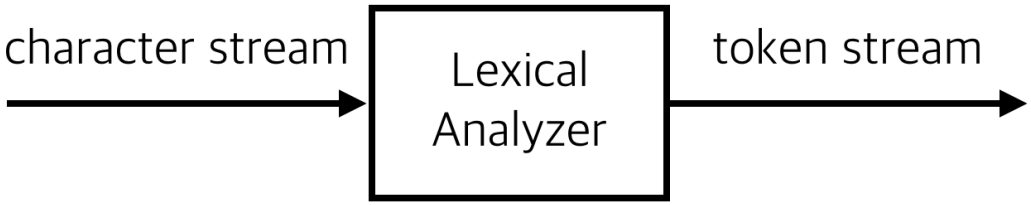
EC3204: Programming Languages and Compilers

Lecture 2 — Lexical Analysis (1)
Specification of Tokens

Sunbeom So
Fall 2024

Roadmap for Building a Lexical Analyzer



1. **Specification**: how to express valid lexeme patterns?
   - Valid C identifiers are strings like `x`, `xy`, `match0`, and `_abc`.

   $\Rightarrow$ regular expressions
2. **Recognition**: how to accept (resp., reject) the valid (resp., invalid) lexical patterns?
   - Accept `match0` as a C identifier, and reject `0match`.

   $\Rightarrow$ deterministic finite automata (DFA)
3. **Implementation**: how to implement such a string recognizer?

   $\Rightarrow$ Thompson's construction and subset construction

This Lecture: Specification of Tokens

- Preliminaries for defining a regular expression
  - alphabets, strings, languages
- Syntax and semantics of regular expressions
- Extensions of regular expressions to simplify the specification.

Preliminary: Alphabet

An alphabet Σ is a finite, non-empty set of symbols.

- $\Sigma = \{0, 1\}$ is a binary alphabet.
- $\Sigma = \{A, B, \ldots, Z\}$ is the English alphabet.
- Realistic examples: ASCII, Unicode

ASCII printable characters (character code 32-127)

Codes 32-127 are common for all the different variations of the ASCII table, they are called printable characters, represent letters, digits, punctuation marks, and a few miscellaneous symbols. You will find almost every character on your keyboard. Character 127 represents the command DEL.

DEC	OCT	HEX	BIN	Symbol	HTML Number	HTML Name	Description
32	040	20	00100000	SP	&#32;		Space
33	041	21	00100001	!	&#33;	&excl;	Exclamation mark
34	042	22	00100010	"	&#34;	&quot;	Double quotes (or speech marks)
35	043	23	00100011	#	&#35;	&num;	Number sign
36	044	24	00100100	$	&#36;	&dollar;	Dollar
37	045	25	00100101	%	&#37;	&percnt;	Per cent sign
38	046	26	00100110	&	&#38;	&amp;	Ampersand
39	047	27	00100111	'	&#39;	&apos;	Single quote
40	050	28	00101000	(	&#40;	&lparen;	Open parenthesis (or open bracket)
41	051	29	00101001	)	&#41;	&rparen;	Close parenthesis (or close bracket)
42	052	2A	00101010	*	&#42;	&ast;	Asterisk
43	053	2B	00101011	+	&#43;	&plus;	Plus
44	054	2C	00101100	,	&#44;	&comma;	Comma
45	055	2D	00101101	-	&#45;		Hyphen-minus
46	056	2E	00101110	.	&#46;	&period;	Period, dot or full stop
47	057	2F	00101111	/	&#47;	&sol;	Slash or divide
48	060	30	00110000	0	&#48;		Zero
49	061	31	00110001	1	&#49;		One
50	062	32	00110010	2	&#50;		Two

(captured from https://www.ascii-code.com)

Preliminary: String (Sentence, Word)

A string is a finite sequence of symbols from an alphabet. For example, $\mathbf{1}$, $\mathbf{01}$, $\mathbf{10110}$ are strings over $\boldsymbol{\Sigma = \{0, 1\}}$.

The following is the selected list of notations about strings.

- $\boldsymbol{\epsilon}$: the empty string.
- $\boldsymbol{wv}$: the concatenation of $\boldsymbol{w}$ and $\boldsymbol{v}$.
  - Given $\boldsymbol{w} =$ dog and $\boldsymbol{v} =$ house, $\boldsymbol{wv} =$ doghouse.
  - The empty string is the identity under the concatenation; for any string $\boldsymbol{s}$, $\boldsymbol{\epsilon s = s\epsilon = s}$.
- $\boldsymbol{|w|}$: the length of string $\boldsymbol{w}$.
  - $\boldsymbol{|\epsilon| = 0}$, $\boldsymbol{|sa| = |s| + 1}$ for some $\boldsymbol{a \in \Sigma}$.
- Regarding concatenation as a product, we can define the "exponentiation" (or "power") of strings.
  - Definition: $\boldsymbol{s^0 = \epsilon}$, $\boldsymbol{s^i = s^{i-1}s}$ for all $\boldsymbol{i > 0}$.
  - $\boldsymbol{s^1 = s}$ since $\boldsymbol{s^1 = s^0 s = \epsilon s = s}$.
  - $\boldsymbol{s^2 = ss}$, $\boldsymbol{s^3 = sss}$, and so on.

Preliminary: Language

A language L is a subset of strings over some alphabet Σ.

The following is the selected list of operations on languages.

- Union: $L_1 \cup L_2 = \{x \mid x \in L_1 \vee x \in L_2\}$
- Concatenation: $L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$
- The power of a language L, denoted L^n:

$$\begin{aligned} L^0 &= \{\epsilon\} \\ L^n &= L^{n-1} L \end{aligned}$$

- The (Kleene) closure of a language L, denoted L^*.

$$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots = \bigcup_{i \geq 0} L^i$$

  In words: the set of strings obtained by concatenating of L zero or more times.
- The positive closure of a language L, denoted L^+:

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \cdots = \bigcup_{i \geq 1} L^i$$

Regular Expression

A regular expression is a notation to denote a particular type of language.

- Syntax is inductively defined using the production rule.
  - **Base cases** define the primitive regular expressions.
  - **Inductive cases** define larger regular expressions from smaller ones.

$$\begin{array}{rcll} R & \rightarrow & \emptyset \mid \epsilon \mid a \in \Sigma & \text{(base cases)} \\ & \mid & R_1 \mid R_2 \mid R_1 \cdot R_2 \mid R_1^* \mid (R) & \text{(inductive cases)} \end{array}$$

- Semantics (meaning) is also inductively defined.

$$\begin{aligned} L(\emptyset) &= \emptyset \\ L(\epsilon) &= \{\epsilon\} \\ L(a) &= \{a\} \\ L(R_1 \mid R_2) &= L(R_1) \cup L(R_2) \\ L(R_1 \cdot R_2) &= L(R_1)L(R_2) \\ L(R^*) &= (L(R))^* \\ L((R)) &= L(R) \end{aligned}$$

Example

$$
\begin{aligned}
L(\emptyset) &= \emptyset \\
L(\epsilon) &= \{\epsilon\} \\
L(a) &= \{a\} \\
L(R_1 \mid R_2) &= L(R_1) \cup L(R_2) \\
L(R_1 \cdot R_2) &= L(R_1)L(R_2) \\
L(R^*) &= (L(R))^* \\
L((R)) &= L(R)
\end{aligned}
$$

Using the semantics above, compute the strings expressed by $a^* \cdot (a \mid b)$.

$$
\begin{aligned}
L(a^* \cdot (a \mid b)) &= L(a^*)L(a \mid b) \\
&= (L(a))^*(L(a) \cup L(b)) \\
&= (\{a\})^*(\{a\} \cup \{b\}) \\
&= \{\epsilon, a, aa, aaa, \ldots\}(\{a, b\}) \\
&= \{a, aa, aaa, \ldots, b, ab, aab, \ldots\}
\end{aligned}
$$

Exercises

Write regular expressions for the following languages.

- The set of all strings over $\Sigma = \{a, b\}$.
  - ▶
- The set of strings of a's and b's, terminated by ab.
  - ▶
- The set of strings with an even number of a's followed by an odd number of b's.
  - ▶
- The set of C identifiers.
  - ▶

Exercises

Write regular expressions for the following languages.

- The set of all strings over $\Sigma = \{a, b\}$.
  - $(a \mid b)^*$
- The set of strings of a's and b's, terminated by ab.
  - $(a \mid b)^* ab$
- The set of strings with an even number of a's followed by an odd number of b's.
  - $(a \cdot a)^* (b \cdot b)^* b$
- The set of C identifiers.
  - $(\texttt{A} \mid \texttt{B} \mid \cdots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \mid \texttt{\_}) \cdot (\texttt{A} \mid \texttt{B} \mid \cdots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \mid \texttt{\_} \mid \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9})^*$

    Alternatively, using **regular definition**,

$$
\begin{aligned}
\mathit{letter} &\rightarrow \texttt{A} \mid \texttt{B} \mid \cdots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \mid \texttt{\_} \\
\mathit{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
\mathit{id} &\rightarrow \mathit{letter}(\mathit{letter} \mid \mathit{digit})^*
\end{aligned}
$$

Regular Definitions

For notational convenience, we give names to regular expressions and use the names in subsequent expressions. A **regular definition** is a sequence of definitions of the form:

$$\begin{aligned} d_1 &\rightarrow r_1 \\ d_2 &\rightarrow r_2 \\ &\cdots \\ d_n &\rightarrow r_n \end{aligned}$$

1. Each d_i is a new name such that $d_i \notin \Sigma$.
2. Each r_i is a regular expression over $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$.
   That is, we restrict r_i to Σ and previously defined regular definitions.

Example

- Unsigned numbers (integers or floating point), e.g., `5280`, `0.01234`, `6.336E4`, or `1.89E-4`:

$$
\begin{aligned}
\textit{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
\textit{digits} &\rightarrow \textit{digit}\ \textit{digit}^* \\
\textit{optionalFraction} &\rightarrow \texttt{.}\ \textit{digits} \mid \epsilon \\
\textit{optionalExponent} &\rightarrow (\texttt{E}\ (\texttt{+} \mid \texttt{-} \mid \epsilon)\ \textit{digits}) \mid \epsilon \\
\textit{number} &\rightarrow \textit{digits}\ \textit{optionalFraction}\ \textit{optionalExponent}
\end{aligned}
$$

Extensions of Regular Expressions

Extensions are used to simplify the specification.

- R^+: the positive closure of R, i.e., $L(R^+) = L(R)^+$.
- $R?$: zero or one instance of R, i.e., $L(R?) = L(R) \cup \{\epsilon\}$.

To represent character classes, we use the following extensions.

- $[a_1 a_2 \cdots a_n]$: the shorthand for $a_1 \mid a_2 \mid \cdots \mid a_n$.
- $[a_1\text{-}a_n]$: the shorthand for $[a_1 a_2 \cdots a_n]$, where $a_1, \ldots, a_n$ are consecutive symbols.
  - $[abc] = a \mid b \mid c$
  - $[a\text{-}z] = a \mid b \mid \cdots \mid z$.

Examples

- C identifiers:

$$\begin{aligned} \textit{letter} &\rightarrow \texttt{[A-Za-z\_]} \\ \textit{digit} &\rightarrow \texttt{[0-9]} \\ \textit{id} &\rightarrow \textit{letter}\ (\textit{letter} | \textit{digit})^* \end{aligned}$$

- Unsigned numbers:

$$\begin{aligned} \textit{digit} &\rightarrow \texttt{[0-9]} \\ \textit{digits} &\rightarrow \textit{digit}^+ \\ \textit{number} &\rightarrow \textit{digits}\ (\texttt{.}\ \textit{digits})?\ (\texttt{E}\ \texttt{[+-]}?\ \textit{digits})? \end{aligned}$$

Summary

We learned how to express valid lexeme patterns (i.e., define valid inputs of lexical analyzers).

- Preliminaries for defining a regular expression
  - alphabets, strings, languages
- Syntax and semantics of regular expressions
- Extensions of regular expressions to simplify the specification.

Next class: finite automata for accepting (resp., rejecting) the valid (resp., invalid) lexical patterns.