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:  $* \succ$ concatenation $\succ \cup$

( $*$ takes precedence over concatenation, which in turn takes precedence over $\cup$)
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

1. if \( r \) is a regexp, then \((r)\) matches the same strings as \( r \).

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

3. 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 \( y \).

4. 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

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 '^

eve% echo "aabb" | egrep '^
aabb

eve% echo "aabb" | egrep '^
aabb

eve% echo "aabb" | egrep '^
eve%

CSI 311
Extended regular expressions

Backreferencing

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

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

\[
\left( \cdots (\cdots) \cdots (\cdots) \cdots \right)
\]

\(1 \ 2 \ 2 \ 3 \ 3 \ 1\)

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

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

2. $\backslash j$ matches the string that has been assigned to it.

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

4. 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 $y$.

5. 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 \( n^{th} \) parenthesized subexpression of the regular expression.

```
% echo "ababb" | egrep '^(ab*)\1'$
% echo "ababb" | egrep '^(ab*)\1(ab*)$'
ababb
% echo "ababa" | egrep '^(ab)*\1'  
ababa
% echo "ababa" | egrep '^(ab)*\1$'
% echo "ababa" | egrep '^(ab)*\1(a*)$'
ababa
% echo "ababa" | egrep '^(ab)*\1(a*)\2$'
% echo "ababaa" | egrep '^(ab)*\1(a*)\2$'
ababaa
% echo "ababaa" | egrep '^(ab)*a\1$'
% echo "abaaba" | egrep '^(ab)*a\1$'
abaaba
```