Strings and Languages

Lecture 1

Tuesday, August 25, 2020

LaTeXed: September 1, 2020 21:18

1.1
Strings

Alphabet

An alphabet is a **finite** set of symbols.

Examples of alphabets:

- $\Sigma = \{0, 1\}$,
- $\Sigma = \{a, b, c, \ldots, z\}$,
- ASCII.
- UTF8.
- $\Sigma = \{\langle \mathbf{moveforward} \rangle, \langle \mathbf{moveback} \rangle\}$

Alphabet

An alphabet is a **finite** set of symbols.
Examples of alphabets:

- $\Sigma = \{0, 1\}$,
- $\Sigma = \{a, b, c, \ldots, z\}$,
- ASCII.
- UTF8.
- $\Sigma = \{\langle \mathbf{moveforward} \rangle, \ \langle \mathbf{moveback} \rangle\}$

String Definitions

Definition

1. A string/word over Σ is a **finite sequence** of symbols over Σ. For example, '0101001', '*string*', '$\langle\mathbf{moveback}\rangle\langle\mathbf{rotate90}\rangle$'
2. ϵ is the empty string.
3. The length of a string w (denoted by $|w|$) is the number of symbols in w. For example, $|101| = 3$, $|\epsilon| = 0$
4. 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.

Inductive/recursive definition of strings

Formal definition of a **string**:

- 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

- $a, b, c, \ldots$ denote elements of Σ
- $w, x, y, z, \ldots$ denote strings
- $A, B, C, \ldots$ 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.
- **concatenation** defined recursively :
  - $xy = y$ if $x = \epsilon$
  - $xy = a(wy)$ if $x = aw$
- xy sometimes written as $x \bullet y$.
- concatenation is **associative**: $(uv)w = u(vw)$
  hence write $uvw \equiv (uv)w = u(vw)$
- **not** commutative: uv not necessarily equal to vu
- The identity element is the empty string ϵ:

$$\epsilon u = u\epsilon = u.$$

Concatenation and properties

- If x and y are strings then xy denotes their concatenation.
- **concatenation** defined recursively :
  - $xy = y$ if $x = \epsilon$
  - $xy = a(wy)$ if $x = aw$
- xy sometimes written as $x \bullet y$.
- concatenation is **associative**: $(uv)w = u(vw)$
  hence write $uvw \equiv (uv)w = u(vw)$
- **not** commutative: uv not necessarily equal to vu
- The identity element is the empty string ϵ:

$$\epsilon u = u\epsilon = u.$$

Concatenation and properties

- If x and y are strings then xy denotes their concatenation.
- **concatenation** defined recursively :
  - $xy = y$ if $x = \epsilon$
  - $xy = a(wy)$ if $x = aw$
- xy sometimes written as $x \bullet y$.
- concatenation is **associative**: $(uv)w = u(vw)$
  hence write $uvw \equiv (uv)w = u(vw)$
- **not** commutative: uv not necessarily equal to vu
- The identity element is the empty string ϵ:

$$\epsilon u = u\epsilon = u.$$

Concatenation and properties

- If x and y are strings then xy denotes their concatenation.
- concatenation defined recursively :
  - $xy = y$ if $x = \epsilon$
  - $xy = a(wy)$ if $x = aw$
- xy sometimes written as $x \bullet y$.
- concatenation is associative: $(uv)w = u(vw)$ hence write $uvw \equiv (uv)w = u(vw)$
- **not** commutative: uv not necessarily equal to vu
- The identity element is the empty string ϵ:

$$\epsilon u = u\epsilon = u.$$

Concatenation and properties

- If x and y are strings then xy denotes their concatenation.
- concatenation defined recursively :
  - $xy = y$ if $x = \epsilon$
  - $xy = a(wy)$ if $x = aw$
- xy sometimes written as $x \bullet y$.
- concatenation is associative: $(uv)w = u(vw)$
  hence write $uvw \equiv (uv)w = u(vw)$
- **not** commutative: uv not necessarily equal to vu
- The identity element is the empty string ϵ:

$$\epsilon u = u\epsilon = u.$$

Substrings, prefix, suffix

Definition

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

String exponents

Definition

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: $(blah)^4 = blahblahblahblah$.

Set Concatenation

Definition

Given two sets X and Y of strings (over some common alphabet Σ) the **concatenation** of X and Y is

$$XY = \{xy \mid x \in X, y \in Y\}$$

Set Concatenation

Definition

Given two sets X and Y of strings (over some common alphabet Σ) the **concatenation** of X and Y is

$$XY = \{xy \mid x \in X, y \in Y\}$$

Example

$X = \{fido, rover, spot\}$,
$Y = \{fluffy, tabby\}$
$\Longrightarrow$
$XY = \{fidofluffy, fidotabby, roverfluffy, \ldots\}$.

Σ^* and languages

Definition

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

Definition

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

Σ^* and languages

Definition

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

THE END

...

(for now)

1.1.1
Exercise solved in detail

Exercise

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

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

Exercise

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

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

Exercise

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

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

Exercise

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

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

Exercise

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

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

Exercise

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

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

Exercise

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

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

Exercise

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

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

Exercise

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

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

THE END

...

(for now)

1.2
Countable sets, countably infinite sets, and languages

Countable sets

Definition

A set X is countable, if its elements can be counted.
There exists an injective mapping from X to natural numbers $\mathsf{N} = \{1, 2, 3, \ldots\}$.

Example

All finite sets are countable: $\{aba, ima, saba, safta, uma, upa\}$.

Example

$\mathbb{N} \times \mathbb{N} = \{(i, j) \mid i, j \in \mathbb{N}\}$ is countable.

: Proof: $f(i, j) = 2^i 3^j$.

Countable sets

Definition

A set X is countable, if its elements can be counted.
There exists an injective mapping from X to natural numbers $\mathsf{N} = \{1, 2, 3, \ldots\}$.

Example

All finite sets are countable: $\{aba, ima, saba, safta, uma, upa\}$.

Example

$\mathbb{N} \times \mathbb{N} = \{(i, j) \mid i, j \in \mathbb{N}\}$ is countable.

: Proof: $f(i, j) = 2^i 3^j$.

Countable sets

Definition

A set X is countable, if its elements can be counted.
There exists an injective mapping from X to natural numbers $\mathsf{N} = \{1, 2, 3, \ldots\}$.

Example

All finite sets are countable: $\{aba, ima, saba, safta, uma, upa\}$.

Example

$\mathbb{N} \times \mathbb{N} = \{(i, j) \mid i, j \in \mathbb{N}\}$ is countable.

: Proof: $f(i, j) = 2^i 3^j$.

Countable sets

Definition

A set X is countable, if its elements can be counted.
There exists an injective mapping from X to natural numbers $\mathrm{N} = \{1, 2, 3, \ldots\}$.

Example

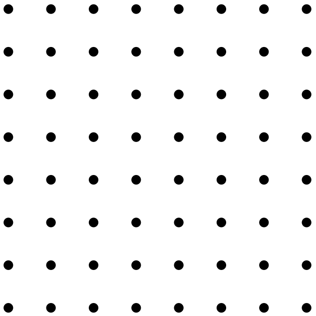
All finite sets are countable: $\{aba, ima, saba, safta, uma, upa\}$.

Example

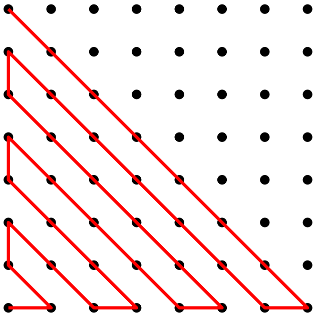
$\mathbb{N} \times \mathbb{N} = \{(i, j) \mid i, j \in \mathbb{N}\}$ is countable.

: Proof: $f(i, j) = 2^i 3^j$.

$\mathbb{N} \times \mathbb{N}$ is countable



$\mathbb{N} \times \mathbb{N}$ is countable



Canonical order and countability of strings

Definition

A set X is countably infinite (countable and infinite) if there is a bijection f between the natural numbers and X.

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

The set of all strings is countable

Theorem

Σ^* is countable for any 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: $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}$.
$\{a, b, c\}^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, \ldots\}$

The set of all strings is countable

Theorem

Σ^* *is countable for any 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: $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}$.
$\{a,b,c\}^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, \ldots\}$

The set of all strings is countable

Theorem

Σ^* *is countable for any 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: $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}$.

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

The set of all strings is countable

Theorem

Σ^* *is countable for any 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: $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}$.
$\{a, b, c\}^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, \ldots\}$

Exercise I

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

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

Exercise I

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

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

Exercise II

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

1. Is a finite set countable?
2. X is countable, and the set $Y \subseteq X$, then is the set Y countable?
3. If X and Y are countable, is $X \setminus Y$ countable?
4. Are all infinite sets countably infinite?
5. If X_i is a countable infinite set, for $i = 1, \ldots, 700$, is $\cup_i X_i$ countable infinite?
6. If X_i is a countable infinite set, for $i = 1, \ldots,$, is $\cup_i X_i$ countable infinite?
7. Let X be a countable infinite set, and consider its power set

$$2^X = \{Y \mid Y \subseteq x\}.$$

The statement "the set 2^X is countable" is correct?

THE END

...

(for now)

1.3
Inductive proofs on strings

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

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

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 $n \leq k$.
- **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 $\mathbf{u}, \mathbf{v} \in \Sigma^*$, $(\mathbf{u}\mathbf{v})^R = \mathbf{v}^R \mathbf{u}^R$.

Proof: by induction.
On what?? $|\mathbf{u}\mathbf{v}| = |\mathbf{u}| + |\mathbf{v}|$?
$|\mathbf{u}|$?
$|\mathbf{v}|$?

What does it mean "induction on $|\mathbf{u}|$"?

1.3.1: Three proofs by induction

1.3.1.1: 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

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

Induction hypothesis: $\forall n \geq 0$, for any string u of length n:

For all strings $v \in \Sigma^*$, $(uv)^R = v^R u^R$.

No assumption about v, hence statement holds for all $v \in \Sigma^*$.

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 $v \in \Sigma^*$, $(uv)^R = v^R u^R$.

No assumption about v, hence statement holds for all $v \in \Sigma^*$.

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 $v \in \Sigma^*$, $(uv)^R = v^R u^R$.

No assumption about v, hence 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

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

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

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

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

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

1.3.1.2: A failed attempt: Induction on $|v|$

Induction on $|v|$

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

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

Induction on $|v|$

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

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

Induction on $|v|$

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

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

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 $|u|$!

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 $|u|$!

1.3.1.3: Induction on $|\boldsymbol{u}| + |\boldsymbol{v}|$

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 $|u| + |v| = n$.

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 $|u| + |v| = n$.

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 $|u| + |v| = n$.

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 $|u| + |v| = n$.

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 $|u| + |v| = n$.

THE END

...

(for now)

1.4
Languages

Languages

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 $\bar{A} = \Sigma^* \setminus A$.

Languages

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.

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

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

Exercise

Problem

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

1. *Is* $\epsilon = \{\epsilon\}$*? Is* $\emptyset = \{\epsilon\}$*?*
2. *What is* $\emptyset \bullet A$*? What is* $A \bullet \emptyset$*?*
3. *What is* $\{\epsilon\} \bullet A$*? And* $A \bullet \{\epsilon\}$*?*
4. *If* $|A| = 2$ *and* $|B| = 3$*, what is* $|A \bullet B|$*?*

Exercise

Problem

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

1. *What is* $\emptyset^0$*?*
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 $f : \Sigma^* \to \Sigma^*$ 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

Languages and Computation

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.

Definition

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

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

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

Languages and Computation

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 associate 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?

Language recognition problem

Definition

For a language $L \subseteq \Sigma^*$ the language recognition problem associate 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?

Language recognition problem

Definition

For a language $L \subseteq \Sigma^*$ the language recognition problem associate 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?

The answer my friend is blowing in the slides.

Recall:

Definition

An set X is countable if there is a bijection f between the natural numbers and A.

Theorem

Σ^* is countable 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** countable for any finite Σ.*

How many languages are there?

The answer my friend is blowing in the slides.

Recall:

Definition

An set X is countable if there is a bijection f between the natural numbers and A.

Theorem

Σ^* is countable 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** countable for any finite Σ.*

Cantor's diagonalization argument

Theorem (Cantor)

$\mathbb{P}(\mathbb{N})$ *is not countable.*

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

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

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

Consequences for Computation

- How many C programs are there? The set of C programs is countable 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 languors 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?

Consequences for Computation

- How many C programs are there? The set of C programs is countable 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 languors 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?

Easy languages

Definition

A language $L \subseteq \Sigma^*$ is finite if $|L| = n$ for some integer n.

Exercise: Prove the following.

Theorem

The set of all finite languages is countable.

THE END

...

(for now)

1.5

Overview of whats coming on finite automata/complexity

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
   1. Regular expressions.
   2. DFA: Deterministic finite automata.
   3. NFA: Non-deterministic finite automata.
   4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
   1. Regular expressions.
   2. DFA: Deterministic finite automata.
   3. NFA: Non-deterministic finite automata.
   4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
    1. Regular expressions.
    2. DFA: Deterministic finite automata.
    3. NFA: Non-deterministic finite automata.
    4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
    1. Regular expressions.
    2. DFA: Deterministic finite automata.
    3. NFA: Non-deterministic finite automata.
    4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
   1. Regular expressions.
   2. DFA: Deterministic finite automata.
   3. NFA: Non-deterministic finite automata.
   4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
    1. Regular expressions.
    2. DFA: Deterministic finite automata.
    3. NFA: Non-deterministic finite automata.
    4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
    1. Regular expressions.
    2. DFA: Deterministic finite automata.
    3. NFA: Non-deterministic finite automata.
    4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
    1. Regular expressions.
    2. DFA: Deterministic finite automata.
    3. NFA: Non-deterministic finite automata.
    4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
   1. Regular expressions.
   2. DFA: Deterministic finite automata.
   3. NFA: Non-deterministic finite automata.
   4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

Languages: easiest, easy, hard, really hard, really really hard

1. Finite languages.
2. Regular languages.
    1. Regular expressions.
    2. DFA: Deterministic finite automata.
    3. NFA: Non-deterministic finite automata.
    4. Languages that are not regular.
3. Context free languages (stack).
4. Turing machines: Decidable languages.
5. TM Undecidable languages (halting theorem).
6. TM Unrecognizable languages.

THE END

...

(for now)