CSE 311: Foundations of Computing

Lecture 18: Structural Induction, Regular expressions

[Comic about regular expressions]

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!

BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!

IT'S HOPELESS!

EVERYBODY STAND BACK.

I KNOW REGULAR EXPRESSIIONS.
Recursive Definitions of Sets: 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 the basis step and a finite number of recursive steps
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 $\forall x \in S, P(x)$.
Strings

• An alphabet $\Sigma$ is any finite set of characters

• The set $\Sigma^*$ of strings over the alphabet $\Sigma$ is defined by
  – Basis: $\varepsilon \in \Sigma$ ($\varepsilon$ is the empty string w/ no chars)
  – Recursive: if $w \in \Sigma^*$, $a \in \Sigma$, then $wa \in \Sigma^*$
Functions on Recursively Defined Sets (on $\Sigma^*$)

Length:
len($\varepsilon$) = 0
len(wa) = 1 + len(w) for $w \in \Sigma^*$, $a \in \Sigma$

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

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

Number of $c$’s in a string:
$\#_c(\varepsilon) = 0$
$\#_c(wc) = \#_c(w) + 1$ for $w \in \Sigma^*$
$\#_c(wa) = \#_c(w)$ for $w \in \Sigma^*$, $a \in \Sigma$, $a \neq c$
**Claim:** \( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) for all \( x, y \in \Sigma^* \)

Let \( P(y) \) be “\( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) for all \( x \in \Sigma^* \)”.

We prove \( P(y) \) for all \( y \in \Sigma^* \) by structural induction.
Claim: \( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) for all \( x, y \in \Sigma^* \)

Let \( P(y) \) be “\( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) for all \( x \in \Sigma^* \)”.

We prove \( P(y) \) for all \( y \in \Sigma^* \) by structural induction.

**Base Case:** \( y = \varepsilon \). For any \( x \in \Sigma^* \), \( \text{len}(x \cdot \varepsilon) = \text{len}(x) = \text{len}(x) + \text{len}(\varepsilon) \)

Since \( \text{len}(\varepsilon) = 0 \). Therefore \( P(\varepsilon) \) is true.
Claim: \( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) for all \( x, y \in \Sigma^* \)

Let \( P(y) \) be “\( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) for all \( x \in \Sigma^* \)”.

We prove \( P(y) \) for all \( y \in \Sigma^* \) by structural induction.

**Base Case:** \( y = \varepsilon \). For any \( x \in \Sigma^* \), \( \text{len}(x \cdot \varepsilon) = \text{len}(x) = \text{len}(x) + \text{len}(\varepsilon) \) 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 \( a \in \Sigma \).
**Claim:** \( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) \text{ for all } x, y \in \Sigma^* \)

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

**Base Case:** \( y = \varepsilon \). For any \( x \in \Sigma^* \), \( \text{len}(x \cdot \varepsilon) = \text{len}(x) = \text{len}(x) + \text{len}(\varepsilon) \) 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** \( P(wa) \) \text{ is true for every } a \in \Sigma.

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

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

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

So, by induction \( \text{len}(x \cdot y) = \text{len}(x) + \text{len}(y) \) \text{ for all } x, y \in \Sigma^* \)
Rooted Binary Trees

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

- **Recursive step:**
  
  If \( T_1 \) and \( T_2 \) are rooted binary trees,

  then also is a rooted binary tree.
Defining Functions on Rooted Binary Trees

• \( \text{size}(\bullet) = 1 \)

• \( \text{size} \left( \begin{array}{cc}
T_1 & T_2 \\
\end{array} \right) = 1 + \text{size}(T_1) + \text{size}(T_2) \)

• \( \text{height}(\bullet) = 0 \)

• \( \text{height} \left( \begin{array}{cc}
T_1 & T_2 \\
\end{array} \right) = 1 + \max\{\text{height}(T_1), \text{height}(T_2)\} \)
Claim: For every rooted binary tree $T$, $\text{size}(T) \leq 2^{\text{height}(T)} + 1 - 1$
**Claim:** For every rooted binary tree $T$, $\text{size}(T) \leq 2^{\text{height}(T)} + 1 - 1$

1. Let $P(T)$ be “$\text{size}(T) \leq 2^{\text{height}(T)} + 1 - 1$”. We prove $P(T)$ for all rooted binary trees $T$ by structural induction.

2. Base Case: $\text{size}(\bullet) = 1$, $\text{height}(\bullet) = 0$ and $1 = 2^1 - 1 = 2^{0+1} - 1$ so $P(\bullet)$ is true.
Claim: For every rooted binary tree $T$, $\text{size}(T) \leq 2^{\text{height}(T)} + 1 - 1$

1. Let $P(T)$ be “$\text{size}(T) \leq 2^{\text{height}(T)} + 1 - 1$”. We prove $P(T)$ for all rooted binary trees $T$ by structural induction.

2. Base Case: $\text{size}(\bullet) = 1$, $\text{height}(\bullet) = 0$ and $1 = 2^{1} - 1 = 2^{0+1} - 1$ so $P(\bullet)$ is true.

3. Inductive Hypothesis: Suppose that $P(T_1)$ and $P(T_2)$ are true for some rooted binary trees $T_1$ and $T_2$.


By definition, $\text{size}(\text{rooted tree}) = 1 + \text{size}(T_1) + \text{size}(T_2) \leq 1 + 2^{\text{height}(T_1)} + 1 + 2^{\text{height}(T_2)} + 1 - 1 \leq 2^{\text{height}(T_1)} + 2^{\text{height}(T_2)} - 1 \leq 2^{\max(\text{height}(T_1), \text{height}(T_2)) + 1} - 1$ which is what we wanted to show.
Claim: For every rooted binary tree $T$, $\text{size}(T) \leq 2^{\text{height}(T)} + 1 - 1$

1. Let $P(T)$ be “$\text{size}(T) \leq 2^{\text{height}(T)+1} - 1$”. We prove $P(T)$ for all rooted binary trees $T$ by structural induction.

2. Base Case: $\text{size}(\bullet)=1$, $\text{height}(\bullet)=0$ and $1=2^1-1=2^{0+1}-1$ so $P(\bullet)$ is true.

3. Inductive Hypothesis: Suppose that $P(T_1)$ and $P(T_2)$ are true for some rooted binary trees $T_1$ and $T_2$.


   By defn, $\text{size}(T) = 1 + \text{size}(T_1) + \text{size}(T_2)$

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

   by IH for $T_1$ and $T_2$

   $= 2^{\text{height}(T_1)+1} + 2^{\text{height}(T_2)+1} - 1$

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

   $= 2(2^{\text{height}(\text{max}(T_1,T_2)))} - 1 = 2^{\text{height}(\text{max}(T_1,T_2))+1} - 1$

   which is what we wanted to show.

5. So, the $P(T)$ is true for all rooted bin. 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**

*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*
Examples

001*

{00, 001, 0011, 00111, ...}

0*1*

Any number of 0’s followed by any number of 1’s
Examples

\[(0 \cup 1) \ 0 \ (0 \cup 1) \ 0\]

\[(0*1*)^*\]
Examples

\((0 \cup 1) \ 0 \ (0 \cup 1) \ 0\)

\{0000, 0010, 1000, 1010\}

\((0*1*)^*\)

All binary strings
Examples

\((0 \cup 1)^* \ 0110 \ (0 \cup 1)^*\)

\((00 \cup 11)^* \ (01010 \cup 10001) \ (0 \cup 1)^*\)
Examples

\((0 \cup 1)^* \ 0110 \ (0 \cup 1)^*\)

Binary strings that contain “0110”

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

Binary strings that begin with pairs of characters followed by “01010” or “10001”
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 ∪ ε)
a*   zero or more of a   A*
a+   one or more of a   A**

- e.g.  ^[\−+]?[0–9]* (\ . | \, )?[0–9]+$

  General form of decimal number  e.g.  9.12  or -9,8 (Europe)