Review: Structural Induction

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

- **Base Case:** Show that $P(u)$ is true for all specific elements $u$ 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 $\forall x \in S, P(x)$

Function Definitions on Recursively Defined Sets

- **Length:**
  \[
  \begin{align*}
  \text{len}(\varepsilon) &= 0 \\
  \text{len}(wa) &= \text{len}(w) + 1 \quad \text{for } w \in \Sigma^*, a \in \Sigma
  \end{align*}
  \]

- **Reversal:**
  \[
  \begin{align*}
  \varepsilon^R &= \varepsilon \\
  (wa)^R &= aw^R \quad \text{for } w \in \Sigma^*, a \in \Sigma
  \end{align*}
  \]

- **Concatenation:**
  \[
  \begin{align*}
  x \cdot \varepsilon &= x \quad \text{for } x \in \Sigma^* \\
  x \cdot wa &= (x \cdot w)a \quad \text{for } x \in \Sigma^*, a \in \Sigma
  \end{align*}
  \]
\[ \text{len}(x \bullet y) = \text{len}(x) + \text{len}(y) \text{ for all } x,y \in \Sigma^* \]

Let \( P(y) \) be “\( \text{len}(x \bullet y) = \text{len}(x) + \text{len}(y) \) for all \( x \in \Sigma^* \)”.
We prove \( P(y) \) for all \( y \in \Sigma^* \) by structural induction.

**Base Case:** \( y = \varepsilon \). For any \( x \in \Sigma^* \), \( \text{len}(x \bullet \varepsilon) = \text{len}(x) \) since \( \text{len}(\varepsilon) = 0 \). Therefore \( P(\varepsilon) \) is true.

**Inductive Hypothesis:** Assume that \( P(w) \) is true for some arbitrary \( w \in \Sigma^* \).

**Inductive Step:** Goal: Show that \( P(wa) \) is true for every \( a \in \Sigma \).

Let \( a \in \Sigma \). Let \( x \in \Sigma^* \). Then \( \text{len}(x \bullet wa) = \text{len}((x \bullet w)a) \) by defn of \( \bullet \)

\[ = \text{len}(x \bullet w) + 1 \text{ by defn of len} \]
\[ = \text{len}(x) + \text{len}(w) + 1 \text{ by \text{I.H.}} \]
\[ = \text{len}(x) + \text{len}(wa) \text{ by defn of len} \]

Therefore \( \text{len}(x \bullet wa) = \text{len}(x) + \text{len}(wa) \) for all \( x \in \Sigma^* \), so \( P(wa) \) is true.

So, by induction \( \text{len}(x \bullet y) = \text{len}(x) + \text{len}(y) \) for all \( x,y \in \Sigma^* \).

---

**Functions Defined on Rooted Binary Trees**

- \( \text{size}(\bullet) = 1 \)
- \( \text{size}(\quad T_1 \quad \quad T_2) = 1 + \text{size}(T_1) + \text{size}(T_2) \)
- \( \text{height}(\bullet) = 0 \)
- \( \text{height}(\quad T_1 \quad \quad T_2) = 1 + \max\{\text{height}(T_1),\text{height}(T_2)\} \)

**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
  - $\Sigma^* = \text{All strings over alphabet } \Sigma$
  - 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”

- $\varepsilon$ matches the empty string
- $a$ matches the one character string $a$
- $(A \cup B)$ matches all strings that either $A$ matches or $B$ matches (or both)
- $(AB)$ 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

Examples

- 001*
- 0*1*
- $(0 \cup 1)0(0 \cup 1)0$
- $(0*1*)^*$
- $(0 \cup 1)^* 0110 (0 \cup 1)^*$
- $(00 \cup 11)^* (01010 \cup 10001)(0 \cup 1)^*$
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!

Regular Expressions in Java

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

\[01\] a 0 or a 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 \cup B)
a? zero or one of a (A \cup \varepsilon)
a* zero or more of a A*
a+ one or more of a AA*

- e.g. ^[\-+]?[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