# Lec 14 RegEx

## # Reg Ex

* Chomsky ... ? language hierarchy

Ex.


What we can get:

- c
- cc
- ccc
- $c^*$
- cgc
- ...

Operations: e.g. $(g + w)^* c$ (or)

## # Notation & Definition

$\Sigma$ – alphabet of characters

$\Sigma^*$ – all finite string over $\Sigma$ $\quad$ $aabba \in \{a, b\}^*$

$\varepsilon$ – empty string

<u>Language</u> – subset of $\Sigma^*$

<u>Regular expression</u>:

| | | |
|---|---|---|
| $a$ | for all $a \in \Sigma$ | |
| $0$ | special symbols | |
| $1$ | | |
| $r_1 + r_2$ | or | with $r$, $r_1$, $r_2$ being RegEx |
| $r_1 r_2$ | concat | |
| $r^*$ | repeat | |

# Regular Language ↔ Finite automata, tokenisation

SML is a context-free language
Turing machine is regular plus two stacks

<u>Regular language</u> denoted $L(r)$

$L(a) = \{a\}$ ← singleton
$L(0) = \{\ \}$ ← empty language
$L(1) = \{\varepsilon\}$ ← language with empty string only $L(1) = \{""\}$
$L(r_1 + r_2) = \{s \mid s \in L(r_1) \vee s \in L(r_2)\}$
$L(r_1 r_2) = \{s \mid s \in L(r_1) \wedge s \in L(r_2)\}$
$L(r^*) = \{s \mid s = s_1 s_2 \cdots s_n,\ n \geq 0,\ \text{each } s_i \in L(r)\}$ note $\varepsilon \in L(s^*)$

* $L$ is regular if $L = L(r)$ f.s. regex $r$.

Ex. suppose $\Sigma = \{a, b\}$

- $L(aa) = \{aa\}$
- $L((a+b)^*) = \Sigma^*$
- $L((a+b)^* aa (a+b)^*)$ = all strs in $\Sigma^*$ that has "aa" in it
- $L((a+1)(b+ba)^*)$ = all strs in $\Sigma^*$ without "aa" in it
- $\underbrace{L(r^*) = L(1 + rr^*)}$

Notice multiple can mean same thing

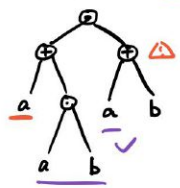
# Acceptor

```
(* accept : regexp -> string -> bool
   REQ  true
   ENS  accept r s ↪ true if s ∈ L(r)
                   ↪ false otherwise
*)
```

Consider : $r = (a+ab)(a+b) \rightarrow L(r) = \{aa, ab, aba, abb\}$

ab / a      a / ba   notice we may not know where to split. Need backtracking → Continuation !!

There's actually backtracing going on. see aside trace



```
(* match : regexp -> char list -> (char list -> bool) -> bool
   REQ  k total
   ENS  match r cs k  ↳ true if  cs ≅ p@s  s.t.  p ∈ L(r) & k(s) ≅ true
                      ↳ false  otherwise
*)
```

## Code

turn into char list        check if empty

```
fun accept r s = match r (String.explode s) List.null

datatype regexp = Char of char
                | Zero
                | One
                | Plus of regexp * regexp
                | Times   regexp * regexp
                | Star of regexp

fun match (Char a) cs k =
    (case cs of
      [] => false |          <- can't split out p
      c::cs' => (a=c) andalso (k cs')
    )
  | match Zero _ _ = false   <- nothing in that language
```

```
| match One cs k = k cs
| match (Plus (r1, r2)) cs k =
    (match r1 cs k) orelse (match r2 cs k)
| match (Times (r1, r2)) cs k =
    match r1 cs (fn cs' => match r2 cs' k)
| match (Star (r)) cs k =        <- has bug
    (k cs) orelse (match r cs (fn cs' => match (Star r) cs' k))
```

Recall: $L(r^*) = L(1 + rr^*)$

# Proof-directed Debugging

We can't prove (fn cs' => match (Star r) cs' k) is total

Try : match (Star One) [#"a"] List.null , that loops!

Solution ?

1. Require input regex in standard form

Standard form $\Leftrightarrow$ for any subexpression Star(r'), $\varepsilon \notin r'$ ← looping to look for empty is bad :c

We can actually write preprocessor to get any r to this form

2. Runtime check that cs' is shorter than cs , viz. cs' is proper prefix

```
| match (Star (r)) cs k =
    (k cs) orelse (match r cs (fn cs' =>
                                 properSuffix (cs', cs)
                                 andalso
                                 match (Star r) cs' k)
                   )
```

# Aside : example trace

Let r = (a+ab)(a+b)

accept r "aba" (actually array)

⇒ match r "aba" List.null

⇒ match (a+ab) "aba" $\underbrace{\text{(fn cs' ⇒ match (a+b) cs' List.null)}}_{k_1}$

⇒ (match a "aba" $k_1$) orelse (match ab "aba" $k_1$)

⇒ $k_1$ "ba"

⇒ match (a+b) "ba" List.null

... ⇒ false

≅ (match ab "aba" k )

... ⇒ true

# Proof of Correctness

Assume termination

WTS $\varphi(r) \equiv$ match $r$ cs $k \cong$ true iff $\exists p,s$ , $cs \cong p@s$ , $p \in L(r)$ , $k(s) \cong$ true

$\Rightarrow$ completeness

$\Leftarrow$ soundness

Go structural induction on $r$

BC1: Zero

BC2: One [omit]

BC3: Char (a)

IC1 : $r =$ Plus $(r_1, r_2)$

IH: $\varphi(r_1)$, $\varphi(r_2)$

soundness

Suppose match (Plus$(r_1, r_2)$) cs k $\cong$ true

LHS $\cong$ (match $r_1$ cs k) orelse (match $r_2$ cs k)

By IH, $\exists$ p, s, $p@s \in L(r_1)$ , k s true

Then $p \in L($Plus$(r_1, r_2))$ as req

completeness

Suppose $\exists p,s$, $cs \cong p@s$ , $p \in L($Plus$(r_1, r_2))$ , $k(s) \cong$ true

[omit]