AFS und Buk

Lernen macht Spaß und diese Definitionen zu lernen macht um so viel mehr Spaß

1 Vorlesungen

1. Vorlesung	2. Vorlesung	3. Vorlesung	4. Vorlesung
5. Vorlesung	6. Vorlesung	7. Vorlesung	8. Vorlesung
9. Vorlesung	10. Vorlesung	11. Vorlesung	12. Vorlesung
13. Vorlesung	Have	fun	studying!!

2 Einführung

- Eine Menge Σ heißt **Alphabet**, falls Σ eine endliche, nicht-leere Menge ist, i.Z.: $|\Sigma| < \infty$ und $\Sigma \neq \emptyset$
- Die Elemente eines Alphabets heißen **Buchstaben** oder **Zeichen**
- Eine **Sprache** L (über Σ) ist eine Teilmenge des von der Menge Σ frei erzeugten Monoids, i.Z.:$L \subseteq \Sigma^*$
- Die Elemente einer Sprache heißen auch **Wörter**
- Das **kartesische Produkt** - $X \times Y = \{(x, y) | x \in X \wedge y \in Y\}$
- **Lemma**: Gilt $|X|, |Y| < \infty$, so gilt $|X \times Y| = |X| \cdot |Y|$
- Eine **Abbildung** $f : X \to Y$ ist eine Vorschrift, die jedem Element aus X höchstens ein Element aus Y zuordnet.
- **Lemma**: Gilt $|X|, |Y| < \infty$, so gilt $|Y^X| = |Y|^{|X|}$
- Ein Element aus X^n heißt auch **Folge der Länge** n **über** X (**Wort** falls X ein Alphabet ist)
- $X^+ := \bigcup_{n \geq 1} X^n$
- **Halbgruppe** - eine Struktur $(H, \circ)$ wobei $\circ$ eine assoziative Verknüpfung auf H ist
- **$(X^+, \cdot)$** - die frei erzeugte Halbgruppe **KEIN MONOID, leeres Wort fehlt**
- **Satz** - $(X^+, \cdot)$ ist eine Halbgruppe
- **Lemma** - Es sei X ein Alphabet mit $|X| > 1$.
  1. Die Konkatenation auf x^+ ist (im Allgemeinen) **nicht** kommutativ
  2. Die Konkatenation ist (im Allgemeinen) **nicht** idempotent
- **Monoid** - eine Struktur $(M, \circ, e)$ mit einer Halbgruppe $(H, \circ)$ und ein neutrales Element e
- $(X^*, \cdot, \lambda)$ ist ein Monoid, das so gennante **frei erzeugte Monoid (über X)**
- **Satz** - Für jede Menge X sind $(2^X, \cup, \emptyset)$ und $(2^X, \cap, X)$ Monoide

- **(Homo-)Morphismus** - eine strukturhaltende Abbildung
- **(Halbgruppen-)Morphismus** - eine Abbildung $h : H \to G$ mit Halbgruppen $(H, \circ), (G, \square)$ so dass $\forall x, y \in H : h(x \circ y) = h(x) \square h(y)$
- **Satz** - Sind $(H, \circ)$ und $(G, \square)$ Halbgruppen und $h : H \to G$ ein Morphismus, so ist $(\{h(x)|x \in H\}, \square)$ eine Halbgruppe. Besitzt $(H, \circ)$ darüber hinaus ein neutrales Element $e \in H$, so ist $h(e)$ neutrales Element von $(\{h(x)|x \in H\}, \square)$.
- **Satz** - Sind $(H, \circ, e)$ und $(G, \square, 1)$ Monoide und $h : H \to G$ ein Morphismus, so ist $(\{h(x)|x \in H\}, \square, 1)$ ein Monoid.
- **Satz** - Die Komposition von zwei Morphismen liefert ein Morphismus.
- Ist G eine Gruppe, dann heißt eine Teilmenge $E \subseteq G$ ein **Erzeugendensystem** von G, wenn sich jedes Element $g \in G$ als endliches Produkt von Elementen aus E und deren Inversen darstellen lässt
- Eine Sprache $L \subseteq \Sigma^*$ heißt **regulär** gdw. es ein endliches Monoid $(M, \circ, e)$ einen Monoidhomomorphismus $h : (\Sigma^*, \cdot, \lambda) \to (M, \circ, e)$ sowie eine endliche Menge $F \subseteq M$ gibt mit $L = \{w \in \Sigma^* | h(w) \in F\}$
- **Sprachfamilie** - Menge von Sprachen

3 Endliche Automaten und reguläre Sprachen

3.1 Deterministische endliche Automaten

- Eine Sprache $L \subseteq \Sigma^*$ heißt **regulär** gdw. es ein endliches Momoid $(M, \circ, e)$, einen Monoidmorphismus $h : (\Sigma^*, \cdot, \lambda) \to (M, \circ, e)$ sowie eine endliche Menge $F \subseteq M$ gibt mit $L = \{w \in \Sigma^* | h(w) \in F\}$
- **deterministischer endlicher Automat(DEA)** - wird beschrieben durch ein Quintupel

$$A = (Q, \Sigma, \delta, q_0, F)$$

Q: endliche Menge von **Zuständen**

Σ: endliche Menge von **Eingabezeichen**

δ: $Q \times \Sigma \to Q$: (totale) **Überführungsfunktion**

q_0: $\in Q$ **Anfangszustand**

F: $\subseteq Q$ **Endzustände**

- **Überführungstafel** - eine Art um ein endlichen Automat vollständig zu beschreiben, z.B.:

δ	a	b	c
$\rightarrow$q	r	r	s
r$\rightarrow$	s	s	r
s	r	s	s

- **L(A)** bezeichnet die von A akzeptierte Sprache
- **Relationenpotenzen** - induktiv Definiert: $R^0 := \Delta_X$ und $R^n := R^{n-1} \circ R$ für $n > 1$
- Ist M eine Menge und $R \subseteq M \times M$ eine zweistellige Relation auf M, dann heißt R **transitiv**, wenn gilt: $\forall x, y, z \in M : xRy \wedge yRz \Rightarrow xRz$
- Die **transitive Hülle** $R^+ := \bigcup_{n \geq 1} R^n$ ist die kleinste R umfassende transitive Relation auf X für gegebenes $R \subseteq X \times X$
- Die **reflexiv-transitive Hülle** $R^* := \bigcup_{n \geq 0} R^n$ ist die kleinste umfassende reflexive und transitive Relation (auch bekannt als Quasiordnung) auf X für gegebenes $R \subseteq X \times X$
- Ein Element aus $C = Q \times \Sigma^*$ heißt **Konfiguration** von A
- Definiere eine Binärrelation $\vdash_A$ auf C durch $(q, w) \vdash_A (q', w')$ gdw. $\exists a \in \Sigma : w = aw'$ und $q' = \delta(q, a)$
- $\vdash_A$ beschreibt den Konfigurationsübergang in einem Schritt (Vorlesung 2, Folie 12)
- Entsprechend beschreibt $\vdash_A^n$ "n Schritte non A"
- Die von einem DEA A akzeptierte Sprache kann man formal wie folgt beschreiben (Vorlesung 2, Folie 12):

$$L(A) = \{w \in \Sigma^* | \exists q \in F : (q_0, w) \vdash_A^* (q, \lambda)\}$$

- $\widehat{\delta} : Q \times \Sigma^* \rightarrow Q, (q, w) \mapsto \begin{cases} q, & w = \lambda \\ \widehat{\delta}(\delta(q, a), w'), & w = aw', a \in \Sigma \end{cases}$
- **Lemma** - Es sei $A = (Q, \Sigma, \delta, , q_0, F)$ ein DEA. Seien $p, q \in Q$ und $w \in \Sigma^*$ beliebig. Dann gilt: $(p, w) \vdash_A^* (q, \lambda) \Leftrightarrow \widehat{\delta}(p, w) = q$.
- **Alternative Definition(en) DEA** - (Vorlesung 2, Folie 15-16)
- **Schlingenlemma** - Es sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA. Es sei $q \in Q$ und sei $X = \{a \in \Sigma : (q, a) \vdash_A (q, \lambda)\}$. Dann gilt: $X^* \subseteq \{w \in \Sigma^* | (q, w) \vdash_A^* (q, \lambda)\}$.
- **Lemma** - $L(A) = A$ mit $L := \{a^n b^m | n \geq 0, m \geq 1\}$
- **Satz** - Wird $L \subseteq \Sigma^*$ von einem DEA erkannt, so ist L regulär
- **Satz** - Ist $L \subseteq \Sigma^*$ regulär, so wird L von einem DEA erkannt

3.2 Nichtdeterministische endliche Automaten

- Vereinigung, Durchschnitt, Kompliment,... von Sprachen liefern wieder Sprachen, sind also **Operationen auf Sprachen**

  Ist f eine k-stellige Operation auf Sprachen und ist L eine Sprachfamilie, so heißt L **abgeschlossen gegen** f gdw. für alle für alle $L_1, ..., L_k \in L$ gilt $f(L_1, ..., L_k) \in L$

- **Satz** - **DEA**-Sprachen sind komplementabgeschlossen

- **Sprachkomplement** entspricht Endzustandsmengenkomplement ($w \in L(A) \rightarrow w \notin L(A')$)

- **nichtdeterministischer endlicher Automat(NEA)** - wird beschrieben durch ein Quintupel

$$A = (Q, \Sigma, \delta, q_0, F)$$

  Q: endliche Menge von **Zuständen**

  Σ: endliche Menge von **Eingabezeichen**

  δ: $Q \times \Sigma \times Q$: **Überführungsrelation**

  q_0: $\in Q$ **Anfangszustände**

  F: $\subseteq Q$ **Endzustände**

  - Sprachfamilie: **NEA**

- **Überführungstafel** - eine Art um ein endlichen Automat vollständig zu beschreiben, z.B.:

δ	a	b
→q	q	r
r→	q	s
s	s	s
→q'	r'	q'
r'→	s'	q'
s'	s'	s'

- **Satz** - Jede endliche Sprache ist **NEA**-Sprache

- **Skelettautomaten** - Beispiel auf: Vorlesung 3, Folie 16

- **Unterschiede DEA/NEA Spezifikation** anhand der Überführungstafel; wie DEA, ABER

  - Einträge dürfen leer sein, d.h. der Automat ist **unvollstandig**
  - Es gibt mehr als ein Eintrag(Nichtdeterminismus)
  - Es gibt eine Anfangszustandsmenge(Nichtdeterminismus)

  - Manchmal werden neben Buchstaben auch Wörter als Spaltenüberschrift zugelassen, insbesondere das leere Wort: **NEA mit λ-Übergängen**

- **Satz** - **NEA** ist unter Vereinigung abgeschlossen
- **Satz** - NEAs kennzeichnen die regulären Sprachen
- **Lemma** - $L_k = \{x \in \{0,1\}^* | l(x) \geq k, \text{das k-letzte Zeichen von x ist } 0\}$
  L_k kann durch einen NEA mit $k+1$ Zuständen erkannt werden, aber durch keinen DEA mit weniger als 2^k Zuständen
- **Satz** - Zu jedem λ-NEA gibt es einen äquivalenten NEA
- **Lemma** - Zu jedem NEA (mit λ-Übergang) gibt es einem äquivalenten NEA mit λ-Übergängen, der nur einen Anfangs- und nur einen Endzustand besitzt; der Anfangszustand hat nur ausgehende Kanten und der Endzustand nur eingehende Kanten.

3.3 Reguläre Ausdrücke

- **Satz** - REG ist gegen Komplementbildung abgeschlossen
- Es seien $(M, \circ, e)$ und $(N, \square, 1)$ Monoide. Dann kann man die Menge $M \times N$ zu einem Monoid machen durch *komponentenweises Anwenden* der Operationen; definiere daher: $(m,n)[\circ,\square](m',n') := (m \circ m', n \square n')$

  Satz - $(\mathcal{M} \times \mathcal{N}, [\circ,\square], (e,1))$ ist ein Monoid, das Produktmonoid.(siehe DS)

  Satz - Sind $h_M : (X, \Delta, I) \to (M, \circ, e)$ und $h_N : (X, \Delta, I) \to (N, \square, 1)$ Monoidmorphismen, so auch der Produktmorphismus $h_M \times h_N : X \to M \times N, x \rightarrowtail (h_M(x), h_N(x))$
- **Satz** - Ist f eine k-stellige Mengenoperation, so ist REG gegen f abgeschlossen
- Ist $(M, \circ, e)$ ein Monoid, so kann die Menge 2^M durch das **Komplexprodukt** zu einem Monoid gemacht werden. Dazu definieren wir: $A \circ B := \{a \circ b | a \in A \wedge b \in B\}$ Das zugehörige neutrale Element in $\{e\}$
- **Satz** - REG ist gegen Konkatenation abgeschlossen
- **Definition** - $A^+ = \bigcup_{n \geq 1} A^n$
- **Definition** - (Kleene-Stern) $A^* = \bigcup_{n \geq 0} A^n$
- **Satz** - Satz $L^+(L^*)$ ist die (das) durch L bezüglich der Konkatenation erzeugte Halbgruppe (Monoid)
- **Satz** - REG ist gegen Kleene-Stern abgeschlossen
- **Reguläre Ausdrücke** über festem aber beliebigem Alphabet Σ - Definition durch strukturelle Induktion:
  - $\emptyset$ und a sind RA (über Σ) für edes $a \in \Sigma$

- Ist R ein RA (über Σ), so auch $(R)*$
- Sind R_1 und R_2 RAs (über Σ) so auch R_1R_2 und $(R_1 \cup R_2)$
- **Beispiel**: $((\mathsf{b} \cup \mathsf{a})) * \mathsf{aaa}(\mathsf{bb})*$

- **Durch einen RA beschriebene Sprache** - induktiv gegeben:
  - $L(\emptyset) = \emptyset; L(\mathsf{a}) = \{a\}$
  - Ist R ein RA, setze $L((R)*) = (L(R))*$
  - Sind R_1 und R_2 RA, setze $L(R_1R_2) = L(R_1) \cdot L(R_2)$ und $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$

  Ein RA über Σ beschreibt also eine Sprache über Σ

- **Satz** - Jede RA-Sprache ist regulär
- **Satz** - Jede reguläre Sprache ist durch einen RA beschreibbar
- **dynamisches Programmieren** Vorlesung 4 Folie 20,23,25

3.4 Nichtreguläre Sprachen

- **Pumping Lemma** - Zu jeder regulären Sprache L gibt es eine Zahl $n > 0$, so dass jedes Wort $w \in L$ mit $l(w) \geq n$ als Konkatenation $w = xyz$ dargetellt werden kann mit geeigneten x, y, z mit folgenden Eigenschaften:
  1. $l(y) > 0$;
  2. $l(xy) < n$;
  3. $\forall i \geq 0 : xy^iz \in L$

- **Anwendung des Pumping Lemma**
  1. Wir vermuten, eine vorgegebene Sprache L ist nicht regulär
  2. Im Widerspruch zu unserer Annahme nehmen wir an, L wäre doch regulär. Dann gibt es die im Pumping-Lemma genannte **Pumping-Konstante** n
  3. Wir wählen ein geeignetes, hinreichend langes Word $w \in L$ (d.h., $l(w) \geq n$). Dies ist der Schritt, wo man leicht "gut" oder "schlecht" wählt! Bemerkung: Da wir ja vermuten, L ist nicht regulär, ist L insbesondere unendlich, d.h., zu jedem n finden wir ein $w \in L$ mit $l(w) \geq n$
  4. Wir diskutieren alle möglichen Zerlegungen $w = xyz$ mit $l(y) > 0$ und $l(xy) \leq n$ und zeigen für jede solche Zerlegung, dass es ein $i \geq 0$ gibt, sodass $xy^iz \notin L$ gilt.

- **Spiegeloperation** - Informell: $w^R = w$ Rückwärtsgelesen. Induktiv: $\lambda^R = \lambda$; für $w = va$ mit $v \in \Sigma^*, a \in \Sigma$ definiere: $w^R := a(v^R)$

- **Satz** - Die regulären Sprachen sind unter Spiegelung abgeschlossen

- **Palindrom** - $L = \{w \in \{a,b\}^* | w = w^R\}$
- Folien 10-12 angucken
- **Äquivalenzrelation** - Eine Relation $R \subseteq X \times X$ mit:
  1. $R^0 = \Delta_X \subseteq R$ (Reflexivität)
  2. $R^2 = R \circ R \subseteq R$ (Transitivität)
  3. Mit $R^{-1} = \{(y,x)|(x,y) \in R\}$ gilt $R^{-1} \subseteq R$ (Symmetrie)

  Eine ÄR auf X induziert eine Partition von X in **Äquivalenzklassen** $[x]_R = \{y \in X | xRy\}$
- Definiere $x \equiv_h y$ gdw. $h(x) = h(y)$, mit $h : (\Sigma^*, \cdot, \lambda) \to (M, \circ, e)$ ein Monoidmorphismus

  Satz - $x \equiv_h y$ ist eine Äquivalenzrelation auf Σ^*

  Erinnerung - Der Kern eines Homomorphismus ist (sogar) eine Kongruenzrelation, also eine Äquivalenzrelation, die mit den Monoid-Operationen verträglich ist
- Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein vollständiger DEA. Definiere $u \equiv_A v$ gdw. $\exists q \in Q : ((q_0 u) \vdash_A^* (q, \lambda)) \wedge ((q_0 v) \vdash_A^* (q, \lambda))$

  Satz - $u \equiv_A v$ ist eine Äquivalenzrelation auf Σ^*
- **Definition** - L **trennt** zwei Wörter $x, y \in \Sigma^*$ gdw. $|\{x,y\} \cap L| = 1$
- **Definition** - Zwei Wörter u und v heißen **kongruent modulo L**(i.Z.:$u \equiv_L v$), wenn für jedes beliebige Wort w aus Σ^* die Sprache L diw Wörter uw un vw nicht trennt, d.h. wenn gilt: $(\forall w \in \Sigma^*)(uw \in L \Leftrightarrow vw \in L)$
- **Satz** - Für jede Sprache $L \subseteq \Sigma^*$) ist $\equiv_L$ eine Äquivalenzrelation
- **Definition** - $\equiv_L$ heißt auch Myhill-Nerode Äquivalenz
- **Beobachtung** - Gilt $u \in L$ und $v \equiv_L u$, so auch $v \in L$
- **Lemma** - $\equiv_L$ ist sogar eine **Rechtskongruenz**, d.h., aus $u \equiv_L v$ folgt für bel. Wörter $x \in \Sigma^* : ux \equiv_L vx$.

  Zu zeigen bliebe dazu: $(\forall w \in \Sigma^*)(uw \in L \Leftrightarrow vw \in L)$ impliziert: $(\forall x, w' \in \Sigma^*)(uxw' \in L \Leftrightarrow vxw' \in L)$
- **Lemma** - Es sei $L \subseteq \Sigma^*$ regulär, d.h. L ist durch ein endliches Monoid $M, \circ, e)$, ein Monoidmorphismus $h : \Sigma^* \to M$ und eine endliche Menge $F \subseteq M$ beschrieben. Dann gilt: Falls $u \equiv_h v$ so $u \equiv_L v$
- **Folgerung** - Ist L regulär, so hat $\equiv_L$ nur endlich viele Äquivalenzklassen

- **Folgerung** aus dem letzten Beweis (VL5F20) - Betrachte reguläre Sprache $L \subseteq \Sigma^*$ und sie beschreibende Homomorphismen h bzw. Automaten A:

  Ist $\mathcal{L} := \{L_1, \ldots, L_n\}$ die durch $\equiv_L$ induzierte Zerlegung von Σ^*, so gilt für die durch $\equiv_h$ induzierte Zerlegung $\mathcal{H} := \{H_1, \ldots, H_n\}$ von Σ^*(bzw. für die durch $\equiv_A$ induzierte Zerlegung $\mathcal{A} := \{A_1, \ldots, A_n\}$ von Σ^*):

  Für jedes H_i(bzw. A_i) gibt es ein L_j mit $H_i \subseteq L_j$(bzw. $A_i \subseteq L_j$)

  Daher heißen $\mathcal{H}$ und $\mathcal{A}$ auch **Verfeinerung**en von $\mathcal{L}$

- **Satz**[Myhill und Nerode] - Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn es nur endlich viele Äquivalenzklassen bezüglich $\equiv_L$ gibt.

- Definiere de **Minimalautomaten** $A_{min}(L) = (Q, \Sigma, \delta, s_0, F)$ durch

  $Q = \{[x_1], ..., [x_k]\}$

  $q_0 := [\lambda]$

  F bestehe aus allen Äquivalenzklassen $[x_i]$ mit $x_i \in L$

  $\delta([x], a) := [xa]$

  Wichtig: Mit $[x] = [y]$ ist $xaw \in L \Leftrightarrow yaw \in L$, also auch

  $$[xa] = [ya], \rightsquigarrow \delta([x], a) = [xa] = [ya] = \delta([y], a)$$

- **Lemma** - Ist L regulär, so ist $A_{min}(L)$ der L akzeptierende DEA mit der kleinsten Anzahl von Zuständen.

- **Definition** - Es seien $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ und $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ DEAs.

  Eine Funktion $f : Q_1 \to Q_2$ heißt **Automaten(homo)morphismus** von A_1 nach A_2 gdw.:

  - Für alle $a \in \Sigma$ und für alle $q \in Q_1$ gilt $f(\delta_1(q, a)) = \delta_2(f(q), a)$
  - $f(q_{01}) = q_{02}$
  - Für alle $q \in Q_1$ gilt: $q \in F_1 \Leftrightarrow f(q) \in F_2$

  Ist f bijektiv, ist f ein **Automatenisomorphismus**

- **Satz** - Es seien $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ und $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ DEAs und $f(Q_{01}) = Q_{02}$ ein Automatenmorphismus. Dann gilt: $L(A_1) = L(A_2)$

- **Lemma** - Der Minimalautomat ist "Bis auf Isomorphie" (also bis auf Umbenennen der Zustände) eindeutig bestimmt

- **Folgerung**[Aus dem Satz von Myhill und Nerode] - Hat $\equiv_L$ unendlich viele Äquivalenzklassen, so ist L nicht regulär

- **Definition** - $\equiv_L^{synt}$ auf VL5 F29

- **Definition** - Es sei $L \subseteq \Sigma^*$. $u, v \in \Sigma^*$ heißen **syntaktisch kongruent modulo L**, i.Z. $u \equiv_L^{synt} v$ gdw. $\forall x, y \in \Sigma^* : (xvy \in L \Leftrightarrow xvy \in L)$

- **Satz** - Für jede Sprache L ist $u \equiv_L^{synt} v$ eine Kongruenzrelation

  Auf der Menge der Kongruenzklassen ist die "Konkatenation" $[u] \cdot [v] := [uv]$ wohldefiniert und macht diese zu einem Monoid, dem **syntaktischen Monoid** von L

- **Folgerung** - Eine Sprache ist regulär gdw. sie besitzt ein endliches syntaktisches Monoid.

- **Satz** - Ist L regulär, so ist das syntaktische Monoid non L isomorph zum Transformationsmonoid des Minimalautomaten $A_{min}(L)$ von L

- **Folgerung** - Ist $L \subseteq \Sigma^*$ regulär und $(M, \circ, e)$ das Transformationsmonoid von $A_{min}(L)$ mit zugehörigem Morphismus $h : \Sigma^* \to M$, so ist $\equiv_h$ die syntaktische Kongruenz von L

3.5 Logik und endliche Automaten

- **Grunüberlegungen zu formalen Logik**

  Wir gehen davon aus, es gäbe eine Menge $\mathfrak{A}$ von **atomaren Formeln**

  Über ein Teil $\mathfrak{D} \subseteq \mathfrak{A}$ dieser Formeln "wissen wir Bescheid", d.h., wir können eine Abbildung $\beta : \mathfrak{D} \to \{0, 1\}$ angeben mit der Bedeutung:

  - $\beta(a) = 0.$ falls a falsch ist;
  - $\beta(a) = 1.$ falls a wahr ist.

  $\mathfrak{D}$ ist also eine Menge definierter Aussagen, und β ist eine **Belegungsfunktion**. $\mathfrak{A} \setminus \mathfrak{D}$: **logische Variablen** oder **Unbestimmte**

  Wir wollen dann zusammengesetzte Aussagen untersuchen

  Wir beschreiben nun, was das formal bedeutet

- **()Aussagenlogische) Formeln** werden durch einen induktiven Prozess definiert:

  - Jede atomare Formel ist eine Formel
  - Ist F eine Formel, so auch $\neg F$ **Negation von F**
  - Sind F und G Formeln, so auch $(F \wedge G)$ **Konjunktion von F und G**

  Eine Formel, die als Teil einer Formel in F auftritt, heißt **Teilformel** von F. $\mathfrak{F}$ bezeichne die Gesamtheit aller aussagenlogischen Formeln (bzgl. $\mathfrak{A}$).

- **Übliche Abkürzungen**

  - $(F \vee G)$ steht für $\neg(\neg F \wedge \neg G)$ **Disjunktion von F und G**
  - $(F \to G)$ steht für $(\neg F \vee G)$ **Implikation**
  - $(F \leftrightarrow G)$ steht für $((F \to G) \wedge (G \to F))$ **Äquivalenz**

  Die Objekte $\neg, \wedge, \vee, \to, \leftrightarrow$ heißen auch **Junktoren**

- Ist F eine Formel, so bezeichne $\mathfrak{A}(F)$ die in F vorkommenden atomaren Formeln

  Mit $\mathfrak{D} \subseteq \mathfrak{A}$ heißt $\beta : \mathfrak{D} \to \{0,1\}$ **passend** zu F, falls $\mathfrak{A}(F) \subseteq D$.

  Ist β eine zu F passende Belegungsfunktion, so heißt β ein **Modell** für F, falls $\beta(F) = 1$. Man schreibt dann auch: $\beta \vDash F$.

  Zwei Formeln F und G heißen **(semantisch) äquivalent** gdw. für jede Belegungsfunktion β, die sowohl für F als auch für G passend ist, gilt: $\beta(F) = \beta(G)$.

  Man schreibt dafür auch: $F \equiv G$.

- **Satz** - (**Idempotenz**) Für jede Formel F gilt: $F \equiv (F \wedge F)$

- **Lemma** - **(Absorption)** $F \equiv (F \wedge (F \vee G))$

3.5.1 Syntax von Büchis Logik erster Stufe über Alphabet Σ

Atomare Formeln: *wahr*, $x \leq y$ und $R_a x$, wobei x und y (Positions-)Variablen sind und $a \in \Sigma$. Daraus induktiv **Formeln erster Stufe**:

- Atomare Formeln sind Formeln erster Stufe.
- Sind ϕ und ψ Formeln erster Stufe, so auch $(\neg\phi), (\phi \wedge \psi), (\phi \vee \psi)$
- Ist ϕ eine Formel erster Stufe und ist x eine Variable, so sind auch $(\exists x\phi)$ und $(\forall x\phi)$ Formeln erster Stufe

Wieder Begriffe wie gebundene Variablen(vorkommen), freie Variablen, Aussagenform usf. Die Menge der freien Variablen FV(ϕ) von Formel ϕ könnte man auch einfach induktiv definieren:

- $FV(x \leq y) = \{x, y\}, FV(R_a x) = x$
- Sind ϕ und ψ Formeln erster Stufe, so gilt:

  $FV(\neq \phi) = FV(\phi)$

  $FV(\phi \wedge \psi) = FV(\phi \vee \psi) = FV(\phi) \cup FV(\psi)$

- Ist eine Formel erster Stufe und x eine Variable, so ist $FV(\exists x\phi) = FV(\forall x\phi) = FV(\phi) \setminus \{x\}$

- Erinnere: Elemente aus Σ^* sind Wörter über Σ oder auch Abbildungen $[n] \to \Sigma$. Speziell:$\lambda : [0] \to \Sigma$ mit $[0] = \emptyset$.

  Formeln sollen Mengen von Wörtern über Σ beschreiben (als Modelle).

  "Mögliche Welten" sind im engeren Sinne $\{[n] | n \in \mathbb{N}\}$ mit Halbordnung $\leq$.

  $Dom(u)$ gibt dann den Definitionsbereich von $u \in \Sigma^*$ an.

  R_a a soll die Menge der Positionen von Vorkommen von Zeichen a angeben.

  Bsp.: $\Sigma = \{a, b\}, u = abbaab, Dom(u) = [6], R_a = \{0, 3, 4\}, R_b = \{1, 2, 5\}$.

- **Belegungen** sind Abbildungen $V \to Dom(u)$ für Variablenmenge V und $u \in \Sigma^*$

  (u, v) ist Modell für Formel ϕ, kurz $(u, v) \vDash \phi$, falls folgende induktive Definition für u mit Belegung v zutrifft. Hierbei ist $V = FV(\phi)$.

  $(u, v) \vDash (x \leq y)$ gdw. $v(x) \leq v(y)$ sowie $(u, v) \vDash R_a x$ gdw. $x \in R_a$;

  Sind ϕ und ψ Formeln erster Stufe, so gilt:

  - $(u, v) \vDash \neg\phi$ gdw. (u, v) ist kein Modell für ϕ;
  - $(u, v) \vDash (\phi \wedge \psi)$ gdw. $(u, v) \vDash \phi$ und $(u, v) \vDash \psi$
  - $(u, v) \vDash (\phi \vee \psi)$ gdw. $(u, v) \vDash \phi$ oder auch $(u, v) \vDash \psi$

  Ist ϕ eine Formel erster Stufe und x eine Variable, so ist:

  - $(u, v) \vDash (\exists x\phi)$ gdw. es gibt $d \in Dom(u)$, sodass$(u, v[x \mapsto d]) \vDash \phi$
  - $(u, v) \vDash (\forall x\phi)$ gdw. für alle $d \in Dom(u)$ gilt$(u, v[x \mapsto d]) \vDash \phi$

  Ist $FV(\phi) \vDash \emptyset$, so sagen wir auch: **u erfüllt** ϕ, falls $(u, \emptyset) \vDash \phi$.

- Ist eine Formel über Σ ohne freie Variablen, so ist $L(\phi) = \{u \in \Sigma^* | u \text{ erfüllt } \phi\}$

  Dann gilt für beliebige Alphabete Σ mit $a \in \Sigma$: $L(\phi) = \{a\}\Sigma^*$ und $L(\psi) = L(\phi) \cup \{\lambda\}$.

  Beispiele auf Vorlesung 6, Folie 19 evtl relevant

- **Logik zweiter Stufe** erweiter Syntax um Automare Formeln (Xx) sowie Formeln der Bauart $(\exists X\phi)$ und $(\forall X\phi)$;

  - erweitert Semantik so, dass X als Mengen von Positionen interpretiert werden;
  - entsprechend wird v (induktiv) erweitert;
  - v ordnet Mengenvariablen Mengen zu.
  - $(u, v) \vDash (Xx)$ gdw. $v(x) \in v(X)$;
  - $(u, v) \vDash (\exists X\phi)$ gdw. es gibt $D \subseteq Dom(u)$, sodass $(u, v[X \mapsto D]) \vDash$;
  - $(u, v) \vDash (\forall X\phi)$ gdw. für alle $D \subseteq Dom(u)$ gilt $(u, v[X \mapsto D]) \vDash$.
  - Damit kann dann schließlich wieder $L(\phi)$ definiert werden.

- **Satz von Kleene/Büchi I**: Jede reguläre Sprache ist MSO-definierbar.

  Idee: $L \subseteq \Sigma^*$ wird durch DEA beschrieben.

  Für jedes $w \in \Sigma^*$ induziert Durchlauf der Zustandsmenge Q Partition von $Dom(w)$ in $\leq |Q|$ Klassen. Wird die leere Menge als Klasse zugelassen, so sind es o.E. $|Q|$ viele Klassen; sei $Q = [n]$.

  Formel für L soll also beschreiben:

  - $X_0, ..., X_{n-1}$ ist Klasseneinteilung;
  - die Zustandsübergänge werden befolgt;
  - Anfangs- und Endzustände werden beachtet.

- Klasseneinteilung: $(\bigwedge_{q \neq p} \neg\exists x(X_q x \wedge X_p x)) \wedge (\forall x \bigvee_q X_q x)$
- Zustandsübergänge (bis auf Wortende): $\forall x \forall y(S(x,y)) \mapsto \bigvee_{q \in Q} \bigvee_{a \in \Sigma} (X_q x \wedge R_a y \wedge X_{\delta(q,a)} y)$
- Zusammen Formel für L: $\exists X_0 ... \exists X_{n-1}$(Klasseneinteilung $\wedge$ Zustandsübergänge $\wedge$ Randbedingungen)
- Satz: REG ist abgeschlossen gegenüber Homomorphismen.

  Das meint: Ist $h : \Sigma^* \mapsto \Gamma^*$ ein Homomorphismus und ist $L \subseteq \Sigma^*$ regulär, so ist $h(L) \subseteq \Gamma^*$ regulär.

  Idee: Arbeite induktiv mit RA-Definition.
- (p,q)**-erweiterte Alphabete** $\Sigma_{(p,q)} = \Sigma^* \times \{0,1\}^p \times \{0,1\}^q$.

  Beachte natürliche Bijektion $f : \Sigma^n_{(p,q)} \mapsto \Sigma^n \times (\{0,1\}^n)^{p+q}$ für alle n.

  $h : \{0,1\}^* \mapsto \{0,1\}^*$ sei Morphismus, gegeben durch $h(1) = 1$ und $h(0) = \lambda$.

  π_i : Projektion auf Komponente i. Hier: $\pi_0, \pi_1, ..., \pi_{p+q}$ sinnvoll.

  $K_{p,q} = \{\lambda\} \cup \{w \in \Sigma^+_{(p,q)} | \forall i = 1, ..., p : \ell(h(\pi_i(f(w)))) = 1\}$

  Wegen Durchschnitts- und Homomorphismen-Abgeschlossenheit von REG folgt: $K_{(p,q)} \in$ REG.
- Zusammenhang Logik zweiter Stufe und erweiterte Alphabete

  Betrachte $(u_0, u_1, ..., u_p, u_{p+1}, ..., u_{p+q})\Sigma^n \times (\{0,1\}^n)^{p+q}$.

  Deute R_a als $\{i \in Dom(u_0) | u_0(i) = a\}$;

  Variable x_i belegt durch die eindeutig bestimmte Position j, für die $u_i(j) = 1$ gilt.

  Variable X_i meint Menge der Positionen j, für die $u_{p+i}(j) = 1$ gilt.

  So können wir davon sprechen, dass $u \in K_{(p,q)}$ eine Formel $\phi(x_1, ..., x_p, X_1, ..., X_q)$ erfüllt und umgekehrt ϕ die Sprache $L_{p,q}(\phi) \subseteq K_{p,q}$ zuordnen.
- Hilfsalphabete:

  $C_i = \{w \in \Sigma_{(p,q)} | \pi_i(w) = 1\}$

  $C_{i,a} = \{w \in C_i | \pi_0(w) = a\}$
- **Satz von Kleene/Büchi II** Idee: Beschreibe für jede Formel die Sprache $L_{p,q}(\phi) \subseteq K_{p,q}$ induktiv.
  - $L_{p,q}(R_a x_i) = K_{p,q} \cap \Sigma^*_{(p,q)} C_{i,a} \Sigma^*_{(p,q)}$;
  - $L_{p,q}(x_i \leq x_j) = K_{p,q} \cap \Sigma^*_{(p,q)} (C_i \Sigma^*_{(p,q)} C_j \cup (C_i \cap C_j)) \Sigma^*_{(p,q)}$;
  - $L_{p,q}(X_j x_i) = K_{p,q} \cap \Sigma^*_{(p,q)} (C_i \cap C_{j+p}) \Sigma^*_{(p,q)}$;
  - $L_{p,q}(\phi \vee \psi) = L_{p,q}(\phi) \cup L_{p,q}(\psi)$;

- $L_{p,q}(\phi \wedge \psi) = L_{p,q}(\phi) \cup L_{p,q}(\psi)$;
- $L_{p,q}(\neg) = K_{p,q} \setminus L_{p,q}(\phi)$;
- $l_i(w)$: Löschen der i-ten Komponente von $w \in \Sigma^+_{(p,q)}$, interpretiert als $f(w)$.

Beachte: $l_i : \Sigma^*_{(p,q)} \mapsto \Sigma^*_{(p-1,q)}$ falls $1 \leq i \leq p$ und $l_i : \Sigma^*_{(p,q)} \mapsto \Sigma^*_{(p,q-1)}$, falls $i > p$, sind auffassbar als Morphismen.

Nach de Morgan genügt nun die Interpretation der Existenzquantoren: $L_{p-1,q}(\exists x_i \phi) = l_i(L_{p,q}(\phi))$ und $L_{p,q-1}(\exists X_i) = l_{i+p}(L_{p,q}(\phi))$. Enthält schließlich ϕ keine freien Variablen mehr, so gilt: $L(\phi) = L_{0,0}(\phi)$ ist regulär wegen der bekannten Abschlusseigenschaften der regulären Sprachen.

3.6 Algorithmen mit/für endliche Automaten

- **Wann ist ein DEA A nicht minimal?**

  Wenn es nicht erreichbare Zustände gibt, d.h. es gibt q mit $(q_0, y) \vdash^*_A (q, \lambda)$ für kein Wort $y \in \Sigma^*$

  Wenn es Zustände $q \neq q'$ gibt mit $\forall w \exists p, p' \in Q : |\{p, p'\} \cap F| \neq 1 \Rightarrow ((q, w) \vdash^*_A (p, \lambda) \Leftrightarrow (q', w) \vdash^*_A (p', \lambda))$ d.h. q und q' sind nicht **trennbar**, sondern **äquivalent**

- Es bezeichne $[q]$ die Menge aller Zustände, die zu q äquivalent sind
- **Lemma** - "Nicht-Trennbarkeit" ist tatsächlich eine Äquivalenzrelation uf Q
- **Eigenschaften äquivalenter Zustände**
  1. Sind q und q' äquivalent, dann auch $\delta(q, a)$ $\delta(q', a)$
  2. Sind q und q' äquivalent, dann gilt $q \in F \Leftrightarrow q' \in F$

3.6.1 Konstruktion des Minimalautomaten

- Definiere zu $A = (Q, \Sigma, \delta, q_0, F)$ neuen Automaten $A_{[]} = (Q_{[]}, \Sigma_{[]}, \delta_{[]}, [q_0], F_{[]})$ mit:
  - Anfangszustand $[q_0]$
  - Endzuständen $F_{[]} := \{[q] | q \in F\}$
  - Übergangsfunktion $\delta_{[]}([q], a) := [\delta(q, a)]$

  Mit A hat auch $A_{[]}$ keine nicht-erreichbare Zustände
- Betrachte $f : Q \mapsto Q_{[]}$ mit $f(q) := [q]$. Aus den aufgeführten Eigenschaften folgt:

  Satz: f ist Automatenmorphismus; und damit gilt $L(A) = L(A_{[]})$.
- **Satz** - $A_{[]}$ isomorph zum Minimalautomaten von A, also zu $A_{min}(L(A))$
- **Schritte zur Minimierung**:
  Gegeben sei DEA $A = (Q, \Sigma, \delta, q_0, F)$

1. Bestimme die Menge der von q erreichbaren Zustände E!

   Bezeichne E_i die Menge der in $\leq i$ Schritten erreichbaren Zustände

   - Setze $E_0 := \{q_0\}$
   - Wiederhole $E_{i+1} := E_i \cup \{\delta(q,a) | q \in E_i, a \in \Sigma\}$ bis erstmals $E_i = E_i$ gilt
   - Dann ist $E = E_i$
   - Entferne die Zustände $Q \setminus E$ aus dem Automaten
   - Genauer/Alternative: Definiere zu DEA $A = (Q, \Sigma, \delta, q_0, F)$ die **1-Schritt-Zustandserreichbarkeitsrelation** $R_A = \{(p,q) \in Q \times Q | \exists a \in \Sigma : \delta(p,a) = q\}$

     Ist R_A^* die reflexiv-transitive Hülle von R_A, so ist $\{q \in Q | (q_0, q) \in R_A^*\}$ die Menge der erreichbaren Zustände

2. Bestimme die Äquivalenzrelation $\equiv_A$ im nach (1) verkleinerten Automaten wie folgt mit folgenden **Markierungsalgorithmus**:

   - Verwende eine Tabelle aller ungeordneten Zustandspaare $\{q, q'\}$ mit $q \neq q'$
   - Markiere alle Paare $\{q, q'\}$ als nicht-äquivalent, bei denen $|\{q, q'\} \cap F| = 1$
   - Wiederhole, solange noch Änderungen in der Tabelle entstehen:

     Für jedes nicht markierte Paar $\{q, q'\}$ und jedes$a \in \Sigma$ Teste, ob $\{\delta(q,a), \delta(q',a)\}$ bereits markiert ist. Wenn ja $\rightarrow$ markiere $\{q, q'\}$.
   - Alle am Ende nicht-markierten Paare sind äquivalent!

   Gesamtaufwand (mit geeigneten Datenstrukturen und $k = |\Sigma|$ und $n = |Q|$): $O(k \cdot n^2)$

3. Beispiel VL6F11-12

- Weitere Fragen an vorgegebenen DEA A:

  - **Leerheitsproblem** Ist $L(A) = \emptyset$? - Die vom Automaten beschriebene Sprache ist leer gdw. E keine Endzustände enthält.
    - Alternativ: Betrachte zu DEA $A = (Q, \Sigma, \delta, q_0, F)$ die erweiterte **1-Schritt-Zustandserreichbarkeitsrelation**

      $$R_{A,ext} = (p,q) \in Q \times Q | \exists a \in \Sigma : \delta(p,a) = q \cup F \times q_f,$$

    - wobei $q_f \notin Q$ und mit $Q' = Q \cup \{q_f\}$ gilt $R_{A,ext} \subset Q' \times Q'$.
    - $L(A) = \emptyset$ gdw. $(q_0, q_f) \in R_{A,ext}^*$.
    - Die Existenz einer Punkt-zu-Punkt-Verbindung kann sogar in Linearzeit $O(|Q|)$ berechnet werden. (z.B.: Dijkstras Algorithmus)
  - **Äquivalenzproblem** Ist $L(A) = L(A')$? und **Teilmengenproblem** Ist $L(A) \subseteq L(A')$?
    - Beobachte: $L(A) \subseteq L(A')$ gdw. $L(A) \setminus L(A') = \emptyset$
    - Daher:

1. Aus gegebenen DEAs A und A' berechne DEA A'' mit $L(A'') = L(A) \setminus L(A')$.

   Dies geht direkt mit Produktautomatenkonstruktion, wie auf Monoidebene erläutert

2. Entscheide on $L(A'') = \emptyset$ mit vorher skizziertem Verfahren

* Wegen $L(A) = L(A')$ gdw. $L(A) \subseteq L(A')$ und $L(A') \subseteq L(A)$ folgt damit die Entscheidbarkeit des Äquivalenzproblems

– **Endlichkeitsproblem** Ist $L(A)$ endlich?

* Wie im Beweis zum Pumping-Lemma sieht man:
* Ist $L(A)$ unendlich, so gibt es einen Zustand q, einen (evtl. leeren) Weg vom Anfangszustand $q0$ nach q, einen nicht-leeren Weg von q nach q und einen (evtl. leeren) Weg von q zu einem Endzustand.
* Die Umkehrung gilt sogar trivialerweise!
* Bezeichnet R_A die 1-Schritt-Zustandserreichbarkeitsrelation, so berechne E': die Menge der Zustände, die sowohl erreichbar als auch co-erreichbar sind
* (d.h., für alle $q \in E'$ gilt: $(q_0, q) \in R_A^*$ und $\exists q_f \in F : (q, q_f) \in R_A^*$). Dann gilt: $L(A)$ ist unendlich gdw. $\exists q \in E' : (q, q) \in R_A^+$.

- **Endlicher Automat zu Mustersuche**

  – **Beispiel**: Finde Vorkommen des Musters (Pattern) $p = ababac$ in einem Text $t \in \{a, b, c\}^*$.

  – Wir haben schon früher gesehen:

    NEAs sind nützlich für diese Aufgabe.

  – In RA-artiger Notation beobachten wir:

    Lemma: $t \in \Sigma^*$ enthält das Muster p gdw. $t \in \Sigma^*\{p\}\Sigma^*$.

- **Einige Hilfsbegriffe**

  – u heißt **Teilwort** von $x \in \Sigma^*$ gdw. $x \in \Sigma^*\{u\}\Sigma^*$.

    Mustersuche ist also die Suche nach Teilwörtern.

  – u heißt **Präfix** oder **Anfangswort** von $x \in \Sigma^*$ gdw. $x \in \{u\}\Sigma^*$.

  – u heißt **Suffix** oder **Endwort** von $x \in \Sigma^*$ gdw. $x \in \Sigma^*\{u\}$.

  – Ein Teilwort / Präfix / Suffix u von x heißt **echt** gdw. $\ell(u) < \ell(x)$.

  – Ein echtes Teilwort u von x, das sowohl Präfix als auch Suffix von x ist, heißt **Rand (der Breite $\ell(u)$)** von x.

- Am besten Beispiel auf Vorlesung 7, Folien 20-28 anschauen. Hier abzutippen ist nicht sinnvoll.

- **Lemma** - Seien r, s Ränder eines Wortes x mit $l(r) < l(s)$. Dann ist r ein Rand von s.

4 Kontextfreie Grammatiken und kontextfreie Sprachen

4.1 Kontextfreie Grammatiken und Baumautomaten

- Eine **kontextfreie Grammatik** ist ein Quadrupel $G = (\Sigma, N, R, S)$ mit:
  - Σ ist das **Terminalalphabet**
  - N ist das **Nonterminalalphabet** (die **Variablenmenge**, mit $N \cap \Sigma = \emptyset$)
  - $R \subset N \times (\Sigma \cup N)^*$ ist das Alphabet der **Regeln** oder **Produktionen**; übliche Schreibweise:$A \to v$ anstelle von $(A, v) \in R$, wobei $A \in N$ und $v \in (\Sigma \cup N)^*$ auch **linke Seite** bzw. **rechte Seite** der Regel heißen
  - $S \in N$ ist das **Startsymbole** oder **Anfangszeichen**

  Ein Wort über dem **Gesamtalphabet** $(\Sigma \cup N)$ heißt auch **Satzform**

- Der **Ableitungsmechanismus** einer Kontextfreien Grammatik:

  1-Schritt-Ableitungsrelation $\Rightarrow_G$ zwischen zwei Satzformen u, v einer kfG $G : u \Rightarrow_G v$ (manchmal kurz $u \Rightarrow v$) gdw. es gibt Regel $A \to y$ sodass u und v wie folgt zerlegt werden können: $u = xAz$ und $v = xyz$; hierbei sind x und z wiederum Satzformen

  Etwas formaler, ggb. $G = (\Sigma, N, R, S)$:

$$\forall u, v \in (\Sigma \cup N)^* : u \Rightarrow_G v \Leftrightarrow$$
$$(\exists x, z \in (\Sigma \cup N)^* \exists (A \to y) \in R : u = xAz \wedge v = xyz)$$

- $\Rightarrow^n$: **n-Schritt-Ableitungsrelation**
- $\overset{*}{\Rightarrow}$: **Ableitung mit beliebig vielen Schritten**
- Die von einer kfG G **erzeugte** oder **abgeleitete** Sprache ist gegeben durch:

$$L(G) := \{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$$

- Bequeme Schreibweise einer **Ableitung(sfolge)**:

$$u_0 \Rightarrow u_1 \Rightarrow u_2 \Rightarrow u_3$$

- Gilt $u \overset{*}{\Rightarrow} v$, so gilt für ein $k : u \Rightarrow^k v$, bezeugt durch die Ableitungsfolge $U = U_0 \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_{k-1} \Rightarrow u_k = v$. Dieses k heißt auch **Länge der Ableitung**
- **KF**: Familie der **kontextfreien Sprachen**
- $G = (\{a, b\}, \{S\}, R, S)$ mit den Regeln $r_1 = S \mapsto aSb$ und $r_2 = S \mapsto \lambda$

  Lemma: $L(G) = \{a^n b^n | n \geq 0\}$
- Eine kfG heißt **rechtslinear** gdw. alle rechten Regelseiten haben die Form zA oder z, wobei z ein Terminalwort ist und A ein Nichtterminalzeichen.

- **Satz**: L ist regulär gdw. L ist durch rechtslineare kfG erzeugbar
- **Folgerung**: **REG⊂KF**
- **Beschreibung arithmetischer Ausdrücke**

  $\Sigma = \{v, -, +, *, /, (,)\}, N = \{E\}$

Die Regeln seien die folgenden:	Beispielableitung:
$E \to (E)$	
$E \to -(E)$	$E \Rightarrow -(E)$
$E \to (E + E)$	$\Rightarrow -((E + E))$
$E \to (E - E)$	$\Rightarrow -(((E * E) + E))$
$E \to (E * E)$	$\Rightarrow -(((-(E) * E) + E))$
$E \to (E/E)$	$\Rightarrow -(((-(E) * E) + (E - E)))$
$E \to v$	$\Rightarrow -(((-(v) * v) + (v - v)))$

- Hinweis: Der **Scanner**, ein endlicher Automat, produziert idealerweise als Ausgabe die Eingabe für den **Parser** zwecks **syntaktischer Analyse** eines Programmtextes.

  "Nebensächlichkeiten" wie Zahlen oder Zahlenvariablen werden in einen Statthalter v übersetzt
- **Beispiele Ableitungen**: Vorlesung 7, Folie 12
- Eine **Syntaxbaum**(oder **Ableitungsbaum**) für (Formel) → VL7 F13

  Linksableitung: Tiefensuche mit Linksabstieg durch Syntaxbaum

  Rechtsableitung: Tiefensuche mit Rechtsabstieg durch Syntaxbaum
- Unter dem **Typ einer algebraischen Struktur** versteht mein ein Paar (F, σ). Hierbei ist:

  F die Menge der **Funktionensymbole**

  $\sigma : F \to \mathcal{N}$ liefert die **Stelligkeit** zu dem betreffenden Symbol

  Der Typ einer Struktur ist ein rein syntaktisches Objekt. (**syntaktische Ebene**)

  Wenn es auf die Namen der Symbole nicht ankommt, erwähnt man oft auch nur den Stelligkeitstyp.

  Informatisch gesprochen beschreibt ein Typ das Interface zwischen Strukturen.
- **Strukturen und Algebren**
  - Es sei $A \neq \emptyset$ eine Menge und $n \in N$.
  - Eine Abbildung $A^n \to A$ heißt **n-stellige Operation auf A**.
    (Hier ist $A^n = A \times ... \times A$ das (n-1)-fache kartesische Produkt.)
  - Nullstellige Operationen heißen auch **Konstanten**.
  - $Op_n(A)$ sei die Menge der n-stelligen Operationen auf A, d.h., $Op_n(A) = A^{A^n}$.
  - $Op(A) = \bigcup_{n=0}^{\infty} Op_n(A)$.

– Eine algebraische Struktur vom Typ (F, σ) ist ein Paar $\mathbb{A} = (A, F), A \neq \emptyset$, mit $F = \{f_{\mathbb{A}} | f \in F\}$, wobei jedem $f \in F$ genau eine Operation $f_{\mathbb{A}} \in Op_{\sigma(f)}(A)$ zugeordnet ist.

Algebraische Strukturen liefern die **semantische Ebene**.

- Es sei $\mathbb{A} = (A, F)$ eine Algebra vom Typ (F, σ).

  Dann sind **Terme über** $\mathbb{A}$ induktiv wie folgt definiert:

  1. Jedes $a \in A$ ist ein Term
  2. Sind $t_1, \ldots, t_n$ Terme über $\mathbb{A}$ und ist $f \in F$ mit $\sigma(f) = n$, so ist $f(t_1, \ldots, t_n)$ ein Term über $\mathbb{A}$
  3. Nichts anderes sind Terme über $\mathbb{A}$

  Wir sammeln aller Terme über $\mathbb{A}$ in der Menge $Term(\mathbb{A})$

- **Zur Auswertung von Bäumen**

  – **Definition** - Es sei $\mathbb{A} = (A, F)$ eine Algebra vom Typ (F, σ). Wir definieren die **Auswertefunktion** $eval : Term(\mathbb{A}) \to A$ induktiv wie folgt:

  1. Für $a \in A$ sei $eval(a) := a$
  2. Sind $t_1, \ldots, t_n$ Terme über $\mathbb{A}$ und ist $f \in F$ mit $\sigma(f) = n$, so ist
  $$eval(f(t_1, \ldots, t_n)) := f_{\mathbb{A}}(eval(t_1), \ldots, eval(t_n))$$

  – Unser Beispiel

$$\begin{aligned}
eval(max(min(0,6), max(2, min(3,4)) &= \underset{\mathbb{N}}{max}(eval(min(0,6)), eval(max(2, min(3,4)))) \\
&= \underset{\mathbb{N}}{max}(\underset{\mathbb{N}}{min}(eval(0), eval(6)), \underset{\mathbb{N}}{max}(eval(2), eval(min(3,4)))) \\
&= \underset{\mathbb{N}}{max}(\underset{\mathbb{N}}{min}(0,6), \underset{\mathbb{N}}{max}(2, \underset{\mathbb{N}}{min}(eval(3), eval(4)))) \\
&= \underset{\mathbb{N}}{max}(0, \underset{\mathbb{N}}{max}(2, \underset{\mathbb{N}}{min}(3,4))) \\
&= \underset{\mathbb{N}}{max}(0, \underset{\mathbb{N}}{max}(2,3)) \\
&= \underset{\mathbb{N}}{max}(0,3) \\
&= 3
\end{aligned}$$

- Es sei (F, σ) der Typ einer algebraischen Struktur.

  Einnerung: nullstellige Operatoren sind Konstanten und damit die Beschriftung von Blättern der Termbäume.

  Wir interpretieren diese im Folgenden als Teil des Zustandsalphabets.

  Es sei Q ein endliches Zustandsalphabet mit Endzustandsmenge $Q_f \subseteq Q$.

  Zustandsübergangsmenge Δ mit Elementen der Bauart: $(f, q_1, ..., q_n, q)$ mit $f \in F; \sigma(f) = n; q_1, ..., q_n, q \in Q$.

  Ein **endlicher Baumautomat** kann spezifiziert werden durch das Quadrupel

$$A = (Q, (F, \sigma), Q_f, \Delta)$$

- Q ist ein endliches Zustandsalphabet
- (F, σ) ist der Typ einer algebraischen Struktur
- $Q_f \subseteq Q$ die Endzustandsmenge
- Δ die Zustandsübergangsmenge

- Ein **Baum** über (F, σ) (als rein syntaktisches Objekt) ist gegeben durch:
  1. Jedes $a \in \{f \in F | \sigma(f) = 0\}$ ist ein Baum
  2. Sind $t_1, \ldots, t_n$ Bäume über (F, σ) und ist $f \in F$ mit $\sigma(f) = n$, so ist $f(t_1, \ldots, t_n)$ ein Baum über (F, σ)
  3. Nichts anderes sind Terme über (F, σ)

- **Arbeitsweise endlicher Automaten**

  Interpretiere $A = (Q, (F, \sigma), Q_f, \Delta)$ als Algebra $\mathbb{A}_A = (2^Q, F)$ durch:

  $$f(Q_1, ..., Q_n) := \{q \in Q | \exists q_1 \in Q_1, ..., \exists q_n \in Q_n : (f, q_1, ..., q_n, q) \in \Delta\}$$

  für $f \in F$ mit $\sigma(f) = n$.

  Beachte, dass wir nullstellige Operatoren (Blattbeschriftungen) als Zustände und weiter als einelementige Zustandsmengen interpretieren.

  Daher sind Bäume über (F, σ) auch Terme über $\mathbb{A}$.

  A akzeptiert die Baumsprache $B(A) := \{b \in \mathbb{B}(F, \sigma) | eval_{\mathbb{A}_A}(b) \cap Q_f \neq \emptyset\}$.

  Eine Baumsprache B (Menge von Bäumen) ist **regulär** gdw. es gibt einen endlichen Baumautomaten, der B akzeptiert.

- **Wort-Algebra: Eine weitere Algebra für** (F, σ)

  Nullstellige Operationssymbole werden als Buchstaben interpretiert

  $\rightsquigarrow$ Alphabet $\Sigma \rightsquigarrow$ Grundmenge Σ^+ für die Algebra $\mathbb{A}_\Sigma$

  Jedes einstellige Operationssymbol wird als Identität gedeutet.

  Jedes mehrstellige Operationssymbol wird als Konkatenation der Argumentwörter gelesen.

  Die Terme dieser Algebra sind gerade die Bäume von (F, σ).

  Für das Ergebnis der Auswertung eines Baumes b in dieser Algebra schreiben wir auch:

  $$yield : \mathbb{B}(F, \sigma) \to \Sigma^+, yield(b) := eval_{A_\Sigma}(b)$$

- **Der Satz von Doner, Thatcher und Wright**

  Satz: Es sei $L \subseteq \Sigma^*$ eine Sprache.

  L ist kontextfrei gdw. $L = yield(B(A))$ für einen endlichen Baumautomaten A.

Idee 1 : Eine Regel $(f, q_1, ..., q_n, q)$ des Baumautomaten wird zur Grammatikregel $(f, q) \to (f_1, q_1)...(f_n, q_n)$ für geeignet geratene Symbole $f_1, ..., f_n$.

Idee 2 : Eine kontextfreie Regel $A \to B_1...B_n$ wird zur Regel $(X_n, B_1, ..., B_n, A)$ des Baumautomaten; hierbei gibt es genau ein Symbol X_n der Stelligkeit n.

- **Weitere Kommentare**
  - Die Idee 2 führt unmittelbar zu einer Formalisierung des Ableitungsbaumbegriffs; hierbei werden lediglich die "Dummy-Symbole" X_n nicht hingeschrieben, dafür allerdings die Nichtterminale (also die Zustände des Baumautomaten).
  - Die Hintereinanderschaltung beider Ideen führt dazu, dass es zu jeder kontextfreien Grammatik eine äquivalente G gibt, bei der wir jedem Nichtterminalzeichen A eine Stelligkeit $\sigma(A)$ zuordnen können, sodass falls $A \to w$ eine Regel von G ist, so gilt $\sigma(A) = \ell(w)$.
  - In der Literatur wird zumeist zwischen "Blattbeschriftungen" und Zuständen getrennt. Dann braucht man jedoch weitere Regeln der Form $(a, q) \in \Delta$ für $a \in \{f \in F | \sigma(f) = 0\}$ und $q \in Q$
- **Lemma** - Zu der kfG G gibt es eine kfG G' mit $L(G) = L(G')$, bei der Regeln, die Terminalzeichen enthalten, alleinig von der Form $A \to a$ sind mit $a \in \Sigma$
- λ**-Regeln** sind Regeln der Form $A \to \lambda$
- **Lemma** - Zu jeder kfG G gibt es eine kfG G' ohne λ-Regeln mit $L(G) \setminus \{\lambda\} = L(G')$
- **Wie erhält man G' aus G algorithmisch?**

  Problem: Berechne $N^\lambda = \{A \in N | A \overset{*}{\Rightarrow}_G \lambda\}$

  Induktive/rekursive Konstruktion:

  1. $N_0^\lambda := \{A \in N | A \to_G \lambda\}$
  2. $N_k^\lambda := \{A \in N | A \to_G X \wedge X \in (N_{k-1}^\lambda *)$ für $k > 0\}$

  Beachte Monotonieeigenschaft: $N_0^\lambda \subseteq N_1^\lambda \subseteq N_2^\lambda ...$
- **Lemma**: $A \in N^\lambda$ gdw. $A \in \bigcup_{k=0}^{\infty} N_k^\lambda =: N_*^\lambda$ gdw. $A \in N_{|N|}^\lambda$
- **Kettenregeln** sind Regeln der Form $A \to B$, $B \in N$
- **Lemma** - Zu jeder kfG G gibt es eine kfG G' ohne Kettenregeln mit $L(G) = L(G')$.

4.2 Normalformen

- Eine **kontextfreie Grammatik in Chomsky-Normalform** ist ein Quadrupel $G = (\Sigma, N, R, S)$ mit:
  - Σ ist das **Terminalalphabet**
  - N ist das **Nonterminalalphabet** (die **Variablenmenge**)

- $R \subset (N \times N^2) \cup (N \times \Sigma)$ ist das Alphabet der **Regeln** oder **Produktionen**
- $S \in N$ ist das **Startsymbole** oder **Anfangszeichen**

- **Lemma** - Zu jeder kfG G gibt es eine kfG G' in Chomsky-Normalform mit $L(G) \setminus \{\lambda\} = L(G')$

- **Algorithmus zur Vermeidung zu langer rechter Regelseiten**

  Grundidee: Einfügen von "Zwischenzuständen" beim "Buchstabieren"

  Eingabe: Regelmenge R (die im Folgenden modifiziert wird)

  WHILE $(\exists A \to w \in R : \ell(w) > 2)$ DO:

  1. Stelle $w = Bu$ dar.
  2. Erzeuge neues Nichtterminalzeichen X.
  3. Ersetze $A \to w$ durch $A \to BX$ und $X \to u$ in R.

  Korrektheit ist klar.

- **Satz**: Es gibt einen Algorithmus, der zu jeder vorgelegten kfG $G = (\Sigma, N, R, S)$ und jedem $w \in \Sigma^*$ entscheidet, ob $w \in L(G)$ gilt.

  Cocke, Younger und Kasami haben gezeigt, wie man die Komplexität des beschriebenen Verfahrens durch **dynamisches Programmieren** erheblich vermindern kann.

- **Satz**: Ist eine kfG G in Chomsky-Normalform fixiert, so lässt sich die Frage "$w \in L(G)$? in einer Zeit beantworten, die sich durch ein kubisches Polynom in $\ell(w)$ abschätzen lässt:

- Das Verfahren von Cocke, Younger und Kasami(**CYK-Algorithmus**)

$$T[i,i] = \{A \in N | A \to a_i \in R\}$$
$$T[i,k] = \{A \in N | A \to BC \in R \wedge \exists i \leq j < k : B \in T[i,j], C \in T[j+1,k]\}$$

- **Satz** - **KF** ist unter Vereinigung abgeschlossen

- **Satz** - **KF** ist unter Konkatenation abgeschlossen

- **Satz** - **KF** ist unter Kleene Stern abgeschlossen

- **Satz** - **KF** ist unter Durchschnitt mit regulären Sprachen abgeschlossen

- Ein **binärer Wurzelbaum** ist gegeben durch ein Tripel $B = (V, \phi, r)$ mit ausgezeichneter **Wurzel** $r \in V$ und einer **Vater-Abbildung** $\phi : V \setminus \{r\} \to V$ mit der Eigenschaft

$$\forall v \in V : \#\underbrace{\{u \in V | \phi(u) = v\}}_{k(v):=} \leq 2$$

$k(v)$ liefert also die **Kinder** von v.

Knoten v mit $k(v) = \emptyset$ heißen **Blätter**.

- **Lemma** - Der Ableitungsbaum eines jeden von einer kontextfreien Grammatik in Chomsky-Normalform akzeptierten Wortes kann als binärer Wurzelbaum aufgefasst werden.

- Die Höhe eines binären Wurzelbaumes $B = (V, \phi, r)$ ist gegeben durch

$$h(B) = \max_{v \in V \setminus \{r\}} \{k \in N | \phi^k(v) = r\}$$

  Sonderfall $h(B) = 0$ bedeutet: $B = (\{r\}, \phi, r)$ mit trivialem *phi*.

- **Lemma** - Hat ein binärer Wurzelbaum B mehr als 2^h Blätter, so gilt $h(B) > h$.

- **Lemma** - Ist $G = (\Sigma, N, R, S)$ eine kfG in Chomsky-Normalform und ist $w \in L(G)$ mit $\ell(w) > 2^{\#N}$, so gilt für jeden Ableitungsbaum von w bzgl. G, dass es einen Weg von S zu einem Blatt gibt, auf dem mehr als $\#N$ viele Nichtterminalzeichen ersetzt werden

- **Folgerung** - Auf besagtem Weg von der Wurzel zum Blatt im Ableitungsbaum von w finden wir also nach dem Schubfachprinzip zwei Regelanwendungen $A \to v$ und $A \to u$ mit gleicher linker Seite.

- **Satz** - Jede $L \in KF$ lässt sich beschreiben durch eine kfG $G = (\Sigma, N, R, S)$ mit Regeln der Form $N \times ((NN) \cup (\Sigma))$; zusätzlich darf eine Regel $S \to \lambda$ existieren, wobei dann gefordert ist, dass S in keiner rechten Regelseite vorkommt.

- **Ein Pumping-Lemma für KF**

  Satz - Zu jeder kfS L gibt es eine Konstante $n > 0$, sodass jedes Wort $w \in L$ mit $\ell(w) \geq n$ als Konkatenation $w = uvxyz$ dargestellt werden kann mit geeigneten u, v, x, y, z mit folgenden Eigenschaften:

  1. $\ell(v) > 0$ oder $\ell(y) > 0$;
  2. $\ell(vxy) \leq n$;
  3. $\forall i \geq 0 : uv^i xy^i z \in L$

- **Das Pumping-Lemma kennzeichnet nicht Kontextfreiheit**

  Betrachte

$$L = a^k d^r a^k d^s a^k | k, r, s \in \mathbb{N}$$

  Mit der gleichen Intuition wie vorher ist die Sprache nicht kontextfrei.

  L erfüllt aber das Pumping-Lemma:

  Ein Wort der Form $w = a^k d^r a^k d^s a^k$ mit $r > 0$ lässt sich zerlegen in: $w = uvxyz$ mit (z.B.) $u = a^k$, $v = d$ und $y = \lambda$.

Dann gilt $uv^ixy^iz \in L$ für alle $i \in \mathbb{N}$.

Der Fall $r = 0$ aber $s > 0$ geht analog.

Gilt $r = s = 0$, so gilt für $w = a^{3k}$, und die Wahl $v = a^3$, $y = \lambda$ führt wiederum auf $uv^ixy^iz \in L$ für alle $i \in \mathbb{N}$

- **Pumping-Lemma Beispiele VL9 F 25-29**
- **Satz - KF** ist nicht unter Durchschnitt abgeschlossen
- **Satz - KF** ist nicht unter Komplementbildung abgeschlossen
- **Satz** - Das Leerheitsproblem ist für kfG entscheidbar
- **Endlichkeitsproblem**: Gegeben Sprachbeschreibung G für L, ist L endlich?
- **Satz** - Das Endlichkeitsproblem ist für kfG entscheidbar
- **Lemma** - Zu jeder Sprache $L \in KF$ gibt es eine Konstante $n > 0$, sodass gilt: L ist unendlich gdw. es gibt ein Wort $t \in L$ mit $n \leq \ell(t) < 2n$

4.3 Automaten mit unendlichem Speicher

4.4 Nichtkontextfreie Sprachen

4.5 Algorithmen für kontextfreie Grammatiken

5 Chomsky-Hierarchie

- Eine **(Phasenstruktur-)Grammatik** ist ein Quadrupel $G = (\Sigma, N, R, S)$ mit:
  - Σ ist das **Terminalalphabet**
  - N ist das **Nonterminalalphabet** (die **Variablenmenge**
  - $R \subset (\Sigma \cup N)^*N(\Sigma \cup N)^* \times (\Sigma \cup N)^*$ ist das Alphabet der **Regeln** oder **Produktionen**; übliche Schreibweise:$xAy \to v$ anstelle von $(xAy, v) \in R$, wobei $A \in N$ und $x, y, v \in (\Sigma \cup N)^*$ auch **linke Seite** bzw. **rechte Seite** der Regel heißen
  - $S \in N$ ist das **Startsymbole** oder **Anfangszeichen**

  Ein Wort über dem **Gesamtalphabet** $(\Sigma \cup N)$ heißt auch **Satzform**

5.0.1 MON und KS

- Eine Grammatik heißt **monoton** oder **nichtverkürzend** gdw. für alle Regeln gilt, dass die rechte Regelseite nicht kürzer als die linke ist.
- Eine monotone Grammatik heißt **kontextsensitiv**, wenn alle Regeln von der Form $\zeta A\eta \to \zeta w\eta$ sind für irgendein $A \in N$.

- Hinweis: Bei kontextfreien Regeln gilt $\zeta = \eta = \lambda$.
- Sonderfall λ-Regeln: Um die Ableitung des leeren Wortes zu ermöglichen, kann eine nichtmonotone Regel $S \to \lambda$ für das Startzeichen S bei **MON** und bei **KS** zugelassen werden. Dann darf aber S auf keiner rechten Regelseite vorkommen.

5.0.2 Wie beweisen wir Sprachgleichheit?

- Standard, getrennte Induktionen für $L(G) \subseteq L$ und $L \subseteq L(G)$.
- Alternativ / Hilfsmittel: Kennzeichnung der ableitbaren Satzformen.

  Lemma: Für $L_k := \{w | a^k Y_a b^k c^k \Rightarrow^{k+2} w\}$ gilt: $L_k = \{a^{k+1}b^{k+1}Y_a c^{k+1}, a^{k+1}b^{k+1}c^{k+1}\}$ für jedes $k \geq 1$.
- Einfacher sogar kann man per Induktion nachweisen:

  Lemma - Aus $a^k b^k Y_a c^k$ ergibt sich nach k Ableitungsschritten $a^k Y_a b^k c^k$(und keine andere Satzform).
- Ein nochmaliges Nachvollziehen der Beweise beider Lemmata liefert:

  Lemma - Die einzigen in den beiden Lemmata beobachteten Satzformen, die nur aus Terminalzeichen bestehen, sind explizit im ersten Lemma angegeben.

5.0.3 Eine Normalform für monotone Grammatiken

(ähnlich bei Typ-0)

- **Satz** - Zu jeder monotonen Grammatik gibt es eine äquivalente, bei der alle Re geln mit Terminalzeichen a von der Form $A \to a$ sind.
- **Satz KS=MON**
- **SatzKF $\subsetneq$ KS**

5.0.4 Abschlusseigenschaften bei Typ-0/-1 Sprachen

- **Vereinigung / Konkatenation** - "Standardkonstruktion" wie Typ-2

  Achtung: Disjunkte Pseudoterminalalphabete bei Konkatenation
- **Durchschnitt** - NF-Grammatiken G_1 und G_2 werden simuliert auf "Produktalphabet" $N_1 \times N_2$. Nur für Paare (X_a, X_a) gibt es terminierende Regeln $(X_a, X_a) \to a$.
- **Komplement** - Typ-1 Sprachen sind abgeschlossen; die entsprechende Technik des induktiven Zählens ist Gegenstand der Komplexitätstheorie.

  Typ-0 Sprachen sind nicht abgeschlossen (Satz von Post in der nächsten Theorie-Vorlesung)

5.0.5 Die Chomsky-Hierarchy in Grammatik-Form

Typ-0 : Phrasenstrukturgrammatiken, also RA. (RE: recursively enumerable)

Typ-1 : kontextsensitive Grammatiken, also KS. (CS: context-sensitive)

Typ-2 : kontextfreie Grammatiken, also KF. (CF: context-free)

Typ-3 : rechtslineare Grammatiken, also REG.

- **Satz** - REG $\subsetneq$ KF $\subsetneq$ KS $\subsetneq$ RA

6 Turingmaschinen

- Eine **Turingmaschine (TM)** ist durch ein 7-Tupel beschrieben:

$$TM = (S, E, A, \delta, s_0, \square, F)$$

  Dabei bedeuten

  - $S = \{s_0, s_1, ..., s_n\}$ die Menge der **Zustände**,
  - $E = \{e_0, e_1, ..., e_r\}$ das endliche **Eingabealphabet**,
  - $A = \{a_0, a_1, ..., a_m\}$ das endliche **Arbeitsalphabet** (auch Bandalphabet genannt), es sei dabei $E \subset A$,
  - s_0 der Startzustand,
  - $a_0 = \square$ das **Blank-Symbol**, das zwar dem Arbeitsalphabet, aber nicht dem Eingabealphabet angehört,
  - $F \subseteq S$ die Menge der **Endzustände**,
  - δ sei die Überführungsfunktion/-relation mit (im deterministischen Fall)

$$\delta : (S \setminus F) \times A \to S \times A \times \{L, R, N\}$$

    bzw. (im nichtdeterministischen Fall) $\delta \subseteq ((S \setminus F) \times A) \times (S \times A \times \{L, R, N\})$

  Hier bedeutet: L= links, R = rechts, N= neutral

- $\delta(s, a) = (s', b, x)$ bzw. $((s, a), (s', b, x)) \in \delta$ (mit $x \in \{L, R, N\}$) haben folgende Bedeutung:

  Wenn sich der Automat im Zustand s befindet und unter dem Kopf das Zeichen a steht, so schreibt der Automat b und geht ein Feld nach rechts (R), bzw. links (L), bzw. bewegt sich nicht (N) und geht in den Zustand s' über.

- Eine **Konfiguration** einer Turingmaschine $TM = (S, E, A, \delta, s_0, \square, F)$ ist ein Tripel (u, s, v) aus $A^* \times S \times A^+$:

  - uv ist aktuelle Bandinschrift
  - s ist der aktuelle Zustand.
  - Schreib-Lesekopf über erstem Zeichen von v, daher $v \neq \varepsilon$.

- Start der Maschine: $v \in E^{*\cup\{\square\}}$ (Eingabe), $s = s_0, u = \varepsilon$.
- O.B.d.A. $S \cap E = \emptyset$, daher Schreibweise usv statt (u, s, v)

- **Anfangskonfiguration** beim Start der Turingmaschine mit Eingabe $w \in E^*$ ist $s_0 w$ (bzw. $s_0\square$, falls $w = \varepsilon$).

- **Endkonfigurationen** sind alle Konfigurationen $us_f v$ mit $s_f \in F$. Hier kann die Berechnung nicht mehr fortgesetzt werden.

- Weiter ist

$$L(TM) := \{w \in E^* | s_0 w \vdash^* us_f v, s_f \in F, u, v \in A^*\}$$

  die **von der Turingmaschine akzeptierte Sprache L**

- Simulation endlicher Automat durch Turingmaschine z.B. wie folgt:
  - Schreib-Lesekopf hat ausschließlich lesende Funktion
  - Lesekopf zeichenweise lesend nach rechts
  - Zustandsübergänge des endlichen Automaten übernommen
  - Akzeptieren bei Erreichen von $\square$ (d.h. Ende der Eingabe) im 'Automaten-Endzustand'

- **Typ-0 Sprachen lassen sich durch TM-Akzeptanz kennzeichnen**

  Eine Phrasenstrukturgrammatik, die eine TM simulieren soll, arbeitet wie folgt:

  1. Es wird die Sprache "$s0w\$w$" generiert für beliebige Eingabewörter w der Turingmaschine.
  2. Die Arbeitsweise der Turingmaschine (Konfigurationsübergänge) wird nun auf dem ersten Wortteil simuliert.
  3. Wird eine Finalkonfiguration erreicht, so besteht die Möglichkeit, den ersten Wortteil und den Trenner \$ zu löschen und so w zu generieren

- Eine TM heißt **linear beschränkter Automat (LBA)**, wenn sie keine Blankzeichen überschreiben darf.

  Satz - L ist Typ-1 gdw. L wird von LBA akzeptiert.

- Automaten mit unendlicher Speicherstruktur
  - besitzen endliche Kontrolleinheit wie Turingmaschinen;
  - anzugeben: Zugriff auf Speicherstruktur:
    - Lesen (des aktuellen Speicherelements);
    - Schreiben;
    - Bewegung des Schreib-Lese-Kopfes (evtl. implizit);
    - evtl. weitere Abfragen / Tests

- Ein Kellerautomat, engl. Pushdown automaton, ist ein Sextupel $A = (Q, \Sigma, \Gamma, q_0, \Delta, F)$:
  - Q ist das **Zustandsalphabet**,
  - Σ ist das **Eingabealphabet**,
  - Γ ist das **Kelleralphabet**,
  - $q_0 \in Q$ ist der Startzustand (Anfangszustand),
  - $\Delta \subset (Q \times (\Sigma \cup \{\lambda\}) \times \Gamma^*) \times (Q \times \Gamma^*)$ ist die (endliche) **Übergangsrelation**,
  - $F \subseteq Q$ ist die **Endzustandsmenge**.
- Vorlesung 10, Folien 38-45 gute Beispiele

6.1 Algorithmen für kontextfreie Grammatiken

Compiler(auf)bau: Eine Übersicht der Phasen

1. Scanner: lexikalische Analyse (endliche Automaten mit Ausgabe)
2. Parser: syntaktische Analyse (kontextfreie Sprachen)
3. semantische Analyse
4. Codegenerierung
5. Optimierung

6.1.1 Parsing

- Linksableitung: Tiefensuche mit Linksabstieg durch Syntaxbaum
- Rechtsableitung: Tiefensuche mit Rechtsabstieg durch Syntaxbaum
- Ein **Linksparser** für Grammatik $G = (\Sigma, Q, R, s)$ liefert zu einem Wort $w \in L(G)$ eine Linksableitung von w, beschrieben durch ein Wort über R, und NEIN, falls $w \notin L(G)$.
- Analog: **Rechtsparser** liefert Rechtsableitung.

  Hinweis: Es gibt **mehrdeutige** kfG, d.h., kfG, bei denen (manche) Wörter mehr als eine Linksableitung besitzen; dann liefert ein Linksparser irgendeine solche Linksableitung.
- **Prädikative (Links-)Parser**

  Erinnerung: KfG $\rightarrow$ Kellerautomat Konstruktion

  Sei $G = (\Sigma, N, R, S)$ eine kfG. Betrachte folgende Transitionen:
  - $((s, \lambda, \lambda), (f, S))$,
  - für jede Regel $C \rightarrow w : ((f, \lambda, C), (f, w))$,
  - für jedes Terminalzeichen $a : ((f, a, a), (f, \lambda))$.

f ist der einzige Endzustand und s der Startzustand

- Hieraus **Linksparser**

  Der Linksparser simuliert im Wesentlichen den beschriebenen Kellerautomaten (und ist daher im Allgemeinen nichtdeterministisch):

  Simuliert der Kellerautomat die kfG, so gibt der Parser die entsprechende Regel aus (**Produktionsschritt**).

  Schritte der Form $((f, a, a), (f, \lambda))$ heißen **Leseschritte (Shifts)** und führen zu keiner Ausgabe.

  Im Wesentlichen kann der Linksparser die Zustandsinformation des Kellerautomaten ignorieren (Kellerautomaten-Normalformen !)

- **Konflikte bei Linksparsern**

  ein Terminalzeichen auf dem Keller liegt, muss ein Linksparser einen Leseschritt durchführen (stimmt das oberste Kellerzeichen nicht mit dem aktuellen Eingabezeichen überein, so NEIN).

  Bei den Produktionsschritten kann es jedoch zu **Konflikten** kommen: Welche Produktion (Regel) ist anzuwenden (zu simulieren) und damit auszugeben ?

- **Greibach-Normalform**

  Eine kfG $G = (\Sigma, Q, R, s)$ ist in Greibach-Normalform gdw.

  $$R \subseteq (N \times \Sigma(N \setminus \{S\})^*) \cup (S \times \{\lambda\}).$$

  Hinweis: Umformung Kellerautomat $\rightarrow$ kfG lieferte "fast" Greibach-NF.

  Satz: Jede kontextfreie Sprache besitzt eine sie erzeugende kfG in Greibach-NF.(ohne Beweis)

  Eine kfG $G = (\Sigma, N, R, S)$ mit $R \subseteq (N \times \Sigma(N \cup \Sigma)^*)$ heißt **simpel** oder **s-Grammatik** gdw. $\forall A \in N \forall a \in \Sigma : |\{\beta \in (N \cup \Sigma)^* | A \rightarrow a\beta \in R\}| \leq 1$.

  Einfach anzugeben: eine äquivalente kfG in Greibach-NF zu einer s-Grammatik. Überlegen Sie sich die Details dazu! Hinweis: Unser Weg zur Chomsky-NF.

  Lemma: Linksparser für s-Grammatiken arbeiten deterministisch.

- Linksparser heißen auch **Top-Down-Parser**

  Der Ableitungsbaum wird von oben nach unten durchforstet.

  Problem: Regelanwendungskonflikte.

  Mögliche Lösung: Verzeichne zu jeder Regel $A \rightarrow w$ die Menge der Eingabezeichen aus Σ (oder allgemein Wörter über Σ einer vorgegebenen Maximallänge k), die als Präfix der Länge k für ein aus w ableitbares Terminalwort vorkommen können, als **Vorschau (Lookahead)**.

  Tatsächlich wird die Bestimmung einer Vorschau-Menge noch verkompliziert durch mögliche λ-Regeln; das betrachten wir hier nicht im Einzelnen.

  Auf dieser Idee fußend werden **LL(k)-Grammatiken** definiert

- **Parsen mit rekursivem Abstieg (für LL(1))**
  - Für jedes oberste Kellerzeichen X gibt es Extra-Prozedur $P_X(au, K)$, wobei au die restliche Eingabe ist (mit a als aktuelles Eingabezeichen) und K der Keller (ohne oberstes Kellerzeichen X).
  - $P_b(au, K)$ für Terminalzeichen b:
    1. liefert Fehlermeldung, falls $b \neq a$;
    2. falls $b = a$, wird das oberste Kellerzeichen entfernt; gilt nun
       (a) $a = \$$ und $u = K = \lambda$, so akzeptiere; andernfalls ist
       (b) $K = \lambda$ ein Fehlerfall, da kein neues oberstes Kellerzeichen vhd.; für
       (c) $K \neq \lambda$ sei $K = X'K'$ und rekursiv rufe $P_{X'}(u, K')$ auf.
  - In $P_X(au, K)$ wird für ein Nichtterminalzeichen X nach verschiedenen Möglichkeiten für a rekursiv verzweigt.

    (Hier könnte man auch noch tiefer in der Ausgabe vorausschauen, wenn LL(k) gefordert.)

    Das heißt, die betreffende Regel $X \to w$ wird ausgeführt und der neue Keller erfüllt $X'K' = wK$ für ein Zeichen X'; rekursiv wird nun $P_{X'}(au, K')$ aufgerufen.
  - **Beobachte** den Rekursionskeller!
  - **Gefahr** durch Linksrekursion in der Grammatik, d.h.: $X \stackrel{*}{\Rightarrow} X\alpha$

- **Rechtsparser**: bottom-up Simulation von Ableitungsbäumen

  Sei $G = (\Sigma, N, R, S)$ eine kfG.

  Betrachte folgende Transitionen (Kellerautomat hat nur einen Zustand):

  - für jede Regel $C \to w : ((f, \lambda, w), (f, C))$ (Reduktionsschritt),
  - für jedes Terminalzeichen $a : ((f, a, \lambda), (f, a))$ (Leseschritt).

  Der Kellerautomat akzeptiert bei leerer Eingabe und nur S auf dem Keller.

  Es ist hier einfacher, beim Keller das oberste Zeichen "rechts" anzunehmen.

- **Konflikte bei Rechtsparsern**

  Es gibt zwei Arten von **Konflikten**:

  Welche Produktion (Regel) unter mehreren ist anzuwenden (reduzierend zu simulieren) und damit auszugeben ? Reduktionsschritt oder (weiterer) Leseschritt ?

  Idee wiederum: Möglicherweise hilft die Resteingabe weiter. . .

6.2 Schnelle Zusammenfassung Typen

6.2.1 Effektive Charakterisierungen: Typ 3

- Eine Sprache L ist **regulär** ("Monoid-Kennzeichnung") $\Leftrightarrow$
- L wird von einem DEA akzeptiert $\Leftrightarrow$
- L besitzt endlich viele Nerode-Äquivalenzklassen (endlicher Index) $\Leftrightarrow$
- L wird von einem NEA akzeptiert $\Leftrightarrow$
- L wird von einem NEA mit λ-Übergängen akzeptiert $\Leftrightarrow$
- L wird durch einen regulären Ausdruck beschrieben $\Leftrightarrow$
- L wird durch eine rechtslineare Grammatik erzeugt

6.2.2 Effektive Charakterisierungen: Typ 2

- Eine Sprache L ist **kontextfrei** $\Leftrightarrow$
- L wird von einem Kellerautomaten akzeptiert $\Leftrightarrow$
- L wird von einem Kellerautomaten mit Leerkellerakzeptanz erkannt $\Leftrightarrow$
- L wird von einem Kellerautomaten mit Endzustandsakzeptanz erkannt $\Leftrightarrow$
- L wird von einer kontextfreien Grammatik erzeugt $\Leftrightarrow$
- L wird von einer kontextfreien Grammatik mit Linksableitung erzeugt $\Leftrightarrow$
- L wird von einer kontextfreien Grammatik in erweiterter Chomsky-Normalform erzeugt

6.2.3 Effektive Charakterisierungen: Typ 1

- Eine Sprache L ist vom **Typ 1** $\Leftrightarrow$
- L wird von einem linear beschränkten Automaten akzeptiert $\Leftrightarrow$
- L wird von einer monotonen Grammatik erzeugt $\Leftrightarrow$
- L wird von einer kontextsensitiven Grammatik erzeugt

6.2.4 Effektive Charakterisierungen: Typ 0

- Eine Sprache L ist vom **Typ 0** oder rekursiv aufzählbar $\Leftrightarrow$
- L wird von einer Turingmaschine akzeptiert $\Leftrightarrow$
- L wird von einer Phrasenstrukturgrammatik erzeugt

6.2.5 Abschlusseigenschaften

(mit konstruktiven Beweisen, falls "ja" !)

	Typ-0	Typ-1	Typ-2	Typ-3
Durchschnitt	ja	ja	nein	ja
Kompliment	nein	ja	nein	ja
Vereinigung	ja	ja	ja	ja
Konkatenation	ja	ja	ja	ja
Kleene $*$	ja	ja	ja	ja
Morphismen	ja	nein	ja	ja
$\cap$ mit Typ-3	ja	ja	ja	ja

6.2.6 Entscheidbarkeitsfragen

Gibt es einen Algorithmus zum Lösen folgender Fragen: Leerheit, Endlichkeit, (uniformes) Wortproblem

	Typ-0	Typ-1	Typ-2	Typ-3
Leerheit	nein	nein	ja	ja
Endlichkeit	nein	nein	ja	ja
(uniformes) Wortproblem	nein	ja	ja	ja

Jedes "Ja" ist durch einen Algorithmus begründet ! Die "Neins" bei Typ-0 werden erst in der nächsten Grundlagenvorlesung klar ! Warum ist das Leerheitsproblem für Typ-1 nicht "einfacher" als für Typ-0 ? Ersetze löschende Regeln $\zeta A \eta \to \lambda$ durch $\zeta A \eta \to a^{\ell(\zeta A \eta)}$ für irgendein Terminalzeichen a.