announcements

Reading assignment

Midterm back today
Average 73%

Graded Homework 5 back Friday

Homework 6 out later today

review: structural induction

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

– **Base Case:** Show that $P$ is true for all specific elements 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 $P$ holds for each of the new elements constructed in the *Recursive step* using the named elements mentioned in the Inductive Hypothesis
– **Conclude** that $\forall x \in S, P(x)$

review: rooted binary trees

• **Basis:** is a rooted binary tree

• **Recursive step:** If $T_1$ and $T_2$ are rooted binary trees then so is:

\[
\begin{array}{c}
\includegraphics[width=0.2\textwidth]{binary_tree1} \\
\includegraphics[width=0.2\textwidth]{binary_tree2}
\end{array}
\]
functions defined on rooted binary trees

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

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:**
  - \( \emptyset, \lambda \) 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:
    - \( (A \cup B) \)
    - \( (AB) \)
    - \( A^* \)

examples

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

each regular expression is a “pattern”

- \( \lambda \) 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

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 hypertext scripting language PHP used for web programming
  - Also in text processing programming language Perl
regular expressions in php

- int `preg_match` (string $pattern, string $subject, ...)
- $pattern syntax:
  - [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 ∪ B)
  - a? zero or one of a (A ∪ λ)
  - 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