôn ngữ gồm các chuỗi ký tự kết thúc bằng chữ cái.
   - R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng ít nhất 3 chữ số
   - R chỉ định ngôn ngữ gồm các chuỗi ký tự gồm 3 ký tự và kết thúc bằng 0 hoặc 1
   - R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng chữ cái và kết thúc bằng chữ số
   - R chỉ định ngôn ngữ gồm các chuỗi ký tự bắt đầu bằng chữ cái hoặc kết thúc bằng chữ số

Bài tập

3. a. Xây dựng biểu thức chính quy chỉ định các lớp token trong một ngôn ngữ lập trình nào đó:

- key: A fix set of reserved words {“if”, “then”,“else”}
- num: a non-empty string of digits 0, 1, 2, 3, 4, 5
- ide: strings of letters, digits 6, 7, 8, 9 and starting with a letter
- ope: = or > or <
- pun: (or) or ;
- spa: a non-empty sequence of blanks ‘ ‘, newlines \n, and tabs \t

b. Xây dựng biểu thức chính quy R hợp của tất cả các biểu thức chính quy trên

c. Xây dựng các DFA tương ứng mỗi biểu thức chính quy chỉ định mỗi lớp token ở trên.

d. Đưa ra bảng danh biểu, bảng chuỗi các token đầu ra đối với chuỗi đầu vào

if (a6>123) then b9=5410
else c8=3412;

Bài tập

Bảng danh biểu

attribute value	*lexeme*	*token name*

Bảng chuỗi các token đầu ra

Lexeme	The following tokens are returned by scanner to parser in specified order

Bài tập

4. a. Xây dựng biểu thức chính quy chỉ định các lớp token trong một ngôn ngữ lập trình nào đó:

- word: A fix set of reserved words {“for”, “to”,“do”}
- int: a non-empty string of digits 4, 5, 6, 7, 8, 9
- id: strings of letters, digits 1, 2, 3 and starting with a letter
- opt: = or +
- punc: (or) or ;
- space: a non-empty sequence of blanks ‘ ‘, newlines \n, and tabs \t

b. Xây dựng biểu thức chính quy R hợp của tất cả các biểu thức chính quy trên

c. Xây dựng các DFA tương ứng mỗi biểu thức chính quy chỉ định mỗi lớp token ở trên.

b. Đưa ra bảng danh biểu, bảng chuỗi các token đầu ra đối với chuỗi đầu vào

```
for i = 5 to 47
do a2  = (d1 + 896);
```