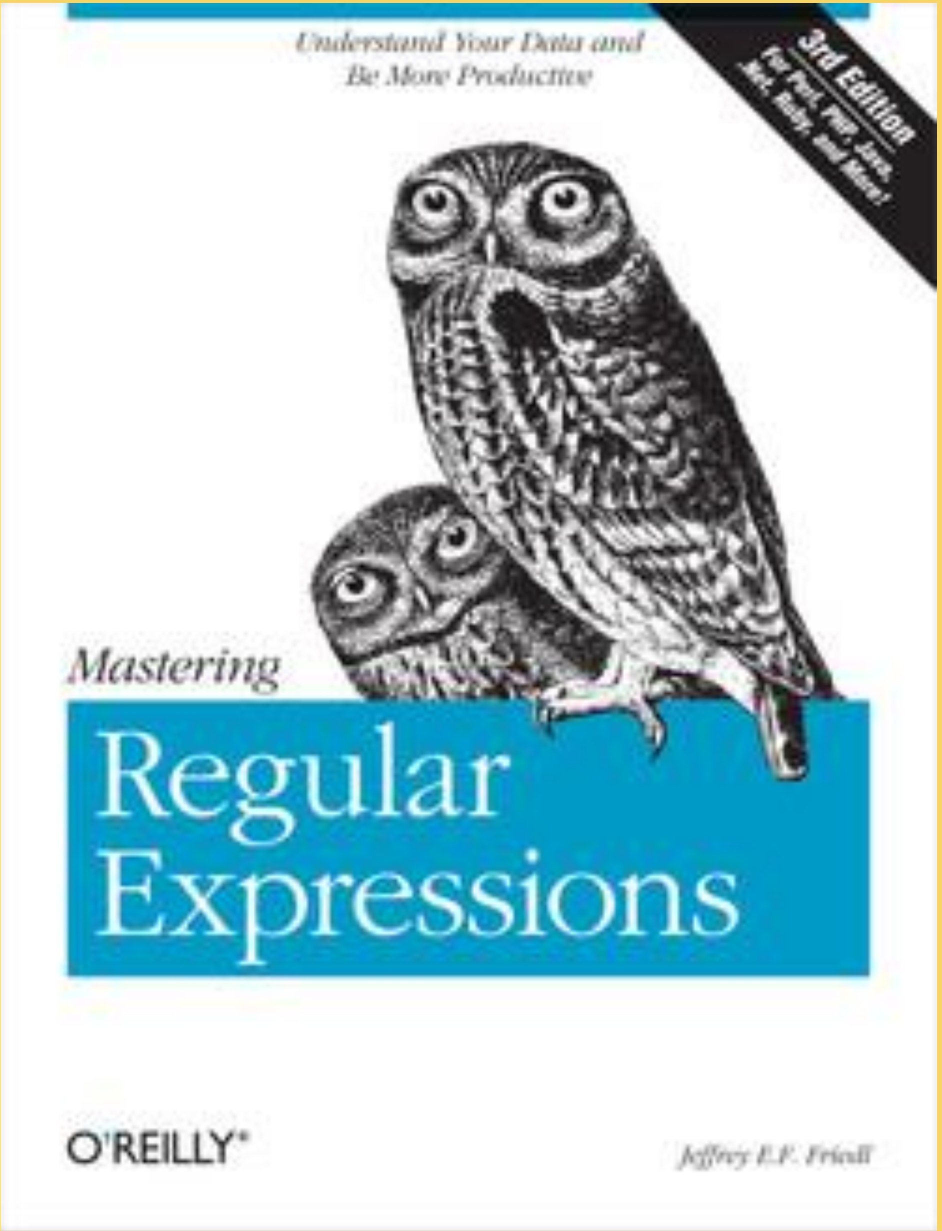


HW 1 due this Friday, Jan 30 (10:00pm)

Midterm next Tuesday (Feb 3) 9:30am in classroom

Class 5: *Regular Expressions*

University of Virginia
CS3120: DMT2
https://weikailin.github.io/cs3120-toc
Wei-Kai Lin

Plan

- **Regular Expressions**
- **Midterm review**

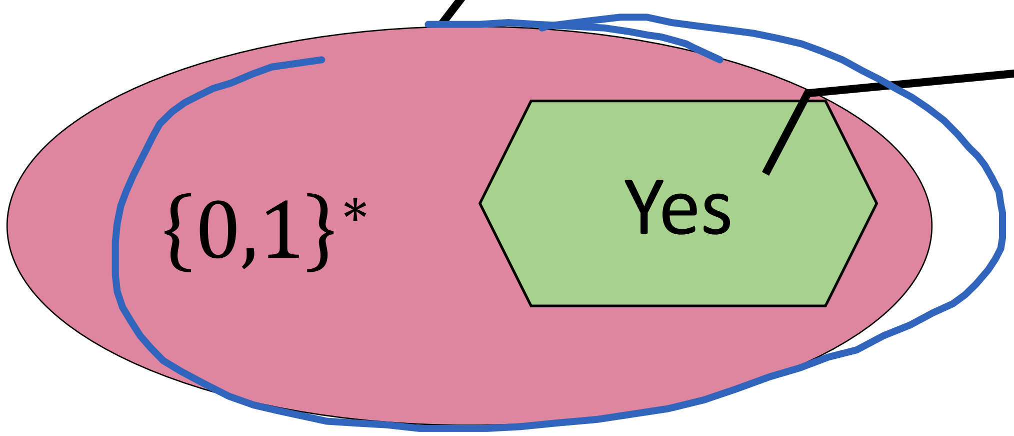
Today: Chapter 6 in the TCS book

https://introtcs.org/public/lec_05_infinite.html#regexpsec

Recap: Problems and Languages

Problem = *Language* = ~~Binary~~ Boolean Function

Any binary string $x \in \{0,1\}^*$ is an *instance*.



The subset of Yes instances:
The *Problem*, or the *Language*,
or the Boolean function
that outputs 1

Class DFA

Recap: Deterministic Finite Automata

Definition 6.2 (Deterministic Finite Automaton)

A deterministic finite automaton (DFA) with C states over $\{0,1\}$ is a pair $(T, \mathcal{S})$ with $T : [C] \times \{0,1\} \to [C]$ and $\mathcal{S} \subseteq [C]$. The finite function T is known as the *transition function* of the DFA. The set $\mathcal{S}$ is known as the set of *accepting states*.

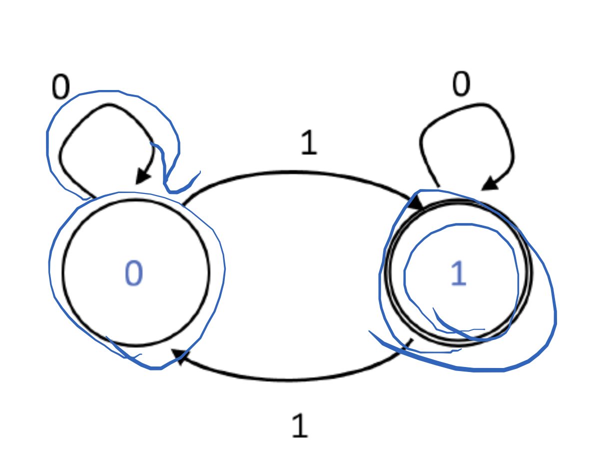
Let $F : \{0,1\}^* \to \{0,1\}$ be a Boolean function with the infinite domain $\{0,1\}^*$. We say that $(T, \mathcal{S})$ *computes* a function $F : \{0,1\}^* \to \{0,1\}$ if for every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, if we define $s_0 = 0$ and $s_{i+1} = T(s_i, x_i)$ for every $i \in [n]$, then

$$s_n \in \mathcal{S} \Leftrightarrow F(x) = 1$$

Definition of DFA

Definition of DFA output

Goal: Same machine that computes on every binary string $x \in \{0,1\}^*$.



Regular Expressions

A (seemingly) completely different way of dealing with infinite languages (i.e., functions on infinite input sets)

Motivation and example

Suppose we want to describe key-words for search

Simple examples: "book" or "February" or "a" or "10"
Intense example: a sorted sequence of integers unbound

Maybe crazy: a very large integer that is prime

Can we support all above?
Do we want to support all?

Motivation: simple rules

How about patterns with simple rules.

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

0 0 0 0 0

1 1 1 1

$(00)(0)^* \mid (11)(1)^*$

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

and or if

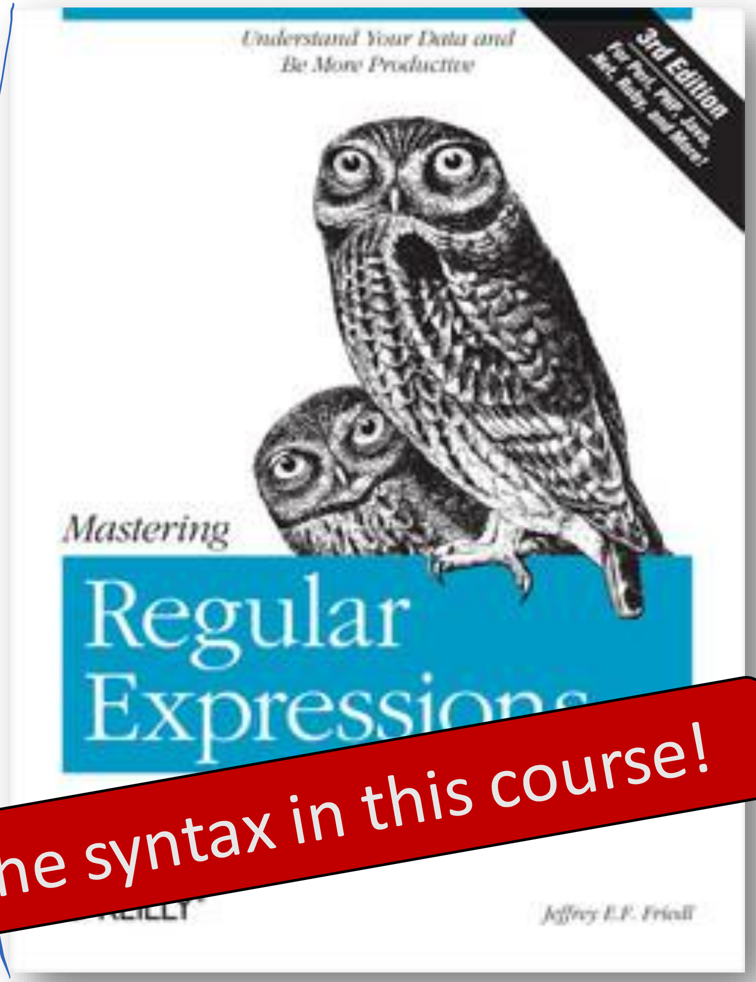
Regular Expression is Widely Used

Examples:

*.pdf: any string ends with ‘.pdf’

?.pdf: any string ends with ‘.pdf’ that is exactly 5 chars

Wildcards is a subset of regular expressions.



This is Not the syntax in this course!

Writing "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:

Often $\Sigma = \{0,1\}$ to be simple

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.

(Informal) Intuition

Σ^*

- $\Sigma = \{0,1\}$: exact match
- Ø : 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

base

concat

induct

Example: XOR

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

XOR(x) = 1 iff

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

$$\underline{(0^*10^*)}\underbrace{(0^*10^*10^*)^*}_{\text{even 1's}}$$

Parenthesis and Precedence

- Drop parentheses when inferred from context

- highest precedence to
  $*$, then
  concatenation, and then
  OR

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

Example: prefix and suffix

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

Prefix: 1101 Suffix: 1000

110|000

$e = 1101(0|1)^* 1000$

$(0|^*)$ N $(\phi)^*$ Y

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

Evaluating Regular Expressions

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

Such that

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

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 $e = ""$

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

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

Φ_0, $\Phi_1(x) = \begin{cases} 1 & \text{iff } x = "1" \\ 0 & \text{o.w.} \end{cases}$

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

$\Phi_{e'}, \Phi_{e''}$

$\Phi_e(x) = =$

$\Phi_{e'}(x) = 1$ OR $\Phi_{e''}(x) = 1$

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. $e = (e')(e'')$ $\Phi_e(x) = 1$ iff $\exists\, x_1, x_2$ s.t. $x_1 x_2 = x$ and $\Phi_{e'}(x_1) = 1$ and $\Phi_{e''}(x_2) = 1$

$n+1$

4. $e = (e')^*$ $\Phi_e(x) = 1$ if $\exists\, x_1, x_2, \dots x_k$ s.t. $\underline{x_1 x_2 \cdots x_k = x}$

$\underline{\Phi_{e'}(x_i) = 1}$ all $i = 1 \dots k$

$k = ?$ $|x| = n \in \mathbb{N}$

$x_1 = ""$ $\cancel{k \le n}$ $\cancel{k \le n}$ $k \in \mathbb{N}$

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 $\Phi_{e'}(x_i) = 1$ for every $i \in [k]$.
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 $\Phi_e(x) = 1$.

n

""

$\Phi_{e'}("") = 1$

Don't consider

An (Python) algorithm evaluates regular expression

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 … ble. `ab*` will match ‘a’, ‘ab’, or ‘a’ followed by any number of ‘b’…

`+`

Causes the resultin… RE. `ab+` will match ‘a’ followed by any non-zero numb… ‘a’.

This is Not the syntax in this course!

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

Syntactic Sugar

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

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

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

(0|1|2|…|9|a|b|…|z|…)

(e')(e')*

¬(e')

Negation? Stay tuned…

Complexity class: Regular Functions

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.

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.

Complexity class: Regular Functions

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

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

Theorem:
Reg-Fun $=$ DFA-Comp

Theorem 6.17 (DFA and regular expression equivalency)

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

reg exp   iff   DFA

Definitions:

Reg-Fun: the set $\{f \mid f = \Phi_e$ for some reg. exp. $e\}$

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

Theorem:

Reg-Fun = DFA-Comp (stay tuned for proof)

Theorem:
Reg-Fun $=$ DFA-Comp

class $\equiv$ set of func

For any two complexity classes C_1 and C_2,
what do we mean $C_1 = C_2$?

Set

$A = B$ iff $A \subseteq B$
$B \subseteq A$

What are C_1 and C_2 as math objects?

Set of funcs

Thm =

Interpret: Reg-Fun ⊆ DFA-Comp:

- For every F ∈ Reg-Fun, F ∈ DFA-Comp

  ∃ DFA M s.t. M(x) = F(x)
  all x

- Consequence: Every reg. exp. e, matching $\Phi_e(x)$ is computable in time $O(|x|)$ for all $x \in \{0,1\}^*$

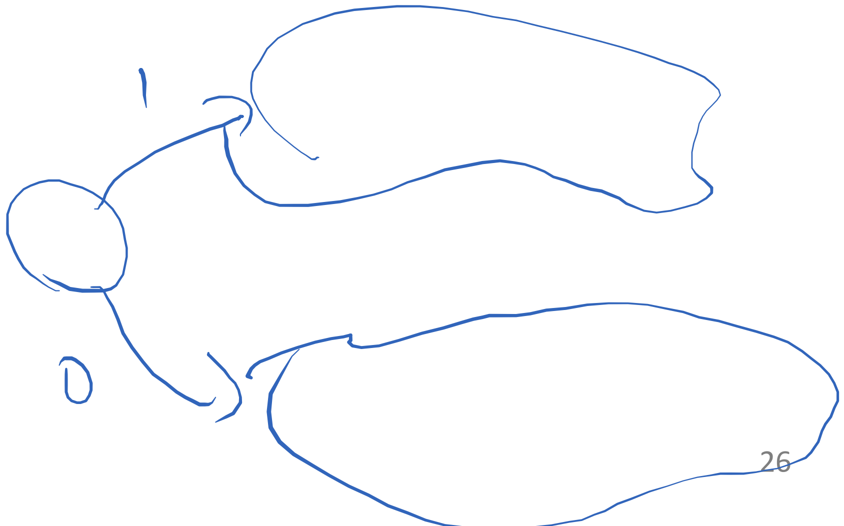
why? bcs DFA runs in O(|x|) time
|M|

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

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



Prepare for Midterm 0

Scope of Midterm 0

- Class 0 to Cantor's Thm in Class 4
- Quiz 0 to Quiz 2, HW 0 to HW 1
- Textbook Chapters 1 to 2 (some are not covered)

" ~~pe~~ prefix encoding "

Recap of Module 0

- Constructive definitions, natural numbers, binary strings, and induction
- Functions (bijection, surjection, …)
- Sets, cardinality of sets, infinite, countable, uncountable
- Cantor’s theorem

How to prepare?

- The HW and Quiz questions are closer format/material
  > Go through them
- Slides are close in materials (but you may want question
- Textbook exercises are good (some are not in scope)
- Ask LLM for more related exercises

Find some exercise questions that are similar to the following.

Which one of these objects can be represented by a binary string?

An integer x

An undirected graph G

A directed graph H

All of the above.

That's a classic foundational question in computer science! It touches on the concept of **encoding**, which is the idea that almost any structured data can be flattened into a sequence of 0s and 1s for a computer to process.

Here are a few practice questions that explore similar themes—ranging from data structures to complexity and logic.

Format of Midterm 0

- We start 9:30am next Tuesday, Feb 3
- 20 minutes
- Closed book and everything, except for 1 page of letter-size double-sided sheet
- T/F, multiple choice, proof
- Simple calculation, reciting is not expected (not even the sheet)

Purposes behind exam

- Grading: to meet the criteria of CS department
  https://uvacsadvising.org/policies.html#cs-department-grading-guidelines

- WK's selfish goal:
  to discover your potential in theoretical CS
  (No need to be discouraged if it is too hard)

Logistics

- HW 2 will be posted in 24 hours
  about DFA and reg. exp., due next Friday
- We want to publish HW 1, but the earliest is next Monday
- WK will be in person on Feb 3.