Consider the problem of a \( n \)-input \( \text{AND} \) function. The input \( (x) \) is a string \( n \)-digits long with \( \Sigma = \{0, 1\} \) and has an output \( (y) \) which is the logical \( \text{AND} \) of all the elements of \( x \).

Formulate a language that describes the above problem.
Consider the problem of a n-input AND function. The input \((x)\) is a string n-digits long with \(\Sigma = \{0, 1\}\) and has an output \((y)\) which is the logical AND of all the elements of \(x\).

Formulate a language that describes the above problem.
Consider the problem of a n-input AND function. The input \( (x) \) is a string n-digits long with \( \Sigma = \{0, 1\} \) and has an output \( (y) \) which is the logical AND of all the elements of \( x \).

Formulate a language that describes the above problem.

This is an example of a regular language which we’ll be discussing today.
Regular Languages
Theorem (Kleene’s Theorem)

A language is regular if and only if it can be obtained from finite languages by applying the three operations:

• Union
• Concatenation
• Repetition

a finite number of times.
Regular Languages

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

**Base Case**

- $\emptyset$ is a regular language.
- $\{\epsilon\}$ is a regular language.
- $\{a\}$ is a regular language for each $a \in \Sigma$. Interpreting $a$ as string of length 1.
Inductive step:

We can build up languages using a few basic operations:

- 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 \( L^* = \bigcup_{n \geq 0} L^n \) is regular. The \( \cdot^* \) operator name is *Kleene star*.
- If \( L \) is regular, then so is \( \overline{L} = \Sigma^* \setminus L \).

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

Have basic operations to build regular languages.

Important: Any language generated by a finite sequence of such operations is regular.

Lemma
Let $L_1, L_2, \ldots$, be regular languages over alphabet $\Sigma$. Then the language $\bigcup_{i=1}^{\infty} L_i$ is not necessarily regular.

Example:
Some simple regular languages

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

Example: $\{aba\}$ or $\{abbabbab\}$. Why?
Some simple regular languages

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?
1. \( L_1 = \{0^i \mid i = 0, 1, \ldots, \infty\} \). The language \( L_1 \) is regular. T/F?
1. $L_1 = \{0^i \mid i = 0, 1, \ldots, \infty\}$. The language $L_1$ is regular. T/F?

2. $L_2 = \{0^{17i} \mid i = 0, 1, \ldots, \infty\}$. The language $L_2$ is regular. T/F?
1. \( L_1 = \{0^i \mid i = 0, 1, \ldots, \infty\} \). The language \( L_1 \) is regular. T/F?

2. \( L_2 = \{0^{17i} \mid i = 0, 1, \ldots, \infty\} \). The language \( L_2 \) is regular. T/F?

3. \( L_3 = \{0^i \mid i \text{ is divisible by 2, 3, or 5}\} \). \( L_3 \) is regular. T/F?
4. $L_4 = \{ w \in \{0, 1\}^* \mid w \text{ has at most } 374 \text{ 1s} \}$. $L_4$ is regular. T/F?
Regular Expressions
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 names after him \(^1\).

**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 $R_1 \cup R_2$
- $(r_1 \cdot r_2) = r_1 \cdot r_2 = (r_1 r_2)$ denotes the language $R_1 R_2$
- $(r_1)^*$ 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>$r_1 \cdot r_2$ denotes $R_1R_2$</td>
</tr>
<tr>
<td>$R^*$ is regular if $R$ is</td>
<td>$r^<em>$ denote $R^</em>$</td>
</tr>
</tbody>
</table>

Regular expressions denote regular languages — they explicitly show the operations that were used to form the language.
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 $\{0, 1\}$
• 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 $L(r_1) = L(r_2)$. 
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: $\ast$, concatenate, $+$. 

  **Example:** $r^*s + t = ((r^*)s) + t$
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 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: $\ast$, concatenate, $\plus$.
  
  Example: $r^\ast s + t = ((r^\ast)s) + t$

• 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 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: $\ast$, concatenate, $\oplus$.

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

• 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 $+$.** For convenience, define $r^+ = rr^*$. Hence if $L(r) = R$ then $L(r^+) = R^+$. 

• 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: \( \ast \), concatenate, \( + \).

**Example:** \( r^*s + t = ((r^*)s) + t \)

• 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** \( + \). For convenience, define \( r^+ = rr^* \). Hence if \( L(r) = R \) then \( L(r^+) = R^+ \).
Some examples of regular expressions
Interpreting regular expressions

1. \((0 + 1)^*:\)
Interpreting regular expressions

1. \((0 + 1)^*:\)
2. \((0 + 1)^*001(0 + 1)^*:\)
Interpreting regular expressions

1. \((0 + 1)^*:\)
2. \((0 + 1)^*001(0 + 1)^*:\)
3. \(0^* + (0^*10^*10^*10^*)^*:\)
Interpreting regular expressions

1. \((0 + 1)^*:\)
2. \((0 + 1)^*001(0 + 1)^*:\)
3. \(0^* + (0*10*10*10*)^*:\)
4. \((\epsilon + 1)(01)^*(\epsilon + 0):\)
1. All strings that end in 1011?
Creating regular expressions

1. All strings that end in 1011?
2. All strings except 11?
Creating regular expressions

1. All strings that end in 1011?
2. All strings except 11?
3. All strings that do not contain 000 as a subsequence?
Creating regular expressions

1. All strings that end in 1011?
2. All strings except 11?
3. All strings that do not contain 000 as a subsequence?
4. All strings that do not contain the substring 10?
Consider the problem of a n-input AND function. The input \((x)\) is a string n-digits long with \(\Sigma = \{0, 1\}\) and has an output \((y)\) which is the logical AND of all the elements of \(x\).

Formulate the regular expression which describes the above language:
Regular expressions in programming
One last expression....
Bit strings with odd number of 0s and 1s

The regular expression is

\[(00)^+11)(01)^+10)((00)^+11)((01)^+10)((00)^+11)\]

\[\times((01)^+10)\]

(Solved using techniques to be presented in the following lectures...)
Bit strings with odd number of 0s and 1s

The regular expression is

\[(00 + 11)^* (01 + 10) \left( 00 + 11 + (01 + 10)(00 + 11)^*(01 + 10) \right)^* \]
Bit strings with odd number of 0s and 1s

The regular expression is

\[(00 + 11)^* (01 + 10) \]

\[\left(00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)\right)^*\]

(Solved using techniques to be presented in the following lectures...)