Strings, Languages, and Regular expressions

Lecture 2
Strings
Definitions for strings

- **alphabet** $\Sigma = \text{finite set of symbols}$
- **string** = finite sequence of symbols of $\Sigma$
- **length** of a string $w$ is denoted $|w|$
- **empty string** is denoted “$\varepsilon$”.

<table>
<thead>
<tr>
<th>$\varepsilon$</th>
<th>$\text{cat}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>\varepsilon</td>
</tr>
</tbody>
</table>

**Variable conventions** (for this lecture)

- $a, b, c, \ldots$ elements of $\Sigma$ (i.e., strings of length 1)
- $w, x, y, z, \ldots$ strings of length 0 or more
- $A, B, C, \ldots$ sets of strings

- e.g., $\Sigma = \{0,1\}$,
  $\Sigma = \{\alpha, \beta, \ldots, \omega\}$, $\Sigma = \text{set of ascii characters}$

- Could formalize as a function $w: [n] \rightarrow \Sigma$ where $|w| = n$
Much ado about nothing

- $\varepsilon$ is a string containing no symbols. It is not a set.
- $\{\varepsilon\}$ is a set containing one string: the empty string $\varepsilon$. It is a set, not a string.
- $\emptyset$ is the empty set. It contains no strings.
Concatenation & its properties

- $xy$ denotes the **concatenation** of strings $x$ and $y$ (sometimes written $x \cdot y$)
- Associative: $(uv)w = u(vw)$ and we write $uvw$.
- Identity element $\varepsilon$: $\varepsilon w = w \varepsilon = w$
- Can be used to define strings (set of all strings $\Sigma^*$) inductively
- NOT commutative: $ab \neq ba$

If $|x| = m$, $|y| = n$

$xy : [m+n] \rightarrow \Sigma$
such that

$xy(i) = x(i)$ if $i \leq m$
$xy(i) = y(i-m)$ else
Substring, Prefix, Suffix, Exponents

• \( \nu \) is a **substring** of \( w \) iff there exist strings \( x, y \), such that \( w = xy \).

  – If \( x = \varepsilon \) (\( w = vy \)) then \( \nu \) is a **prefix** of \( w \).
  
  – If \( y = \varepsilon \) (\( w = xv \)) then \( \nu \) is a **suffix** of \( w \).

• If \( w \) is a string, then \( w^n \) is defined inductively by:

  – \( w^n = \varepsilon \) if \( n = 0 \)
  
  – \( w^n = ww^{n-1} \) if \( n > 0 \)
Set Concatenation

- If $X$ and $Y$ are sets of strings, then

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

e.g. $X = \{\text{fido, rover, spot}\}$, $Y = \{\text{fluffy, tabby}\}$

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

| $XY$ | = 6
---

$A = \{a, aa\}$, $B = \emptyset$

$AB = \emptyset$

$A = \{a, aa\}$, $B = \{\varepsilon, a\}$

$|AB| = 3$
\[ \Sigma^n, \Sigma^*, \text{ and } \Sigma^+ \]

- \( \Sigma^n \) 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:
  \[ \Sigma^+ = \bigcup_{n \geq 1} \Sigma^n \]
\( \Sigma^n, \Sigma^*, \text{ and } \Sigma^+ \)

- \( |\Sigma^n| = |\Sigma|^n \)
- \( |\emptyset^n| = ? \)
  - \( \emptyset^0 = \{ \varepsilon \} \)
  - \( \emptyset^n = \emptyset \emptyset^{n-1} = \emptyset \text{ if } n > 0 \)
- \( |\emptyset^n| = 1 \text{ if } n = 0 \)
- \( |\emptyset^n| = 0 \text{ if } n > 0 \)
\( \Sigma^n, \Sigma^*, \text{ and } \Sigma^+ \)

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

- \( x \) is a string iff \( x=\varepsilon \) or \( x=au \) where \( |u|=|x|-1 \)

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

- How long is the longest string in \( \Sigma^* \)?

- How many infinitely long strings in \( \Sigma^* \)?

no longest string!

This can be the formal definition of a "string"
$\Sigma^*$, $\Sigma^+$, and $\Sigma^n$

- $\Sigma^+$ is the set of all nonempty finite length strings:
  \[ \Sigma^+ = \bigcup_{n \geq 1} \Sigma^n \]

- $\Sigma^+$ is equal to:
  - $\Sigma \Sigma^*$
  - $\Sigma^* \Sigma$
  - $\Sigma \bigcup \Sigma^2 \Sigma^*$
Enumerating Strings

- Canonical (standard) ordering is the lexicographical (dictionary) ordering
  - Order by length (starting with 0)
  - Order the $|\Sigma|^n$ strings of length $n$ by comparing characters left to right

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\varepsilon$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>000</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>001</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>010</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>101</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>110</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>111</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>1001</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>1010</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>1011</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>1100</td>
<td>4</td>
</tr>
</tbody>
</table>
Inductive Definitions

• Often operations on strings are formally defined inductively
  – e.g., $w^n$ in terms of $w^{n-1}$
  – Another example: $w^R$ (w reversed) inducting on length
    • If $|w| = 0$, $w^R = \varepsilon$
    • If $|w| \geq 1$, $w^R = u^Ra$ where $w = au$

  – e.g. $(\text{cat})^R = (\text{c} \cdot \text{at})^R = (\text{at})^R \cdot \text{c} = (\text{a} \cdot \text{t})^R \cdot \text{c}
    = (\text{t})^R \cdot \text{a} \cdot \text{c} = (\text{t} \cdot \varepsilon)^R \cdot \text{ac} = \varepsilon^R \cdot \text{tac} = \text{tac}$
Inductive Proofs

- Inductive proofs follow inductive definitions

**Theorem:** \((uv)^R = v^R u^R\)

**Proof:** By induction

But on what? \(|u|, |v|, |u+v|\), double induction on \(|u|, |v|\)?

\(|u|, (or \ |v|)\) is good enough:

**Base case:** \(|u| = 0\): i.e., \(u = \varepsilon\).

Then: \((uv)^R = v^R\)

\& \(v^R u^R = v^R \varepsilon^R = v^R \varepsilon = v^R \checkmark\)

**Definition of Reversal:**

- base-case

\[\varepsilon^R = \varepsilon\]

\[(au)^R = u^R a\]
Inductive Proofs

• Inductive proofs follow inductive definitions

**Theorem**: \((uv)^R = v^Ru^R\)

**Proof**: By induction

Inductive step: Let \(n > 0\). Assume \((wv)^R = v^Rw^R\) \(\forall w, |w| < n\).

Consider any \(u\) with \(|u| = n\). So \(u = aw, a \in \Sigma, w \in \Sigma^*\).

\[
(\begin{align*}
(\uv)^R &= (awv)^R = (a(wv))^R = (wv)^Ra \\
&= v^Rw^Ra \\
&= v^R(aw)^R \\
&= v^Ru^R
\end{align*})
\]

Definition of Reversal: inductive-case

Inductive Hypothesis: \(|w| < n\)

Definition of Reversal: inductive-case
Languages
Recall

**Problem:**
To compute a function $F$ that maps each input (a string) to an output bit $P(x)$ computes $F$ if for every $x$, $P(x)$ outputs $F(x)$ and halts

Too restrictive?

**Program:**
A finitely described process taking a string as input, and outputting a bit (or not halting)

Enough to compute functions with longer outputs too: $P(x,i)$ outputs the $i^{\text{th}}$ bit of $F(x)$

Enough to model *interactive* computation too: $P^*(x,\text{state})$ outputs $(y, \text{new\_state})$
Language

- A function from $\Sigma^*$ to $\{0,1\}$ can be identified with the set of strings mapped to 1

- A language is a subset of $\Sigma^*$
  - Computational problem for a language: given a string in $\Sigma^*$, decide if it belongs to the language

- Examples of languages: $\emptyset$, $\Sigma^*$, $\Sigma$, $\{\varepsilon\}$, set of strings of odd length, set of strings encoding valid C programs, set of strings encoding valid C programs that halt, …

- There are uncountably many languages (but each language has countably many strings)
Operations on Languages

• Already seen concatenation: $L_1 L_2 = \{ xy \mid x \in L_1, y \in L_2 \}$

• Set operations:
  – Complement: $\overline{L} = \Sigma^* - L = \{ x \in \Sigma^* \mid x \notin L \}$
  – Union: $L_1 \cup L_2$
  – Intersection, difference (can be based on the above two)

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

• $L^* = \bigcup_{n \geq 0} L^n$, and $L^+ = LL^*$

• $\{ \varepsilon \}^* = \_ ? \_$, $\emptyset^* = \_ ? \_ $
Complexity of Languages

• How *computable* is a language?

• Singleton languages
  – $L$ such that $|L| = 1$. Example: $L = \{374\}$
  – An algorithm can have the single string hard-coded into it

• More generally, finite languages
  – Algorithm can have all the strings hard-coded into it

• Many interesting languages are uncomputable

• But many others are neither too easy nor impossible…
Regular Languages
Regular Languages

- The set of regular languages over some alphabet $\Sigma$ is defined inductively by:
  - $\emptyset$ is a regular language
  - $\{\varepsilon\}$ is a regular language
  - $\{a\}$ is a regular language for each $a \in \Sigma$
  - 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 $L^*$ is regular
Regular Languages Examples

- \( L = \{w\} \) where \( w \in \Sigma^* \) is any fixed string
  - e.g., \( L = \{aba\} = \{a\}\{b\}\{a\} \) and \( \{a\} \& \{b\} \) are both regular
  - Proof by induction on \( |w| \), using concatenation for induction

- \( L = \) any finite set of strings
  - e.g., \( L = \) set of all strings of length at most 10
  - Proof by induction on \( |L| \), using union for induction (and the above)
  - Beware: Induction applicable only for \( |L| \in \mathbb{N} \), not \( |L| = \aleph_0 \)
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, without leading 0’s
  • $L = \{1\}{0,1}^* \cup \{0\}$

– All binary integers which are multiples of 37
  • later
Regular Expressions
Regular Expressions

• A short-hand to denote a regular language as strings that match a pattern

• Useful in
  – text search (editors, Unix/grep)
  – compilers: lexical analysis

• Dates back to 50’s: Stephen Kleene, who has a star named after him*

* The star named after him is the Kleene star “*”
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>$\varepsilon$</td>
<td>$L(\varepsilon) = { \varepsilon }$</td>
</tr>
<tr>
<td>$a$ for $a \in \Sigma$</td>
<td>$L(a) = { a }$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inductively defined expressions</th>
<th>$L(r_1+r_2) = L(r_1) \cup L(r_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(r_1r_2)$</td>
<td>$L(r_1r_2) = L(r_1)L(r_2)$</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:
    * before concatenation before +
  
    • e.g. \( r^*s + t \equiv ((r^*) s) + t \)
  
    – 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

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

- $0^* + (0^*10^*10^*10^*)^*$
  - All binary strings with $\#1s \equiv 0 \pmod{3}$

- $(01)^* + (10)^* + 1(01)^* + 0(10)^*$
  - Alternating 0s and 1s. Also, $(1+\varepsilon)(01)^*(0+\varepsilon)$

- $(01+1)^*(0+\varepsilon)$
  - 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: \(0^*(10^*10^*)^*\)
A non-regular language
An inductively defined language

Define $L$ over $\{0,1\}^*$ by:

- $\varepsilon \in L$
- if $w \in L$, then $0w1 \in L$

What do strings in $L$ look like?

Give a characterization of $L$ and prove it correct.

Can you find a regular expression for $L$?

will show impossible!
An inductively defined language

Define $L$ over $\{0,1\}^*$ by:

- $\varepsilon \in L$
- if $w \in L$, then $0w1 \in L$

**Conjecture**: $L = \{ 0^i1^i : i \geq 0 \}$

How can we prove this is correct?

Prove (by induction) that

(a) $L \subseteq \{ 0^i1^i : i \geq 0 \}$
(b) $L \supseteq \{ 0^i1^i : i \geq 0 \}$
\[ L \subseteq \{ \, 0^i1^i : \, i \geq 0 \} \]

Show by induction on \(|w|\), that if \(w \in L\), then \(w\) is of the form \(0^i1^i\).

**Base case:** \(|w| = 0\).

Then \(w = \varepsilon = 0^01^0\)

**Inductive Step:** Let \(n > 0\).

Assume: for all \(k < n\),

any \(w\) in \(L\) with \(|w| = k\), is of form \(0^i1^i\)

Prove: Any \(w\) in \(L\) with \(|w| = n\) is of form \(0^i1^i\)
Inductive step

Consider arbitrary $w \in L$, with $|w| = n$.

Then $w = 0u1$ where $u \in L$ has size $n-2 < n$
(by definition of $L$)

By induction, $u$ is of form $0^i1^i$.

Then $w = 0u1 = 00^i1^i1 = 0^{i+1}1^{i+1}$, the required form
\[ L \supseteq \{ 0^i 1^i : i \geq 0 \} \]

Show by induction on \( n \), that if \( w \) is of the form \( 0^n 1^n \), then \( w \in L \).

Base case: \( n = 0 \).

Then \( w = 0^0 1^0 = \varepsilon \), which is in \( L \) by definition.

Inductive step:

Let \( n > 0 \), and assume for all \( k < n \) that \( 0^k 1^k \in L \).

\[ 0^n 1^n = 00^{n-1} 1^{n-1} 1 = 0u1, \text{ with } u \in L \text{ by induction.} \]

Since \( u \in L \), so is \( 0u1 = 0^n 1^n \) by definition of \( L \).