CS304: Automata and Formal Languages

Lec 16

CNF and PDA

Rachit Nimavat

September 11, 2025

Outline

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

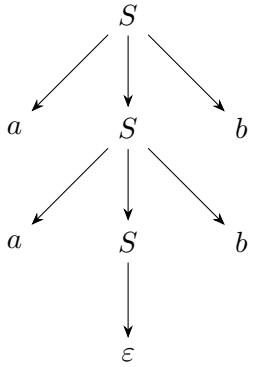
S: The **start symbol** ($S \in V$).

Visualizing Derivations: Parse Trees

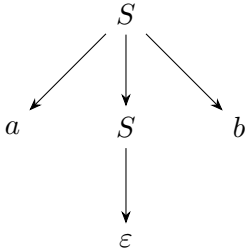
Similar to DFA, wer have **Parse Trees** to visualize derivation in CFGs.
Consider $L = \{a^n b^n \mid n \geq 0\}$ over $\Sigma = \{a, b\}$ and CFG $S \to \varepsilon \mid \mathtt{a}S\mathtt{b}$.

Parse tree for ε:

The parse tree for $aabb$:



Parse tree for ab:



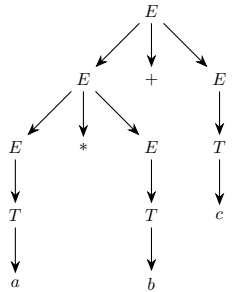
Definition 1.2

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

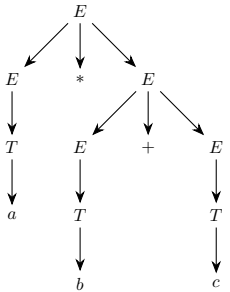
Example - Revisted

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

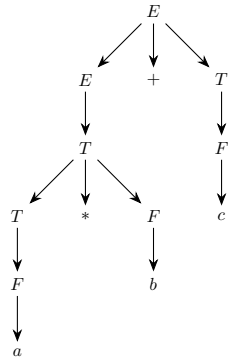
Forces a single, correct interpretation.

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

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

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

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



Resolving Ambiguity

Enforcing Precedence and Associativity

The Solution:

Rewrite G with a hierarchy of rules

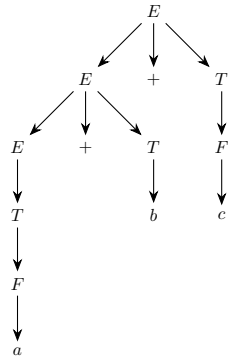
Forces a single, correct interpretation.

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

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

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

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.

Simplifying CFGs

So far, our grammars contained complex, redundant, and/or useless rules

Messy grammars are hard to analyze and convert into parsers. How can we be sure a grammar doesn't contain infinite loops or dead-end rules?

Standardize CFGs! This makes grammars easier to work with, both for humans and for algorithms

This is the first step for industrial-grade parsers

Useless Productions and Symbols

Non-Generating Symbols. A variable A is non-generating if it can never derive a string of only terminals. It's a dead end.

e.g. $S \to aA \mid b$ and $A \to aA$

How to identify these?

Identify generating symbols. Iteratively find all variables that produce a string of terminals and/or generating symbols. Any variable that is not marked generating is **non-generating**.

Unreachable Symbols. A variable A is unreachable if there is no derivation from the start variable S that uses A.

e.g. $S \to aS \mid b$ and $A \to c$

How to identify these?

Start with the reachable symbol: the start symbol. Iteratively add all symbols that appear on the right-hand side of productions for any already-reachable variable. Any variable that is not marked reachable is **non-reachable**.

Chomsky Normal Form (CNF)

Goal: Convert any CFG to a standard form without changing its language.

Definition 2.1

A CFG in CNF has production rules of the form:

- $A \to BC$ Variable yields two variables. Note B or C might be equal or equal to A.
- $A \to a$ Variable yields a single terminal.
- Does not have any useless symbols.
- The empty string ε is not allowed. If $\varepsilon \in L$, add a special rule $S \to \varepsilon$.

Benefits:

- Clean and Standard grammar
- No messy edge-cases, only two types of rules
- Great for theoretical analysis and writing algorithms

CFG to CNF Example

Balanced parenthesis: $S \to \texttt{()} \mid \texttt{(S)} \mid \texttt{SS}$

On Board!

CFG to CNF Example 2

$S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$

HW!

Equivalent of a DFA for CFGs?

Benefits of Hindsight

Consider a CFG $S \to () \mid (S) \mid SS$ of the language L of balanced parentheses.
How to build a machine that accepts L?

```
# Returns true if w is in the language L
def recognize(w):
    stack = []
    for c in w:
        if c == '(': stack.append(c)
        elif c == ')':
            if not len(stack):  return False
            stack.pop()
    return not len(stack)
```

Stripping the Syntax

Read-only, one-way input (just like an NFA)

Finite states: While no explicit state variable, we go in "reject state" once the stack is empty

The stack: New component!

Transition rules: What to do based on the current input symbol and the top of the stack

This machine (NFA + stack) is a Pushdown Automaton (PDA)!

Pushdown Automaton (PDA): Formal Definition

Definition 3.1

PDA is a 7-tuple: $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where:

- Q is a finite set of states
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accepting states
- Σ is the alphabet
- Γ is the stack alphabet
- $Z_0 \in \Gamma$ is the initial stack symbol
- δ is the transition function
  - $\delta : Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$ is the transition function
  - $\delta(q, a, x)$ denotes what to do when we are in state q, read a, and the top of the stack is x: tells us which state to move to, and what stack alphabet symbols to replace x with in the stack

Homework!

Write the PDA for recognizing the language of balanced parentheses.