A Play on Regular Expressions

Sebastian Fischer, Frank Huch, and Thomas Wilke

Christian-Albrechts University of Kiel

ICFP 2010
A Play on Regular Expressions

in 3 acts!

Sebastian Fischer, Frank Huch, and Thomas Wilke

Christian-Albrechts University of Kiel

ICFP 2010
• intuitive method for regular expression matching
• automata construction with elegant Haskell implementation
• can be generalized in surprising ways
((a|b)*c(a|b)*c) *(a|b)*
symbols

\(( (a|b)*c (a|b)*c )^* (a|b)^* \)
alternatives

\[(\textcolor{red}{(a|b)*c(a|b)*c})*(a|b)*\]
sequences

\((a|b)*c(a|b)*c)*(a|b)*\n
((a|b)*c (a|b)*c) *(a|b) *

repetitions
((a|b)*c(a|b)*c) *(a|b) *
"abc"
((a|b)*c(a|b)*c)*(a|b)*
"abc"
preceded by c
((a|b)*c(a|b)*c) * (a|b) *
"abc"
not at "the end": c still to come

\[( (a|b)*c(a|b)*c) * (a|b) * \]

"abc"
\(( (a|b)^*c(a|b)^*c)^* (a|b)^* \) at "the end"
accepts empty word
"abcc"
data Reg = Eps
  | Sym Bool Char
  | Alt Reg Reg
  | Seq Reg Reg
  | Rep Reg

empty
alternative
sequence
repeterition
mark
symbol
does regexp match word ?

match :: Reg -> String -> Bool
match r "" = empty r
...

predicate : accepts empty word ?
empty :: Reg -> Bool
empty Eps = True
empty (Sym _ _) = False
empty (Alt p q) = empty p || empty q
empty (Seq p q) = empty p && empty q
empty (Rep r) = True
... 
match r (c:cs) =
  final $ foldl (shift False)
  (shift True r c)
  cs

predicate: mark at "the end"? 

shifts marks
final :: Reg -> Bool

final Eps = False
final (Sym m _) = m
final (Alt p q) = final p || final q
final (Seq p q) = final q || final p && empty q
final (Rep r) = final r
match r (c:cs) =
  final $ foldl (shift False)
  (shift True r c)
  cs
shift :: Bool -> Reg -> Char -> Reg

preceding mark

current symbol
shift _ Eps       _ = Eps
shift m (Sym _ x) c = Sym (m && x==c)

...
\[
\begin{align*}
\text{shift } m \ (\text{Alt } p \ q) \ c &= \\
\text{Alt } (\text{shift } m \ p \ c) \ (\text{shift } m \ q \ c)
\end{align*}
\]
\[
\text{shift } m \ (\text{Seq } p \ q) \ c = \\
\text{Seq } (\text{shift } m \ p \ c) \\
(\text{shift } (m \land \text{empty } p \lor \text{final } p) \ q \ c)
\]
\[
\begin{align*}
\text{shift } m \ (\text{Rep } r) \ c &= \\
\text{Rep } (\text{shift } (m \ || \ \text{final } r) \ r \ c)
\end{align*}
\]
replace:

- False $\mapsto 0$
- True $\mapsto 1$
- (||) $\mapsto (+)$
- (&&) $\mapsto (*)$

match :: Reg $\rightarrow$ String $\rightarrow$ Int

number of matchings
match \( (a|a*) \) "a" == 2
match \(((a|a*)(b|b*))\) "ab" == 4 \(=\) 2*2

ambiguous regexps

addition

multiplication
match :: Semiring s

=> Reg -> String -> s

• position of leftmost matching
• length of longest matching
• ...

results of match depend on specific semiring
Laziness $\leadsto$ infinite regular expressions!

non-regular languages like:

\[
\{a^n b^n | n \in \mathbb{N}\}
\]

\[
\{a^n b^n c^n | n \in \mathbb{N}\}
\]

and more.
- intuitive method for regular expression matching
- automata construction with elegant Haskell implementation
- can be generalized in surprising ways
curious? read the play!

cabal install weighted-regexp

github.com/sebfisch/haskell-regexp

Thanks!