Predicate:
a mathematical predicate can be True or False.
predicates are functions with boolean return values:
P, Q(n), R(x,y,z)

Logical Operators:
AND: $P \wedge Q$ || OR: $P \vee Q$ || NOT: $\neg P$
Implication: $P \implies Q = \neg P \vee Q$

Distributive Rule
$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$
$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$

De Morgans Law
$\neg(P \wedge Q) = \neg P \vee \neg Q$
$\neg(P \vee Q) = \neg Q \implies \neg P$
$P \implies Q = \neg Q \implies \neg P$

Implication
$P \implies Q = True \;\&\; \neg P \implies Q = True$
$P \implies \neg Q = False$
$\neg P \implies \neg Q = True$

Quantors
OR: $\bigvee_{k=0}^{n} P_k$
P true for any $k \in 0..n$
AND: $\bigwedge_{k=0}^{n} P_k$
P true for all $k \in n$
All: $\forall k \in 0..n = P_k$
for all k P = True
Exists: $\exists k \in 0..n = P_k$
a k exists where P = True

Normalforms
disjunctive
$(x1 \wedge x2) \vee (\overline{x1} \wedge x2) \vee (x1 \wedge \overline{x2})$
conjunctive
$(x1 \vee x2) \wedge (\overline{x1} \vee x2) \wedge (x1 \vee \overline{x2})$
These are useful for true and false tables
This one would result to true if x1 or x2 is true.

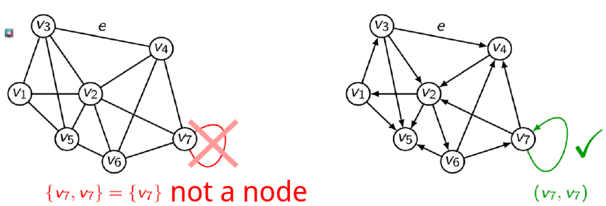
Quantities:
$\emptyset = \{\}$ $[n] = \{0..n\}$ $\{a .. z\}$
Union:
$A \cup B = \{x | x \in A \vee x \in B\}$
Intersection:
$A \cap B = \{x | x \in A \wedge x \in B\}$
Complement:
$\overline{A} = \{x | x \notin A\}$
Difference:
$A \setminus B = \{x \in A | x \notin B\}$

Pairs $A \times B = \{(a,b) | a \in A \wedge b \in B\}$

n-Tuples $\bigtimes_{k=0}^{n} A_i = \{(a_o, a_i, .., a_n) | a_i \in A_i\}$

Undirected Graph
doesn't have directions, and therefore can't have edges to itself

Directed Graph
This does have directions, therefore an edge to itself is valid!



Vertices V $= \{v_1, v_2, ..., v_n\}$ Edge e $= \{v_3, v_4\}$
EdgeCount E $= \{e | eEdge\}$

Proofs:
constructive Proof (proof by reforming)
Consider $ax^2 + bx + c = 0$, we can proof this to have 2 solutions by reforming.

$$ax^2 + bx + c = 0$$
$$x^2 + 2\frac{b}{2a}x + \frac{b^2}{4a^2} - \frac{b^2}{4a^2} + \frac{c}{a} = 0$$
$$x^2 + 2\frac{b}{2a}x + \frac{b^2}{4a^2} = \frac{b^2 - 4ac}{4a^2}$$
$$\left(x + \frac{b}{2a}\right)^2 = \frac{b^2 - 4ac}{4a^2}$$
$$x + \frac{b}{2a} = -\frac{b}{2a} \pm \sqrt{\frac{b^2 - 4ac}{4a^2}}$$

If $b^2 - 4ac > 0$ then we have 2 solutions!

Proof by contradiction
Take -2, it isn't a natural number. We can prove this by claiming the opposite.
If -2 is a natural number, then it has all the attributes of a natural number.
For example, it should be possible to take the square root of -2.
$\sqrt{-2} = NaN$
As you can see -2 does not have this attribute and is therefore
NOT a natural number!

Proof by Induction
This is particularly useful if you want to check an attribute
for a range of numbers such as n or n+1

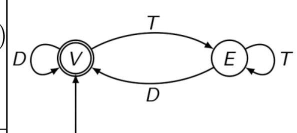
Base claim:
$$P(n) = \sum_{k=1}^{n} = \frac{n(n+1)}{2}$$
Anker: check for n=1
$$P(1) = \frac{1(1+1)}{2} = 1$$

Hypothesis: it also works for n+1
$$P(n+1): \sum_{k=1}^{n+1} k = \left(\sum_{k=1}^{n} k\right) + n + 1 = \frac{(n+1)(n+2)}{2} = \frac{(n+1)(n+1+1)}{2} \checkmark$$

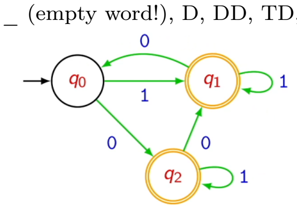
Alphabet and Word
Σ = Alphabet: Nonempty Quantity of characters
$\Sigma^n = \Sigma \times ... \times \Sigma$ = String
$w \in \Sigma^n$ An element in that string is a Word with length n.
$\varepsilon \in \Sigma^0$ The empty word, don't forget the empty word!
Quantity of all words:
$\Sigma^* = \{\varepsilon\} \cup \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup ... = \bigcup_{k=0}^{\infty} \Sigma^k$

Language $L \subset \Sigma^*$ = Language
$L = \emptyset \subset \Sigma^*$ = Empty Language
$L = \Sigma^* \to \Sigma = \{0, 1\}$ all binary strings.
A language is regular if a DFA can be formed out of it.

Deterministic Finite Automaton (DFA/DEA)



A very simple machine that accepts a variety of inputs.
Only requirement is that a D follows after T.
This means all the following inputs are valid:
_ (empty word!), D, DD, TD, TTTTTTTD, DDDDDTD, DDDDD, ...



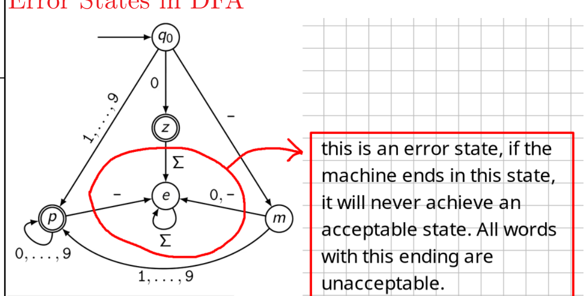
Machine A: $\{Q, \Sigma, \delta, q_0, F\}$
State = Q -> $\{q_1, q_2, ..., q_n\}$
Alphabet = Σ
Transitioning-Function = $\delta : Q \times \Sigma \to Q$
Starting State = $q_0 = L(\varepsilon) = L$
Acceptable Endstates = $F \subset Q$

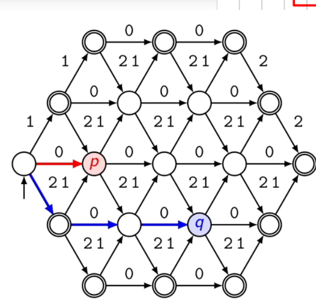
	0	1
q_0	q_2	q_1
q_1 /F	q_0	q_1
q_2 /F	q_1	q_2

all states Q
all acceptable states. F
Σ possible inputs, here 0 or 1
δ current position. -> change to q1

Language of DFA A:
$L(A) = \{w \in \Sigma^* | A \text{ accepts } w\} = \{w \in \Sigma^* | \delta(q_0, w) \in F\}$
The language of a DFA is simply all accepted words!

Error States in DFA





from q, many paths lead to F
This means 11, or 0 would be the "same"
$L(q) = \{0, 10, 11, 12, ...\}$
The same would obviously apply to P

Myhill-Nerode

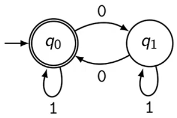
Adding a word to a word, to make it compatible with a language

$L(w) = \{w' | ww' \in L\}$ including: $L(\varepsilon)$!

w	$L(w)$	Q
ε	$L(\varepsilon) = L$	q_0
0	$L(0) = \{w \in \Sigma^* \mid \lvert w\rvert_0 \text{ uneven}\}$	q_1
1	$L(1) = \{w \in \Sigma^* \mid \lvert w\rvert_0 \text{ even}\} = L$	q_0
$\vdots$	$\vdots$	$\vdots$

even and uneven amounts of 0s

mod 2 zero's, 1's don't matter



Detecting Nonregular Languages with Myhill

The examples before always had a specific amount of words/characters that one had to add, in order to accept the word.

However, there are languages that would need infinite states in order to find the entire language of a DFA

A good example for this is the language 1^n0^n

w	$L(w)$	Q
ε	$\{0^n1^n \mid n \geq 0\}$	q_0
0	$\{0^n1^{n+1} \mid n \geq 0\}$	q_1
00	$\{0^n1^{n+2} \mid n \geq 0\}$	q_2
000	$\{0^n1^{n+3} \mid n \geq 0\}$	q_3
0^k	$\{0^n1^{n+k} \mid n \geq 0\}$	q_k
	$\vdots$	
01	$\{\varepsilon\}$	
001	$\{1\}$	
0001	$\{11\}$	
	$\vdots$	
1	$\emptyset$	e
10	$\emptyset$	e
	$\vdots$	

for every 0 that we add, we need a 1

this means that for n+k 0's we need k 1's

as $\lim_{k\to\infty}$ we need ∞ states!

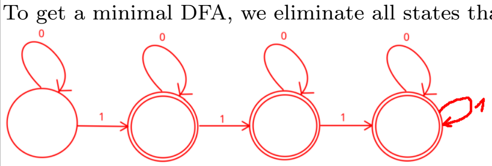
not possible with a Deterministic automaton!

also note: we have clear error states

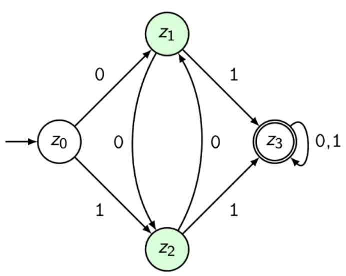
anything starting with 1 is an error.

Differentiation of States

To get a minimal DFA, we eliminate all states that are superfluous.



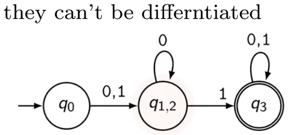
Here the 3 acceptable states can be put together, they are the same!



	z_0	z_1	z_2	z_3
z_0	$\equiv$	$\times$	$\times$	$\times$
z_1	$\times$	$\equiv$	$\equiv$	$\times$
z_2	$\times$	$\equiv$	$\equiv$	$\times$
z_3	$\times$	$\times$	$\times$	$\equiv$

z1 and z2 are the same!

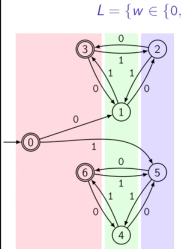
they can't be differntiated



New minimal Automaton!

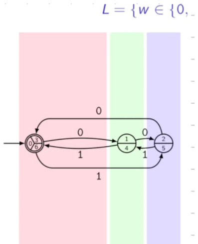
This algorithm makes it easy to see whether or not states are the same!

$L = \{w \in \{0,1\}^* \mid |w|_0 \equiv |w|_1 \mod 3\}$



	0	1	2	3	4	5	6
0	$\equiv$	$\otimes$	$\otimes$	$\equiv$	$\otimes$	$\otimes$	$\equiv$
1	$\otimes$	$\equiv$	$\otimes$	$\otimes$	$\equiv$	$\otimes$	$\otimes$
2	$\otimes$	$\otimes$	$\equiv$	$\otimes$	$\otimes$	$\equiv$	$\otimes$
3	$\equiv$	$\otimes$	$\otimes$	$\equiv$	$\otimes$	$\otimes$	$\equiv$
4	$\otimes$	$\equiv$	$\otimes$	$\otimes$	$\equiv$	$\otimes$	$\otimes$
5	$\otimes$	$\otimes$	$\equiv$	$\otimes$	$\otimes$	$\equiv$	$\otimes$
6	$\equiv$	$\otimes$	$\otimes$	$\equiv$	$\otimes$	$\otimes$	$\equiv$

$L = \{w \in \{0,$



Beauty of a language / Pumping Lemma

a language L can be pumped if the following is valid:

$\exists N > 0$ where $w \in L \wedge |w| \geq N$

If this word can be divided into 3 parts x,y,z while:

$|xy| \leq N \wedge |y| > 0 \wedge |x| \geq 0 \wedge |z| \geq 0 \wedge xy^kz \in L \rightarrow \forall k \in \mathbb{N}$

Find a number N that is bigger than 0 but less than the length of xy

while the length of y is greater than 0 and xy^kz is still part of the language

Note: The power of xy^kz is NOT a power, but an indicator how many y's!!!

Any language that can be pumped is a regular language, and is therefore "schön"

Consider the following 2 examples:

$L(s1) = \{0, 1|\text{ending with } 1\}$

z = 1, xy = any combination of 1 and 0

try: x=0 y=1010111 z=1,True -> $N \geq 7$

try: x=0 y=1 z=1,True -> $N \geq 1$

This can be done with any y, x

It will always satisfy all requierements of the pumping lemma.

this language is regular.

$L(s1) = \{0^n, 1^n | n \geq 0\}$

if z=1, more 1's than 0's, FALSE

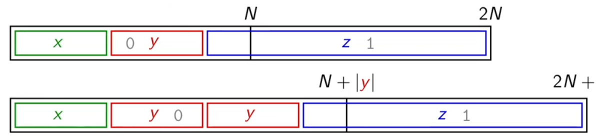
if x - 0, more 0's than 1's, FALSE

This means we need x=0, z=0

If we do that, then y is empty

No matter what we do, it is FALSE

0^n1^n is therefore not regular



Pumping Lemma usage guide !!FOLLOW THIS!!

1. Claim that L is regular
2. According to pumping lemma $\exists N$

Don't make claims about the size of N!

3. Choose a word $w \in L||w| \geq N$

Definition with N has to be written!

4. Division into parts according to Pumping Lemma

$w = xyz, |xy| \leq N, |y| > 0$,etc

5. Check if word is in language

min 1 word not in language: $xy^kz \notin L|k \in \mathbb{N}$

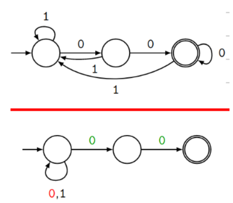
Explain why this word is not in the language

6. Contradiction, aka explain that this language is not regular.

Non-Deterministic Finite Automaton (NFA / NEA)

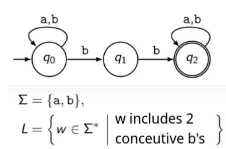
Before every step was clear, there was no other determinism other than the input and the current state. A non-deterministic automaton Can have other things, like only accept the last 2 0's

This is illustrated as both a DFA and an NFA:



Both have the same goal, only the last 2 zero's lead to the acceptance state. However one is obviously easier, while not giving clear info on in what state it is, it is hence non-deterministic.

Formal Definition:



Machine A: $\{Q,\Sigma,\delta,q_0,F\}$

State = Q -> $\{q_1, q_2, ..., q_n\}$

Alphabet = Σ

Transitioning-Function = $\delta : Q \times \Sigma \rightarrow \mathbf{P(Q)}$

Starting State = $q_0 = L(\varepsilon) = L$

Acceptable Endstates = $F \subset Q$

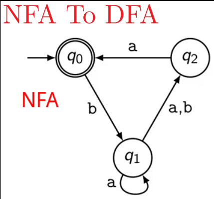
P(Q) is the Potence Quantity!!

Note the P(Q), it means that we have more complex transitions!

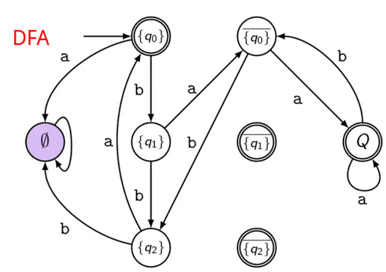
no arrow, multiple arrows for a certain character. See b in example

General tipp for NFA, only write what you need to accept the word!

NFA To DFA



The is also called a Thompson NFA



$\overline{q}_1$ is the complement Quantity. It means, any state other than q_1

Q is the full Quantity, in here the NFA can be in any state.

$\emptyset$ is the error state.

Formal Definition of the Transition

DFA is marked with ', the regular expression is for NFA

Given δ of an NFA and Transitions $M \subset Q$

$\delta' : Q \times \Sigma \to P(Q) : (M, a) \mapsto \delta'(M, a) = \bigcup_{q \in M} \delta(q, a)$

$Q' = P(Q)$

$\Sigma' = \Sigma$

$q_0' = \{q_0\}$

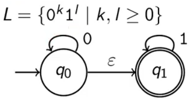
$F' = \{M \in P(Q) | F \cap M \neq \emptyset\}$

M is the union of all possible endstates.

Note: a Thompson NFA needs at least 1 acceptable endstate!

ε Transitions in NFA's

$L = \{0^k 1^l \mid k, l \geq 0\}$



$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$

This means that ε signifies a transition without using a character!!

Conversion from ε-NFA to regular NFA

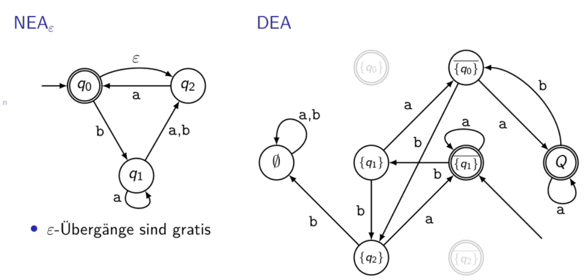
Any ε-NFA can be converted to a regular NFA!

$E(q) =$Quantity of all ε-Transitions from q

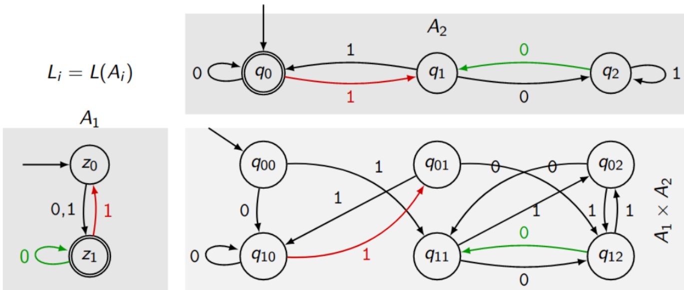
$E(M) = \cup_{q \in M} E(q)$ Quantity of all ε-Transitions

$\delta = Q \times (\Sigma \cap \{\varepsilon\} \to P(Q) : (q, a) \mapsto E(\delta(q, a)))$

ε-NFA to DFA:



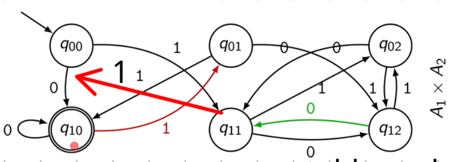
Set-Operations with Automatons



Intersection $L_1 \cap L_2$

$F = F_1 \times F_2$

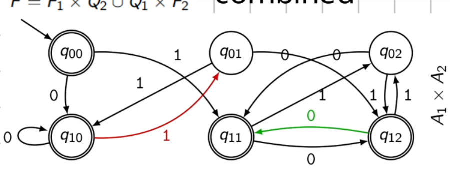
carthesian product -> acceptable endstate which both share



Union $L_1 \cup L_2$

$F = F_1 \times Q_2 \cup Q_1 \times F_2$

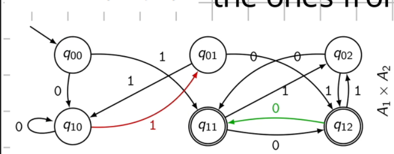
acceptable endstate of both combined



Difference $L_1 \setminus L_2$

$F = F_1 \times (Q_2 \setminus F_2)$

acceptable endstate of one without the ones from the other



Pump-able, but not regular

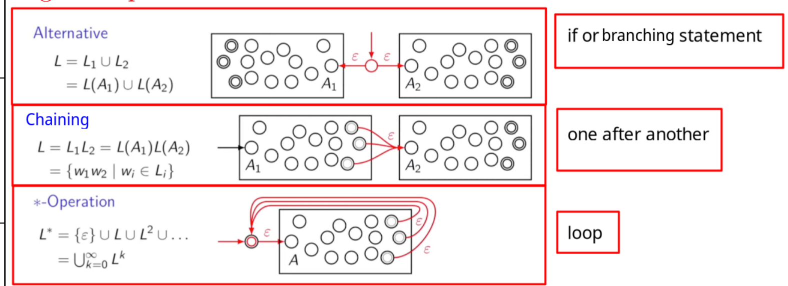
$$L = \{a^i b^j c^k \mid i = 0 \vee j = k\} = \underbrace{\{b^j c^k \mid j, k \geq 0\}}_{= L_1} \cup \underbrace{\{a^i b^j c^k \mid i > 0 \wedge j = k\}}_{= L_2}$$

The first part L1 is regular, but the second isn't

The only way we can figure that out is by using the Myhill method!

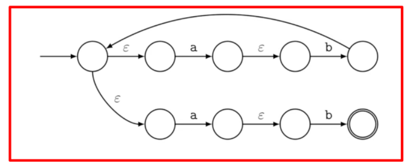
Because L2 is not regular, the composite Language L is not regular

Regular Operations

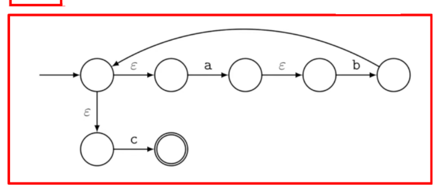


Regular Expressions

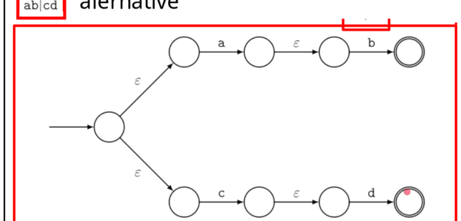
$\boxed{(ab)^+}$ do while



$\boxed{(ab)^* c}$ chain



$\boxed{ab|cd}$ alernative



Regular expression

String r to describe a language

L = L(r)

Regular Operations

$L(r_1) \cup L(r_2) = L(r_1 | r_2)$ Alternative

$L(r_1)L(r_2) = L(r_1 r_2)$ Chaining

$L(r_1)^* = L(r_1 *)$ *-Operation

Primitive regular expressions

expressions for Words with length <= 1

$L = L(r)$	r	NEA
$\emptyset$	$\emptyset$	
$\{\varepsilon\}$	ε	
$\{a\}$	a	a
{o, s, t}	[ost]	
{a, b, ..., s}	[a-s]	
Σ	.	

Contractions

$r+ = rr*$ $\quad$ $r? = \varepsilon | r = | r$

$r\{2\} = rr$ $\quad$ $r\{2,3\} = rr | rrr$

$r\{,3\} = | r | rr | rrr$ $\quad$ $r\{3,\} = rrrr*$

Generalized NFA/NEA

NEA$_\varepsilon$, NFA$_\varepsilon$, that has it's transitions described as regular expressions

There is a DFA for every regular expression!