## CSE 311: Foundations of Computing

**Fall 2013**

**Lecture 18: Structural induction, regular expressions**


## announcements

**Reading assignment**

7th Edition, pp. 878-880 and pp. 851-855

6th Edition, pp. 817-819 and pp. 789-793

Midterm back today

Average 73%

Graded Homework 5 back Friday

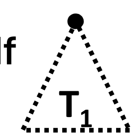
Homework 6 out later today

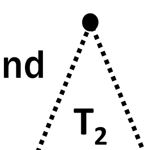
## review: structural induction

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

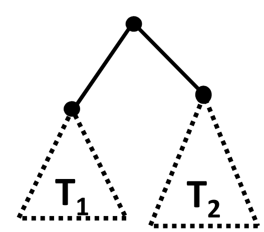
- **Base Case:** Show that $P$ is true for all specific elements of $S$ mentioned in the *Basis step*
- **Inductive Hypothesis:** Assume that $P$ is true for some arbitrary values of each of the existing named elements mentioned in the *Recursive step*
- **Inductive Step:** Prove that $P$ holds for each 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

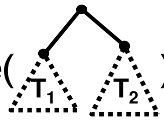
- **Basis:** • is a rooted binary tree
- **Recursive step:** If 

and 

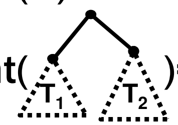
are rooted binary trees

then so is: 

## functions defined on rooted binary trees

- size( • )=1
- size( 

) = 1+size($T_1$)+size($T_2$)

- height( • )=0
- height( 

)=1+max{height($T_1$),height($T_2$)}

## for every rooted binary tree $T$, $size(T) \le 2^{height(T)+1} - 1$

## for every rooted binary tree $T$, $size(T) \le 2^{height(T)+1} - 1$

## 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**:
  - $\varnothing$, $\lambda$ 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 ∪ B)**
    - **(AB)**
    - **A***



## each regular expression is a "pattern"

$\lambda$ 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)

**(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 ∪ 1)0(0 ∪ 1)0***
- ***(0*1*)****
- ***(0 ∪ 1)* 0110 (0 ∪ 1)****
- ***(00 ∪ 11)* (01010 ∪ 10001)(0 ∪ 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 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 | ($\mathbf{A} \cup \mathbf{B}$) |
| a**?** | zero or one of a | ($\mathbf{A} \cup \lambda$) |
| 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**