CS 3313
Foundations of Computing:

Introduction to Context Free Grammars

http://gw-cs3313.github.io

1

Recall: Three key concepts

- 1. Languages
  - Set of sentences (strings) formed using characters from some alphabet
    - How do we specify the properties of the language?
- 2. Grammars
  - model for mathematically defining the properties of a language
    - rules for generating the sentences in a formal language
- 3. Automata (aka machines)
  - Mathematical model of machines (of different capabilities)
    - Reads input, produces output and may have temporary storage and can make decisions

2

A better formalism to define language ?
Some questions

- Set notation works but does not specify a way to generate the words/strings in the language
- Regular expressions work for regular languages but will not work for languages that are not regular….why ?
  - Do you have proof of this ??
- Ex: how do you define a syntactically valid C program ?
- Ex: how do you define the construction of a sentence in the English language ?

3

Need for formalism and Math rigor

- We would like to capture precisely, and logically, the properties (problems) in the language
- Ex.: actual and formal parameters (arguments) should match in a program
- Ex.: valid sentences in English ? How do we define the a language to be ambiguous…which is a bad thing in a programming language
  - How do we define, using a mathematical structure, what ambiguity means (in an unambiguous manner ☺)

4

Grammars: Definition

- Formal model (tool?) for describing and analyzing languages
  - Set of rules by which valid sentences/strings in a language are constructed
- Definition 1: A grammar G= (V,T,P,S) consists of:

  V: a finite set of ***variable*** or non-terminal symbols; each denotes a *property*

  T: a finite set of ***terminal*** symbols (the *alphabet*!)

  S: a variable called the ***start*** symbol

  P: a set of ***productions*** (also called production rules -- *rules of the grammar*)
- Example 1:

$$V = \{ S, A \}$$

$$T = \{ a, b \}$$

$$P = \{ S \rightarrow aSb,\ A \rightarrow \lambda \}$$

<sentence> = <noun phrase><verb phrase>

5

Grammars - comments

- Basic idea is to use “variables” to stand for sets of strings (i.e., languages)
  - Variable for <nouns>, <verbs>
- These variables are defined recursively, in terms of one another
- Recursive rules (Productions) involve only concatenation
  - Alternate rules for a variable allow union
- Production rule is of the form $x \rightarrow y$ where $x,y \in (V \cup T)^+$
  - Production rules are akin to the transition function of an automaton

6

Grammars: Derivation of Strings

- Beginning with the start symbol, strings are derived by repeatedly replacing string on left hand side symbols with the expression on the right-hand side of any applicable production
- Any applicable production can be used, in arbitrary order, until the string contains no variable symbols.
- Sample derivation using grammar in Example:

$S \Rightarrow aSb$ (applying first production)

$\Rightarrow aaSbb$ (applying first production)

$\Rightarrow aabb$ (applying second production)

7

The Language Generated by a Grammar

- Definition: For a given grammar G, the language generated by G, L(G), is the set of all terminal strings derived from the start symbol by using a sequence of production rules

8

Example: English Grammar

- A subset of the complete English grammar:

<sentence> → <subject> <verb phrase> <object>

<verb phrase> → <adverb> <verb> | <verb>

<object> → *the* <noun> | a <noun> | <noun>

<subject> → *This* | *Computers* | *I*

<adverb> → *never*

<verb> → *run* | *tell* | *am*

<noun> → *university* | *world* | *lies* | *cheese*

*"**or**" symbol*

Interpret rule as "<verb phrase> derives <adverb><verb> or <verb>"

Using above rules, we can derive sentences such as:

Computers run the world

object

the <noun>

world

Verb phrase

subject

I never tell lies

Computers run cheese

This am a lies

Grammars for Programming Languages

- The syntax of a programming language is described using grammars
  - Commonly referred to as Backus-Naur Form (BNF)
- in a hypothetical programming language,
  - Identifiers (id) consist of digits and the letters a, b, or c
  - Identifiers must begin with a letter and the rest can be digits or letters or empty
- Productions for a sample grammar:

```
<identifier> → <letter> <rest>
<rest> → <letter> <rest> | <digit> <rest> | λ
<letter> → a | b | c |…|z
<digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Snippet of syntax of your favorite prog. language

```
<statement> ::= <labeled-statement> | <expression-statement> |
                 <compound-statement> | <selection-statement> |
                 <iteration-statement> | <jump-statement>
```

```
<iteration-statement> ::= while ( <expression> ) <statement> |
               do <statement> while ( <expression> ) ; |
               for ( {<expression>}? ; {<expression>}? ;
                                  {<expression>}? ) <statement>
```

Parsing a program (checking for syntax errors) = determine if the program is generated by the grammar

11

Grammars and Language Classes

- By placing constraints on what type of productions are allowed, we define different language classes.
  - Regular Grammars
  - Context free grammars
  - Context sensitive grammars
  - …..

12

Regular Grammars

- In a right-linear grammar, at most one variable symbol appears on the right side of any production. If it occurs, it is the rightmost symbol.
- In a left-linear grammar, at most one variable symbol appears on the right side of any production. If it occurs, it is the leftmost symbol.
- A *regular grammar* is either right-linear or left-linear.
- Example: a regular (right-linear) grammar:

V = { S }, T = { a, b }, and productions S $\rightarrow$ abS | a

generates (ab)* a

13

Regular Grammars Generate Regular Languages

Theorem: It is always possible to construct a nfa to accept the language generated by a regular grammar G:

- Assume it is right linear
- Label the NFA start state with S and a final state V_f
- For every variable symbol V_i in G, create a nfa state and label it V_i
- For each production of the form $A \rightarrow aB$, label a transition from state A to B with symbol a
- For each production of the form $A \rightarrow a$, label a transition from state A to V_f with symbol a (may have to add intermediate states for productions with more than one terminal on RHS)

Given how much easier it is to write regular expressions vs regular grammars, the common approach to define a regular language is to use regular expressions

14

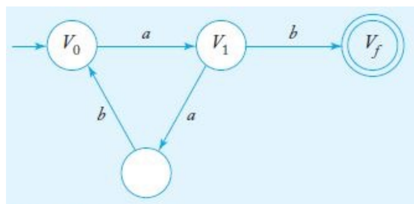
Example: Construction of a nfa to accept a language L(G)

Given the regular grammar G with productions

$V_0 \rightarrow aV_1$

$V_1 \rightarrow abV_0 \mid b$

NFA to accept L(G) can be constructed systematically



15

Grammars and Languages

- Regular Grammars = Regular Languages

- We have a context free grammar for L= $\{a^n b^n\}$ but we know that L is not regular:

  Context Free Grammars <> Regular Languages
  - DFAs cannot accept context free languages
  - PDAs accept context free languages

16

Context Free Grammars

- A context free grammar is a grammar G=(V,T,P,S) where all production rules are of the form: $V \rightarrow (V \cup T)^*$
  - Production rules have exactly one variable on the left and a string consisting of variables and terminals on the right.

V = { S }

T = { a, b }

P = { $S \rightarrow aSb$, $S \rightarrow \lambda$ }

generates $\{ a^n b^n \mid n >= 0 \}$

Example: Context-Free Grammar

- **Example**: $\{a^m b^n \mid m > n \geq 0\}$
- "Parallel" generation: generating two parallel parts; still from the "outside" to the "inside".
- Grammar ?

$S \rightarrow AC$ S generates more a's than b's

$C \rightarrow aCb \mid \lambda$ C generates equal number of a's and b's

$A \rightarrow aA \mid a$ A generates at least one a

Recall: Derivations

- Beginning with the start symbol, strings are derived by repeatedly replacing a variable in string on left hand side with the expression on the right-hand side of any applicable production
- We say $\alpha A\beta \Rightarrow \alpha\gamma\beta$ if $A \rightarrow \gamma$ is a production.

Example: S -> 01; S -> 0S1.

- S => 0S1 =>. 00S11 => 000111.
- we are replacing the occurrence of variable A in string $\alpha A\beta$ on LHS with the RHS of the production $A \rightarrow \gamma$
- =>* means "zero or more derivation steps."

19

19

Definition: Language Generated by a Grammar

- Definition: For a given grammar G, the language generated by G, L(G), is the set of all terminal strings derived from the start symbol by using a sequence of production rules
- If G is a CFG, then L(G), the *language of G*, is

  $L(G) = \{w \mid S \Rightarrow^* w \text{ and } w \text{ is a string over set } T\}$.
- Example: G has productions $S \rightarrow \lambda$ and $S \rightarrow 0S1$
  - $L(G) = \{0^n1^n \mid n \geq 0\}$.
- To show a language L is generated by G:
  - Show every string in L can be generated by G
  - Show every string generated by L is in G
- Definition: a string α is in *sentential form* if $S \Rightarrow^* \alpha$
  - The string α can derive a sentence in the language

20

Leftmost and Rightmost Derivations

- In a *leftmost derivation*, the leftmost variable in a sentential form is replaced at each step
- In a *rightmost derivation*, the rightmost variable in a sentential form is replaced at each step
- Consider the grammar :

  V = { S, A, B }, T = { a, b }, and productions

  $S \rightarrow aAB$

  $A \rightarrow bBb$

  $B \rightarrow A \mid \lambda$
- The string abb has two distinct derivations:
  - Leftmost: $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abbB \Rightarrow abb$
  - Rightmost: $S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abb$

Another way to represent derivations.....

- Graphs and trees anyone ??

Derivations as Parse Trees

- Represent derivation of a string as a (directed) Tree

- Why is this useful ?

23

Parse Trees, also known as Derivation Trees

- *Parse trees* are trees labeled by symbols of a particular CFG.
- Leaves: labeled by a terminal or λ.
- Interior nodes: labeled by a variable (occurring on LHS of production).
  - Children are labeled by the RHS body of a production for the parent.
- Root: must be labeled by the start symbol.

- Studying properties of a parse tree = properties of trees !

24

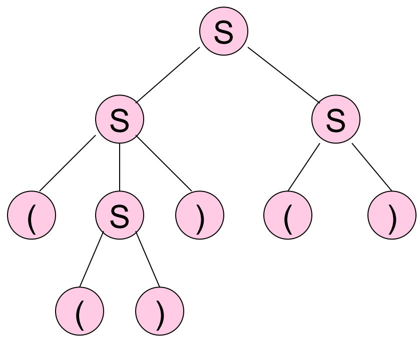
24

Example: Parse Tree

Grammar for correctly nested parenthesis

S -> SS | (S) | ( )

Parse tree for the string (( )) ( )

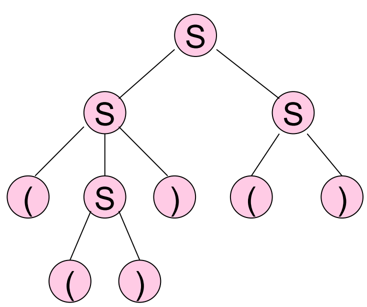


25

25

Yield of a Parse Tree

- The concatenation of the labels of the leaves in left-to-right order
  - That is, *in the order of a preorder traversal*.

  is called the *yield* of the parse tree.
- Example: yield of  is (( )) ( )

26

26

Example: Context-Free Grammar & Parse tree

- **Example**: $\{a^m b^n \mid m > n \geq 0\}$
- "Parallel" generation: generating two parallel parts; still from the "outside" to the "inside".
- Grammar ?

  S $\rightarrow$ AC   S generates more a's than b's

  C $\rightarrow$ aCb | λ C generates equal number of a's and b's

  A $\rightarrow$ aA | a   A generates at least one a

- Parse Tree for a^3b

Generalization of Parse Trees

- We sometimes talk about trees that are not exactly parse trees, but only because the root is labeled by some variable A that is not the start symbol.
- Call these *parse trees with root A.*

28

Equivalence of Parse Trees, Leftmost and Rightmost Derivations

- Trees, leftmost, and rightmost derivations correspondence
  for every leftmost/righmost derivation of string w,
  there is a unique parse tree with yield w
- We'll prove (next week!):
  1. If there is a parse tree with root labeled A and yield w, then
     A =>*$_{lm}$ w.
  1. If A =>*$_{lm}$ w, then there is a parse tree with root *A* and yield w.

29

29

Exercises

- 1. grammar G_2: $S \rightarrow aSa \mid bSb \mid \lambda$
  - Show parse tree for abba
- 2. Give a CFG for L= $\{a^m b^n \mid m \neq n,\ m,\ n \geq 0\}$
  - Apply your " non determinism skills" from PDA design!!
  - From start, go and generate either (a) more a's than b's
    or (b) more b's than a's
  - Within each path, "Parallel" generation: generating two parallel parts; still from the "outside" to the "inside".

30

Why do we need formal method – another motivation: Recall example

- What does this (English) sentence mean:
  "Oliver made Linnea duck"
- What does this (English) sentence mean:
  "Time flies like an arrow."

31

Why do we need formal method – another motivation: Recall example

- What does this (English) sentence mean:
  "Oliver made Linnea duck"

  1. duck is a noun

  2. duck is a verb
- What does this (English) sentence mean:
  "Time flies like an arrow."

  1. flies is a verb: meaning = "time goes by fast"

  2. flies is a noun (like house flies):
  meaning = this species of flies like arrows,
  similar to "house flies like an apple"

Ambiguity – sentence has more than one meaning

32

Ambiguous Grammars: Definition

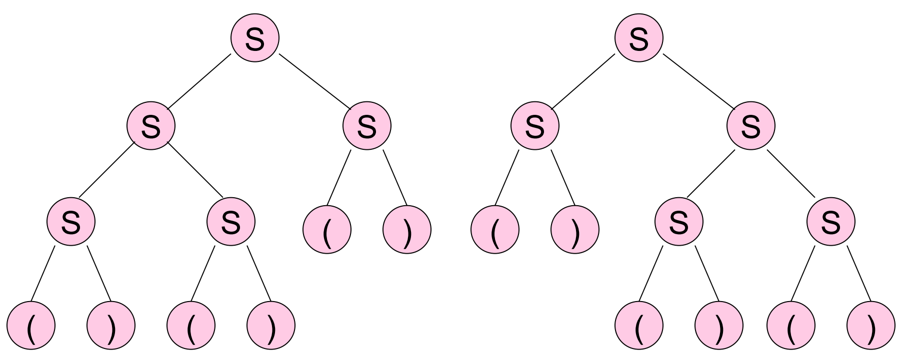
- A CFG is *ambiguous* if there is a string in the language that is the yield of two or more parse trees.
  - Equivalent: If there is more than one leftmost (or rightmost) derivation for a string in the language
- Why is ambiguity a problem ?....in a programming language?

33

33

Example

Example: S -> SS | (S) | ( )
Two parse trees for ( ) ( ) ( )



34

34

Ambiguity is a Property of Grammars, not Languages

- For the balanced-parentheses language, here is another CFG, which is unambiguous.

B -> (RB | λ

R ->) | (RR

B, the start symbol, derives balanced strings.

R generates certain strings that have one more right paren than left.

35

35

Example: Unambiguous Grammar

B –> (R B | λ

R –>) | (R R

- Construct a unique leftmost derivation for a given balanced string of parentheses by scanning the string from left to right.
  - If we need to expand B, then use B -> (RB if the next symbol is "("; use λ if at the end.
  - If we need to expand R, use R ->) if the next symbol is ")" and (RR if it is "(".

36

36

Inherent Ambiguity

- It would be nice if for every ambiguous grammar, there were some way to "fix" the ambiguity, as we did for the balanced-parentheses grammar.
- Unfortunately, certain languages (including CFL's) are *inherently ambiguous*, meaning that every grammar for the language is ambiguous.
- English is an ambiguous language
  - Proof ??

37

37

Interesting Question on Ambiguity

- Given a CFG G, can we determine if the language is inherently ambiguous ?
  - Oops!
- Programming language syntax
  - Unambiguous grammars
  - Or semantic rules to resolve ambiguity
    - Ex: Precedence rules in expressions a+a*a ?

38

Next….the quest for "automation"!

- We would like to answer the question "Does G derive string w" *i.e.,* Is w generated by the grammar?
  - Ex: Given the grammar of Python and a program in Python, does the program satisfy all the rules of the grammar.
- Design an algorithm that takes as input the grammar G and string w, and outputs the parse tree for w or returns "syntax error"
- How do we proceed ?
  - Grammars seem to be built "arbitrarily"
    - Algorithm should handle all possible representations
    - Complicates the algorithm

39

Next….Simplification and Parsing and …

1. provide a procedure to simplify a grammar such that:
   - Resulting grammar generates the same language
   - Resulting grammar has production rules in a specific format
     - Simplifies the data structures and the parsing algorithm
2. Normal Forms: specifications on how the production rules must be defined
   - Chomsky Normal Form (CNF)
   - Greiback Normal Form (GNF)
3. Parsing Algorithm: Design a parsing algorithm (CYK algorithm) that takes a grammar in a standard form (CNF) and checks if string is generated by the grammar
4. Equivalence of PDAs and CFGs
5. Properties of Context Free Languages

40