Languages with string pattern-matching capabilities

SNOBOL 1962
AWK 1977
PERL 1987

- Built-in primitives for pattern-matching
- mainly used for scripting
Strings

- finite sequence of symbols/tokens
- over an alphabet $\Sigma$

The empty string is denoted $\epsilon$

*Concatenation* of $x$ and $y$ is written $xy$

A *language* is a set of strings
Operations on languages

usual set operations

concatenation of languages

\[ LM = \{ xy \mid x \in L, y \in M \} \]

Kleene closure

\[ L^* = \{ x_1 \ldots x_k \mid k \geq 0, x_1, \ldots, x_k \in L \} \]

\[ \epsilon \in L^* \ (k = 0) \]
Kleene closure

$L^*$

- $\epsilon \in L^*$
- if $x \in L$ then $x \in L^*$
- if $x \in L^*$ and $y \in L^*$ then $xy \in L^*$
- nothing else is in $L^*$

$L^+ = \{ x_1 \ldots x_k \mid k \geq 1, x_1, \ldots, x_k \in L \}$

- $\epsilon$ need not be in $L^+$

($\epsilon$ is in $L^+$ iff $\epsilon$ is in $L$)
Regular expressions

to specify languages/patterns over an alphabet $\Sigma$

**Building blocks:**

- $\emptyset$, denotes the empty language
- $\epsilon$, denotes $\{\epsilon\}$
- every symbol $a \in \Sigma$
- operations concatenation, $\cup$ and $*$
operator precedence

* ⪰ concatenation ⪰ ∪

(* takes precedence over concatenation, which in turn takes precedence over ∪)

Thus

\[ ab^* \neq (ab)^* \]
\[ a \cup b^* \neq (a \cup b)^* \]
\[ a \cup ba \neq (a \cup b)a \]
Regular expressions

Examples:

$(0 \cup 1)^*0$  binary strings ending in 0

$(0 \cup 1)^*00(0 \cup 1)^*$  binary strings that contain two consecutive 0’s

$0^*01^*1$  strings of the form one or more 0’s followed by one or more 1’s
regexp matching

- if $r$ is a regexp, then $(r)$ matches the same strings as $r$.

- if $r_1$ and $r_2$ are regexps, then $r_1 \cup r_2$ matches any string matched by either $r_1$ or $r_2$.

- if $r_1$ and $r_2$ are regexps, then $r_1r_2$ matches any string of the form $xy$ where $r_1$ matches $x$ and $r_2$ matches $y$.

- if $r$ is a regexp, then $r^*$ matches any string of the form $x_1 \ldots x_n$, $n \geq 0$, where $r$ matches all $x_i$s ($1 \leq i \leq n$).
### In Egrep and Perl

<table>
<thead>
<tr>
<th>Classical notation</th>
<th>egrep/perl notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \cup y$</td>
<td>$x \mid y$</td>
</tr>
<tr>
<td>$x^*$</td>
<td>$x^*$</td>
</tr>
</tbody>
</table>

looks for *substrings* to match

```bash
eve% echo "aabb" | egrep 'ab'
aabb
eve% echo "aabb" | egrep 'ba'
eve%
```
Anchors in egrep and perl

<table>
<thead>
<tr>
<th>character</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>position at the start of the line</td>
</tr>
<tr>
<td>$</td>
<td>position at the end of the line</td>
</tr>
</tbody>
</table>

```
eve% echo "aabb" | egrep 'ˆab'
eaabb

eve% echo "aabb" | egrep 'ˆa*b'
aabb

eve% echo "aabb" | egrep 'ˆa*b$'
aabb

eve% echo "aabb" | egrep 'ˆa* b*$'
eve%
```
Extended regular expressions

Backreferencing

Number left parentheses, starting with 1, from the left.

Give the same numbers to the corresponding (matching) right parentheses.

\[(\ldots (\ldots)\ldots (\ldots)\ldots)\]

\[1\quad 2\quad 2\quad 3\quad 3\quad 1\]

\[n\] stands for the string matched by the regular expression between the \(n^{th}\) left parenthesis and the corresponding right parenthesis.
extended regexp matching

- if \( r \) matches a string \( x \), then \( (r) \) matches \( x \) \( i \ i \) and the value \( x \) is assigned to \( \backslash i \).

- \( \backslash j \) matches the string that has been assigned to it.

- if \( r_1 \) and \( r_2 \) are regexps, then \( r_1 \cup r_2 \) matches any string matched by either \( r_1 \) or \( r_2 \).

- if \( r_1 \) and \( r_2 \) are regexps, then \( r_1 r_2 \) matches any string of the form \( xy \) where \( r_1 \) matches \( x \) and \( r_2 \) matches \( y \).

- if \( r \) is a regexp, then \( r^* \) matches any string of the form \( x_1 \ldots x_n \), \( n \geq 0 \), where \( r \) matches all \( x_i \)'s \( (1 \leq i \leq n) \).
From the `egrep` man page:

The backreference \n, where \n is a single digit, matches the substring previously matched by the \n\textsuperscript{th} parenthesized subexpression of the regular expression.

```
% echo "ababb" | egrep 'ˆ(ab*)\1$'
ababb
% echo "ababb" | egrep 'ˆ(ab*)(ab*)$'
ababb
% echo "ababa" | egrep 'ˆ(ab)*\1'
ababa
% echo "ababa" | egrep 'ˆ(ab)*\1$'
```
More examples

% echo "ababa" | egrep 'ˆ(\(a|b\)*\1\2$'
ababa

% echo "ababaa" | egrep 'ˆ((a|b)*\1\2$'
ababaa

% echo "abaaba" | egrep 'ˆ((a|b)*\1\2$'
abaaba

% echo "abaabaa" | egrep 'ˆ((a|b)*\1\2$'
abaabaa

% echo "abaabab" | egrep 'ˆ((a|b)*\1\2$'
abaabab

% echo "abaabaa" | egrep 'ˆ((a|b)*\1\2$'
abaabaa
From the Perl documentation:

Within a pattern, you may designate subpatterns for later reference by enclosing them in parentheses, and you may refer back to the $n^{th}$ subpattern later in the pattern using the metacharacter `\n`. Subpatterns are numbered based on the left to right order of their opening parenthesis. A backreference matches whatever actually matched the subpattern in the string being examined, not the rules for that subpattern. Therefore, `(0|0x)\d*\s1\d*` will match “0x1234 0x4321”, but not “0x1234 01234”, because subpattern 1 actually matched “0x”, even though the rule `0|0x` could potentially match the leading 0 in the second number.
Perl matching operator \(=^\sim\) 

Regular expressions are contained in slashes.

Sample program:

```perl
#!/usr/local/bin/perl
.pattern = $ARGV[0];
$string = $ARGV[1];
if ($string =~ /$pattern/)
    {print $string, "\n";};
```
Sample Perl program

#!/usr/local/bin/perl
$pattern = $ARGV[0];
$string = $ARGV[1];
if ($string =~ /$pattern/)
    {print $string, "\n";};
#!/usr/local/bin/perl

$pattern = $ARGV[0];
$string = $ARGV[1];

if ($string =~ /$pattern/) {
    print $string, "\n";
}