Regular Expressions & Languages

Regular expressions are a formalism for describing patterns in strings (used in tools like Emacs, Perl,...)

Language of a regular expression is the set of strings described

R1 \cup R2 \quad (R1, R2 reg. exp)
L(R1) \cup L(R2)

R1 \circ R2 \quad L(R1) \circ L(R2)

(R1)^* \quad (L(R1))^*

What kind of definition is this?

Regular Expressions, continued

(a \cup b) \quad (a)^* \quad (a \cup b)^* \quad a \cup b \quad * \quad \circ \quad \cup

Precedence of operations:

Associate to left (same op):

(a U b)^* U c \quad \text{is (a U b^*) U c}
but ( ) must always be followed if included!

Shorthand: \Sigma for alphabet

Omit \circ

More operations:

R1 \mid R2 \quad L(R1) \ U L(R2)

(R1)^* \quad L(R1)^* - \{ \varepsilon \}

(R1) ? \quad L(R1) \ U \{ \varepsilon \}

Ex. (static) (/b)^* (void) ? (/b) (MAIN)
More Regular Expressions

Ex. 1. Write a regular expression to describe IP addresses:
    a sequence of four 3-digit numbers, separated by .’s.

Ex. 2. Write a regular expression to describe dollar amounts,
    with optional cents.

Ex. 2. Write a regular expression to represent July 4,
    with Jul or July for the month,
    4, 4th, or fourth for the date.

Ex. 4. Change ex. 1 so each number is in range 0-255.

Lexical Analysis (Scanning) in Compilers

First phase of the compiler: strings → tokens
Programmning-language dependent

Tokens can include
    keywords if then end
    operators < > =
    identifiers
    literals decimal integer, character

One regular expression to describe each of the token types

Qu: How recognize tokens?
What is the Regular Expression?

A Java identifier must start with a letter, underscore, or $, subsequent characters can also contain digits (Letters are A-Z, a-z, digits are 0-9)

A Java decimal integer literal consists of a sequence of digits, without a leading zero, and with an optional L or l (indicating it is a long integer)
   It must have at least one digit.

Lex and Flex

Automatic tools for creating a lexical analyzer
Based on regular expressions

Lex source
  lex.l
  reg. ex. + actions

lex.yy.c

input stream

Lex Compiler

C code
  lex.yy.c
  tabular rep. of reg. ex.
  + table-driven routines

C Compiler

Lexical Analyzer
  executable

a.out

sequence of tokens

Lex. Anal.
  a.out
Lex and Flex, Continued

Lex language

Definitions

- Regular Definitions
  - Ex. DIGIT [0-9]
  - ID [a-zA-Z][a-z 0-9]*

Rules

- Pattern {Action}
  - Extended Reg. Exp. C code
  - 0 or more blanks [\s]* putchar(' ') ;
  - 0 or 1 a [a-?]
  - not A-Z [\^A-Z]

User Code C code

Regular expressions and languages

Th.: A language is regular iff some regular expression describes it

Lemma: If language A is described by some reg. exp., then there is a NFA that recognizes A

Proof:

- Reg. Exp. NFA
  - a
  - ε
  - ϕ
  - R1 ∪ R2
  - R1 ∩ R2
  - R1 *

Ex. {+, ε, -} ( {0-9} ) *
DFA to Regular Expression

Lemma: If language L is accepted by a DFA, then there is a reg. exp. that describes it.

Proof Idea: DFA $\rightarrow$ NFA with reg exp labels $\rightarrow$ reg. exp.

Example: 1.35 p. 75

![Diagram of DFA to NFA]

Perl's Use of Regular Expressions

Perl is a scripting language with operations to process strings described via its regular expressions.

```
m/regex/ matches pattern described by regex
         returns true or false
```

Ex. if ($reply =~ m/^\d+$/) {print "only digits \n";}
    else {print "not only digits \n"};

```
s/re1/re2  substitution, search for re1, and on match, replace with re2
```

Ex. $yourname = 'Tom Cruise';
    $mynname = 'Jeanne Ferrante';
    $var = 'Congratulations, Tom Cruise!';
    $var =~ s/$yourname/$mynname/;
    print "$var \n";

### Perl’s Regular Expressions

#### Literals

Most alphanumerics and strings
e.g. A a 1 2 _

Escape characters
e.g. \t (tab) \n (newline) \r (return) ...

#### Character classes

Used to specify collection

- **[A-Z]** Any uppercase letter
- **[^A-Z]** Any character but uppercase letter
- **\w** Any alphanumeric, _
- **\s** Any whitespace char.
- **\d** Any digit

#### Metacharacters

- **•** Any character
- **\** Ignore "meta"
- **$** End of line
- ^ Start of line
- () Grouping

**Concatenation**

#### Quantifiers

- **(Standard)**
- **{n,m}** at least $n, \leq m$

### Perl’s Regular Expressions: Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td></td>
</tr>
<tr>
<td>[hello]</td>
<td></td>
</tr>
<tr>
<td>[^hello]</td>
<td></td>
</tr>
<tr>
<td>^[hello]</td>
<td></td>
</tr>
<tr>
<td>hello+</td>
<td></td>
</tr>
<tr>
<td>(hello)*</td>
<td></td>
</tr>
<tr>
<td>^abc</td>
<td></td>
</tr>
<tr>
<td>^abc$</td>
<td></td>
</tr>
<tr>
<td>^(abc)$</td>
<td></td>
</tr>
<tr>
<td>he...o</td>
<td></td>
</tr>
<tr>
<td>he..o</td>
<td></td>
</tr>
<tr>
<td>^the{1,10}\s</td>
<td></td>
</tr>
</tbody>
</table>
**Perl: Going beyond Regular Expressions**

In Perl, can assign values to variables

\$_   the default scalar variable, can use other variables

Ex.  \$sal = 51000;  
     print "$sal\n";  51000  
     print "No\$name\n";  No

Parentheses capture string values in reg. exp.’s
special var’s numbered via open paren., from left
Within single regex:  ([A-Z a-z]+) \s\s
string within ( ) will be assigned to \1; match doublewords
Between regex:

if ( $temp =~ m/(\-?[0-9]+) ([C F])\)  
1st ( sets \$1; 2nd ( sets \$2 
{$num = \$1;  $type = \$2;  if ($type eq "C") 
{$fahr = (\$temp * 9 / 5) + 32}  else ....