

b

Strings and Languages

Lecture 1b

Tuesday, January 15, 2019

LaTeXed: January 14, 2019 20:01

Part I

Strings

String Definitions

Definition

- 1 An **alphabet** is a **finite** set of symbols. For example
 $\Sigma = \{0, 1\}$, $\Sigma = \{a, b, c, \dots, z\}$,
 $\Sigma = \{\langle \text{moveforward} \rangle, \langle \text{moveback} \rangle\}$ are alphabets.
- 2 A **string/word** over Σ is a **finite sequence** of symbols over Σ .
For example, '0101001', 'string', ' $\langle \text{moveback} \rangle \langle \text{rotate90} \rangle$ '
- 3 ϵ is the **empty string**.
- 4 The **length** of a string w (denoted by $|w|$) is the number of symbols in w . For example, $|101| = 3$, $|\epsilon| = 0$
- 5 For integer $n \geq 0$, Σ^n is set of all strings over Σ of length n .
 Σ^* is the set of all strings over Σ .

Formally

Formally strings are defined recursively/inductively:

- ϵ is a string of length 0
- ax is a string if $a \in \Sigma$ and x is a string. The length of ax is $1 + |x|$

The above definition helps prove statements rigorously via induction.

- Alternative recursive definition useful in some proofs: xa is a string if $a \in \Sigma$ and x is a string. The length of xa is $1 + |x|$

Convention

- a, b, c, \dots denote elements of Σ
- w, x, y, z, \dots denote strings
- A, B, C, \dots denote sets of strings

Much ado about nothing

- ϵ is a **string** containing no symbols. It is not a set
- $\{\epsilon\}$ is a **set** containing one string: the empty string. It is a set, not a string.
- \emptyset is the **empty set**. It contains no strings.
- $\{\emptyset\}$ is a **set** containing one element, which itself is a set that contains no elements.

Concatenation and properties

- If x and y are strings then xy denotes their concatenation. Formally we define concatenation recursively based on definition of strings:
 - $xy = y$ if $x = \epsilon$
 - $xy = a(wy)$ if $x = aw$

Sometimes xy is written as $x \bullet y$ to explicitly note that \bullet is a binary operator that takes two strings and produces another string.

- concatenation is associative: $(uv)w = u(vw)$ and hence we write uvw
- **not** commutative: uv not necessarily equal to vu
- identity element: $\epsilon u = u\epsilon = u$

Substrings, prefix, suffix, exponents

Definition

- 1 v is **substring** of w iff there exist strings x, y such that $w = xvy$.
 - If $x = \epsilon$ then v is a **prefix** of w
 - If $y = \epsilon$ then v is a **suffix** of w
- 2 If w is a string then w^n is defined inductively as follows:
 $w^n = \epsilon$ if $n = 0$
 $w^n = ww^{n-1}$ if $n > 0$

Example: $(\text{blah})^4 = \text{blahblahblahblah}$.

Set Concatenation

Definition

Given two sets A and B of strings (over some common alphabet Σ) the concatenation of A and B is defined as:

$$AB = \{xy \mid x \in A, y \in B\}$$

Example: $A = \{fido, rover, spot\}$, $B = \{fluffy, tabby\}$ then $AB = \{fidofluffy, fidotabby, roverfluffy, \dots\}$.

Definition

- 1 Σ^n is the set of all strings of length n . Defined inductively as follows:
$$\Sigma^n = \{\epsilon\} \text{ if } n = 0$$
$$\Sigma^n = \Sigma\Sigma^{n-1} \text{ if } n > 0$$
- 2 $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$ is the set of all finite length strings
- 3 $\Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$ is the set of non-empty strings.

Definition

- ① Σ^n is the set of all strings of length n . Defined inductively as follows:

$$\Sigma^n = \{\epsilon\} \text{ if } n = 0$$

$$\Sigma^n = \Sigma\Sigma^{n-1} \text{ if } n > 0$$

- ② $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$ is the set of all finite length strings
- ③ $\Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$ is the set of non-empty strings.

Definition

A **language** L is a set of strings over Σ . In other words $L \subseteq \Sigma^*$.

Exercise

Answer the following questions taking $\Sigma = \{0, 1\}$.

- 1 What is Σ^0 ? $\{\epsilon\}$
- 2 How many elements are there in Σ^3 ? $2^3 = 8$
- 3 How many elements are there in Σ^n ? 2^n
- 4 What is the length of the longest string in Σ^* ? Does Σ^* have strings of infinite length?
- 5 If $|u| = 2$ and $|v| = 3$ then what is $|u \cdot v|$?
- 6 Let u be an arbitrary string Σ^* . What is ϵu ? What is $u \epsilon$?
- 7 Is $uv = vu$ for every $u, v \in \Sigma^*$?
- 8 Is $(uv)w = u(vw)$ for every $u, v, w \in \Sigma^*$?

Canonical order and countability of strings

Definition

An set A is **countably infinite** if there is a bijection f between the natural numbers and A .

Alternatively: A is countably infinite if A is an infinite set and there enumeration of elements of A

Canonical order and countability of strings

Definition

An set A is **countably infinite** if there is a bijection f between the natural numbers and A .

Alternatively: A is countably infinite if A is an infinite set and there enumeration of elements of A

Theorem

Σ^* is countably infinite for every finite Σ .

Enumerate strings in order of increasing length and for each given length enumerate strings in dictionary order (based on some fixed ordering of Σ).

Example: $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \dots\}$.

$\{a, b, c\}^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, \dots\}$

Exercise

Question: Is $\Sigma^* \times \Sigma^* = \{(x, y) \mid x, y \in \Sigma^*\}$ countably infinite?

Exercise

Question: Is $\Sigma^* \times \Sigma^* = \{(x, y) \mid x, y \in \Sigma^*\}$ countably infinite?

Question: Is $\Sigma^* \times \Sigma^* \times \Sigma^* = \{(x, y, z) \mid x, y, z \in \Sigma^*\}$ countably infinite?

Inductive proofs on strings

Inductive proofs on strings and related problems follow inductive definitions.

Definition

The **reverse** w^R of a string w is defined as follows:

- $w^R = \epsilon$ if $w = \epsilon$
- $w^R = x^R a$ if $w = ax$ for some $a \in \Sigma$ and string x

Inductive proofs on strings

Inductive proofs on strings and related problems follow inductive definitions.

Definition

The **reverse** w^R of a string w is defined as follows:

- $w^R = \epsilon$ if $w = \epsilon$
- $w^R = x^R a$ if $w = ax$ for some $a \in \Sigma$ and string x

Theorem

Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Example: $(dog \cdot cat)^R = (cat)^R \cdot (dog)^R = tacgod$.

Principle of mathematical induction

Induction is a way to prove statements of the form $\forall n \geq 0, P(n)$ where $P(n)$ is a statement that holds for integer n .

Example: Prove that $\sum_{i=0}^n i = n(n+1)/2$ for all n .

Induction template:

- **Base case:** Prove $P(0)$
- **Induction hypothesis:** Let $k > 0$ be an **arbitrary** integer. Assume that $P(n)$ holds for any $k \leq n$.
- **Induction Step:** Prove that $P(n)$ holds, for $n = k + 1$.

Structured induction

- ① Unlike simple cases we are working with...
- ② ...induction proofs also work for more complicated “structures”.
- ③ Such as strings, tuples of strings, graphs etc.
- ④ See class notes on induction for details.

Proving the theorem

Theorem

Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof: by induction.

On what?? $|uv| = |u| + |v|$?

$|u|$?

$|v|$?

What does it mean to say “induction on $|u|$ ”?

By induction on $|u|$

Theorem

Prove that for any strings $u, v \in \Sigma^$, $(uv)^R = v^R u^R$.*

Proof by induction on $|u|$ means that we are proving the following.

Base case: Let u be an arbitrary string of length 0. $u = \epsilon$ since there is only one such string. Then

$$(uv)^R = (\epsilon v)^R = v^R = v^R \epsilon = v^R \epsilon^R = v^R u^R$$

By induction on $|u|$

Theorem

Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof by induction on $|u|$ means that we are proving the following.

Base case: Let u be an arbitrary string of length 0 . $u = \epsilon$ since there is only one such string. Then

$$(uv)^R = (\epsilon v)^R = v^R = v^R \epsilon = v^R \epsilon^R = v^R u^R$$

Induction hypothesis: $\forall n \geq 0$, for any string u of length n (for all strings $v \in \Sigma^*$, $(uv)^R = v^R u^R$).

By induction on $|u|$

Theorem

Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof by induction on $|u|$ means that we are proving the following.

Base case: Let u be an arbitrary string of length 0 . $u = \epsilon$ since there is only one such string. Then

$$(uv)^R = (\epsilon v)^R = v^R = v^R \epsilon = v^R \epsilon^R = v^R u^R$$

Induction hypothesis: $\forall n \geq 0$, for any string u of length n (for all strings $v \in \Sigma^*$, $(uv)^R = v^R u^R$).

Note that we did not assume anything about v , hence the statement holds for all $v \in \Sigma^*$.

Inductive step

- Let u be an arbitrary string of length $n > 0$. Assume inductive hypothesis holds for all strings w of length $< n$.
- Since $|u| = n > 0$ we have $u = ay$ for some string y with $|y| < n$ and $a \in \Sigma$.
- Then

Inductive step

- Let u be an arbitrary string of length $n > 0$. Assume inductive hypothesis holds for all strings w of length $< n$.
- Since $|u| = n > 0$ we have $u = ay$ for some string y with $|y| < n$ and $a \in \Sigma$.
- Then

$$(uv)^R =$$

Inductive step

- Let u be an arbitrary string of length $n > 0$. Assume inductive hypothesis holds for all strings w of length $< n$.
- Since $|u| = n > 0$ we have $u = ay$ for some string y with $|y| < n$ and $a \in \Sigma$.
- Then

$$\begin{aligned}(uv)^R &= ((ay)v)^R \\ &= (a(yv))^R \\ &= (yv)^R a^R \\ &= (v^R y^R) a^R \\ &= v^R (y^R a^R) \\ &= v^R (ay)^R \\ &= v^R u^R\end{aligned}$$

Theorem

Prove that for any strings $u, v \in \Sigma^$, $(uv)^R = v^R u^R$.*

Proof by induction on $|v|$ means that we are proving the following.

Theorem

Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof by induction on $|v|$ means that we are proving the following.

Induction hypothesis: $\forall n \geq 0$, for any string v of length n (for all strings $u \in \Sigma^*$, $(uv)^R = v^R u^R$).

Theorem

Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof by induction on $|v|$ means that we are proving the following.

Induction hypothesis: $\forall n \geq 0$, for any string v of length n (for all strings $u \in \Sigma^*$, $(uv)^R = v^R u^R$).

Base case: Let v be an arbitrary string of length 0 . $v = \epsilon$ since there is only one such string. Then

$$(uv)^R = (u\epsilon)^R = u^R = \epsilon u^R = \epsilon^R u^R = v^R u^R$$

Inductive step

- Let v be an arbitrary string of length $n > 0$. Assume inductive hypothesis holds for all strings w of length $< n$.
- Since $|v| = n > 0$ we have $v = ay$ for some string y with $|y| < n$ and $a \in \Sigma$.
- Then

$$\begin{aligned}(uv)^R &= (u(ay))^R \\ &= ((ua)y)^R \\ &= y^R(ua)^R \\ &= ??\end{aligned}$$

Inductive step

- Let v be an arbitrary string of length $n > 0$. Assume inductive hypothesis holds for all strings w of length $< n$.
- Since $|v| = n > 0$ we have $v = ay$ for some string y with $|y| < n$ and $a \in \Sigma$.
- Then

$$\begin{aligned}(uv)^R &= (u(ay))^R \\ &= ((ua)y)^R \\ &= y^R(ua)^R \\ &= ??\end{aligned}$$

Cannot simplify $(ua)^R$ using inductive hypothesis. Can simplify if we extend base case to include $n = 0$ and $n = 1$. However, $n = 1$ itself requires induction on $|u|$!

Induction on $|u| + |v|$

Theorem

Prove that for any strings $u, v \in \Sigma^$, $(uv)^R = v^R u^R$.*

Proof by induction on $|u| + |v|$ means that we are proving the following.

Induction on $|u| + |v|$

Theorem

Prove that for any strings $u, v \in \Sigma^$, $(uv)^R = v^R u^R$.*

Proof by induction on $|u| + |v|$ means that we are proving the following.

Induction hypothesis:

Induction on $|u| + |v|$

Theorem

Prove that for any strings $u, v \in \Sigma^$, $(uv)^R = v^R u^R$.*

Proof by induction on $|u| + |v|$ means that we are proving the following.

Induction hypothesis: $\forall n \geq 0$, for any $u, v \in \Sigma^*$ with $|u| + |v| \leq n$, $(uv)^R = v^R u^R$.

Induction on $|u| + |v|$

Theorem

Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof by induction on $|u| + |v|$ means that we are proving the following.

Induction hypothesis: $\forall n \geq 0$, for any $u, v \in \Sigma^*$ with $|u| + |v| \leq n$, $(uv)^R = v^R u^R$.

Base case: $n = 0$. Let u, v be an arbitrary strings such that $|u| + |v| = 0$. Implies $u, v = \epsilon$.

Induction on $|u| + |v|$

Theorem

Prove that for any strings $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.

Proof by induction on $|u| + |v|$ means that we are proving the following.

Induction hypothesis: $\forall n \geq 0$, for any $u, v \in \Sigma^*$ with $|u| + |v| \leq n$, $(uv)^R = v^R u^R$.

Base case: $n = 0$. Let u, v be an arbitrary strings such that $|u| + |v| = 0$. Implies $u, v = \epsilon$.

Inductive step: $n > 0$. Let u, v be arbitrary strings such that $|u| + |v| = n$.

Part II

Languages

Definition

A **language** L is a set of strings over Σ . In other words $L \subseteq \Sigma^*$.

Definition

A **language** L is a set of strings over Σ . In other words $L \subseteq \Sigma^*$.

Standard set operations apply to languages.

- For languages A, B the **concatenation** of A, B is $AB = \{xy \mid x \in A, y \in B\}$.
- For languages A, B , their **union** is $A \cup B$, **intersection** is $A \cap B$, and **difference** is $A \setminus B$ (also written as $A - B$).
- For language $A \subseteq \Sigma^*$ the **complement** of A is $\bar{A} = \Sigma^* \setminus A$.

Exponentiation, Kleene star etc

Definition

For a language $L \subseteq \Sigma^*$ and $n \in \mathbb{N}$, define L^n inductively as follows.

$$L^n = \begin{cases} \{\epsilon\} & \text{if } n = 0 \\ L \cdot (L^{n-1}) & \text{if } n > 0 \end{cases}$$

And define $L^* = \bigcup_{n \geq 0} L^n$, and $L^+ = \bigcup_{n \geq 1} L^n$

Exercise

Problem

Answer the following questions taking $A, B \subseteq \{0, 1\}^*$.

- 1 Is $\epsilon = \{\epsilon\}$? Is $\emptyset = \{\epsilon\}$?
- 2 What is $\emptyset \cdot A$? What is $A \cdot \emptyset$? \emptyset
- 3 What is $\{\epsilon\} \cdot A$? And $A \cdot \{\epsilon\}$? A
- 4 If $|A| = 2$ and $|B| = 3$, what is $|A \cdot B|$? 6

Exercise

Problem

Consider languages over $\Sigma = \{0, 1\}$.

- 1 What is \emptyset^0 ? $\{\epsilon\}$
- 2 If $|L| = 2$, then what is $|L^4|$?
- 3 What is \emptyset^* , $\{\epsilon\}^*$, ϵ^* ?
- 4 For what L is L^* finite?
- 5 What is \emptyset^+ , $\{\epsilon\}^+$, ϵ^+ ?

Languages and Computation

What are we interested in computing? Mostly functions.

Informal definition: An algorithm \mathcal{A} computes a function $f : \Sigma^* \rightarrow \Sigma^*$ if for all $w \in \Sigma^*$ the algorithm \mathcal{A} on input w terminates in a finite number of steps and outputs $f(w)$.

Examples of functions:

- Numerical functions: length, addition, multiplication, division etc
- Given graph G and s, t find shortest paths from s to t
- Given program M check if M halts on empty input
- Posts Correspondence problem

Definition

A function f over Σ^* is a boolean if $f : \Sigma^* \rightarrow \{0, 1\}$.

Definition

A function f over Σ^* is a boolean if $f : \Sigma^* \rightarrow \{0, 1\}$.

Observation: There is a bijection between boolean functions and languages.

- Given boolean function $f : \Sigma^* \rightarrow \{0, 1\}$ define language $L_f = \{w \in \Sigma^* \mid f(w) = 1\}$

Definition

A function f over Σ^* is a boolean if $f : \Sigma^* \rightarrow \{0, 1\}$.

Observation: There is a bijection between boolean functions and languages.

- Given boolean function $f : \Sigma^* \rightarrow \{0, 1\}$ define language $L_f = \{w \in \Sigma^* \mid f(w) = 1\}$
- Given language $L \subseteq \Sigma^*$ define boolean function $f : \Sigma^* \rightarrow \{0, 1\}$ as follows: $f(w) = 1$ if $w \in L$ and $f(w) = 0$ otherwise.

Language recognition problem

Definition

For a language $L \subseteq \Sigma^*$ the language recognition problem associated with L is the following: given $w \in \Sigma^*$, is $w \in L$?

Language recognition problem

Definition

For a language $L \subseteq \Sigma^*$ the language recognition problem associated with L is the following: given $w \in \Sigma^*$, is $w \in L$?

- Equivalent to the problem of “computing” the function f_L .
- Language recognition is same as boolean function computation
- How difficult is a function f to compute? How difficult is the recognizing L_f ?

Language recognition problem

Definition

For a language $L \subseteq \Sigma^*$ the language recognition problem associated with L is the following: given $w \in \Sigma^*$, is $w \in L$?

- Equivalent to the problem of “computing” the function f_L .
- Language recognition is same as boolean function computation
- How difficult is a function f to compute? How difficult is the recognizing L_f ?

Why two different views? Helpful in understanding different aspects?

How many languages are there?

Recall:

Definition

An set A is **countably infinite** if there is a bijection f between the natural numbers and A .

Theorem

Σ^* is countably infinite for every finite Σ .

The set of all languages is $\mathbb{P}(\Sigma^*)$ the power set of Σ^*

How many languages are there?

Recall:

Definition

An set A is **countably infinite** if there is a bijection f between the natural numbers and A .

Theorem

Σ^* is countably infinite for every finite Σ .

The set of all languages is $\mathbb{P}(\Sigma^*)$ the power set of Σ^*

Theorem (Cantor)

$\mathbb{P}(\Sigma^*)$ is **not** countably infinite for any finite Σ .

Cantor's diagonalization argument

Theorem (Cantor)

$\mathbb{P}(\mathbb{N})$ is not countably infinite.

- Suppose $\mathbb{P}(\mathbb{N})$ is countable infinite. Let S_1, S_2, \dots , be an enumeration of all subsets of numbers.
- Let D be the following diagonal subset of numbers.

$$D = \{i \mid i \notin S_i\}$$

1 \ 1 \notin S_1

- Since D is a set of numbers, by assumption, $D = S_j$ for some j .
- **Question:** Is $j \in D$? *$\Rightarrow j \notin S_j \Rightarrow j \notin D$*
 $j \notin D \Rightarrow j \in S_j \Rightarrow j \in D$

Consequences for Computation

- How many \mathcal{C} programs are there? The set of \mathcal{C} programs is countably infinite since each of them can be represented as a string over a finite alphabet.
- How many languages are there? Uncountably many!
- Hence some (in fact almost all!) languages/boolean functions do not have any \mathcal{C} program to recognize them.

Questions:

Consequences for Computation

- How many C programs are there? The set of C programs is countably infinite since each of them can be represented as a string over a finite alphabet.
- How many languages are there? Uncountably many!
- Hence some (in fact almost all!) languages/boolean functions do not have any C program to recognize them.

Questions:

- Maybe interesting languages/functions have C programs and hence computable. Only uninteresting languages uncomputable?
- Why should C programs be the definition of computability?
- Ok, there are difficult problems/languages. what languages are computable and which have efficient algorithms?