Formal Languages

FOCS

Sept 14,16, 2020

Math notation review

Sets: $\{ 2, 3, 5, 7, 11, \ldots\}$ $\{a, b, c\}$

- S finite if there is an n with $|S| = n$, infinite otherwise

Empty set: $\{\}$ or $\varnothing$

Membership: $x \in S$

$A = B$ if A and B have exactly the same elements

$A \subseteq B$ if every element of A is an element of B

- $\varnothing \subseteq A$; $A \subseteq A$; if $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$; $A = B$ if $A \subseteq B$ and $B \subseteq A$

Math notation review

$A \subset B$ if $A \subseteq B$ but $A \neq B$

Set comprehension $\{ x \mid P(x) \}$ where P is a property

Union $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$

Intersection $A \cap B = \{ x \mid x \in A \text{ and } x \in B \}$

Complement $\sim A = \{ x \mid x \in U \text{ and } x \notin A \}$ (with respect to a universe)

Cartesian product $A \times B = \{ (x,y) \mid x \in A \text{ and } y \in B \}$

Generalizes to $A_1 \times A_2 \times \ldots \times A_k$ in the obvious way

Math notation review

Function $f : A \rightarrow B$ (domain A, codomain B)

Image of f = $\{ b \in B \mid f(a) = b \text{ for some } a \in A\}$

Composition: if $f : A \rightarrow B$ and $g : B \rightarrow C$ then:

$g \circ f : A \rightarrow C$ defined by $(g \circ f)(a) = g(f(a))$

- A function is one-to-one if it maps distinct values to distinct values
- A function is onto if every value in the codomain is in the image
- A function is a correspondence if it is one-to-one and onto

Function definitions and computations

Intuitively, given a function $f : A \to B$, a computation of f is an *implementation* of f

The goal of the first half of the course is to describe possible choices for what implementation means

We want to define: when if a function $f : A \to B$ *computable*, that is

Looking at functions of the form $A \to B$ is way too broad — there's nowhere to being

So historically we have been restricted the forms that A and B take

Types of functions

Logicians study *natural number functions* of the form

$$\mathbb{N} \times \ldots \times \mathbb{N} \to \mathbb{N}$$

Computer scientists study *decision functions* of the form

$$\Sigma^* \to \{ \text{true}, \text{false} \}$$

where Σ^* is the set of strings over alphabet Σ

- If $\Sigma = \{ 0,1 \}$ then Σ^* is binary strings

Decision functions

Decision functions are studied because they are easier to describe:

A function $f : \Sigma^* \rightarrow \{ \text{true}, \text{false} \}$ can be completely described by a set

$C_f = \{ x \in \Sigma^* \mid f(x) = \text{true} \}$ - the characteristic set of f $C_f \subseteq \Sigma^*$

Indeed, given C_f you can reconstruct f by taking:

$f(x) = \text{true}$ if $x \in C_f$ and false otherwise

So we are going to study which decision functions are computable by studying characteristic sets.

Strings

Let Σ be a finite set of symbols - an *alphabet*

A string over alphabet Σ is a possibly empty finite sequence of symbols from Σ

The length of a string is the number of symbols in the sequence

E.g. *abcbbc* is a string over alphabet { a, b, c} of length 6

Empty string is represented by ε (length = 0)

Σ^* = set of all strings over Σ

String concatenation

String concatenation : if $u = a_1a_2...a_m$ $v = b_1b_2...b_n$ then $uv = a_1a_2...a_mb_1b_2...b_n$

- $\varepsilon u = u$
- $u\varepsilon = u$
- $u^0 = \varepsilon$
- $u^1 = u$
- $u^2 = uu$
- $u^3 = uuu$
- …
- Property: $u^m u^n = u^{m+n}$

Formal languages

A formal language over alphabet Σ is a set of strings over Σ (= a subset of Σ^*)

Because languages over Σ are sets, they support $\cup$, $\cap$, ~

They support other operations because they are sets of *strings*

- $A \cdot B = \{ s \mid s = uv \text{ for some } u, v \text{ with } u \in A \text{ and } v \in B\}$

  Example: $\Sigma = \{a, b, x, y\}$ $A = \{ aa, b \}$ $B = \{ x, xy \}$

  $A \cdot B = \{ aax, aaxy, bx, bxy \}$ $B \cdot A = \{ xaa, xb, xyaa, xyb \}$
- $A^0 = \{ \varepsilon \}$
- $A^1 = A$
- $A^2 = A \cdot A$ $A^3 = A \cdot A \cdot A = A \cdot A^2$ $A^{n+1} = A \cdot A^n$
- Property: $A^m \cdot A^n = A^{m+n}$

Kleene star iteration

$A^* = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \ldots$

$\quad = \bigcup_{k \geq 0} A^k$

A^* is the set of strings you get by concatenating a finite number of strings from A

Example: $A = \{ aa, bc \}$

$A^0 = \{ \varepsilon \}$

$A^1 = \{ aa, bc \}$

$A^2 = \{ aaaa, aabc, bcaa, bcbc \}$

$A^3 = \{ aaaaaa, aaaabc, aabcaa, aabcbc, bcaaaa, bcaabc, bcbcaa, bcbcbc \}$

...

$A^* = \{ \varepsilon, aa, bc, aaaa, aabc, bcaa, bcbc, aaaaaa, aaaabc, aabcaa, aabcbc, \ldots \}$

Regular languages

Let Σ be an alphabet $\{ a_1, a_2, \ldots, a_k \}$

A language $A \subseteq \Sigma^*$ is **regular** if A can be constructed by a finite sequence of steps using operations $\cup$, $\cdot$, and $*$ starting with the sets $\varnothing$, $\{ a_1 \}$, $\{ a_2 \}$, …, $\{ a_k \}$

Example: Take $\Sigma = \{ a, b \}$

The set E of strings of a's and b's of even length is regular:

$\{ a \} \cup \{ b \} = \{ a, b \}$
$\{ a, b \} \cdot \{ a, b \} = \{ aa, ab, ba, bb \}$
$\{ aa, ab, ba, bb \}^* = E$

Regular expressions

You may have heard about regular expressions.

They've been used since the late 60s for searching for patterns in text — cf. grep

The use of the term "regular" is not accidental. Regular expressions are just a convenient notation for regular languages.

Definition of regular expressions

A regular expression over alphabet Σ is an expression of the form:

1	
0	
x	(where $x \in \Sigma$)
$r_1 + r_2$	(where r_1 and r_2 are regular expressions)
$r_1 r_2$	(where r_1 and r_2 are regular expressions)
$r_1{}^*$	(where r is a regular expression)

Examples with $\Sigma = \{ a,b \}$:

ab,   a*b,   ab*a,   a(a+b)*b,   ((a+b)(a+b))*

Usual intuition: matching

There is the idea of a string *matching* a regular expression over Σ:

The empty string is the only string matching 1
No string matches 0
The string x is the only string matching x
A string matches $r_1 + r_2$ if it matches r_1 or it matches r_2
A string s matches $r_1\ r_2$ if $s = uv$ and u matches r_1 and v matches r_2
A string s matches r^* if $s = u_1u_2...u_k$ $(k \geq 0)$ and each u_i matches r

Example

Consider the regular expression ab*a. I claim *abbba* matches ab*a

You can split as r_1 = a and r_2 = b*a

abbba matches a b*a because *abbba* = *a bbba* and:

- *a* matches a : immediate
- *bbba* matches b*a : taking r_1 = b* and r_2 = a and *bbba* = *bbb a*
    - *bbb* matches b*, because *bbb* = *b b b* and each *b* matches b
    - *a* matches a

Language denoted by a regular expression

Easiest way to define matching is to define the set of strings denoted by a regular expression - defined recursively over the structure of regular expression:

$\text{lang}(1) = \{ \boldsymbol{\varepsilon} \}$
$\text{lang}(0) = \varnothing$
$\text{lang}(x) = \{ x \}$
$\text{lang}(r_1 + r_2) = \text{lang}(r_1) \cup \text{lang}(r_2)$
$\text{lang}(r_1\ r_2) = \text{lang}(r_1) \cdot \text{lang}(r_2)$
$\text{lang}(r^*) = \text{lang}(r)^*$

Example: lang(ab*a) = { *aa*, *aba*, *abba*, *abbba*, *abbbba*, *abbbbba*, ...} }

A string s matches regular expression r exactly when $s \in \text{lang}(r)$

Regular expressions denote regular languages

***Theorem**: A language L is regular if and only if there exists a regular expression r with lang(r) = L*