CSE 311: Foundations of Computing

Lecture 18: Structural Induction, Regular expressions

Recall: Definitions of Sets from Predicates

S = the set of all x in U for which P(x) is true

$$S = \{x \in U : P(x)\}$$

This is a *non-constructive* definition.

It does not tell us how to find these elements or how to build this set ourselves.

Last time: Recursive Definition of Sets

Even numbers

Basis: $0 \in S$

Recursive: If $x \in S$, then $x+2 \in S$

Recursive definition of set S

- Basis Step: list specific elements of S
- Recursive Step: build new elements of S from existing elements of S
- Exclusion Rule: Every element in S built from the basis step and a finite number of recursive steps.

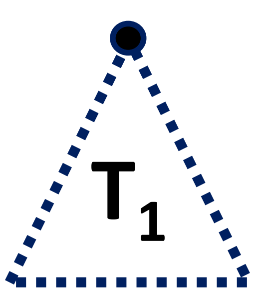
Last time: Strings

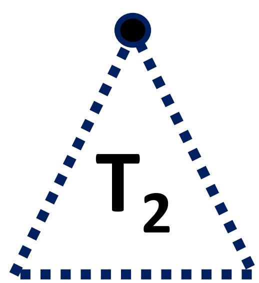
- An *alphabet* Σ is any finite set of characters

- The set Σ^* of *strings* over the alphabet Σ is defined by
  - Basis: $\varepsilon \in \Sigma^*$ (ε is the empty string w/ no chars)
  - Recursive: if $w \in \Sigma^*$, $a \in \Sigma$, then $wa \in \Sigma^*$

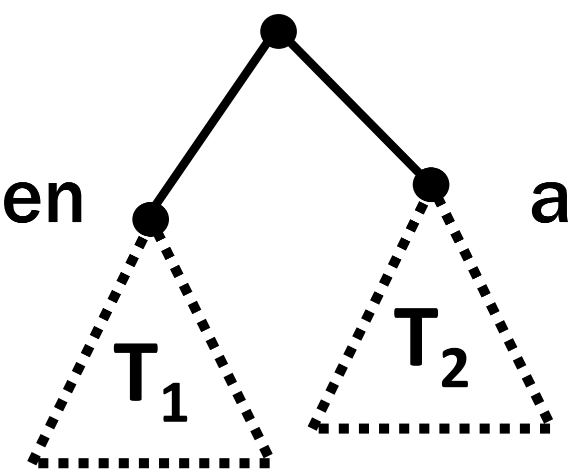
Last time: Rooted Binary Trees

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

If 

and 

are rooted binary trees,

then 

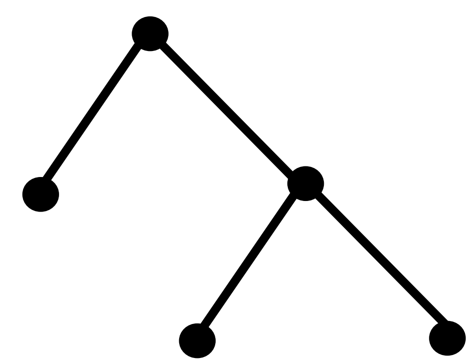
also is a rooted binary tree.

Last time: Rooted Binary Trees in Java

```
public static class BinaryTree {
  static BinaryTree LEAF = ...;
  public BinaryTree(
     BinaryTree T1, BinaryTree T2) {...}
}
```

Create general binary trees:

```
new BinaryTree(
    BinaryTree.LEAF,
    new BinaryTree(
        BinaryTree.LEAF,
        BinaryTree.LEAF)
```



Lsat time: Functions on Recursively Defined Sets

Length:

$len(\varepsilon) = 0$

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

Concatenation:

$x \bullet \varepsilon = x$ for $x \in \Sigma^*$

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

Reversal:

$\varepsilon^R = \varepsilon$

$(wa)^R = a \bullet w^R$ for $w \in \Sigma^*$, $a \in \Sigma$

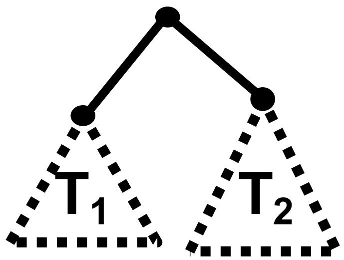
Number of c**'s in a string:**

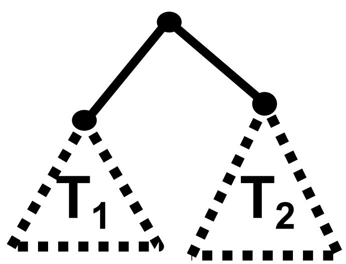
$\#_c(\varepsilon) = 0$

$\#_c(wc) = \#_c(w) + 1$ for $w \in \Sigma^*$

$\#_c(wa) = \#_c(w)$ for $w \in \Sigma^*$, $a \in \Sigma$, $a \neq c$

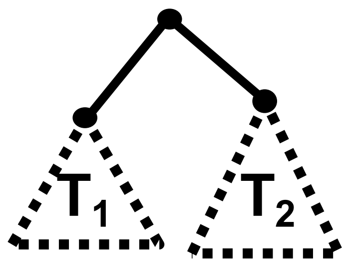
Last time: Functions on Rooted Binary Trees

- size(•) = 1
- size (

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

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

Last time: Java Functions on Rooted Binary Trees

- size(•) = 1
- size (

) = 1 + size($\mathbf{T_1}$) + size($\mathbf{T_2}$)

```
public int size(BinaryTree T) {
  if (T == BinaryTree.LEAF) {
    return 1;
  } else {
    return 1 + size(T.left()) + size(T.right());
  }
}
```

Summary

- Any recursively defined set can be translated into a Java class
- Any recursively defined function can be translated into a Java function
  – some (but not all) can be written more cleanly as loops
- Recursively defined functions and sets are our mathematical models of code and the data it operates on
- Next: how to <u>prove things</u> about code and data

Structural Induction

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

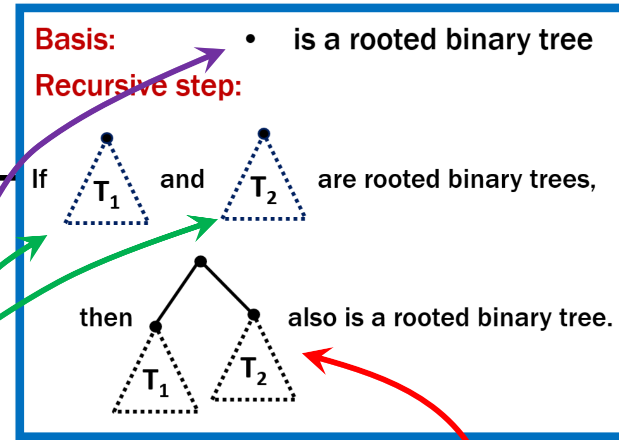
Base Case: **Show that $P(u)$ is true for all specific elements u of S mentioned in the *Basis step***

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 $\forall\, x \in S, P(x)$**

Structural Induction



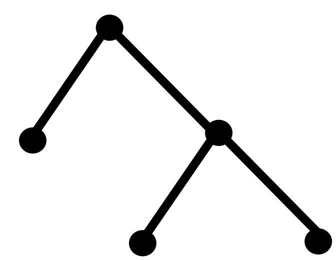
How to prove $\forall\, x \in S, P(x)$ is true:

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



Structural Induction vs. 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:

Define $Q(n)$ to be "for all $x \in S$ that can be constructed in at most n recursive steps, $P(x)$ is true."

Using Structural Induction

- Let S be given by…
  - **Basis:** $6 \in S \quad 15 \in S$
  - **Recursive:** if $x, y \in S$ then $x + y \in S$.

Claim: Every element of S is divisible by 3.

Claim: Every element of S is divisible by 3.

1. Let P(x) be "3|x". We prove that P(x) is true for all $x \in S$ by structural induction.

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.

1. Let P(x) be "3|x". We prove that P(x) is true for all x ∈ S by structural induction.
2. Base Case: 3|6 and 3|15 so P(6) and P(15) are true

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.

1. Let P(x) be "3|x". We prove that P(x) is true for all x ∈ S by structural induction.
2. Base Case: 3|6 and 3|15 so P(6) and P(15) are true
3. Inductive Hypothesis: Suppose that P(x) and P(y) are true for some arbitrary x,y ∈ S
4. Inductive Step: Goal: Show P(x+y)

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.

1. Let P(x) be "3|x". We prove that P(x) is true for all x ∈ S by structural induction.
2. Base Case: 3|6 and 3|15 so P(6) and P(15) are true
3. Inductive Hypothesis: Suppose that P(x) and P(y) are true for some arbitrary x,y ∈ S
4. Inductive Step: Goal: Show P(x+y)

   Since P(x) is true, 3|x and so x=3m for some integer m and since P(y) is true, 3|y and so y=3n for some integer n.

   Therefore x+y=3m+3n=3(m+n) and thus 3|(x+y).

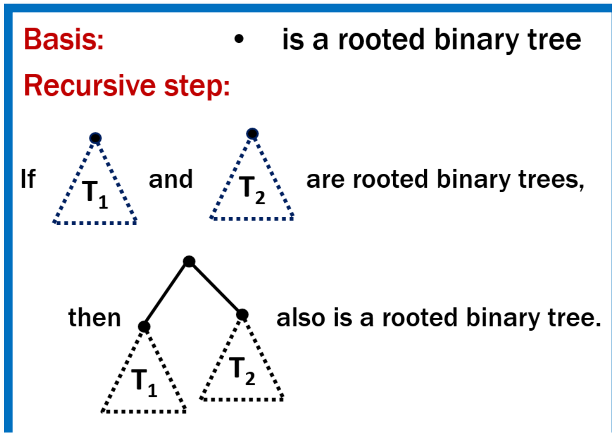
   Hence P(x+y) is true.
5. Therefore by induction 3|x for all x ∈ S.

Basis: $6 \in S$; $15 \in S$;

Recursive: if $x, y \in S$ then $x + y \in S$

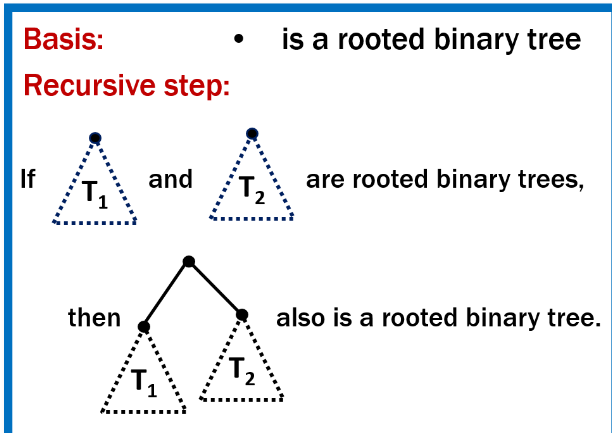
Claim: For every rooted binary tree **T**, $\text{size}(\mathbf{T}) \leq 2^{\text{height}(\mathbf{T}) + 1} - 1$

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



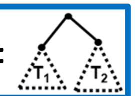
Claim: For every rooted binary tree T, size(T) ≤ $2^{\text{height}(T)+1} - 1$

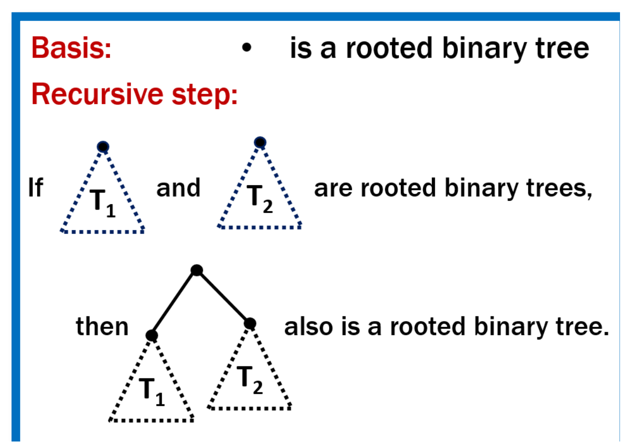
1. Let P(T) be "size(T) ≤ $2^{\text{height}(T)+1}-1$". We prove P(T) for all rooted binary trees T by structural induction.
2. **Base Case:** size(•)=1, height(•)=0, and $2^{0+1}-1=2^1-1=1$ so P(•) is true.



Claim: For every rooted binary tree T, size(T) ≤ $2^{\text{height}(T)+1} - 1$

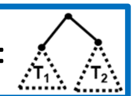
1. Let P(T) be "size(T) ≤ $2^{\text{height}(T)+1}-1$". We prove P(T) for all rooted binary trees T by structural induction.
2. **Base Case:** size(•)=1, height(•)=0, and $2^{0+1}-1=2^1-1=1$ so P(•) is true.
3. **Inductive Hypothesis:** Suppose that P(T_1) and P(T_2) are true for some rooted binary trees T_1 and T_2, e.g, size(T_1) ≤ $2^{\text{height}(T_1)+1}-1$
4. **Inductive Step:**

Goal: Prove P(**T**), where **T** = 



Claim: **For every rooted binary tree T,** size(**T**) $\leq 2^{\text{height}(\mathbf{T})+1} - 1$

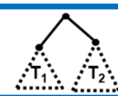
1. **Let** P(**T**) **be** "size(T) $\leq 2^{\text{height}(T)+1}-1$". **We prove** P(**T**) **for all rooted binary trees T by structural induction.**
2. **Base Case:** size(•)=1, height(•)=0, **and** $2^{0+1}-1=2^1-1=1$ **so** P(•) **is true.**
3. **Inductive Hypothesis: Suppose that** P($\mathbf{T_1}$) **and** P($\mathbf{T_2}$) **are true for some rooted binary trees** $\mathbf{T_1}$ **and** $\mathbf{T_2}$, e.g, **size(**$\mathbf{T_1}$**)** $\leq \mathbf{2^{\text{height}(T_1)+1}-1}$
4. **Inductive Step:**

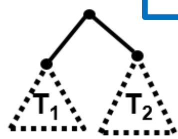
Goal: Prove P(**T**), where **T** = 

size(•) = 1	size(**T**) = size($\mathbf{T_1}$) + size($\mathbf{T_2}$) + 1
height(•) = 0	height(**T**) = max{height($\mathbf{T_1}$), height($\mathbf{T_2}$)} + 1

Claim: For every rooted binary tree T, $\text{size}(\mathbf{T}) \le 2^{\text{height}(\mathbf{T})+1} - 1$

---

1. Let P(T) be "$\text{size}(T) \le 2^{\text{height}(T)+1} - 1$". 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 $2^{0+1}-1=2^1-1=1$ so $P(\bullet)$ is true.
3. **Inductive Hypothesis:** Suppose that $P(\mathbf{T_1})$ and $P(\mathbf{T_2})$ are true for some rooted binary trees $\mathbf{T_1}$ and $\mathbf{T_2}$.
4. **Inductive Step:** Goal: Prove P().

By defn, size(

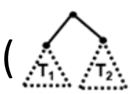
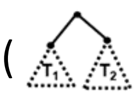
) $= 1+\text{size}(\mathbf{T_1})+\text{size}(\mathbf{T_2})$

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

by IH for $\mathbf{T_1}$ and $\mathbf{T_2}$

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

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

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

which is what we wanted to show.

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

Claim: $\text{len}(x \bullet y) = \text{len}(x) + \text{len}(y)$ **for all** $x,y \in \Sigma^*$

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

$\text{len}(\varepsilon) = 0$	$\text{len}(wa) = \text{len}(w) + 1$
$x \bullet \varepsilon = x$	$x \bullet wa = (x \bullet w)a$

Claim: $\text{len}(x \bullet y) = \text{len}(x) + \text{len}(y)$ **for all** $x,y \in \Sigma^*$

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

Base Case ($y = \varepsilon$): **Let** $x \in \Sigma^*$ **be arbitrary. Then,** $\text{len}(x \bullet \varepsilon) = \text{len}(x) = \text{len}(x) + 0 = \text{len}(x) + \text{len}(\varepsilon)$. **Since** x **was arbitrary,** $P(\varepsilon)$ **holds.**

$\text{len}(\varepsilon) = 0$	$\text{len}(wa) = \text{len}(w) + 1$
$x \bullet \varepsilon = x$	$x \bullet wa = (x \bullet w)a$

Claim: len(x•y) = len(x) + len(y) **for all** x,y $\in \Sigma^*$

Let P(y) **be** "len(x•y) = len(x) + len(y) for all x $\in \Sigma^*$".
We prove P(y) **for all** y $\in \Sigma^*$ **by structural induction.**

Base Case (y= ε): **Let** x $\in \Sigma^*$ **be arbitrary. Then,** len(x • ε) = len(x) = len(x) + 0 = len(x) + len(ε). **Since** x **was arbitrary,** P(ε) **holds.**

Inductive Hypothesis: Assume that P(w) **is true for some arbitrary** w $\in \Sigma^*$, **i.e.,** len(x•w) = len(x)+len(w) **for all** x

Inductive Step: **Goal: Show that** P(wa) **is true for every** a $\in \Sigma$

len(ε) = 0	len(wa) = len(w) + 1
x • ε = x	x • wa = (x • w)a

Claim: len(x•y) = len(x) + len(y) **for all** x,y ∈ Σ^*

Let P(y) **be** "len(x•y) = len(x) + len(y) for all x ∈ Σ^*" .
We prove P(y) **for all** y ∈ Σ^* **by structural induction.**

Base Case (y= ε): **Let** x ∈ Σ^* **be arbitrary. Then,** len(x • ε) = len(x) = len(x) + 0 = len(x) + len(ε). **Since** x **was arbitrary,** P(ε) **holds.**

Inductive Hypothesis: Assume that P(w) **is true for some arbitrary** w ∈ Σ^***, i.e.,** len(x•w) = len(x)+len(w) **for all** x

Inductive Step: **Goal: Show that** P(wa) **is true for every** a ∈ Σ

Let a ∈ Σ**. Let** x ∈ Σ^*. **Then** len(x•wa) = len((x•w)a) **by defn of •**
= len(x•w)+1 **by defn of len**
= len(x)+len(w)+1 **by I.H.**
= len(x)+len(wa) **by defn of len**

len(ε) = 0	len(wa) = len(w) + 1
x • ε = x	x • wa = (x • w)a

Claim: $len(x \bullet y) = len(x) + len(y)$ **for all** $x,y \in \Sigma^*$

Let $P(y)$ **be** "$len(x \bullet y) = len(x) + len(y)$ for all $x \in \Sigma^*$" .
We prove $P(y)$ **for all** $y \in \Sigma^*$ **by structural induction.**

Base Case ($y = \varepsilon$): **Let** $x \in \Sigma^*$ **be arbitrary. Then,** $len(x \bullet \varepsilon) = len(x) = len(x) + 0 = len(x) + len(\varepsilon)$. **Since** x **was arbitrary,** $P(\varepsilon)$ **holds.**

Inductive Hypothesis: Assume that $P(w)$ **is true for some arbitrary** $w \in \Sigma^*$**, i.e.,** $len(x \bullet w) = len(x) + len(w)$ **for all** x

Inductive Step: **Goal: Show that** $P(wa)$ **is true for every** $a \in \Sigma$

Let $a \in \Sigma$**. Let** $x \in \Sigma^*$. **Then** $len(x \bullet wa) = len((x \bullet w)a)$ **by defn of** $\bullet$
$= len(x \bullet w) + 1$ **by defn of len**
$= len(x) + len(w) + 1$ **by I.H.**
$= len(x) + len(wa)$ **by defn of len**

Therefore $len(x \bullet wa) = len(x) + len(wa)$ **for all** $x \in \Sigma^*$**, so** $P(wa)$ **is true.**

So, by induction $len(x \bullet y) = len(x) + len(y)$ **for all** $x,y \in \Sigma^*$

Theoretical Computer Science

Languages: Sets of Strings

- Subsets of strings are called *languages*
- Examples:
  - $\Sigma^* =$ All strings over alphabet Σ
  - Palindromes over Σ
  - Binary strings that don't have a 0 after a 1
  - Binary strings with an equal # of 0's and 1's
  - Legal variable names. keywords in Java/C/C++
  - Syntactically correct Java/C/C++ programs
  - English sentences

Foreword on Intro to Theory C.S.

- Look at different ways of defining languages
- See which are more expressive than others
  - i.e., which can define more languages

- Later: connect ways of defining languages to different types of (restricted) computers
  - computers capable of recognizing those languages i.e., distinguishing strings in the language from not

- Consequence: computers that recognize more expressive languages are more powerful

Regular Expressions

Regular expressions over Σ

- **Basis:**

  ε is a regular expression (could also include $\varnothing$)

  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("aaaaab");
- 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	**AA***

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

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