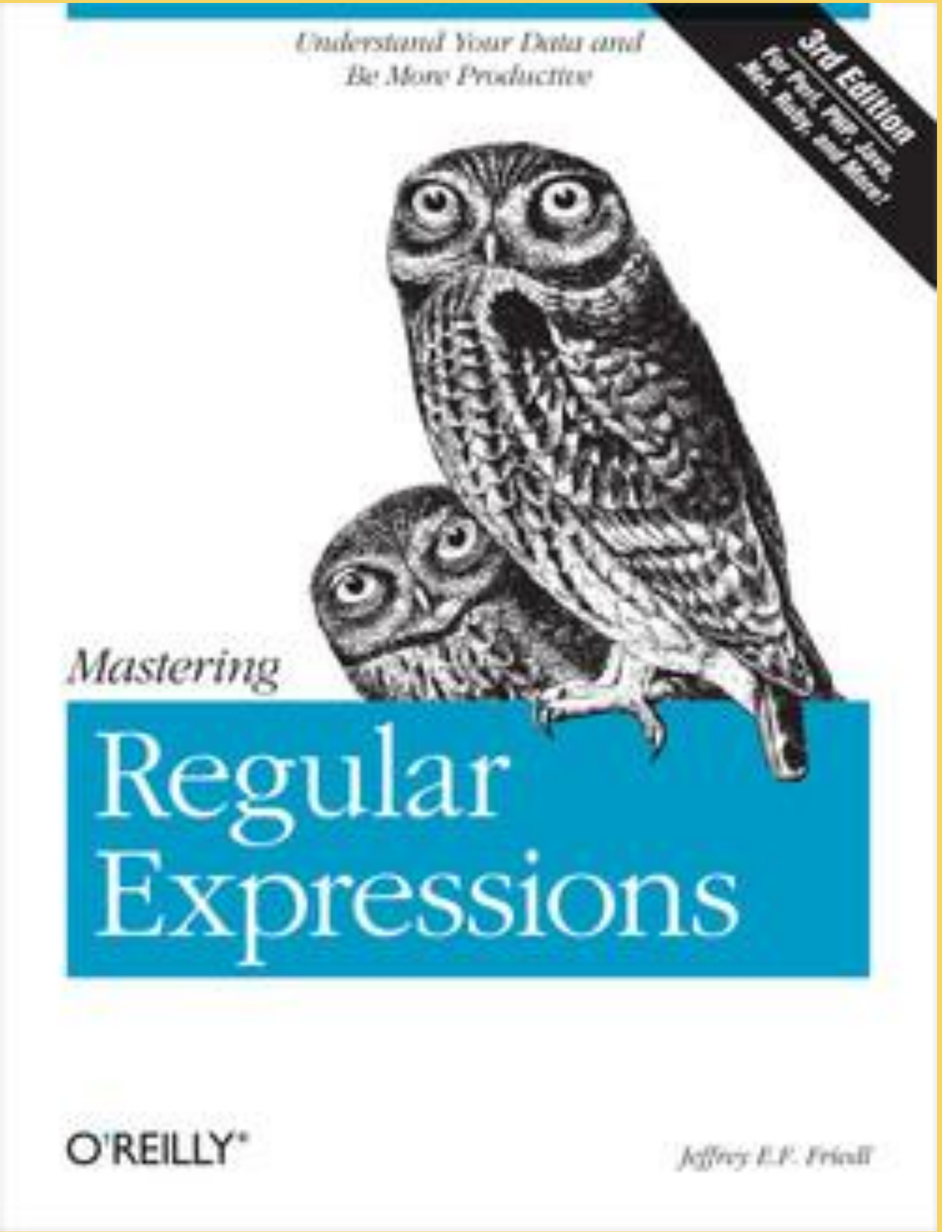


Midterm: Mar 6, 12:30pm
Same classroom

Problem Set 5:
Due this Friday (Feb 28)

PS 6 PRP 7

Class 13:
Regular Expression

University of Virginia
cs3120: DMT2
Wei-Kai Lin

Recap: Regular Expressions

Searching <u>patterns</u> with simple rules.

Example: we want a string of zeros only or ones only of length at least 2

OR

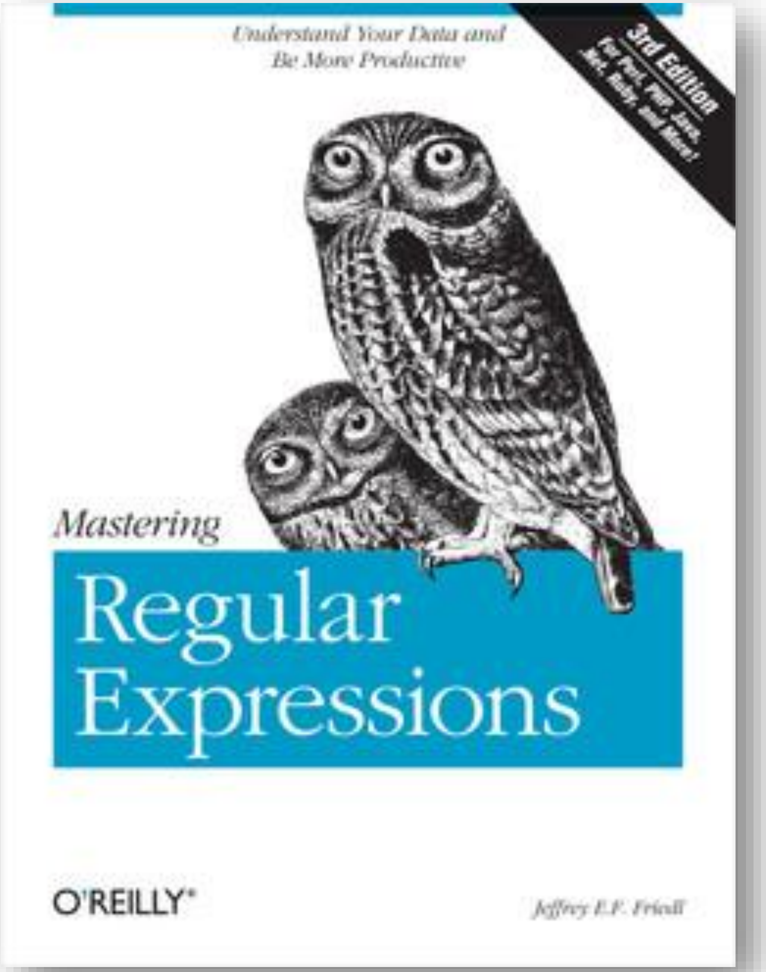
We denote them as: $(00(0^*)|11(1^*))$

- 00 simply means string 00
- 0^* means repeating 0 zero, or one, or two, or … times.
- | means OR

Many variants!

- (Linux command line, eg, Bash)
  *.pdf: any string ends with ‘.pdf’

- PRR5
  .*\.pdf: any string ends with ‘.pdf’



Python: 're'

10+ pages...

Regular Expression Syntax

A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

The special characters are:

`.`

(Dot.) In the default mode, this matches any character except a newline. If the `DOTALL` flag has been specified, this matches any character including a newline. `(?s:.)` matches any character regardless of flags.

`^`

(Caret.) Matches the start of the string, and in `MULTILINE` mode also matches immediately after each newline.

`$`

Matches the end of the string or just before the newline at the end of the string, and in `MULTILINE` mode also matches before a newline. `foo` matches both 'foo' and 'foobar', while the regular expression `foo$` matches only 'foo'. More interestingly, searching for `foo.$` in `'foo1\nfoo2\n'` matches 'foo2' normally, but 'foo1' in `MULTILINE` mode; searching for a single `$` in `'foo\n'` will find two (empty) matches: one just before the newline, and one at the end of the string.

`*`

Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible. `ab*` will match 'a', 'ab', or 'a' followed by any number of 'b's.

`+`

Causes the resulting RE to match 1 or more repetitions of the preceding RE. `ab+` will match 'a' followed by any non-zero number of 'b's; it will not match just 'a'.

https://docs.python.org/3/library/re.html#regular-expression-syntax

Regular expressions

For simplicity:
We consider matching (searching) on **binary strings**

Writing valid "regular expressions" using "regular operations"

Definition 6.6 (Regular expression)

A *regular expression* e over an alphabet Σ is a string over $\Sigma \cup \{(,),|,*,\emptyset,""\}$ that has one of the following forms:

$\{0,1\}$

1. $e = \sigma$ where $\sigma \in \Sigma$
2. $e = (e'|e'')$ where e', e'' are regular expressions.
3. $e = (e')(e'')$ where e', e'' are regular expressions. (We often drop the parentheses when there is no danger of confusion and so write this as $e'\ e''$.)
4. $e = (e')^*$ where e' is a regular expression.

Finally we also allow the following "edge cases": $e = \emptyset$ and $e = ""$. These are the regular expressions corresponding to accepting no strings, and accepting only the empty string respectively.

Syntax (Definition of Reg. Exp.)

empty set ~~exist~~ empty string

- Alphabet $\Sigma = \{0,1\}$
- Special symbols: (,) , * , | , Ø , “”

Reg. Exp. is just a string of {alphabet and special symbols}

1. $e =$ 0, 1, Ø, or “”
2. $e = (e')|(e'')$ (OR)
3. $e = (e')(e'')$ (concatenation)
4. $e = (e')^*$ (Kleene star)

where e' and e'' are reg. exp.

$x = \underset{e'}{x_1} \; \underset{e''}{x_2}$

$x = \underset{e'}{x_1} \; \underset{e'}{x_2} \cdots \underset{e'}{x_k}$

(Informal) Intuition

- $\Sigma = \{0,1\}$: exact match
- $\emptyset$: matches nothing
- “” : matches only empty string
- Special symbols: (,) , *
- (e’) | (e’’) : e’ matches OR e’’ matches
- (e’)(e’’) : e’ matches the prefix and e’’ matches the suffix
- (e’)* : matches a string “repeatedly” for 0 or more times

Example: prefix and suffix

Alphabet $\Sigma = \{0,1\}$. All strings with:

Prefix: 01 Suffix: 00

Eg: matches 0100, but not 01001

$(0^*)|(1^*)$

Intuitive:

$$e = (\cancel{00})(0|1)^*(00)$$

01

Haven't yet define "**how to match**" e to a string!

Example: XOR

ex: 0 0 1
1 0
1 1 1

NOT: 1 1 0
1 1

For $x = x_1 \dots x_n$ let $XOR(x) = x_1 \oplus x_2 \dots \oplus x_n$

Observation: there are **odd** number of '1's

Intuitive:

$e =$ $0^* 1 (0^* 1 0^* 1 0^*)^* 0^*$

$0^* 1 (0^* 1 0^* 1)^* 0^*$

Parenthesis and Precedence

- Drop parentheses when inferred from context
- Precedence (high to low)
  - Kleene star: $*$
  - Concatenation: (e’)(e’’)
  - OR: (e’) | (e’’)

- 00^* | 11 instead of $((0)(0^*))$ | $((1)(1))$

Regular Expressions as Functions

Goal: For every regular expression e,
there is a corresponding function $\Phi_e : \{0,1\}^* \to \{0,1\}$

Such that

$$\Phi_e(x) = 1 \text{ if } \; x \text{ matches } e.$$

Need definition!

Defining Φ_e defines the evaluation of e.

Recursive definition, but cumbersome.

Definition 6.6 (Regular expression)

A *regular expression* e over an alphabet Σ is a string over $\Sigma \cup \{(,),|,*,\emptyset,""\}$ that has one of the following forms:

0. $e = \emptyset$, or $e = ""$

$\Phi_{\emptyset}(x) = 0$, $\Phi_{""}(x) = \begin{cases} 1 & x = "" \\ 0 & \text{o.w.} \end{cases}$

1. $e = \sigma \in \{0,1\}$

$\Phi_{\sigma}(x) = \begin{cases} 1 & x = \sigma \\ 0 & \text{o.w.} \end{cases}$

2. $e = (e'|e'')$

$\Phi_{e}(x) = OR(\Phi_{e'}(x), \Phi_{e''}(x))$

$e = (0|1)$

e', e'' are regular expressions

Definition 6.6 (Regular expression)

A *regular expression* e over an alphabet Σ is a string over $\Sigma \cup \{(,),|,*,\emptyset,""\}$ that has one of the following forms:

3. $e = (e')(e'')$

$$\Phi_e(x) = \text{AND}(\Phi_{e'}(x'), \Phi_{e''}(x''))$$

$$x = x'x'' \in \{0,1\}^*$$

$x', x'' \rightarrow$ concatenate

4. $e = (e')^*$

$$\Phi_e(x) = \underset{i=1\cdots k}{\text{AND}}(\Phi_{e'}(x_i))$$

$$x = x_1 x_2 \cdots x_k \quad k \in \mathbb{N}$$

$$\exists k, x_1, \ldots x_k \in \{0,1\}^*$$

e', e'' are regular expressions

Definition 6.7 (Matching a regular expression)

Let e be a regular expression over the alphabet Σ. The function $\Phi_e : \Sigma^* \rightarrow \{0,1\}$ is defined as follows:

1. If $e = \sigma$ then $\Phi_e(x) = 1$ iff $x = \sigma$.
2. If $e = (e'|e'')$ then $\Phi_e(x) = \Phi_{e'}(x) \vee \Phi_{e''}(x)$ where $\vee$ is the OR operator.
3. If $e = (e')(e'')$ then $\Phi_e(x) = 1$ iff there is some $x', x'' \in \Sigma^*$ such that x is the concatenation of x' and x'' and $\Phi_{e'}(x') = \Phi_{e''}(x'') = 1$.
4. If $e = (e')*$ then $\Phi_e(x) = 1$ iff there is some $k \in \mathbb{N}$ and some $x_0, \ldots, x_{k-1} \in \Sigma^*$ such that x is the concatenation $x_0 \cdots x_{k-1}$ and $\underline{\Phi_{e'}(x_i) = 1}$ for every $\underline{i \in [k]}$. OR $x = ""$
5. Finally, for the edge cases $\Phi_\emptyset$ is the constant zero function, and $\Phi_{""}$ is the function that only outputs 1 on the empty string `""`.

We say that a regular expression e over Σ *matches* a string $x \in \Sigma^*$ if $\Phi_e(x) = 1$.

An (Python) algorithm evaluates regular expression

Syntactic Sugar

Useful in practice,
but **not** used in cs3120

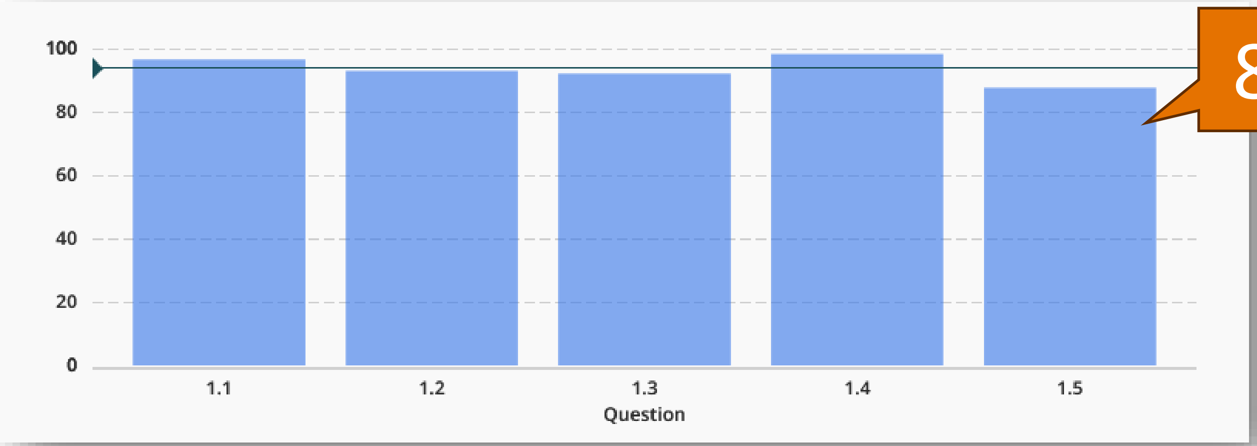
- Large alphabet:
  Digits 0,1,…,9. Letters a,b,…,z. Punctuations ‘,’ ‘.’ ‘(‘ …

  (0|1|2|……|a|b|……)

- Many special symbols:
  Any char: .   Any digit: \d
  Any in list: [abc…]
  Once or more: (e’)+
  Constant repetition: (e’){n}, (e’){n,m}
  Negation? Stay tuned…

$\underbrace{(e')(e')\cdots(e')}_{n}$

PRR5



88%

Q1.5
2 Points

Consider the following expression:

e = .*\.[pdf]*

x

Does it matches "ps5.pdf"?

- (●) Yes
- ( ) No

Syntax of the variant:
'.' : matches any char
'*' : Kleene star
'\' : escapes the next char
'[' ... ']' : matches any char in bracket

PRR5

"ps5" '\.' "*"

[pdf]*

(p|d|f)*

"pdf"

Q1.5
2 Points

Consider the following expression:

.*\.[pdf]*

Does it matches "ps5.pdf"?

- (●) Yes
- ( ) No

Syntax of the variant:
'.' : matches any char
'*' : Kleene star
'\' : escapes the next char
'[' ... ']' : matches any char in bracket

Regular Functions / Language

Definition:
We call a Boolean function $F: \{0,1\}^* \to \{0,1\}$ is **regular**,
If $F = \Phi_e$ for some regular expression e.

XOR

Equivalently, a language $L \subseteq \{0,1\}^*$ is **regular**
if and only if there is a regular expression e such that
$x \in L$ iff e matches x.

$L_{XOR} = \{1, 10, 01, \dots\}$

Complexity class: Regular Functions

Definition:

Let Reg-Fun be the set of all regular functions.

DFA-Comp , SIZE(s)

By definition:

For every $F \in$ Reg-Fun, there exists a regular expression e such that $\Phi_e = F$.

Theorem:
Reg-Fun $=$ DFA-Comp

F^c $\overline{\text{iff}}$ F^c

Theorem 6.17 (DFA and regular expression equivalency)

Let $F : \{0,1\}^* \to \{0,1\}$. Then F is regular if and only if there exists a DFA $(T, \mathcal{S})$ that computes F.

Definitions:

Reg-Fun: the set of all regular functions.

DFA-Comp: the set $\{f \mid f \text{ is computed by some DFA } M\}$

Theorem:

Reg-Fun $=$ DFA-Comp

Interpret: Reg-Fun ⊆ DFA-Comp:

- For every $F \in$ Reg-Fun, $F \in$ DFA-Comp

$\forall F = \Phi_e \quad \exists$ DFA M

$M(x)$ in $O(|x|)$

- Consequence: Every reg. exp. e, every x, matching $\Phi_e(x)$ is computable in time $O(|x|)$

e, x

Interpret: Reg-Fun $\supseteq$ DFA-Comp:

- For every F $\in$ DFA-Comp, F $\in$ Reg-Fun

- Consequence: instead of writing DFA, enough to write a regular expression

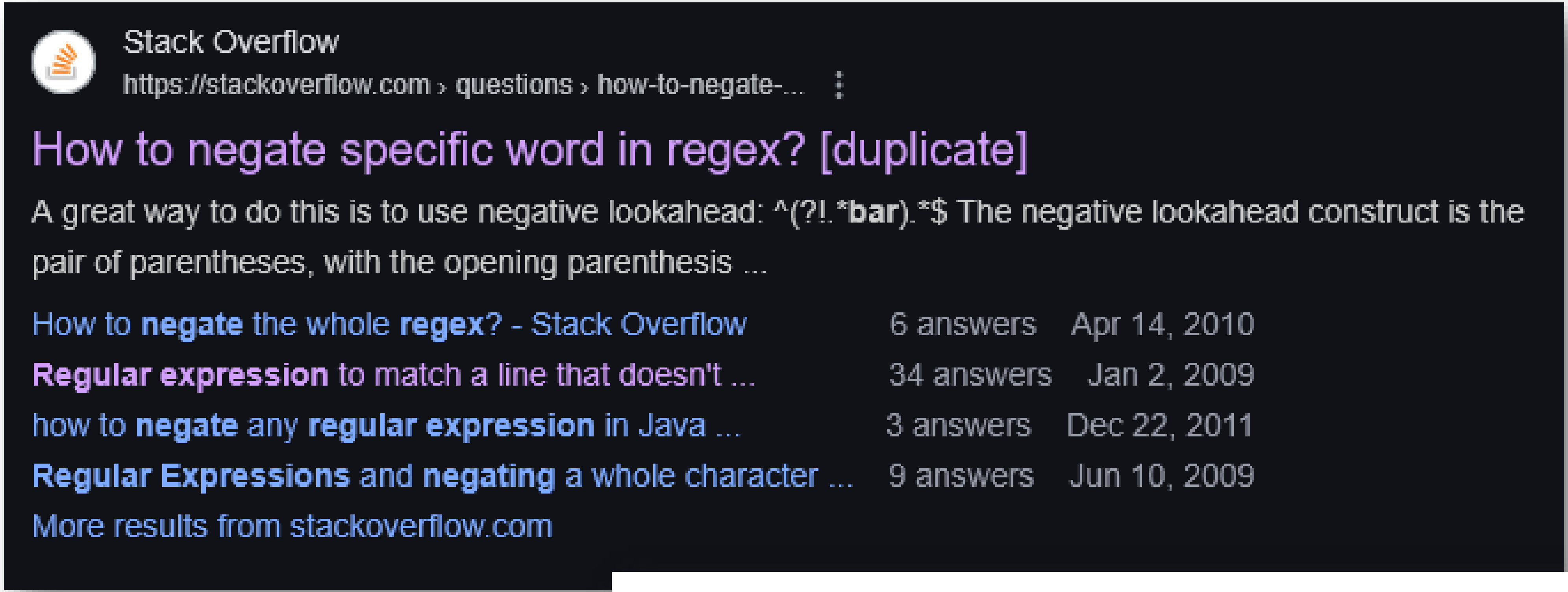
More Implication of Reg-Fun $=$ DFA-Comp

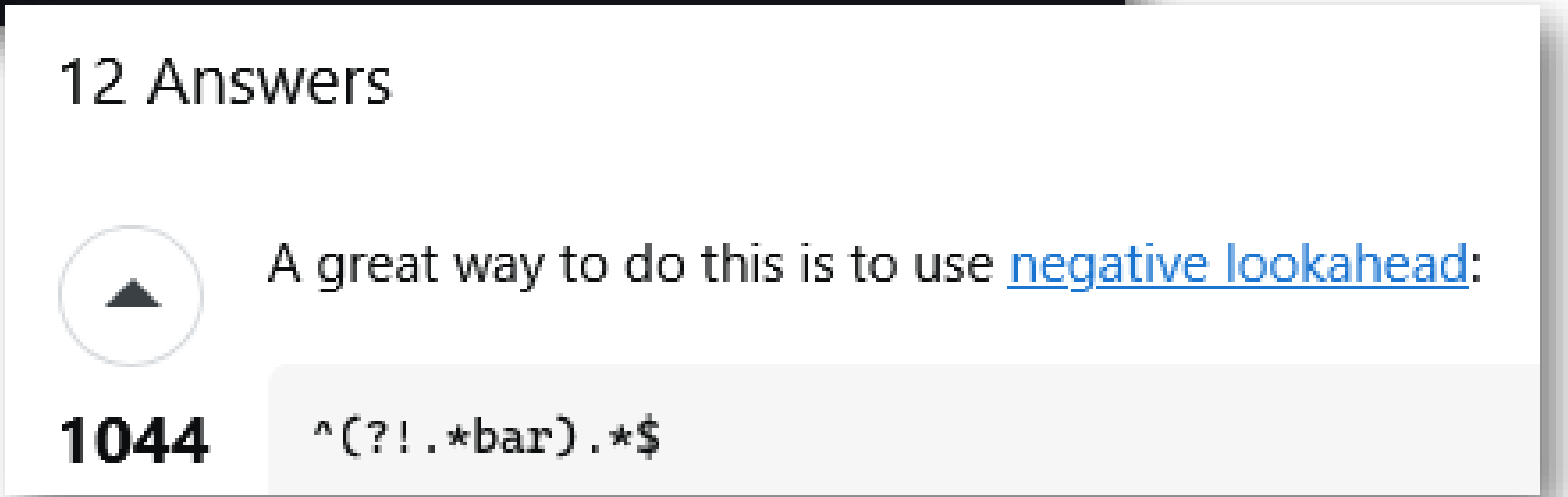
Complement of regular expression?

- For any reg exp e, is there a “negate” reg exp e’?
- I.e., to find e' such that for all string x,

$$\Phi_{e'}(x) = NOT(\Phi_e(x))$$

- Highly asked question!


Stack Overflow
https://stackoverflow.com › questions › how-to-negate-...
How to negate specific word in regex? [duplicate]
A great way to do this is to use negative lookahead: ^(?!.*bar).*$ The negative lookahead construct is the pair of parentheses, with the opening parenthesis ...
How to negate the whole regex? - Stack Overflow 6 answers Apr 14, 2010
Regular expression to match a line that doesn't ... 34 answers Jan 2, 2009
how to negate any regular expression in Java ... 3 answers Dec 22, 2011
Regular Expressions and negating a whole character ... 9 answers Jun 10, 2009
More results from stackoverflow.com


12 Answers
A great way to do this is to use negative lookahead:
1044
^(?!.*bar).*$

Reg-Fun = DFA-Comp

Complement of regular expression

- Suppose F is regular ($F = \Phi_e$ for some e)
- Does $\bar{F} = NOT(F(x))$ also have a regular expression?
- Yes!
- Suppose M is a DFA for F
- Let $\bar{M}$ be DFA that switches the accept/reject states of M
- $\bar{M}$ computes $\bar{F}$
- Then $\bar{F} \in$ DFA-Comp = Reg-Langs

$\bar{e}$

M q_0 q_1

$\bar{M}$ q_0 q_1

OR of DFA-comp functions are DFA-comp?

M_1, M_2

- Suppose $F_1, F_2 \in$ DFA-Comp.
- Does it hold $F(x) = OR\big(F_1(x), F_2(x)\big) \in$ DFA-Comp?
- Yes!
- Suppose e_b is a reg expression for F_b
- Let $e = (e_1)|(e_2)$
- F is Φ_e
- Therefore $F \in$ Reg-Fun = DFA-Comp

M

How about more complicated transformations?

- Suppose functions $f_1, f_2, \ldots, f_k$ are all regular functions.
- Let $T: \{0,1\}^* \to \{0,1\}$ be an arbitrary Boolean function.
- Is the function $f(x) = T\big(f_1(x), f_2(x), \ldots, f_k(x)\big)$ also regular?
- Yes!

- Proof sketch:
  - Union of two languages is the same as OR of their Boolean functions
  - Complement of a language is the same as NOT of its Boolean function
  - {OR,NOT} is universal set of gates.

The same for Regular Languages

- A language $L \subseteq \{0,1\}^*$ corresponds to a Boolean function $F: \{0,1\}^* \to \{0,1\}$

language $L \subseteq \{0,1\}^*$ is **regular**
if and only if there is a regular expression e such that $x \in L$ iff e matches x.

Limits of finite state computation

There is at least one non-regular function

- Proof:
- Reg-Fun is a countable set (why?)
- Set of all functions (All-Fun) is uncountable (why?)

- If All-Fun $\subseteq$ Reg-Fun, then All-Fun would have been countable

Any "natural" languages that is not regular?

Boolean Functions

- By Reg-Fun = DFA-Comp,
  Any regular function must be computable by a DFA
- Any DFA has constant "memory", ie, num of states

- Find a function needs more than const mem

Example A = $\{\mathbf{0}^{k}\mathbf{1}^{k}: k \in$ N $\}$

Theorem: A is not regular

Proof:

Give it 00 … 0 … as inputs till a state q is repeated.

Let $x = 0^{i}, y = 0^{j}$ such that $j > 0$ and
both x and $x \mid\mid y$ on q.

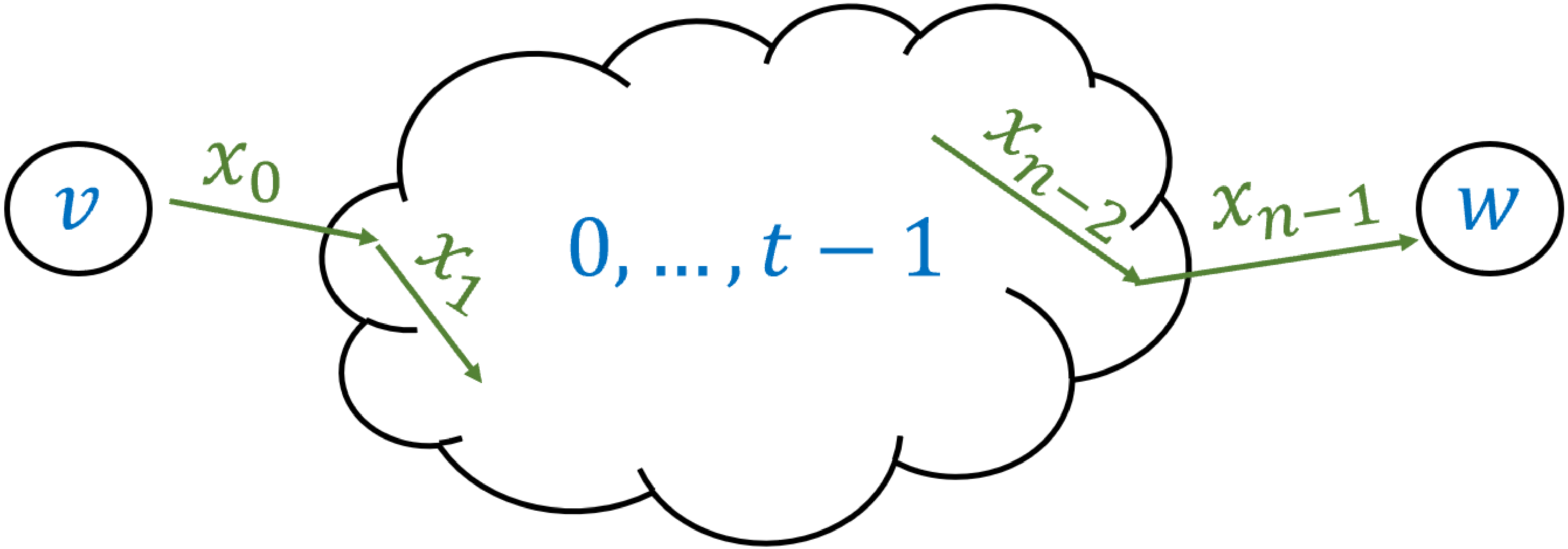
Consider $0^{i}1^{i}$ and $0^{i+j}1^{i}$.

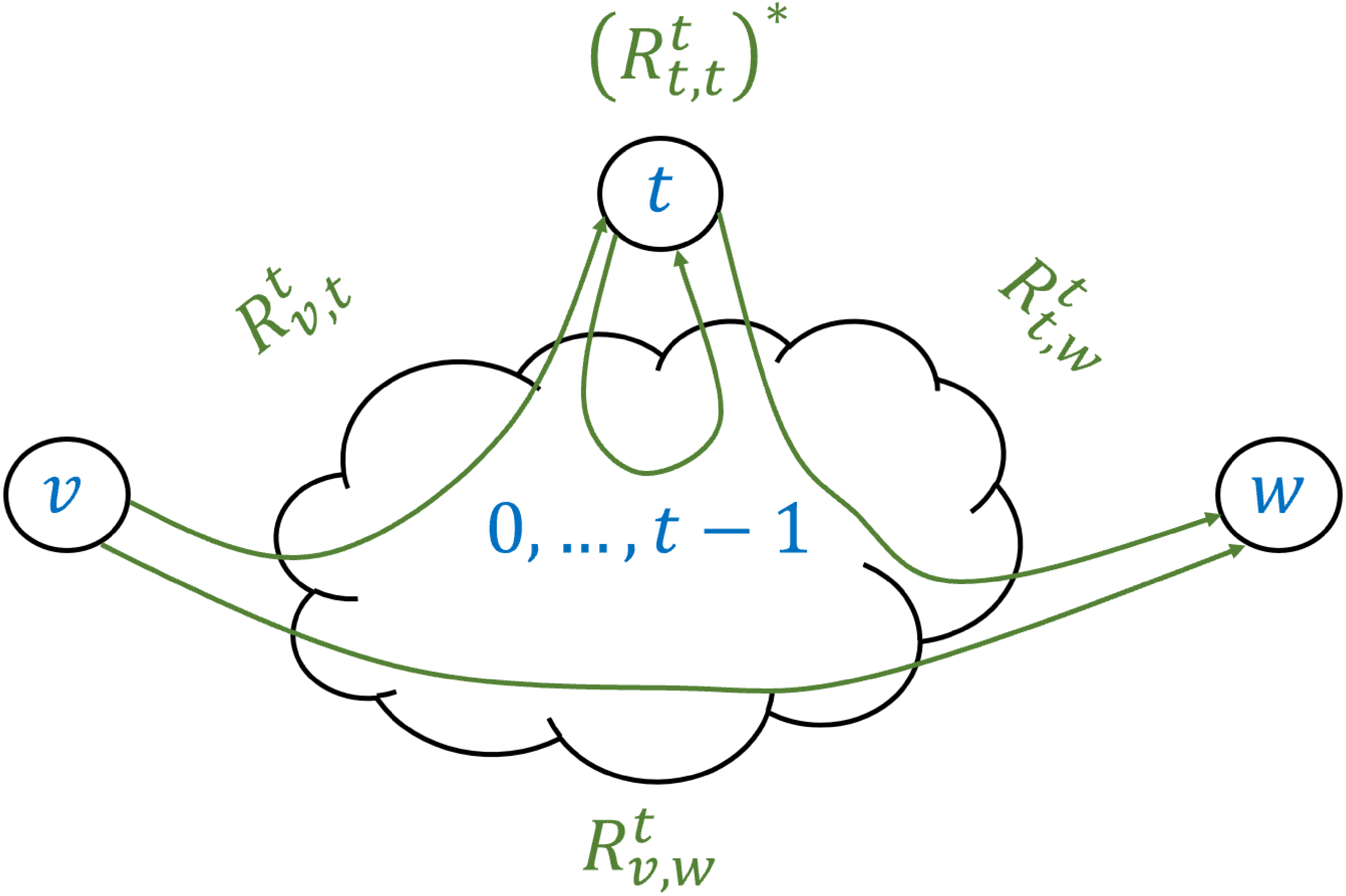
Either both will be accepted,

Or both will be rejected.

Reg-Fun $\subseteq$ DFA-Comp

TCS, Section 6.4.2


v
x_0
x_1
$0, \ldots, t-1$
x_{n-2}
x_{n-1}
w



$\left(R_{t,t}^{t}\right)^{*}$
t
$R_{v,t}^{t}$
$R_{t,w}^{t}$
v
$0, \ldots, t-1$
w
$R_{v,w}^{t}$

Charge

Regular Expressions

Reg-Fun = DFA-Comp

PS5: due this Friday, Feb 28, 10pm