More Applications of the Pumping Lemma
The Pumping Lemma:

• Given a infinite regular language \( L \)

• there exists an integer \( m \)

• for any string \( w \in L \) with length \( |w| \geq m \)

• we can write \( w = x y z \)

• with \( |x y| \leq m \) and \( |y| \geq 1 \)

• such that: \( x y^i z \in L \quad i = 0, 1, 2, ... \)
Non-regular languages

\[ L = \{ ww^R : w \in \Sigma^* \} \]

Regular languages
Theorem: The language

\[ L = \{ww^R : w \in \Sigma^*\} \quad \Sigma = \{a, b\} \]

is not regular

Proof: Use the Pumping Lemma
Let \( L = \{ww^R : w \in \Sigma^*\} \)

Assume for contradiction that \( L \) is a regular language.

Since \( L \) is infinite, we can apply the Pumping Lemma.
\[ L = \{ww^R : w \in \Sigma^*\} \]

Let \( m \) be the integer in the Pumping Lemma

Pick a string \( w \) such that: \( w \in L \) and

\[ \text{length } |w| \geq m \]

We pick \( w = a^m b^m b^m a^m \)
Write \( a^m b^m b^m a^m = x \ y \ z \)

From the **Pumping Lemma**

it must be that length \( \ | \ x \ y \ | \leq m, \ | \ y \ | \geq 1 \)

Thus:

\[ y = a^k, \quad k \geq 1 \]
\[ x \ y \ z = a^m b^m b^m a^m \] \[ y = a^k, \quad k \geq 1 \]

From the **Pumping Lemma:**

\[ x \ y^i \ z \in L \]

\[ i = 0, 1, 2, \ldots \]

**Thus:**

\[ x \ y^2 \ z \in L \]
From the Pumping Lemma:

\[ x y z = a^m b^m b^m a^m \quad y = a^k, \quad k \geq 1 \]

From the **Pumping Lemma**: \( x y^2 z \in L \)

Thus:

\[ a^{m+k} b^m b^m a^m \in L \]
$a^{m+k} b^m b^m a^m \in L \quad k \geq 1$

**BUT:**

$L = \{ww^R : w \in \Sigma^*\}$

$\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad a^{m+k} b^m b^m a^m \not\in L$

**CONTRADICTION!!!**
Therefore: Our assumption that $L$ is a regular language is not true.

Conclusion: $L$ is not a regular language.
Non-regular languages

\[ L = \{ a^n b^l c^{n+l} : n, l \geq 0 \} \]
**Theorem:** The language

\[ L = \{a^n b^l c^{n+l} : n, l \geq 0\} \]

is not regular

**Proof:** Use the Pumping Lemma
\[ L = \{a^n b^l c^{n+l} : n, l \geq 0\} \]

Assume for contradiction that \( L \) is a regular language.

Since \( L \) is infinite, we can apply the Pumping Lemma.
Let \( m \) be the integer in the Pumping Lemma.

Pick a string \( w \) such that: \( w \in L \) and

\[
|w| \geq m
\]

We pick \( w = a^m b^m c^{2m} \).
Write \( a^m b^m c^{2m} = x y z \)

From the **Pumping Lemma**

it must be that length \( |x y| \leq m, \; |y| \geq 1 \)

Thus:

\[ y = a^k, \quad k \geq 1 \]
From the Pumping Lemma:

\[ x y z = a^m b^m c^{2m} \quad y = a^k, \quad k \geq 1 \]

Thus:

\[ x y^0 z = xz \in L \]
From the Pumping Lemma: $xz \in L$

Thus: $a^{m-k}b^mc^{2m} \in L$
\[ a^{m-k} b^m c^{2m} \in L \quad k \geq 1 \]

**BUT:**

\[ L = \{ a^n b^l c^{n+l} : n, l \geq 0 \} \]

\[ a^{m-k} b^m c^{2m} \notin L \]

**CONTRADICTION!!!**
Therefore: Our assumption that $L$ is a regular language is not true

**Conclusion:** $L$ is not a regular language
Non-regular languages

\[ L = \{ a^{n!} : n \geq 0 \} \]

Regular languages
Theorem: The language \( L = \{ a^{n!} : n \geq 0 \} \) is not regular.

Proof: Use the Pumping Lemma

\[ n! = 1 \cdot 2 \cdots (n-1) \cdot n \]
Assume for contradiction that $L$ is a regular language.

Since $L$ is infinite, we can apply the Pumping Lemma.
Let \( m \) be the integer in the Pumping Lemma

Pick a string \( w \) such that: \( w \in L \)

We pick \( w = a^m! \)
Write \( a^{m!} = x \ y \ z \)

From the **Pumping Lemma**

it must be that length \( |x\ y| \leq m, \ |y| \geq 1 \)

\[
xyz = a^{m!} = a \ldots a a \ldots a a \ldots a a \ldots a
\]

\[
x y z
\]

**Thus:** \( y = a^k, \ 1 \leq k \leq m \)
From the Pumping Lemma:

\[ x \ y \ z = a^{m!} \]

\[ y = a^k, \quad 1 \leq k \leq m \]

Thus:

\[ x \ y^i \ z \in L \]

\[ i = 0, 1, 2, \ldots \]

Thus:

\[ x \ y^2 \ z \in L \]
\[ x \ y \ z = a^{m!} \]
\[ y = a^k, \ 1 \leq k \leq m \]

From the **Pumping Lemma**:
\[ x \ y^2 \ z \in L \]

Thus:
\[ a^{m!+k} \in L \]
Since: \[ L = \{ a^{n!} : n \geq 0 \} \]

There must exist \( p \) such that:

\[ m! + k = p! \]
However:

\[ m! + k \leq m! + m \quad \text{for} \quad m > 1 \]

\[ \leq m! + m! \]

\[ < m!m + m! \]

\[ = m!(m + 1) \]

\[ = (m + 1)! \]

\[ \Downarrow \]

\[ m! + k < (m + 1)! \]

\[ \Downarrow \]

\[ m! + k \neq p! \quad \text{for any} \quad p \]
\[ a^{m!+k} \in L \quad 1 \leq k \leq m \]

**BUT:**

\[ L = \{ a^{n!} : n \geq 0 \} \]

\[ a^{m!+k} \notin L \]

**CONTRADICTION!!!**
Therefore: Our assumption that $L$ is a regular language is not true

**Conclusion:** $L$ is not a regular language
Lex
Lex: a lexical analyzer

- A Lex program recognizes strings

- For each kind of string found, the lex program takes an action
Input

Var = 12 + 9;
if (test > 20)
  temp = 0;
else
  while (a < 20)
    temp++;

Output

Identifier: Var
Operand: =
Integer: 12
Operand: +
Integer: 9
Semicolumn: ;
Keyword: if
Parenthesis: ( Identifier: test
  ....
In Lex strings are described with regular expressions

Lex program

Regular expressions

"+" /* operators */

"-" "="

"if" "then" /* keywords */
Lex program

Regular expressions

(0|1|2|3|4|5|6|7|8|9)+  /* integers */

(a|b|..|z|A|B|...|Z)+  /* identifiers */
integers

$(0|1|2|3|4|5|6|7|8|9)^+ \
[0-9]^+$
identifiers

(a|b|..|z|A|B|...|Z)^+

[a-zA-Z]^{+}
Each regular expression has an associated action (in C code)

**Examples:**

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>linenum++;</td>
</tr>
<tr>
<td>[0-9]+</td>
<td>printf(“integer”);</td>
</tr>
<tr>
<td>[a-zA-Z]+</td>
<td>printf(“identifier”);</td>
</tr>
</tbody>
</table>
Default action: ECHO;

Prints the string identified to the output
A small program

%%

[ \t\n] ; /*skip spaces*/

[0-9]+ printf("Integer\n");

[a-zA-Z]+ printf("Identifier\n");
### Input

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>var 566</td>
<td>78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9800</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Output

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Integer</td>
</tr>
</tbody>
</table>
Another program

```c
int linenum = 1;
%
%
[ \t] ; /*skip spaces*/

linenum++;

[0-9]+ printf(“Integer\n”);

[a-zA-Z]+ printf(“Identifier\n”);

. printf(“Error in line: %d\n”, linenum);
```
Input

1234    test
var 566    78
9800    +
temp

Output

Integer
Identifier
Identifier
Integer
Integer
Integer
Error in line: 3
Identifier
Lex matches the longest input string

Example: Regular Expressions

```
Input:       ifend       if       ifn
Matches:    "ifend"    "if"    nomatch
```

“if”
“ifend”
The final states of the DFA are associated with actions.