1) Regular Expressions
   a) Compilers
      i) Token Analyzer
         (1) The analysis of expressions performed by compilers is facilitated by regular expressions.
      ii) Parser (Syntax Analyzer)
         (1) Context Free Grammars
         (2) Once the token analyzer breaks all the code up the parser ensures validity of your syntax.
      iii) Semantic Analyzer
   2) Prove that Regular Expressions are the Equivalence of Finite Automata
      a) Them. (S66) (M56) A language is regular if and only if some regular expression describes it.
      b) Prove
         i) If we have a regular language a regular expression describes it.
         ii) If we have a regular expression the language that describes it is regular.
      c) Given: Regex R
         i) Let us convert R into an NFA
            (1) R = a for some a ∈ ∑
            (a) Then L(R) = { a }
            (2) Six Rules
               (a) R is a regular expression if R is:
                  (i) 1. a for some a in the alphabet ∑
                  (ii) 2. ε
                  (iii) 3. ∅
                  (iv) 4. (R₁ ∪ R₂) where R₁ and R₂ are regular expressions
                  (v) 5. (R₁ ∘ R₂) where R₁ and R₂ are regular expressions
                  (vi) 6. (R₁*) where R is a regular expression.
            (b) In rules 1 and 2 the regular expressions a and ε represent languages { a } and { ε } respectively. Rule 3 is the empty language. Rules 4, 5, and 6 are the union, concatenation, and the star operations.
            (c) If R = ε then L(R) = { ε } : rule 2
            (d) If R = ∅ then L(R) = ∅ : rule 3
            (e) For rules 4, 5, and 6: if we have regular expressions equivalent to these NFA’s we’ve already proven they can be converted to DFA’s and thus are regular.
         ii) In the example to construct and NFA from the given string ( a ∪ b )*aba the strategy is to first construct the NFA for ( a ∪ b )* , then the NFA for aba, then combine the two.
      d) Note: we did not cover converting NFA/DFA to regular expression.
   3) Non Regular Languages
      a) Cannot be described by NFA, DFA, or Regular Expression
      b) Regular languages use finite automata which models computers with limited memory.
c) The amount of memory needed to determine whether a string is in the language is finite and independent on the length of the string.
d) If $|L|$ is infinite and would require an infinite number of subsets (states) to determine if the string is in the language
   i) Example:
      (1) $B = \{ 0^n 1^n : n \geq 0 \}$ requires infinite number of states to track.
4) The Pumping Lemma (S78)(M68)
a) Let $A$ be a regular language. Then $\exists$ an integer $p \geq 1$ called the pumping length, such that the following holds:
   i) Every string $s$ in $A$ with $|s| \geq p$ can be written as $s = xyz$ such that:
      (1) For each $i \geq 0$, $xy^iz \in A$
      (2) $|y| > 0$ ($y \neq \epsilon$)
      (3) $|xy| \leq p$
b) Proof
   i) Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA that recognizes a language $A$.
   ii) Assign the pumping length to $p = |Q|$: numbers of states
   iii) Now any string $s \in A$ where $|s| \leq p$ may be written as $s = xyz$ (1) If $|s| < p$: true
      (2) For $s \in A$, $|s| \geq p$, consider the states visited in $M$ as we compute $s$. We start in $q_1$ and end in an accept state.
      (3) Let $n$ be $|s|$, the sequence of states has length $n+1$
      (4) Since $n$ is at least $p$, $n+1 > p$ which is true, Thus using pigeonhole we must have visited at least one state twice.
   iv) $B = \{ 0^n 1^n : n \geq 0 \}$ is not regular.
      (1) Proof:
         (a) Assume $B$ is regular and let $p$ be the pumping length
         (b) Choose $s \in B$ to be $0^p 1^p$
         (c) Divide $s$ into three parts
            (i) Since $s \in B$ and $|s| \geq p$, we can divide $s = xyz$ where $xy^iz, i \geq 0$ is also in $B$
               1. $y$ is just zero’s, $xy^2z$: # too many 0’s $\in B$
               2. $y$ is all ones, $xyyz$: # of 1’s is too many $\in B$
               3. $y$ is split w 0’s and 1’s, $xyyz$: zeros came after 1
         (d) Since $s$ is a string in the language and we reach a contradiction assuming $B$ is regular, $B$ is not regular.
REGULAR EXPRESSIONS

1. PROVE THAT REGULAR EXPRESSIONS ARE

COMPILER

1. token analyzer

```
int | name | = | 4 | ; |
```

literals

variable
keyword
variable
operator

END OF STATEMENT

2. THIS ANALYSIS OF EXPRESSIONS PERFORMED BY

COMPILERS IS FACILITATED BY REGULAR

EXPRESSIONS:

PARSER (SYNTAX ANALYZER)

1. CONTEXT FREE GRAMMERS

ONCE THE TOKEN ANALYZER BREAKS

ALL THE CODE UP THE PARSER ENSURES

VALIDITY OF YOUR SYNTAX.

3. SEMANTIC ANALYZER

Equivalence of Finite Automata

Thm (S66) (M56) A language is regular
if and only if some regular expression
describes it.

Prove

If we have a regular language a regular expression

describes it

If we have a regular expression the language

that it describes is regular
Let's convert $R$ into an NFA

**Six Rules**

1. $R = a$ for some $a \in \Sigma$
   
   Then $L(R) = \{a\}$

   $$\rightarrow \bigcirc \xrightarrow{a} \bigcirc$$

**Defn**: $R$ is a regular expression if $R$ is

1. $a$ for some $a$ in the alphabet $\Sigma$
2. $\varepsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$ where $R_1$ and $R_2$ are regex
5. $(R_1 \cdot R_2)$ where $R_1$ and $R_2$ are regex
6. $(R^*)$ where $R$ is a regex

In 1, 2, 3, the regex $a$ and $\varepsilon$ represent the languages $\{a\}$ and $\{\varepsilon\}$ respectively. $\emptyset$ is the empty language.

4, 5, 6 are the unions, concatenations, and the star operations.

2. $R = \varepsilon$ then $L(R) = \{\varepsilon\}$

   $$\rightarrow \bigcirc$$

3. $R = \emptyset$ then $L(R) = \emptyset$

   $$\rightarrow \emptyset$$

4, 5, 6 \to if we have regular expressions equivalent to these NFAs we've already proven they can be converted to DFAs and thus are regular.
Example: \((a \cup b)^* \text{aba}\)

\[
\begin{align*}
a & \rightarrow o \quad a \rightarrow o \\
b & \rightarrow o \quad b \rightarrow o \\
e & \rightarrow o \\
a & \rightarrow o \\
e & \rightarrow o \\
b & \rightarrow o \\
e & \rightarrow o \\
\end{align*}
\]

\((a \cup b)^*\)

\[
\begin{align*}
e & \rightarrow o \\
e & \rightarrow o \\
a & \rightarrow o \\
e & \rightarrow o \\
b & \rightarrow o \\
e & \rightarrow o \\
\end{align*}
\]

\(\star\) - Star means zero or more of these things

\(\text{aba}\)

\[
\begin{align*}
o & \rightarrow o \quad a \rightarrow o \\
o & \rightarrow o \quad e \rightarrow o \\
o & \rightarrow o \quad b \rightarrow o \\
e & \rightarrow o \\
o & \rightarrow o \\
e & \rightarrow o \\
\end{align*}
\]

\((a \cup b)^* \text{aba}\)

\[
\begin{align*}
o & \rightarrow e \\
e & \rightarrow o \\
e & \rightarrow e \\
e & \rightarrow e \\
e & \rightarrow e \\
e & \rightarrow e \\
e & \rightarrow e \\
e & \rightarrow e \\
e & \rightarrow e \\
\end{align*}
\]

That was converting a regular expression to NFA.
We did not cover converting NFA/DFA to regular expression.

Non regular Languages
- cannot be described by NFA, DFA, or Regular Expression
- Regular languages use finite automata which models computers w/ limited memory.
- The amount of memory needed to determine whether a string is in the language is finite and independent on the length of the string.
- If $|L|$ is infinite and would be require infinite # of subsets (states) to determine if the string is in the language.

Ex:

$$B = \{0^n \mid n \geq 0 \}^3$$ requires infinite # of states to tracking.

The Pumping Lemma (578)(568)
Let $A$ be a regular expression language. Then $\exists$ an integer $p \geq 1$ called the pumping length, $s$ such that the following holds:
Every string $s$ in $A$ with $|s| \geq p$
can be written as $s = xyz$ such that
1. for each $i \geq 0$, $xy^iz \in A$
2. $|y| > 0$ ($y \neq \varepsilon$)
3. $|xy| \leq p$
Proof

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes a language $A$.

Assign the pumping length to $p = |Q|$, the number of states.

Now any string $s \in A$ where $|s| \geq p$ may be written as $s = xy \varepsilon$.

- if $|s| < p$, $s$ is true.
- For $s \in A$, $|s| \geq p$, consider the states visited in $M$ as we compute $s$. We start in $q_0$, and ends in an accept state.

Let $n$ be $|s|$, the sequence of states has length $n+1$.

Since $n$ is at least $p$, $n+1 > p$, which is the true

Thus, by using pigeonhole, we must have visited at least one state twice.

Since $s$ is a string in the language and we reach a contradiction assuming $B$ is regular, $B$ is not regular.
B = \{0^n 1^n : n \geq 0\}

is not regular

Proof. Assume B is regular and let p
be the pumping length

Choose s \in B to be 0^p 1^p

Since s \in B and |s| \geq p, we can divide
s = xyz where xy^i z, i \geq 0 is also
in B

1. y is just zero's xy^2 z \# too many 0's \notin B
2. y is all ones, xyzz \# of 1's is too many
   \notin B
3. y is split w 0's and 1's, xyzz \# of zeros came after 1

Since s is a string in the language and
we reach a contradiction assume B is
regular, B is not regular
Compiler consists of:

1. **Token analyzer** – will break up the code:

2. **Parser** – context free grammar – determines validity of statement, checks for correct syntax
3. **Semantic analyzer** – determines what the statement means, what the statement wants it to do.

### Equivalence w/ Finite Automata

**THM (S66)(M56)** A language is regular if and only if some regex describes it

\[
\text{Regex } R
\]

Let's convert R into an NFA

R is a regular expression if

1. \( R = a \) for some \( a \in \Sigma \), then \( L(R) = \{a\} \)
2. \( R = \epsilon \) then \( L(R) = \{\epsilon\} \)
3. \( R = \phi \), \( L(R) = \phi \)
4. \( (R_1 \cup R_2) \) where \( R_1 \) and \( R_2 \) are regex (this is the union operation)
5. \( (R_1 \circ R_2) \) where \( R_1 \) and \( R_2 \) are regex (this is the concatenation operation)
6. \( (R_1) \) where \( R_1 \) is a regex
Converting Regular Expression to NFA

Nonregular Languages

- The amount of memory needed to determine whether a string is in the language is finite and independent of the length of the string
- If the size of the language (|L|) is infinite and would require an infinite number of subsets (states) to determine if a string is in the language

Ex. B = \{ 0^n1^n : n \geq 0 \} since n is infinite, would need an infinite number of states to track this

The Pumping Lemma

Let A be a regular language. Then there exists an integer p \geq 1 called the pumping length, such that the following holds:

Every string s in A with |s| \geq p can be written as s = xyz such that

1. For each i \geq 0 \ xy^i z \in A (y is the segment of the string that keeps repeating/looping, x and z are the parts that are not looping)
2. |y| > 0 (y \neq \varepsilon)
3. |xy| \leq p
Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes a language $A$. Assign the pumping length to be $p = |Q|$. Now any string $s \in A$ where $|s| \geq p$ may be written as $s = xyz$

- If $|s| < p$
- For $s \in A$, $|s| \geq p$, consider the states visited in $M$ as we compute $s$. We start in $q_1$ and end in an accept state $S = q_1 s_1 q_2 s_2 q_3 \ldots s_{n-1} q_n s_n q_F$ (q is the states, we have $q_1$ because we start in a state before we read anything). Let $n$ be $|s|$, the sequence of states $(q)$ has length $n+1$
  
  Since $n$ is at least $p$, $n+1 > p$ which is the number of states, thus via pigeon hole principle, we must have visited at least one state twice.

B = \{ 0^n 1^n : n \geq 0 \} is not regular

Proof: Assume $B$ is regular and let $p$ be the pumping length

Choose $s = 0^p 1^p$ to be $0^p 1^p$

Since $s \in B$ and $|s| \geq p$, we can divide $s$ into 3 parts: $s = xyz$ where $xy^l z$, $l \geq 0$ is also in $B$

1. $y$ is just zeroes $xy^2z$, number too many 0’s $\notin B$
2. $y$ is all 1’s, $xyyz$, number of 1’s is too many $\notin B$
3. $y$ is split with 0’s and 1’s, $xyyz$, ex. 0..001..10..011..1 the zeroes and ones are in the wrong order $\notin B$

Since $s$ is a string in the language, and we reach a contradiction assuming $B$ is regular, $B$ is not regular