

# **Introduction to Theoretical Computer Science**

## **Lecture 4: Beyond the Context-Free Languages**

Dr. Liam O'Connor

University of Edinburgh  
Semester 1, 2023/2024

A language is *context-free* (a CFL) iff it is recognised by a context-free grammar.

**Exercise:** Are all regular languages context-free?

## Uses of CFLs

Many programming languages are syntactically context-free. Even the syntax we defined last lecture for regular expressions is context free. Suppose  $\Sigma = \{a, b\}$ .

$$S \rightarrow \emptyset \mid \varepsilon \mid a \mid b \mid S \cup S \mid S \circ S \mid S^* \mid (S)$$

**Exercise:** Derive with this grammar that  $(a \cup b \circ a)^*$  is a regular expression.

Always replace the leftmost remaining non-terminal at each step, giving a *leftmost derivation*.

# Parse Trees

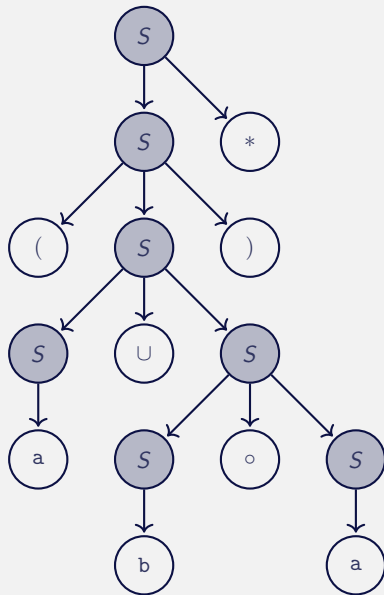
A *parse tree* is a tree that shows how to derive a string from a non-terminal.

The *yield* of a parse tree is the concatenation of all symbols at the leaves of the tree. If the root of the tree is  $S$  then the yield  $x \in \mathcal{L}(G)$ .

**Exercise:** Are there multiple parse trees possible for our example?

## Ambiguity

A grammar is *ambiguous* if there is more than one parse tree (or leftmost derivation) for a given string. This can cause problems with parsing and with interpretation.



# Eliminating Ambiguity

We want to eliminate ambiguity while still accepting all strings we accepted before. This is possible for our regular expressions language.

Define first the **atomic** expressions:

$$A \rightarrow (S) \mid \emptyset \mid \varepsilon \mid a \mid b$$

Then expressions that may include Kleene star:

$$K \rightarrow A \mid A^*$$

Then the expressions that may include composition (but **left-associatively**):

$$C \rightarrow K \mid C \circ K$$

Lastly, expressions that may include union:

$$S \rightarrow C \mid S \cup C$$

**Question:** What order of operations is assumed here?

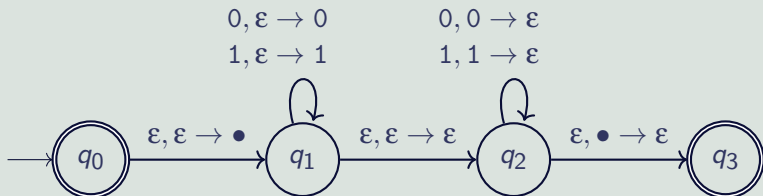
# Push-down Automata

*Push-down Automata* (PDAs) are to CFGs what Finite Automata are to regexps. Just as recursion is implemented with a stack in computer programming, a PDA is a  $\epsilon$ -NFA with an additional *stack*.

It is **more powerful** than an NFA as it has infinite memory, but can only use it by pushing and popping symbols.

# Push-down Automata

## Example (Push-down Automaton)



Read  $x, y \rightarrow z$  as consuming input  $x$ , popping  $y$  off the top of the stack, and pushing  $z$  on to the stack. The transition may only fire if  $y$  is on top of the stack.

In the above example, the input alphabet  $\Sigma$  is  $\{0, 1\}$  and the *stack alphabet*  $\Gamma$  is  $\{0, 1, \bullet\}$ .

**Exercise:** What language is accepted here? Derive the string 1001.

# Formally

## Definition

A *push-down automaton* is a 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$  where  $Q, \Sigma, \Gamma$  are all finite sets.  $\Gamma$  is the stack alphabet, and  $\delta$  now may take a stack symbol as input or return one as output:

$$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$$

All other components are as with  $\epsilon$ -NFAs.

## Acceptance

A string  $w$  is accepted by a PDA if it ends in a final state, i.e.  $\delta^*(q_0, w, \epsilon)$  gives a state  $q$  and a stack  $\gamma$  such that  $q \in F$ .

# Claim

## Theorem

A language is context-free iff it is recognised by a push-down automaton.

- Think about why this might be.
- Can you think about languages that might not be context-free?
- Next lecture: beyond the context-free languages.



# Claim

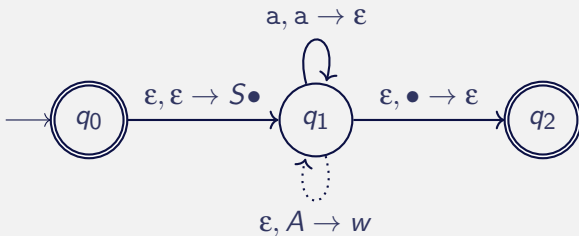
## Theorem

A language is context-free iff it is recognised by a push-down automaton.

The details of the proof of this are in Sipser's book, but I will give a sketch here.

## CFG to PDA

The upper self-loop is added for every terminal  $a$  in the CFG. The lower self-loop is a shorthand for a looping sequence of states added for each production  $A \rightarrow w$  that builds up  $w$  on the stack one symbol at a time.



## PDA to CFG

First, we make sure that the PDA has only one accept state, empties its stack before terminating, and has only transitions that either push or pop a symbol (but not transitions that do both or neither).

Given such a PDA  $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ , we provide a CFG  $(V, \Sigma, R, S)$  with  $V$  containing a non-terminal  $A_{pq}$  for every pair of states  $(p, q) \in Q \times Q$ . The non-terminal  $A_{pq}$  generates all strings that go from  $p$  with an empty stack to  $q$  with an empty stack. Then  $S$  is just  $A_{q_0 q_{\text{accept}}}$ .  $R$  consists of:

- $A_{pq} \rightarrow aA_{rs}b$  if  $p \xrightarrow{a, \varepsilon \rightarrow t} r$  and  $s \xrightarrow{b, t \rightarrow \varepsilon} q$  (for intermediate states  $r, s$  and stack symbol  $t$ ).
- $A_{pq} \rightarrow A_{pr}A_{rq}$  for all intermediate states  $r$ .
- $A_{pp} \rightarrow \varepsilon$

Proofs of why this works are in Sipser.

# Closure properties

Are context-free languages closed under:

- Union? **Yes**
- Concatenation? **Yes**
- Kleene Star? **Yes**
- Intersection? **No**

## Example

Consider  $L_1 = \{a^i b^i c^j \mid i, j \in \mathbb{N}\}$  and  $L_2 = \{a^j b^i c^i \mid i, j \in \mathbb{N}\}$ .

- Complementation? **No** (via de Morgan's laws)