

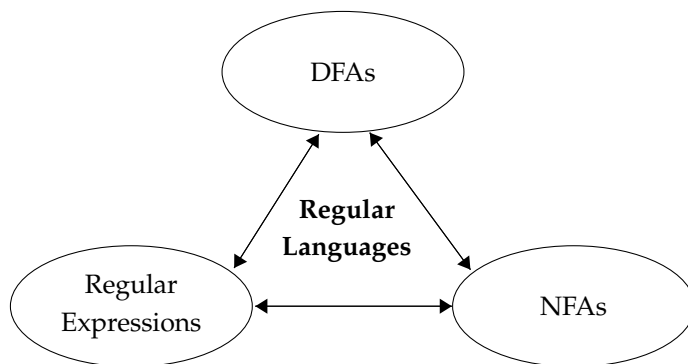
# Recitation 03: CFGs, CFLs, PDAs, Pumping Lemma For CFLs

These notes cover: context-free grammars (CFGs), context-free languages (CFLs), pushdown automata (PDAs), and the Pumping Lemma for CFLs as a method of proving certain languages are not context-free. We'll see that CFLs have fewer closure properties than regular languages, and go through a few examples of non-context-free languages.

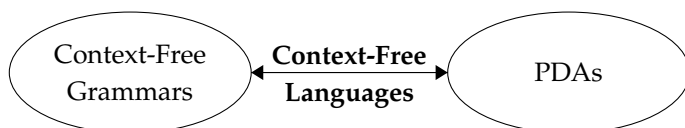
## Context-Free Languages

We saw in class that a CFL is a language generated by a CFG (context-free grammar) or a language recognized by a PDA (pushdown automaton).

We have seen three methods for representing regular languages, and ways to convert between them:



Context-free grammars (CFGs) and pushdown automata (PDAs) are two equivalent models for representing context-free languages:



The following table summarizes important closure properties for regular and context-free languages.

| class        | $A^*$ | $A \circ B$ | $A \cup B$ | $A \cap B$      | $\overline{A}$ |
|--------------|-------|-------------|------------|-----------------|----------------|
| regular      | yes   | yes         | yes        | yes             | yes            |
| context-free | yes   | yes         | yes        | no <sup>†</sup> | no             |

†: yes if intersected with a regular language

In this recitation, we show that

1. The intersection of a CFL and a regular language is a CFL.
2. The class of CFLs is not closed under intersection.
3. The class of CFLs is not closed under complement.

**Theorem 1.** *The intersection of a CFL and a regular language is a CFL.*

*Proof.* Let  $A$  be a CFL and  $B$  a regular language. Let  $P$  recognize  $A$  and  $M$  recognize  $B$ . The idea is to simulate both  $P$  and  $M$  simultaneously with a PDA  $\tilde{P}$ , similar to our proof of closure under union and intersection for regular languages (textbook Theorem 1.25).

We think of  $P$  as an NFA equipped with a LIFO stack. As we have seen in the proof that regular languages are closed under intersection, we can track the states of an NFA and a DFA simultaneously by making each state of  $\tilde{P}$  a pair  $(q, r) \in Q_P \times Q_M$ , an element of the Cartesian product of  $P$ 's and  $M$ 's state sets. The two components of the pair advance exactly according to the transitions of  $P$  and  $M$ , respectively.

The main difference from the regular language intersection proof is that  $P$  also has a stack. Since  $M$  has no stack,  $\tilde{P}$ 's stack can operate exactly as  $P$ 's does. Whenever  $P$  pushes or pops,  $\tilde{P}$  does the same.

The start state is the pair of  $P$ 's and  $M$ 's start states, and the set of accept states consists of the pairs  $(q, r)$  where  $q$  is an accept state of  $P$  and  $r$  is an accept state of  $M$ .

We include the formal 6-tuple construction in Appendix . □

**Theorem 2.** *The class of CFLs is not closed under intersection.*

*Proof.* We provide a counterexample, where the intersection of two CFLs is not a CFL.

Let

$$A = \{a^i b^j c^j \mid i, j \geq 0\}$$

$$B = \{a^i b^j c^j \mid i, j \geq 0\}$$

These are context-free. Consider  $A$ : a PDA can check whether the numbers of  $a$ 's and  $b$ 's in a string are equal by pushing onto the stack

The fact that we only need to simulate one stack is crucial to why this proof cannot be generalized to “prove” the false claim that “the intersection of two CFLs is a CFL” (a claim we disprove in the next theorem). If both  $P$  and  $M$  had stacks, we would not be able to simulate both simultaneously using the single stack of  $\tilde{P}$ .

when reading an  $a$  and popping when reading a  $b$ , then checking that the stack is empty. We can construct a PDA for  $B$  similarly.

However,

$$A \cap B = \{a^i b^i c^i \mid i \geq 0\},$$

which is not a CFL, as shown in lecture.  $\square$

**Theorem 3.** *The class of CFLs is not closed under complement.*

*Proof.* We present two ways to prove this.

1. We use De Morgan's law.

Assume for contradiction that CFLs are closed under complement.

Let  $A$  and  $B$  be CFLs. Since CFLs are closed under union,

$$A \cap B = \overline{\overline{A} \cup \overline{B}}$$

would also be a CFL. But then CFLs would be closed under intersection, a contradiction.

2. We provide a counterexample. Let

$$A = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } (i \neq j \text{ or } j \neq k)\}.$$

$A$  is a CFL: a PDA can nondeterministically choose to check either whether the numbers of  $a$ 's and  $b$ 's differ, or whether the numbers of  $b$ 's and  $c$ 's differ.

Note that

$$\overline{A} \cap \{a^* b^* c^*\} = \{a^i b^i c^i \mid i \geq 0\}.$$

Since  $\{a^* b^* c^*\}$  is regular, if  $\overline{A}$  were a CFL, then by Theorem 1,  $\overline{A} \cap \{a^* b^* c^*\}$  would be one too. But we know from lecture that it is not, a contradiction.  $\square$

Note that, as with NFAs, it is not enough to flip the accept and reject states of a PDA and claim that the modified PDA recognizes the complement of the original PDA's language, because of non-terminism.

To check whether the numbers of  $a$ 's and  $b$ 's differ, the PDA pushes an  $a$  onto the stack whenever it reads an  $a$  and pops one whenever it reads a  $b$ , similar to previous examples. It accepts if the stack is non-empty after the  $b$ 's are read (more  $a$ 's than  $b$ 's), or if it reads a  $b$  when the stack is already empty (more  $b$ 's than  $a$ 's).

### Pumping Lemma for CFLs

The **pumping lemma** for context-free languages is useful for proving that a particular language is not context-free.

**Theorem 4.** *If  $A$  is a context-free language, then there is a number  $p$ , known as the pumping length, such that if  $s$  is any string in  $A$  with length at least  $p$ , then  $s$  can be divided into five pieces  $s = uvxyz$  satisfying the following three conditions:*

1. for each  $i \geq 0$ ,  $uv^i xy^i z \in A$ ,
2.  $|vy| > 0$ , and

Similar to applying the pumping lemma for regular languages, *given*  $p$ , we *design*  $s$ , and show that *any* five-part split violates at least one of the conditions (typically, we assume 2 and 3, and show that *picking*  $i = 0$  or  $i > 1$  violates 1).

3.  $|vxy| \leq p$ .

We go through an example using the pumping lemma.

**Example 1.** Show that the language  $A = \{ww \mid w \in \{0,1\}^*\}$  is not a CFL.

**Solution.** Assume for contradiction that  $A$  is context-free. Let  $p$  be the pumping length given by the pumping lemma. We show that the string  $s = 0^p 1^p 0^p 1^p$  cannot be pumped.

Note that  $|s| = 4p \geq p$ . By condition 3 of the pumping lemma,  $s$  can be divided into  $s = uvxyz$  where  $|vxy| \leq p$ . We consider where the substring  $vxy$  can occur.

First, if  $vxy$  occurs only in the first half of  $s$ , pumping  $s$  down to  $uv^0xy^0z = uxz$  gives  $0^i 1^j 0^p 1^p$ , where at least one of  $i$  and  $j$  is less than  $p$ . There is no way to split this string into  $ww$ , so this is not in  $A$ .

Similarly, if  $vxy$  covers the middle of  $s$ , pumping down gives  $0^p 1^i 0^j 1^p$ . If  $vxy$  is in the rightmost half of  $s$ , pumping down gives  $0^p 1^p 0^i 1^j$ . It is also not possible to split these strings into the form  $ww$ .

Thus,  $s$  cannot be pumped, and  $A$  is not a CFL.

The five-part split of  $s$  makes for much more casework when using the pumping lemma for CFLs than when using the one for regular languages. Always be careful about where  $v$  and  $y$  can fall in the chosen string, and what the effects of pumping are. There are often many combinations!

Intuitively, we want to select an  $s$  for which it is easy to exploit the pumping property to leave the language. Including  $p$  in  $s$  this way limits what  $v$  and  $y$  can be, due to condition 3:  $|vxy|$  must be contained within at most two of the four parts of  $s$ .

To see why  $0^i 1^j 0^p 1^p$  cannot be written as  $ww$ , let  $k = |vy| \geq 1$ , so  $i, j \leq p$  and  $i + j = 2p - k$ . If the string's length  $4p - k$  is odd, it clearly cannot be  $ww$ . Otherwise  $|w| = 2p - k/2$ , so the second copy of  $w$  is the suffix  $0^{p-k/2} 1^p$ . The first copy of  $w$  must then also be  $0^{p-k/2} 1^p$ , which forces  $i = p - k/2$  and  $j = p$ . But then  $i + j = 2p - k/2 \neq 2p - k$ , a contradiction.

The same approach works for these two cases: compare the lengths of the two halves, then compare what the first copy of  $w$  must be with what the second copy must be.

## Appendix: Formal Construction for Theorem 1

Let  $P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, F_P)$  recognize the CFL  $A$ , and

let  $M = (Q_M, \Sigma, \delta_M, r_M, F_M)$  recognize the regular language  $B$ .

Construct  $\tilde{P}$  to recognize  $A \cap B$ , where  $\tilde{P} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ .

1.  $Q = Q_P \times Q_M = \{(q, r) \mid q \in Q_P \text{ and } r \in Q_M\}$ .  
That is, the states of  $\tilde{P}$  are the Cartesian product of the states of  $P$  and the states of  $M$ .
2.  $\Sigma$ , the input alphabet, is the same as in  $P$  and  $M$ .
3.  $\Gamma$ , the stack alphabet, is the same as in  $P$ . Because  $M$  has no stack,  $\tilde{P}$  uses its stack exactly as  $P$  does.
4.  $\delta$ , the transition function, is defined as follows. For each  $(q, r) \in Q$ , each  $a \in \Sigma_\epsilon$ , and each  $s \in \Gamma_\epsilon$ , let

$$\delta((q, r), a, s) = \begin{cases} \{((q', \delta_M(r, a)), s') \mid (q', s') \in \delta_P(q, a, s)\} & \text{if } a \in \Sigma, \\ \{((q', r), s') \mid (q', s') \in \delta_P(q, \epsilon, s)\} & \text{if } a = \epsilon. \end{cases}$$

When  $\tilde{P}$  reads an input symbol  $a \in \Sigma$ , both components advance on  $a$ , and  $\tilde{P}$  performs the same stack operation that  $P$  would. When  $P$  makes an  $\epsilon$ -move, no input is read, so the  $M$  component stays where it is.

5.  $q_0 = (q_P, r_M)$ , the pair of  $P$ 's and  $M$ 's start states.
6.  $F$  is the set of pairs in which both components are accept states of  $P$  and  $M$ , respectively:

$$F = F_P \times F_M = \{(q, r) \mid q \in F_P \text{ and } r \in F_M\}.$$

A branch of  $\tilde{P}$  on input  $w$  reaches a state in  $F$  after reading all of  $w$  exactly when the corresponding branch of  $P$  accepts  $w$  and  $M$  accepts  $w$ . Therefore  $\tilde{P}$  recognizes  $A \cap B$ , and  $A \cap B$  is context-free.