CSE 311 Foundations of Computing I

Lecture 20
Context-Free Grammars and Languages
Autumn 2012

Announcements

- Reading Assignment
  - 7th Edition, Section 9.1 and pp. 594-601
  - 6th Edition, Section 8.1 and pp. 541-548
  - 5th Edition, Section 7.1 and pp. 493-500

Highlight from last lecture: 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***

More examples

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

Regular expressions can't specify everything we might want

- Even some easy things like palindromes
- 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
  $\mathbf{A} \to w_1 \mid w_2 \mid \ldots \mid w_k$ where each w_i is a string of variables and terminals – that is $w_i \in (\mathbf{V} \cup \mathbf{\Sigma})^*$

How Context-Free Grammars 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**
  - Write this as x**A**y ⇒ 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** → 0**S**0 | 1**S**1 | 0 | 1 | λ
- Example: **S** → 0**S** | **S**1 | λ

Sample Context-Free Grammars

- Grammar for $\{0^n1^n : n \geq 0\}$ all strings with same # of 0's and 1's with all 0's before 1's.
- Example: **S** → **(S)** | **SS** | λ

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

Context-Free Grammars 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 in 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 $\rightarrow$

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

Relations

Definition of Relations

Let A and B be sets,
A **binary relation from A to B** is a subset of $A \times B$

Let A be a set,
A **binary relation on A** is a subset of $A \times A$

Relation Examples

$R_1 = \{(a, 1), (a, 2), (b, 1), (b, 3), (c, 3)\}$

$R_2 = \{(x, y) \mid x \equiv y \pmod 5\}$

$R_3 = \{(c_1, c_2) \mid c_1$ is a prerequisite of $c_2\}$

$R_4 = \{(s, c) \mid$ student s had taken course c $\}$

Properties of Relations

Let R be a relation on A

R is reflexive iff $(a,a) \in R$ for every $a \in A$

R is symmetric iff $(a,b) \in R$ implies $(b, a) \in R$

R is antisymmetric iff $(a,b) \in R$ and $a \neq b$ implies $(b,a) \notin R$

R is transitive iff $(a,b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$