- Alphabet: $\Sigma \neq \emptyset$
- Worte der Länge n: Σ^n
- alle Worte einschl. ε: Σ^*
  ↳ ist Obermenge aller Sprachen
- $\varepsilon w = w = w\varepsilon$
- $A^0 = \{\varepsilon\}$ mit Sprache A

DFA: $\mathcal{M} = (Z, \Sigma, \delta, z_0, E)$

- Z: Zustände $\{z_0, z_1, \dots\}$
- Σ: Eingabealphabet $\{a, b, \dots\}$
- δ: Übergangsfkt:

$\xrightarrow{a,b}$ / $\delta(z_0, a) = z_1$ /

δ	z_0	z_1
0	z_0	z_1
1	z_1	z_0

- z_0: Startzustand
- E: Menge Endzustände $\{z_1, \dots\}$

Sprache negieren: im DFA Endzustand zu Zustand und alle Zustände zu Endzuständen machen

DFA wichtig:
- jeder Knoten hat für jedes Zeichen genau eine Kante
- keine ε-Übergänge

Erweiterte Übergangsfkt:

$\hat{\delta}(z, w) = \begin{cases} z & w = \varepsilon \\ \hat{\delta}(\delta(z, a), x) & w = ax \end{cases}$

Bsp.: $\hat{\delta}(z_0, aaba) = \hat{\delta}(\delta(z_0, a), aba)$
$= \hat{\delta}(z_0, aba) = \dots = z_E$ ← vorletzter Schritt: $\hat{\delta}(z_E, \varepsilon)$

⇒ $\hat{\delta}$ bestimmt Endzustand des Wortes

NFA: wie DFA, aber kann mehrere Startzustände haben und hat nicht an jedem Knoten für alle Zeichen Kanten

$\mathcal{M} = (Z, \Sigma, \delta, S, E)$

NFA → DFA:

- Menge von Zustand/Zuständen ist neuer Zustand
- Folgezustand ist Menge von Zuständen, die im NFA mit Zeichen erreicht werden können (ausgehend von der Menge der Ursprungszustände)
- jeder Zustand, in dem Menge ein Endzustand des NFAs ist, ist Endzustand des DFAs

ein DFA/NFA/PDA akzeptiert, wenn Endzustand erreicht wird.

$\Sigma^n \Sigma^m = \prod_1^n \Sigma \prod_1^m \Sigma = \prod_1^{n+m} \Sigma = \Sigma^{n+m}$

reg. Exp in NFA (möglichst klein):
Für jeden Fall NFA aufstellen und dann zusammenführen $\varepsilon \mid 1 \mid (0|1)^*(0|01)$
1. Fall, 2. Fall, 3. Fall, 4. Fall

regulärer Ausdruck:

- $\emptyset$ ist regEx: $L(\emptyset) = \emptyset$
- a ist regEx: $L(a) = a$
- $E_1 | E_2 \Rightarrow L(E_1 | E_2) = L(E_1) \cup L(E_2)$ $\to E_1$, $\to E_2$
- $E_1 E_2 \Rightarrow L(E_1 E_2) = L(E_1) L(E_2)$ $\to E_1 \to E_2$
- $E_1^* \Rightarrow L(E_1^*) = L(E_1)^*$ Pfad zurück
- $E^+ = EE^*$
- $E^? = \varepsilon | E$

wenn E_1, E_2 reg, dann auch $(E_1 | E_2), (E_1 E_2), (E_1^*)$

wenn $L_{1/2}$ regulär, dann auch $\overline{L_1}, \overline{L_1^*}, L_1 L_2, L_1 \cup L_2, L_1 \cap L_2$

PDA: (kontextfreie Sprache)

$\mathcal{M} = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$

Γ: Stackalphabet (inkl. #)

#: unterstes Stackzeichen

Stack: $\xrightarrow{a, \gamma / \gamma_1 \dots \gamma_n}$

a: Eingabezeichen

γ vom Stack

$\gamma_1 \dots \gamma_n$ auf Stack (letztes zuerst)

Grammatik in PDA:

$\varepsilon, A/\gamma$ - - - - → Regel $A \to \gamma$

$\to \xrightarrow{\varepsilon, \varepsilon/\varepsilon} \xrightarrow{\varepsilon, \#/\varepsilon}$

$a, a/\varepsilon$ - - - → wenn a auf Stack liegt

Ablauf: Stack beginnt mit # und dann Startzustand. dann abarbeiten des Wortes

1. | # | 2. | S / # |

CNF (hat keine ε):

Alle Regeln in Form: $A \to BC$ oder $A \to a$
(ist bin. Wurzelbaum bis auf unterste Ebene)

L (ohne ε) in CNF:

1. Terminalsymb. durch neue Variablen ersetzen:
   $S \to aSa \Rightarrow S \to ASA$, $A \to a$
2. Mehrfache Variablen auf einer Seite ersetzen:
   $S \to ASA \Rightarrow S \to AR$, $R \to SA$

CYK: braucht CNF!

Vorgehen: $S \to AX$, $X \to BC$, $A \to a$, $B \to b$, $C \to c$

a	b	c

⇒

A	B	C
a	b	c

⇒

	S	
	X	
A	B	C
a	b	c

es müssen (Lgsf) alle Regeln gelistet werden:

X	Y	

Pumping Lemma (Beweisführung):

- Angenommen L sei regulär
- Nach PL existiert $n > 0$, sodass sich alle $x \in L$ mit $|x| \geq n$ gemäß PL zerlegen lassen
- Sei $x = \dots$ (passendes suchen)
- x einschränken durch Def PL:
  1. $|v| \geq 1$ 2. $|uv| \leq n$
  (u, w können ε sein)
- Mit 3. $uv^kw \in L$ für alle $k \geq 0$ zum Widerspruch führen

Wenn es ein $x \in L$ mit $n \leq |x| < 2n$ gibt (n: Anz. Zustände Min-Automat), dann $|L| = \infty$

kontextfreie Grammatik:

$G = (V, \Sigma, P, S)$

V: Variablen/Nonterminalsymbole

Σ: Alphabet / Terminalsymbole

P: Regeln

Bsp.: $S \to aS \mid \varepsilon$
$X \to aXb$ (in Tiefe wachsend)
$X \to XX$ (in Breite wachsend)

Umformungssyntax:
$S \Rightarrow aS \Rightarrow aaS \Rightarrow aa$
bzw. $S \Rightarrow^* aa$ (transitive Hülle)

Eindeutigkeit:

Operatorpriorität:
- höhere Priorität weiter „unten"

Assoziativität:
- links-assoziatives nur links

$E \to E+T \mid E-T \mid T$
$T \to T*F \mid T/F \mid F$
$F \to x \mid y \mid z$

Chomsky Hierarchie:

0. rekursiv aufzählbar — Regeln: $u \to v$
1. kontextsensitiv — u, r bel., $u = \alpha X \beta$, $r = \alpha \gamma \beta$
2. kontextfrei — $u \in V$, $v \in (V \cup \Sigma)^*$
3. regulär — $u \in V$, $v \in \Sigma \cup \{\varepsilon\} \cup \Sigma V$

Bsp:	Grammatik	Sprache ($n \geq 0$)
0	$ab \to c$	
1	$aAb \to aBb$	$a^n b^n c^n$
2	$A \to aBb$	$a^n b^n$
3	$A \to a$, $A \to aB$	a^n

- Sprache hat mind. gl. Klasse wie Grammatik, kann auch höhere haben
- jede kf enthält reg. Sprache: $\emptyset$
- nicht jede Teilmenge einer reg. Sprache ist kf.: $a^n b^n c^n \subseteq \{a, b, c\}^*$
  kontextsensitiv ↯ regulär
- nicht jede Teilmenge einer kf Sprache ist regulär: $a^n b^n \supseteq a^n b^n$
  kontextfr. ↯ nicht reg.
- nicht jede Obermenge einer reg. Sprache ist regulär: $a^n b^n \subseteq a^n b$
  nichtreg. ↯ reg.

Top-Down-Parser:
(Rekursive Descent)

```
bool parse(String inp){
  in = inp + "#";
  pos=0;
  return S() && match("#");
}
bool match(char c){
  if(next()==c){
    pos++; return true;
  }else{
    return false;
  }
}
char next(){
  return in.charAt(pos);
}
```

Regeln:

bsp.: $S \to aSb \mid \varepsilon$

```
bool S(){
  if (next()=='a')
    return { match('a') &&
             S(), &&
             match('b');   } S->aSb
  else return true;        } S->ε
```

nur möglich, wenn nicht links-rekursiv: sonst Endlosschleife!

$\Rightarrow$ $S \to SS$ geht nicht

$S \to (S)S$ ok

für links-rekursives: EBNF

$E \to T\{(+|-)\,T\}$

$T \to F\{(*|/)\,F\}$ ← $\{..\} = (..)^*$

$F \to x \mid y \mid z$

$\Rightarrow$ Parser: (Bsp.)

```
E(){
  T();
  while(next()='+'){
    match('+'); T();
  }
}
```

Grammatik arithm.:

$E \to E+T \mid E-T \mid T$

$T \to T*F \mid T/F \mid F$

$F \to x \mid y \mid z \mid -F \mid (E)$

Grammatik Palindrom:

$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

Gleiche Anzahl:

$S \to 1S0 \mid 0S1 \mid SS \mid \varepsilon$

NP-vollst. | P | NP | Entsch. spr.

Bottom-Up-Parser:

shift: nächste Zeichen auf Stack

reduce: Regel auf ein/mehrere Zeichen vom Stack anwenden

accept: Stack nur noch Startsymb. und Eingabe leer

error: Syntaxfehler

Bsp für $E \to E+T \mid E-T \mid T$
$T \to T*F \mid T/F \mid F$
$F \to x \mid y \mid z$

Stack	Eingabe	Aktion
	x+y*z	shift
x	+y*z	reduce
F	+y*z	reduce
T	+y*z	reduce
E	+y*z	shift
E+	y*z	⋮
⋮	⋮	
E+T	–	reduce
E	–	accept

* shift noch nicht möglich, da es keine Regel x+, F+, T+ gibt.
$\Rightarrow$ so lange reducen, bis shift sinnvoll geht

weitere Unentscheidbare:
$H_\varepsilon = \{P \mid P$ hält für Eingabe $\varepsilon\}$
$H^* = \{P \mid P$ hält für jede Eingabe$\}$
$A' = \{(P_1,P_2) \mid P_1, P_2$ berechnen gleiche Fkt.$\}$

Programm für A':
int F(Input w){return 0;}
int G(Input w){P(w); return 0;}
return $P_A(F,G)$; ← hält nicht garantiert ↯

$\underline{\underline{P}} = \bigcup_{k \geq 0} \{L \mid L$ ist entsch.bar durch Programm mit LZ $O(n^k)\}$

bspw.: • $\emptyset, \Sigma^*$: $O(1)$
• a^n : $O(n)$
• kf-Sprache: CYK : $O(n^3)$

Anwendung SAT:
• Schaltkreise (Logikgatter)
• Paketinstallation (Abhängigkeiten)
• Software Model Checking:

```
int a[2];
if(i==0) j=1;
else     j=2;
assert(0<=j && j>2);
x=a[j];
```

Travelling-Salesman (LZ: $O(n^2 2^n)$)

LZ: Tiefen/Breitensuche: $O(|V|+|E|)$
lineare Suche: $O(n)$

Entscheidbarkeit:

nur für Mengen, bspw. L ?

• reg. Ausdruck $\to$ DFA ist entscheidbar
• $L = \emptyset$ entscheidbar (immer false)
• $L = \Sigma^*$ entscheidbar (immer true)
• L als kontextfreie Sprache
↳ mit CYK entscheidbar

Entscheidbar mit P_L (Programm/Entscheidungsverfahren), wenn P_L = true bei $w \in L$ und P_L = false bei $w \notin L$.

Bsp.:

```
bool P(DFA M){
  return (Minimalautomat(M) == ...)
}
```

ist DFA — muss auch DFA sein

Halteproblem:

$K = \{P \mid$ Programm hält für Eingabe $P\}$ ist unentscheidbar

Beweisführung Unentscheidbarkeit von B.:
• unentscheidbares A (bspw. K) verwenden
• Annahme B ist entscheidbar, gibt also Entscheidungsverfahren
• Daraus lässt sich Entsch.verfahren für A konstruieren

Bsp.:
$H = \{(P,w) \mid P$ hält für $w\}$
• Angenommen H entsch. $\Rightarrow \exists$ Entsch.verf. P_H
• Programm konstruierbar:

```
bool P_K (Prog P){
  return P_H(P,P);
}
```

• Dann gilt: $P \in K \Leftrightarrow P$ hält für P
$\Leftrightarrow (P,P) \in H \Leftrightarrow P_H(P,P)$ ist true ↯

$\underline{\underline{NP}} = \bigcup_{k \geq 1} \{L \mid L$ ist verifizierbar in $O(n^k)\}$
n: Eingabelänge

Bsp.: SAT $= \{F \mid F$ ist erfüllbare Formel der Aussagenlogik$\}$
LZ: $O(2^n) \Rightarrow$ Wahrheitstabelle
SAT $\notin$ P ; SAT $\in$ NP $\to$ Verifikation mit erfüllender Formel $O(|F|)$

Tautologie: unbek., ob in NP
$P \overset{?}{=} NP$ unbek.

NP-vollständigkeit:

L ist NP-vollständig, wenn:
1. L $\in$ NP
2. kein effizientes, polynomielles Entscheidungsverfahren bekannt ist

SAT: NP vollständig — LZ: $O(p(n) \cdot 2^n)$
3SAT: — " — — LZ: $O(p(n) \cdot 1{,}308^n)$
2SAT $\in$ P