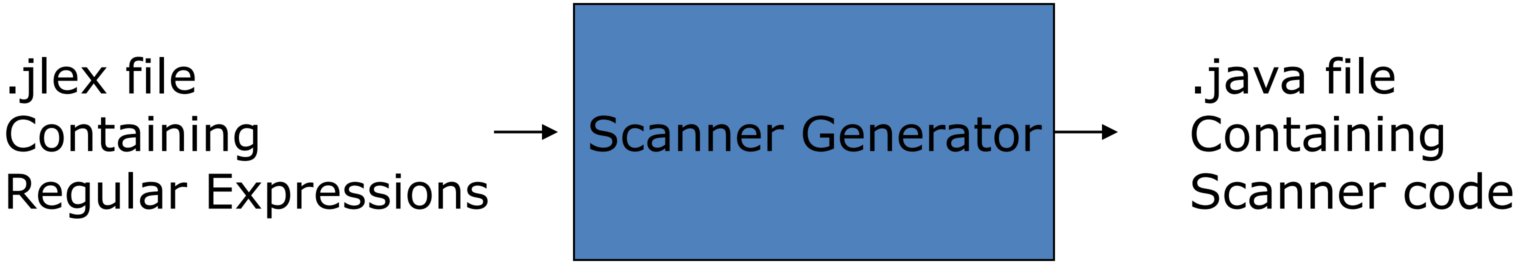
Introduction to Compiler Design

Lesson 4:

Scanners, Regular Expressions

Scanner Generator



.jlex file
Containing
Regular Expressions

Scanner Generator

.java file
Containing
Scanner code

To understand Regular Expressions
you need to understand Finite-State Automata

Terms to Know

- Alphabet (Σ) – any finite set of symbols e.g. binary, ASCII, Unicode
- String – finite sequence of symbols e.g. 010001, banana, bãër
- Language – any countable set of strings e.g.
  - Empty set
  - Well-formed C programs
  - English words

Regular Expressions

- Easy way to express a language that is accepted by FSA
- Rules:
  - ε is a regular expression
  - Any symbol in Σ is a regular expression

  If *r* and *s* are any regular expressions then so is:
  - *r*|*s* denotes union e.g. "*r* or *s*"
  - *rs* denotes *r* followed by *s (concatination)*
  - (*r*)* denotes concatination of r with itself zero or more times (Kleene closer)
  - () used for controlling order of operations

Example Regular Expressions

Regular Expression	Corresponding Language
ε	{“”}
a	{“a”}
abc	{“abc”}
a|b|c	{“a”,”b”,”c”}
(a|b|c)*	{“”,”a”,”b”,”c”,”aa”,”ab”,”ac”,”aaa”,…}
a|b|c|…|z|A|B|…|Z	Any letter
0|1|2|…|9	Any digit

Precedence in Regular Expressions

- * has highest precedence, left associative

- Concatenation has second highest precedence, left associative

- | has lowest associative, left associative

More Regular Expression Examples

<table>
<tr><th>Regular Expression</th><th>Corresponding Language</th></tr>
<tr><td>ε|a|b|ab*</td><td>{“”, “a”, “b”, “ab”, “abb”, “abbb”,…}</td></tr>
<tr><td>ab*c</td><td>{“ac”, “abc”, “abbc”,…}</td></tr>
<tr><td>ab*|a*</td><td>{“”, “a”, “ab”, “aa”, “aaa”, “abb”,…}</td></tr>
<tr><td>a(b*|a*)</td><td>{“a”, “ab”, “aa”, “abb”, “aaa”, …}</td></tr>
<tr><td>a(b|a)*</td><td>{“a”, “ab”, “aa”, “aaa”, “aab”, “aba”,…}</td></tr>
</table>

Examples

- What is the language described by each Regular Expression?

a*

(a|b)*

a|a*b

(a|b)(a|b)

aa|ab|ba|bb

(+|-|ε)(0|1|2|3|4|5|6|7|8|9)*

Regular Definition

If Σ is an alphabet of basic symbols, then a regular definition is a sequence of definitions of the form:

$D_1 \rightarrow R_1$

$D_2 \rightarrow R_2$

…

$D_n \rightarrow R_n$

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

Regular Definitions Example

Example C identifiers:

Σ = ASCII

*letter_* → a|b|c|…|z|A|B|C|…|Z|_

digit → 0|1|2|…|9

id → *letter_* (*letter_* | *digit*)*

Regular Definitions Example

Example Unsigned Numbers (integer or float):

Σ = ASCII

digit → 0|1|2|…|9

digits → *digit digit**

optionalFraction → . *digits* | ε

optionalExponent → (E(+|-| ε)digits)| ε

number → *digits optionalFraction optionalExponent*

Special Characters in Reg. Exp.

What does each of the following mean?

* – Kleene Closure

| – or

() – grouping

[] – creates a character class

+ – Positive Closure

? – zero or one instance

“” – anything in quotes means itself, e.g. “*”

. – matches any single character (except newline)

\ – used for escape characters (newline, tab, etc.)

^ – matches beginning of a line

$ – matches the end of a line

Extensions to Regular Expressions

- + means one or more occurrence (positive closure)
- ? means zero or one occurrence
- Character classes
  - a|r|t can be written [art]
  - a|b|…|z can be written [a-z]

    As long as there is a clear ordering to characters
  - [^a-z] matches any character except a-z

Example Using Character Classes

^[^aeiou]*$

Matches any complete line that does not contain a lowercase vowel

What if we remove the first ^ and the $?

Examples

- Create Character Classes for:
  - First ten letters (up to “j”)
  - Lowercase consonants
  - Digits in hexadecimal
- Create Regular Expressions for:
  - Case Insensitive keyword such as SELECT (or Select or SeLeCt) in SQL
  - Java string constants
  - Any string of whitespace characters

Creating a Scanner

- Create a set of regular expressions, one for each token to be recognized
- Convert regular expressions into one combined DFA
- Run DFA over input character stream
  - Longest matching regular expression is selected
  - If a tie then use first matching regular expression
- Attach code to run when a regular expression matches