CSE322 Theory of Computation (L9)

ⓐ whether a DFA has infinitely many strings.

def Algo(D( ), #states, #symbols):

Today

Regular expressions (Regex)

0 1 2 3 4

0 + 2*3 4 * 5 + 6 7

Definition by induction. Fix some alphabet A. $A = \{0,1\}$

R is a regular expression (regex) if R is ...

* symbol a, L(R) = {a} $\boxed{0}$ $\boxed{1}$ $L(0) = \{0\}$ $L(1) = \{1\}$
* e, L(R) = {e} ε $L(\varepsilon) = \{\varepsilon\}$
* empty set, L(R) = {} $\emptyset$ $L(\emptyset) = \emptyset$
* (R1 U R2) for regular R1 and R2; L(R) = $L(R1) \cup L(R2)$ $R1 + R2$ "$(R1 \cup R2)$"
* (R1 o R2) for regular R1 and R2; L(R) = $L(R1) \cdot L(R2)$ $(R1 \cdot R2)$
* (R1*) for regular R1; L(R) = $(R1^*)$ $L((R1)^*) = (L(R))^*$

regular op.

brackets are often dropped if they are clear

$0|1$ $L(0 \cup 1)$ $1 \cup 0$ $L(0+1)$

string s matches regex R if s is in L(R)

$A = \{a,b,c\}$ $b \cup {*}\varepsilon$ X $((b^*) \cup (\varepsilon^*))$ $\left.\begin{array}{l} b^* \\ \varepsilon^* \end{array}\right\}$ $b^* \cup \varepsilon^*$ $L(0 \cup 1) = L(0) \cup L(1)$

$0 \cup 1$ & $1\,0 \cup$ accept the same language. They are equivalent.

Exercise: Are these regular expressions?

$= \Sigma^*$

$R = (0 \cup 1)^*$ $= (L(0) \cup L(1))^*$ $= \{0\} \cup \{1\}$

What is L(R)?

$L(0) \cup L(1) = \{0,1\}$

$=$ all binary strings

$= (0(0 \cup 1)^*) \cup (((0 \cup 1)^* 1)$

$R_2 = (0\,\Sigma^*) \cup (\Sigma^* 1)$ $= 0(0 \cup 1)^*$ → start with 0; $(0 \cup 1)^* 1$ → end with 1

What is $L(R_2)$?

$=$ start with 0 or end with 1

Notation:

$0 \cup 1 = \Sigma / A$

- $(R1+)$ is regular for regular $(R1 \cdot R1^*)$ $(0+)(1+) = (0 \cdot 0^*)(1 \cdot 1^*)$ $=$ one or more 0's followed by one or more 1's
- $(R1?)$ is regular for regular $(R1 \cup \varepsilon)$ $L(0?) = \{0, \varepsilon\}$
- $(R1^k)$ is regular for regular R1 and fixed integer k $\underbrace{R1 \cdot R1 \cdot \cdots R1}_{k}$
- $(R1 \mid R2)$ is notation for $(R1 \cup R2)$ $L((abc)?) = \{\varepsilon, abc\}$
- $([\wedge a])$ is notation for $(a1 \cup a2 \cup \ldots)$ except a

$\{0,1,2,3\}$ $[\wedge 1] = 0 \cup 2 \cup 3$

Omit parentheses, if possible.

Evaluation order: star/plus, concatenation, union

$R1^*R2 \cup R3 = ((R1^*)R2) \cup R3$

Exercise: Languages of Regex

$R = 0^*10^*$ = strings containing only one 1

$R \cup \{\} \equiv R$

$L(R \cup e) = R? = L(R) \cup \{\varepsilon\}$

$R \circ e \equiv R$

$(0 \cup \varepsilon)1^*$ = strings in which 0 doesn't appear anywhere except maybe at the beginning

$R \circ \{\} \equiv \{\}$

$(\Sigma\Sigma)^*$

$= ((0 \cup 1)(0 \cup 1))^*$

= all even length strings

Regex identities

$(R1 \cup R2) \cup R3 \equiv R1 \cup (R2 \cup R3)$

$R1 . (R2 . R3) \equiv (R1 . R2) . R3$

$R1 (R2 \cup R3) \equiv R1 R2 \cup R1 R3$

$(R1 \cup R2) R3 \equiv R1 R3 \cup R2 R3$

$R1 \cup R2 \equiv R2 \cup R1$

$(R^*)^* \equiv R^*$

$L(LHS) = L(RHS)$

$L(LHS)$

$= L(R1 \cup R2) \cup L(R3)$

$= (L(R1) \cup L(R2)) \cup L(R3)$

$= L(R1) \cup L(R2) \cup L(R3)$

Regular Expressions

odd number of 1, even number of 0

(credit: Stackoverflow)

```
((1|0(00)*01)((11|10(00)*01))*|(0(00)*1|(1|0(00)*01)((11|10(00)*01))*(0|10(00)*1))...
   ...((1(00)*1|(0|1(00)*01)((11|10(00)*01))*(0|10(00)*1)))*(0|1(00)*01)((11|10(00)*01))*)
```

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?: \r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[
\t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:( ?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\]
\000-\0 31]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\ ](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+ (?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?: (?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z
|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n) ?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\ r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n) ?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*) *:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+
|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r \n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?: \r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t ]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031 ]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\]( ?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(? :(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(? :\r\n)?[ \t])*))*\>(?:(?:\r\n)?[ \t])*)|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(? :(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?  [ \t]))*"(?:(?:\r\n)?[ \t])*)*:(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]| \\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<> @,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|" (?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?  :[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[
\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000- \031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|( ?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[
\t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,; :\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([ ^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[
\t])*(?:[^()<>@,;:\\" .\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\ ]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\ [\]
\000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\ r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\] |\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \0 00-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\ .|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@, ;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?  :[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])* (?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".
\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[ ^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]
]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*\>(?:(?:\r\n)?[ \t])*)(?:,\s*( ?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:( ?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[
\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t ])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t
])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?  :\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|
\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?: [^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\
]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*\<(?:(?:\r\n) ?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["
()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n) ?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>
@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,
;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)? (?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".
\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?: \r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\[
"()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t]) *))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])
+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\ .(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z
|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*\>(?:( ?:\r\n)?[ \t])*))*)?;\s*)
```

RFC822 (Mail Address)

```
^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$
```

IP address

(credit: https://www.regular-expressions.info/

Power of Regex

"15-02-2022 12:16 PM +05:30"

Is there a regex to recognize ... valid HTTP packets?

Valid email addresses? Valid date-time strings?

Valid JPEG files? Valid adjacency lists with cycles?

Is there a regex to recognize ... valid C programs?

$L = \{ w :$

w is of the form "int main() $\underbrace{\{\{\{}_{k}$ x $\underbrace{\}\}\}}_{k}$"

for some x not containing { and }

$\}$

$0^n 1^n$

$w \cdot w^{rev}$

$w : w = w^{rev}$

$w : \#0(w) = \#(1,w)$

$\{ \varepsilon, 00, 11, 010, 1011, \dots \} \Leftarrow$ DFA

$\Downarrow$

$\{0, 1, \dots\}$

$L2 = \{ x : x$ can be written as $ww \}$

$L(10 \cup 0^*1) = \{0^k 1 \mid k \geq 0\} \cup \{10\}$

$L((10) \cup (0^*1)) = L(10) \cup L(0^*1) = L(1) \cdot L(0) \cup L(0^*) L(1)$

$= \{10\} \cup \{0^k 1 : k \geq 1\}$

Regex for ...

All strings containing the substring 000 $= (0 \cup 1)^* \, 000 \, (0 \cup 1)^*$

$= w_1 000 w_2$

$\{0, 1\}$

All strings not containing the substring 000

$(1 \cup 01 \cup 001)^* (0 \cup 00)$

break string at 1's ⟹ each substring contains at most 2 0s and ends with 1.

Every string except 000

$\varepsilon \cup (0 \cup 1)^4 (0 \cup 1)^* \cup 0 \cup 1 \cup 00 \cup 01 \cup 10 \cup 11 \cup 001$
$\cup 011 \cup 010 \cup 100 \cup 101 \cup 110 \cup 111$

19 '∪'.

* Design a regex with at most 8 '∪'

$\#(0, w) = \#(1, w)$ for all even-length prefix

$(01 \cup 10)^* (0 \cup 1)$

All strings w s.t. in every prefix of w, the numbers of 0s and 1s differ by at most 1.

ε, 0, 01, 010, 0101

1, 10, 011, 0110

100, 1001

101, 1010

$L(1(0 \cup 1))^*(1 \cup \varepsilon) = ?$

Exercise

Can you design a regex for { w: w has equal occurrences of 01 and 10 }

010, 0110, 101, 1001, 000

Regex: $0 (0 \cup 1)^* 0$
$\cup \; 1 (0 \cup 1)^* 1$

$= \{ w : $ start and end with the same symbol $\}$

Kleene's Theorem

Regular language: Languages of Regexes

Thm: Regular languages are equivalent to DFA

Regex is not as powerful as Java

Regex is okay for verifying patterns ...

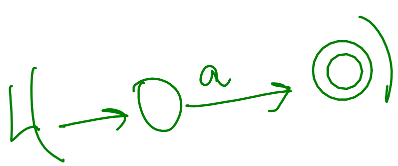
like C variable names, URLs ... (lexical an.)

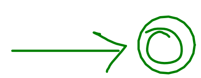
Can Regex verify syntax of C programs?

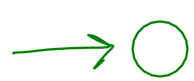
Thm: If L is described by a regex R, $L = L(R)$, then L is accepted by an NFA.

Proof by structural induction on R

Base cases:

$R = a$  $= L(R)$

$R = \varepsilon$ $\equiv$ 

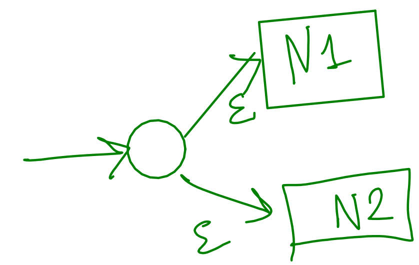
$R = \{\}$ $\equiv$ 

$R = R1 \cup R2$

$\Downarrow \quad \Downarrow$

$N1 \quad N2$

$L(N1) \quad L(N2)$



N s.t. $L(N) = L(N1) \cup L(N2)$
$= L(R)$

$R = R1 \cdot R2$

$\Downarrow \quad \Downarrow$

$L1 \quad L2$

$\Downarrow \quad \Downarrow$

$N1 \quad N2$

N s.t. $L(N) = L(N1) \cdot L(N2)$
$= L(R)$

$R = (R1)^*$

$\Downarrow$

$N1$

N s.t. $L(N) = (L(N1))^*$
$= (L(R1))^*$

Induction case:

Show that $(1(0 \cup 1))^+$ is regular

construct an equivalent NFA