COMP3173 Compiler Construction
Regular Expression

Dr. Zhiyuan Li

Outline

- Formal language theory
- Regular Expression
- Lexical Analysis

- Read Dragon book Chapter 3.1 and 3.4

Alphabet

- An ***alphabet*** for a language is a set of symbols.
- Usually, we use Greek letter Σ to denote the alphabet.
- Here, the language, in general, is not restricted. It can be a formal language, a programming language, a natural language, etc.
- For example,
  - the C programming language uses ASCII alphabet;
  - Java uses Unicode;
  - English uses ‘a’-‘z’, ‘A’-‘Z’, and punctuations;
  - some languages like Chinese have a large alphabet;
  - and some other languages like binary have a small alphabet.

Strings

- A ***string*** over an alphabet is defined as a ***finite*** sequence of symbols from the alphabet.
- This concept is more general than the strings in a program (the sequence between double-quotations, like "…").
- The ***length*** of a string s, denoted by $|s|$, is the number of symbols in s.
- There is a special string of length zero denoted by ε, also called ***empty string***.

Strings

- A ***prefix*** of a string s is formed by removing zero or more symbols from the end of s.
- Similarly, a ***suffix*** if a string is formed by removing zero or more symbols from the beginning. A ***substring*** is formed by removing both a prefix and a suffix. (The prefix or suffix can be empty).
- A ***proper prefix / suffix / substring*** of a string s is a non-empty string t, such that t is a prefix / suffix / substring of s; and s and t are different.
- Every string is a prefix / suffix / substring of itself.

Strings

- For example, for the string “Goliath”

Prefix:
- Goliath
- Goliat
- Golia
- Goli
- Gol
- Go
- G
- ε

Suffix:
- Goliath
- oliath
- liath
- iath
- ath
- th
- h
- ε

Proper Prefix:
- Goliat
- Golia
- Goli
- Gol
- Go
- G

Proper Suffix:
- oliath
- liath
- iath
- ath
- th
- h

Substrings:
- Goliath
- Goliat
- oliath
- olia
- liat
- li

Strings

- The ***concatenation*** of two strings s and t is the string st obtained by appending t to the end of s.
- The ***exponentiation*** of a string s is defined recursively
  - $s^0 = \varepsilon$
  - $s^i = s^{i-1}s$ for $i > 0$
- For example, if s =“Goliath”, t =“Li”, then st =“GoliathLi” and t^3 =“LiLiLi”.

Language

- A ***language*** over an alphabet is a set of strings.
- The size of a language can be infinite.
- Example:
  - $\emptyset$ is a language which contains no string.
  - $\{\varepsilon\}$ is a language which only contains an empty string.
  - $\{\varepsilon, 0, 00, 000\}$ is a language over an alphabet $\Sigma = \{0\}$ which contains 4 strings.
  - $\{\varepsilon, 0, 00, 000, \cdots\}$ is an infinite language which consists of strings of 0's.

Language

- Since a language is a set, we can easily define that the ***union*** $L \cup M$ of two languages L and M is the set of string s such that s is either in L or in M.
- Formally, $L \cup M = \{s | s \in L \vee s \in M\}$.
- The ***concatenation*** LM of two languages L and M is the set of strings st such that s is a string in L and t is a string in M.
- Formally, $LM = \{st | s \in L \wedge t \in M\}$. (Similar to the Cartesian product of two sets)
- The ***exponentiation*** L^i can also be defined recursively.
  - $L^0 = \{\varepsilon\}$, note that $L^0 \neq \emptyset$.
  - $L^i = L^{i-1}L$

Language

- Assume $L = \{a, ab\}$ and $M = \{b, bb\}$ are two languages over the alphabet $\Sigma = \{a, b\}$. Then,
- $L \cup M = \{a, b, ab, bb\}$
- $LM = \{ab, abb, abbb\}$ (The string abb has a duplication.)
- $L^2 = \{aa, aab, aba, abab\}$

Kleene Closure

- The ***Kleene Closure*** (named after Stephen C. Kleene) L^* of a language L is a language defined as $L^* = \cup_{i=0}^{\infty} L^i$.
- In other words, $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$.
- For example, $L = \{aa, b\}$ is a language over the alphabet $\Sigma = \{a, b\}$. $L^* = \{\varepsilon, aa, b, aab, aaaa, baa, bb, \cdots\}$.

- The Kleene Closure can also be defined on an alphabet Σ, if we regard Σ is a language such that each string is of length 1.
- For example, if $\Sigma = \{0,1\}$, then Σ^* is the set of all possible strings of 0's and 1's, in other words the set of binaries.

- Note: The Kleene Closure always contains ε.

Formal Language

- A ***formal*** language over an alphabet Σ is a language such that there is a ***method*** which can justify whether the given string over Σ is in the language or not (informal definition).
- This method can be a machine/computing device (hardware) or an algorithm (software).
- And we say the machine ***recognizes*** the language.
- Thus, a formal language is ***equivalent*** to
  - a ***machine***, which tells you whether an input string is in the language or not;
  - also, a ***grammar***, a set of production rules to produce ***every*** string in the language.
- This concept is called ***soundness*** and ***completeness***, same as Armstrong's axioms in the functional dependency theory (COMP3013 DBMS) and functionally complete (MATH2003).

Formal Language

- In this course, we will see
  - regular language, for lexical analysis
  - context-free language, for syntax analysis
  - context-sensitive language, for semantic analysis.

Language	Grammar	Machine	Chomsky hierarchy
Regular	Regular grammar / Regular expression	Finite automata	Type-3
Context-free	Context-free grammar	Push-down automata	Type-2
Context-sensitive	Context-sensitive grammar	Linear bounded automata	Type-1
Recursive	Recursive grammar	Decider (TMs always stop)	-
Recursively Enumerable	Unrestricted grammar	Turing Machine (May not stop)	Type-0

Regular Expression

- A regular expression is a ***search pattern*** for all the strings in a regular language.
- A language is ***regular*** if it is defined by a regular expression.
- The regular expression over an alphabet Σ is recursively constructed.

Base case:

- ε is the regular expression that denotes the language $\{\varepsilon\}$.

  Note: even we are using the same symbol, but please don't be confused by the regular expression ε, the string ε, and the language $\{\varepsilon\}$.
- If a is a character in Σ, then a is a regular expression denoting the language $\{a\}$. Similarly, the regular expression a, the character a, and the string a are of different types.

Regular Expression

Recursion:

- If r and s are regular expressions for the languages $L(r)$ and $L(s)$ respectively (may contains one symbol like the ones above may also be a long expression).
  - $(r)|(s)$ denotes the language $L(r) \cup L(s)$. The symbol | in a regular expression acts as an "or" operator.
  - $(r)(s)$ denotes the language $L(r)L(s)$, the concatenation.
  - $(r)^*$ denotes the language $\big(L(r)\big)^*$. The Kleene closure $*$, also called Kleene star means "zero or more instances of".

Regular Expression

- To simplify the expressions, we have the followings.
  - (r) denotes the same language $L(r)$ as r does. Extra parentheses do not matter in regular expressions.
  - $(r)^+ = rr^*$ meaning “at least one instances of”.
- Because there are three different operators in the regular expression, we define the precedence.
  - The $*$ operator has the highest precedence.
  - Then, concatenation.
  - The | is the lowest.
  - All these three operators are left associative, doing operations from left to right.

Regular Expression

- These operators have the following algebraic properties:
  - $r|s = s|r$, the operator | is **commutative**.
  - $r|(s|t) = (r|s)|t$, the operator | is **associative**.
  - $r(st) = (rs)t$, concatenation is **associative**.
  - $r(s|t) = rs|rt$ and $(s|t)r = sr|tr$, concatenation **distributes** over |.
  - $\varepsilon r = r = r\varepsilon$, the regular ε is the **identity** element for regular expression concatenation. (But it is not the identity for |.)
  - $r^* = (r|\varepsilon)^*$
  - $r^{**} = r^*$, the Kleene star is **idempotent**.

Regular Expression

Some examples, if $\Sigma = \{0,1\}$

- $0^*10^* = \{w|w$ contains a single $1\}$.
- $\Sigma^*1\Sigma^* = \{w|w$ contains at least one $1\}$.
- $(01^+)^* = \{w|$ every 0 in w is followed by at least one $1\}$.
- $(\Sigma\Sigma)^* = \{w|w$ is a string of even length$\}$.
- $(\Sigma\Sigma\Sigma)^* = \{w|$ the length of w is a multiple of three$\}$.
- $(0|1)(0|1) = 0(0|1)|1(0|1) = 00|01|10|11 = \{00,01,10,11\}$.

The last example shows that the regular expression for a language may not be unique.

Regular definition

- To simplify the regular expressions, we also use ***regular definitions*** (regular grammar).
- Assume Σ is the alphabet, a regular definition is a sequence of ***production rules*** in the following form:

$$d_1 \rightarrow r_1$$
$$d_2 \rightarrow r_2$$
$$\cdots$$
$$d_n \rightarrow r_n$$

where

1. each d_i is a new symbol, not in Σ and not the same as any other d's,
2. each r_i is a regular expression over the alphabet $\Sigma \cup \{d_1, d_2, \cdots, d_{i-1}\}$.

- The grammar for other formal languages may not satisfy the above conditions, context-free grammars for example.

Regular definition

- For example, the regular expression on the alphabet
$$\Sigma = \{+, -, 0,1,2,3,4,5,6,7,8,9, .\}$$
for signed rational numbers:
$$(+|-|\varepsilon)(0|1|2|3|4|5|6|7|8|9)^{+}(\varepsilon|.(0|1|2|3|4|5|6|7|8|9)^{+})$$
- This is complicated. But by regular definition

$$\begin{aligned} digit &\rightarrow 0|1|2|3|4|5|6|7|8|9 \\ natural &\rightarrow digit^{+} \\ unsigned\_rational &\rightarrow natural(\varepsilon|.natural) \\ signed\_rational &\rightarrow (+|-|\varepsilon)unsigned\_rational \end{aligned}$$

Regular definition

- To further simply the production rules, we can define some new notations.
  - $r? = r|\varepsilon$, the symbol ? means "zero or one instance of".
  - $0|1|2|3|4|5|6|7|8|9 = [0-9]$, classes of characters can be abbreviated.

- The previous example then can be simplified as

$$
\begin{aligned}
digit &\rightarrow [0-9] \\
natural &\rightarrow digit^{+} \\
unsigned\_rational &\rightarrow natural(.natural)? \\
signed\_rational &\rightarrow (+|-)?\, unsigned\_rational
\end{aligned}
$$

Properties

- Regular expressions are equivalent to regular definitions.
  - The languages defined by regular expressions are same as the languages defined by regular definitions.
  - Every regular expression can be converted to a regular definition which defines the same language and vice versa.
- Sketch of the proof:
  - For the regular expression r, trivially construct the regular definition of one production rule: $d \rightarrow r$.
  - For the regular definition $d_1 \rightarrow r_1, \cdots, d_n \rightarrow r_n$, simply substitute each d_i on the righthand side of $d_{i+1} \rightarrow r_{i+1}$ by r_i. Eventually, we can eliminate all d's in r_n. Then, r_n is the equivalent regular expression.

- Regular language is ***closed*** under union.
  - The union of two regular language is a regular language.
  - True because of the definition of the operator | in the regular expression.

Properties

- The regular language/expression/definition is not ultimate. Some languages are not regular.
- For example, $\{w|w$ begins with 0's and followed by the same number of 1's$\}$.
- One may try
  - the regular expression $r = 0^n1^n$, but exponential is undefined in regular expressions; or
  - the regular definition $d \to 0d1$, but violates the second condition: for the production rule $d_i \to r_i$, r_i is a regular expression over $\Sigma \cup \{d_1, \cdots, d_{i-1}\}$; or
  - the regular definition $d_1 \to d_2 1$, $d_2 \to 0d_1$, same as above.
- Actually, this pattern is very useful when compilers are checking parenthesis.
- To prove a language is not regular, you need the pumping lemma from the theory of computation (not for this course).

Exercise

- Use English to describe the languages over the alphabet $\Sigma = \{a, b\}$ denoted by the following regular expressions:
  - $a(a|b)^*a$
  - $a^*ba^*ba^*ba^*$
  - $(a|b)^*bb(a|b)^*$
  - a^*a^*

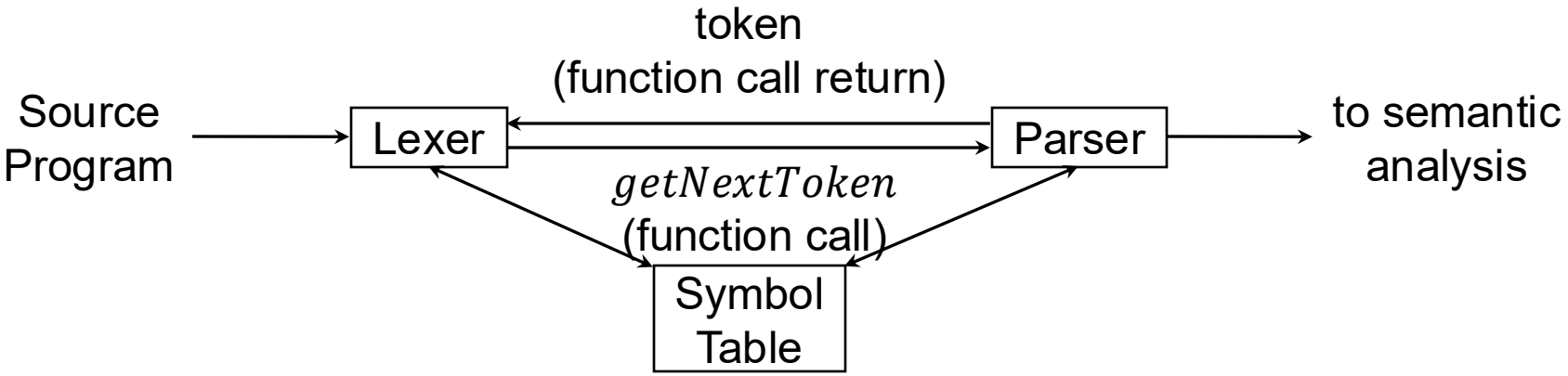
Exercise

- Write the regular definition for the language $Comments$, consisting the strings surrounded by $/*$ and $*/$ and without an intervening $*/$. For simplifications, we assume that the alphabet $\Sigma = [0-9] \cup [a-zA-Z] \cup \{-,*,/\}$.
- Write the regular definition for the language consisting the strings which has even number of a's, assuming the alphabet $\Sigma = \{a, b\}$.

- Note: when you verify the correctness of a regular expression/definition, try to find counter examples. A counter example can be
  - a string in the language but cannot be produced by the regular expression/definition; or
  - a string is produced by the regular expression/definition but not in the language.

Lexical Analysis

- The lexical analyzer (lexer) is a program to split a stream of input characters into a stream of tokens, which is the input to the parser.
- But in practice, the parser won't wait until all characters are transformed into tokens.
- When the lexer is called by the parser, it reads more characters and returns the next lexeme has been found.
- The lexer also removes the spaces and comments.
- It also put identifiers into the symbol table.



Lexical Analysis

Theoretically, the lexer can be combined with the parser as a big piece of program. But in practice we never do that for these reasons.

- Separating them makes each simpler to implement.
- The lexer can works much faster and use buffering when it works alone.
- It becomes easier to change the compiler to support special characters or foreign keyworks because these changes are only related to the lexer.

Token

- A ***token*** is a set of strings over an alphabet. Thus, a token is a (regular) language.
- Usually, a token is given a meaningful name. For example,
  - "identifier" presents all the strings that can be used as an identifier in the program.
  - "integer_literal" presents integer constants.
- Some of tokens are finite, like "if" which only contains one string; while some other tokens are infinite, like "identifier" because identifiers are user-defined.
- In this course, we often use the same name for a lexeme and the token if the token has only one lexeme. But you need to distinguish the different meanings.

Token

- A ***pattern*** is a mathematical rule that describes the set of strings for a token.
- We will use ***regular expressions*** to specify a pattern for each token.
- A ***lexeme*** is a sequence of alphabet symbols (a string) which is matched by a pattern, and thus belongs to a token.
- Thus, we can also say a token is a set of lexemes.

- Building one regular expression for each token is fine in theory, but inefficient in practice.
- Usually, we use a combined regular expression for all tokens in compilers because the regular language is closed under union.
- The union of two regular languages is a regular language.

Token

- Recall the example we used in the last lecture, the following piece of code

$$num = 1 + 23;$$

has 13 characters, 6 lexemes, and 5 different tokens.

No.	Lexeme	Token
1	num	identifier
2	=	ass_opt
3	1	int_literal
4	+	plus_opt
5	23	int_literal
6	;	semicolon

- Here the tokens `ass_opt`, `plus_opt`, and `semicolon` have only one lexeme in each; while `identifier` and `int_literal` are infinite.

Token

- Usually, if a token has multiple lexemes, the lexer returns the lexeme and its ***attribute*** (forming a pair) to the parser because semantic analysis needs to use attributes to do type checking.
- If we go back to the above example, these are the pairs returned by the lexer.

```
<identifier, pointer to symbol table entry for num>
<ass_opt, null>
<int_literal, 1>
<plus_opt, null>
<int_literal, 23>
<semicolon, null>
```

Token

- Taking the token id for example, the regular definition is

$$
\begin{aligned}
digit &\rightarrow [0-9] \\
letter &\rightarrow [aA-zZ] \\
id &\rightarrow letter(letter|digit)^*
\end{aligned}
$$

The End of Lecture 2