GBI Definitionen

Formale Sprachen

Eine formale Sprache L ist eine Menge aus Wörtern.
zB.: $L_{ab} = \{a, b\}$ ist die Sprache mit den Wörtern a und b

- Die Potenz eine Sprache ist definiert als:

$$L^0 = \{\epsilon\}$$
$$L^{n+1} = L \cdot L^n$$

- Sprache konkateniert mit eine Sprache ist einfach jede mögliche Kombination der Wörter. Als Beispiel nehmen wir wieder $L_{ab} = \{a, b\}$

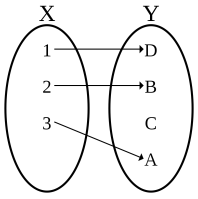
$$L_{ab} \cdot \{aa, bb\} = \{aaa, bbb, baa, abb\}$$

$$\begin{aligned} L_{ab}^3 &= L_{ab} \cdot L_{ab} \cdot L_{ab} \\ &= \{a, b\} \cdot \{a, b\} \cdot \{a, b\} \\ &= \{aa, bb, ab, ba\} \cdot \{a, b\} \\ &= \{aaa, aab, bba, bbb, aba, abb, baa, bab\} \end{aligned}$$

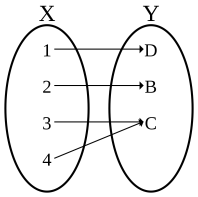
Mengenlehrer (GBI Niveau :D)

Eine Abbildung $f : X \to Y$ ist...

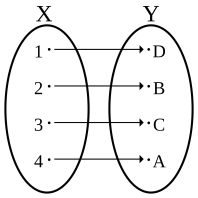
- **Injektiv** (Linkseindeutig), wenn jedes Element in der Zielmenge Y höchstens ein Urbild aus der Ursprungsmenge X hat.



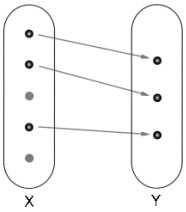
- **Surjektiv** (Rechtstotal), wenn jedes Element in der Zielmenge Y mindestens ein Urbild in der Ursprungsmenge X hat.



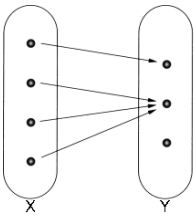
- **Bijektiv**, wenn die Abbildung sowohl injektiv als auch surjektiv ist.



- **Rechtseindeutig**, wenn es für jedes Element in der Ursprungsmenge X maximal ein Zielbild in der Zielmenge Y gibt.



- **Linkstotal**, wenn es für jedes Element in der Ursprungsmenge X mindestens ein Zielbild in der Zielmenge Y gibt.



Relationen die sowohl Linkstotal und Rechtseindeutig sind, nennt man auch Abbildungen. Wenn das gegeben ist kann man eigentlich erst $f : X -> Y$ schreiben. Y ist dann der Zielbereich/die Zielmenge und X ist der Definitionsbereich/die Ursprungsmenge

RegEx

Wissenswertes:

- Hilfssymbole :=$\{|, (,), *, \emptyset\}$
- "$*$ vor $\cdot$ (Konkatenation)"
- "$\cdot$ vor Strich "—" (Oder)
- $\langle R \rangle$ ist die formale Sprache ist, welche mit R gebildet werden kann
- $\langle \emptyset \rangle = \{\}$
- $\langle R_1 | R_2 \rangle = \langle R_1 \rangle \cup \langle R_2 \rangle$
- $\langle R_1 \cdot R_2 \rangle = \langle R_1 \rangle \cdot \langle R_2 \rangle$
- $\langle R* \rangle = \langle R \rangle^*$
- Es gibt **kein** $R+$ sondern $RR*$ Bsp.: Statt $(ab)+$ einfach $ab(ab)*$

Bsp.:
$R = a|b$ dann ist:

$$\langle R \rangle = \langle a|b \rangle = \langle a \rangle \cup \langle b \rangle = \{a\} \cup \{b\} = \{a, b\}$$

$R = (a|b)*$ dann ist:

$$\langle R \rangle = \langle (a|b)* \rangle = \langle a|b \rangle^* = \{a, b\}^*$$

$R = (a * b*)*$ dann ist:

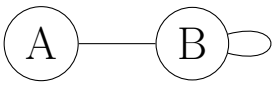
$$\begin{aligned} \langle R \rangle &= \langle (a * b*)* \rangle = \langle a * b* \rangle^* \\ &= (\langle a* \rangle \langle b* \rangle)^* = (\langle a \rangle^* \langle b \rangle^*)^* = (\{a\}^* \{b\}^*)^* \\ &= \{a, b\}^* \end{aligned}$$

Graphen

- Ein gerichteter Graph ist das Paar $G = (V, E)$
  - Knotenmenge V ist endlich und nichtleer (V für engl. vertex)
  - Kantenmenge $E \subseteq V \times V$ (E für engl. edge)
    - muss damit auch endlich sein, darf aber leer sein
- Pfade können über mehrer Kanten führen
- $V^{(+)}$: Menge der nichtleeren Listen von Elementen aus V
- Ein Pfad ist $p = (v_0, \ldots, v_n) \in V^{(+)}$ wenn für jedes $i \in \mathbb{Z}_n$ gilt: $(v_i, v_{i+1}) \in E$
- Die Länge eines Pfades ist die Anzahl der Kanten
- v_n von v_0 ist erreichbar, wenn ein Pfad $p = (v_0, \ldots, v_n)$ existiert
- Wenn der start und endpunkt identisch sind heißt der Pfad geschlossen
- Wenn der geschlossene Pfad größer gleich 1 ist, heißt er Zyklus
- Pfad heißt wiederholungsfrei, wenn
  - der erste bis zum vorletzten Konten verschieden sind $(v_0, \ldots, v_{n-1})$
  - der zweite bis zum letzten Knoten verschieden sind $(v_1, \ldots, v_n)$
  - der erste und letzte Knoten drüfen gleich sein (v_0 und v_n)
  - Einfach: Außer der letzte und erste darf jeder Knoten nur einmal "betreten" werden
- azyklischer Graph: kein Teilgraph ist zyklisch
- Ein Graph ist streng zusammenhängend wenn
  - zwischen jeden beliebigen zwei Knoten (Knotenpaar) aus dem Graphen ein Pfad existiert. Also jeder Punkt von jedem anderen Punkt (sich eingeschlossen) erreichbar ist.

- Ein Graph ist ein gerichteter Baum wenn:
  - es eine Wurzel $r \in V$ gibt, für die gilt:
    - zu jedem Knoten existiert **genau** ein Pfad
    - Wurzel ist immer **eindeutig**
- Der Eingangsgrad eines Knoten ist die Anzahl aller Kanten die zu dem Knoten hinführen
- Der Ausgangsgrad eines Knoten ist die Anzahl aller Kanten die von den Knoten wegführen
- Der Grad eines Knoten ist die Anzahl der Kanten des Knotens (Also Ausgangsgrad + Eingangsgrad)
- Knoten eines Baumes werden Blätter genannt, wenn Sie das Ende des Baumes sind, also Ausgangsgrad $= 0$
- innere Knoten sind dann alle mit Ausgangsgrad > 0
- E^n ist ein Pfad der länge n. Bsp.: E^2 ist ein Pfad der Länge 2
- $(x, y) \in E^2 \Leftrightarrow$ es existiert ein Pfad der Länge 2 von x nach y
- Ein ungerichteter Graph hat einfach nur Kanten und keine "Richtungs" Pfeile
- Knotengrad für ungerichtete Graphen: man zählt alle "Kantenenden"

Beispiel:



$d(B) = 3$

Kontextfreie Grammatik

- N sind alle Nichtterminalsymbole
- T sind alle Terminalsymbole
- $N \cap T = \emptyset$
- S ist der Start und $S \in N$
- P Produktionen, endliche Menge und $P \in N \times V^*$
  - $V = N \cup T$ die Menge aller Symbole
  - Für jeder $(X, w) \in P$ schreibt man $X \to w$
  - Man "ersetz" P durch w

  Bei einem Ableitungsschritt wird ein Terminalsymbol durch abgeleitet. Dieser wird dann mit "$\Rightarrow$" dargestellt, nicht mit Implikations verwechseln!

  Bsp.:
  $G = (\{X\}, \{\text{a,b}\}, X, P)$ mit $P = \{X \to \epsilon, X \to \text{a}X\text{b}\}$

  Dann gilt zB.: $\text{aba}X\text{ba}XXXX \Rightarrow \text{aba}X\text{baa}X\text{b}XXX$ als Ableitungsschritt. (Man ersetzt das X nach ba mit aXb)

  Man kann auch einfach mit einem Index angeben, wie viele Ableitungsschritt getätigt werden.
  $\Rightarrow^2$ aber auch $u \Rightarrow^* v$ wenn v aus u ableitbar ist. $\Rightarrow^0$ ist einfach wieder das selbe.

  - Eine Grammatik erzeugt eine formale Sprache, also einfach alle Wörter die man aus einer Grammtik ableiten kann.
  - $G = (N, T, S, P)$ erzeugt die formale Sprache
    $L(G) = \{w \in T^* | S \Rightarrow^* w\}$
  - Diese formalen Sprachen heißen kontextfrei

Zweierkomplement und Zahlensysteme

Wir können jedes Zahlensystem einer Basis als Alphabet auffassen:
zB.: $Z_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist das Alphabet für Dezimalzahlen. Wir können diese Ziffern und eine Ziffernfolge (Wort) nun umrechnen in unserer bekanntes Dezimalsystem:

$$\begin{aligned} &Num_{10}(x_{k-1} \cdots x_1 x_0) \\ =&10^{k-1} \cdot num_{10}(x_{k-1}) + \cdots + 10^1 \cdot num_{10}(x_1) + 10^0 \cdot num_{10}(x_0) \end{aligned}$$

Das sieht jetzt deutlich komplizierter aus als es eigentlich ist. Deswegen machen wir einfach ein kleines Beispiel:
Wir nehmen das Wort (Ziffernfolge) $w = 324 \in Z_{10}^*$, nun schreiben scheiben wir dieses Wort in unsere Formel:

$$\begin{aligned} Num_{10}(324) &= 10^2 \cdot num_{10}(3) + 10^1 \cdot num_{10}(2) + 10^0 \cdot num_{10}(4) \\ &= 10^2 \cdot 3 + 10 \cdot 2 + 1 \cdot 4 = 324 \end{aligned}$$

Jetzt fragt man sich warum man das überhaupt so umständlich rechnet? Man kann das doch ablesen? Natürlich ist uns im bekannten Dezimalsystem die Umrechnung nur einleuchtend und ablesbar. Wir haben das Dezimalsystem über 10 Jahre in der Schule kennen gelernt. Da wir jetzt aber gesehen habe wie es funktioniert können wir auch andere Zahlensysteme ausprobieren:
zB.: Wir nehmen die Ziffernfolge $w = 1010 \in Z_2^*$. Die Umrechnung sieht wie folgt aus:

$$\begin{aligned} Num_2(1010) &= 2^3 \cdot num_2(1) + 2^2 \cdot num_2(0) + 2^1 \cdot num_2(1) + 2^0 \cdot num_2(0) \\ &= 16 \cdot 1 + 8 \cdot 0 + 4 \cdot 1 + 1 \cdot 0 = 20 \end{aligned}$$

Das kann man jetzt immer weiter führen mit allen möglichen Basen:

- $Z_8 = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$
- $Z_{16} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ Wobei dann hier $num_{16}(A) = 10$, $num_{16}(B) = 11$, ... und so weiter sind.
- Kurzgesagt ist $Num_{base}(w)$ einfach die Übersetzung der Ziffernfolge w eines Zahlensystems der Basis *base* in das uns bekannte Dezimalsystem.

- $num(\epsilon)$ ist immer 0

Das Zweierkomplement hat die Besonderheit eines Vorzeichen-Bits (Sign-Bit) was einfach das Vorzeichen der Dezimalzahl ist und ist immer das höchstwertige Bit (Das erste :D). Dabei markiert 1 eine negative Dezimalzahl und 0 eine positive Dezimalzahl.
zB. 00011010 ist positiv und 11100110 ist negativ

Zur Umrechnung können wir uns einen einfachen Trick merken:

- Wenn das Sign-Bit 0 ist rechnen wir einfach nur die Binärzahl aus:

$$\begin{aligned} 00011010 &= 2^6 \cdot 0 + 2^5 \cdot 0 + 2^4 \cdot 1 + 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 0 \\ &= 16 + 8 + 2 = 26 \end{aligned}$$

- Wenn das Sign-Bit 1 ist rechnen wir zunächst die maximal darstellbare Dezimalzahl für diese Binärzahl aus (Alles auf 1 setzen) und rechnen da nochmal 1 drauf, das ergibt unsere Wertigkeit. Das kann man sich auch einfach merken (In unserem 8-bit Fall ist die Wertigkeit immer 128). Dann rechnen wir noch den Dezimalwert unserer Binärzahl aus und ziehst dann die Wertigkeit ab (Da wir ja eine 1 als Sign-Bit haben):

  Zunächst berechnen wir die Wertigkeit von 11100110:

$$1111111 + 1 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2 + 1 + 1 = 128$$

  Dann rechnen wir einfach die Dezimalwert unserer Binärzahl aus:

$$11100110 = 64 + 32 + 4 + 2 = 102$$

  Und nun ziehen wir einfach die Wertigkeit ab:

$$102 - 128 = -26$$

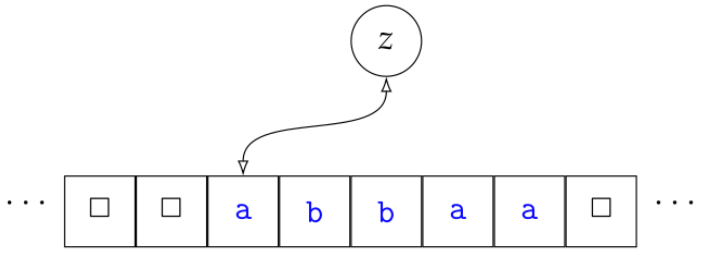
  Wenn man die Wertigkeiten nicht jedes mal ausrechnen will, hier eine Tabelle:

Wertigkeit	256	128	64	32	16	8	4
Bitfolge	9 Bit	8 Bit	7 Bit	6 Bit	5 Bit	4 Bit	3 Bit

Turing-Maschinen

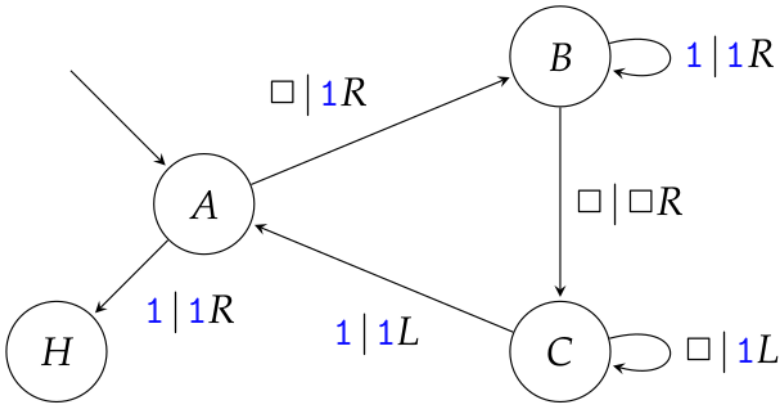
Eine Turingmaschine ist im Grunde einfach nur ein spezieller Automat. Die Besonderheit der Turingmaschinen ist das "Speicherband", der "Schreib-Lese-Kopf" und die Anweisungen für den Schreib-Lese-Kopf in der Ausgabe.

- Das Speicherband ist in einzelne Zellen eingeteilt (Wie ein Array) das nach links und rechts "unendlich" ist.
- Eine leere Zelle wird mit dem Symbol □ markiert.
- Der Schreib-Lese-Kopf zeigt immer auf eine Zelle im Speicherband und auf einen Zustand in der Turingmaschine.
- Wie auch bei Automaten gibt es akzeptierende und nicht akzeptierende Zustände.
- Die Eingabe ist der Inhalt der Zelle, auf welche der Schreib-Lese-Kopf zeigt, und die Ausgabe wird eben in diese nun geschrieben. Dabei gibt es in der Ausgabe zwei Anweisungen:
  - L Der Schreib-Lese-Kopf wird um eins nach links verschoben.
  - R Der Schreib-Lese-Kopf wird um eins nach rechts verschoben.
  - 0 Der Schreib-Lese-Kopf bleibt einfach in der Zelle.
- Eine Turingmaschine muss nicht jedes Szenario abhandeln können. Im Falle eines unbehandelten Falles bleibt die Turingmaschine einfach stehen.
- Wenn man zB. nun ein Wort in die Turingmaschine geben möchte, muss man dieses Wort einfach auf das Speicherband schreiben.



Hier sieht man eine simple Turingmaschine mit einem einzigen Zustand Z. Die Zellen unter dem Zustand sind das Speicherband mit der Eingabe. Der Pfeil symbolisiert den Schreib-Lese-Kopf. Aktuell befindet sich die Turingmaschine im Zustand Z und liest/schreibt in die erste Zelle des Speicherbandes.

Jede Turingmaschine kann zu einer Tabelle umgeschrieben, das Beispiel sollte fürs Verständnis ausreichen:



	A	B	C	H
□	1, R, B	□, R, C	1, L, C	
1	1, R, H	1, R, B	1, L, A	

Die rechte Spalte repräsentiert die mögliche Eingabe (1 oder □). Die obere Zeile sind die verschiedenen Zustände (A, B, C, H). In die Tabelle wird dann Eingetragen was welcher Zustand für welche Eingabe macht.
zB. für die Eingabe □ in Zustand A, schreibt der Schreib-Lese-Kopf eine 1 in die Zelle des Speicherbandes, das R bewegt den Schreib-Lese-Kopf eins nach rechts und der Zustand wechselt zu Zustand B.