CSCI 742 - Compiler Construction

Lecture 9

Ambiguous Grammars

Instructor: Hossein Hojjat

February 5, 2018

"Fighting tigers can be dangerous"

... Let's talk about ambiguity!

Parse Tree

- Context-Free Grammar (CFG) is a 4-tuple $G = (T, N, S, R)$
- Parse trees are trees where
  - root is labeled with the start symbol S
  - internal nodes are labeled with symbols $\in N$
  - leaf nodes are labeled with symbols $\in T \cup \{\epsilon\}$
  - if v is a node with label X and its child nodes $v_1, \cdots, v_n$ are labeled with $X_1, \cdots, X_n$ then

$$X \to X_1 \cdots X_n \text{ is a production rule } \in R$$

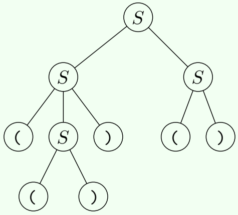
Example.

Grammar: $G = (\{\mathbf{(}, \mathbf{)}\}, \{S\}, S, R)$ where

$$R = \left\{ S \to \ SS \ \mid \mathbf{(}S\mathbf{)} \mid \mathbf{()} \right\}$$

Derivation:

$$S \Rightarrow SS \Rightarrow \mathbf{(}S\mathbf{)}S \Rightarrow \mathbf{(())}S \Rightarrow \mathbf{(())()}$$



Leftmost and Rightmost Derivations

$$S \to \ SS \ \mid \mathbf{(}S\mathbf{)} \mid \mathbf{()}$$

With this grammar there is a choice of variables to expand

Sample derivation:

$$S \Rightarrow SS \Rightarrow SSS \Rightarrow S\mathbf{()}S \Rightarrow S\mathbf{()()} \Rightarrow \mathbf{()()()}$$

Leftmost derivation: always expand the leftmost variable first

$$S \Rightarrow SS \Rightarrow SSS \Rightarrow \mathbf{()}SS \Rightarrow \mathbf{()()}S \Rightarrow \mathbf{()()()}$$

Rightmost derivation: always expand the rightmost variable first

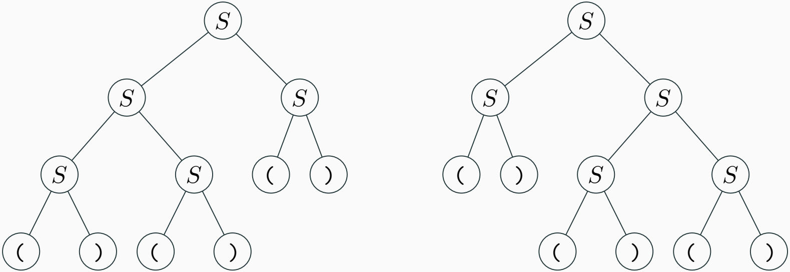
$$S \Rightarrow SS \Rightarrow SSS \Rightarrow SS\mathbf{()} \Rightarrow S\mathbf{()()} \Rightarrow \mathbf{()()()}$$

Ambiguous Grammars

- Ambiguous CFG:
  there is a word in the language that has two or more parse trees
- Example:

$$S \rightarrow \; SS \;\mid \mathbf{(}S\mathbf{)} \mid \mathbf{()}$$

Two parse trees for **()()()**



Ambiguity, Left- and Rightmost Derivations

- To show that a grammar is ambiguous:

1) Give two different parse trees for a word, or
2) Give two different leftmost derivations for a word, or
3) Give two different rightmost derivations for a word

- One leftmost and one rightmost derivation for a word is not sufficient
- Leftmost and rightmost derivations might correspond to the same parse tree

One leftmost and one rightmost is Insufficient

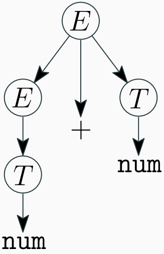
- Grammar for additive arithmetic expressions:

$$E \to E + T$$
$$E \to T$$
$$T \to \mathrm{num}$$

- Derivation for $\mathtt{num} + \mathtt{num}$:

<u>Leftmost Derivation:</u>

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow T + T \\ &\Rightarrow \mathrm{num} + T \\ &\Rightarrow \mathrm{num} + \mathrm{num} \end{aligned}$$



<u>Rightmost Derivation:</u>

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow E + \mathrm{num} \\ &\Rightarrow T + \mathrm{num} \\ &\Rightarrow \mathrm{num} + \mathrm{num} \end{aligned}$$

Ambiguity is Bad

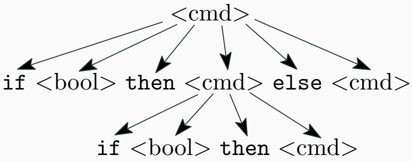
- Sometimes ambiguity in grammar can leave meaning of some programs ill-defined

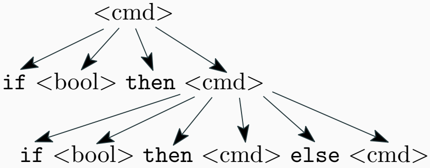
Example: <cmd> ::= `if` <bool> `then` <cmd>

| `if` <bool> `then` <cmd> `else` <cmd>

- Do not know if `else` clause is paired with the outermost or with the innermost `then`

```
if (x > 0) then
  if (y > 0) then
    print(1)
  else
    print(2)
```





Ambiguity

- Ambiguity is a property of grammars not languages
- For the balanced parentheses language, here is another CFG which is unambiguous:

$$B \rightarrow \mathbf{(}RB \mid \epsilon$$
$$R \rightarrow \mathbf{)} \mid \mathbf{(}RR$$

- Start symbol B generates balanced strings
- R generates strings that have one more right parentheses than left

Example: Unambiguous Grammar

$$B \to \textbf{(}RB \mid \epsilon$$
$$R \to \textbf{)} \mid \textbf{(}RR$$

- This grammar constructs a unique leftmost derivation for a given balanced string of parentheses
- When scanning the input string from left to right:
- If we need to expand B:
  - If the next symbol is **(** then use $B \to \textbf{(}RB$
  - If it is at the end then use $B \to \epsilon$
- If we need to expand R
  - If the next symbol is **)** then use $R \to \textbf{)}$
  - If the next symbol is **(** then use $R \to \textbf{(}RR$

Ambiguity

Theorem
The problem of deciding whether a given CFG is ambiguous is undecidable

- Bad news:
  There is no general algorithm to remove ambiguity from a CFG
- More bad news:
  Some CFL's have only ambiguous CFG's
- CFL L is inherently ambiguous if all grammars for L are ambiguous
- There are heuristics that can be used to remove ambiguity from a grammar

Inherent Ambiguity

- Parikh first proved the existence of context-free, inherently ambiguous languages (1961)
- He proved the inherent ambiguity of

$$M = \{a^i b^j a^i b^k \mid i, j, k \geq 1\} \cup \{a^i b^j a^k b^j \mid i, j, k \geq 1\}$$

Inherent Ambiguity: Example

- $L = \{0^i 1^j 2^k \mid i = j \text{ or } j = k\}$
- Intuitively strings of the form $0^n 1^n 2^n$ can be generated by two different parse trees:

- one checks that the number of 0's and 1's are equal,
- the other one checks that the number of 1's and 2's are equal

Inherent Ambiguity: Example

One Possible Ambiguous Grammar for $L = \{0^i 1^j 2^k \mid i = j \text{ or } j = k\}$

$$
\begin{aligned}
S &\to AB \mid CD \\
A &\to 0A1 \mid 01 \\
B &\to 2B \mid 2 \\
C &\to 0C \mid 0 \\
D &\to 1D2 \mid 12
\end{aligned}
$$

- A generates equal numbers 0's and 1's
- B generates any number of 2's
- C generates any number of 0's.
- D generates equal numbers 1's and 2's

Inherent Ambiguity: Example

One Possible Ambiguous Grammar for $L = \{0^i 1^j 2^k \mid i = j \text{ or } j = k\}$

$$
\begin{aligned}
S &\to AB \mid CD \\
A &\to 0A1 \mid 01 \\
B &\to 2B \mid 2 \\
C &\to 0C \mid 0 \\
D &\to 1D2 \mid 12
\end{aligned}
$$

- There are two derivations of every string with equal numbers of 0's, 1's and 2's

$$S \Rightarrow AB \Rightarrow 01B \Rightarrow 012$$
$$S \Rightarrow CD \Rightarrow 0D \Rightarrow 012$$

Ambiguity Exercise

Question

Show that the following grammar is ambiguous:

$$A \to BC$$

$$B \to 1B1 \mid 1$$

$$C \to 1C1 \mid \epsilon$$

Ambiguity Exercise

Question

Show that the following grammar is ambiguous:

$$A \to BC$$
$$B \to 1B1 \mid 1$$
$$C \to 1C1 \mid \epsilon$$

Answer

Two different leftmost derivations for 111

- $$A \Rightarrow BC \Rightarrow 1C \Rightarrow 11C1 \Rightarrow 111$$
- $$A \Rightarrow BC \Rightarrow 1B1C \Rightarrow 111C \Rightarrow 111$$

Chomsky Normal Form

- Consider the grammar $G_\epsilon = (\emptyset, \{S\}, S, R)$ with the following production rules

$$S \to SSSSS \mid \epsilon$$

- Grammar is obviously ambiguous
- It has infinitely many parse trees which can be arbitrarily large!

Chomsky Normal Form

- Bad news: we cannot eliminate ambiguity from CFGs in general
- Good news: we can at least eliminate the possibility to have infinitely many parse trees for a given string
- There is an equivalent grammar in Chomsky Normal Form (CNF) for any context-free grammar
- Grammar in CNF guarantees
  - every string has a finite number of parse trees
  - every parse tree for a given string has the same size (binary tree)

Chomsky Normal Form (CNF)

A CFG is in Chomsky Normal Form if each rule is of the form

$$A \to BC$$
$$A \to a$$

where

- a is any terminal
- A,B,C are non-terminals
- B, C cannot be start variable

We allow the rule $S \to \epsilon$ if $\epsilon \in L$

Example

- For the balanced parentheses language,

$$S \rightarrow \ SS \ \mid \mathbf{(}S\mathbf{)} \mid \mathbf{()}$$

- Equivalent Chomsky Normal Form (CNF) grammar (S_0 is start symbol):

$$\begin{aligned} S_0 &\rightarrow SS \mid LA \mid LR \\ S &\rightarrow SS \mid LA \mid LR \\ A &\rightarrow SR \\ L &\rightarrow \mathbf{(} \\ R &\rightarrow \mathbf{)} \end{aligned}$$

- Any context-free grammar can be converted through an algorithm into one in Chomsky Normal Form
  - We will discuss this in more detail later in the course