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?

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 $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$$S \to \emptyset \mid \varepsilon \mid \mathtt{a} \mid \mathtt{b} \mid S \cup S \mid S \circ S \mid S^* \mid (S)$$

Exercise: Derive with this grammar that $(\mathtt{a} \cup \mathtt{b} \circ \mathtt{a})^*$ is a regular expression.
Always replace the leftmost remaining non-terminal at each step, giving a *leftmost derivation*.

Parse Trees

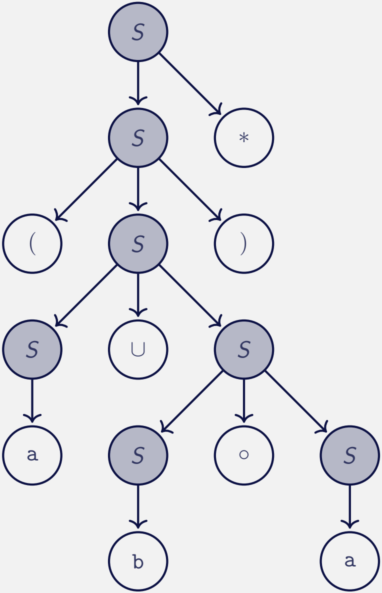
A *parse tree* is a tree that shows how to derive a string from a non-terminal.

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:

$$A \to (S) \mid \emptyset \mid \varepsilon \mid \mathtt{a} \mid \mathtt{b}$$

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:

$$A \to (S) \mid \emptyset \mid \varepsilon \mid \mathtt{a} \mid \mathtt{b}$$

Then expressions that may include Kleene star:

$$K \to A \mid A*$$

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:

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

Then expressions that may include Kleene star:

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

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

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:

$$A \to (S) \mid \emptyset \mid \varepsilon \mid \mathtt{a} \mid \mathtt{b}$$

Then expressions that may include Kleene star:

$$K \to A \mid A*$$

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

$$C \to K \mid C \circ K$$

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:

$$A \to (S) \mid \emptyset \mid \varepsilon \mid \mathtt{a} \mid \mathtt{b}$$

Then expressions that may include Kleene star:

$$K \to A \mid A*$$

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

$$C \to K \mid C \circ K$$

Lastly, expressions that may include union:

$$S \to C \mid S \cup C$$

Question: What order of operations is assumed here?

Push-down Automata

Push-down Automata (PDAs) are to CFGs what Finite Automata are to regexps.

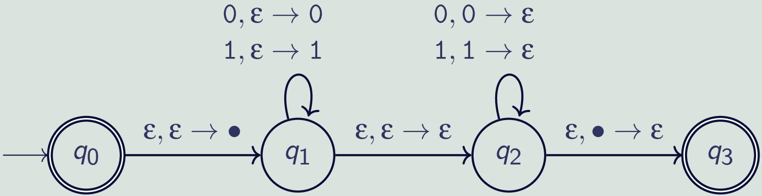
Push-down Automata

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 ε-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 $x, y \to z$ 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 Σ is $\{0, 1\}$ and the *stack alphabet* Γ 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, Σ, Γ are all finite sets. Γ is the stack alphabet, and δ now may take a stack symbol as input or return one as output:

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$$

All other components are as with ε-NFAs.

Acceptance

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

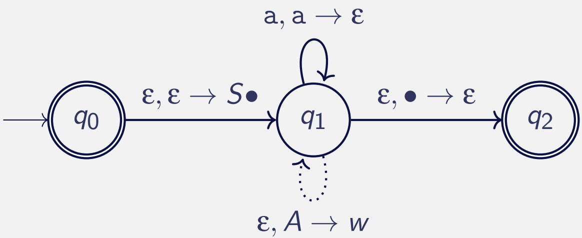
Theorem

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 $A \to w$ 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).

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, Σ, 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 $A_{q_0 q_{\text{accept}}}$.

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, Σ, 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:

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

Proofs of why this works are in Sipser.

Closure properties

Are context-free languages closed under:

- Union?

Closure properties

Are context-free languages closed under:

- Union? **Yes**
- Concatenation?

Closure properties

Are context-free languages closed under:

- Union? **Yes**
- Concatenation? **Yes**
- Kleene Star?

Closure properties

Are context-free languages closed under:

- Union? **Yes**
- Concatenation? **Yes**
- Kleene Star? **Yes**
- Intersection?

Closure properties

Are context-free languages closed under:

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

Example

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

- Complementation?

Closure properties

Are context-free languages closed under:

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

Example

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

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