CSE 311: Foundations of Computing

Fall 2013

Lecture 19: Regular expressions & context-free grammars

announcements

- Reading assignments
  - 7th Edition, pp. 878-880 and pp. 851-855
  - 6th Edition, pp. 817-819 and pp. 789-793
- Today and Wednesday
  - 7th Edition, Section 9.1 and pp. 594-601
  - 6th Edition, Section 8.1 and pp. 541-548

review: languages—sets of strings

Sets of strings that satisfy special properties are called *languages*. Examples:

- English sentences
- Syntactically correct Java/C/C++ programs
- Σ^* = All strings over alphabet Σ
- Palindromes over Σ
- 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

review: regular expressions

Regular expressions over Σ

- Basis:

  $\varnothing$, λ 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^*

review: each regular expression is a "pattern"

λ matches the **empty string**

a matches the one character string *a*

(**A** ∪ **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****
- (***0*** ∪ ***1***)***0***(***0*** ∪ ***1***)***0***
- (***0*1****)*
- (***0*** ∪ ***1***)* ***0110*** (***0*** ∪ ***1***)*
- (***00*** ∪ ***11***)* (***01010*** ∪ ***10001***)(***0*** ∪ ***1***)*

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 hypertext scripting language PHP used for web programming
  - Also in text processing programming language Perl

regular expressions in php

- int **preg_match** (string $pattern , string $subject,...)
- $pattern syntax:

`[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** ∪ **λ**)
`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)

more examples

- All binary strings that have an even # of 1's
- All binary strings that *don't* contain 101

the 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 $w_i \in (\mathbf{V} \cup \Sigma)^*$

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

sample context-free grammars

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

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

sample 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)

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

simple arithmetic expressions

E→ E+E | E∗E | (E) | x | y | z | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Generate (2∗x) + y

Generate x+y∗z in two fundamentally different ways

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

another name for CFGs

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 →

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