

Notes for Lecture 6

winter 2008

trev

1

Context Free Grammars

- Regular expressions are not powerful enough to describe the structure or syntax of programming language constructs – context free grammars
- Context free grammars support the notion of self-embedding for bracket matching
- Consider the productions to produce matching nested brackets
 - ▶ $S \rightarrow (S)$
 - ▶ $S \rightarrow SS$
 - ▶ $S \rightarrow \epsilon$
 - S is a nonterminal
 - $($ and $)$ are terminals or tokens
- Using alternate
 - ▶ $S \rightarrow (S) | SS | \epsilon$
- Rewriting creates: $((())())$
 - ▶ think of the nested structure in arithmetic expressions and of programs in general

winter 2008

trev

2

Context Free Grammars

- Four parts to a context free grammar (cfg)
 1. Set of terminals or tokens
 2. Set of nonterminals
 3. Set of productions: left side \rightarrow right side
 - left side – a single nonterminal
 - right side – a sequence of tokens or nonterminals
 4. A start symbol – one of the nonterminals
- Recursion – arbitrary long strings
 - ▶ $A \rightarrow Ab$ – (direct) left recursion
 - ▶ $B \rightarrow aB$ – (direct) right recursion
 - ▶ $C \rightarrow dCf$ – (direct) middle recursion/self-embedding
- Indirect recursion – no extra power
 - ▶ $P \rightarrow zQz$
 - ▶ $Q \rightarrow wPy$

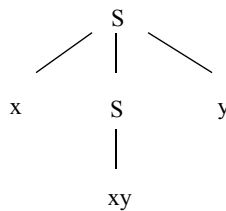
winter 2008

trev

3

Context Free Grammars

- No self-embedding implies regular
 - ▶ $S \rightarrow aS \mid bS \mid aA$
 - ▶ $A \rightarrow bB$
 - ▶ $B \rightarrow bC$
 - ▶ $C \rightarrow \epsilon$
- Produces the RE $(a|b)^*abb$
- Matching brackets – not regular
 - ▶ $S \rightarrow xSy \mid xy$
 - ▶ $\{x^n y^n \mid n > 0\}$
- Corresponding parse tree



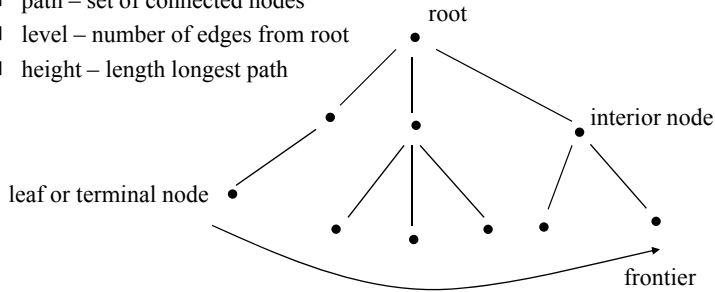
winter 2008

trev

4

Trees: Terminology

- 1 root node – no incoming edges (assume edges directed downwards)
- leaf node – no outgoing edges
- interior – not a root or leaf
- frontier – concatenation of leaves from left to right (right to left)
- path – set of connected nodes
- level – number of edges from root
- height – length longest path



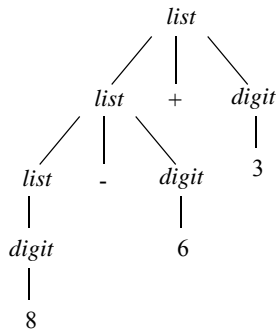
winter 2008

trev

5

Grammars and Trees

- Additive expressions
 - ▶ $list \rightarrow list + digit \mid list - digit \mid digit$
 - ▶ $digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \dots \mid 9$
 - ▶ tokens are + - 0 1 2 3 4 ...
- Parse tree for $8 - 6 + 3$



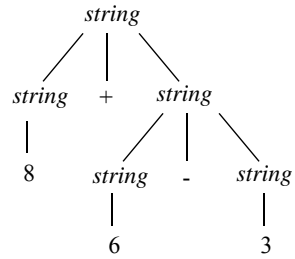
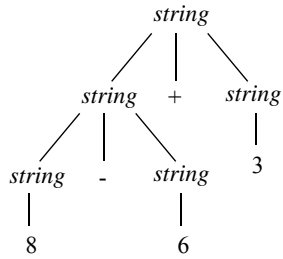
winter 2008

trev

6

Grammars and Trees

- $string \rightarrow string + string \mid string - string \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \dots \mid 9$
 - ▶ nonterminals in italic
- What is the parse tree for $8 - 6 + 3$?
- Two possibilities



winter 2008

trev

7

Ambiguity

- We have just seen an ambiguous grammar
- The language is not inherently ambiguous
- There is an unambiguous grammar that generates the language
- We can often resolve ambiguity with additional rules
 - ▶ operator precedence
 - ▶ operator associativity
- We can also define the grammar to impose precedence and associativity

winter 2008

trev

8

Operator Precedence and Associativity

- Operator precedence is why the expression $8 + 6 * 3$ is calculated as $8 + (6 * 3)$, giving 26, and not as $(8 + 6) * 3$, giving 42
- We say that the multiplication operator ($*$) has higher “precedence” or “priority” than the addition operator ($+$), so the multiplication must be performed first.
- Operator associativity is why the expression $8 - 6 - 3$ is calculated as $(8 - 6) - 3$, giving -1, and not as $8 - (6 - 3)$, giving 5
- We say that the subtraction operator ($-$) is “left associative”, so the left multiplication must be performed first. When we can't decide by operator precedence alone in which order to calculate an expression, we must use associativity.
- Since the operators $+$ and $-$ have the same precedence, and are left-associative, the following expressions are all equivalent:
 - ▶ $x + 6 - y + 3$
 - ▶ $(x + 6) - y + 3$
 - ▶ $((x + 6) - y) + 3$
 - ▶ $((x + 6) - y + 3)$

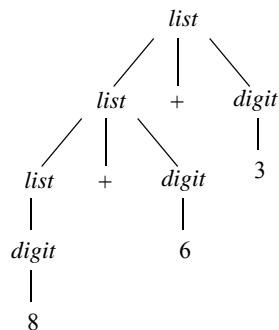
winter 2008

trev

9

Grammar for Left Associativity (e.g. +)

- Left – grows down to the left
- $list \rightarrow list + digit / digit$



winter 2008

trev

10

Grammar for Right Associativity (e.g. =)

- Right – grows down to the right
- $right \rightarrow letter = right \mid letter$

