CSE 311 Lecture 20: Regular Expressions

Emina Torlak and Sami Davies

Topics

Structural induction

A brief review of Lecture 19.

Regular expressions

Definition, examples, applications.

Context-free grammars

Syntax, semantics, and examples.

Structural induction

A brief review of Lecture 19.

Structural induction proof template

① **Let $P(x)$ be *[definition of $P(x)$]*.**

We will show that $P(x)$ is true for every $x \in S$ by structural induction.

② **Base cases:**

[Proof of $P(s_0), \dots, P(s_m)$.]

③ **Inductive hypothesis:**

Assume that $P(y_0), \dots, P(y_k)$ are true for some arbitrary $y_0, \dots, y_k \in S$.

④ **Inductive step:**

We want to prove that $P(y)$ is true.

*[Proof of $P(y)$. The proof **must** invoke the structural inductive hypothesis.]*

⑤ **The result follows for all $x \in S$ by structural induction.**

Recursive definition of S

Basis step:

$s_0 \in S, \dots, s_m \in S.$

Recursive step:

if $y_0, \dots, y_k \in S$, then $y \in S$.

If the **recursive step** of S includes multiple rules for constructing new elements from existing elements, then ③ **assume** P for the existing elements in every rule, and ④ **prove** P for the new element in every rule.

Structural induction works just like ordinary induction

① **Let $P(x)$ be** *[definition of $P(x)$]*.

We will show that $P(x)$ is true for every $x \in \mathbb{N}$ by structural induction.

② **Base cases:**

[Proof of $P(0)$.]

③ **Inductive hypothesis:**

Assume that $P(n)$ is true for some arbitrary $n \in \mathbb{N}$.

④ **Inductive step:**

We want to prove that $P(n+1)$ is true.

*[Proof of $P(n+1)$. The proof **must** invoke the structural inductive hypothesis.]*

⑤ **The result follows for all $x \in \mathbb{N}$ by structural induction.**

Recursive definition of $\mathbb{N}$

Basis step: $0 \in \mathbb{N}$.

Recursive step:

if $n \in \mathbb{N}$, then

$n + 1 \in \mathbb{N}$.

Ordinary induction is just structural induction applied to the recursively defined set of natural numbers!

Understanding structural induction

$$\frac{P(\bullet);\forall L,R\in S.\,(P(L)\wedge P(R))\rightarrow P(\text{Tree}(\bullet,L,R))}{\therefore\ \forall x\in S.\,P(x)}$$

How do we get $P(\text{Tree}(\bullet,\bullet,\text{Tree}(\bullet,\bullet,\bullet)))$ from $P(\bullet)$ and $\forall L,R\in S.\,(P(L)\wedge P(R))\rightarrow P(\text{Tree}(\bullet,L,R))$?

Define S by

Basis: $\bullet\in S$.

Recursive:

if $L,R\in S$, then $\text{Tree}(\bullet,L,R)\in S$

1. First, we have $\forall L,R\in S.\,(P(L)\wedge P(R))\rightarrow P(\text{Tree}(\bullet,L,R))$
2. Next, we have $P(\bullet)$.
3. Intro $\wedge$ on 2 gives us $P(\bullet)\wedge P(\bullet)$.
4. Elim $\forall$ on 1 gives us $(P(\bullet)\wedge P(\bullet))\rightarrow P(\text{Tree}(\bullet,\bullet,\bullet))$.
5. Modus Ponens on 3 and 4 gives us $P(\text{Tree}(\bullet,\bullet,\bullet))$.
6. Intro $\wedge$ on 2 and 5 gives us $P(\bullet)\wedge P(\text{Tree}(\bullet,\bullet,\bullet))$.
7. Elim $\forall$ on 1 gives us $(P(\bullet)\wedge P(\text{Tree}(\bullet,\bullet,\bullet)))\rightarrow P(\text{Tree}(\bullet,\bullet,\text{Tree}(\bullet,\bullet,\bullet)))$.
8. Modus Ponens on 6 and 7 gives us $P(\text{Tree}(\bullet,\bullet,\text{Tree}(\bullet,\bullet,\bullet)))$.

$P(\bullet)$

$P(\bullet)\wedge P(\bullet)$

$\Downarrow\ (P(\bullet)\wedge P(\bullet))\rightarrow P(\text{Tree}(\bullet,\bullet,\bullet))$

$P(\text{Tree}(\bullet,\bullet,\bullet))$

$P(\bullet)\wedge P(\text{Tree}(\bullet,\bullet,\bullet))$

$\Downarrow\ (P(\bullet)\wedge P(\text{Tree}(\bullet,\bullet,\bullet))\rightarrow P(\text{Tree}(\bullet,\bullet,\text{Tree}(\bullet,\bullet,\bullet)))$

$P(\text{Tree}(\bullet,\bullet,\text{Tree}(\bullet,\bullet,\bullet)))$

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

① **Let $P(y)$ be $\forall x \in \Sigma^*.\ \text{len}(x \bullet y) = \text{len}(x) + \text{len}(y)$.**

We will show that $P(y)$ is true for every $y \in \Sigma^*$ by structural induction.

② **Base case ($y = \varepsilon$):**

Let x in Σ^* be arbitrary. Then, $\text{len}(x \bullet \varepsilon) = \text{len}(x) = \text{len}(x) + \text{len}(\varepsilon)$ since $\text{len}(\varepsilon) = 0$. So $P(\varepsilon)$ is true.

③ **Inductive hypothesis:**

Assume that $P(w)$ is true for some arbitrary $w \in \Sigma^*$.

④ **Inductive step:**

We want to prove that $P(wa)$ is true for every $a \in \Sigma$.

Let $a \in \Sigma$ and $x \in \Sigma^*$ be arbitrary. Then

$$\begin{aligned} \text{len}(x \bullet wa) &= \text{len}((x \bullet w)a) && \text{by defn of } \bullet \\ &= \text{len}(x \bullet w) + 1 && \text{by defn of len} \\ &= \text{len}(x) + \text{len}(w) + 1 && \text{by IH} \\ &= \text{len}(x) + \text{len}(wa) && \text{by defn of len} \end{aligned}$$

So $\text{len}(x \bullet wa) = \text{len}(x) + \text{len}(wa)$ for all $x \in \Sigma^*$, and $P(wa)$ is true.

⑤ **The result follows for all $y \in \Sigma^*$ by structural induction.**

Define Σ^* by

Basis: $\varepsilon \in \Sigma^*$.

Recursive:

if $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$

Length

$\text{len}(\varepsilon) = 0$

$\text{len}(wa) = \text{len}(w) + 1$

Concatenation

$x \bullet \varepsilon = x$

$x \bullet (wa) = (x \bullet w)a$

Example: prove $|t| \leq 2^{\lceil t \rceil + 1} - 1$ for any rooted binary tree t

Define S **by**

Basis: $\bullet \in S$.

Recursive:

if $L, R \in S$, then

Tree$(\bullet, L, R) \in S$

Size

$| \bullet | = 1$

$|\text{Tree}(\bullet, L, R)| =$

$1 + |L| + |R|$

Height

$\lceil \bullet \rceil = 0$

$\lceil \text{Tree}(\bullet, L, R)) \rceil =$

$1 + \max(\lceil L \rceil, \lceil R \rceil)$

Example: prove $|t| \leq 2^{\lceil t \rceil + 1} - 1$ for any rooted binary tree t

① **Let $P(t)$ be** $|t| \leq 2^{\lceil t \rceil + 1} - 1$.

We will show that $P(t)$ is true for every $t \in S$ by structural induction.

Define S by

Basis: $\bullet \in S$.

Recursive:

if $L, R \in S$, then

Tree$(\bullet, L, R) \in S$

Size

$|\bullet| = 1$

$|\text{Tree}(\bullet, L, R)| =$

$1 + |L| + |R|$

Height

$\lceil \bullet \rceil = 0$

$\lceil \text{Tree}(\bullet, L, R)) \rceil =$

$1 + \max(\lceil L \rceil, \lceil R \rceil)$

Example: prove $|t| \leq 2^{\lceil t \rceil + 1} - 1$ for any rooted binary tree t

① **Let $P(t)$ be $|t| \leq 2^{\lceil t \rceil + 1} - 1.$**

We will show that $P(t)$ is true for every $t \in S$ by structural induction.

② **Base case ($t = \bullet$):**

$| \bullet | = 1 = 2^1 - 1 = 2^{0+1} - 1 = 2^{\lceil \bullet \rceil + 1} - 1$ so $P(\bullet)$ is true.

Define S by

Basis: $\bullet \in S$.

Recursive:

if $L, R \in S$, then

Tree$(\bullet, L, R) \in S$

Size

$| \bullet | = 1$

$|\text{Tree}(\bullet, L, R)| =$

$1 + |L| + |R|$

Height

$\lceil \bullet \rceil = 0$

$\lceil \text{Tree}(\bullet, L, R) \rceil =$

$1 + \max(\lceil L \rceil, \lceil R \rceil)$

Example: prove $|t| \leq 2^{\lceil t \rceil + 1} - 1$ for any rooted binary tree t

① **Let $P(t)$ be $|t| \leq 2^{\lceil t \rceil + 1} - 1$.**

We will show that $P(t)$ is true for every $t \in S$ by structural induction.

② **Base case ($t = \bullet$):**

$|\bullet| = 1 = 2^1 - 1 = 2^{0+1} - 1 = 2^{\lceil \bullet \rceil + 1} - 1$ so $P(\bullet)$ is true.

③ **Inductive hypothesis:**

Assume that $P(L)$ and $P(R)$ are true for some arbitrary $L, R \in S$.

Define S by

Basis: $\bullet \in S$.

Recursive:

if $L, R \in S$, then

Tree$(\bullet, L, R) \in S$

Size

$|\bullet| = 1$

$|\text{Tree}(\bullet, L, R)| = 1 + |L| + |R|$

Height

$\lceil \bullet \rceil = 0$

$\lceil \text{Tree}(\bullet, L, R) \rceil = 1 + \max(\lceil L \rceil, \lceil R \rceil)$

Example: prove $|t| \leq 2^{\lceil t \rceil + 1} - 1$ for any rooted binary tree t

① **Let $P(t)$ be $|t| \leq 2^{\lceil t \rceil + 1} - 1$.**

We will show that $P(t)$ is true for every $t \in S$ by structural induction.

② **Base case ($t = \bullet$):**

$|\bullet| = 1 = 2^1 - 1 = 2^{0+1} - 1 = 2^{\lceil \bullet \rceil + 1} - 1$ so $P(\bullet)$ is true.

③ **Inductive hypothesis:**

Assume that $P(L)$ and $P(R)$ are true for some arbitrary $L, R \in S$.

④ **Inductive step:**

We want to prove that $P(\text{Tree}(\bullet, L, R))$ is true.

Define S by

Basis: $\bullet \in S$.

Recursive:

if $L, R \in S$, then

$\text{Tree}(\bullet, L, R) \in S$

Size

$|\bullet| = 1$

$|\text{Tree}(\bullet, L, R)| =$

$1 + |L| + |R|$

Height

$\lceil \bullet \rceil = 0$

$\lceil \text{Tree}(\bullet, L, R) \rceil =$

$1 + \max(\lceil L \rceil, \lceil R \rceil)$

Example: prove $|t| \leq 2^{\lceil t \rceil + 1} - 1$ for any rooted binary tree t

① **Let $P(t)$ be $|t| \leq 2^{\lceil t \rceil + 1} - 1$.**
We will show that $P(t)$ is true for every $t \in S$ by structural induction.

② **Base case ($t = \bullet$):**
$| \bullet | = 1 = 2^1 - 1 = 2^{0+1} - 1 = 2^{\lceil \bullet \rceil + 1} - 1$ so $P(\bullet)$ is true.

③ **Inductive hypothesis:**
Assume that $P(L)$ and $P(R)$ are true for some arbitrary $L, R \in S$.

④ **Inductive step:**
We want to prove that $P(\text{Tree}(\bullet, L, R))$ is true.

$$
\begin{aligned}
|\text{Tree}(\bullet, L, R)| &= 1 + |L| + |R| && \text{by defn of } || \\
&\leq 1 + (2^{\lceil L \rceil + 1} - 1) + (2^{\lceil R \rceil + 1} - 1) && \text{by IH} \\
&\leq 2^{\lceil L \rceil + 1} + 2^{\lceil R \rceil + 1} - 1 && \text{algebra} \\
&\leq 2^{\max(\lceil L \rceil, \lceil R \rceil) + 1} + 2^{\max(\lceil L \rceil, \lceil R \rceil) + 1} - 1 && \text{by defn of max} \\
&\leq 2(2^{\max(\lceil L \rceil, \lceil R \rceil) + 1}) - 1 && \text{algebra} \\
&= 2(2^{\lceil \text{Tree}(\bullet, L, R) \rceil}) - 1 && \text{by defn of } \lceil\rceil \\
&= 2^{\lceil \text{Tree}(\bullet, L, R) \rceil + 1} - 1 && \text{as desired.}
\end{aligned}
$$

Define S by
Basis: $\bullet \in S$.
Recursive:
if $L, R \in S$, then
$\text{Tree}(\bullet, L, R) \in S$

Size
$| \bullet | = 1$
$|\text{Tree}(\bullet, L, R)| =$
$1 + |L| + |R|$

Height
$\lceil \bullet \rceil = 0$
$\lceil \text{Tree}(\bullet, L, R)) \rceil =$
$1 + \max(\lceil L \rceil, \lceil R \rceil)$

Example: prove $|t| \le 2^{\lceil t \rceil + 1} - 1$ for any rooted binary tree t

① **Let $P(t)$ be $|t| \le 2^{\lceil t \rceil + 1} - 1$.**
We will show that $P(t)$ is true for every $t \in S$ by structural induction.

② **Base case ($t = \bullet$):**
$|\bullet| = 1 = 2^1 - 1 = 2^{0+1} - 1 = 2^{\lceil \bullet \rceil + 1} - 1$ so $P(\bullet)$ is true.

③ **Inductive hypothesis:**
Assume that $P(L)$ and $P(R)$ are true for some arbitrary $L, R \in S$.

④ **Inductive step:**
We want to prove that $P(\text{Tree}(\bullet, L, R))$ is true.

$$
\begin{aligned}
|\text{Tree}(\bullet, L, R)| &= 1 + |L| + |R| && \text{by defn of } |\,| \\
&\le 1 + (2^{\lceil L \rceil + 1} - 1) + (2^{\lceil R \rceil + 1} - 1) && \text{by IH} \\
&\le 2^{\lceil L \rceil + 1} + 2^{\lceil R \rceil + 1} - 1 && \text{algebra} \\
&\le 2^{\max(\lceil L \rceil, \lceil R \rceil) + 1} + 2^{\max(\lceil L \rceil, \lceil R \rceil) + 1} - 1 && \text{by defn of max} \\
&\le 2(2^{\max(\lceil L \rceil, \lceil R \rceil) + 1}) - 1 && \text{algebra} \\
&= 2(2^{\lceil \text{Tree}(\bullet, L, R) \rceil}) - 1 && \text{by defn of } \lceil\,\rceil \\
&= 2^{\lceil \text{Tree}(\bullet, L, R) \rceil + 1} - 1 && \text{as desired.}
\end{aligned}
$$

⑤ **The result follows for all $t \in S$ by structural induction.**

Define S by
Basis: $\bullet \in S$.
Recursive:
if $L, R \in S$, then
$\text{Tree}(\bullet, L, R) \in S$

Size
$|\bullet| = 1$
$|\text{Tree}(\bullet, L, R)| =$
$1 + |L| + |R|$

Height
$\lceil \bullet \rceil = 0$
$\lceil \text{Tree}(\bullet, L, R)) \rceil =$
$1 + \max(\lceil L \rceil, \lceil R \rceil)$

Regular expressions

Definition, examples, applications.

Sets of strings as languages

A *language* is a sets of strings with specific syntax, e.g.:

Syntactically correct Java/C/C++ programs.

The set Σ^* of all strings over the alphabet Σ.

Palindromes over Σ.

Binary strings with no 1's before 0's.

Sets of strings as languages

A *language* is a sets of strings with specific syntax, e.g.:

Syntactically correct Java/C/C++ programs.

The set Σ^* of all strings over the alphabet Σ.

Palindromes over Σ.

Binary strings with no 1's before 0's.

Regular expressions let us specify *regular languages*, e.g.:

All binary strings.

The strings $\{0000, 0010, 1000, 1010\}$.

All strings that contain the string "CSE311".

Regular expressions over Σ: syntax

Basis step:

$\emptyset$, ε 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 AB, $A \cup B$, and A^*.

Regular expressions over Σ: syntax

Basis step:

$\emptyset$, ε 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

AB, $A \cup B$, and A^*.

Examples: regular expressions of $\Sigma = \{0, 1\}$

Basis: $\emptyset, \varepsilon, 0, 1$.

Recursive: 01011, $0^* 1^*$, $(0 \cup 1)0(0 \cup 1)0$, etc.

Regular expressions over Σ: semantics

A regular expression over Σ represents a set of strings over Σ.

Regular expressions over Σ: semantics

A regular expression over Σ represents a set of strings over Σ.

$\emptyset$ represents the set with no strings.

Regular expressions over Σ: semantics

A regular expression over Σ represents a set of strings over Σ.

$\emptyset$ represents the set with no strings.

ε represents the set $\{\varepsilon\}$.

Regular expressions over Σ: semantics

A regular expression over Σ represents a set of strings over Σ.

- $\emptyset$ represents the set with no strings.
- ε represents the set $\{\varepsilon\}$.
- a represents the set $\{a\}$.

Regular expressions over Σ: semantics

A regular expression over Σ represents a set of strings over Σ.

$\emptyset$ represents the set with no strings.

ε represents the set $\{\varepsilon\}$.

a represents the set $\{a\}$.

AB represents the concatenation of the sets represented by A and B: $\{a \bullet b \mid a \in A, b \in B\}$.

Regular expressions over Σ: semantics

A regular expression over Σ represents a set of strings over Σ.

- $\emptyset$ represents the set with no strings.
- ε represents the set $\{\varepsilon\}$.
- a represents the set $\{a\}$.
- AB represents the concatenation of the sets represented by A and B: $\{a \bullet b \mid a \in A, b \in B\}$.
- $A \cup B$ represents the union of the sets represented by A and B: $A \cup B$.

Regular expressions over Σ: semantics

A regular expression over Σ represents a set of strings over Σ.

- $\emptyset$ represents the set with no strings.
- ε represents the set $\{\varepsilon\}$.
- a represents the set $\{a\}$.
- AB represents the concatenation of the sets represented by A and B: $\{a \bullet b \mid a \in A, b \in B\}$.
- $A \cup B$ represents the union of the sets represented by A and B: $A \cup B$.
- A^* represents the concatenation of the set represented by A with itself zero or more times: $A^* = \{\varepsilon\} \cup A \cup AA \cup AAA \cup AAAA \cup \ldots$

Regular expressions over Σ: semantics

A regular expression over Σ represents a set of strings over Σ.

$\emptyset$ represents the set with no strings.

ε represents the set $\{\varepsilon\}$.

a represents the set $\{a\}$.

AB represents the concatenation of the sets represented by A and B: $\{a \bullet b \mid a \in A, b \in B\}$.

$A \cup B$ represents the union of the sets represented by A and B: $A \cup B$.

A^* represents the concatenation of the set represented by A with itself zero or more times: $A^* = \{\varepsilon\} \cup A \cup AA \cup AAA \cup AAAA \cup \ldots$

This just defines a recursive function definition for computing the meaning of a regular expression:

$$\begin{aligned}
\text{language}(\emptyset) &= \{\,\} \\
\text{language}(\varepsilon) &= \{\varepsilon\} \\
\text{language}(AB) &= \{a \bullet b \mid a \in \text{language}(A), b \in \text{language}(B)\} \\
\text{language}(A \cup B) &= \text{language}(A) \cup \text{language}(B) \\
\text{language}(A^*) &= \{\varepsilon\} \cup \text{language}(A) \cup \text{language}(AA) \cup \ldots
\end{aligned}$$

Examples of regular expressions

001^*

0^*1^*

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

$(0^*1^*)^*$

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

Examples of regular expressions

001^*

Binary strings with “00” followed by any number of 1s.

0^*1^*

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

$(0^*1^*)^*$

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

Examples of regular expressions

001^*

Binary strings with “00” followed by any number of 1s.

0^*1^*

Binary strings with any number of 0s followed by any number of 1s.

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

$(0^*1^*)^*$

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

Examples of regular expressions

001^*

Binary strings with “00” followed by any number of 1s.

0^*1^*

Binary strings with any number of 0s followed by any number of 1s.

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

$\{0000, 0010, 1000, 1010\}$

$(0^*1^*)^*$

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

Examples of regular expressions

001^*

Binary strings with “00” followed by any number of 1s.

$0^* 1^*$

Binary strings with any number of 0s followed by any number of 1s.

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

$\{0000, 0010, 1000, 1010\}$

$(0^* 1^*)^*$

All binary strings.

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

Examples of regular expressions

001^*

Binary strings with “00” followed by any number of 1s.

$0^* 1^*$

Binary strings with any number of 0s followed by any number of 1s.

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

$\{0000, 0010, 1000, 1010\}$

$(0^* 1^*)^*$

All binary strings.

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

Binary strings that contain “0110”.

Regular expressions in practice

Used to define the *tokens* in a programming language.

Legal variable names, keywords, etc.

Used in `grep`, a Unix program that searches for patterns in a set of files.

For example, `grep "311" *.md` searches for the string "311" in all Markdown files in the current directory.

Used in programs to process strings.

These slides are generated with the help of regular expressions :)

Context-free grammars

Syntax, semantics, and examples.

Regular expressions can specify only regular languages

But many languages aren't regular, including simple ones such as

palindromes, and
strings with an equal number of 0s and 1s.

Many programming language constructs are also irregular, such as

expressions with matched parentheses, and
properly formed arithmetic expressions.

Regular expressions can specify only regular languages

But many languages aren't regular, including simple ones such as

palindromes, and

strings with an equal number of 0s and 1s.

Many programming language constructs are also irregular, such as

expressions with matched parentheses, and

properly formed arithmetic expressions.

Context-free grammars are a more powerful formalism that lets us specify all of these example languages (i.e., sets of strings)!

Context-free grammars over Σ: syntax

A context-free grammar (CFG) is a finite set of *production rules* over:

- An alphabet Σ of *terminal symbols*.
- A finite set V of *nonterminal symbols*.
- A *start symbol* from V, usually denoted by $\mathbf{S}$ (i.e., $\mathbf{S} \in V$).

Context-free grammars over Σ: syntax

A context-free grammar (CFG) is a finite set of *production rules* over:

An alphabet Σ of *terminal symbols*.

A finite set V of *nonterminal symbols*.

A *start symbol* from V, usually denoted by $\mathbf{S}$ (i.e., $\mathbf{S} \in V$).

A production rule for a nonterminal $\mathbf{A} \in V$ takes the form

$\mathbf{A} \to w_1 \mid w_2 \mid \ldots \mid w_k$

where each $w_i \in (V \cup \Sigma)^*$ is a string of nonterminals and terminals.

Context-free grammars over Σ: syntax

A context-free grammar (CFG) is a finite set of *production rules* over:

An alphabet Σ of *terminal symbols*.

A finite set V of *nonterminal symbols*.

A *start symbol* from V, usually denoted by $\mathbf{S}$ (i.e., $\mathbf{S} \in V$).

A production rule for a nonterminal $\mathbf{A} \in V$ takes the form

$\mathbf{A} \rightarrow w_1 \mid w_2 \mid \ldots \mid w_k$

where each $w_i \in (V \cup \Sigma)^*$ is a string of nonterminals and terminals.

Only nonterminals can appear on the left-hand side of a production rule.

Context-free grammars over Σ: semantics

A CFG over Σ represents a set of strings over Σ.

Compute (or *generate*) a string from this set as follows:

1. Begin with the start symbol $\mathbf{S}$ as the current string.
2. If the current string contains a nonterminal $\mathbf{A}$, apply the rule $\mathbf{A} \to w_1 \mid \ldots \mid w_k$ to replace $\mathbf{A}$ in the current string with one of the w_i's.
3. Repeat step 2 until the current string contains only terminals.

Context-free grammars over Σ: semantics

A CFG over Σ represents a set of strings over Σ.

Compute (or *generate*) a string from this set as follows:

1. Begin with the start symbol $\mathbf{S}$ as the current string.
2. If the current string contains a nonterminal $\mathbf{A}$, apply the rule $\mathbf{A} \rightarrow w_1 \mid \dots \mid w_k$ to replace $\mathbf{A}$ in the current string with one of the w_i's.
3. Repeat step 2 until the current string contains only terminals.

A CFG represents the set of all strings over Σ that can be generated in this way.

Example context-free grammars

$\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

$\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

$\mathbf{S} \rightarrow (\mathbf{S}) \mid \mathbf{SS} \mid \varepsilon$

CFG for $\{0^n 1^n \ : n \geq 0\}$, strings an equal number of 0s and 1s.

Example context-free grammars

$\mathbf{S} \to 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes.

$\mathbf{S} \to 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

$\mathbf{S} \to (\mathbf{S}) \mid \mathbf{SS} \mid \varepsilon$

CFG for $\{0^n 1^n \ : n \geq 0\}$, strings an equal number of 0s and 1s.

Example context-free grammars

$\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes.

$\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

The set of strings denoted by the regular expression 0^*1^*.

$\mathbf{S} \rightarrow (\mathbf{S}) \mid \mathbf{SS} \mid \varepsilon$

CFG for $\{0^n 1^n : n \geq 0\}$, strings an equal number of 0s and 1s.

Example context-free grammars

$\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes.

$\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

The set of strings denoted by the regular expression $0^* 1^*$.

$\mathbf{S} \rightarrow (\mathbf{S}) \mid \mathbf{SS} \mid \varepsilon$

The set of all strings of matched parentheses.

CFG for $\{0^n 1^n \ : n \geq 0\}$, strings an equal number of 0s and 1s.

Example context-free grammars

$\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes.

$\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \varepsilon$

The set of strings denoted by the regular expression 0^*1^*.

$\mathbf{S} \rightarrow (\mathbf{S}) \mid \mathbf{SS} \mid \varepsilon$

The set of all strings of matched parentheses.

CFG for $\{0^n 1^n : n \geq 0\}$, strings an equal number of 0s and 1s.

$\mathbf{S} \rightarrow 0\mathbf{S}1 \mid \varepsilon$

Summary

To prove $\forall x \in S.\, P(x)$ using structural induction:

Show that P holds for the elements in the basis step of S.

Assume P for every existing element of S named in the recursive step.

Prove P for every new element of S created in the recursive step.

A regular expression defines a set of strings over an alphabet Σ.

$\emptyset$, ε, and $a \in \Sigma$ are regular expressions.

If A and B are regular expressions, then so are (AB), $(A \cup B)$, A^*.

Many practical applications, from `grep` to everyday programming.

Context-free grammars (CFGs) are a more expressive formalism for specifying strings over an alphabet Σ.

A CFG consists of a set of *terminal symbols*, a set of *nonterminal symbols* including the distinguished *start symbol*, and a set of *production rules* that specify how to rewrite nonterminals in a string.

Used for specifying programming language syntax and for parsing.