Languages and Regular expressions

Lecture 3
Alphabet, Strings, and Languages

• An **alphabet** $\Sigma = \{a, b, c\}$ is a finite set of **letters/symbols**.

• A string over an **alphabet** $\Sigma$ is a finite sequence of symbols, e.g.
  - sequences $cab, baa$, and $aaa$ are some strings over $\Sigma = \{a, b, c\}$
  - sequences $\varepsilon, 0, 1, 00$, and $01$ are some strings over $\Sigma = \{0, 1\}$

• $\Sigma^*$ is the **set of all strings** over $\Sigma$, e.g. $aabbbaa \in \Sigma^*$,

• Naturally, A **language** $L$ is a collection/set of strings over some alphabet, i.e. $L \subseteq \Sigma^*$ e.g.,
  - $L_{\text{even}} = \{w \in \Sigma^* : w$ is of even length$\}$
  - $L_{\{a^n b^n\}} = \{w \in \Sigma^* : w$ is of the form $a^n b^n$ for $n \geq 0$}$
Sets of strings: $\Sigma^n$, $\Sigma^*$, and $\Sigma^+$

- $\Sigma^n$ is the set of all strings over $\Sigma$ of length exactly $n$. Defined inductively as:
  - $\Sigma^0 = \{ \varepsilon \}$
  - $\Sigma^n = \Sigma \Sigma^{n-1}$ if $n > 0$

- $\Sigma^*$ is the set of all finite length strings:
  $$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$$

- $\Sigma^+$ is the set of all nonempty finite length strings:
  $$\Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$$
\( \Sigma^n, \Sigma^*, \text{ and } \Sigma^+ \)

- \( |\Sigma^n| = |\Sigma|^n \)

- \( |\varnothing^n| = ? \)
  - \( \varnothing^0 = \{\varepsilon\} \)
  - \( \varnothing^n = \varnothing \varnothing^{n-1} = \varnothing \) if \( n > 0 \)

- \( |\varnothing^n| = 1 \) if \( n = 0 \)
  - \( |\varnothing^n| = 0 \) if \( n > 0 \)
\( \Sigma^*, \Sigma^+, \) and \( \Sigma^+ \)

- \( |\Sigma^*| = ? \)
  - Infinity. More precisely, \( \aleph_0 \)
- \( |\Sigma^*| = |\Sigma^+| = |N| = \aleph_0 \)

- How long is the longest string in \( \Sigma^* \)?
- How many infinitely long strings in \( \Sigma^* \)?
  - no longest string!
  - none
Languages
Language

• **Definition:** A formal language \( L \) is a set of strings over some finite alphabet \( \Sigma \) or, equivalently, an arbitrary subset of \( \Sigma^* \).

  *Convention*: Italic upper case letters denote languages.

• Examples of languages:
  - the empty set \( \emptyset \)
  - the set \( \{\varepsilon\} \),
  - the set \( \{0,1\}^* \) of all boolean finite length strings.
  - the set of all strings in \( \{0,1\}^* \) with an odd number of 1’s.
  - The set of all python programs that print “Hello World!”

• There are uncountably many languages (but each language has countably many strings)
Much ado about nothing

• $\varepsilon$ is a string containing no symbols. It is not a language.

• $\{\varepsilon\}$ is a language containing one string: the empty string $\varepsilon$. It is not a string.

• $\emptyset$ is the empty language. It contains no strings.
Building Languages

• Languages can be manipulated like any other set.

• Set operations:
  • Union: $L_1 \cup L_2$
  • Intersection, difference, symmetric difference
  • Complement: $L^\neg = \Sigma^* \setminus L = \{ x \in \Sigma^* \mid x \notin L \}$
  • (Specific to sets of strings) concatenation: $L_1 \cdot L_2 = \{ xy \mid x \in L_1, y \in L_2 \}$
Concatenation

• $L_1 \cdot L_2 = L_1L_2 = \{ xy \mid x \in L_1, y \in L_2 \}$ (we omit the bullet often)

  e.g. $L_1 = \{ \text{fido, rover, spot} \}, L_2 = \{ \text{fluffy, tabby} \}$

  then $L_1L_2 = \{ \text{fidofluffy, fidotabby, roverfluffy, ...} \}$

$L_1 = \{ a, aa \}, L_2 = \{ \varepsilon \}$

$L_1L_2 = L_1$

$L_1 \cdot L_2 = L_1L_2 = \{ \varepsilon \}$

$L_1L_2 = \emptyset$
Building Languages

• $L^n$ inductively defined: $L^0 = \{ \varepsilon \}$, $L^n = LL^{n-1}$

Kleene Closure (star) $L^*$

Definition 1: $L^* = \bigcup_{n \geq 0} L^n$, the set of all strings obtained by concatenating a sequence of zero or more stings from $L$
Building Languages

- $L^n$ inductively defined: $L^0 = \{ \varepsilon \}$, $L^n = LL^{n-1}$
  
  **Kleene Closure (star) $L^*$**

  **Recursive Definition: $L^*$ is the set of strings $w$ such that either**
  
  - $w = \varepsilon$ or
  
  - $w = xy$ for $x$ in $L$ and $y$ in $L^*$
Building Languages

- $\{\varepsilon\}^* = ?$  \(\emptyset^* = ?\) \(\{\varepsilon\}^* = \emptyset^* = \{\varepsilon\}\)

- For any other L, the Kleene closure is infinite and contains arbitrarily long strings. It is the smaller superset of L that is closed under concatenation and contains the empty string.

- **Kleene Plus**

  \[ L^+ = LL^* , \text{ set of all strings obtained by concatenating a sequence of at least one string from } L. \]

  —When is it equal to $L^*$ ?
Regular Languages
Regular Languages

• The set of regular languages over some alphabet $\Sigma$ is defined inductively by:
  • $L$ is empty
  • $L$ contains a single string (could be the empty string)
  • If $L_1, L_2$ are regular, then $L = L_1 \cup L_2$ is regular
  • If $L_1, L_2$ are regular, then $L = L_1 L_2$ is regular
  • If $L$ is regular, then $L^*$ is regular
Regular Languages Examples

- $L$ = any finite set of strings. E.g., $L$ = set of all strings of length at most 10
- $L$ = the set of all strings of 0’s including the empty string

- Intuitively $L$ is regular if it can be constructed from individual strings using any combination of union, concatenation and unbounded repetition.
Regular Languages Examples

• Infinite sets, but of strings with “regular” patterns
  • $\Sigma^*$ (recall: $L^*$ is regular if $L$ is)
  • $\Sigma^+ = \Sigma\Sigma^*$
• All binary integers, starting with 1
  • $L = \{1\}\{0,1\}^*$
• All binary integers which are multiples of 37
  • later
Regular Expressions
Regular Expressions

• A compact notation to describe regular languages
• Omit braces around one-string sets, use + to denote union and juxtapose subexpressions to represent concatenation (without the dot, like we have been doing).
• Useful in
  • text search (editors, Unix/grep)
  • compilers: lexical analysis
Regular Expressions

• In arithmetic, we can use operations $\times$, $+$ to build up expressions such as $(5 + 3) \times 4$
• Similarly, we can use regular operations to build up expressions describing languages, which are called regular expressions.
• E.g $(0 \cup 1)0^*$
• Value of arithmetic expression above is 32.
• Value of a regular expression is a language (which one?)
Inductive Definition
A regular expression $r$ over alphabet $\Sigma$ is one of the following
($L(r)$ is the language it represents):

<table>
<thead>
<tr>
<th>Atomic expressions (Base cases)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$L(\emptyset) = \emptyset$</td>
</tr>
<tr>
<td>$w$ for $w \in \Sigma^*$</td>
<td>$L(w) = {w}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inductively defined expressions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$(r_1 + r_2)$</td>
<td>$L(r_1 + r_2) = L(r_1) \cup L(r_2)$</td>
</tr>
<tr>
<td>$(r_1 r_2)$</td>
<td>$L(r_1 r_2) = L(r_1) L(r_2)$</td>
</tr>
<tr>
<td>$(r^*)$</td>
<td>$L(r^<em>) = L(r)^</em>$</td>
</tr>
</tbody>
</table>

Any regular language has a regular expression and vice versa
Regular Expressions

• Can omit many parentheses
  • By following precedence rules:
    star (*) before concatenation (⋅), before union (+)
    (similar to arithmetic expressions)
      • e.g. $r^*s + t \equiv ((r^*) s) + t$
      • 10* is shorthand for $\{1\} \cdot \{0\}* $ and NOT $\{10\}*$
      • By associativity: $(r+s)+t \equiv r+s+t$, $(rs)t \equiv rst$
  • More short-hand notation
    • e.g., $r^+ \equiv rr^*$ (note: + is in superscript)
Regular Expressions: Examples

• $(0+1)^*$
  • All binary strings

• $((0+1)(0+1))^*$
  • All binary strings of even length

• $(0+1)^*001(0+1)^*$
  • All binary strings containing the substring 001

• $0^* + (0^*10^*10^*10^*)^*$
  • All binary strings with #1s $\equiv 0 \mod 3$

• $(01+1)^*(0+\varepsilon)$
  • All binary strings without two consecutive 0s
Exercise: create regular expressions

• All binary strings with either the pattern 001 or the pattern 100 occurring somewhere
  
  one answer: \((0+1)^*001(0+1)^* + (0+1)^*100(0+1)^*\)

• All binary strings with an even number of 1s

  one answer: \(0^*(10^*10^*)^*\)
Regular Expression Identities

• \( r^*r^* = r^* \)
• \( (r^*)^* = r^* \)
• \( rr^* = r^*r \)
• \( (rs)^*r = r(sr)^* \)
• \( (r+s)^* = (r^*s^*)^* = (r^* + s^*)^* = (r+s^*)^* = \ldots \)
Equivalence

• Two regular expressions are equivalent if they describe the same language. eg.
  • $(0+1)^* = (1+0)^*$ (why?)

• Almost every regular language can be represented by infinitely many distinct but equivalent regular expressions
  • $(L \emptyset)^* L \varepsilon + \emptyset$ = ?
Regular Expression Trees

• Useful to think of a regular expression as a tree. Nice visualization of the recursive nature of regular expressions.

• Formally, a regular expression tree is one of the following:

  • a leaf node labeled $\emptyset$
  • a leaf node labeled with a string
  • a node labeled $+$ with two children, each of which is the root of a regular expression tree
  • a node labeled $\cdot$ with two children, each of which is the root of a regular expression tree
  • a node labeled $*$ with one child, which is the root of a regular expression tree
A regular expression tree for \(0 + 0^*1(10^*1 + 01^*0)^*10^*\)
Not all languages are regular!
Are there Non-Regular Languages?

• Every regular expression over \{0,1\} is itself a string over the 8-symbol alphabet \{0,1,+,*,(,),\varepsilon, \emptyset\}.

• Interpret those symbols as digits 1 through 8. Every regular expression is a base-9 representation of a unique integer.

• Countably infinite!

• We saw (first few slides) there are uncountably many languages over \{0,1\}.

• In fact, the set of all regular expressions over the \{0,1\} alphabet is a non-regular language over the alphabet \{0,1,+,*,(,),\varepsilon, \emptyset\}!!