CSE 311

Foundations of Computing I

* All slides are a combined effort between previous instructors of the course

CSE 311: Foundations of Computing

Lecture 19: Regular Expressions and Context-Free Grammars

Regular Expression Examples

- All binary strings that have an even # of 1's
- All binary strings that *don't* contain 101
- Let Σ = {a, b, e}. All strings with no two consecutive vowels.

Regular Expression Examples

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

$$0^*(10^*10^*)^*$$

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

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

- **Let Σ = {a, b, e}. All strings with no two consecutive vowels.**

$$b^* \cup (b^*(b^*(a \cup e)b)^*(a \cup e)b^*)$$

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 **V** of *variables* that can be replaced
  – Alphabet **Σ** of *terminal symbols* that can't be replaced
  – One variable, usually **S**, is called the *start symbol*

- The rules involving a variable **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 $w_i \in (\mathbf{V} \cup \mathbf{\Sigma})^*$

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**
  - $\mathbf{A} \rightarrow w_1 \mid w_2 \mid \cdots \mid w_k$
  - Write this as $x\mathbf{A}y \Rightarrow xwy$
  - Repeat until no variables left

- The set of strings the CFG generates are all strings produced in this way that have no variables

Context-Free Grammar Example

CFG: $S \rightarrow 0S \mid 1S \mid \varepsilon$

$S \Rightarrow 1S \Rightarrow 10S$

$\Rightarrow 10\varepsilon$

$\Rightarrow 10$

$(1 \cup 0)^*$

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

Context-Free Grammar Example

CFG: $S \rightarrow 0S \mid 1S \mid \varepsilon$

$$S \Rightarrow 0S \Rightarrow 00S \Rightarrow 000S \Rightarrow 000\varepsilon \Rightarrow 000$$

$$S \Rightarrow 0S \Rightarrow 01S \Rightarrow 010S \Rightarrow 010\varepsilon \Rightarrow 010$$

All binary strings!

Equivalent Regular Expression: $(0 \cup 1)^*$

Context-Free Grammar Example

Regular Expression: $(0 \cup 1)^*1^*$

$S \rightarrow 0S \mid 1S \mid 1$

$S_1 \rightarrow 0S_1 \mid 1S_1 \mid S_2$

$S_2 \rightarrow 1S_2 \mid \varepsilon$

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

$S_1 \Rightarrow S_2 \Rightarrow \varepsilon$

0000.... 1111

0^*1^*

Context-Free Grammar Example

Regular Expression: $(0 \cup 1)^*1$

CFG: $S \rightarrow 0S \mid 1S \mid 1$

These accept the same sets of strings!

Regular Expressions vs. CFGs

There is no regular expression for **palindomes** (with Σ={0,1}). (We'll prove this later.)

Is there a CFG for it?

$S \to \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$

~~$S \to \varepsilon \mid 0 \mid 1 \mid SS$~~

~~$S \Rightarrow SS \Rightarrow 0S \Rightarrow 01$~~

Regular Expressions vs. CFGs

There is no regular expression for **palindomes** (with Σ={0,1}). (We'll prove this later.)

Is there a CFG for it?

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

Is there a CFG for every regular expression?

There is! We won't prove this, though.

Example Context-Free Grammars

Find a CFG for $\{0^n1^n : n \geq 0\}$.

e.g. 000111, 01, ε, 00001111

$S \rightarrow 0S1 \mid \varepsilon$

What strings does $S \rightarrow (S) \mid SS \mid \varepsilon$ generate?

ε, (), (()), ()()

$[()]^*$

((()))

Example Context-Free Grammars

Find a CFG for $\{0^n1^n : n \geq 0\}$.

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

What strings does $S \rightarrow (S) \mid SS \mid \varepsilon$ generate?

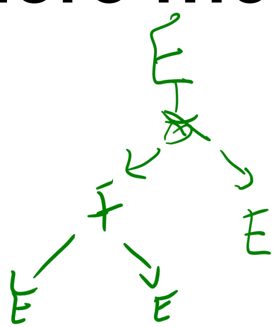
Balanced Parentheses!

Simple Arithmetic Expressions

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

Is there more than one "meaning" of "x+y*z"?

Yes: (x+y)*z, x+(y*z)



Generate it once for each meaning.

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

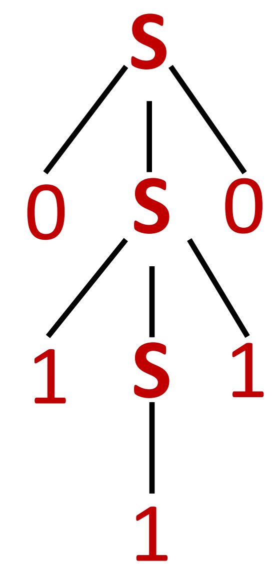
$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow x + y * z$$

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

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



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 Arithmetic Expressions

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

1 – 5

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