Theory of Computation

Lecture 7: Context-free grammars

Outline

- Context Free Grammars
- Languages generated by CFGs
- Ambiguity
- Chomsky Normal Form

From Sipser Chapter 2.1

Overview

- So far we introduced two equivalent methods for describing regular languages: Finite Automata and Regular Expressions
- In the next part we something analogous:
  - We introduce context free grammars (CFGs) which describe context free languages (CFLs)
  - We introduce push-down automata (PDA) which recognize CFGs
  - We even have another pumping lemma!

Context Free Grammars

- They were first used to study human languages
- They are used for “real” computer languages (C, C++, etc.)
  – They define “rules” of the language
  – A parser uses the grammar to parse the input

A CFG example

- Here is an example grammar G1

  $A \rightarrow 0A1$

  $A \rightarrow B$

  $B \rightarrow \#$

- A grammar has substitution rules or productions
  - Each rule is stated as a variable (usually capitalized) a right pointing arrow and a sequence of variables and terminal symbols (usually non capitalized)
  - Here A and B are the variables and the terminals are 0, 1, #
  - One variable is designated as the start variable
    - Usually on the left-hand side of topmost rule
    - In this example A is the start variable

Generating a language from a CFG

We use a CFG to generate the strings of a language by replacing variables using the rules in the grammar:

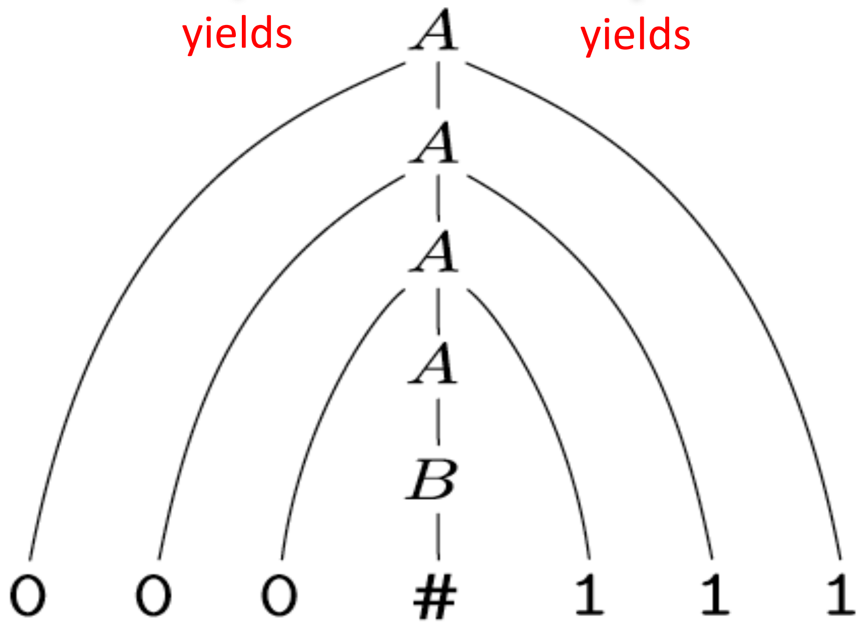
- – Start from the start variable
- – Apply productions until only terminal symbols are left
- – This process can also be represented as a parse tree
- – This process referred as derivation of a string

- Give me some strings that grammar G1 generates?

Example of derivation

A=> 0A1=> 00A11=>000A111=>000B111=>000#111

yields yields yields yields yields



A=>*000#111 means “A derives 000#111”

The Language of Grammar G1

- The set of all strings generated by G1 is the language of G1
  - Denoted as L(G1)
- We say that a language associated with a CFG is a context free language (CFL)
- What is the language of G1?
  - $L(G1) = \{0^n\#1^n \mid n \geq 0\}$
  - This should look familiar. Can we generate this with a FA? Why?

Formal Definition of a CFG

A CFG is a 4-tuple (V, Σ, R, S) where

1. V is a finite set called the variables
2. Σ is a finite set, disjoint from V, called the terminals
3. R is a finite set of rules (or productions), with each rule being a variable and a string of variables and terminals
4. $S \in V$ is the start variable

Example

Grammar G3 = ({S}, {a,b}, R, S), where:

$S \rightarrow aSb \mid SS \mid \varepsilon$

– Short form for

$S \rightarrow aSb$

$S \rightarrow SS$

$S \rightarrow \varepsilon$

– What does this generate?

- abab, aaabbb, aababb
- If you view a as “(“ and b as “)” then you get all strings of properly nested parentheses

Example

Grammar G3 = ({S}, {a,b}, R, S), where:

$S \rightarrow aSb \mid SS \mid \varepsilon$

– Derivation for "aababb"

- $S \rightarrow aSb \rightarrow aSSb \rightarrow aaSbSb \rightarrow aabSb \rightarrow aabaSbb \rightarrow aababb$

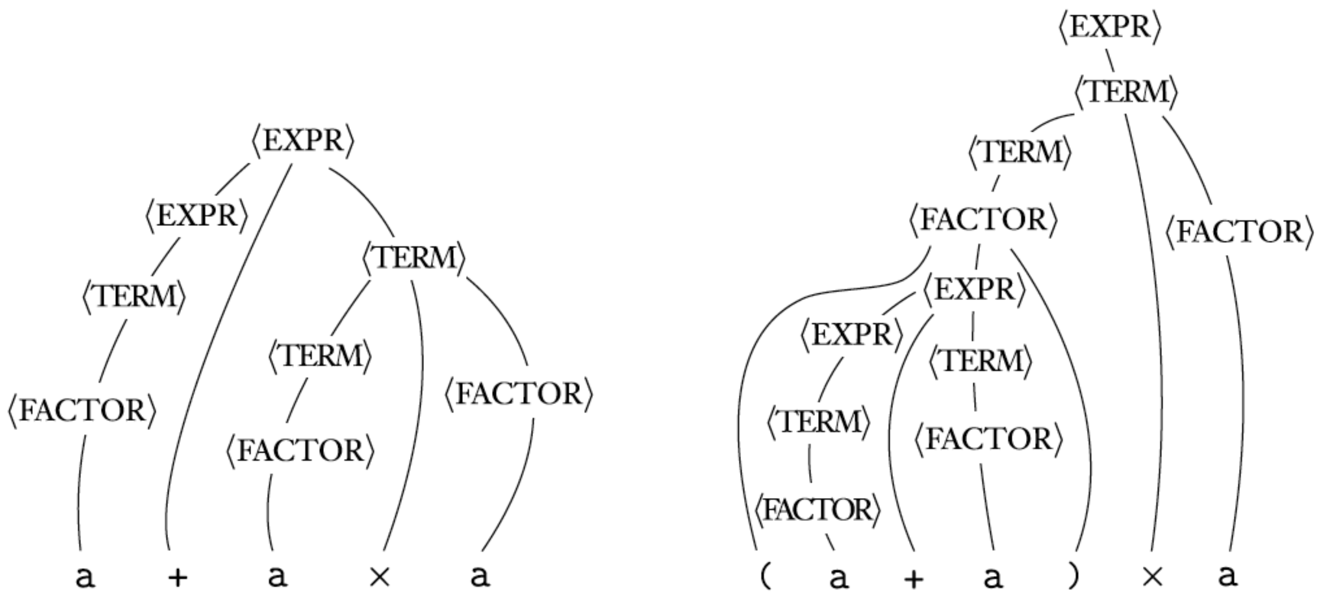
Example 2.4 Page 103 (2^{nd} ed)

Consider grammar $G_4 = (V, \Sigma, R, \langle\text{EXPR}\rangle)$.

V is $\{\langle\text{EXPR}\rangle, \langle\text{TERM}\rangle, \langle\text{FACTOR}\rangle\}$ and Σ is $\{\texttt{a}, \texttt{+}, \texttt{x}, \texttt{(}, \texttt{)}\}$. The rules are

$$\begin{aligned}
\langle\text{EXPR}\rangle &\to \langle\text{EXPR}\rangle\texttt{+}\langle\text{TERM}\rangle \mid \langle\text{TERM}\rangle \\
\langle\text{TERM}\rangle &\to \langle\text{TERM}\rangle\texttt{x}\langle\text{FACTOR}\rangle \mid \langle\text{FACTOR}\rangle \\
\langle\text{FACTOR}\rangle &\to \texttt{(}\,\langle\text{EXPR}\rangle\,\texttt{)} \mid \texttt{a}
\end{aligned}$$

The two strings a+axa and (a+a)xa can be generated with grammar G_4. The parse trees are shown in the following figure.



Designing CFGs

- Some creativity is required!
- Guidelines:
  - If the CFL is the union of simpler CFLs, design grammars for the simpler ones and then combine
    - For example, $S \to G1 \mid G2 \mid G3$
  - If the language is regular, then can design a CFG that imitates a DFA
    - Make a variable R_i for every state q_i
    - If $\delta(q_i, a) = q_j$, then add $R_i \to aR_j$
    - Add $R_i \to \varepsilon$ if q_i is an accept state
    - Make R_0 the start variable where q_0 is the start state of the DFA

This implies that CFGs subsume regular languages

Designing CFGs using unbounded space

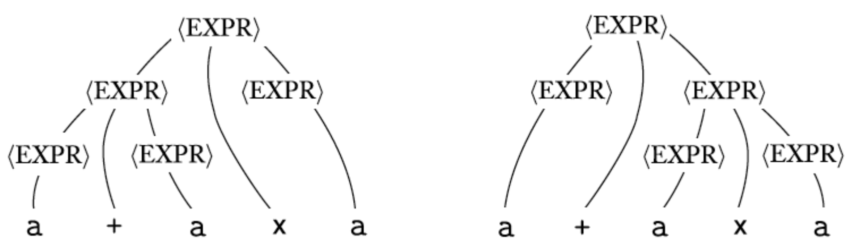
Certain CFLs contain strings that are linked in the sense that a machine for recognizing this language would need to remember an unbounded amount of information about one substring to "verify" the other substring.

- Sometimes trivial with a CFG
- Example: 0^n1^n
  - $S \rightarrow 0S1 \mid \varepsilon$

Ambiguity

Certain CFGs allow to generate the same string in multiple ways

- E.g.: EXPR $\rightarrow$ EXPR + EXPR | EXPR $\times$ EXPR |(EXPR) | a
  - EXPR→EXPR+EXPR→ EXPR+ EXPR × EXPR → a+a × a
  - EXPR→EXPR × EXPR→ EXPR+ EXPR × EXPR → a+a × a
  - The two derivations have different parse trees



  - We say that a+a × a is generated ambiguously
- A CFG which generates a string ambiguously is ambiguous

Ambiguity and ordering

A grammar generates a string ambiguously if there are two different parse trees for the string derivation:

- Two derivations may differ in the order that the rules are applied, but if they generate the same parse tree, it is not really ambiguous

- This notion of ambiguity corresponds to "linguistic notion"
  - "The girl touches the boy with a flower"
  - 2 possible meanings → 2 possible ways of "parsing the sentence" → ambiguity

Leftmost derivations and Ambiguity

- A derivation is a leftmost derivation if at every step the leftmost remaining variable is replaced
  - Are these leftmost derivations?
    - EXPR→EXPR+EXPR→ EXPR+ EXPR × EXPR → a+a×
    - EXPR→EXPR×EXPR→ EXPR+ EXPR × EXPR → a+a×
- A string *w* is derived ambiguously in a CFG G if it has two or more different leftmost derivations.

Chomsky Normal Form

- In general we want to avoid ambiguity in CFGs
- A CFG is in Chomsky normal form if every rule is of the form:

  $A \rightarrow BC$

  $A \rightarrow a$

  Where a is any terminal and A, B, and C are any variables–except B and C may not be the start variable. The start variable can also go to ε
- Any CFL can be generated by a CFG in Chomsky normal form

Converting CFG to Chomsky Normal Form

Proof by construction: in a series of steps we replace rules that violate the definition with equivalent satisfactory ones

1. Add rule $S_0 \rightarrow S$, where S was original start variable
2. Remove ε-rules. Remove $A \rightarrow \varepsilon$ and for each occurrence of A add a new rule with A deleted.

   E.g.: If we have $R \rightarrow uAvAw$, we get: $R \rightarrow uvAw \mid uAvw \mid uvw$
3. Handle all unit rules

   E.g.: If we had $A \rightarrow B$, then whenever a rule $B \rightarrow u$ exists, we add $A \rightarrow u$.
4. Replace rules $A \rightarrow u_1u_2u_3\ldots u_k$ with:

   $A \rightarrow u_1A_1;\ A_1 \rightarrow u_2A_2;\ A_2 \rightarrow u_3A_3;\ \ldots;\ A_{k-2} \rightarrow u_{k-1}u_k$

Application of these rules may require multiple passes!