A Play on Regular Expressions

Sebastian Fischer, Frank Huch, and Thomas Wilke

Christian-Albrechts University of Kiel

ICFP 2010

A Play on Regular Expressions

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 suprising ways

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
"abc"
```

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
"abc"
```

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
"abc"
```

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
"abc"
```

```
((a|b)*c(a|b)*c)*(a|b)*
```

```
"abcc"
```

```
data Reg = Eps
         | Sym Bool Char
         | Alt Reg Reg
         | Seq Reg Reg
         | Rep Reg
```

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

```
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
```

```
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
```

```
shift _ Eps         _ = Eps
shift m (Sym _ x) c = Sym (m && x==c)
...
```

```
...
shift m (Alt p q) c =
  Alt (shift m p c) (shift m q c)
...
```

```
...
shift m (Seq p q) c =
  Seq (shift m p c)
      (shift (m && empty p || final p)
             q c)
...
```

```
...
shift m (Rep r) c =
  Rep (shift (m || final r) r c)
```

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

```
match :: Reg -> String -> Int
```

```
match (a|a*)          "a"  == 2
match ((a|a*)(b|b*)) "ab" == 4
```

```
match :: Semiring s
      => Reg -> String -> s
```

- position of leftmost matching
- length of longest matching
- ...

Laziness $\rightsquigarrow$ 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 suprising ways

curious? read the play!

```
cabal install weighted-regexp
```

```
github.com/sebfisch/haskell-regexp
```