CS304: Automata and Formal Languages

Lec 15

Parsers

Rachit Nimavat

September 8, 2025

Outline

The World of Regular Languages

Three Faces of Same Power

The Limit: What Can't Finite Automata Do?

Consider the language $L = \{a^n b^n \mid n \geq 0\} = \{\epsilon, ab, aabb, aaabbb, \dots\}$.

To accept L, a DFA/NFA must **count** the number of a's

DFA/NFA has only a finite number of states and cannot remember an arbitrarily large count n.

Next Step?

We need machine with more power:

- Context-Free Grammars: A way to describe languages with recursive structure
- Pushdown Automata: NFA + Stack

Example

Consider $L = \{\text{balanced parentheses}\}$ over $\Sigma = \{(,)\}$. e.g. $()(()) \in L$ but $)(\notin L$.
Let's build a grammar!

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

Derivation

is a sequence of steps applying the production rules.
Generating string ( )(( )):

$$\begin{aligned} S &\to \mathsf{SS} \\ &\to \mathsf{(\,)S} \\ &\to \mathsf{(\,)(S)} \\ &\to \mathsf{(\,)(\,(\,)\,)} \end{aligned}$$

Defn: CFG

A CFG is a set of recursive rules used to generate patterns of strings. Think of it as a blueprint for a language.

Definition 1.1

A Context-Free Grammar is a 4-tuple $G = (V, T, P, S)$, where:

V: Finite set of **variables**. think: syntactic categories or placeholders. (e.g., S, E, T in previous examples)

T: Finite set of **terminals** or **tokens**. actual symbols or words of the language. (e.g., $\mathsf{a}, \mathsf{b}, \mathsf{c}, +, \star$ in previous examples)

P: Finite set of **production rules**. Rules have the form $A \to \alpha$, where $A \in V$ and $\alpha \in (V \cup T)^*$, i.e., α is a string consisting of variables and terminals. specifies how to replace variables with strings of variables and terminals. (e.g., $S \to \mathsf{a}S\mathsf{b}$ and $E \to T$) in previous examples

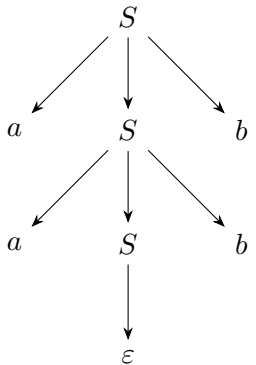
S: The **start symbol** ($S \in V$).

Visualizing Derivations: Parse Trees

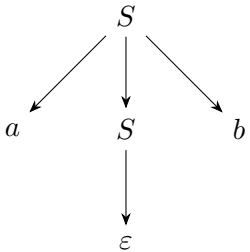
Similar to DFA, wer have **Parse Trees** to visualize derivation in CFGs.
Consider $L = \{a^n b^n \mid n \geq 0\}$ over $\Sigma = \{a, b\}$ and CFG $S \to \varepsilon \mid \mathtt{a}S\mathtt{b}$.

Parse tree for ε:

The parse tree for $aabb$:



Parse tree for ab:



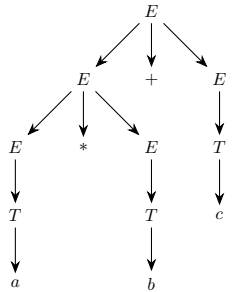
Definition 1.2

A CFG G is **ambiguous** if there exists a string $w \in L(G)$ such that w has more than one distinct parse trees.

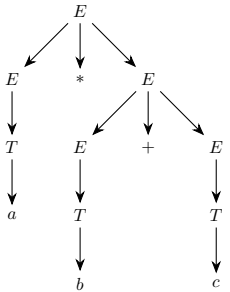
Example - Revisted

Consider the language of expressions over $\Sigma = \{a, b, c, +, *\}$ with CFG:
$E \to E + E \mid E * E \mid T$ and $T \to$ a | b | c.

Parsing $a * b + c$:



Parsing $a * b + c$:



Resolving Ambiguity

Enforcing Precedence and Associativity

Learning from our maths classes, we can enforce precedence and associativity rules in our CFGs to resolve ambiguity. Won't work for all ambiguities though!

The Solution:

Rewrite G with a hierarchy of rules

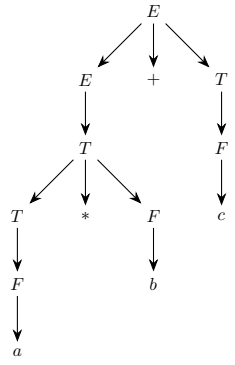
Forces a single, correct interpretation.

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

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

$$F \to \mathsf{a} \mid \mathsf{b} \mid \mathsf{c}$$

Unique parse tree of $a * b + c$:



Resolving Ambiguity

Enforcing Precedence and Associativity

The Solution:

Rewrite G with a hierarchy of rules

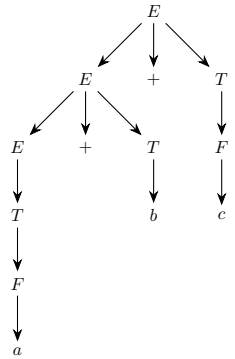
Forces a single, correct interpretation.

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

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

$$F \to \mathsf{a} \mid \mathsf{b} \mid \mathsf{c}$$

Unique parse tree of $a + b + c$:



Bonus:

Multiplication is given more precedence than addition What enforces this in G?

Operators are evaluated from left to right What enforces this in G?

Algorithm for Ambiguity Resolution?

No such algorithm!

No algorithm that given a CFG G, decides whether G is ambiguous or not.

There are CFLs for which all possible CFGs are ambiguous.

See, Chapter 5.4 in ALC for more details.

Derivation and Inference

Derivation

- start from the start symbol and apply production rules to derive a string in the language
- goes from root to leaves in parse trees
- $S \to \alpha_1 \to \alpha_2 \ldots \to w$, where w is a string in the language
- shorthand: $S \overset{*}{\Rightarrow} w$ if such a chain exists

  $\to$ denotes a production rule,
  $\Rightarrow$ denotes a single step in derivation,
  and $\overset{*}{\Rightarrow}$ denotes zero-or-more steps of derivation

Inference

- compute the language for each individual variables (a.k.a. **non-terminals**)
- start from a string and apply production rules to infer the non-terminals that can generate it
- can be thought of as the reverse process of derivation
- don't worry about it for this course

Homework

See Chapter 5.2 in ALC to see proofs of equivalence between leftmost/rightmost derivations and parse trees.

Parsers

- CFGs describe programming languages
- You'll learn more about this in the compilers course
- There's an algorithm to convert a CFG to a parser
- There are two main types of parsers:

  top-down parsers. think: derivations (e.g., recursive descent)
  bottom-up parsers. think: recursive inference (e.g., LR parsers)

Example

Parsing **if** and **else** clauses in C. **if** may appear balanced by an **else** clause or may appear unbalanced. **else** clause must be matched to a preceding **if** clause.

$$S \to \varepsilon \mid \mathtt{i}S \mid \mathtt{i}S\mathtt{e} \mid SS$$

`ieie`, `iie`, `iei` are valid strings generated by this CFG.

`ei`, `ieeii`, `e` cannot be generated by this CFG.

For more examples and applications involving YACC, XML, YAML, and more, wait until compilers course!

HW: Is this CFG ambiguous?

CFGs in the Wild

Syntax, Semantics, and Chomsky

Noam Chomsky

Noam Chomsky (1928-) is a linguist, cognitive scientist, and political activist

Revolutionized linguistics by proposing that human languages have an underlying structure that can be modeled using formal grammars in 1950s

Syntax vs. Semantics CFGs can mostly capture syntax of a language, while other formalisms (e.g., lambda calculus) capture semantics.

The dog runs. The dogs run. The verb is not context-free depends on whether the subject is singular or plural

Colorless green ideas sleep furiously. Grammatically correct but no logical meaning