Formale Grundlagen der Informatik 1
Klausur WS2005/2006 bei Prof. Otto

- Chomsky Hierarchie

Typ/Grammatik	*Akzeptor/Algorithmus*	*Abschluss*	*Trennung (nach oben)*
0 – Allgemein **beliebige Produktionen**	**(D)TM akzeptiert/ entscheidet nur teilweise** (rekursiv aufzählbar und nur teilweise entscheidbar)	• **Unter Schnitt, Vereinigung, Konkatenation und Stern** • **Kann nicht unter Komplement abgeschlossen sein, da sie sonst entscheidbar wäre!**	**Existenz einer Grammatik**
1 Kontextsensitiv **Produktionen nicht verkürzend** (also nur harmloses epsilon*** erlaubt)	**(D)TM akzeptiert/entscheidet** (rekursiv aufzählbar und entscheidbar)	• **Unter Komplement, Vereinigung, Stern, Durchschnitt, Konkatenation**	**? (Entscheidbarkeitsgrenze liegt in Typ0)**
2 – Kontextfrei **Auf der linken Seite jeder Produktion nur eine Variable. CNF***** möglich! (X -> α)**	**PDA akzeptiert/ CYK entscheidet** (rekursiv aufzählbar und entscheidbar)	• **Unter Vereinigung, Stern, Konkatenation,** • **Nicht unter Durchschnitt* und Komplement***	**PL für kontextfreie Sprachen**
3 – Regulär **Rechts- (oder Links-) lineare Produktionen (nur eine Variable und ein Terminal)**** **Auch regulärer Ausdruck möglich!**	**NFA akzeptiert/ DFA entscheidet** (rekursiv aufzählbar und entscheidbar)	• **Unter Komplement, Vereinigung, Stern, Durchschnitt, Konkatenation**	**PL für reg. Sprachen**

*Wenn eine Sprache unter Vereinigung/Durchschnitt und Komplement abgeschlossen ist, dann auch unter Durchschnitt/Vereinigung, denn ($A \cup B$) = ($A \cup B$)'' = ($\bar{A} \cap \bar{B}$)'
und auch ($A \cap B$) = ($A \cap B$)'' = ($\bar{A} \cup \bar{B}$)' (De Morgan)

** Nicht verkürzend: α ->β und | β| >= |α| ($\alpha, \beta \in (\Sigma \cup V)^{(*)}$)
*** Harmloses-ε: Xo -> ε und Xo nur als Startsymbol (also nicht nochmal ableitbar)

**** Rechtstlinear: X -> ε | a | aY

***** CNF = Chomsky Normalform: X -> a | YZ

$Typ3 \subset Typ2 \subset Typ1 \subset Typ0 \subset \Sigma - Sprachen$ (echte Teilmengen!)

Grammatik Tricks:
- Vereinigung
  Xo(neu) -> Xo(1) | Xo(2)
- Konkatenation
  Xo(neu) -> Xo(1) Xo(2)
- Sternbildung
  Xo(neu) -> ε | Xo(1) Xo(neu)

- Definitionen
  - Grammatik G = (Σ, V, P, S) (Terminalalphabet, Variablen, Produktionen, Startsymbol)
  - N/D Finite Automaton A = (Σ, Q, q0, Δ/δ, A) (Eingabealalphabet, Zustände, Startzustand, Überführungsrelation/funktion, Akzeptierende (End-)Zustände) $\Delta = Q \times \Sigma \times Q$ (von Z, mit x, nach Z)
  - Pushdown Automaton P = (Σ, Γ, Q, q0, A, Δ, #) (Eingabealalphabet, Kelleralphabet, Zustände, Startzustand, Akzeptierende (End-)Zustände, Überführungsrelation, Keller(leer-)zeichen/symbol)
    $\Delta = Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \times Q$ (von Z, KellerPop, Lesezeichen, KellerPush, nach Z)
  - Turingmaschine M = (Σ, Q, q0, δ, q+, q-) (Bandalphabet, Zustände, Startzustand, Überführungsfunktion, akzepterender Endzustand, verwerfender Endzustandl) $\delta = Q \times (\Sigma \cup \{\square\}) \rightarrow (\Sigma \cup \{\square\}) \times \{<, o, >\} \times Q$
    (von Z, lese vom Band) -> (schreibe aufs Band, Bewege, nach Z)
- Rezepte
  - Determinisierung per Potenzmengenkonstrukt (Aus NFA wird DFA)
    Ein NFA kann über eine Tabelle zum DFA überführt werden:

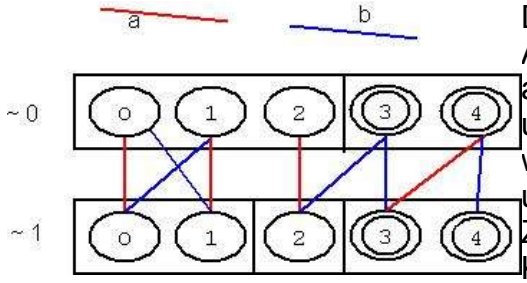
Es ist ganz gut sich vorher ne Tabelle für Δ des Quell A. zu machen, damit man keine ablese Fehler macht!

Zustand	***a***	***b...***
{ Zo }	{ Z1, Z2 }	Ø
{ Z1, Z2 }	{ Z3 }	....

Die Idee ist einfach, dass jede Zustandsmenge simuliert in welchen Zuständen, sich der NFA befinden könnte. Akzeptiert wird daher, wenn einer der Zustände einer Menge akzeptiert! (BTW: Wenn man einen NFA mit mehreren Startzuständen hat, kann man diese auch einfach als Menge sehen mit der man hier los legen kann...)

Jeder Minimale DFA ist isomorph zu allen minimalen DFAs die die gleiche Sprache beschreiben!

- Minimalisierung (vom DFA zum Minimalautomat)
  Ein DFA kann mit Hilfe von ~A bzw. ~L minimalisiert werden



Die Idee ist einfach, die Zustände an Hand ihres Akzeptierverhaltnes zu Unterscheiden. Man fängt also bei ~0 mit einer Trennung der Akzeptierenden und Normalen Zustände an. Dann untersucht man welche Transitionen zwei Zustände einer Klasse unterscheiden, d.h. man schaut ob man z.B. von Z1 mit b in der Klasse bleibt, während man von Z2 mit b in eine andere Klasse kommt (siehe Bsp.Diagram)... Man konstruiert auf diese Weise solange neue ~i+1 bis entweder Alle Zustände eine eigene Klasse haben oder ~i+1 = ~i ist (also keine Änderung mehr eintritt)! (btw: w ~A w' gdw. δhut(qo,w) = δhut(qo,w'))

Q	***ε***	***b***	***ba***
0	-	-	...
1	-	-	...
2	-	+	...
3	+	+	...
4	+	+	...

Trenne alle Zustände an Hand ihres akzeptier Verhaltens per Untersuchung bel. langer Worte

Achtung: Anschließend muss man noch eine Tabelle machen um zu bestätigen, dass die Zustandsanzahl wirklich minimal ist, falls ein Beweis gefordert ist! (Ist zwar eigentlich unnütz, aber immerhin kann man so noch mal überprüfen, dass die Minimierung fehlerfrei ist!)

Die Klassen sind die neuen Zustände

Die neuen Zustände erhalten alle Transitionen der alten aus denen sie zusammengesetzt sind!

BTW: Der Minimale Komplement Automat hat die gleiche Zustandsanzahl!

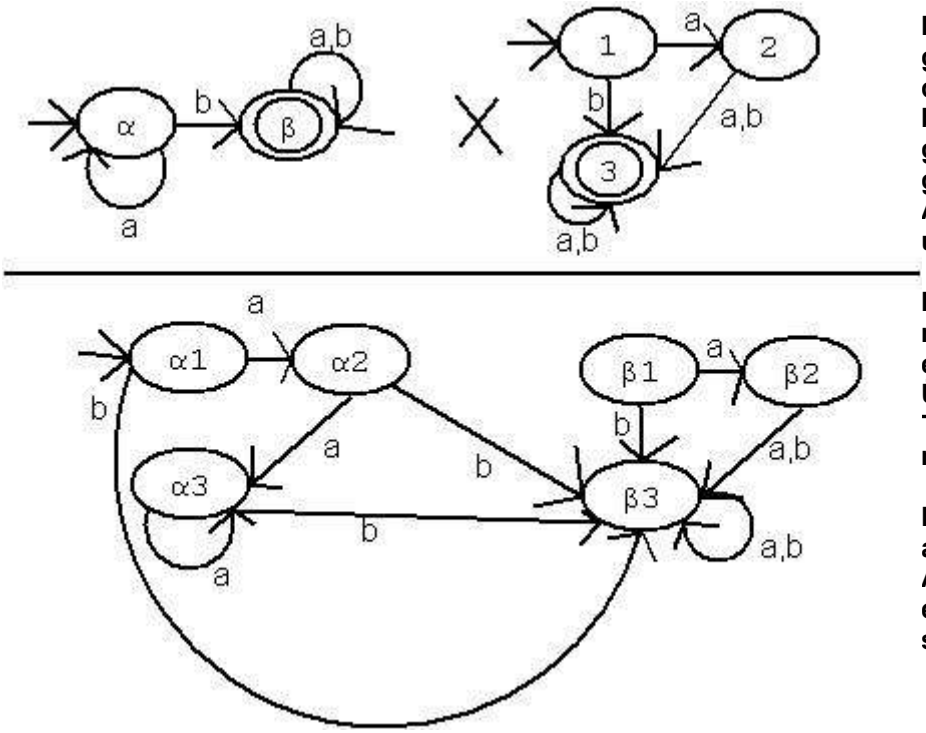
- Produktautomat für Vereinigung/ Durchschnitt
  Wenn man einen Automat basteln will der den Schnitt oder die Vereinigung zweier regulärer Sprachen akzeptiert, dann bastelt man einfach zwei DFAs für jede dieser Sprachen und bastelt daraus einen Produktautomaten.
  Die Überführungsfunktion delta δ sieht dann so aus:
  (Alpha,Beta... steht für Zustände des einen Automaten und 1,2... für die des anderen)

δ	*a*	*b*
α1	α2	β3
α2	α	β
α3	α	β
β1	β2	β3
β2	β	β
β3	β	β

Es wird mit jeder Transition ein Übergang in beiden Automaten durchgeführt.
Ein Zustand simuliert also immer wo sich die beiden Automaten befinden!

Ein Übergang mit a von α1 nach β2 bedeutet zum Beispiel das Automat1 mit a von α nach β wechselt und Automat2 von 1 nach 2...

Statt sich erst die Funktion Tabellarisch darzustellen, kann man auch direkt den Automaten aufzeichnen. Indem man einfach einen der beiden Automaten so oft aufzeichnet, wie der andere Automat Zustände hat!



Im Beispiel sieht man sehr gut, dass z.B. α1 mit a nach α2 geht, da A1 mit a in α bleibt und A2 mit a nach 2 geht. Der Produktautomat geht aber mit b nach β3, da A1 mit b nach β wechselt und A2 mit b nach 3...

In diesem Diagram verliert man aber beim salopp erstellen schnell die Übersicht wo man noch Transitionen eintragen muss...

Deshalb ist eine ganz andere Methode den Automaten in eine Tabelle ein zu zeichnen, aber dazu später mehr.

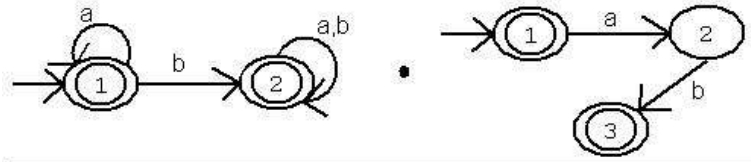
Zunächst zu den akzeptierenden Zuständen des Produktautomaten: Bei Vereinigung akzeptieren einfach alle Zustände die mindestens in einer Komponente einen alten akzeptierenden Zustand hat. Also im Bsp β1, β2, β3 und α3 !
Bei Durchschnitt akzeptieren nur die Zustände die in beiden Komponenten akzeptierende Zustände der ursprünglichen Automaten besitzen!
Also im Bsp nur β3 !
Aber jetzt zum erstellen des Automaten in einer Tabelle. Jedes Feld steht für einen Zustand und es müssen nur noch die Transitionen eingetragen werden:

	α	β
1	α1 b	...
2	...	...
3	...	β3

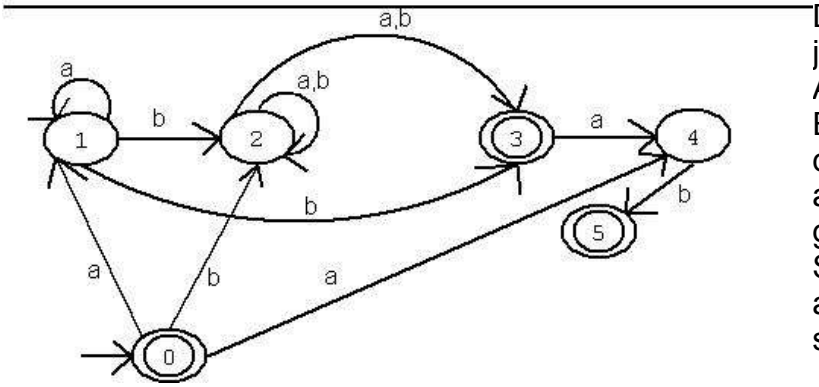
Ganz alternativ kann man auch einfach den einen Automaten x-mal nebeneinander zeichnen und mit Doppelpfeilen den Übergang des anderen Automaten symbolisieren!

- Konkatenations-Automat (NFA)

Will man einen Automaten konstruieren, der die Konkatenation zweier Sprachen akzeptiert. Dann kann man einfach 2 Automaten (einen für jede der beiden Sprachen) hinter einander hängen.



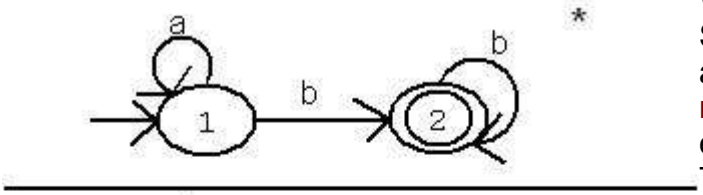
Dazu zeichnet man zunächst beide Automaten nebeneinander (inklusive aller Transitionen).

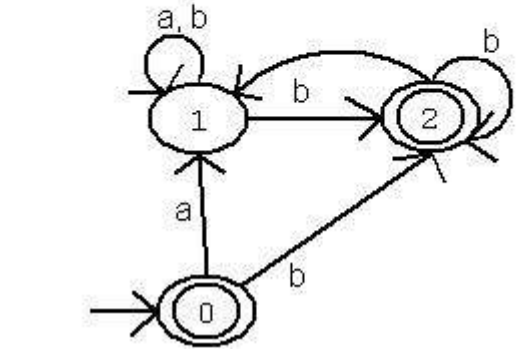


Dann baut man zusätzlich für jede Transition des ersten Automaten in dessen Endzustand, eine Transition in den Anfangszustand des anderen Automaten ein. (Dies gilt insbesondere auch für Schleifen-Transitionen die vom akzeptierenden Zustand auf sich selbst zeigen!)

Akzeptierende Zustände sind nur die des zweiten Automaten!
Achtung: Wenn der erste Automat epsilon akzeptiert, ist es hilfreich einen extra Startzustand einzuführen, der alle Transitionen **BEIDER** Startzustände besitzt! (Ansonsten ist der Startzustand von A1 der neue Startzustand!)
Wenn außerdem beide Automaten epsilon akzeptieren, ist der neue Startzustand akzeptierend!

- Sternautomat (NFA)





Will man einen NFA dazu bringen die Stern-Sprache der aktuellen Sprache zu akzeptieren, baut man erst mal einen neuen akzeptierenden Startzustand (da * enthält epsilon!) und gibt diesem alle Transitionen des alten Starts.
Jetzt muss nur noch für die Schleife gesorgt werden, indem man alle Transitionen die auf einen akzeptierenden Zustand (auch von sich selbst) gehen, zusätzlich auf den **alten** Start lenkt! (Lässt sich auch mit dem neuen Start machen...)

- Komplement DFA

Einen DFA dazu zu bringen, genau das Komplement der Sprache, die er akzeptiert, zu akzeptieren ist einfach. Mann muss lediglich alle Akzeptierenden Zustände zu normalen Zuständen machen und umgekehrt!

- Rückwärtslesen Automat (NFA)
  Angenommen man will die Sprache akzeptieren, die genau aus den Rückwärtsgelesenen Wörtern einer anderen Sprach besteht.
  Dann kann man einfach alle Transitionen umkehren. Der alte Startzustand wird zum Endzustand und die Endzustände zu neuen Startzuständen!!! Das ist natürlich nicht erlaubt, deshalb muss man den erhaltenen Automat determinisieren indem man mit der Menge von Startzuständen anfängt! (Alternativ kann man auch einen neuen Startzustand einführen und ihm alle Transitionen der anderen Startzustände verpassen, oder gar einen epsilon Automaten bauen...)

- PDA aus Grammatik erstellen

Um einen PDA so zu erstellen dass L(PDA) = L(G) ist, kann man sich direkt an der Kontextfreien Grammatik orientieren:
Man braucht nur einen Zustand. Für jede Variable der Grammatik ergänzt man die Überführungsrelation wie folgt:

(q, X, epsilon, α, q) – Im Zustand q, X aufem Keller, kein Zeichen gelesen, ersetze X mit alpha

q = der einzige Zustand, X = die Var der Grammatik, α = nach was X abgeleitet wird (X -> α)

Also, so dass der PDA einfach die Ableitung in den Keller schreibt.
Außerdem braucht man noch weitere Transitionen:

(q, a, a,epsilon,q) Für alle a element Sigma.

Damit kann der PDA wenn der Buchstabe oben auf dem Keller liegt der als nächstes gelesen wird, diesen aus dem Keller entfernen.

Als Kellersymbol wählt man Xo also das Startsymbol der Grammatik!
Der Keller ist dann wie gewohnt fertig, wenn Cf = (q, epsilon, epsilon) eintritt. Man brauchst sich auch nicht drum zu kümmern # am Ende runter zu hohlen, da wir stattdessen ja Xo hatten und das gleich von den Produktionen ersetzt wurde!

Dieser PDA ist natürlich nicht deterministsich (der ganze epsilon Kram)...
Im Script S.60 steht außerdem wie man aus nem PDA eine Grammatik macht...

- Die Berechnung eines PDAs protokolieren
  Diese wird als Konfigurationsfolge gesehen. Man fängt bei Co[w] = (qo, w, #) an und endet bei Cf = (q, ε,ε). (Vgl. Berechnung eines DFAs ist **Zustandsfolge**!)
  Jede Config. beschreibt dabei (Zustand, Resteingabe, Kellerinhalt)

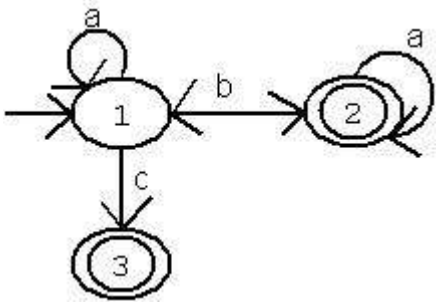
- Reg Ausdruck aus Automat erstellen
  Reguläre Ausdrücke bestehen aus: $\emptyset$, a, (α+β), (α β), α*
  Meistens kann man den Ausdruck direkt am Automaten (vorzugsweise am Minimalautomat) ablesen. Man beschreibt einfach alle möglichen Wege vom Start- zu allen Endzuständen. (auf der nächsten Seite ein Bsp.)
  Bei komplexeren Automaten verstrickt man sich aber leicht. Deshalb gibt es ein strukturiertes Verfahren:
  Man gibt Hilfs-Ausdrücke an und erhöht sukzessive die Menge erlaubter Zwischenzustände.

  Auf diese Weise kann man neue Ausdrücke aus bereits erstellten zusammen setzen. Der Ausdruck für die Sprache ergibt sich dann am Schluss als Summe aus allen Ausdrücken, die vom Startzustand in einen akzeptierenden Zustand mit allen Zuständen als erlaubte Zwischenzustände führen!

$\alpha^{\{\}}_{l\ m}$

{} steht dabei für die Menge erlaubter Zwischenstops (soll am Ende also alle Zustände erfassen) l steht für den „Beginn"zustand einer Transition m für den „End"zustand einer Transition bzw. einer Reise bzw. eines Pfads.

Ein Beispiel fürs direkte Ablesen:



Der Ausdruck für diesen Automat setzt sich zusammen aus dem „Weg“ von 1 nach 2 und von 1 nach 3. Daraus ergibt sich:
α = (a + b a* b)* b a* + (a + b a* b)* c
Grün beschreibt hierbei was der Automat machen kann um erst mal im Startzustand zu verweilen. Rot zeigt dann wie man in den Endzustand 2 kommt und blau wie man in Endzustand 3 kommt.
Das lässt sich natürlich vereinfachen:

α = (a + b a* b)* (b a* + c) Es kommt L(α) = L(A) wie gewünscht heraus!

(das + dies)*	(jenes + dortiges)	(noch)*
um im Start zu bleiben	um zum Endzustand zu kommen	und am ende

- CYK-Algorithmus
  Bestimmung des Wortproblems (Entscheidungsproblem) für Kontextfreie Sprachen in CNF. Der Algo ermittelt sukzessive alle ableitbaren Teilworte. Man fängt dabei mit Worten der Länge 1 also Buchstaben an und ergänzt dann die folgenden Zeilen der Tabelle mit Hilf der vorherigen.
  So sieht man z.B. dass das Feld V2,3 (ac) aus V2,2 und V3,3 zusammen gesetzt werden muss. Mit der Produktion W -> YW wird man dabei fündig.

  Wenn im letzten Feld (V1,n) das Startsymbol auftaucht, weiß man das das Wort erzeugbar ist. Ansonsten ist es nicht teil der Sprache!

```
X -> XY | ZZ | a
Y -> WZ | a
Z -> XY | b
W -> YW | c
```

Länge	b	a	c	b
1	V1,1 Z	V2,2 X,Y	V3,3 W	V4,4 Z
2	V1,2 { }	V2,3 W	V3,4 Y	
3	V1,3 { }	V2,4 X ,Y, Z (X -> XY) (Z -> XY) (Y -> WZ)		
4	V1,4 X (X -> ZZ)			

- Sätze, Lemmas, etc.
  - Satz von Kleene
    L mit Automat erkennbar gdw. L regulär (d.h. mit reg. Ausdruck darstellbar)
    Aus dem Satz leitet sich z.B. die Erstellbarkeit eines reg Ausdruck aus einem Automaten ab und somit die Existenz eines Reg Ausdruck für Komplement(L), etc.

  - Satz Myhill Nerode
    ~L hat endlichen Index gdw. L regulär
    ~L ist eine Äquivalenzrelation. w ~L w' gdw. $\forall x \in \sum^{*} wx \in L \; gdw. \; w'x \in L$ (rechts invarianz) Damit entspricht ~L ~A und es hat einen endlichen Index wenn ~A endlich viele Zustände hat und damit regulär ist! (Index ~L <= Index ~A, bedeutet DFA hat mindestens Index ~L Zustände! Hieraus folgt der Minimalautomat)

- Hinweis: Äquivalenzrelationen
  - reflexiv - w ~L w
  - symetrisch – w ~L w' gdw. w' ~L w
  - transitiv – u ~L v & v ~L w => u ~L w
- Hinweis: Beschreibung von ~L
  - Jeder Zustand des Minimalautomaten entspricht einer Klasse [qo], [q1]...
    Man kann nun einen regulären Ausdruck angeben, der L([qi]) beschreibt in dem man angibt wie man von q0 in den Zustand qi kommt.
    Alle L([qi]) vereinigt ergeben dann wieder rum L(A) (alle akzeptierenden Klassen oder?)

- Pummping Lemma für reguläre Sprachen
  Wenn L regulär => L pumpbar
  Das Pummping Lemma ist vor allem in der Kontrapositon nützlich:
  Wenn L nicht pumpbar => L nicht regulär
  Damit lässt sich per Widerspruchsbeweis zeigen, dass eine Sprache nicht regulär ist.
  Angenommen L ist regulär. Dann sei n wie im PL (FAB). (n = |Q| des Automaten...)
  Dann lässt sich ein $x \in L \, mit \, |x| \geq n$ wählen, dass in x = u v w zerlegt werden kann, mit $v \neq \varepsilon \wedge |uv| \leq n$. (An dieser Stelle wählt man ein konkretes x z.B. a^n b^n)
  Jetzt müsste für alle Zerlegungen und für ein $m \neq 1 \quad x' = uv^m w \in L$ gelten.
  Gibt man an dieser Stelle also (evtl. per Fallunterscheidung) ein m an mit dem x' nicht in L liegt. So tritt ein Widerspruch auf und L kann nicht regulär sein!
  (Bsp. für a^n b^n kann uv nur aus as bestehen, also verändert sich beim pumpen nur die Anzahl as und nicht die der bs und das Wort ist nicht mehr in L!)

- Pummping Lemma für kontextfreie Sprachen
  Wenn L kontextfrei => L pumpbar
  Dieses Lemma ist ähnlich wie das PL für reg. Sprachen.
  Hier lässt sich per **Kontraposition** zeigen, dass eine Sprache nicht kontextfrei ist!
  Angenommen L ist kontextfrei. Dann sei n wie im PL (FAB). (n = 2^|V| der Gram.)
  Dann lässt sich ein $x \in L \, mit \, |x| \geq n$ wählen, dass in x = y u v w z zerlegt werden kann, mit $uw \neq \varepsilon \wedge |uvw| \leq n$. (An dieser Stelle wählt man ein konkretes x z.B. a^n b^n c^n)
  Jetzt müsste für alle Zerlegungen und für ein $m \neq 1 \quad x' = x u^m v w^m y \in L$ gelten.
  Gibt man an dieser Stelle also (evtl. per Fallunterscheidung) ein m an mit dem x' (für alle Zerlegungen) nicht in L liegt, so tritt ein Widerspruch auf und L kann nicht kontextfrei sein!
  (Bsp. für a^n b^n c^n können nur as und bs oder bs und cs gepumpt werden und das Wort ist nicht mehr in L!)

- Beweismethoden
  - Verschiedene Arten von Induktionsbeweisen
    - über die Wortlänge n = |w|
      - IHypthose/Aussage A(n) = Aussage über Worte der Länge n
      - IAnfang A(n) gilt für n=0
      - IISchritt A(n+1) gilt durch Umformung auf A(n) zurückführen
    - über die Anzahl Ableitungsschritte
      - IHypthose/Aussage A(w) = Aussage über Worte (von einer Grammatik) erzeugt
      - IAnfang A(w) gilt für alle Worte die in einem Schritt ableitbar sind (->G)
      - ISchritt A(w) gilt für alle Worte über ->*G mit n+1 Ableitungsschritten

(Zurückführen auf ein Wort mit n Ableitungsschritten und zeigen des letzten nötigen Schrittes)
- über den Erzeugungsprozess (Konkatenation)
  - IHypthose/Aussage A(w) = Aussage über alle Worte der Sprache
  - IAnfang A(w) gilt für w = ε
  - ISchritt A(wa) nachweisen über A(w) und Konkatenation mit a $\forall a \in \Sigma$
- über eine bel. Menge M ($\forall m \in M$ gilt A(m))
  - IHypthose/Aussage A(m) = Aussage über alle Elemente der Menge M
  - IAnfang A(m) gilt für $\forall m \in M0$ (Anfangssume)
  - ISchritt für jede Operation F:
    - für m = F(m1...mn) ist A(m) wahr, wenn A(mi) wahr ist für i=1...n
  - Beispiel: Induktion über (c * c) Terme

- Begriffe und Hinweise
  - Sprachen allgemeines
    - Σ = Alphabet, $a \in \Sigma$ Buchstaben
    - Σ-Wort = endliche Sequenz von Buchstaben aus Sigma
    - Σ* = Menge aller Σ-Worte, $\varepsilon \in \Sigma^*$
    - Σ-Sprache = Menge von Σ-Wörtern, $L \subset \Sigma^*$

Σ+ = Σ* \ ε
$\Sigma^n = \{ w \in \Sigma^* : |w| = n \}$

  - Mengen allgemeines
    - $A \subset B \wedge B \subset A \; gdw. \; A = B$ Äquivalenz durch Inklusionen zeigen
    - $c \in A \cap B \rightarrow c \in A \wedge B$ Durchschnitt
    - $c \in A \cup B \rightarrow c \in A \vee B$ Vereinigung
    - $c \in A \setminus B \rightarrow c \in A \wedge c \notin B$ Mengendifferenz
    - $\overline{B} = M \setminus B$ Komplement bezüglich Obermenge M
    - $A \cap B = \emptyset$ Disjunkte Mengen

BOOLSCHE OPERATIONEN!

  - Reguläre Ausdrücke
    - Reg(Σ) = $\{ \emptyset$, a, α*, α+ β, αβ)
    - L(α+β) = $L(\alpha) \cup L(\beta)$
    - L(α β) = $L(\alpha) \cdot L(\beta)$
      Nützlich um aus komplexen Ausdrücken Automaten zu machen ^^

ε, Σ*, Σ , Σ^+ sind als Abkürzungen auch zugelassen als reg(Σ) !!!!!

  - Grammatiken
    - CNF für Kontextfreie Sprachen
      - X -> a | YZ erlaubt
  - Transitionssystem
    - System mit endlicher Zustandsfolge und Übergängen als Graph mit Knoten und Kanten
  - Relationen (n-Tupel)
    - Kreuzprodukt A x B = Menge aller geordneten paare (a,b)
    - $(a,b) \in R$ bedeutet dass a R b (in Relation R zu einander stehen)
    - Präfixrelation (u,uw)
    - Äquivalenzrelation (reflexiv, symetrisch, transitiv) trennt Menge A in disjunkte Teilmengen
      - Äquivalenzklasse [a]R = $\{ b \in A : a \, R \, b \}$
      - A / R = Menge aller Äquivalenzklassen
      - natürliche Projektion πR : A -> A/R (ordnet jedem Element seine Klasse zu)
      - Index von R = |A / R| = Anzahl Klassen

- Funktionen
  - f : A -> B a |-> f(a)
  - a = Urbild b= f(a) = Bild
  - Surjetkiv = Jedes Element des Bildbereichs wird (min 1x) angenommen)
  - Injektiv = Jedes (angenommene) Element des Bildbereich wird (höchstens) 1x angenommen
  - Bijektiv (beides) = Jedes Element des Bildbereichs wird genau 1x angenommen
- n-Stellige Funktionen = Operationen
  - f: A^n -> A
  - Bsp. *(a,b) = c //jedem n-Tupel wird ein Element zugeordnet!
  - Assoziativität = Klammern vertauschbar
    (a * b) * c = a * (b * c)
  - Kommutativität = drehbar
    a * b = b * a
  - Neutrales Element = ändert nix
    a * e = e * a = a (e = 1)
  - Inverses Element = canceled
    a * a' = a' * a = e (a' = 1/a)

Distributiv = Ausmultiplizierbar!

Junktoren
- $\wedge$ Konjunktion
- $\vee$ Disjunktion
- $\neg$ Negation

A => B ist wahr, wenn A nicht erfüllt ist!

- Algebraische Strukturen
  - Trägermenge (N)
  - Konstante (0)
  - Operation (+)
  - Relation (=)
  - Bsp. (Σ* ,·, ε)
  - Boolsches Algebra:

(B, ^, V, !, 0, 1), (P(M) $,\cup,\cap,\bar{},\emptyset,M$)
Es gelten die Boolschen Axiome

Halbgruppe
-hat assoziative zweistellige Operation *
-wenn die Struktur (A, *, e) ein neutrales Element e hat spricht man von einem Monoid!
-wenn * auch noch Inverse Elemente hat spricht man von einer Gruppe!

- Homomorphismen
  - Strukturerhaltene Abbildung
  - F: A -> B a |-> F(a) FUNKTION
    - A,B Trägermenge der Strukturmenge **A**,**B** des selben Typs
    - $C^A = C^B$ KONSTANTEN
    - $f^A = f^B$ OPERATIONEN
    - $R^A = R^B$ RELATIONEN
  - F ist Homomorphismus gdw:
    - F(C^A) = C^B FÜR ALLE KONSTANTEN
    - $F(f^A(a1...an)) = f^B(F(a1),...F(an))$ FÜR ALLE OPERATIONEN
      Erst auswerten dann überführen = erst überführen dann auswerten
    - $(a1,...an) \in R^A \rightarrow (F(a1)...F(an)) \in R^B$ FÜR ALLE RELATIONEN
- bsp Längenfunktion ||: Σ* -> N von (Σ*, *, ε) -> (N,+,0)
  - ||(ε) = |ε| = 0 ($C^A = C^B$)
  - |u * v | = |u| + |v|
  - u < v => |u| < |v| wäre noch ein Bsp. für die Relationsbedingung ^^
- Isomorphismus
  - bijektive Abbildung zwischen 2 Strukturen!

- Automatentheorie
  - δ-Hut ordnet jedem Wort und Zustand den „End“zustand (nicht akzeptier Zustand) der Berechnung mit diesem Wort zu!
  - Ein DFA kann 2^n soviele Zustände wie der NFA haben (potenzmengenkonstruktion)
  - Leerheitsproblem (L = { })
    Verfahren: Ermittle ob ein akzeptierender Zustand vom Anfangszustand **erreichbar** ist!
  - Äquivalenz Problem (L1 = L2 ?)
    lässt sich mit Hilfe des Leerheitsproblem lösen:
    L1 \ L2 = $L1 \cap \bar{L2}$ = L2 \ L1 = $L2 \cap \bar{L1}$ = $\emptyset$ gdw. L1 = L2
  - L endlich bedeutet endlich viele Wörter in L (keine Schleifen!)
  - Rechtslineare Grammatik -> Automat
    X -> aX' als (x,a,x') Transition |V| = |Q|
    Wenn X-> ε aus X akzeptierenden Zustand machen
- Grammatiken
  - Ableitbarkeit in einem Schritt: x ->G x' gdw. x = uvw ^ x' = uv'w ^ (v,v') in P
  - Ableitbarkeit in * Schritten: x ->*G x' gdw. ein ->G pfad von x nach x' existiert reflexiv (Ableitbarkeit in 0 Schritten ^^) und transitiv
  - w in L(G) gdw. Xo ->*G w ^ w in Σ* (keine Var enthalten)
  - Backus-Naur-Form
    x-> v1 | v2 | v3..
    - Erweiterte BNF (Abkürzungen)
      - Optionalität X -> u [v] w gdw. X -> uw | uvw
      - Schleifen X -> u {v} w gdw. X -> u Z w | uw Z -> ZZ|v
  - Eindeutige Ableitbarkeit einer Grammatik erreichen:
    - Variablen die nach verschiedenen Sachen abgeleitet werden können aufbröseln, so dass man eine bestimmte Sache zu erst ableiten muss, bevor man eine andere ableiten kann.
      - Bsp:
      - E -> (E+E) | (E*E) | x ... umwandeln in
      - E -> (E+F) | (F+E) | F
        F -> (F*F) | x ...
      - um jetzt x * x + x abzuleiten muss man erst das + erstellen, dann das *
  - Wortproblem ist entscheidbar wenn Grammatik nicht verkürzend!
    (Also v -> v' => |v'| >= |v|) Es müssen nur so viele Ableitungen bis |w| erreicht ist (und gehalten wird) untersucht werden!
- TuringMaschienen
  - universelles Berechnungsmodell (Berechenbarkeit & Entscheidbarkeit)
  - alles was prinzipiell algorithmisch lösbar ist, kann mit TM gelöst werden! (Church Turing These)
  - Sequentialität und Determiniertheit
  - endliche Beschreibung der Einzelschritte
  - Jedes Entscheidungsproblem lässt sich auf das Wortproblem einer günstigen Sprache zurückführen
  - Halteproblem ist aufzählbar (alles was hält, hält ^^)
  - DTM hat ja/nein Ausgabe
  - Berechnung ist auch hier eine Folge von Konfigurationen:

C = (α, q, x, β) = (Band links, Zustand, Band unter Lesekopf, Band rechts)
Co[w] = (ε, qo, □, w) – start links vorm Wort. Band kann jederzeit erweitert werden! (aus ε -> □ wenn nötig)
Sobald q = q+ oder q- ist Endkonfiguration erreicht!

- Unendliche Berechnung ist divergent w ->M ∞
  (im Gegensatz zu w ->M STOP)

- Beweis Tipps
  - Mengenverhältnisse wie $A \subseteq B$ am besten über die Elemente zeigen...
    (Beachte $x \in B \rightarrow x \in M$ wenn $B \subseteq M$, brauchbar um Teilmengen zu zeigen!)
    Man muss eigentlich nur aufsplitten von was x alles element ist und dann daraus schließen, dass es auch Element der anderen Seite ist...
  - Das die Allaussage $\forall n \in N\, A(n)$ nicht gilt, lässt sich durch ein Gegenbeispiel $\exists n \in N\, \neg A(n)$ zeigen
  - L(G) = L
    „ ⊇ “ Jedes w aus L ist in G
    Induktion über die Wortlänge!
    „ ⊆ “ Jedes ableitbare Wort von G ist in L
    Induktion über die Anzahl an Ableitungsschritten
  - Bei beweisen L u ... = L \ ...etc.
    kann es nützlich sein zu untersuchen ob ε in nur einer Menge vorkommt
    (Also mit Gegenbeispiel widerlegbar!)
  - Bei Untersuchung von Äquivalenzrelationen und KonkatenationsVerträglichkeit beachten, dass durch Konkatenation neue Worte entstehen
    (Also z.B. bei der Relation (u R1 v) für |u|abb = |v|abb kann durch Konkatenation ein neues abb entstehen und das evlt nur bei u aber nicht bei v!
- Um zu beweisen, dass z.B. eine Sprache nach Verdopplung aller as immer noch regulär ist, kann man eine Abbildung s : reg(Σ) -> reg(Σ) angeben.
  (Man muss angeben was in den Fällen $\emptyset$, a, αβ, α+β,α* geschehen soll. Also im Bsp. s(a) -> aa und s(α+β) = s(α) + s(β)...)

- Sprachbeispiele

Sprache	***Typ***	***Beweis***
a^n b^m	Typ3 regulär	L = L(a*b*) also mit Automat akzeptierbar!
a^n b^n	Typ2 kontextfrei	Pumping Lemma1 gilt nicht! Es wird ein Speicher benötigt! X -> aXb | ε
a^n b^n c^n	Typ1 kontextsensitiv	Pumping Lemma2 gilt nicht. Es wird ein frei benutzbarer Speicher (TM) benötigt!
a^(n^2)	Typ1 kontextsensitiv	PL2 gilt nicht. Wähle x = a^(n^2)
w = a* ^ |w| = 2^n	Typ1 kontextsensitiv	PL2 gilt nicht. Wähle x = a^(2^n)
a^(2n) b^(3n)	Typ2 kontextfrei	PL1 gilt nicht. X -> aaXbbb | ε
a^i b^k c^n ^ i<=k<=n	Typ1 kontextsensitiv	PL2 gilt nicht. Wähle x = a^n b^n c^n
ww (2x das gleiche wort)	Typ1 kontextsensitiv	PL2 gilt nicht. Wähle x = a^n b^n a^n b^n

- Noch ein paar Beweise
  - Beweise das || ein Homomorphismus von (Σ*, *, ε) -> (N,+,0) ist
    d.h. |w * v| = |w| + |v| bzw. f(x*y) = f(x) + f(y)
    Also erst operieren dann überführen = erst überführen dann operieren
    - Z.z.: f(ε) = 0, f(u*v) = f(u) + f(v)
    - Per Induktion:
      - I Hypothese: falls |w| < n dann gilt für jede Zerlegung w = u v
        |w| = |w| + |v|
      - I Anfang (nicht so richtig, eigentlich nur Beweis für die Konstante)
        |ε| = 0
      - I Schritt
        Sei |w| = n, Betrachte Zerlegung w = u v
        - falls v = ε => u = w => |u v| = |w| = |w| + 0 = |w| + |ε|
        - falls v != ε => v = v' a für geeignetes v' aus Σ*, a aus Σ
          |w| = |u v| = |u v' a| = |u v'| + 1
          Laut I Hypothese = |u| + |v'| + 1 = |u| + |v' a| = |u| + |v|
    - Anderer Homomorphismus 2^|w|... Beweis ähnlich
  - Bsp. Induktion über den Erzeugungsprozess:
    - Erzeugung von Worten mit 2 Funktionen
      - f(x) = xb g(x) = xaya Start mit ε
    - IH: A(w) = jedes w in L hat gerade Anzahl von as
    - IA: |ε|a = 0 ist gerade
    - IS: |f(x)|a = |xb|a = |x|a |g(x,y)|a = |xaya|a = |x|a + |y|a + 2 beide gerade
      => A(w) korrekt
  - Allgemeines Bsp Induktion über die Wortlänge:
    - Aussage gilt für wörter der Länge n
    - I Anfang mit Länge 0 also epsilon
    - I Schritt: Wort der Länge n+1 auf Länge n zurück führen...