An alphabet $\Sigma$ is any finite set of characters.  

- e.g. $\Sigma = \{0, 1\}$ or $\Sigma = \{A, B, C, \ldots, X, Y, Z\}$ or $\Sigma = \{A, B, C, \ldots, X, Y, Z\}$

The set $\Sigma^*$ of strings over the alphabet $\Sigma$ is defined by

- Basis: $\varepsilon \in \Sigma^*$ ($\varepsilon$ is the empty string)
- Recursive: If $w \in \Sigma^*$, $a \in \Sigma$, then $wa \in \Sigma^*$

How to prove $\forall x \in S, P(x)$ is true:

**Base Case**: Show that $P(w)$ is true for all specific elements $w$ of $S$ mentioned in the Basis step

**Inductive Hypothesis**: Assume that $P$ is true for some arbitrary values of each of the existing named elements mentioned in the Recursive step

**Inductive Step**: Prove that $P(w)$ holds for each of the new elements $w$ constructed in the Recursive step using the named elements mentioned in the Inductive Hypothesis

Conclude: that $\forall x \in S, P(x)$
**prove:** \( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) for all \( x, y \in \Sigma^* \)

Let \( P(y) \) be "\( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) for all \( x \in \Sigma^* \)."

---

**Length:**

\[
\begin{align*}
\text{len}(\epsilon) & = 0; \\
\text{len}(wa) & = 1 + \text{len}(a); \text{ for } w \in \Sigma^*, a \in \Sigma
\end{align*}
\]

---

**review: rooted binary trees**

- **Basis:** is a rooted binary tree
- **Recursive step:**
  
  ![Binary tree diagram]

  if \( T_1 \) and \( T_2 \) are rooted binary trees,

  then so is:

  \[
  \begin{array}{c}
  T_1 \\
  T_2
  \end{array}
  \]

---

**size vs. height**

Claim: For every rooted binary tree \( T \), \( \text{size}(T) \leq 2^{\text{height}(T)+1} - 1 \)

---

**languages: sets of strings**

Sets of strings that satisfy special properties are called **languages**.

**Examples:**

- English sentences
- Syntactically correct Java/C/C++ programs
- \( \Sigma^* \) = All strings over alphabet \( \Sigma \)
- Palindromes over \( \Sigma \)
- Binary strings that don’t have a 0 after a 1
- Legal variable names, keywords in Java/C/C++
- Binary strings with an equal # of 0’s and 1’s

---

**regular expressions**

Regular expressions over \( \Sigma \)

- **Basis:**
  
  \( \emptyset, \varepsilon \) are regular expressions
  
  \( a \) is a regular expression for any \( a \in \Sigma \)

- **Recursive step:**
  
  - If \( A \) and \( B \) are regular expressions then so are:
    
    \( A \cup B \)
    
    \( AB \)
    
    \( A^* \)

each regular expression is a "pattern"

ε matches the empty string
a matches the one character string a

(A ⊕ B)
matches all strings that either A matches or B matches (or both)

(A)
matches all strings that have a first part that A matches followed by a second part that B matches

A*
matches all strings that have any number of strings (even 0) that A matches, one after another

regular expressions in practice

• Used to define the "tokens": e.g., legal variable names, keywords in programming languages and compilers
• Used in grep, a program that does pattern matching searches in UNIX/LINUX
• Pattern matching using regular expressions is an essential feature of PHP
• We can use regular expressions in programs to process strings!

examples

• 001*
• 0*1*
• (0 ⊕ 1)(0 ⊕ 1)0
• (0*1) *
• (0 ⊕ 1)*0110 (0 ⊕ 1)*
• (00 ⊕ 11)* (01010 ⊕ 10010)(0 ⊕ 1)*

regular expressions in java

• Pattern p = Pattern.compile("a*b");
• Matcher m = p.matcher("aaaaab");
• boolean b = m.matches();

```
[01]  a 0 or 1  \^ start of string  $ end of string
[0–9]  any single digit  \. period  \\ comma  \- minus
  any single character
ab  a followed by b  (AB)
(a | b)  a or b  (A ⊕ B)
a?  zero or one of a  (A ⊕ ε)
a*  zero or more of a  A*
a+  one or more of a  AA*
```

• e.g. ^ \[^0–9]+ \[0–9]* \.[ ]\[0–9]* $ General form of decimal number e.g. 9.12 or -9,8 (Europe)

more examples

• All binary strings that have an even # of 1’s
• All binary strings that don’t contain 101

matching email addresses: RFC 822

What?! No nested comments?
limitations of regular expressions

• Not all languages can be specified by regular expressions
• Even some easy things like
  – Palindromes
  – Strings with equal number of 0’s and 1’s
• But also more complicated structures in programming languages
  – Matched parentheses
  – Properly formed arithmetic expressions
  – etc.