ChatGPT

# Closure Properties of Context-Free Grammars (CFGs)

## Introduction

Context-free grammars (CFGs) generate **context-free languages (CFLs)**, which are recognized by pushdown automata. *Closure properties* describe which operations on languages will always produce another language of the same class. In other words, if a class of languages (such as CFLs) is **closed under** an operation, that means applying that operation to any languages in the class results in a language that is also in the class [1]. Knowing the closure properties of CFLs is useful because it helps us determine if a language formed by operations (like union or intersection) will remain context-free or not [1].

CFGs (or CFLs) turn out to be closed under some common operations but not others. Below, we summarize which operations **preserve** the context-free nature and which do not, with explanations and examples for each.

## Operations That Preserve Context-Free Languages

The class of context-free languages is closed under several fundamental operations. This means if we take one or more CFLs and perform these operations, the resulting language will **always** be context-free. We will discuss each such operation and illustrate with simple grammars.

### Union

**Claim:** If $L_1$ and $L_2$ are context-free languages, then their union $L_1 \cup L_2$ is also context-free [2] [3].

**Reasoning:** We can construct a context-free grammar for $L_1 \cup L_2$ by combining grammars for $L_1$ and $L_2$. Let $G_1$ be a CFG for $L_1$ with start symbol $S_1$, and $G_2$ be a CFG for $L_2$ with start symbol $S_2$. Assume $G_1$ and $G_2$ have disjoint sets of variables (nonterminals) – if not, we can rename them. Then a grammar for the union can be built by introducing a new start symbol S with a rule that allows deriving either an $L_1$ string or an $L_2$ string:

```
S -> S1 | S2
```

Here, $S_1$ generates strings in $L_1$ and $S_2$ generates strings in $L_2$. Using this combined grammar, any string generated is from either $L_1$ or $L_2$, so the language of this grammar is exactly $L_1 \cup L_2$ [4]. This proves closure under union.

**Example:** Suppose - $L_1$ = { a^n b^n ∣ n ≥ 0 } (equal numbers of `a` and `b`), and - $L_2$ = { c^n d^n ∣ n ≥ 0 } (equal numbers of `c` and `d`).

Both $L_1$ and $L_2$ are context-free (each can be generated by a simple CFG). The union $L_1 \cup L_2$ = { a^n b^n **or** c^n d^n ∣ n ≥ 0 } is also context-free. We can construct a grammar:

```
S  -> S1 | S2
S1 -> a S1 b | ε    // generates L1 (strings of a^n b^n)
S2 -> c S2 d | ε    // generates L2 (strings of c^n d^n)
```

This grammar's language is all strings of the form a^n b^n or c^n d^n, i.e. $L_1 \cup L_2$. Thus union is confirmed to preserve context-free languages [5] [6].

### Concatenation

**Claim:** If $L_1$ and $L_2$ are CFLs, then the concatenation $L_1L_2$ (all strings formed by a string from $L_1$ followed by a string from $L_2$) is also a CFL [7] [8].

**Reasoning:** Given CFGs $G_1$ for $L_1$ (start symbol $S_1$) and $G_2$ for $L_2$ (start symbol $S_2$) with disjoint variables, we can construct a grammar for the concatenation by using a new start symbol S that derives an $L_1$ string followed by an $L_2$ string. We add a production:

```
S -> S1 S2
```

When $S_1$ generates a string from $L_1$ and $S_2$ generates a string from $L_2$ in sequence, the combined result is in $L_1L_2$. All derivations from S will produce a string from $L_1$ then a string from $L_2$, so the language of this grammar is $L_1L_2$ [8].

**Example:** Using the same $L_1$ and $L_2$ as above (the a^n b^n and c^n d^n languages), their concatenation $$L_1 L_2 = \{ a^n b^n c^m d^m \mid m \geq 0,\, n \geq 0 \}$$ is context-free. A grammar for $L_1L_2$ can be obtained by extending the previous grammars:

```
S  -> S1 S2
S1 -> a S1 b | ε     // generates L1
S2 -> c S2 d | ε     // generates L2
```

For instance, S1 might generate `aabb` (from $L_1$ with n=2) and S2 generate `ccdd` (from $L_2$ with m=2), so S -> S1 S2 produces `aabbccdd` which is in $L_1L_2$. As expected, context-free languages are closed under concatenation [9] [10].

### Kleene Star

**Claim:** If L is a CFL, then the Kleene star L* (zero or more concatenations of strings from L) is also a CFL [11].

**Reasoning:** Let G be a CFG for L with start variable $S_1$. We construct a new grammar for L *by adding a new start symbol S with productions that allow either stopping immediately (deriving the empty string ε) or generating one L-string and then repeating:*

```
S -> ε
S -> S1 S
```

*In other words, S can produce zero copies of L (via S → ε) or one copy of an L string followed by more (via S → S1 S). Using these rules, S can derive any number of strings from L back-to-back, which exactly yields L*. This shows CFLs are closed under the Kleene star operation [12].

**Example:** If $L_1 = \{ a^n b^n \mid n \geq 0 \}$ as above, then $$L_1\text{^} = \{ (a^{n_1} b^{n_1})(a^{n_2} b^{n_2}) \cdots (a^{n_k} b^{n_k}) \mid k \geq 0,\ n_i \geq 0 \},$$ *which is essentially any number of balanced* `a...b` *blocks one after another (including the empty string when k=0). Our constructed grammar would generate exactly this set. For example, using the grammar, S → S1 S → a S1 b S could generate one* `a^n b^n` *block (e.g.* `ab` *if S1 goes to* `ab`*) and then S recurses to generate more. A full derivation could produce a string like* `abab` *(which is two* `ab` *blocks in a row, an element of $L_1$^*). This confirms the construction works.

### Other Closure Properties

In addition to the above operations, CFLs are also closed under other operations such as **reversal** and certain substitutions. For example, if L is context-free, its reversal $L^R = \{ w^R \mid w \in L \}$ (the set of all reversed strings from L) is also context-free [13]. We can prove this by taking a grammar for L and reversing the right-hand side of every production, which yields a grammar for $L^R$ [14]. Context-free languages are also closed under **homomorphisms** (symbol-by-symbol rewriting mappings) and their inverses [15] [16]. However, to keep this document focused, we will not delve into those in detail. The key take-away is that many common language operations do preserve context-free-ness.

One particularly useful fact is that if you intersect a context-free language with a **regular** language, the result is always context-free [17]. (Intuitively, a pushdown automaton can be combined with a finite automaton to enforce both a context-free condition and a regular condition at once [17].) We will use this fact in some proofs below.

## Operations That Do *Not* Preserve Context-Free Languages

Not all operations maintain the context-free property. Here we discuss two important operations under which CFLs are **not** closed: intersection (with another CFL) and complement. We provide intuitive explanations and counterexamples.

### Intersection (of Two CFLs)

**Claim:** In general, if $L_1$ and $L_2$ are context-free languages, their intersection $L_1 \cap L_2$ **need not** be context-free [18].

In other words, two CFLs can have an intersection that lies outside the CFL class. A pushdown automaton (PDA) has a single stack, which effectively allows it to **compare one pair** of quantities or handle one nested structure at a time. If we require it to simultaneously satisfy two independent context-free conditions, it may not be possible with one stack.

**Counterexample:** Let - $L_1 = \{ a^n b^n c^m \mid n, m \geq 0 \}$ – the language of strings where the number of `a`'s equals the number of `b`'s (and any number of `c`'s can follow), and - $L_2 = \{ a^m b^n c^n \mid m, n \geq 0 \}$ – the language of strings where the number of `b`'s equals the number of `c`'s (with any number of leading `a`'s).

Each of $L_1$ and $L_2$ is context-free (each can be generated by a CFG or recognized by a PDA). However, consider their intersection: $$L_1 \cap L_2 = \{ a^n b^n c^n \mid n \geq 0 \}.$$ In $L_1 \cap L_2$, a string must simultaneously satisfy "number of a = number of b" **and** "number of b = number of c," which implies all three counts are equal. The language {a^n b^n c^n} is a classic example of a language that is **not** context-free (this can be proven using the pumping lemma or other methods). Intuitively, a single-stack PDA cannot compare three quantities (the counts of `a`, `b`, and `c`) pairwise in the required way [19]. As a result, $L_1 \cap L_2$ is not a CFL, even though each of $L_1$ and $L_2$ is. This example proves that CFLs are not closed under intersection [19].

### Complement

**Claim:** If L is a context-free language, its complement Σ^ \ *L (over the same alphabet) is not guaranteed** to be context-free [18] [20].

The lack of closure under complement is actually related to the lack of closure under intersection. We can reason about complement using the above result. If CFLs **were** closed under complement, then for any two CFLs $L_1$ and $L_2$, we could express their intersection as

$$L_1 \cap L_2 = \text{complement of } (\text{complement of } \mathbf{L_1} \cup \text{complement of } \mathbf{L_2}).$$

Here "complement of X" denotes the complement of language X. If complement($L_1$) and complement($L_2$) were context-free (under our assumption), their union complement($L_1$) ∪ complement($L_2$) would be context-free (since union is closed). Then the complement of that union would also be context-free, yielding $L_1 \cap L_2$ as a context-free language. This would imply **all** intersections of CFLs are CFL, contradicting the fact we established above [21] [20]. Therefore, CFLs cannot be closed under complementation.

It is worth noting a special case: **deterministic** context-free languages (DCFLs) *are* closed under complement, but general CFLs are not. The distinctions between general CFLs and DCFLs, however, are beyond the scope of this document.

## Practice Questions on CFG Closure Properties

Below are some practice problems to test understanding of closure properties of context-free grammars (and their languages). Try to solve these medium-level questions on your own before checking the detailed solutions that follow.

1. **Grammar Union Construction:** Suppose you have context-free grammars $G_1$ and $G_2$ generating languages $L_1$ and $L_2$ respectively. Describe how to construct a new CFG that generates $L_1 \cup L_2$. Illustrate the construction with a specific example of two simple grammars.

2. **Grammar Concatenation Construction:** Given CFGs for $L_1$ and $L_2$, how can you construct a CFG for the concatenation $L_1 L_2$? Provide a general construction rule and apply it to an example.

3. **Kleene Star Grammar:** Let G be a CFG for language L. Explain how to modify or extend G to obtain a grammar for L^* (the Kleene star of L). Why does this construction work? Give an example using a simple language.

4. **Non-Closure under Intersection:** Give an example of two specific context-free languages $L_1$ and $L_2$ such that $L_1 \cap L_2$ is **not** context-free. Explain why each of $L_1$ and $L_2$ is context-free but their intersection is not.

5. **Non-Closure under Complement:** (a) Is the complement of a context-free language always context-free? Explain your answer. (b) Provide a brief argument or proof to justify why context-free languages are not closed under complementation (you may use results about intersection in your explanation).

## Detailed Solutions

**1. Grammar Union Construction:** To build a grammar for $L_1 \cup L_2$, introduce a fresh start symbol that can go to either of the start symbols of the original grammars [4]. Formally, if $G_1=(V_1, \Sigma, R_1, S_1)$ and $G_2=(V_2, \Sigma, R_2, S_2)$ are grammars for $L_1$ and $L_2$, ensure $V_1$ and $V_2$ have no overlap. Create a new grammar $G=(V_1 \cup V_2 \cup \{S\}, \Sigma, R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}, S)$. This G generates $L_1 \cup L_2$ [4].

*Example Solution:* Consider $L_1 = \{a^n b^n \mid n \geq 1\}$ with grammar $G_1$: $S_1 \to a\, S_1\, b \mid ab$ (which generates strings like `ab`, `aabb`, `aaabbb`, etc.), and $L_2 = \{b^m c^m \mid m \geq 1\}$ with grammar $G_2$: $S_2 \to b\, S_2\, c \mid bc$. To get a grammar for $L_1 \cup L_2$, use a new start S and include rules $S \to S_1 \mid S_2$. The combined grammar G is:

```
S  -> S1 | S2
S1 -> a S1 b | ab
S2 -> b S2 c | bc
```

G will generate any string of the form `a^n b^n` or `b^m c^m`. For example, `aabb` can be derived via S → S1 → a S1 b → a ab b, and `bbcc` via S → S2 → b S2 c → b bc c. Both types of strings are in $L_1 \cup L_2$.

**2. Grammar Concatenation Construction:** For concatenation $L_1 L_2$, the construction links the two grammars in sequence [8]. Using the same notation as above, form grammar $G'=(V_1 \cup V_2 \cup \{S\}, \Sigma, R_1 \cup R_2 \cup \{S \to S_1 S_2\}, S)$. Intuitively, S first generates a string from $L_1$ via $S_1$, then a string from $L_2$ via $S_2$ [8].

*Example Solution:* Using the $L_1$ and $L_2$ from the previous example ($L_1$ with grammar $S_1 \to a\, S_1\, b \mid ab$, and $L_2$ with $S_2 \to b\, S_2\, c \mid bc$), a grammar for $L_1 L_2$ would have $S \to S_1 S_2$. The rules are:

```
S  -> S1 S2
S1 -> a S1 b | ab       // generates strings in L1
S2 -> b S2 c | bc       // generates strings in L2
```

This grammar generates strings where an a^n b^n segment (from $L_1$) is immediately followed by a b^m c^m segment (from $L_2$). For example, starting from S, one derivation is $S \Rightarrow S_1 S_2 \Rightarrow a\, S_1\, b\, S_2 \Rightarrow a\, ab\, b\, S_2$ (finishing $S_1$ to produce `aabb`), then $S_2 \Rightarrow b\, S_2\, c \Rightarrow b\, bc\, c$ to produce `bbcc`. The final string is `aabbbbcc` which indeed has a prefix in $L_1$ (`aabb`) followed by a suffix in $L_2$ (`bbcc`). All strings derived by G′ are of this concatenated form and thus belong to $L_1 L_2$.

**3. Kleene Star Grammar:** To construct a grammar for L^ *given a grammar for L, introduce a new start symbol that can loop over the original grammar* [12]*. If $G=(V, \Sigma, R, S_0)$ generates L, create $$G^* = (V \cup {S},* Σ, R \cup {S → ε | $S_0$ S}, S)\,. $$ The added rules $S \to \varepsilon$ and $S \to S_0 S$ allow the new start S to produce zero copies of L (yielding the empty string) or to generate one L-string via $S_0$ and then again S (recursively allowing any number of additional L-strings) [12].

*Example Solution:* Let L = {a^n b^n | n ≥ 0} with grammar $S_0 \to a\ S_0\ b \mid \varepsilon$. To get a grammar for L^, *we add a new start S with rules* $S \to \varepsilon \mid S_0\ S$. *The rules are:*

```
S   -> ε | S0 S
S0  -> a S0 b | ε
```

*This grammar G^* generates strings that are any number of a^n b^n blocks in a row. For example, $S \to S_0\ S \to a\ S_0\ b\ S$ produces one a...b block (`ab` when $S_0$ goes to `ab`) and then S can either stop (go to ε) or produce another block. A derivation $S \to S_0\ S \to a\ S_0\ b\ S \to a\ \varepsilon\ b\ S \to a\ b\ S \to a\ b\ S_0\ S \to a\ b\ a\ S_0\ b\ S \to a\ b\ a\ \varepsilon\ b\ S \to a\ b\ ab\ S \to a\ b\ ab\ \varepsilon$ yields the string `abab` (which is two blocks `ab` concatenated). This confirms the construction works: G^ *generates exactly L^*.

**4. Non-Closure under Intersection:** One example of CFLs $L_1$, $L_2$ whose intersection is not context-free is: - $L_1$ = { a^n b^n c^m | n, m ≥ 0 } (strings where the number of `a` equals the number of `b`), - $L_2$ = { a^m b^n c^n | m, n ≥ 0 } (strings where the number of `b` equals the number of `c`).

As argued in the main text, both $L_1$ and $L_2$ are context-free (each can be generated by a pushdown automaton or grammar). However, $L_1 \cap L_2$ = { a^n b^n c^n | n ≥ 0 }, which is not a CFL [19].

*Why are $L_1$ and $L_2$ context-free?* Here are possible grammars:

```
G1 for L1:    S1 -> a S1 b | ε    (ensures equal a's and b's)
              S1 -> S1 C         // C -> c C | ε (any number of c's after
balancing a and b)

G2 for L2:    S2 -> B S2 c | B    (ensures equal b's and c's)
              B  -> a B | ε      (any number of a's before the b^n c^n core)
```

(There are other ways to construct these grammars; one could also use PDAs or known results to see these languages are CFL.)

*Why is the intersection not context-free?* Intuitively, a single-stack PDA cannot enforce both conditions (#a = #b **and** #b = #c) at once [22]. We would need one stack to track the a vs b count and another to track b vs c count, but only one stack is available. Formally, one can prove {a^n b^n c^n} is not context-free using the pumping lemma for CFLs [23]. Thus, this example demonstrates that intersection of two CFLs can fall outside the CFL family.

**5. Non-Closure under Complement:** (a) No, the complement of a CFL need not be context-free [21] [20]. While some specific context-free languages happen to have context-free complements, in general there are CFLs whose complements are not CFL.

(b) A convincing argument relies on the results for intersection. We know CFLs are not closed under intersection (from question 4). If we assume CFLs were closed under complement, we reach a contradiction as follows [21]:

- Let $L_1$ and $L_2$ be any two context-free languages. If complements are always context-free, then complement($L_1$) and complement($L_2$) are CFLs.
- Since CFLs are closed under union, complement($L_1$) ∪ complement($L_2$) would be context-free.

- If complements are closed, then taking the complement of this union would also be context-free. But complement(complement($L_1$) ∪ complement($L_2$)) = $L_1 \cap L_2$ by De Morgan's law.
- This implies $L_1 \cap L_2$ is context-free for *any* $L_1$, $L_2$ CFL, which is false (as we've seen with a counterexample).

Therefore, our assumption was wrong – CFLs cannot be closed under complementation [21]. In summary, there exists at least one context-free language whose complement is not context-free.

---

This document is also available as a formatted PDF: **Download the PDF file**

---


1 2 3 4 7 8 11 12 13 14 15 16 18 19 20 21 22 Closure Properties of Context Free Languages | GeeksforGeeks
https://www.geeksforgeeks.org/closure-properties-of-context-free-languages/

5 6 9 10 17 23 cfl3.5.dvi
https://www.cs.unc.edu/~plaisted/comp455/slides/cfl3.5.pdf