Lexical Analysis

Principles of Programming Languages

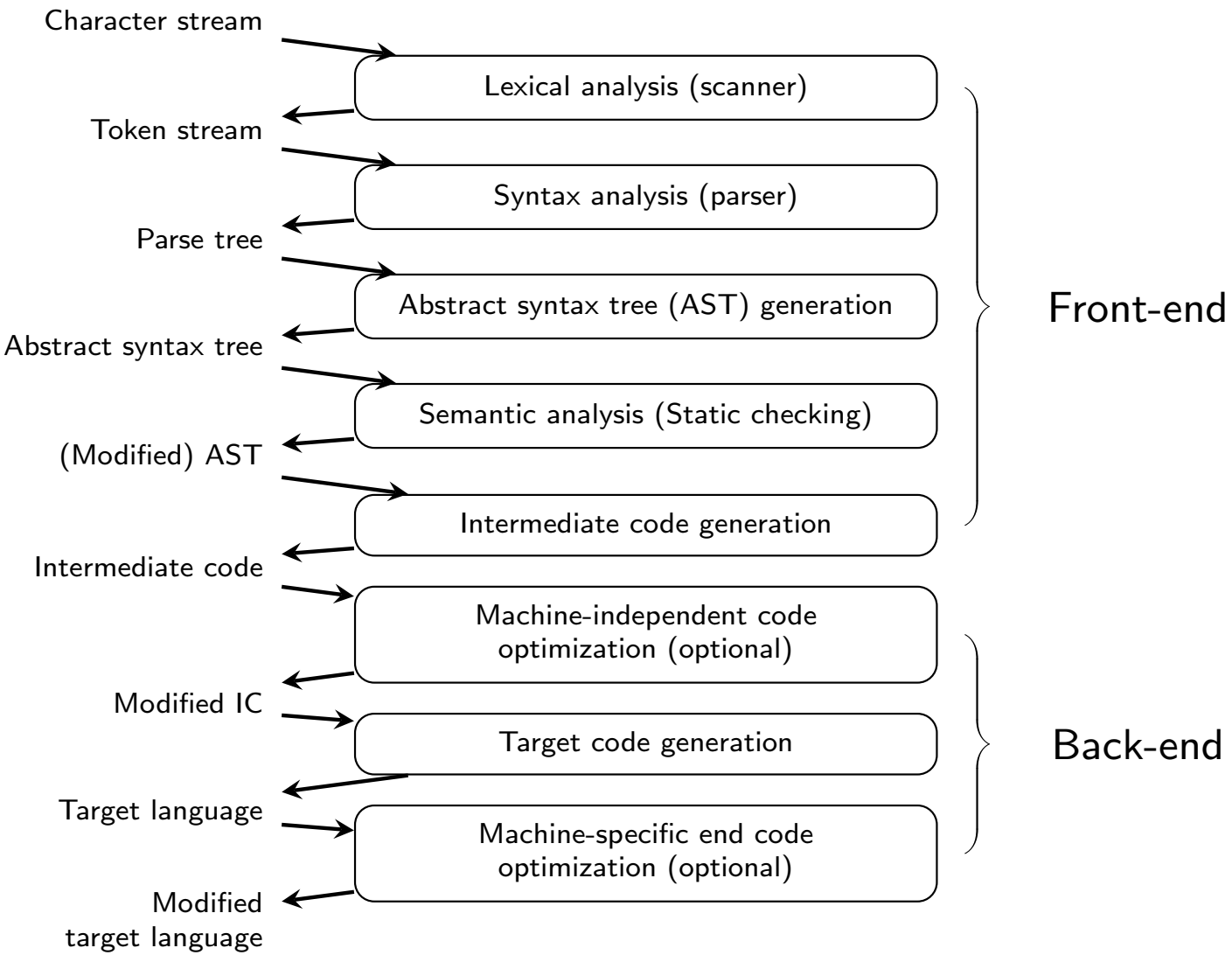
Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy
Department of Computer Science
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

TOKEN AND REGULAR EXPRESSION

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

An overview of compilation



Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Tokens

Definition

Tokens are the basic building blocks of programs — the shortest strings of characters with individual meaning, informally to refer to:

- the generic kind
- the specific string (lexeme)

1. Definition of Tokens
Tokens là các thành phần cơ bản của chương trình.
Chúng là chuỗi ký tự ngắn nhất (shortest strings of characters) có ý nghĩa riêng biệt (individual meaning).
Ví dụ:
Trong đoạn mã sau:
int x = 10;
Các token sẽ là:
int → Một từ khóa (keyword).
x → Một tên định danh (identifier).
= → Một toán tử gán (assignment operator).
10 → Một hằng số nguyên (integer literal).
; → Một dấu chấm phẩy (semicolon).

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Informal Use of Tokens
Định nghĩa token thường được dùng để chỉ:

a) The Generic Kind
Token đại diện cho loại tổng quát (generic kind) của một chuỗi ký tự.
Ví dụ:
int thuộc loại keyword (từ khóa).
x thuộc loại identifier (tên định danh).
= thuộc loại operator (toán tử).
10 thuộc loại integer literal (hằng số nguyên).
b) The Specific String (Lexeme)
Token cũng có thể ám chỉ chuỗi ký tự cụ thể (lexeme) mà nó đại diện.
Ví dụ:
Token loại keyword sẽ chứa lexeme là int.
Token loại identifier sẽ chứa lexeme là x.
Token loại integer literal sẽ chứa lexeme là 10.
3. Tóm gọn ý nghĩa của Tokens
Tokens vừa mang ý nghĩa chung (loại) vừa mang ý nghĩa cụ thể (chuỗi ký tự):

Loại (Kind): Xác định nhóm từ (keyword, identifier, operator, ...).
Chuỗi ký tự (Lexeme): Chuỗi cụ thể đại diện cho token đó.

Ví dụ minh họa cụ thể
Với đoạn mã:

float area = 3.14 * radius * radius;
Chúng ta phân tích được:

Lexeme (Cụ thể)	**Kind of Token (Loại)**
float	KEYWORD (Từ khóa)
area	IDENTIFIER (Tên định danh)
=	ASSIGN_OP (Toán tử gán)
3.14	FLOAT_LIT (Hằng số thực)
*	MULT_OP (Toán tử nhân)
radius	IDENTIFIER
;	SEMICOLON (Dấu chấm phẩy)

Tokens

Definition

Tokens are the basic building blocks of programs — the shortest strings of characters with individual meaning, informally to refer to:

- the generic kind
- the specific string (lexeme)

Example

C has more than 100 kinds of tokens:

- 44 keywords (`double`, `if`, `return`, `struct`, ...)
- identifiers (`my_variable`, `your_type`, `sizeof`, `printf`, ...)
- integer
- floating-point
- character
- ...

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Keywords (Từ khóa)

Định nghĩa:
Là các từ khóa cố định trong ngôn ngữ, dùng để biểu thị các ý nghĩa đặc biệt (ví dụ: khai báo, điều kiện, lặp, kiểu dữ liệu).
Chúng không thể được sử dụng như tên biến, tên hàm, hoặc tên định danh khác.
int, float, double, char, void, if, else, while, for, return, struct, sizeof, typedef, ...

Identifiers (Tên định danh)

Định nghĩa:
Là các tên do lập trình viên định nghĩa, dùng để đặt tên biến, hàm, kiểu, hoặc các đối tượng khác trong chương trình.
Không được trùng với từ khóa.
Quy tắc đặt tên:
Bắt đầu bằng chữ cái (hoặc _).
Có thể chứa chữ cái, chữ số, hoặc _.
my_variable, your_type, sizeof, printf

Literals (Hằng số)

Literals là các giá trị cụ thể được gán trong mã nguồn. Có nhiều loại literals:
a) Integer Literals (Hằng số nguyên) Là các số nguyên.
Ví dụ: 1, 10, 12345, 0, -5.
b) Floating-point Literals (Hằng số thực)
Là các số thực, có thể viết dưới dạng dấu chấm động.
Ví dụ: 3.14, -0.001, 1.2e3 (1.2×10^3).
c) Character Literals (Hằng ký tự)
Là các ký tự đơn lẻ, đặt trong dấu '.
Ví dụ: 'a', 'b', '\n', '1'.

Operators (Toán tử)

Định nghĩa:
Các ký hiệu dùng để thực hiện các phép toán trên dữ liệu.
Ví dụ: +, -, *, /, =, ==, <, >, &&, ||, !.

Delimiters (Ký tự phân cách)

Định nghĩa:
Các ký tự đặc biệt dùng để phân cách các phần của chương trình.
Ví dụ: ;, ,, (), {}, [].

Regular expressions

To specify tokens, we use the notation of regular expressions. A regular expression is one of the following:

1. A character.
2. The empty string, denoted ϵ.
3. Two regular expressions next to each other, meaning any string generated by the first one followed by (concatenated with) any string generated by the second one.
4. Two regular expressions separated by a vertical bar ($|$), meaning any string generated by the first one or any string generated by the second one.
5. A regular expression followed by a Kleene star ($*$), meaning the concatenation of zero or more strings generated by the expression in front of the star.

Parentheses are used to avoid ambiguity about where the various subexpressions start and end.

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

1. Một ký tự

Định nghĩa: Một biểu thức chính quy có thể chỉ là một ký tự duy nhất.

Ví dụ:

a → Chuỗi a.

5 → Chuỗi 5.

2. Chuỗi rỗng (ε)

Định nghĩa: ε đại diện cho chuỗi rỗng, nghĩa là không có ký tự nào.

Ví dụ:

Regular expression: aεb.

Kết quả: Chuỗi ab (vì không thêm gì vào chuỗi).

3. Nối (Concatenation)

Định nghĩa: Hai biểu thức chính quy được đặt cạnh nhau sẽ tạo ra chuỗi kết hợp (concatenation) của các chuỗi mà hai biểu thức đó tạo ra.

Ví dụ:

Regular expression: ab.

Kết quả: Chuỗi ab.

Regular expression: (a|b)c.

Kết quả: Chuỗi ac hoặc bc.

4. Phép chọn (Alternation - |)

Định nghĩa: Hai biểu thức chính quy được ngăn cách bởi ký hiệu | có nghĩa là một trong hai chuỗi mà biểu thức đó tạo ra.

Ví dụ:

Regular expression: a|b.

Kết quả: Chuỗi a hoặc chuỗi b.

Regular expression: (cat|dog).

Kết quả: Chuỗi cat hoặc chuỗi dog.

5. Phép lặp Kleene star (*)

Định nghĩa: Một biểu thức chính quy theo sau bởi dấu * biểu diễn chuỗi kết hợp của 0 hoặc nhiều lần xuất hiện của chuỗi được tạo ra bởi biểu thức đó.

Ví dụ:

Regular expression: a*. Kết quả: Chuỗi rỗng (0 lần a), a, aa, aaa, ...

Regular expression: (ab)*. Kết quả: Chuỗi rỗng, ab, abab, ababab, ...

Sử dụng dấu ngoặc đơn (Parentheses)

Định nghĩa: Dấu ngoặc đơn được dùng để nhóm các biểu thức con, tránh sự mơ hồ về cách các biểu thức được phân tích.

Ví dụ:

Regular expression: a|bc. Kết quả: Chuỗi a hoặc bc. Regular expression: (a|b)c. Kết quả: Chuỗi ac hoặc bc.

Regular expressions: Example

For example, the representation of numeric constants accepted by a simple hand-held calculator:

$$
\begin{aligned}
\textit{number} &\longrightarrow \textit{integer} \mid \textit{real} \\
\textit{integer} &\longrightarrow \textit{digit}\ \textit{digit}* \\
\textit{real} &\longrightarrow \textit{integer}\ \textit{exponent} \mid \textit{decimal}\ (\textit{exponent} \mid \epsilon) \\
\textit{decimal} &\longrightarrow \textit{digit}*\ (\ .\ \textit{digit} \mid \textit{digit}\ .\ )\ \textit{digit}* \\
\textit{exponent} &\longrightarrow (\texttt{e} \mid \texttt{E})\ (\texttt{+} \mid \texttt{-} \mid \epsilon)\ \textit{integer} \\
\textit{digit} &\longrightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}
\end{aligned}
$$

The symbols to the left of the $\longrightarrow$ signs provide names for the regular expressions.

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

2. Cách xây dựng

a) Biểu diễn số (number)

number -> integer | real

Một number (số) có thể là:

integer (số nguyên).

real (số thực).

Ký hiệu | nghĩa là hoặc.

b) Biểu diễn số nguyên (integer)

integer -> digit digit*

Một integer được định nghĩa là:

Một chữ số (digit) bắt buộc có.

Theo sau là 0 hoặc nhiều chữ số (biểu diễn bằng digit*).

Ví dụ:

Hợp lệ: 0, 5, 123, 45678.

Không hợp lệ: Chuỗi rỗng (không có chữ số nào).

c) Biểu diễn số thực (real)

real -> integer exponent | decimal (exponent | €)

123e5, 456E-2

3.14e2, 0.001E-3, 3.14

Một real có thể là:

Một integer theo sau bởi exponent (phần mũ).

Một decimal (phần thập phân), có thể đi kèm exponent hoặc không (€ là chuỗi rỗng).

Ví dụ:

Hợp lệ: 123e5, 0.45, 3.14E-2.

Không hợp lệ: 123. (thiếu phần thập phân sau dấu .).

d) Biểu diễn số thập phân (decimal)

decimal -> digit* (. digit | digit .) digit*

Một decimal được định nghĩa là:

0 hoặc nhiều chữ số (biểu diễn bằng digit*).

Theo sau bởi:

Một dấu chấm thập phân (.) và ít nhất một chữ số.

Hoặc ít nhất một chữ số, sau đó là dấu chấm (.).

Kết thúc bởi 0 hoặc nhiều chữ số.

Ví dụ:

Hợp lệ: 0.5, .123, 456..

Không hợp lệ: . (chỉ có dấu . mà không có chữ số).

e) Biểu diễn phần mũ (exponent)

exponent -> (e | E) (+ | - | €) integer

Một exponent gồm:

Ký tự e hoặc E.

Theo sau là:

Dấu +, -, hoặc chuỗi rỗng (€).

Cuối cùng là một integer (số nguyên).

Ví dụ:

Hợp lệ: e5, E-10, E+123.

Không hợp lệ: E (thiếu số nguyên).

f) Biểu diễn chữ số (digit)

digit -> 0|1|2|3|4|5|6|7|8|9

Một digit (chữ số) là bất kỳ ký tự nào từ 0 đến 9.

3. Tóm tắt bằng biểu thức chính quy (Regex)

Nếu muốn viết dưới dạng regular expressions hoàn chỉnh:

Số nguyên có thể kèm phần mũ

Số thập phân có thể kèm phần mũ

digit: [0-9].

integer: [0-9][0-9]*.

decimal: [0-9]*(\.[0-9]+|[0-9]+\.)[0-9]*.

exponent: [eE][+-]?[0-9]+.

real: ([0-9][0-9]*([eE][+-]?[0-9]+)?)|([0-9]*(\.[0-9]+|[0-9]+\.)[0-9]*([eE][+-]?[0-9]+)?).

123e+9, 456... - 0.45, .123e-2, .0e-5, 0.e-5

Regular expressions: Convenience notations

1. $\alpha^+ = \alpha\alpha*$
2. $\alpha? = \epsilon|\alpha$
3. [xyz] = x | y | z
4. [x-y] = α with α is one of characters from x to y in ASCII digits
5. [^x-y] = α with α is one of characters other than [x-y] in ASCII digits
6. . matches any character

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Lexer roles

Lexer, hay bộ phân tích từ vựng, là giai đoạn đầu tiên trong trình biên dịch (compiler). Nó chuyển đổi mã nguồn (source code) thành các đơn vị nhỏ hơn gọi là tokens. Cụ thể, vai trò của nó bao gồm:

1. Identify lexemes (Xác định lexeme):

Lexeme là các chuỗi ký tự (substring) trong mã nguồn thuộc về một đơn vị ngữ pháp (grammar unit).

Ví dụ: Trong dòng lệnh:

result = oldsum - value / 100;

Các lexeme có thể là:

result, oldsum, value: đại diện cho identifier.

=, -, /: đại diện cho toán tử.

100: đại diện cho hằng số nguyên (integer literal).

- Identify **lexemes**
- Return **tokens**
- Ignore **spaces** such as blank, newline, tab
- Record the **position** of tokens that are used in next phases

2. Return tokens (Trả về tokens):

Token là dạng biểu diễn trừu tượng (abstraction) của lexeme.

Mỗi lexeme được phân loại thành một loại token dựa trên ý nghĩa (semantic meaning).

Ví dụ:

Lexeme: result → Token: IDENT.

Lexeme: = → Token: ASSIGN_OP.

Lexeme: 100 → Token: INT_LIT.

3. Ignore spaces (Bỏ qua khoảng trắng):

Lexer bỏ qua các ký tự không cần thiết như:

Khoảng trắng (space).

Dấu xuống dòng (newline).

Dấu tab (tab).

Những ký tự này thường không mang ý nghĩa trong ngữ nghĩa lập trình và chỉ để định dạng mã nguồn.

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

4. Record the position of tokens (Ghi lại vị trí của token):
Lexer ghi lại vị trí (position) của mỗi token trong mã nguồn, để hỗ trợ các giai đoạn tiếp theo:
Syntax Analyzer: Phân tích cú pháp sử dụng vị trí này để kiểm tra cấu trúc của chương trình.
Error Reporting: Nếu xảy ra lỗi, vị trí của token được dùng để báo cáo lỗi (ví dụ: dòng và cột trong mã).
Debugging: Giúp lập trình viên dễ dàng xác định và sửa lỗi.

LEXER: HOW TO RECOGNIZE?

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

An esay example

Consider the following set of tokens:

Các token là những thành phần cơ bản của chương trình. Ví dụ:

assign -> :=: Dùng để gán giá trị.
plus -> +: Dấu cộng.
id: Một định danh (identifier), gồm một chữ cái ban đầu và có thể theo sau bởi chữ cái hoặc số.
number: Biểu diễn một số, có thể là số nguyên hoặc số thực.
comment: Hai kiểu comment:
/* ... */: Kiểu comment nhiều dòng như trong C.
// ...: Kiểu comment một dòng.

$$assign \longrightarrow :=$$
$$plus \longrightarrow +$$
$$minus \longrightarrow -$$
$$times \longrightarrow *$$
$$div \longrightarrow /$$
$$lparen \longrightarrow ($$
$$rparen \longrightarrow)$$
$$id \longrightarrow letter\ (letter \mid digit)^*$$
except for **read** and **write**
$$number \longrightarrow digit\ digit^* \mid digit^*\ (\ .\ digit \mid digit\ .\ )\ digit^*$$

To make the task of the scanner a little more realistic, we borrow the two styles of comment from C:

$$comment \longrightarrow /*\ (non\text{-}* \mid *\ non\text{-}/)^*\ *^+/ \mid //\ (non\text{-}newline)^*\ newline$$

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

An ad hoc scanner

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

bỏ qua các kí tự không cần thiết

Nếu ký tự hiện tại (cur_char) là một trong các ký tự đặc biệt đơn lẻ ('(', ')', '+', '-', '*'), scanner trả về token tương ứng:
(→ lparen
) → rparen
+ → plus
- → minus
* → times

```
skip any initial white space (spaces, tabs, and newlines)
if cur_char ∈ {'(', ')', '+', '-', '*'}
    return the corresponding single-character token
if cur_char = ':'
    read the next character
    if it is '=' then return assign else announce an error
if cur_char = '/'
    peek at the next character
    if it is '*' or '/'
        read additional characters until "*/" or newline is seen, respectively
        jump back to top of code
    else return div
if cur_char = .
    read the next character
    if it is a digit
        read any additional digits
        return number
    else announce an error
if cur_char is a digit
    read any additional digits and at most one decimal point
    return number
if cur_char is a letter
    read any additional letters and digits
    check to see whether the resulting string is read or write
    if so then return the corresponding token
    else return id
else announce an error
```

nếu := -> trả về token assign, còn nếu : + 1 cái gì đó khác -> lỗi

nếu hiện tại '/', kiểm tra kí tự tiếp theo
-> nếu là / -> // -> tìm newline
-> nếu là * -> /* -> tìm */
-> nếu không, return token div

nếu hiện tại là '.'
-> kiểm tra kí tự kế tiếp, nếu là một chữ số -> đọc thêm các chữ số còn lại và trả về token number
-> nếu không, báo lỗi

nếu hiện tại là một số -> đọc tiếp các chữ số và nhiều nhất một dấu chấm thập phân nếu có
-> trả về token number

nếu hiện tại là kí tự -> đọc tiếp các kí tự và chữ số khác
-> check xem phải === "read" | "write" -> trả về token read/write
-> nếu không -> trả về token id

nếu không khớp bất kì quy tắc nào ở trên -> báo lỗi

Figure: Outline of an ad hoc scanner for tokens

An ad hoc scanner

- Reasonable to check the simpler and more common cases first
- Unstructured way to build a scanner

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Finite automaton: Structured ways to build scanner

It is usually preferable to build a scanner in a more structured way, as an explicit representation of a *finite automaton*.

- It can be generated automatically from a set of regular expressions, making it easy to regenerate a scanner when token definitions change.
- **How**: The automaton starts in a distinguished initial state. It then moves from state to state based on the next available character of input. When it reaches one of a designated set of final states it recognizes the token associated with that state.
- The "longest possible token" rule means that the scanner returns to the parser only when the next character cannot be used to continue the current token.

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

1. Structured way to build scanner (Cách xây dựng scanner có cấu trúc)

Máy trạng thái hữu hạn (finite automaton) là một phương pháp có cấu trúc rõ ràng để xây dựng scanner.

Trong phương pháp này, bạn có thể mô hình hóa các tokens như là các trạng thái trong một máy trạng thái. Mỗi token (như dấu cộng +, định danh id, số thực number,...) sẽ được biểu diễn dưới dạng một trạng thái cuối trong máy trạng thái này.

Ưu điểm của cách tiếp cận này là:

Tự động hóa: Scanner có thể được tự động tạo ra từ một tập hợp các biểu thức chính quy (regular expressions). Điều này giúp bạn dễ dàng tạo lại scanner khi có sự thay đổi trong định nghĩa các token.
Cấu trúc rõ ràng: Các bước xử lý và các trạng thái đều được định nghĩa một cách chính xác và dễ hiểu.

2. Cách hoạt động của máy trạng thái hữu hạn (Finite Automaton)

Khởi đầu: Máy trạng thái bắt đầu ở trạng thái khởi tạo (initial state), đây là trạng thái đầu tiên mà máy phải luôn luôn bắt đầu.

Di chuyển giữa các trạng thái: Máy trạng thái sẽ di chuyển qua lại giữa các trạng thái tùy thuộc vào ký tự đầu vào tiếp theo mà nó nhận được. Quy trình này giống như việc quét mã nguồn ký tự theo ký tự để nhận diện các token.

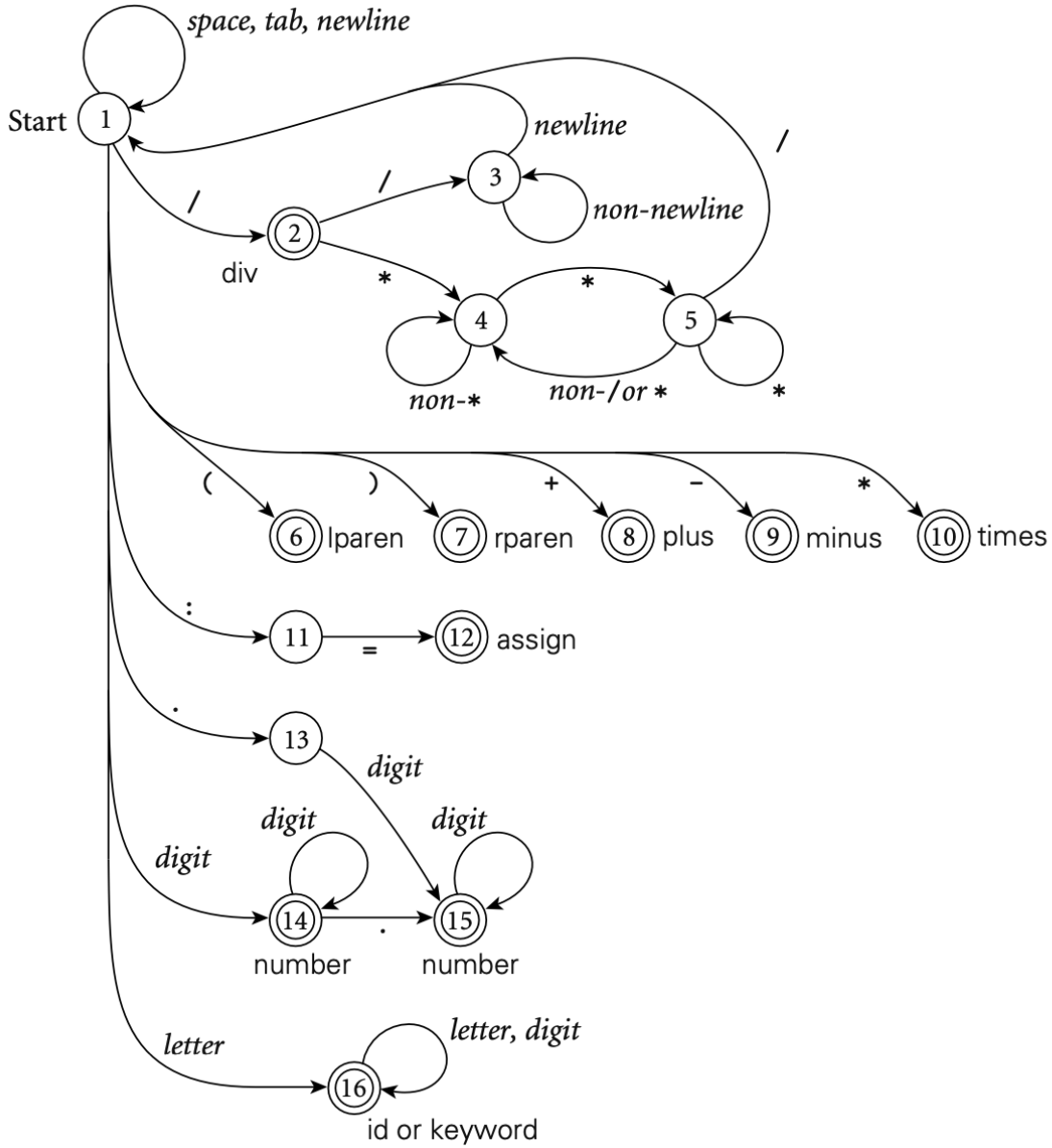
Nhận diện token: Khi máy trạng thái đạt đến một trong các trạng thái cuối (final states) đã được định nghĩa, máy sẽ nhận diện token tương ứng với trạng thái đó.

Ví dụ: Khi máy đạt đến trạng thái cuối của id, máy nhận diện rằng token hiện tại là một identifier (định danh).

3. Quy tắc "longest possible token"

Quy tắc này có nghĩa là scanner sẽ luôn cố gắng nhận diện token dài nhất có thể. Điều này có nghĩa là scanner sẽ không dừng lại khi nhận diện được một phần token mà thay vào đó, nó sẽ tiếp tục đọc ký tự tiếp theo để xem liệu có thể tạo ra một token dài hơn không.

Finite automaton: Example



Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Generating a Finite Automaton

1. Change from a Regular Expression to an NFA (Chuyển từ Biểu thức Chính quy sang NFA)
NFA (Nondeterministic Finite Automaton) là một loại máy trạng thái hữu hạn, trong đó có thể có nhiều trạng thái khả thi mà máy có thể chuyển đến từ một trạng thái cụ thể dựa trên cùng một ký tự đầu vào.

Quá trình này thực hiện việc chuyển đổi từ biểu thức chính quy (regular expression) sang NFA.

Biểu thức chính quy mô tả cú pháp của các token bằng cách sử dụng các ký hiệu như: * (lặp lại), | (hoặc), . (kết hợp).
Chuyển từ regular expression sang NFA thường được thực hiện theo cách sử dụng phương pháp thuật toán để xây dựng các trạng thái và chuyển tiếp dựa trên các ký tự trong biểu thức chính quy.
Ví dụ: Một biểu thức chính quy a|b có thể được chuyển thành một NFA có hai trạng thái, mỗi trạng thái sẽ nhận diện một ký tự a hoặc b.

1. Change from a Regular Expression to an NFA
2. Change from an NFA to a DFA
3. Minimizing the DFA

2. Change from an NFA to a DFA (Chuyển từ NFA sang DFA)
DFA (Deterministic Finite Automaton) là một loại máy trạng thái hữu hạn trong đó mỗi trạng thái có chỉ một chuyển tiếp duy nhất cho mỗi ký tự đầu vào.

Chuyển từ NFA sang DFA là một bước quan trọng vì NFA có thể có nhiều trạng thái khả thi cùng một lúc (do tính không xác định), trong khi DFA chỉ có một trạng thái duy nhất cho mỗi ký tự đầu vào.

Để chuyển một NFA sang DFA, ta sử dụng một kỹ thuật gọi là phương pháp đóng (subset construction). Phương pháp này cho phép bạn tạo một trạng thái DFA bằng cách nhóm các trạng thái của NFA lại với nhau thành một trạng thái duy nhất.

Ví dụ: Nếu NFA có hai trạng thái có thể đạt được từ một trạng thái hiện tại bằng một ký tự, thì trong DFA, bạn sẽ tạo ra một trạng thái duy nhất đại diện cho sự kết hợp của cả hai trạng thái NFA đó.

3. Minimizing the DFA (Tối ưu hóa DFA)
Tối ưu hóa DFA là bước cuối cùng trong quá trình tạo ra máy trạng thái hữu hạn. Mục đích của việc tối ưu hóa là giảm số lượng trạng thái trong DFA, làm cho DFA trở nên hiệu quả hơn trong việc xử lý đầu vào.

Phương pháp tối ưu hóa DFA thường sử dụng kỹ thuật chia nhóm các trạng thái tương đương:

Hai trạng thái được coi là tương đương nếu, từ cả hai trạng thái đó, bạn luôn nhận được các trạng thái giống nhau cho mỗi ký tự đầu vào.
Sau khi nhóm các trạng thái tương đương lại với nhau, bạn có thể giảm số lượng trạng thái trong DFA mà không làm thay đổi ngữ nghĩa của máy.
Lợi ích của việc tối ưu hóa DFA là:

Giảm số lượng trạng thái, từ đó giảm độ phức tạp và tăng tốc độ hoạt động của máy trạng thái.
Tạo ra một DFA đơn giản hơn, dễ bảo trì và hiệu quả hơn khi triển khai.

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Lexical errors

In some cases the next character of input may be neither an acceptable continuation of the current token nor the start of another token, called **lexical errors**.

- Unclosed string
- Illegal escape in string
- Error token

In such cases the scanner must print an error message.

Khi gặp phải lỗi từ vựng, scanner không thể tiếp tục phân tích bình thường. Do đó, scanner phải in ra một thông báo lỗi để thông báo cho người dùng hoặc lập trình viên về sự cố này. Thông báo lỗi này cần phải đủ rõ ràng để giúp người lập trình sửa lỗi, bao gồm các thông tin như:

Vị trí lỗi (dòng và cột trong mã nguồn)
Loại lỗi (ví dụ: chuỗi không đóng, escape không hợp lệ, token không hợp lệ)
Ví dụ thông báo lỗi:

Error: Unclosed string literal at line 3, column 5
Error: Illegal escape sequence '\w' in string at line 10
Error: Unknown token '@' at line 15, column 8

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Các loại lỗi từ vựng (Lexical errors):

Unclosed string (Chuỗi chưa đóng):

Mô tả: Khi bạn bắt đầu một chuỗi (string) nhưng không đóng chuỗi đó bằng dấu nháy kép (") hoặc dấu nháy đơn (') tương ứng.
Ví dụ:
Dòng mã sau sẽ gây lỗi: string = "Hello (thiếu dấu nháy kép đóng).
Hành động: Scanner sẽ thông báo lỗi khi phát hiện chuỗi không được đóng đúng cách.

Illegal escape in string (Escape không hợp lệ trong chuỗi):

Mô tả: Escape sequences trong chuỗi là các ký tự đặc biệt được đại diện bởi dấu gạch chéo ngược (\), chẳng hạn như \n (xuống dòng), \t (tab). Lỗi xảy ra khi gặp phải một escape sequence không hợp lệ.
Ví dụ:
string = "Hello\world" sẽ gây lỗi vì \w không phải là một escape sequence hợp lệ.
Hành động: Scanner sẽ phát hiện escape không hợp lệ và báo lỗi cho người dùng.

Error token (Token lỗi):

Mô tả: Một token lỗi xảy ra khi không có bất kỳ quy tắc nào có thể nhận diện chuỗi ký tự tiếp theo. Điều này có thể là do sự kết hợp của các ký tự không hợp lệ hoặc không thuộc bất kỳ loại token nào trong ngôn ngữ.
Ví dụ:
Dòng mã sau có thể gây lỗi: int x = @value; vì @ không phải là một ký tự hợp lệ cho bất kỳ token nào trong ngữ pháp đã định nghĩa.
Hành động: Scanner sẽ thông báo lỗi khi phát hiện một token không hợp lệ và không thể phân tích cú pháp.

ANTLR:
LANGUAGE RECOGNITION TOOL

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

ANTLR: Introduction

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

ANTLR (ANother Tool for Language Recognition) is a powerful lexer and parser generator by only writing grammar (by regular expressions) of a language.

- Author: Terence Parr, Professor of CS at the University of San Francisco, USA.
- Current version: v4 - ANTLRv4, has some important new capabilities that reduce the learning curve and make developing grammars and language applications much easier.

Các bước cơ bản để sử dụng ANTLR:
Định nghĩa ngữ pháp (grammar): Bạn bắt đầu bằng cách mô tả ngữ pháp của ngôn ngữ bằng cách sử dụng cú pháp đặc biệt của ANTLR. Ngữ pháp này sẽ bao gồm các quy tắc cho các token (lexer rules) và các quy tắc cú pháp (parser rules).

Chạy ANTLR: Sau khi đã định nghĩa ngữ pháp, bạn chạy ANTLR để nó tự động tạo ra các lexer và parser từ ngữ pháp đó.

Sử dụng Lexer và Parser: Sau khi có lexer và parser, bạn có thể sử dụng chúng để phân tích và xử lý mã nguồn của ngôn ngữ bạn đã định nghĩa.

ANTLRv4: Lexer rules

Token names must begin with an uppercase letter, which distinguishes them from parser rule names.

Syntax	Meaning
`A`	Match lexer rule or fragment named `A`
`A B`	Match `A` followed by `B`
`(A|B)`	Match either `A` or `B`
`'text'`	Match literal *"text"*
`A?`	Match `A` zero or one time
`A*`	Match `A` zero or more times
`A+`	Match `A` one or more times
`[A-Z0-9]`	Match one character in the defined ranges (in this example between A-Z or 0-9)
`'a'..'z'`	Alternative syntax for a character range
`~[A-Z]`	Negation of a range - match any single character *not* in the range
`.`	Match any single character

Figure: How to write regular expression in ANTLR

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Tên của các token phải bắt đầu bằng chữ cái in hoa. Điều này giúp phân biệt giữa token và quy tắc parser trong ANTLR.
Ví dụ: NUMBER, ID, PLUS, MINUS là tên token hợp lệ, trong khi đó các quy tắc parser sẽ có tên bắt đầu bằng chữ cái in thường (như expression, statement, v.v.).

ANTLRv4: Lexer rules

Several lexer rules can match the same input text. In that case, the token type will be chosen as follows:

- First, select the lexer rule which matches the **longest** input
- If the text matches an implicitly defined token, use the implicit rule
- If several lexer rules match the same input length, choose the first one, based on definition order

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Quy tắc chọn Token khi có nhiều quy tắc Lexer trùng lặp:
Khi có nhiều quy tắc lexer có thể khớp với cùng một chuỗi văn bản đầu vào, ANTLR sẽ lựa chọn token nào để trả về dựa trên các nguyên tắc sau:

Chọn quy tắc lexer khớp với chuỗi dài nhất:
ANTLR ưu tiên các quy tắc lexer có thể nhận diện chuỗi dài hơn, nghĩa là sẽ chọn token tương ứng với quy tắc khớp dài nhất với văn bản đầu vào.
Ví dụ: Nếu cả hai quy tắc NUMBER (số) và FLOAT (số thực) đều có thể khớp với chuỗi 123.45, và FLOAT có thể khớp với chuỗi dài hơn, thì FLOAT sẽ được chọn.

Chọn quy tắc được định nghĩa ngầm định:
Nếu có một token được định nghĩa ngầm định (implicit rule), thì ANTLR sẽ ưu tiên sử dụng quy tắc đó. Quy tắc này có thể là một quy tắc do hệ thống tự động nhận diện.

Nếu nhiều quy tắc khớp với cùng một độ dài:
Nếu có nhiều quy tắc lexer khớp với cùng một độ dài của chuỗi đầu vào, ANTLR sẽ chọn quy tắc lexer đầu tiên, dựa trên thứ tự định nghĩa các quy tắc trong ngữ pháp.

ANTLRv4 Lexer: A small example

```
grammar Hello.g4;

// match any integer literals
INTEGER: [0-9]+;

// match any identifiers
IDENTIFIER: [a-zA-Z_] [a-zA-Z_0-9]*;

// match opening and closing parentheses
OPEN_PAREN: '(';
CLOSE_PAREN: ')';
```


Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

ANTLRv4: Action blocks

A lexer action is a block of arbitrary code in the target language surrounded by {...}, which is executed during matching:

```
grammar Hello.g4;

// match any integer literals
INTEGER: [0-9]+ {print(self.text)};

// match any identifiers
IDENTIFIER: [a-zA-Z_] [a-zA-Z_0-9]*;

// match opening and closing parentheses
OPEN_PAREN: '(';
CLOSE_PAREN: ')';
```


Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

ANTLRv4: Action Blocks

Trong ANTLRv4, action blocks (khối hành động) cho phép bạn nhúng mã lệnh vào trong quy tắc lexer để thực thi khi quy tắc đó khớp với văn bản đầu vào. Các hành động này được bao quanh bởi dấu ngoặc nhọn {} và có thể chứa bất kỳ mã lệnh hợp lệ nào của ngôn ngữ đích (target language). Điều này cho phép bạn thực hiện các hành động như in ra giá trị, lưu trữ thông tin, hoặc xử lý dữ liệu ngay khi lexer nhận diện được một token.

Cấu trúc của Action Block:

Mã hành động trong lexer được bao quanh bởi dấu ngoặc nhọn {}.

Bạn có thể viết mã hành động bằng ngôn ngữ đích mà ANTLR đang tạo ra (chẳng hạn như Java, Python, v.v.).

ví dụ:

```
grammar Hello;
INTEGER : [0-9]+ { print(self.text) };     // In ra giá trị của số nguyên khi nhận diện được token INTEGER
IDENTIFIER : [a-zA-Z_] [a-zA-Z_0-9]*;      // Nhận diện bất kỳ chuỗi nào bắt đầu với ký tự chữ cái hoặc dấu gạch dưới
OPEN_PAREN : '(';                          // Nhận diện dấu mở ngoặc
CLOSE_PAREN : ')';                         // Nhận diện dấu đóng ngoặc
```

Giải thích các quy tắc trong ví dụ:

INTEGER:

Quy tắc này sẽ nhận diện các chuỗi chỉ chứa các chữ số từ 0 đến 9 và có một hoặc nhiều chữ số liên tiếp.

Sau khi nhận diện một số nguyên, mã hành động { print(self.text) } sẽ được thực thi, tức là nó sẽ in ra văn bản (số nguyên đã được nhận diện).

IDENTIFIER:

Quy tắc này nhận diện một từ khóa (identifier), bắt đầu với một ký tự chữ cái hoặc dấu gạch dưới ([a-zA-Z_]), và có thể theo sau bởi một chuỗi các ký tự chữ cái, số, hoặc dấu gạch dưới ([a-zA-Z_0-9]*).

OPEN_PAREN và CLOSE_PAREN:

Các quy tắc này chỉ đơn giản nhận diện dấu ngoặc mở (và dấu ngoặc đóng).

ANTLRv4: Fragments

Fragments trong ANTLRv4 là các phần có thể tái sử dụng của các quy tắc lexer, nhưng chúng không thể khớp với đầu vào một cách độc lập mà cần phải được tham chiếu từ các quy tắc lexer khác. Chúng được sử dụng để tái sử dụng các biểu thức chính quy trong nhiều quy tắc lexer mà không cần phải viết lại mã của chúng.

Fragments are reusable parts of lexer rules which cannot match on their own - they need to be referenced from a lexer rule.

```
grammar Hello.g4;

INTEGER: DIGIT+
       | '0' [Xx] HEX_DIGIT+
       ;

fragment DIGIT: [0-9];
fragment HEX_DIGIT: [0-9A-Fa-f];
```


Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Cách sử dụng Fragments:
Fragment là một cách để tạo ra các phần quy tắc nhỏ có thể tái sử dụng. Bạn không thể sử dụng chúng trực tiếp để nhận diện token, nhưng chúng có thể được sử dụng trong các quy tắc lexer khác.
Điều này giúp tiết kiệm mã nguồn và làm cho quy tắc lexer dễ dàng bảo trì và quản lý hơn.
Ví dụ về Fragments:

```
grammar Hello;
INTEGER : DIGIT+                    // Một hoặc nhiều chữ số (dùng fragment DIGIT)
      | '0' [Xx] HEX_DIGIT+;        // Một số hex, bắt đầu với '0x' hoặc '0X'

fragment DIGIT : [0-9];             // Fragment để nhận diện chữ số
fragment HEX_DIGIT : [0-9A-Fa-f];   // Fragment để nhận diện chữ số hex
```

INTEGER là quy tắc lexer chính để nhận diện số nguyên, bao gồm cả số hex. Quy tắc này sử dụng fragment DIGIT để nhận diện chữ số thập phân và fragment HEX_DIGIT để nhận diện chữ số hex (0-9, A-F, a-f).

DIGIT và HEX_DIGIT là fragments: chúng không thể nhận diện token một cách độc lập, nhưng có thể được sử dụng trong các quy tắc lexer khác như INTEGER.

ANTLRv4: Fragments

A lexer rule can have associated commands:

```
grammar Hello.g4;

WHITESPACE: [ \r\n] -> skip;
```

Commands are defined after a -> at the end of the rule.

- `skip`: Skips the matched text, no token will be emited
- `type(n)`: Changes the emitted token type

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

Lexer Commands (Lệnh trong Lexer)
Các lệnh trong lexer giúp bạn thay đổi hành vi của lexer khi nhận diện các token, chẳng hạn như bỏ qua một phần đầu vào hoặc thay đổi loại token. Lệnh được định nghĩa sau -> ở cuối quy tắc lexer.

Các lệnh phổ biến:
skip:
Lệnh skip cho phép bạn bỏ qua đoạn văn bản đã khớp mà không tạo ra bất kỳ token nào.
Điều này có thể hữu ích khi bạn muốn bỏ qua khoảng trắng hoặc chú thích trong mã nguồn.
Ví dụ:
```
WHITESPACE : [ \r\n\t]+ -> skip;  // Bỏ qua khoảng trắng, tab và ký tự newline
```

type(n):
Lệnh type(n) thay đổi loại token đã nhận diện và trả về token có loại khác (thường dùng khi bạn muốn nhóm các token lại).
Ví dụ:
```
COMMENT : '//' ~[\r\n]* -> type(COMMENT_TOKEN);  // Đổi loại token thành COMMENT_TOKEN
```

Các lệnh trong lexer như skip và type(n) giúp bạn kiểm soát hành vi của lexer, chẳng hạn như bỏ qua các phần văn bản không cần thiết hoặc thay đổi loại token khi cần thiết.

ANTLRv4: Reference

- https://riptutorial.com/antlr/topic/2856/introduction-to-antlr-v4
- https://github.com/antlr/antlr4/blob/master/doc/index.md
- Book: The Definitive ANTLR 4 Reference, T. Parr. Pragmatic Bookshelf, Raleigh, NC, 2 edition, (2013)

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy

THANK YOU.

Dr. Nguyen Hua Phung, MEng. Tran Ngoc Bao Duy