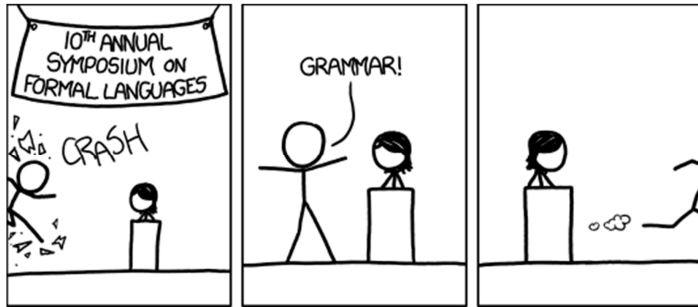


CSE 311: Foundations of Computing

Fall 2013

Lecture 20: Context-free grammars/Relations



announcements

- Reading assignments
 - 7th Edition, pp. 851-855
 - 6th Edition, pp. 789-793
- Today and Friday
 - 7th Edition, Section 9.1 and pp. 594-601
 - 6th Edition, Section 8.1 and pp. 541-548

more examples

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

review: 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.

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

review: how CFGs generate strings

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

review: sample context-free grammars

Example: $\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \lambda$

Example: $\mathbf{S} \rightarrow 0\mathbf{S} \mid \mathbf{S}1 \mid \lambda$

review: 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: $\mathbf{S} \rightarrow (\mathbf{S}) \mid \mathbf{S}\mathbf{S} \mid \lambda$

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

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

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

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

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

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

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

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

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 $(a,c) \in R$

combining relations

Let R be a relation from A to B .

Let S be a relation from B to C .

The **composite** of R and S , $S \circ R$ is the relation from A to C defined by:

$$S \circ R = \{(a, c) \mid \exists b \text{ such that } (a,b) \in R \text{ and } (b,c) \in S\}$$

examples

$(a,b) \in \text{Parent}$: b is a parent of a

$(a,b) \in \text{Sister}$: b is a sister of a

What is $\text{Sister} \circ \text{Parent}$?

What is $\text{Parent} \circ \text{Sister}$?

$$S \circ R = \{(a, c) \mid \exists b \text{ such that } (a,b) \in R \text{ and } (b,c) \in S\}$$

examples

Using the relations: Parent, Child, Brother, Sister, Sibling, Father, Mother, Husband, Wife express:

Uncle: b is an uncle of a

Cousin: b is a cousin of a