



Lexical Analysis

Recognizing tokens

Language to describe various token types

+ ; id, 12, +

Regular expression pattern matching

Each token type is specified by a pattern (regular expr)

A string (of text) matches a regular expr, or it doesn't.

Lex analyzer reads the input until (EOS or) some prefix of the input matches one of the regular expressions

Greedy - looks for longest possible prefix of the input that matches a reg exp.

In C: `c = a+++b;`
 Annotations:
 - 'a' is an identifier (id)
 - '=' is an assignment operator (assign)
 - '+' is an addition operator (add)
 - 'b' is an identifier (id)
 - ';' is a semicolon (inc)
 - The greedy match for '+' is '+++', which is not a valid token, leading to a 'Bad' parse.

`c = a+++b;`

Regular exprs for tokenizing

Primitives

$\emptyset$ matches nothing
 ϵ matches the empty string and nothing else
 a 'a' is an ASCII char (except for certain special meta-chars)

primitive operators

$\rightarrow$ union, OR, $|$

r, s are regexprs, then $r|s$ is a regexp matching anything that r matches or that s matches

$\rightarrow$ concatenation (juxtaposition)

rs matches any string with some prefix matching r and the rest of the string matching s .

$\rightarrow$ * (Kleene closure)

r^* match any string that is the concatenation of any number (zero or more) strings matching r .

$r^* = \epsilon \cup r \cup rr \cup rrr \cup \dots$
 $= \epsilon | r | rr | rrr | \dots$

$0|1|2|3|4|5|6|7|8|9$
 $(0|1|2|\dots|9)^*$

int const:

$(0|1|2|\dots|9)(0|1|\dots|9)^*$

"(" vertigule: |

* highest precedence (unary, postfix)

concatenation - lower precedence (binary, associative)

$rst = (rs)t = r(st)$

union, | - lowest precedence (binary infix, assoc, cannot use parentheses for controlling the grouping.)

Non-primitives:

char set:

$[a|g|z] := a|d|g|z$

$[*]$ metachars usually treated literally.

$[()]$

$[0-9] = [0|2|3|4|5|6|7|8|9]$
 $= 0|1|2|\dots|9$

int const:

$[0-9][0-9]^*$

$[A-Z] [A-Z]$

$[a-z] [A-Za-z]$

$[0|1|2|3|4|5]$

$[0|2|3|4|5]$

$[+]$ $[[]]$

$[\backslash]$ $[\backslash n]$ $[\backslash \backslash n]$

↑ matches only
↑ matches only
↑ matches only

$[\backslash]$

complement set

$[\wedge abc]$ matches any single char other than a,b,c

$[abc^+]$

$r+ = rr^*$

one or more matches of r.

$[0-9]^+$

$[+-]^?$ — zero or one occurrences of (optional)

$r? = r|e$

$[+-]^?[0-9]^+$

optionally signed int const

" $xx[\backslash]$ "

"main" = main

flex input file

preamble
%%
rules (sequence of rules)
%%
util:iter

rule:

regex action
 C code

example

$[0-9]^+$ { return INT_CONST; }

$[A-Za-z][A-Za-z0-9]^*$
 { return ID; }