Summary

Inhalt

[1. Regular Languages 3](#_Toc61851777)

[1.1 Formal Languages 3](#_Toc61851778)

[1.2 Deterministic Finite Automata 3](#_Toc61851779)

[1.2.1 Definition 3](#_Toc61851780)

[1.2.2 Transition table 4](#_Toc61851781)

[1.2.3 Regular languages 4](#_Toc61851782)

[1.3 Nondeterministic Finite Automata 4](#_Toc61851783)

[1.3.1 Definition 4](#_Toc61851784)

[1.3.2 DFA to NFA Conversion 4](#_Toc61851785)

[1.3.3 NFA to DFA Conversion 5](#_Toc61851786)

[1.3.4 Powerset Construction 5](#_Toc61851787)

[1.4 Nondeterministic Finite Automata with ε-Transitions 6](#_Toc61851788)

[1.4.1 DFA to ε-NFA-Conversion 6](#_Toc61851789)

[1.4.2 ε-NFA to DFA-Conversion 6](#_Toc61851790)

[1.5 Regular Expressions 7](#_Toc61851791)

[1.5.1 Definition 7](#_Toc61851792)

[1.5.2 Rules 7](#_Toc61851793)

[1.5.3 Regular expression to DFA conversion 8](#_Toc61851794)

[1.5.4 DFA to Regular Expression conversion 9](#_Toc61851795)

[1.6 Closure Properties of Regular Languages 9](#_Toc61851796)

[1.7 The Pumping Lemma for Regular Languages 9](#_Toc61851797)

[1.7.1 Definition 9](#_Toc61851798)

[2. Context-Free Languages 10](#_Toc61851799)

[2.1 Context-Free Grammars (CFG) 10](#_Toc61851800)

[2.1.1 Definition 10](#_Toc61851801)

[2.2 Normalizing Context-Free Grammars 11](#_Toc61851802)

[2.2.1 Eliminating ε-Productions 11](#_Toc61851803)

[2.2.2 Eliminating Unit Productions 12](#_Toc61851804)

[2.2.3 Eliminating Useless Symbols 12](#_Toc61851805)

[2.2.5 The Chomsky Normal Form 13](#_Toc61851806)

[2.3 The CYK Algorithm 14](#_Toc61851807)

[2.4 The Pumping Lemma for Context-Free Languages 14](#_Toc61851808)

# 1. Regular Languages

## 1.1 Formal Languages

**Alphabet**: Finite, nonempty set. Denoted by Σ.

**Characters, Letters, Symbols**: Elements of an alphabet

**Word, String**: Sequence that can be obtained by concatenating any finite number of letter of the alphabet

**Empty word**: Consists of no letters. Denoted by εn

**Language**: finite or infinite set that contains words over the alphabet

**Σk**: language that contains all words with exactly k letters. **Σ0** is **{ε}**

**Kleene Closure**: Σ∗ = Σ0 ∪ Σ1 ∪ Σ2 ∪ ...

Σ+ = Σ0 ∪ Σ1 ∪ Σ2 ∪ ... : all non-empty words

**Complement of L**: LC = Σ∗ \ L

**Concatenation of L1 and L2**: L1 ◦ L2 = L1L2 = {vw | v ∈ L1 and w ∈ L2}

**kth power of a language L**: Lk+1 = LkL

## 1.2 Deterministic Finite Automata

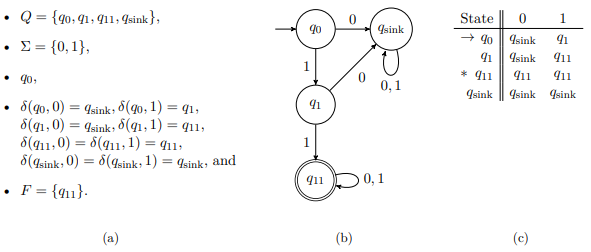
### 1.2.1 Definition

* Q is a finite set of states
* Σ is an alphabet, called the input alphabet
* δ : Q × Σ → Q is the transition function
* q0 ∈ Q is the start state
* F is the set of accepting states

How to draw DFAs:

* States are circles
* A transition is represented by an arrow
* The start state is marked by an incoming arrow
* All accepting states are marked by a double circle

### 1.2.2 Transition table



### 1.2.3 Regular languages

A language is regular if there if there exists a DFA for it.

## 1.3 Nondeterministic Finite Automata

An NFA can go into multiple different states from state q with the same input or to no state at all. If a run goes through a state from which it can’t get anywhere with the next input, then the run ends and is rejected by the NFA.

### 1.3.1 Definition

* Q is a finite set of state
* Σ is the input alphabet
* δ : Q × Σ → Pow(Q) is the transition function
* q0 ∈ Q is the start state
* F is the set of accepting states

A word is accepted by the NFA if there exists a run through the NFA where the word ends in an accepting state.

Both DFA and NFA have the same expressive power. All languages that can be described by a NFA can also be described by a DFA.

### 1.3.2 DFA to NFA Conversion

Every DFA can be converted into an equivalent NFA.

Proof: Every DFA is an NFA with no nondeterministic guesses.

### 1.3.3 NFA to DFA Conversion

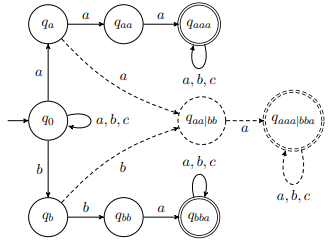
Every NFA can be converted into an equivalent DFA.

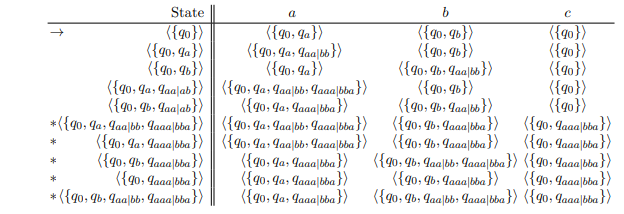
Proof: This is done with the powerset construction.

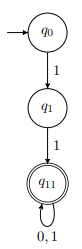
### 1.3.4 Powerset Construction

For every state q list all other states it can go to with every possible input. Of all these combinations do the same until there are no new states.

Example 1:



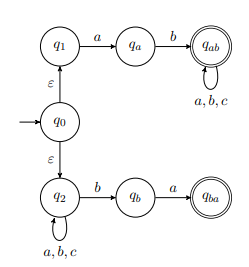


 If stuck it goes into the empty state.

## 1.4 Nondeterministic Finite Automata with ε-Transitions

NFAs with the added property that transitions are allowed to be labeled with the empty word ε.

Example:

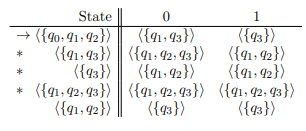
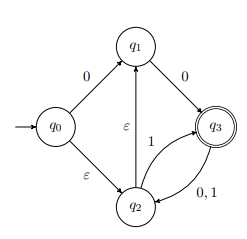
### 1.4.1 DFA to ε-NFA-Conversion

Same as DFA to NFA. No nondeterministic guesses nor ε-transitions.

### 1.4.2 ε-NFA to DFA-Conversion

**powerset construction with ε-closures**: We use a variant of the powerset construction that incorporates the possibility to follow ε-transitions after every letter from the input word that is read

Example:



## 1.5 Regular Expressions

### 1.5.1 Definition

For a given regular expression R, Lang(R) ⊆ Σ ∗ , for some alphabet Σ, denotes the language described by R.

### 1.5.2 Rules

* 
* 
* 
* 
* 
* 
* 
* 
* 

### 1.5.3 Regular expression to DFA conversion

Idea: Design an ε-NFA from a given regular expression in a bottom-up fashion. All intermediate ε-NFAs should have exactly one start state and one accepting state.

* We start with the subexpression ∅, for which we build an ε-NFA:



* For the subexpression ε, we build an ε-NFA:



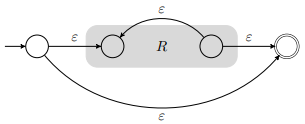
* For every subexpression that corresponds to a single letter l ∈ Σ, we apply this construction:



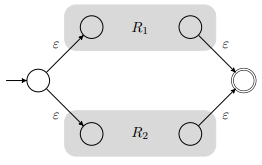
* Suppose the given regular expression contains the subexpression R1R2. Then we build an ε-NFA for R1R2 by connecting R1 and R2 by an ε-transition using the accepting state from R1 and the start state from R2.



* If the subexpression R∗ appears where we already designed an ε-NFA for the subexpression R:



* R1 + R2:



### 1.5.4 DFA to Regular Expression conversion

* R(0)ij is a regular expression that corresponds to going from state i to j without using another state. R(0)ij= a1+a2+ak
* if there is no transition from i to j, R(0)ij = ∅.
* A special case is that i = j: R(0)ii = ε + a1+a2+ ak
* Now we iteratively allow more and more intermediate states
* R(1)ij we look at all possibilities to go from I to j while allowing to use the intermediate state 1:
  + Either the intermediate state 1 is not used, i.e., we can go from i to j directly, which corresponds to the regular expression R(0)ij
  + or the intermeditate state 1 is used. In this case, we can split the tour from i to j into three parts. First, a transition from i to 1 is taken. Second, we can follow any number of loops in 1 (if there is a loop). Last, a transition from 1 to j is taken.
* For R(k)ij for k>1 the same is done with intermediate state k.



* The regular expression becomes:



## 1.6 Closure Properties of Regular Languages

* If a language is regular, this implies the existence of a corresponding DFA, NFA, ε-NFA or regular expression.
* If a language L is regular, then LC is a regular language too.
* If a language L is regular, then the reversal of L LR is also a regular language.
* If a language L is regular, then L∗ is a regular language, too.
* If two languages L1 and L2 are regular, then L1L2 is a regular language.
* If two languages L1 and L2 are regular, then L1 ∪ L2 (L1 + L2)is a regular language.
* If two languages L1 and L2 are regular, then L1 ∩ L2 is a regular language.
* If two languages L1 and L2 are regular, then L1 \ L2 is a regular language.

## 1.7 The Pumping Lemma for Regular Languages

### 1.7.1 Definition

Let L be a regular language. Then there is a constant *n0* such that, for every word *w ∈ L* with *|w| ≥ n0*, there is a decomposition *w = xyz* such that

* *|xy| ≤ n0*
* *|y| ≥ 1*
* *xywz ∈ L* for every *w ∈ N*

# 2. Context-Free Languages

## 2.1 Context-Free Grammars (CFG)

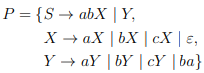
A grammar is a system of rewriting rules. Describes how symbols can be substituted by other symbols.

### 2.1.1 Definition

* *ΣN* is an alphabet, called the nonterminals or variables
* *ΣT* is an alphabet, called the terminals
* *S ∈ ΣN* is the start symbol
* *P* is the set of productions. A production *p ∈ P* is a pair *p = (A, α)* with *A ∈ ΣN* and *α ∈ (ΣN ∪ ΣT)∗* , which, for simplicity, is usually written as
  + *A → α*
* and which states that the nonterminal *A* can be replaced by the word *α*. In this context, *A* is called the head of *p* and *α* is the body of *p*

Example:

*Gab,ba = ({S, X, Y }, {a, b, c}, S, P)* Word starts with ab or ends with ba



**Derivation**: Any sequence of derivation steps that starts with S and ends with a terminal word

A language is context-free if there is a CFG for it.

## 2.2 Normalizing Context-Free Grammars

Goal:

1. no ε-productions
2. no unit productions
3. no useless symbols

## 2.2.1 Eliminating ε-Productions

* Find all productions that can go to ε
* Find all productions that can go into one of the productions from before
* Repeat with the new set of productions until the set stays the same
* Define new productions

Example:

* Starting productions
  + S → AB
  + S → C
  + A → aB
  + A → ε
  + B → bA
  + B → ε
  + C → c
* Set N0:
  + A,B
* Set N1:
  + A,B,S
* Set N2:
  + A,B,S
  + =N1
* New productions:
  + S → AB becomes S → B, S → A, and S → AB
  + A → aB becomes A → a, and A → aB
  + B → bA becomes B → b, and B → bA

### 2.2.2 Eliminating Unit Productions

1. Find all unit pairs:

Unit pairs are:

* All (X,X)
* All (X,Y) where X 🡪 Y without any unit productions!
  + Can include other steps between as long as no units are introduced

2. Create P’.

Example:

* Starting productions
  + A → aB
  + B → C
  + B → D
  + B → b
  + C → D
  + D → de
* Unit pairs
  + (A,A) A 🡪 aB
  + (B,B) B 🡪 b
  + (B,C)
  + (B,D) B 🡪 de
  + (C,C)
  + (C,D) C 🡪 de
  + (D,D) D 🡪 de
* P’
  + A → aB
  + B → b
  + B → de
  + C → de
  + D → de

### 2.2.3 Eliminating Useless Symbols

Useless symbols are non-terminals and terminals that never appear in any derivation.

* Compute generating symbols
* Compute reachable symbols
* Starting productions
  + S → A
  + S → AaB
  + S → BbA
  + B → bB
  + A → aa
  + A → Ab
  + C → cD
  + D → c
  + D → Ad
  + D → EE
  + D → dd
* Gen before step 1 of generating symbols
  + a,b,c,d
* Gen after step 1
  + a,b,c,d,A,D
* Gen after step 2
  + a,b,c,d,S,A,C,D
* Result:
  + S → A
  + A → aa
  + A → Ab
  + C → cD
  + D → c
  + D → Ad
  + D → dd
* Generate reachable symbols
  + ReachV={S}, ReachT=∅
  + ReachV={S,A}, ReachT={a,b}
  + C and D can’t be reached
* End-result
  + S → A
  + A → aa
  + A → Ab

### 2.2.5 The Chomsky Normal Form

Definition: Every X either consists of exactly two non-terminals or one terminal.

Steps:

* Start with an already normalized CFG
* Replace non-alone terminals with non-terminals
* If more than 2 non-terminals in one rule, replace 2 with one new rule

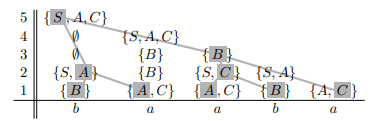
## 2.3 The CYK Algorithm

Each field is all possible combinations to get from the same column and the one on the right and a row lower.

If S appears in the top left corner then the word is in the language.

Example:

* S → AB | BC
* A → BA | a
* B → CC | b
* C → AB | a



## 2.4 The Pumping Lemma for Context-Free Languages