Lecture 18: Structural induction, regular expressions
Midterm back today!

Average  80%
Median   81%

<table>
<thead>
<tr>
<th>Range</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;60</td>
<td></td>
</tr>
<tr>
<td>60 – 69</td>
<td></td>
</tr>
<tr>
<td>70 – 79</td>
<td></td>
</tr>
<tr>
<td>80 – 89</td>
<td></td>
</tr>
<tr>
<td>90+</td>
<td></td>
</tr>
</tbody>
</table>

Graded Homework 5 back Friday
Homework 6 out later today
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 of $u \in S$ 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 constructed in the *Recursive step* using the named elements mentioned in the *Inductive Hypothesis*

- **Conclude that** $\forall x \in S, P(x)$
Rooted Binary Trees

• **Basis:** • is a rooted binary tree

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

\[
\begin{array}{c}
\text{\includegraphics[width=0.5\textwidth]{binary_tree.png}}
\end{array}
\]
Functions Defined on Rooted Binary Trees

- \( \text{size}(\bullet) = 1 \)

- \( \text{size}(\quad) = 1 + \text{size}(T_1) + \text{size}(T_2) \)

- \( \text{height}(\bullet) = 0 \)

- \( \text{height}(\quad) = 1 + \max\{\text{height}(T_1), \text{height}(T_2)\} \)
Claim: For every rooted binary tree $T$, $\text{size}(T) \leq 2^{\text{height}(T)} + 1 - 1$

Let $P(T)$ be “$\text{size}(T) \leq 2^{\text{height}(T)} + 1 - 1$”. We go by structural induction on the definition of trees.

Base Case: When $T = \bullet$, $\text{size}(T) = 1$,

\[
\text{height}(T) = 0,
\]

Note that $1 \leq 2^1 - 1 = 2^0 + 1 - 1$.

So, $P(\bullet)$ is true.

Induction Hypothesis: Suppose $P(T_1)$ and $P(T_2)$ are true for some $T_1, T_2$.

Induction Step: Note that $\text{size}(T) = 1 + \text{size}(T_1) + \text{size}(T_2)$

\[
\leq 1 + 2^{\text{height}(T_1)} + 1 - 1 + 2^{\text{height}(T_2)} + 1 - 1 \quad \text{(by IH)}
\]

\[
\leq 2^{\text{height}(T_1)} + 1 + 2^{\text{height}(T_2)} + 1 - 1
\]

\[
\leq 2(2^{\text{max}(\text{height}(T_1), \text{height}(T_2))} + 1) - 1
\]

\[
\leq 2(2^{\text{height}(T)}) - 1
\]

\[
\leq 2^{\text{height}(T) + 1} - 1
\]

So, the $P(T)$ is true for all trees by structural induction.
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

* matches the one character string a

(A ∪ 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 ∪ 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)