CSE 311 Foundations of Computing I

Lecture 18
Recursive Definitions and Structural Induction
Autumn 2012

Announcements

- Reading assignments
  - 7^{th} Edition, Section 5.3 and pp. 878-880
  - 6^{th} Edition, Section 4.3 and pp. 817-819
  - 5^{th} Edition, Section 3.4 and pp. 766
- Midterm statistics:
  - Min 40, Max 100, Median 80, Mean 78

Highlight from last lecture: Recursive Definitions - General Form

- Recursive definition
  - *Basis step:* Some specific elements are in S
  - *Recursive step:* Given some existing named elements in S some new objects constructed from these named elements are also in S.
  - Exclusion rule: Every element in S follows from basis steps and a finite number of recursive steps

Structural Induction: proving properties of recursively defined sets

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 $\forall x \in S.\ P(x)$

Structural Induction versus Ordinary Induction

- Ordinary induction is a special case of structural induction:
  - Recursive Definition of $\mathbb{N}$
    - Basis: $0 \in \mathbb{N}$
    - Recursive Step: If $k \in \mathbb{N}$ then $k+1 \in \mathbb{N}$
- Structural induction follows from ordinary induction
  - Let Q(n) be true iff for all $x \in S$ that take n Recursive steps to be constructed, P(x) is true.

Using Structural Induction

- Let S be given by
  - Basis: $6 \in S$; $15 \in S$;
  - Recursive: if $x, y \in S$, then $x + y \in S$.
- Claim: Every element of S is divisible by 3

Strings

- An *alphabet* Σ is any finite set of characters.
- The set Σ^* of *strings* over the alphabet Σ is defined by
  - Basis: $\lambda \in \Sigma^*$ (λ is the empty string)
  - Recursive: if $w \in \Sigma^*$, $x \in \Sigma$, then $wx \in \Sigma^*$

Structural Induction for strings

- Let S be a set of strings over {a,b} defined as follows
  - Basis: $a \in S$
  - Recursive:
    - If $w \in S$ then $aw \in S$ and $baw \in S$
    - If $u \in S$ and $v \in S$ then $uv \in S$
- Claim: if $w \in S$ then w has more a's than b's

Function definitions on recursively defined sets

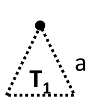
$\text{len}(\lambda) = 0$;
$\text{len}(wa) = 1 + \text{len}(w)$; for $w \in \Sigma^*$, $a \in \Sigma$

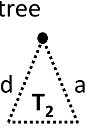
Reversal:
$\lambda^R = \lambda$
$(wa)^R = aw^R$ for $w \in \Sigma^*$, $a \in \Sigma$

Concatenation:
$x \bullet \lambda = x$ for $x \in \Sigma^*$
$x \bullet wa = (x \bullet w)a$ for $x, w \in \Sigma^*$, $a \in \Sigma$

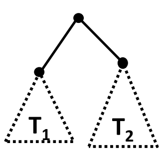
len(x•y)=len(x)+len(y) for all strings x and y

Rooted Binary trees

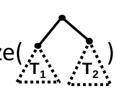
- Basis: • is a rooted binary tree
- Recursive Step: If 

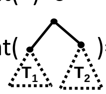
and 

are rooted binary trees then so is:



Functions defined on rooted binary trees

- size(•)=1
- size(

) = $1+\text{size}(T_1)+\text{size}(T_2)$
- height(•)=0
- height(

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

For every rooted binary tree T 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
  - All strings over alphabet Σ
  - Palindromes over Σ
  - 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 (HW6)

Regular Expressions over Σ

- Each is a "pattern" that specifies a set of strings
- Basis:
  - $\varnothing$, λ 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"

- λ 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

- ***0****
- ***0*1****
- (***0*** $\cup$ ***1***)*
- (***0*1****)*
- (***0*** $\cup$ ***1***)* ***0110*** (***0*** $\cup$ ***1***)*
- (***0*** $\cup$ ***1***)* (***0110*** $\cup$ ***100***)(***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 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

Regular expressions can't specify everything we might want

- **Fact**: Not all sets of strings can be specified by regular expressions
  – One example is the set of binary strings with equal #'s of 0's and 1's from HW6