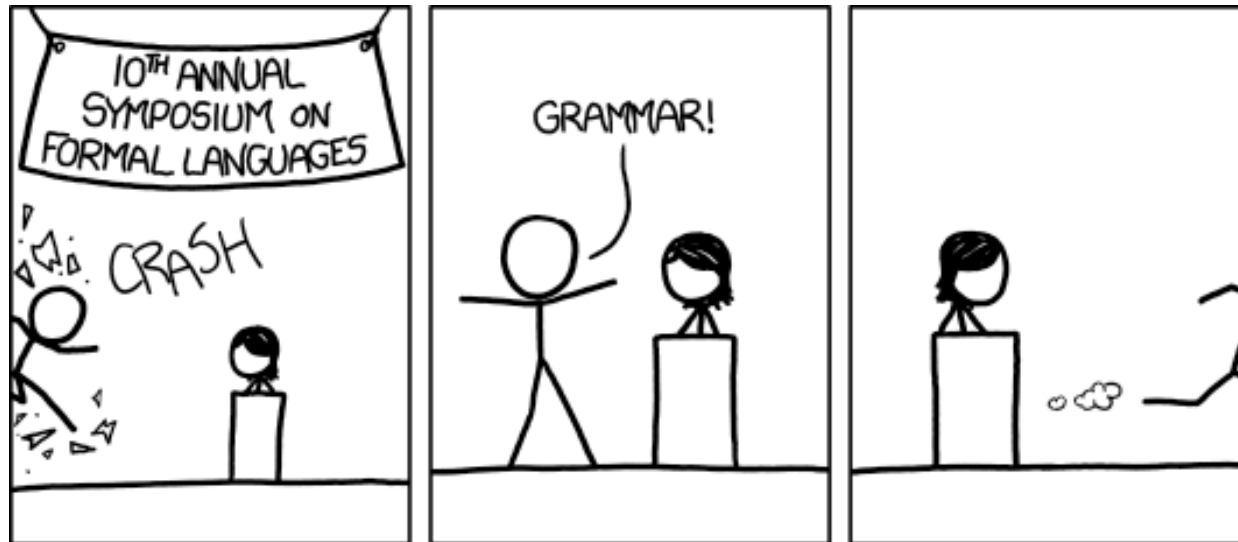


CSE 311: Foundations of Computing

Lecture 18: Regular Expressions & Context-Free Grammars



Pick up
your exam
if you don't
have it already

After class
bring your
exam if
you have a
grading question

[Audience looks around]

"What is going on? There must be some context we're missing"

Review: each regular expression is a “pattern”

ϵ matches the **empty string**

$\overset{\frown}{a}$ matches the one character string a

$\underbrace{(A \cup B)}$ 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

prep

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, 0, 1, 0~~

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

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

101

$\{ \epsilon, 00\dots0, 111\dots1, 010101\dots01 \}$

All binary strings ✓

00101

An

$(0 \cup 1)^*$

Examples

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

{0000, 0010, 1000, 1010}

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

All binary strings

Examples

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

All binary strings w 0110 as a
substring

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



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 \cup B)

a? zero or one of a (A \cup ϵ)

a* zero or more of a A*

a+ one or more of a AA*

- e.g. $^{\wedge}[\backslash-+]?[0-9]^*(\backslash.\|\backslash,)^?[0-9]^+{\$}$

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

Examples

- All binary strings that have an even # of 1's

Handwritten notes for the first example:

- $(0^*10^*)^*0^*$ ✓
- $0^*(0^*10^*)^*$ ✓
- ~~$(0^*10^*10^*)^*$~~ ✗
- ~~$(0^*10^*10^*)^* \cup 0^*$~~ ✗
- $0^*(10^*10^*)^*$ ✓

Additional notes: "110" written in red above the second expression. "00" and "00a" written in red below the first expression.

- All binary strings that *don't* contain 101

Handwritten expression for strings not containing 101:

$$0^*(1 \cup 000^*)^*0^*$$

✓

Handwritten derivation of the expression for strings not containing 101:

$$0^*(1000^*1 \cup 1^*)^*0^*$$

Arrows point from the underlined parts of the expression above to the components below:

- 1^*000^*
- $1^* \cup 000^*$

Examples



- All binary strings that have an even # of 1's

e.g., $0^*(10^*10^*)^*$

- All binary strings that *don't* contain 101

e.g., $0^*(1 \cup 000^*)^* 0^*$

Limitations of Regular Expressions

- **Not all languages can be specified by regular expressions**
- **Even some easy things like**
 - Palindromes 
 - Strings with equal number of 0's and 1's 
- **But also more complicated structures in programming languages**
 - Matched parentheses
 - Properly formed arithmetic expressions
 - etc.

Context-Free Grammars

- A Context-Free Grammar (CFG) is given by a finite set of substitution rules involving
 - A finite set \mathbf{V} of *variables* that can be replaced
 - Alphabet Σ of *terminal symbols* that can't be replaced
 - One variable, usually \mathbf{S} , is called the *start symbol*
- The rules involving a variable \mathbf{A} are written as

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

where each w_i is a string of variables and terminals – that is $w_i \in (\mathbf{V} \cup \Sigma)^*$

How CFGs generate strings

- Begin with start symbol **S**
- If there is some variable **A** in the current string you can replace it by one of the w 's in the rules for **A**
 - $A \rightarrow w_1 \mid w_2 \mid \dots \mid w_k$
 - Write this as $x\underline{A}y \Rightarrow x\underline{w}y$ کلمہ = س
 - Repeat until no variables left
- The set of strings the CFG generates are all strings produced in this way that have no variables)

Example Context-Free Grammars

Example: $S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

Set of all binary palindromes

$S \Rightarrow 0S0 \Rightarrow 00S00 \Rightarrow 001S100 \Rightarrow 001100$

Example: $S \rightarrow 0S \mid S1 \mid \varepsilon$

$0^k 1^k$

Example Context-Free Grammars

Example: $S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes

Example: $S \rightarrow 0S \mid S1 \mid \varepsilon$

0^*1^*

Example Context-Free Grammars

Grammar for $\{0^n 1^n : n \geq 0\}$

(all strings with same # of 0's and 1's with all 0's before 1's)

$$S \rightarrow 0S1 \mid \epsilon$$
$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$$

Example: $S \rightarrow (S) \mid \underline{SS} \mid \epsilon$

All strings of parentheses that are properly balanced.

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)SS \Rightarrow (S)SS \Rightarrow (S)SS$$
$$\Rightarrow (S)S$$
$$\Rightarrow (S)S$$

Example Context-Free Grammars

Grammar for $\{0^n 1^n : n \geq 0\}$

(all strings with same # of 0's and 1's with all 0's before 1's)

$$S \rightarrow 0S1 \mid \varepsilon$$

Example: $S \rightarrow (S) \mid SS \mid \varepsilon$

The set of all strings of matched parentheses

Simple Arithmetic Expressions

$E \rightarrow E + E \mid E * E \mid (E) \mid x \mid y \mid z \mid 0 \mid 1 \mid 2 \mid 3 \mid 4$
 $\mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Generate $(2 * x) + y$

Generate $x + y * z$ in two fundamentally different ways

Simple Arithmetic Expressions

$E \rightarrow E + E \mid E * E \mid (E) \mid x \mid y \mid z \mid 0 \mid 1 \mid 2 \mid 3 \mid 4$
 $\mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Generate $(2 * x) + y$

$E \Rightarrow E + E \Rightarrow (E) + E \Rightarrow (E * E) + E \Rightarrow (2 * E) + E \Rightarrow (2 * x) + E \Rightarrow (2 * x) + y$

Generate $x + y * z$ in two fundamentally different ways

$E \Rightarrow E + E \Rightarrow x + E \Rightarrow x + E * E \Rightarrow x + y * E \Rightarrow x + y * z$

$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow x + E * E \Rightarrow x + y * E \Rightarrow x + y * z$

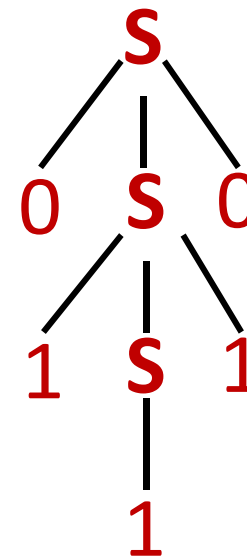
Parse Trees

Suppose that grammar G generates a string x

- A *parse tree* of x for G has
 - Root labeled S (start symbol of G)
 - The children of any node labeled A are labeled by symbols of w left-to-right for some rule $A \rightarrow w$
 - The symbols of x label the leaves ordered left-to-right

$$S \rightarrow OSO \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$$

Parse tree of 01110



CFGs and recursively-defined sets of strings

- A CFG with the start symbol **S** as its only variable recursively defines the set of strings of terminals that **S** can generate
- A CFG with more than one variable is a simultaneous recursive definition of the sets of strings generated by *each* of its variables
 - Sometimes necessary to use more than one

building precedence in simple arithmetic expressions

- **E** – expression (start symbol)
- **T** – term **F** – factor **I** – identifier **N** - number

E → **T** | **E+T**

T → **F** | **F*T**

F → (**E**) | **I** | **N**

I → **x** | **y** | **z**

N → **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Backus-Naur Form (The same thing...)

BNF (Backus-Naur Form) grammars

- Originally used to define programming languages
- Variables denoted by long names in angle brackets, e.g.
 - <identifier>, <if-then-else-statement>,
<assignment-statement>, <condition>
 - ::= used instead of \rightarrow

BNF for C

```
statement:
  ((identifier | "case" constant-expression | "default") ":")*
  (expression? ";" |
  block |
  "if" "(" expression ")" statement |
  "if" "(" expression ")" statement "else" statement |
  "switch" "(" expression ")" statement |
  "while" "(" expression ")" statement |
  "do" statement "while" "(" expression ")" ";" |
  "for" "(" expression? ";" expression? ";" expression? ")" statement |
  "goto" identifier ";" |
  "continue" ";" |
  "break" ";" |
  "return" expression? ";"
  )

block: "{" declaration* statement* "}"

expression:
  assignment-expression%

assignment-expression: (
  unary-expression (
    "=" | "*=" | "/=" | "%=" | "+=" | "-=" | "<<=" | ">>=" | "&=" |
    "^=" | "|="
  )
  )* conditional-expression

conditional-expression:
  logical-OR-expression ( "?" expression ":" conditional-expression )?
```

Parse Trees

Back to middle school:

<sentence> ::= <noun phrase> <verb phrase>

<noun phrase> ::= <article> <adjective> <noun>

<verb phrase> ::= <verb> <adverb> | <verb> <object>

<object> ::= <noun phrase>

Parse:

The yellow duck squeaked loudly

The red truck hit a parked car