Part II

Languages
A language \( L \) is a set of strings over \( \Sigma \). In other words \( L \subseteq \Sigma^* \).
Definition

A language $L$ is a set of strings over $\Sigma$. 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 $\bar{A} = \Sigma^* \setminus A$. 

Nikita Borisov (UIUC) CS/ECE 374 Fall 2019 23 / 33
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 $L^* = \bigcup_{n\geq0} L^n$, and $L^+ = \bigcup_{n\geq1} L^n$
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 $A \cdot \emptyset$? 
3. What is $\{\epsilon\} \cdot A$? And $A \cdot \{\epsilon\}$? 
4. If $|A| = 2$ and $|B| = 3$, what is $|A \cdot B|$?
Consider languages over $\Sigma = \{0, 1\}$.

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

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

A function \( f \) over \( \Sigma^* \) 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 \( L_f = \{ w \in \Sigma^* \mid f(w) = 1 \} \).
A function $f$ over $\Sigma^*$ 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 $w \in L$?
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 $L_f$?
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?

Recall:

**Definition**

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

**Theorem**

$\Sigma^*$ is **countably infinite** for every finite $\Sigma$.

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

$\Sigma^*$ is countably infinite for every finite $\Sigma$.

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

**Theorem (Cantor)**

$\mathbb{P}(\Sigma^*)$ is **not** countably infinite for any finite $\Sigma$. 
Cantor’s diagonalization argument

Theorem (Cantor)

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

- Suppose \( \mathcal{P}(\mathbb{N}) \) 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 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:
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 langues 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 countably infinite.
Regular Languages and Expressions

Lecture 2
August 29, 2019
Part I

Regular Languages
A class of simple but very useful languages. The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

- $\emptyset$ is a regular language
Regular Languages

A class of simple but very useful languages. The set of regular languages over some alphabet \( \Sigma \) is defined inductively as:

- \( \emptyset \) is a regular language
- \( \{\epsilon\} \) is a regular language

Regular languages are closed under the operations of union, concatenation and Kleene star.
A class of simple but very useful languages. The set of *regular languages* over some alphabet $\Sigma$ is defined inductively as:

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
Regular Languages

A class of simple but very useful languages. The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular

Regular languages are closed under the operations of union, concatenation and Kleene star.
Regular Languages

A class of simple but very useful languages. The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
- If $L_1, L_2$ are regular then $L_1L_2$ is regular

Regular languages are closed under the operations of union, concatenation and Kleene star.
Regular Languages

A class of simple but very useful languages. The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
- If $L_1, L_2$ are regular then $L_1L_2$ is regular
- If $L$ is regular, then $L^* = \cup_{n \geq 0} L^n$ is regular
Regular Languages

A class of simple but very useful languages. The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

- $\emptyset$ is a regular language
- $\{\epsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
- If $L_1, L_2$ are regular then $L_1L_2$ is regular
- If $L$ is regular, then $L^* = \cup_{n \geq 0} L^n$ is regular

Regular languages are closed under the operations of union, concatenation and Kleene star.
Regular Languages

A class of simple but very useful languages. The set of regular languages over some alphabet $\Sigma$ is defined inductively as:

- $\emptyset$ is a regular language
- $\{\varepsilon\}$ is a regular language
- $\{a\}$ is a regular language for each $a \in \Sigma$; here we are interpreting $a$ as a string of length 1
- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular
- If $L_1, L_2$ are regular then $L_1 L_2$ is regular
- If $L$ is regular, then $L^* = \bigcup_{n \geq 0} L^n$ is regular

Regular languages are closed under the operations of union, concatenation and Kleene star.
Some simple regular languages

Lemma

If \( w \) is a string then \( L = \{ w \} \) is regular.

Example: \( \{ aba \} \) or \( \{ abbabbab \} \). Why?
## Lemma

If $w$ is a string then $L = \{w\}$ is regular.

### Example:

$\{aba\}$ or $\{abbabbab\}$. Why?

## Lemma

Every finite language $L$ is regular.

### Examples:

$L = \{a, abaab, aba\}$. $L = \{w \mid |w| \leq 100\}$. Why?
More Examples

- \{w \mid w\ is\ a\ keyword\ in\ Python\ program\}\}
- \{w \mid w\ is\ a\ valid\ date\ of\ the\ form\ mm/dd/yy\}\}
- \{w \mid w\ describes\ a\ valid\ Roman\ numeral\}\}
  \{I, II, III, IV, V, VI, VII, VIII, IX, X, XI, \ldots\}\}
- \{w \mid w\ contains\ "CS374"\ as\ a\ substring\}\}.
Part II

Regular Expressions
WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!

BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!

IT'S HOPELESS!

EVERYBODY STAND BACK.

I KNOW REGULAR EXPRESSIONS.

https://xkcd.com/208/
Regular Expressions

A way to denote regular languages

- simple patterns to describe related strings
- useful in
  - text search (editors, Unix/grep, emacs)
  - compilers: lexical analysis
  - compact way to represent interesting/useful languages
  - dates back to 50’s: Stephen Kleene
    who has a star named after him.
Inductive Definition

A regular expression $r$ over an alphabet $\Sigma$ is one of the following:

**Base cases:**
- $\emptyset$ denotes the language $\emptyset$
- $\epsilon$ denotes the language $\{\epsilon\}$.
- $a$ denote the language $\{a\}$.
Inductive Definition

A regular expression \( r \) over an alphabet \( \Sigma \) is one of the following:

**Base cases:**
- \( \emptyset \) denotes the language \( \emptyset \)
- \( \epsilon \) denotes the language \( \{ \epsilon \} \).
- \( a \) denotes the language \( \{ a \} \).

**Inductive cases:** If \( r_1 \) and \( r_2 \) are regular expressions denoting languages \( R_1 \) and \( R_2 \) respectively then,
- \( (r_1 + r_2) \) denotes the language \( R_1 \cup R_2 \)
- \( r_1r_2 \) denotes the language \( R_1R_2 \)
- \( (r_1)^* \) denotes the language \( R_1^* \)
### Regular Languages vs Regular Expressions

<table>
<thead>
<tr>
<th>Regular Languages</th>
<th>Regular Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset ) regular</td>
<td>( \emptyset ) denotes ( \emptyset )</td>
</tr>
<tr>
<td>( { \epsilon } ) regular</td>
<td>( \epsilon ) denotes ( { \epsilon } )</td>
</tr>
<tr>
<td>( { a } ) regular for ( a \in \Sigma )</td>
<td>( a ) denote ( { a } )</td>
</tr>
<tr>
<td>( R_1 \cup R_2 ) regular if both are</td>
<td>( r_1 + r_2 ) denotes ( R_1 \cup R_2 )</td>
</tr>
<tr>
<td>( R_1R_2 ) regular if both are</td>
<td>( r_1r_2 ) denotes ( R_1R_2 )</td>
</tr>
<tr>
<td>( R^* ) is regular if ( R ) is</td>
<td>( r^* ) denote ( R^* )</td>
</tr>
</tbody>
</table>

Regular expressions denote regular languages — they explicitly show the operations that were used to form the language.
For a regular expression \( r \), \( L(r) \) is the language denoted by \( r \). Multiple regular expressions can denote the same language!

**Example:** \((0 + 1)\) and \((1 + 0)\) denote the same language \(\{0, 1\}\)
For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language! Example: $(0 + 1)$ and $(1 + 0)$ denote same language $\{0, 1\}$.

Two regular expressions $r_1$ and $r_2$ are equivalent if $L(r_1) = L(r_2)$. 

Omit parenthesis by adopting precedence order: $\star, \text{concat} , +$. Example: $rs \star + t = (r \cdot s) \star + t$.

Omit parenthesis by associativity of each of these operations. Example: $rst = (rs) t = r \cdot (st)$, $r + s + t = r + (s + t) = (r + s) + t$.

Superscript $\star$. For convenience, define $r^\star = r r^\star$. Hence if $L(r) = R$ then $L(r^\star) = R^\star$.

Other notation: $r + s$, $r [ s ]$, $r | s$ all denote union. $rs$ is sometimes written as $r \cdot s$. 

Nikita Borisov (UIUC)
For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!

**Example:** $(0 + 1)$ and $(1 + 0)$ denote same language $\{0, 1\}$

Two regular expressions $r_1$ and $r_2$ are **equivalent** if $L(r_1) = L(r_2)$.

Omit parenthesis by adopting precedence order: $\ast$, concat, $\; +$. 

**Example:** $rs^* + t = (r(s^*)) + t$
For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!

**Example:** $(0 + 1)$ and $(1 + 0)$ denote same language $\{0, 1\}$

Two regular expressions $r_1$ and $r_2$ are **equivalent** if $L(r_1) = L(r_2)$.

Omit parenthesis by adopting precedence order: $\ast$, concat, $\oplus$.

**Example:** $rs^* + t = (r(s^*)) + t$

Omit parenthesis by associativity of each of these operations.

**Example:** $rst = (rs)t = r(st)$,

$r + s + t = r + (s + t) = (r + s) + t$. 
Notation and Parenthesis

For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!

**Example:** $(0 + 1)$ and $(1 + 0)$ denote the same language $\{0, 1\}$

Two regular expressions $r_1$ and $r_2$ are equivalent if $L(r_1) = L(r_2)$.

Omit parenthesis by adopting precedence order: $\ast$, concat, $\plus$.

**Example:** $rs^* + t = (r(s^*)) + t$

Omit parenthesis by associativity of each of these operations.

**Example:** $rst = (rs)t = r(st)$,
\[ r + s + t = r + (s + t) = (r + s) + t. \]

Superscript $\plus$. For convenience, define $r^+ = rr^*$. Hence if $L(r) = R$ then $L(r^+) = R^+$. 
Notation and Parenthesis

- For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!
  
  **Example:** $(0 + 1)$ and $(1 + 0)$ denote same language $\{0, 1\}$

- Two regular expressions $r_1$ and $r_2$ are equivalent if $L(r_1) = L(r_2)$.

- Omit parenthesis by adopting precedence order: $\ast$, concat, $\oplus$.
  
  **Example:** $rs^* + t = (r(s^*)) + t$

- Omit parenthesis by associativity of each of these operations.
  
  **Example:** $rst = (rs)t = r(st)$,
  
  $r + s + t = r + (s + t) = (r + s) + t$.

- Superscript $\oplus$. For convenience, define $r^+ = rr^*$. Hence if $L(r) = R$ then $L(r^+) = R^+$.

- Other notation: $r + s$, $r \cup s$, $r|s$ all denote union. $rs$ is sometimes written as $r \cdot s$. 
Given a language $L$ “in mind” (say an English description) we would like to write a regular expression for $L$ (if possible)
Given a language $L$ “in mind” (say an English description) we would like to write a regular expression for $L$ (if possible).

Given a regular expression $r$ we would like to “understand” $L(r)$ (say by giving an English description).
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
Understanding regular expressions

- $(0 + 1)^*$: set of all strings over \{0, 1\}
- $(0 + 1)^*001(0 + 1)^*$:
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
Understanding regular expressions

- $(0 + 1)^*$: set of all strings over $\{0, 1\}$
- $(0 + 1)^*001(0 + 1)^*$: strings with $001$ as substring
- $0^* + (0^*10^*10^*10^*)^*$:
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
- \(0^* + (0*10*10*10*)^*\): strings with number of 1’s divisible by 3
Understanding regular expressions

- $(0 + 1)^*$: set of all strings over $\{0, 1\}$
- $(0 + 1)^*001(0 + 1)^*$: strings with $001$ as substring
- $0^* + (0^*10^*10^*10^*)^*$: strings with number of $1$’s divisible by $3$
- $\emptyset$:
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
- \(0^* + (0^*10^*10^*10^*)^*\): strings with number of 1’s divisible by 3
- \(\emptyset\): \(\{\}\)
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
- \(0^* + (0*10*10*10^*)^*\): strings with number of 1’s divisible by 3
- \(\emptyset 0\): \(\{\}\)
- \((\epsilon + 1)(01)^*(\epsilon + 0)\):
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
- \(0^* + (0^*10^*10^*10^*)^*\): strings with number of 1’s divisible by 3
- \(\emptyset^0\): \(\{\}\)
- \((\epsilon + 1)(01)^*(\epsilon + 0)\): alternating 0s and 1s. Alternatively, no two consecutive 0s and no two consecutive 1s
Understanding regular expressions

- \((0 + 1)^*\): set of all strings over \(\{0, 1\}\)
- \((0 + 1)^*001(0 + 1)^*\): strings with 001 as substring
- \(0^* + (0^*10^*10^*10^*)^*\): strings with number of 1’s divisible by 3
- \(\emptyset 0\): \(\{\}\)
- \((\epsilon + 1)(01)^*(\epsilon + 0)\): alternating 0s and 1s. Alternatively, no two consecutive 0s and no two consecutive 1s
- \((\epsilon + 0)(1 + 10)^*\):