Applicative regular expressions

Roman Cheplyaka

April 20, 2012
Dutch HUG Day
Two uses of regular expressions:

- Recognition
  - grep
  - search in text editors
  - lexer generators

- Parsing
Capture groups

Example: a regex for URLs

(http|ftp)://([^./]+\.)+([^./]+)\/(^[^/]+)+/?
A better approach: parsing combinators

<table>
<thead>
<tr>
<th>Regular languages</th>
<th>Applicative functors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language concatenation</td>
<td>&lt;*&gt;</td>
</tr>
<tr>
<td>Language union</td>
<td>&lt;</td>
</tr>
<tr>
<td>Kleene star</td>
<td>many</td>
</tr>
</tbody>
</table>
Example

data Protocol = HTTP | FTP
data URL = URL Protocol [String] [String]

url :: RE Char URL
url = URL <$> protocol <*> string "://" <*> host <*> path

protocol :: RE Char Protocol
host, path :: RE Char [String]
protocol = HTTP <$> string "http"
  <|> FTP <$> string "ftp"
host = (:) <$> s <*> some (sym '.' *> s)
  where s = some $ psym $ not . ('elem' ['.'','/'])
path = some (sym '/' *> s)
  where s = some $ psym $ not . (== '/')
Comparison with monadic parsing libraries:

- better complexity
- better memory usage
- incremental parsing
- longest match
Perl vs POSIX semantics

- Perl for inner matches
- POSIX or Perl for the whole match
Challenges

Type safety vs speed
Current state

- Available on hackage and github
- Feature-full
- Performance needs some improvement
Dominate the world
... of regex libraries for Haskell