COSE215: Theory of Computation

Lecture 10 — Parse Trees and Ambiguity

Hakjoo Oh
2019 Spring

Tree Representation for Derivations

Consider the context-free grammar for expressions:

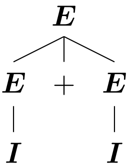
$$G = (\{E, I\}, \{+, *, (,), a, b, 0, 1\}, E, P)$$

$$\begin{array}{rcl} E & \rightarrow & I \mid E + E \mid E * E \mid (E) \\ I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array}$$

A (partial) derivation:

$$E \Rightarrow E + E \Rightarrow I + E \Rightarrow I + I.$$

The tree representationfor the derivation:



The tree is called *parse tree* or *derivation tree*.

Derivation and Parse Tree

- A derivation uniquely defines a parse tree? yes or no.
- A parse tree uniquely defines a derivation? yes or no.

Formal Definition

Definition (Parse Trees)

Let $G = (V, T, S, P)$ be a grammar. The *parse trees* for G are trees with the following conditions:

1. The root is S, the start variable.
2. Each interior node is labeled by a variable in V.
3. Each leaf is labeled by either a variable, a terminal, or ϵ. However, if the leaf is labeled ϵ, it must be the only child of its parent.
4. If an interior node is labeled A, and its children are labeled

$$X_1, X_2, \ldots, X_k$$

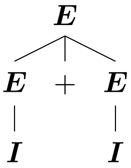
respectively, from the left, then $A \to X_1, X_2, \ldots, X_k$ is a production in P.

Example 1: Expressions

$$G = (\{E, I\}, \{+, *, (,), a, b, 0, 1\}, E, P)$$

$$\begin{array}{rcl} E & \rightarrow & I \mid E + E \mid E * E \mid (E) \\ I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array}$$

A parse tree:

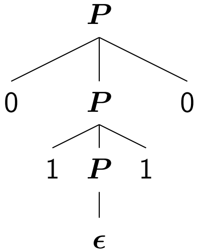


Example 2: Palindromes

$$G = (\{P\}, \{0, 1\}, P, A)$$

$$P \rightarrow \epsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$$

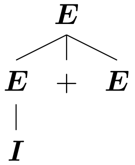
A parse tree:

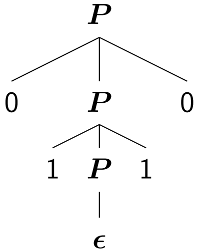


Yields

Definition (Yields)

The string obtained by concatenating the leaves of a parse tree from the left is called the *yield* of the tree.





Relationship between Parse Trees and Derivations

Theorem

Let $G = (V, T, S, P)$ be a context-free grammar. Then, the following are equivalent:

1. $S \Rightarrow^* w$.
2. $S \Rightarrow^*_{lm} w$.
3. $S \Rightarrow^*_{rm} w$.
4. *There is a parse tree whose yield is w.*

Ambiguous and Unambiguous Grammars

Definition

A context-free grammar is *ambiguous* if there exists some $w \in L(G)$ that has at least two distinct parse trees. If each string has at most one parse tree, the grammar is *unambiguous*.

Theorem

For each grammar $G = (V, T, S, P)$ *and string* $w \in T^*$, w *has two distinct parse trees if and only if* w *has two distinct leftmost derivations from* S.

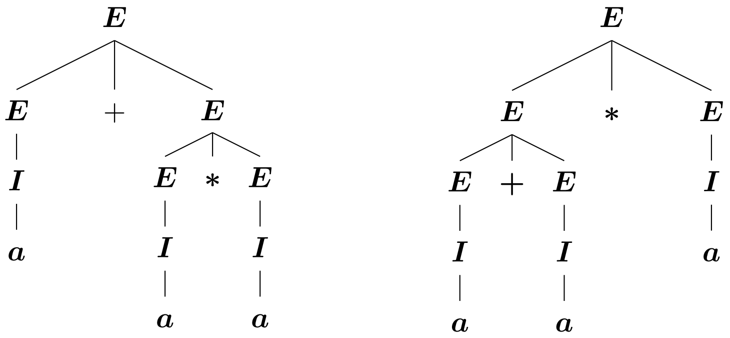
Example

The grammar of expressions:

$$G = (\{E, I\}, \{+, *, (,), a, b, 0, 1\}, E, P)$$

$$\begin{array}{rcl} E & \rightarrow & I \mid E + E \mid E * E \mid (E) \\ I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array}$$

Two distinct parse trees for $a + a * a$:



Example

The grammar of expressions:

$$G = (\{E, I\}, \{+, *, (,), a, b, 0, 1\}, E, P)$$

$$\begin{array}{rcl} E & \rightarrow & I \mid E + E \mid E * E \mid (E) \\ I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array}$$

Two distinct leftmost derivations for $a + a * a$:

- $E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + I * E \Rightarrow a + a * E \Rightarrow a + a * I \Rightarrow a + a * a$
- $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow I + E * E \Rightarrow a + E * E \Rightarrow a + I * E \Rightarrow a + a * E \Rightarrow a + a * I \Rightarrow a + a * a$

General Facts

We would like to transform ambiguous grammars into unambiguous ones. However,

- There is no algorithm to remove ambiguity from a CFG.
- There is no algorithm that can even tell us whether a CFG is ambiguous or not.
- There are context-free languages that are inherently ambiguous; for these languages, removing the ambiguity is impossible.

Finding an unambiguous grammar is possible in practice

- Fortunately, for the sorts of constructs that appear in common programming languages, there are well-known techniques to eliminate ambiguity.
- An ambiguous grammar:

$$
\begin{array}{rcl}
E & \rightarrow & I \mid E + E \mid E * E \mid (E) \\
I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{array}
$$

Finding an unambiguous grammar is possible in practice

- Fortunately, for the sorts of constructs that appear in common programming languages, there are well-known techniques to eliminate ambiguity.
- An ambiguous grammar:

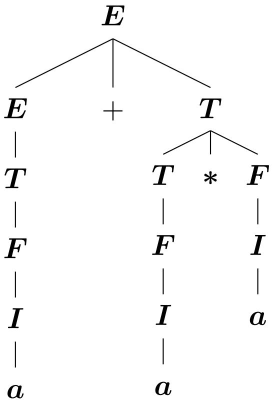
$$\begin{array}{rcl} E & \rightarrow & I \mid E+E \mid E*E \mid (E) \\ I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array}$$

- An equivalent but unambiguous grammar:

$$\begin{array}{rcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow & I \mid (E) \\ T & \rightarrow & F \mid T*F \\ E & \rightarrow & T \mid E+T \end{array}$$

Example

The only parse tree for $a + a * a$:



Eliminating Ambiguity

Let us first analyze why the following grammar is ambiguous.

$$\begin{array}{rcl} E & \rightarrow & I \mid E + E \mid E * E \mid (E) \\ I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array}$$

- The grammar does not respect the precedence of operators.
  - Conventionally, we give higher precedence to $*$ over $+$, e.g., $1 + 2 * 3$.
  - We need to enforce the common precedence rule in the grammar.
- The grammar does not respect the associativity of operators.
  - We assume that operators associate to the left, e.g., $1 + 2 + 3$ should be parsed as $(1 + 2) + 3$.
  - We need to enforce the common associativity rule in the grammar.

Eliminating Ambiguity

To enforce precedence, classify expressions into *factors*, *terms*, and *expressions*:

- A factor is either an identifier or a parenthesized expressions, e.g.,

  $$a, b, (a + b), (a * b), \ldots$$

  In grammar:

  $$F \to I \mid (E)$$

- A term is either a produce of one or more factors, e.g.,

  $$a, b, (a + b), (a * b), a * b, a * (a + b), a * (a * b), \ldots$$

  In grammar

  $$T \to F \mid T * F$$

- An expression is a sum of one or more terms, e.g.,

  $$a * b, a * (a + b), a * b + a * (a + b), \ldots$$

  In grammar

  $$E \to T \mid E + T$$

Eliminating Ambiguity

The unambiguous grammar:

$$\begin{array}{rcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow & I \mid (E) \\ T & \rightarrow & F \mid T * F \\ E & \rightarrow & T \mid E + T \end{array}$$

Questions:

- How does it enforce left associativity?
- How can we modify the grammar to enforce right associativity?
- Is the above grammar really unambiguous?

Inherent Ambiguity

- In the previous example, ambiguity is involved in the grammar.
- For some languages, ambiguity is inherent as it is involved in the language itself. In this case, all of the grammars for the language are ambiguous.

Definition

A language L is *inherently ambiguous* if every grammar that generates L is ambiguous.

Example

Consider the language:

$$L = L_1 \cup L_2$$

where

$$L_1 = \{a^n b^n c^m \mid n, m \geq 0\}, \quad L_2 = \{a^n b^m c^m \mid n, m \geq 0\}$$

A context-free grammar:

$$\begin{array}{rcl} S & \to & S_1 \mid S_2 \\ S_1 & \to & S_1 c \mid A \\ A & \to & aAb \mid \epsilon \\ S_2 & \to & aS_2 \mid B \\ B & \to & bBc \mid \epsilon \end{array}$$

- Why is the grammar ambiguous?

Summary

- Context-free grammars: A way of describing languages by recursive rules called productions.
- Derivations: Beginning with the start symbol, we can derive terminal strings by repeatedly applying production rules.
- Context-free languages: The language of a CFG is the set of terminal strings that can be derived. Such a language is called context-free.
- Parse trees: A tree representation for a derivation.
- Ambiguous grammars: Grammars that have two different parse trees for a terminal string.
- Eliminating ambiguity: For many useful grammars, it is possible to find unambiguous grammars. However, the unambiguous grammar is typically more complex than the original.
- Inherent ambiguity: There are some context-free languages that do not have unambiguous grammars.