Context-Free Languages and Parse Trees

Mridul Aanjaneya

Stanford University

July 12, 2012

Context-Free Grammars

- A context-free grammar is a notation for describing languages.
- It is more powerful than finite automata or regular expressions.
- It still cannot define all possible languages.
- Useful for nested structures, e.g., parentheses in programming languages.

Context-Free Grammars

- Basic idea is to use variables to stand for sets of strings (i.e., languages).
- These variable are defined recursively, in terms of one another.
- Recursive rules (productions) involve only concatenation.
- Alternative rules for a variable allow union.

Example: CFG for $\{0^n 1^n \mid n \geq 1\}$

- Productions:
  $S \rightarrow 01$
  $S \rightarrow 0S1$
- **Basis:** 01 is in the language.
- **Induction:** If w is in the language, then so is 0w1.

CFG Formalism

- **Terminals:** symbols of the alphabet of the language being defined.
- **Variables (nonterminals):** a finite set of other symbols, each of which represents a language.
- **Start symbol:** the variable whose language is the one being defined.

Productions

- A production has the form
  variable $\rightarrow$ string of variables and terminals
- Convention
  - A, B, C,... are variables.
  - a, b, c,... are terminals.
  - ..., X, Y, Z are either terminals or variables.
  - ..., w, x, y, z are strings of terminals only.
  - α, β, γ,... are strings of terminals and/or variables.

Example: Formal CFG

- Here is a formal CFG for $\{0^n1^n \mid n \geq 1\}$.
- Terminals $= \{0,1\}$.
- Variables $= \{S\}$.
- Start symbol $= S$.
- Productions $=$
  $S \rightarrow 01$
  $S \rightarrow 0S1$

Derivations - Intuition

- We derive strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the right side of one of its productions.
- That is, the productions for A are those that have A on the left side of the $\rightarrow$.

Derivations - Formalism

- We say $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if $A \to \gamma$ is a production.
- **Example:** $S \to 01$; $S \to 0S1$.
- $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$.

Iterated Derivation

- $\Rightarrow^*$ means zero or more derivation steps.
- **Basis:** $\alpha \Rightarrow^* \alpha$ for any string α.
- **Induction:** if $\alpha \Rightarrow^* \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \Rightarrow^* \gamma$.

Example: Iterated Derivation

- $S \rightarrow 01$; $S \rightarrow 0S1$.
- $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$.
- So $S \Rightarrow^* S$; $S \Rightarrow^* 0S1$; $S \Rightarrow^* 00S11$; $S \Rightarrow^* 000111$.

Sentential Forms

- Any string of variables and/or terminals derived from the start symbol is called a sentential form.
- Formally, α is a sentential form iff $\mathsf{S} \Rightarrow^* \alpha$.

Language of a Grammar

- If G is a CFG, then L(G), the language of G is $\{w \mid S \Rightarrow^* w\}$.
  - **Note:** w must be a terminal string, S is the start symbol.
- **Example:** G has productions $S \rightarrow \varepsilon$ and $S \rightarrow 0S1$.
  - **Note:** ε is a valid right hand side.
- $L(G) = \{0^n 1^n \mid n \geq 0\}$.

Context-Free Languages

- A language that is defined by some CFG is called a context-free language.
- There are CFL's that are not regular languages, such as the example just given.
- But not all languages are CFL's.
- **Intuition:** CFL's can count two things, not three.

BNF Notation

- Grammars for programming languages are often written in Backus-Naur Form (BNF).
- Variables are words in $< \ldots >$, e.g., $<$statement$>$.
- Terminals are often multicharacter strings indicated by boldface or underline, e.g., **while** or <u>WHILE</u>.

BNF Notation

- Symbol $::=$ is often used for $\rightarrow$.
- Symbol $|$ is used for or.
  - A shorthand for a list of productions with the same left side.

  Example: $S \rightarrow 0S1 \mid 01$ is a shorthand for $S \rightarrow 0S1$ and $S \rightarrow 01$.

BNF Notation: Kleene Closure

- Symbol ... is used for one or more.
- **Example:** <digit> ::= 0|1|2|3|4|5|6|7|8|9
  <unsigned integer> ::= <digit> ...
  - **Note:** that's not exactly the * for RE's.
- **Translation:** Replace $\alpha \ldots$ with a new variable A and productions A $\rightarrow$ Aα $|$ α.
- Grammar for unsigned integers can be replaced by:
  U $\rightarrow$ UD $|$ D
  D $\rightarrow$ 0|1|2|3|4|5|6|7|8|9

BNF Notation: Optional Elements

- Surround one or more symbols by [...] to make them optional.
- **Example:** <statement> ::= **if** <condition> **then** <statement> [;**else** <statement>]
- **Translation:** replace $[\alpha]$ by a new variable A with productions A $\rightarrow$ | ε.
- Grammar for if-then-else can be replaced by:
  S $\rightarrow$ iCtSA
  A $\rightarrow$;eS | ε

BNF Notation: Grouping

- Use $\{\ldots\}$ to surround a sequence of symbols that need to be treated as a unit.
  - Typically, they are followed by a ... for one or more.
- **Example:**
  $<$statement list$>$ ::= $<$statement$>$ [$\{$;$<$statement$>\}\ldots$]

BNF Notation: Grouping (Translation)

- You may, if you wish, create a new variable A for $\{\alpha\}$.
- One production for A: A $\rightarrow \alpha$.
- Use A in place of $\{\alpha\}$.

Example: Grouping

$$L \to S[\{;S\}\ldots]$$

- Replace $L \to S[A\ldots]$; $A \to ;S$
  - A stands for $\{;S\}$.
- Then by $L \to SB$; $B \to A\ldots \mid \varepsilon$; $A \to ;S$
  - B stands for $[A\ldots]$ (zero or more A's).
- Finally by $L \to SB$; $B \to C \mid \varepsilon$; $C \to AC \mid A$; $A \to ;S$.
  - C stands for $A\ldots$

Leftmost and Rightmost Derivations

- Derivations allow us to replace any of the variables in a string.
- Leads to many different derivations of the same string.
- By forcing the leftmost variable (or alternatively, the rightmost variable) to be replaced, we avoid these distinctions without a difference.

Leftmost Derivations

- Say $wA\alpha \Rightarrow_{\text{lm}} w\beta\alpha$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.
- Also, $\alpha \Rightarrow^{*}_{\text{lm}} \beta$ if α becomes β by a sequence of zero or more $\Rightarrow_{\text{lm}}$ steps.

Example: Leftmost Derivations

- Balanced parantheses grammar:

$$S \rightarrow SS \mid (S) \mid ()$$

- $S \Rightarrow_{\text{lm}} SS \Rightarrow_{\text{lm}} (S)S \Rightarrow_{\text{lm}} (())S \Rightarrow_{\text{lm}} (())()$
- Thus, $S \Rightarrow^*_{\text{lm}} (())()$
- $S \Rightarrow SS \Rightarrow S() \Rightarrow (S)() \Rightarrow (())()$ is a derivation, but not a leftmost derivation.

Rightmost Derivations

- Say $\alpha A w \Rightarrow_{\mathrm{rm}} \alpha\beta w$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.
- Also, $\alpha \Rightarrow^*_{\mathrm{rm}} \beta$ if α becomes β by a sequence of zero or more $\Rightarrow_{\mathrm{rm}}$ steps.

Example: Rightmost Derivations

- Balanced parantheses grammar:

$$S \to SS \mid (S) \mid ()$$

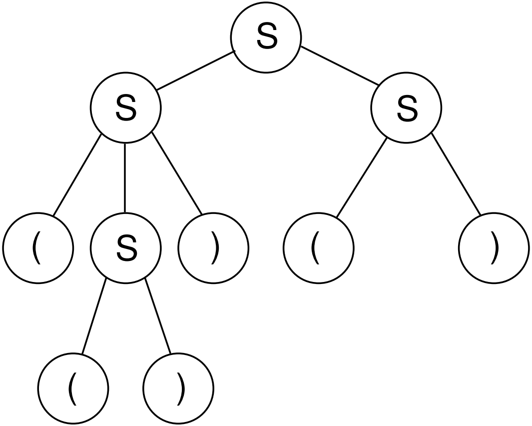
- $S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$
- Thus, $S \Rightarrow^{*}_{rm} (())()$
- $S \Rightarrow SS \Rightarrow SSS \Rightarrow S()S \Rightarrow ()()S \Rightarrow ()()()$ is neither a rightmost derivation nor a leftmost derivation.

Parse Trees

- Parse trees are trees labeled by symbols of a particular CFG.
- **Leaves:** labeled by a terminal or ε.
- **Interior nodes:** labeled by a variable.
  - Children are labeled by the right side of a production for the parent.
- **Root:** must be labeled by the start symbol.

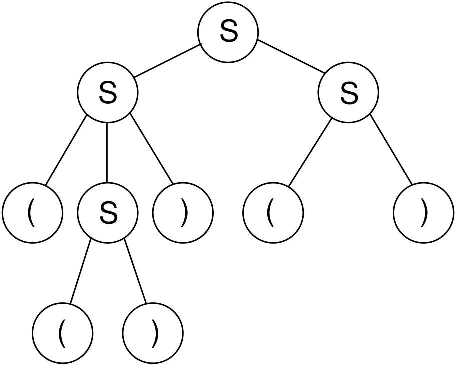
Example: Parse Tree

$$S \rightarrow SS \mid (S) \mid ()$$



Yield of a Parse Tree

- The concatenation of the labels of the leaves in left-to-right order is called the yield of the parse tree.
- **Example:** Yield of the given parse tree is (())().

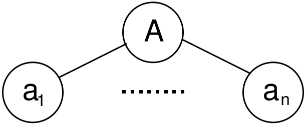


Parse Trees, Leftmost and Rightmost Derivations

- For every parse tree, there is a unique leftmost and a unique rightmost derivation.
- We'll prove:
  1. If there is a parse tree with root labeled A and yield w, then $A \Rightarrow^*_{lm} w$.
  2. If $A \Rightarrow^*_{lm} w$, then there is a parse tree with root A and yield w.

Part 1: Basis

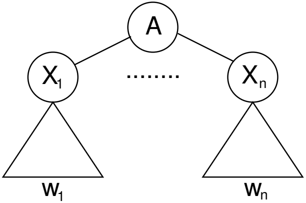
- Induction on the height (length of the longest path from the root) of the tree.
- **Basis:** height 1. Tree looks like



- $A \rightarrow a_1 \ldots a_n$ must be a production.
- Thus, $A \Rightarrow^*_{\text{lm}} a_1 \ldots a_n$.

Part 1: Induction

- Assume (1) for trees of height $< h$, and let this tree have height h:
- By **IH**, $X_i \Rightarrow^*_{lm} w_i$.
  - **Note:** If X_i is a terminal, then $X_i = w_i$.



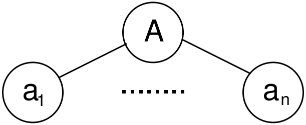
- Thus, $A \Rightarrow_{lm} X_1 \ldots X_n \Rightarrow^*_{lm} w_1 X_2 \ldots X_n \Rightarrow^*_{lm} w_1 w_2 X_3 \ldots X_n \Rightarrow^*_{lm} \cdots \Rightarrow^*_{lm} w_1 \ldots w_n$

Part 2

- Given a leftmost derivation of a terminal string, we need to prove the existence of a parse tree.
- The proof is an induction on the length of the derivation.

Part 2: Basis

- If $A \Rightarrow^{*}_{\text{lm}} a_1 a_2 \ldots a_n$ by a one-step derivation, then there must be a parse tree

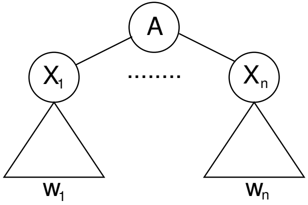


Part 2: Induction

- Assume (2) for derivations of fewer than $k > 1$ steps, and let $A \Rightarrow^*_{lm} w$ be a k-step derivation.
- First step is $A \Rightarrow_{lm} X_1 \ldots X_n$.
- **Key point:** w can be divided such that the first portion is derived from X_1, the next is derived from X_2, and so on.
  - If X_i is a terminal, then $w_i = X_i$.

Part 2: Induction

- That is, $X_i \Rightarrow^*_{\text{lm}} w_i$ for all i such that X_i is a variable.
  - And the derivation takes fewer than k steps.
- By the **IH**, if X_i is a variable, then there is a parse tree with root X_i and yield w_i.
- Thus, there is a parse tree



Parse Trees and Rightmost Derivations

- The ideas are essentially the mirror image of the proof of the leftmost derivations.
- Left to the imagination!

Parse Trees and Any Derivation

- The proof that you can obtain a parse tree from a leftmost derivation doesn't really depend on leftmost.
- First step still has to be $A \Rightarrow_{\text{lm}} X_1 \ldots X_n$.
- And w can still be divided such that the first portion is derived from X_1, the next is derived from X_2, and so on.

Ambiguous Grammars

- A CFG is ambiguous is there is a string in the language that is the yield of two or more parse trees.
- **Example:** S $\rightarrow$ SS | (S) | ()
- Two parse trees for ()()()!

Ambiguity, Leftmost and Rightmost Derivations

- If there are two different parse trees, they must produce two different leftmost derivations by the construction given in the proof.
- Conversely, two different leftmost derivations produce different parse trees by the other part of the proof.
- Likewise for rightmost derivations.

Ambiguity, Leftmost and Rightmost Derivations

- Thus, equivalent definitions for ambiguous grammar are:
  1. There is a string in the language that has two different leftmost derivations.
  2. There is a string in the language that has two different rightmost derivations.