# Zusammenfassung GBI [im WS20/21]
Version 1.0.1

Julian Keck
uwgmn

9. Februar 2021

## Inhaltsverzeichnis

# 1 Kapitel 2 - Signale, Nachrichten, Informationen, Daten

## 1.1 Signal

- Physikalische Vorgänge vermitteln einen "Eindruck" dessen, was mitgeteilt werden soll.

## 1.2 Mitteilung

- Wird als Inschrift gespeichert
- Ein Gebilde, um etwas als Signal zu speichern
- Speichermethoden wie Höhle/Pinsel, Papier/Stift, Eisen/Magnet...

## 1.3 Nachricht

- Das, was da steht, also unabhängig von Signal/Speichermethode ist.
- *Beispiel:* 10001: Eins Null Null Null Eins
- 10001 ist immernoch die gleiche Nachricht!
- Abstraktion der Mitteilung; Unabhängig von der Art der Speicherung/des Transports.

## 1.4 Information

- Informationen erhält man durch **Interpretation** einer Nachricht.
- Einer Nachricht wird eine Bedeutung zugeordnet
- Diese Interpretation erfolgt im **Kopf** und nicht im Computer!
- Die Interpretation einer Nachricht kann unterschiedlich sein:
  *Beispiel:*
  - `10001` interpretieren wir als zehntausendundeins
  - `10001` kann man auch als 17 interpretieren (binär).
  - oder auch als 65537 (`10001` als Hexadezimaldarstellung)
- Der Rechner hat "keine Ahnung", was er da tut, macht es aber trotzdem.

## 1.5 Datum

- Singular von **Daten**, nicht ein Tag im Jahr
- Das Bezugssystem der Interpretation ist relevant.
- Auf eine Interpretation einigen (und diese festhalten)

# 2 Kapitel 3 - Mengen, Alphabete, Abbildungen

## 2.1 Mengen

- Eine Menge ist ein "Behälter" mit "Objekten"
- Diese Definition ist gefährlich Russle's-Paradoxon
- Eine Menge kann ein Objekt enthalten oder nicht (nicht beides und auch nicht keines von beidem)
- *Beispiel:* $A = \{1, 2, 3\}$ ist eine Menge
- Auf Mengen kann man die bekannten Operationen wie Mengenschnitt ($\cap$), Mengenvereinigung ($\cup$) und Mengendifferenz ( $\setminus$ ) ausführen
- Teilmengenrelationen seien von jetzt an bekannt ($\subsetneq, \subseteq$); auf $\subset$ sollte verzichtet werden.
- Sei $A$ eine endliche Menge[1]. Dann bezeichnet $|A|$ die **Kardinalität** von $A$, also genau die Anzahl der Elemente in $A$

## 2.2 Alphabet

- Ein Alphabet ist eine nichtleere Menge von Zeichen oder Symbolen
- Was ein Zeichen ist, sei nicht weiter spezifiziert[2], es handelt sich um einen elementaren Baustein von Inschriften
- *Beispiel:*
  - Das deutsche Alphabet
  - $A = \{\texttt{0,1}\}$ (Alphabet aller binären Worte)
  - $A = \{\texttt{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}\}$ (Alphabet aller hexadezimalen Worte)
  - ASCII
- Manchmal sind Zeichen auch etwas abstrakter, *Beispiel:* $\boxed{\texttt{int}}$ $\boxed{\texttt{counter}}$ $\boxed{\texttt{=}}$ $\boxed{\texttt{42}}$ $\boxed{\texttt{;}}$

## 2.3 Paare

- Ein Paar $(a, b)$ hat die erste Komponente $a$ und die zweite $b$.
- Im Allgemeinen gilt: $(a, b) \neq (b, a)$

## 2.4 Kartesische Produkt

- Seien $A, B$ zwei Mengen
- Das kartesische Produkt $A \times B$ ist definiert als: $A \times B = \{(a, b) \mid a \in A \land b \in B\}$
- $A^2 := A \times A$

[1] Wir beschränken und in GBI auf endliche Mengen. Es gibt zwar auch die Kardinalität unendlicher Mengen, diese sind aber etwas komplizierter definiert und in GBI sowieso nicht wirklich hilfreich.

[2] Dies sollte aus dem täglichen Leben bereits bekannt sein. Beispielsweise ist `a` oder `x` ein Zeichen. Genauso aber auch $\int$

# 3 Kapitel 4 - Wörter (und Sprachen)

## 3.1 Wörter

- *Beispiel:*
  - `Apfelmus`
  - `Milchreis` - Symbole dürfen also auch mehrfach vorkommen
- Das Leerzeichen betrachten wir auch als Zeichen. Manchmal schreiben wir explizit ␣
- `Hallo␣Welt` ist eine Folge von Zeichen, also ein Wort und nicht zwei.
- Formalere Definition eines Wortes:
  - $\mathbb{Z}_n :=$ "die n kleinsten nichtnegativen ganzen Zahlen"
    $\mathbb{Z}_n := \{i \in \mathbb{N}_+ \mid 0 \leq i < n\}$
    *Beispiel:* $\mathbb{Z}_0 = \{\}, \mathbb{Z}_1 = \{0\}, \mathbb{Z}_4 = \{0, 1, 2, 3\}$
  - Ein Wort über dem Alphabet $A$ ist eine surjektive Abbildung:
    $\mathsf{w} : \mathbb{Z}_n \to B$ mit $B \subseteq A$
  - $|w| := n$ ist die Länge des Wortes
  - Das Wort der Länge 0 wird als leeres Wort oder $\varepsilon$ bezeichnet.
- Die Menge aller Wörter der Länge $n$ wird als $A^n$ geschrieben.
  *Beispiel:*
  - $A^0 = \{\varepsilon\}$
  - $A^1 = A = \{\texttt{a},\texttt{b}\}$
  - $A^3 = \{\texttt{aaa},\texttt{aab},\texttt{aba},\texttt{abb},\texttt{baa},\texttt{bab},\texttt{bba},\texttt{bbb}\}$
- Die Menge aller Wörter über dem Alphabet $A$ wird als $A^*$ geschrieben
  $A^* = A^0 \cup A^1 \cup \dots$ oder $A^* = \bigcup_{i \in \mathbb{N}_0} A^i$, wenn man eine Notation ohne Pünktchen bevorzugt.
- Die Konkatenation bezeichnet die Aneinanderreihung mehrerer Worte, geschrieben wird diese durch $\cdot$
  *Beispiel:*
  $x = \texttt{Hallo}, y = \texttt{Welt}$
  $xy := x \cdot y = \texttt{HalloWelt}$
- Die Konkatenation ist im Allgemeinen nicht kommutativ.
  *Beispiel:*
  $\texttt{SCHRANK} \cdot \texttt{SCHLÜSSEL} = \texttt{SCHRANKSCHLÜSSEL} \neq \texttt{SCHLÜSSELSCHRANK} = \texttt{SCHLÜSSEL} \cdot \texttt{SCHRANK}$
- Die Potenzen von Wörtern sind induktiv definiert durch:
  $\forall\, \mathsf{w} \in A^* : \quad \mathsf{w}^0 = \varepsilon \qquad \mathsf{w}^{n+1} = \mathsf{w}^n \cdot \mathsf{w}$

## 3.2 Binäre Operation

- Eine binäre Operation auf einer Menge $M$ ist eine Abbildung $f : M \times M \to M$
- Oftmals wird eine Infixschreibweise benutzt, *Beispiel:* statt $+(3, 8) = 11$ schreibt man $3 + 8 = 11$
- Eine binäre Operation $\star$ ist **kommutativ**, wenn gilt:
  $$\forall x \in M\, \forall y \in M : \quad x \star y = y \star x$$
- Eine binäre Operation $\star$ ist **assoziativ**, wenn gilt:
  $$\forall x, y, z \in M : \quad (x \star y) \star z = x \star (y \star z)$$

# 4 Kapitel 5 - Aussagenlogik

## 4.1 Aussagen

- Aussagen sind objektiv wahr oder falsch
  *Beispiel:*
  - «Die Abbildung $f : \mathbb{R} \to \mathbb{R} : x \mapsto \sqrt{x}$ ist injektiv» **wahr**
  - «Die Abbildung $f : \mathbb{R} \to \mathbb{R} : x \mapsto \sqrt{x}$ ist surjektiv» **falsch**
- Es gibt auch scheinbar sinnvolle Aussagen, die in Wirklichkeit aber sinnlos sind:
  «Ein Barbier ist ein Mann, der genau diejenigen Männer rasiert, die sich nicht selbst rasieren.» *Frage: Wer rasiert den Barbier?*
- Jede Aussage besitzt einen eindeutigen Wahrheitswert (nämlich **wahr** oder **falsch**).

## 4.2 Aussagenlogische Konnektive

Seien $P$ und $Q$ zwei Aussagen

- **Negation:** «nicht $P$»: $\neg P$
- **logisches Und:** «$P$ und $Q$»: $P \wedge Q$
- **logisches Oder:** «$P$ oder $Q$»: $P \vee Q$
- **Implikation:** «$P$ impliziert $Q$»: $P \to Q$
  «Wenn $P$, dann $Q$»

Der Wahrheitswert einer zusammengesetzten Aussage ist durch die Wahrheitswerte der Teilaussagen **eindeutig** festgelegt. Es gibt keine Abhängigkeit vom konkreten Inhalt der Aussagen, auch nicht bei «Wenn..., dann...».
*Beispiel:*

- $P$ : «2014 wurden in Japan etwa 4,7 Mio. PKW neu zugelassen»
- $Q$ : «1999 gab es in Deutschland etwa 11,2 Mio. Internet-Nutzer»
- «Wenn $P$, dann $Q$» ist wahr, da die Teilaussagen wahr sind.

Definition des Alphabet der aussagenlogischen Formeln:
$A_{AL} := \{(,), \neg, \wedge, \vee, \to\} \cup \{\text{Aussagevariablen}\}$
Aussagenlogische Formeln sind die Worte aus $A_{AL}^*$, die sinnvoll geklammert und formuliert sind, dies wird im Weiteren nicht betrachtet.[3]

## 4.3 Boolsche Funktionen

- Aussagenlogische Formeln sind erstmal nur Zeichenketten.
- Sogenannte boolsche Funktionen:
  $f : \mathbb{B}^k \to \mathbb{B}, \qquad \text{wobei } \mathbb{B} := \{\mathbf{w}, \mathbf{f}\}$
  sind Funktionen, die Wahrheitswerte auf Wahrheitswerte abbilden. Für die vorher definierten aussagenlogischen Konnektive gibt es die entsprechenden boolschen Funktionen:
  $b_\neg(x_1), b_\wedge(x_1, x_2), b_\vee(x_1, x_2), b_\to(x_1, x_2)$, wobei diese im Regelfall als Infix notiert werden.

[3] Die korrekte Klammerung sollte durch den gesunden Menschenverstand ableitbar sein.

## 4.4 Bedeutung und Interpretation aussagenlogischer Formeln

- Sei $V$ die Menge von Aussagevariablen
- Interpretation $I : V \to \mathbb{B}$
  Dann ist $\mathbb{B}^V$ die Menge aller Interpretationen
- Für jede aussagenlogische Formel $F$ ist die **Auswertung** $val_I(F)$ folgendermaßen definiert:

$$\begin{aligned} val_I(\neg G) &= b_{\neg}(val_I(G)) \\ val_I(G \wedge H) &= b_{\wedge}(val_I(G), val_I(H)) \\ val_I(G \vee H) &= b_{\vee}(val_I(G), val_I(H)) \\ val_I(G \to H) &= b_{\to}(val_I(G), val_I(H)) \end{aligned}$$

- Zwei Formeln $G$ und $H$ heißen äquivalent, wenn für jede Interpretation $I$ gilt:
  $val_I(G) = val_I(H)$
  *Beispiel:*
  - `¬P ∨ ¬Q` und `¬(P ∧ Q)`
  - `¬¬P` und `P`
- Geschrieben wird dies als $\equiv$
  *Beispiel:* `P` $\equiv$ `¬¬P`
- Randbemerkung:
  G=`P ∧ P` und H=`Q ∧ Q` sind nicht äquivalent, da `P` und `Q` nicht in jeder Interpretation $I$ den selben Wert haben.

## 4.5 Modelle

- Eine Interpretation $I$ ist Modell einer Formel $G$, wenn $val_I(G) = \mathbf{w}$ ist.
- Interpretation I ist Modell der Formelmenge $\Gamma$, wenn gilt:
  $\forall G \in \Gamma : val_I(G) = \mathbf{w}$
- $\Gamma \models G$ bedeutet: «Jedes Modell von $\Gamma$ ist auch Modell von $G$»
  Definiere:
  $H \models G \quad :\Leftrightarrow \quad \{H\} \models G$
  $\models G \quad :\Leftrightarrow \quad \{\} \models G \quad \Leftrightarrow$ $G$ ist für alle Interpretationen wahr ($G$ ist eine **Tautologie**)
  *Beispiel:* `P ∨ ¬P`

## 4.6 Beweisbarkeit

- Diese aussagenlogischen Formeln sind alle durch das in der Vorlesung vorgestellte Kalkül beweisbar.
  Dafür werden folgende drei Axiome (tautologe Formeln) definiert:
  - (G → (H → G)
  - (G → (H → K)) → ((G → H) → (G → K))
  - (¬H → ¬G) → ((¬H → G) → H)

  Als Beweisabschluss gibt es auch noch den Modus Ponens:
  (G ∧ (G → H)) ⇒ H

# 5 Kapitel 6 - Induktives Vorgehen

## 5.1 ...

Das induktive Vorgehen sollte aus HM und LA genug bekannt sein, wenn ich Zeit habe, kommt es hier noch dazu.
Der Vollständigkeit halber sei es hier aber bereits jetzt erwähnt.

# 6 Kapitel 7 - Formale Sprachen

## 6.1 Begriffsklärung und Defnitionen

- Eine formale Sprache $L$ ist eine Teilmenge aller Wörter eines Alphabetes $A$.
  $L \subseteq A^*$
- Das Produkt / die Konkatenation zweier formaler Sprachen $L_1$ und $L_2$ ist folgendermaßen definiert:
  $L_1L_2 := L_1 \cdot L_2 := \{w_1w_2 \mid w_1 \in L_1 \land w_2 \in L_2\}$
  *Beispiel:* $L_1 = \{\mathtt{a}\}$ $L_2 = \{\mathtt{b}\}$ $L_1 \cdot L_2 = \{\mathtt{a}\} \cdot \{\mathtt{b}\} = \{\mathtt{a} \cdot \mathtt{b}\} = \{\mathtt{ab}\}$
- $\{\varepsilon\}$ ist das neutrale Element der Konkatenation zweier formaler Sprachen:
  $\{\varepsilon\} \cdot L = L = L \cdot \{\varepsilon\}$
- Die Potenz formaler Sprachen ist naheliegend definiert:

$$L^0 = \{\varepsilon\} \tag{1}$$
$$\forall n \in \mathbb{N}_0 : L^{n+1} = L \cdot L^n \tag{2}$$

## 6.2 Konkatenationsabschlüsse

- Kleen-Star / kleensche Hülle: $L^* = \bigcup_{i \in \mathbb{N}_0} L^i$
- $\varepsilon$-freier Konkatenationsabschluss: $L^+ = \bigcup_{i \in \mathbb{N}_+} L^i$
- Es resultiert: $L^* = L^0 \cup L^+$
- **Warnung**: $L^+$ ist nur dann tatsächlich $\varepsilon$-frei, wenn gilt: $\varepsilon \notin L(= L^1)$
- Außerdem gilt: $\{\}^* = \{\varepsilon\}$

# 7 Kapitel 8 - Codierungen

## 7.1 Darstellung von Zahlen

- Seien $Z_{10} = \{\texttt{0}, \dots, \texttt{9}\}$ die Ziffern.
- Die Bedeutung einer Ziffer als Zahl wird durch $num_{10} : Z_{10} \to \mathbb{Z}_{10}$ dargestellt.
- $Num_{10} : Z_{10}^* \to \mathbb{N}_0$ gibt die Bedeutung einer Ziffernfolge:
  $Num_{10}(x_{k-1} \cdots x_1 x_0) = 10^{k-1} \cdot num(x_{k-1}) + \dots 10^1 \cdot num_{10}(x_1) + 10^0 \cdot num_{10}(x_0)$
  Bzw.:
  $Num_{10}(\varepsilon) = 0 \qquad Num_{10}(wx) = 10 \cdot Num_{10}(w) + num_{10}(x)$
- Analoge Definitionen gelten auch für andere Basen, das Ziffernalphabet bis Basis 16 lautet:
  $\{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}, \texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}, \texttt{F}\}$
- Es folgt zwangsläufig eine Uneindeutigkeit[4] von Zeichenketten wie `111`:
  - $Num_2\ (\texttt{111})$ = «sieben»
  - $Num_8\ (\texttt{111})$ = «dreiundsiebzig»
  - $Num_{10}(\texttt{111})$ = «einhundertelf»
  - $Num_{16}(\texttt{111})$ = «zweihundertdreiundsbiebzig»

### 7.1.1 Division mit Rest

Operationen div und mod.
Es seien $x \in \mathbb{N}_0$ und $y \in \mathbb{N}_+$:

- $x$ **mod** $y \in \mathbb{N}_0$
  Rest der ganzzahligen Division $x$ durch $y$
  $0 \leq (x \bmod y) < y$
- $x$ **div** $y \in \mathbb{N}_0$
  Ergebnis der ganzzahligen Division $x$ durch $y$
- Es gilt:
  $x = y \cdot (x \operatorname{div} y) + (x \bmod y)$

### 7.1.2 k-äre Darstellung von Zahlen

Sei $k \in \mathbb{N}_0$ und $k \geq 2$

$$\begin{aligned} Repr_k &: \mathbb{N}_0 \to \mathbb{Z}_k \\ n &\mapsto \begin{cases} repr_k(n), & \text{falls } n < k \\ Repr_k(n \operatorname{div} k) \cdot repr_k(n \bmod k), & \text{falls } n \geq k \end{cases} \end{aligned}$$

Es gilt: $Num_k(Repr_k(n)) = n$, umgekehrt auch, außer, dass führende Nullen wegfallen.

## 7.2 Zweierkomplementdarstellung [ZKPL]

Sei $l \in \mathbb{N}_+$:

- $\mathbb{K}_l = \{x \in \mathbb{Z} \mid -2^{l-1} \leq x \leq 2^{l-1} - 1\}$
  *Beispiel:* $\mathbb{K}_4 = \{-2, -1, 0, 1\}$

[4] Hierauf wurde bereits in Kapitel 2 - Information eingegangen.

- $Zkpl_l : \mathbb{K}_l \to \{0, 1\}^l$

$$Zkpl_l(x) = \begin{cases} 0bin_{l-1}(x), & \text{falls } x \geq 0 \\ 1bin_{l-1}(2^{l-1} + x), & \text{falls } x < 0 \end{cases}$$

- Man kann die Zkpl-Darstellung auch erhalten, indem man die Zahl binär aufschreibt, alle Bits einmal *kippt* und dann 1 addiert.

## 7.3 Notationen für Funktionen

- Menge aller Funktionen von $A$ nach $B$: $B^A := \{f \mid f \text{ ist Funktion } f : A \to B\}$
  Für endliche Mengen gilt: $|B^A| = |B|^{|A|}$
- Die Funktionskomposition, Identität, Umkehrfunktion, Links- und Rechtsinverse sind aus der LA und HM bekannt, und seinen von nun auch hier als bekannt vorausgesetzt.

## 7.4 Übersetzungen

- Eine Übersetzung verändert den Inhalt der Nachricht nicht. Lediglich die Darstellung wird verändert.
- Gründe:
  - Legalität: nur bestimmter Zeichensatz erlaubt
  - Lesbarkeit: vergleiche `A3` mit `10100011`
  - Verschlüsselung: keine Lesbarkeit für Ausenstehende
  - Kompression: Übersetzungen können kürzer sein, ohne das Alphabet zu vergrößern. *Beispiel:* Huffman-Codes
  - Fehlererkennung oder Fehlerkorrektur

## 7.5 Codierungen

- Wenn $f$ injektiv ist, gibt es von $f(x)$ einen **eindeutigen** Weg zurück nach $x$
- Codierung: Übersetzung mit injektivem $f$
- Codewörter: die $f(x)$
- Code: $\{f(w) \mid w \in L_A\}$
- Wenn $L_A$ unendlich ist, kann man nicht alle Möglichkeiten aufzählen
  $\Rightarrow$ Homomorphismen und Block-Codierung bieten Auswege.

### 7.5.1 Homomorphismen

Ein Homomorphismus ist eine Abbildung $h : A^* \to B^*$ $\quad (A, B$ Alphabete) mit:
$\forall w_1, w_2 \in A^* :\ h(w_1 w_2) = h(w_1)h(w_2)$
Gilt außerdem: $\forall x \in A :\ h(x) \neq \varepsilon$, so ist $h$ $\varepsilon$-frei.
$\Rightarrow$ Die Bilder der einzelnen Zeichen legen den Homomorphismus eindeutig fest.
Damit die Decodierung fehlerfrei Funktioniert, müssen die Codewörter auch noch präfixfrei sein[5].

[5] Auf dies wird in der Vorlesung noch sehr viel eingegangen und an einigen Beispielen (z.B. auch Unicode) weiter erläutert. Für die Klausur sollten diese Informationen aber nicht sonderlich relevant sein.

## 7.6 Huffman-Codierung

- gegeben ist ein Alphabet $A$ und ein Wort $w \in A^*$
- Huffman-Codierung ist nun:
  - eine Abbildung $h : A^* \to \{0, 1\}$
  - $\varepsilon$-freier Homomorphismus
  - $h$ wird typischerweise auf $w$ angewendet
- Bei der Huffman-Codierung werden häufige Symbole durch kürzere Wörter und seltene Symbole durch längere Wörter ersetzt.

### 7.6.1 Konstruktion des Huffman-Baums

- Man betrachte die Menge $M_i$ als «zu betrachtende Symbolmenge mit ihren Häufigkeiten»
- Am Anfang besteht die Menge aus Tupeln von jeweils Zeichen und Häufigkeit. Man zeichnet für jedes Zeichen einen Knoten mit Zeichen und Häufigkeit.
- Nun verbindet man zwei Knoten mit der kleinsten Häufigkeit. Dies bildet einen neuen Knoten, er bekommt die addierte Häufigkeit der beiden ursprünglichen Knoten.
- Nach *links* führende Kanten werden mit `0` beschriftet, nach *rechts* führende mit `1`
- Die Beschriftung der Kanten (von Wurzel zu Blätter gelesen und konkateniert) ergeben die Codierung des Zeichens.

### 7.6.2 Eigenschaften der Huffman-Codierung

- Die Hufmann-Codierung ist nicht eindeutig.
- Alle Möglichkeiten sind aber «gleich gut»
- Man kann auch «Block-Codierung» anwenden, im Wesentlichen handelt es sich um den selben Algorithmus, allerdings werden nicht einzelne Zeichen sondern mehrere Zeichen auf einmal codiert. (z.B. «ei» oder «au»)

# 8 Kapitel 9 & 10 - Mima

## 8.1 Speicher

### 8.1.1 Begriffsklärung

- Bit: (Genau) ein Zeichen des Aplhabets $\{0, 1\}$
- Byte: Ein Wort aus **8** Bits
- Ein Speicher hat zu jedem Zeitpunkt für jede Adresse Informationen, welcher Wert dort steht.
  Vorstellen, wie eine Tabelle mit zwei Spalten
- Formalisierung des Speichers als Abbildung: $m(a) =$«Wert an Adresse a»
- Abbildung zum Lesen des Speichers: *memread*

$$\begin{aligned} memread : Mem \times Adr &\to Val \\ (m, a) &\mapsto m(a) \end{aligned}$$

- Abbildung zum Schreiben im Speicher: *memwrite*

$$\begin{aligned} memwrite : Val^{Adr} \times Adr \times Val &\to Val^{Adr} \\ (m, a, v) &\mapsto m' \end{aligned}$$

  Wobei folgendes gilt:

$$m'(a') = \begin{cases} v, & \text{falls } a' = a \\ m(a'), & \text{falls } a' \neq a \end{cases}$$

## 8.2 Prozessor

Programmierung der Mima[6]

### 8.2.1 Grober Aufbau der MIMA

Hauptspeicher

- Adressen haben 20 bit
- Werte haben 24 bit
- Im Programmspeicher steht an jeder Adresse *genau* ein Befehl.
  Entweder 4-bit Befehlscodierung und 20 Bit Adresse, oder 8-bit Befehlscodierung und der Rest ist egal.
- Im Speicher der MIMA wird die Zweierkomplementdarstellung genutzt.

Prozessor

- Register: Speicher für ein Datenwort/Adresse, z.B. Akkumulator
- Rechenwerk: ALU (*arithmetic logic unit*), es gibt verschiedene Funktionen, Argumente aus dem Akku bzw Befehl/Speicher

[6] Im weiteren wird nicht auf den genauen Aufbau der MIMA eingegangen, da dies erfahrungsgemäß nicht klausurrelevant ist (Worsch hat noch nie eine Prüfungsaufgabe dazu gestellt). Allerdings solle man sich dies trotzdem nocheinmal kurz in den entsprechenden VL-Folien anschauen.

- Steuerwerk:
  IR: instruction register
  IAR: instruction address register

- Speicherwerk
  SAR: storage address register
  SDR: storage data register

## 8.3 Maschinenbefehle der MIMA

- Verschiedene Typen von Befehlen:
  - Transport von Daten
  - Verarbeitung von Daten
  - Beeinflussing der Befehlsreihenfolge

- Es gibt verschiedene Notationsmöglichkeiten, z.B. Bitfolgen, symbolische Namen für Adressen und Befehle

- Befehlsübersicht:

| Befehl | Erklärung |
|---|---|
| LDC c | c $\rightarrow$ Akku |
| LDV a | M(a) $\rightarrow$ Akku |
| STV a | M(a) $\leftarrow$ Akku |
| LDIV a | M(M(a)) $\rightarrow$ Akku |
| STIV a | M(M(a)) $\leftarrow$ Akku |
| ADD a | Akku + M(a) $\rightarrow$ Akku |
| AND a | Akku AND M(a) $\rightarrow$ Akku |
| OR a | Akku OR M(a) $\rightarrow$ Akku |
| XOR a | Akku XOR M(a) $\rightarrow$ Akku |
| NOT | Einskomplement von Akku $\rightarrow$ Akku |
| RAR | rotiert Akku um 1 nach rechts $\rightarrow$ Akku |
| EQL a | $\text{Akku} \leftarrow \begin{cases} -1, & \text{falls Akku} = \text{M(a)} \\ 0, & \text{sonst} \end{cases}$ |
| JMP a | fahre mit Befehl an Adresse a fort |
| JMN a | falls Akku $<$ 0: JMP a |
| HALT | stoppt die MIMA |
| | c: Konstante, a: Adresse, M(a): Wert an Adresse a |

- Wichtig: Jedes Programm **muss** mit HALT beendet werden, sonst läuft es endlos weiter.

- Mit LDC können keine negativen Konstanten geladen werden, da wir nur 20-bit-Konstanten laden können[7].

[7] Man kann sich dadurch behelfen, dass man die Zahl positiv läd, alle Bits kippt und 1 addiert.

# 9 Kapitel 11 - Dokumente

## 9.1 Was sind Dokumente?

- Überall gibt es Inschriften
  - Briefe, Kochrezepte, Zeitungsartikel
  - Seiten im WWW
  - Diese GBI-Zusammenfassung
- DREI Aspekte sind für den Leser relevant:
  - Inhalt des Textes
  - Struktur des Textes
  - Erscheinungsbild / (äußere) Form des Textes
- In der Regel steht der **Inhalt** für den Autor und Leser im Vordergrund.
  Struktur und Form helfen nur beim Verständnis des Inhalts
- Dokumente sind Texte mit Inhalt, Struktur und Form

## 9.2 Formalisierung von Dokumenten

Für (nicht allzu) komplexe Regeln, wie ein Dokument auszusehen hat, bieten unsere formalen Sprachen keine ausreichende Methodik, um diese Regeln formal zu definieren. Wir benötigen kontextfreie Grammatiken

# 10 Kapitel 12 - Kontextfreie Grammatiken

## 10.1 Das Problem mit formalen Sprachen...

- Nicht jede Spezifikation (z.B. die Sprache aller Java-Programme) kann mittels formaler Sprache ausgedrückt werden.
- *Beispiel:* Den Block-Syntax von Java kann man folgendermaßen versuchen zu spezifizieren: $L = \{\varepsilon\} \cup LL \cup \{\texttt{(}\}L\{\texttt{)}\}$

$\Rightarrow$ Diese Gleichung ist lösbar, aber nicht **eindeutig** lösbar.

## 10.2 Definition kontextfreier Grammatiken

- Eine kontextfreie Grammatik ist ein 4-Tupel: $G = (N, T, S, P)$
- $N$ ist das Alphabet der Nichtterminalsymbole
- $T$ ist das Alphabet der Terminalsymbole
- Es gilt stets: $N \cap T = \{\}$
- $S \in N$ ist das Startsymbol
- $P \subseteq N \times (N \cup T)$ ist die *endliche* Menge von Produktionen
  - $(X, w) \in P$ schreibt man als $X \to w$
  - Bedeutung: man kann $X$ durch $w$ ersetzen.
  - Terminalsymbole werden **nicht** ersetzt, links des Pfeils stehen also immer Nichtterminalsymbole.
  - Eine Ersetzung aus $P$ ist immer möglich, also unabhängig vom Kontext ($\Rightarrow$ kontextfrei).
  - Anstatt $P = \{X \to \mathsf{w}_1, X \to \mathsf{w}_2, X \to \mathsf{w}_3\}$ erlauben wir uns $P = \{X \to \mathsf{w}_1 \mid \mathsf{w}_2 \mid \mathsf{w}_3\}$
- *Beispiel:* $G = (\{X\}, \{\texttt{a}, \texttt{b}\}, X, \{X \to \varepsilon, X \to \texttt{a}X\texttt{b}\})$

## 10.3 Ableitungsschritt

- Die Ableitung von Wörtern erfolgt mittels einer Produktion: $u \Rightarrow v$
- Das bedeutet: $v \in V^*$ ist in einem Schritt aus $u \in V^*$ ableitbar
- *Beispiel:* $G = (\{X\}, \{\texttt{a}, \texttt{b}\}, X, \{X \to \varepsilon, X \to \texttt{a}X\texttt{b}\})$, $\quad u = \texttt{a}X\texttt{b}$, $\quad v = \texttt{ab}$, dann gilt: $u \Rightarrow v$
  Allerdings: $u = \texttt{a}X\texttt{b}$, $\quad \widetilde{v} = \texttt{aabb}: \quad u \not\Rightarrow \widetilde{v}$, sondern $u \Rightarrow \texttt{aa}X\texttt{bb} \Rightarrow \widetilde{v}$
- Es werden auch Ableitungsfolgen definiert:
  $u, v \in V^*, \quad i \in \mathbb{N}_0$
  - $u \Rightarrow^0 v$ gdw. $u = v$
  - $u \Rightarrow^{i+1} v$ gdw. $\exists\, \mathsf{w} \in V^* : u \Rightarrow w \Rightarrow^i v$
  - $u \Rightarrow^* v$ gdw. $\exists\, i \in \mathbb{N}_0 : u \Rightarrow^i v$

## 10.4 Jede Grammatik erzeugt eine formale Sprache

- Sei $G = (N, T, S, P)$ eine kontextfreie Grammatik. Dann ist $L(G) = \{\mathsf{w} \in T^* \mid S \Rightarrow^* \mathsf{w}\} \subseteq T^*$ die erzeugte formale Sprache
- Eine solche Sprache heißt kontextfrei.

# 11 Kapitel ??? - Relationen

## 11.1 Definition einer (binären) Relation

- Eine (binäre) Relation $R$ ist eine Teilmenge $R \subseteq M \times M$
- Man sagt, $a, b \in M$ sind in Relation, wenn gilt: $(a, b) \in R$

## 11.2 Produkt von Relationen

- Seien $R \subseteq M_1 \times M_2$ und $S \subseteq M_2 \times M_3$ zwei Relationen
- Das Produkt der Relationen $R$ und $S$ ist definiert:
  $S \circ R = \{(x, z) \in M_1 \times M_3 \mid \exists\, y \in M_2 : (x, y) \in R \wedge (y, z) \in S\}$
- $\circ$ ist assoziativ:
  $(T \circ S) \circ R = T \circ (S \circ R)$
- Ist $R \subseteq M \times M$ eine binäre Relation auf $M$, dann definiert man die Potenz $R^i$:

$$R^0 = I_M$$
$$\forall i \in \mathbb{N}_0 : \; R^{i+1} = R^i \circ R$$

## 11.3 Eigenschaften von Relationen

Sei $R \subseteq M \times M$ eine Relation auf $M$

- $R$ heißt reflexiv, wenn $I_M \subseteq R$, also wenn $\forall\, x \in M : (x, x) \in R$
- $R$ heißt transitiv, wenn $R \circ R \subseteq R$, also wenn
  $\forall\, x, y, z \in M : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

## 11.4 Reflexiv-transitive Hülle

- Die reflexiv-transitive Hülle einer Relation $R$ ist definiert als: $R^* = \bigcup_{i \in \mathbb{N}_0} R^i$
- *Beispiel:* Sei $R = \{(n, n+1) \mid n \in \mathbb{N}_0\} \subseteq \mathbb{N}_0 \times \mathbb{N}_0$

$$\begin{aligned} R \circ R &= \{(n, n+2) \mid n \in \mathbb{N}_0\} \\ R^0 &= \{(n, n) \mid n \in \mathbb{N}_0\} \\ R^1 &= \{(n, n+1) \mid n \in \mathbb{N}_0\} \\ R^2 &= \{(n, n+2) \mid n \in \mathbb{N}_0\} \\ R^* &= \{(n, m) \mid n \leq m\} \end{aligned}$$

### 11.4.1 Eigenschaften der reflexiv-transisitven Hülle

- $R^*$ ist immer reflexiv
  $I_M = R^0 \subseteq R^*$
- $R^*$ ist immer transitiv
- $R^*$ ist die *kleinste* Relation, die R umfasst und reflexiv und transitiv ist.

# 12 Kapitel 13 - Prädikatenlogik - Erste Stufe

## 12.1 Einführung Prädikatenlogik

- Prädikatenlogische Formeln sind komplizierter aufgebaut als aussagenlogische Formeln
- Sie bestehen aus drei Schritten:
  - Terme (Variablensymbole, Konstantensymbole, Funktionssymbole)
  - Atomare Formeln (aus Termen, mittels Relationssymbolen)
  - Prädikatenlogische Formeln (atomare Formeln unter Verwendung aussagenlogischer Konnektive und Quantoren)

## 12.2 Terme

- Variablensymbole: Aplhabet $Var_{PL}$
  - $\mathtt{x}_i$ (für endlich viele $i \in \mathbb{N}_0$)
  - kurz auch manchmal `x`, `y`, `z`
- Konstantensymbole: Alphabet $Const_{PL}$
  - $\mathtt{c}_i$ (für endlich viele $i \in \mathbb{N}_0$)
  - kurz auch manchmal `c`, `d`
- Funktionssymbole: Alphabet $Fun_{PL}$
  - $\mathtt{f}_i$ (für endlich viele $i \in \mathbb{N}_0$)
  - kurz auch manchmal `f`, `g`, `h`
  - Jedes $\mathtt{f}_i \in Fun_{PL}$ hat Stelligkeit $ar(\mathtt{f}_i) \in \mathbb{N}_+$ [engl: arity]
- $\Rightarrow A_{Ter} = \{\mathtt{(}, \mathtt{,}, \mathtt{)}\} \cup Var_{PL} \cup Const_{PL} \cup Fun_{PL}$
- Aus diesem Alphabet lassen sich alle Terme ableiten, allerdings auch viel Blödsinn. Die korrekte Syntax sei aber von nun an als bekannt vorausgesetzt.[8]
  *Beispiel:* Korrekte Terme: `c` `f(x, g(c))` `g(c)`
  *Beispiel:* Falsche Terme: `xy` `f)x,y(` `c(x)`

## 12.3 Atomare Formeln

- Relationssymbole: Alphabet $Rel_{PL}$
  - $\doteq$ immer dabei
  - $\mathtt{R}_i$ (für endlich viele $i \in \mathbb{N}_0$)
  - kurz auch manchmal `R`, `S`
  - Jedes $\mathtt{R}_i \in Rel_{PL}$ hat Stelligkeit $ar(\mathtt{R}_i) \in \mathbb{N}_+$ [engl: arity]
- $A_{Rel} = A_{Ter} \cup Rel_{PL}$
- Aus $A_{Rel}$ können alle gültigen atomaren Formeln abgeleitet werden. Die korrekte Syntax sein von nun an bekannt.
  *Beispiel:* ableitbar: $\mathtt{g(x)} \doteq \mathtt{f(x,g(z))}$ `S(c)` `R(y,c,g(x))`
  *Beispiel:* nicht ableitbar: $\mathtt{x} \doteq \mathtt{y} \doteq \mathtt{z}$ $\mathtt{R} \doteq \mathtt{f}$ $\mathtt{S(x)} \doteq \mathtt{S(x)}$

[8] Die Ursache dafür, dass dies an dieser Stelle nicht weiter betrachtet wird, hat damit zu tun, dass die korrekte Darstellung dieser Grammatik einen großer Haufen Arbeit in LaTeXbedarf, wobei die eigentliche Aussage durch relativ einfach und Intuitiv ist.

## 12.4 Prädikatenlogische Formeln

- $A_{For} = A_{Rel} \cup \{\neg, \wedge, \vee, \rightarrow, \forall, \exists\}$
  $\forall$ Allquantor
  $\exists$ Existenzquantor

- Aus $A_{For}$ lassen sich alle Prädukatenlogischen Formeln ableiten, die korrekte Syntax sein von nun an bekannt.

## 12.5 Interpretation prädikatenlogischer Formeln

- Seien die Alphabete $Const_{PL}$, $Fun_{PL}$, $Rel_{PL}$ gegeben.

- Die Interpretation $(D, I)$ besteht aus:

  - Der nichtleeren Menge $D$, das Universum [engl: domain, dt: Domäne]
  - $I(\mathtt{c}_i \in D$ für $\mathtt{c}_i \in Const_{PL}$
  - $I(\mathtt{f}_i : D^{ar(\mathtt{f}_i} \rightarrow D$ für $\mathtt{f}_i \in Fun_{PL}$
  - $I(\mathtt{R}_i \subseteq D^{ar(\mathtt{R}_i} \rightarrow D$ für $\mathtt{R}_i \in Rel_{PL}$

  *Beispiel:* Mit $ar(\mathtt{f}) = 2$ und $ar(\mathtt{R}) = 2$:

  - $D = \mathbb{N}_0$
  - $I(\mathtt{c}) = 0$
  - $I(\mathtt{f}) : {\mathbb{N}_0}^2 \rightarrow \mathbb{N}_0 : (x, y) \mapsto x + y$
  - $I(\mathtt{R}) = \{(x, y) \mid x \leq y\} \subseteq {\mathbb{N}_0}^2$

- Außerdem gibt es noch die Variablenbelegung $\beta : Var_{PL} \rightarrow D$
  *Beispiel:* $\beta(\mathtt{x}) = 3$ und $\beta(\mathtt{y}) = 42$

- Für $\beta$ gibt es noch eine besondere Notation:
  Für $\beta : Var_{PL} \rightarrow D, \mathtt{x}_i \in Var_{PL}$ und $d \in D$ sei;

$$\beta^d_{\mathtt{x}_i} : Var_{PL} \rightarrow D : \mathtt{x}_j \mapsto \begin{cases} \beta(\mathtt{x}_j), & \text{falls } j \neq i \\ d, & \text{falls } j = i \end{cases}$$

## 12.6 Auswertung von prädikatenlogischen Formeln

### 12.6.1 Ein Wert aus D für jeden Term

- Seien die Alphabete $Const_{PL}$, $Fun_{PL}$, $Rel_{PL}$, sowie die Interpretation $(D, I)$ und die Variablenbelegung $\beta$ gegeben.

- Wir definieren $\mathrm{val}_{D,I\beta}$: $L_{Ter} \cup L_{For} \rightarrow D \cup \mathbb{B}$
  Für jeden Term $t$ : $\mathrm{val}_{D,I\beta}$(t)$\in D$
  Für jede Formel $G$: $\mathrm{val}_{D,I\beta}$(G)$\in \mathbb{B}$

- *Beispiel:*

  - $D = \mathbb{N}_0, I(\mathtt{c}) = 0, I(f) =$Addition
  - $\beta(\mathtt{x}) = 3$ und $\beta(\mathtt{y}) = 42$
  - $\mathrm{val}_{D,I\beta}(\mathtt{f(y,c)})$
    $= I(\mathtt{f})(\mathrm{val}_{D,I\beta}(\mathtt{y}), \mathrm{val}_{D,I\beta}(\mathtt{c}))$
    $= I(\mathtt{f})(\beta(\mathtt{y}), \beta(\mathtt{c}))$
    $= I(\mathtt{f})(42, 0)$
    $= 42 + 0 = 42$

### 12.6.2 Wahrheitswerte für atomare Formeln

- Gegeben sei die Interpretation $(D, I)$, sowie die Variablenbelegung $\beta$
- Für eine atomare Formel $A \in L_{Rel}$ gibt es zwei Möglichkeiten:
  $\text{val}_{D,I\beta}(\texttt{R}_i(t_1, \ldots, t_k)) = \begin{cases} w, & \text{falls } (val_{D,I,\beta}(t_1), \ldots, val_{D,I,\beta}(t_k)) \in I(\texttt{R}_i) \\ f, & \text{falls } (val_{D,I,\beta}(t_1), \ldots, val_{D,I,\beta}(t_k)) \notin I(\texttt{R}_i) \end{cases}$

### 12.6.3 Wahrheitswert für quantifizierte Formeln

- Gegeben sei die Interpretation $(D, I)$, sowie die Variablenbelegung $\beta$
- $\text{val}_{D,I\beta}(\forall \texttt{x}_i H) = \begin{cases} w, & \text{falls für jedes } d \in D \text{ und } \beta' = \beta^d_{\texttt{x}_i} \text{ gilt: } \text{val}_{D,I,\beta'}(H) = w \\ f, & \text{sonst} \end{cases}$
- $\text{val}_{D,I\beta}(\exists \texttt{x}_i H) = \begin{cases} w, & \text{falls für mindestens ein } d \in D \text{ und } \beta' = \beta^d_{\texttt{x}_i} \text{ gilt: } \text{val}_{D,I,\beta'}(H) = w \\ f, & \text{sonst} \end{cases}$

## 12.7 Allgemeingültige Formeln

- Eine prädikatenlogische Formel $F$ ist allgemeingültig, wenn:
  für jede (passende) Interpretation $(D, I)$ und jede passende Variablenbelegung $\beta$ gilt:
  $\text{val}_{D,I\beta}(\text{F}) = w$

## 12.8 Modelle prädikatenlogischer Formeln

- $(D, I)$ ist Modell für $G \in L_{For}$, wenn:
  $(D, I)$ Interpretation von $G$ und für jedes $\beta$ $\text{val}_{D,I\beta}(G) = w$ ist.
- $(D, I)$ ist Modell für $\Gamma \subseteq L_{For}$, wenn:
  $(D, I)$ Modell für jedes $G \in \Gamma$ ist.
- $\Gamma \models G$, wenn jedes Modell von $\Gamma$ auch Modell von $G$ ist.
- $H \models G$, wenn $\{H\} \models G$
- $\models G$, wenn $\{\} \models G$, also wenn $G$ allgemeingültig ist.

## 12.9 Vorkommen von Variablensymbolen

- Wir betrachten nur die Vorkommen in Termen, nicht die Symbole unmittelbar hinter Quantoren
- Die gleiche Variable kann an mehreren Stellen vorkommen
- Wir unterscheiden zwischen freien und gebundenen Vorkommen
- Definitionen:
  Die Menge $\text{fv}(G)$ der frei vorkommenden Variablen.
  Die Menge $\text{bv}(G)$ der gebunden vorkommenden Variablen.

Sei $G$ eine atomare Formel

- alle Vorkommen von Variablen sind frei
- $bv(G) = \{\}$
- $fv(G) = \{\text{alle in } G \text{ vorkommende Variablen}\}$

Sei $G = \neg H$

- $bv(G) = bv(H)$ und $fv(G) = fv(H)$

Sei $G \in \{H_1 \wedge H_2,\ H_1 \vee H_2,\ H_1 \rightarrow H_2\}$

- $bv(G) = bv(H_1) \cup bv(H_2)$ und $fv(G) = fv(H_1) \cup fv(H_2)$
- Freies Vorkommen in $H_i$ ist freies Vorkommen in $G$
- Gebundenes Vorkommen in $H_i$ ist gebundenes Vorkommen in $G$

Sei $G$ von der Form $(\forall \mathtt{x}_i H)$ oder $(\exists \mathtt{x}_i H)$

- $bv(G) = \begin{cases} bv(H) \cup \{\mathtt{x}_i\}, & \text{falls } \mathtt{x}_i \in fv(H) \\ bv(H), & \text{sonst} \end{cases}$
- $fv(G) = fv(H) \setminus \{\mathtt{x}_i\}$
- in $G$ sind alle Vorkommen von $\mathtt{x}_i$ gebunden
- Alle anderen Variablen sind wie in H
- Die Formel $G$ ist geschlossen, wenn $fv(G) = \{\}$

## 12.10 Substitutionen

- Einie Substitution ist eine Abbildung $\sigma : Var_{PL} \rightarrow L_{Ter}$
- Falls $\sigma$ durch Menge $S$ der Paare $S = \{\mathtt{x}_{i_j}/\sigma(\mathtt{x}_{i_j} \mid 1 \leq j \leq k\}$ eindeutig bestimmt ist, schreibe:
  $\sigma_S = \sigma_{\{}\mathtt{x}_{i_j}/\sigma(\mathtt{x}_{i_j}) \mid 1 \leq j \leq k\}$
- *Beispiel:* $\sigma_{\{}\mathtt{x/c,\ y/f(x)}\}$ ist eine Abbildung mit:
  $\sigma(\mathtt{x}) = \mathtt{c}$
  $\sigma(\mathtt{y}) = \mathtt{f(x)}$
  $\sigma(z) = z$ für alle $z \notin \{\mathtt{x}, \mathtt{y}\}$
- Analog gilt diese Definition auch für Formeln.
  Hierbei werden alle **freien** Variablenvorkommen substituiert (und zwar gleichzeitig)

# 13 Kapitel 14 - Algorithmen

## 13.1 Informelle Definition des Algorithmusbegriffs

- Endliche Beschreibung (man schreibt nur endlich viel Algorithmus, der immer gilt)
- Elementare Anweisungen (jedem, mit der Materie vertrauten, ist klar, was gemeint ist)
- Determinismus (die nächste Anweisung ist stets eindeutig festgelegt)
- Endliche Eingabe $\longrightarrow$ endliche Ausgabe
- Endlich viele Schritte
- Beliebig große Eingaben sind (theoretisch) bearbeitbar
- Nachvollziehbarkeit / Verständlichkeit
  Diese Definition ist sehr informell, gibt aber einen ersten Eindruck

## 13.2 Beweisbarkeit von Algorithmen - Hoare-Kalkül / Hoare-Tripel

### 13.2.1 Definition von Hoare-Tripeln

- $\{P\}S\{Q\}$
  - $S$: Programmstück
  - $P$: Vorbedingung
  - $Q$: Nachbedingung
- $P, Q$ sind Zusicherungen
  - prädikatenlogische Formeln
  - frei vorkommende Variablen, die für das Programm benötigt werden
- Wahrheit von Zusicherungen abhängig von Interpretation und Variablenbelegung
- «relevante» Interpretation: nur *manche* interessieren
  - Grundbereich $D$ [immer fest und explizit definiert]
  - Funktions- und Relationssymbole [naheliegend interpretiert]
  - Konstantensymbole beliebig interpretierbar in $D$
    - Für *alle* Interpretationen sollen Zusicherungen wahr sein
    - «Eingaben» für das Programm
  - Variablenbelegungen
    - beschreiben den Zustand des *Speichers*
    - Veränderung durch Zuweisungen
    - Zuweisung: $x \leftarrow E$ für den Term $E$
      vorher Variablenbelegung $\beta$, hinterher $\beta' = \beta_x^{val_{D,I,\beta}(E)}$
      Es wird also nur der Wert von $x$ verändert.

### 13.2.2 Gültigkeit von Hoare-Tripeln

- $\{P\}S\{Q\}$ gültig, wenn für jede relevante Interpretation $I$ und jede Variablenbelegung $\beta$ gilt:
  wenn $val_{D,I,\beta}(P) = w$ und S endet und hinterher $\beta'$ vorliegt,
  dann $val_{D,I,\beta'}(Q) = w$

### 13.2.3 Axiome und Ableitungsregeln für Hoare-Tripel

Zuweisungsaxiom: Wenn

- Zuweisung $x \leftarrow E$ mit $E \in L_{Ter}$
- $Q$ Nachbedingung zu $x \leftarrow E$
- (Die Substitution $\sigma_{\{}x/E\}$ ist kollisionsfrei für $Q$)

$\Rightarrow$ Dann gilt HT-A: $\{\sigma_{\{x/E\}}(Q)\}\ \ x \leftarrow E\ \ \{Q\}$

- Alternative Schreibweise: $\{Q[x/E]\}\ \ x \leftarrow E\ \ \{Q\}$

Ableitungsregel für Verstärkung der Vorbedingung und Abschwächung der Nachbedingung:

- Wenn $P' \to P$ und $Q \to Q'$,
- Dann HT-E: $\frac{\{P\}S\{Q\}}{\{P'\}S\{Q'\}}$

Ableitungsregel für Hintereinanderausführung

- HT-S: $\frac{\{P\}S_1\{Q\} \quad \{Q\}S_2\{R\}}{\{P\}S_1;S_2\{R\}}$

Regel für bedingte Anweisungen

- HT-I: $\frac{\{P \wedge B\}S_1\{Q\} \quad \{P \wedge \neg B\}S_2\{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{fi}\ \{Q\}}$

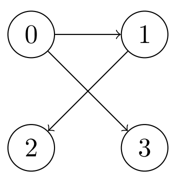
Regel für while - Schleifen

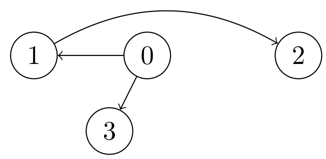
- HT-W: $\frac{\{I \wedge B\}S\{I\}}{\{I\}\ \texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}\ \{I \wedge \neg B\}}$
- $I$ ist die **Schleifeninvariante**, sie gilt vor der Schleife, nach jedem Schleifendurchlauf und nach dem Ende der Schleife.
- $B$ ist die Schleifenbedingung, die ist nach der Schleife ungültig.

# 14 Graphen

## 14.1 Gerichtete Graphen

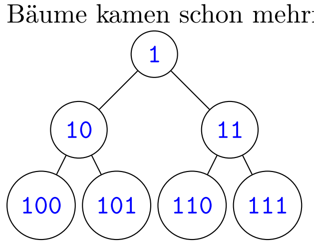
- Ein Gerichteter Graph ist ein Paar $G = (V, E)$
  - Knotenmenge $V$, endlich und nichtleer [9]
  - Kantenmenge $E \subseteq V \times V$ [10]
    Die Kantenmenge ist endlich, darf aber insbesondere auch leer sein.
- Das Aussehen eines Graphens kann sich unterscheiden, trotzdem handelt es sich noch um den selben Graphen.
- *Beispiel:*
  $V = \{0, 1, 2, 3\}$
  $E = \{(0, 1), (0, 3), (1, 2)\}$




### 14.1.1 Bäume

Bäume kamen schon mehrfach vor, daher hier nochmal zur Erinnerung:




### 14.1.2 Teilgraphen

- $G' = (V', E')$ ist ein Teilgraph von $G = (V, E)$, wenn
  - $V' \subseteq V$
  - $E' \subseteq E \cap V' \times V'$
    Endpunkte von Kanten in $E'$ müssen in $V'$ sein!

### 14.1.3 Pfade

- $V^{(+)}$: Menge der nichtleeren Listen von Elementen aus V
- $p = (v_0, \ldots, v_n) \in V^{(+)}$ ist Pfad, wenn $\forall i \in \mathbb{Z}_n : (v_i, v_{i+1}) \in E$
- Länge eines Pfades: Anzahl $n$ der **Kanten**!
- $v_n$ von $v_0$ erreichbar, wenn Pfad $p = (v_0, \ldots, v_n)$ existiert
- Länge 0 für Pfade erlaubt: $p = (v_0)$
- Durch streichen endlich vieler Knoten am Anfang und/oder am Ende erhält man einen Teilpfad.
  Ein Knoten muss mindestens übrig bleiben.

[9] Knoten engl. vertex
[10] Kante engl. edge

### 14.1.4 Zyklen

- Pfad mit $v_0 = v_n$ heißt geschlossen
- Ein geschlossener Pfad heißt Zyklus, wenn $n \geq 1$ ist.
- Pfad $(v_0, \ldots, v_n)$ heißt wiederholungsfrei, wenn
  - Die Knoten $v_0, \ldots, v_{n-1}$ und $v_1, \ldots, v_n$ paarweise verschieden sind
  - $v_0$ und $v_n$ dürfen gleich sein.
- Einfacher Zyklus: wiederholungsfreier Zyklus
- Azyklischer Graph: kein Teilgraph ist Zyklus

### 14.1.5 Strenger Zusammenhang

- Ein gerichteter Graph ist streng zusammenhängend, wenn $\forall x, y \in V : (x, y) \in V^2 \; \exists$ Pfad von x nach y

### 14.1.6 Knotengrad im gerichteten Graphen

- Eingangsgrad eines Knotens $y$ ist:
  $d^-(y) = |\{x \mid (x, y) \in E\}|$
- Ausgangsgrad eines Knotens $y$ ist:
  $d^+(x) = |\{y \mid (x, y) \in E\}|$
- Grad eines Knotens ist: $d(x) = d^-(x) + d^+(x)$

### 14.1.7 Gerichtete Bäume

- Es gibt eine Wurzel $r \in V$ mit der Eigenschaft:
  zu jedem Knoten $x$ existiert **genau** ein Pfad
- Man kann zeigen: Die Wurzel $r$ ist immer eindeutig
- Blätter sind Knoten mit Ausgangsgrad $= 0$
- innere Knoten sind Knoten mit Ausgangsgrad $> 0$

## 14.2 Graphen als Relationen

- $G = (V, E)$ mit $E \subseteq V \times V$ (E binäre Relation auf V)
- $(x, z) \in E^2 \iff$ Pfad $(x, y, z)$ existiert
- $(x, y) \in E^i \iff$ es existiert ein Pfad von $x$ nach $y$ der Länge $i$.

## 14.3 Ungerichtete Graphen

- Ein ungerichteter Graph ist eine Struktur $U = (V, E)$ mit
  - $V$: endliche nichtleere Menge von Knoten
  - $E$: Menge von Kanten mit $E \subseteq \{\{x, y\} \mid x \in V \land y \in V\}$
  - Es gibt keine Reihenfolge bei $\{x, y\}$
  - Beide Knoten gleichberechtigt
- adjezente Knoten sind durch eine Kante miteinander verbunden

- Bei einer Schlinge ist Start und Zielknoten identisch, es ergibt sich also:
  $\{x, y\}$ mit $x = y \Rightarrow \{x\}$

- Ein Graph ohne Schlinge heißt schlingenfrei

### 14.3.1 Teilgraph

Der Teilgraph des ungerichteten Graphen ist analog zum gerichteten Fall definiert.

### 14.3.2 Wege

- $V^{(+)}$: Menge der nichtleeren Listen von Elementen aus V

- $p = (v_0, \ldots, v_n) \in V^{(+)}$ ist Weg, wenn $\forall i \in \mathbb{Z}_n : \{v_i, v_{i+1}\} \in E$

- Länge eines Weges: Anzahl $n$ der **Kanten**!

- $v_n$ von $v_0$ erreichbar, wenn Weg $p = (v_0, \ldots, v_n)$ existiert

- Länge 0 für Wege erlaubt: $p = (v_0)$

### 14.3.3 Kreise

- geschlossener Weg oder Kreis
  $p = (v_0, \ldots, v_n)$ mit $v_0 = v_n$

- einfacher Kreis
  - wiederholungsfreier Kreis
  - mit mindestens drei verschiedenen Kanten

### 14.3.4 Zusammenhang

- Ein ungerichteter Graph $U = (V, E)$ heißt zusammenhängend, wenn der dazugehörige gerichtete Graph $G_U = (V, E_g)$ streng zusammenhängend ist.

### 14.3.5 Ungerichtete Bäume

- Ein ungerichteter Baum ist ein ungerichteter Graph $U = (V, E)$, zu dem ein gerichteter Baum $G = (V, E')$ existiert mit $E = E'_u$

- Verschiedene gerichtete Bäume induzieren den gleichen ungerichteten Baum

- Wurzeln
  - Im gerichteten Baum sind Wurzeln eindeutig zu identifizieren
  - Im ungerichteten Baum kann potentiell jeder Knoten Wurzel sein, diese muss daher explizit angegeben werden.

### 14.3.6 Knotengrad

- Grad $d(x)$ des Knotens $x \in V$ im ungerichteten Graphen:

$$d(x) = |\{y \mid y \neq x \wedge \{x, y\} \in E\}| + \begin{cases} 2, & \text{falls} \{x, x\} \in E \\ 0, & \text{sonst} \end{cases}$$

- Man zählt also die Anzahl der «Kantenenden», Schlingen werden doppelt gezählt.

# 15 Algorithmen in Graphen

## 15.1 Adjazenzmatrix

- Sein $G = (V, E)$ ein ungerichteter Graph mit $n = |V|$ Knoten.
- Dann ist die Adjazenzmatrix $A$ eine $n \times n$-Matrix mit:
  $a_{i,j} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 0 & \text{falls } (i,j) \notin E \end{cases}$
- Salopp gesagt steht überall eine 1, wenn man in **genau** einem Schritt von $i$ nach $j$ kommt. (Insbesondere auch bei Schlingen!)
- Analog gilt das auch für den ungerichteten Fall
- Wir wissen bereits, dass Graphen Relationen darstellen können, daher kann man Relationen auch als Matrix darstellen (auf die naheliegende Weise)
  - Sei $M$ endliche Menge mit $|M| = n$ (naheliegend durchnummeriert)
  - $R \subseteq M \times M$ binäre Relation
  - Entsprechende Matrix: $A(R)$ ($n \times n$ groß)
    $(A(R))_{i,j} = \begin{cases} 1 & \text{falls } (i,j) \in R, \text{ also } iRj \\ 0 & \text{falls } (i,j) \notin R, \text{ also } \neg(iRj) \end{cases}$

## 15.2 Signum-Fuktion

- sgn: $\mathbb{R} \to \mathbb{R} : sgn(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{falls } x = 0 \\ -1 & \text{falls } x < 0 \end{cases}$
- Bei Matrizen funktioniert dies komponentenweise.

## 15.3 Potenzen der Adjazenzmatrix

- Sei $G$ ein gerichteter Graph mit der Adjazenzmatrix $A$.
  Für alle $k \in \mathbb{N}_0$ gilt:
  $(A^k)_{i,j}$ ist die Anzahl der Pfade der Länge $k$ in $G$ von $i$ nach $j$.
- Es folgt: $sgn((A^k)_{i,j}) = \begin{cases} 1 & \text{falls in } G \text{ ein Pfad der Länge } k \text{ von } i \text{ nach } j \text{ existiert} \\ 0 & \text{falls in } G \text{ kein Pfad der Länge } k \text{ von } i \text{ nach } j \text{ existiert} \end{cases}$
- $sgn(A^k)$ repräsentiert also die Relation $E^k$

## 15.4 Wegematrix

- Die Wegematrix ist die Matrix $W$ der Erreichbarkeitsrelation $E*$.

$$W_{i,j} = \begin{cases} 1 & \text{falls } (i,j) \in E^* \\ 0 & \text{falls } (i,j) \notin E* \end{cases}$$
$$= \begin{cases} 1 & \text{falls es in } G \text{ einen Pfad von } i \text{ nach } j \text{ gibt} \\ 0 & \text{falls es in } G \text{ keinen Pfad von } i \text{ nach } j \text{ gibt} \end{cases}$$

- Um die Wegematrix effizient berechnen zu können, muss man aber $E^* = \bigcup_{i \in \mathbb{N}_0} E^i$ umschreiben

- Es folgt für $G = (V, E)$ mit $|V| = n \in \mathbb{N}_+$:

$$E^* = \bigcup_{i \in \mathbb{Z}_n} E^i = \bigcup_{i \in \mathbb{Z}_k} E^i \quad (\text{für } k \geq n)$$

- Sei $G$ ein gerichteter Graph, $A$ seine Adjazenzmatrix.
  Dann gilt für alle $k \geq n - 1$:
  - Die Matrix $sgn\left(\sum_{i=0}^{k} A^i\right)$ repräsentiert die Relation $E^*$
  - Es gilt also: $W = sgn\left(\sum_{i=0}^{k} A^i\right)$ ist die Wegematrix von $G$.
  - Dieser Algorithmus ist nicht der effizienteste.[11]

[11] In der Vorlesung wurde in Kapitel 16 und 17 sehr viel darauf rumgeritten, wie effizient dieser Algorithmus nun ist, oder eben nicht. Dann wurde auch noch der Algorithmus von Strassen eingeführt. Dies dient alles nur zur Vorbereitung auf die "Quantitativen Aspekte von Algorithmen", im Wesentlichen ist dies aber nicht besonders relevant.

# 16 Kapitel 17 - Quantitative Aspekte von Algorithmen

## 16.1 Groß-O-Notation

### 16.1.1 (Warum) keine exakten Angaben?

- Man will nicht
  - Faulheit
  - Vergänglichkeit der genauen Werte (bald neuer Prozessor)
  - Mangelndes Interesse an genauen Werten
- Man kann nicht
  - Unkenntnis von Randbedingungen
  - Ungenauigkeiten im «Algorithmus»[12]
- Man «soll» nicht
  - Unabhängigkeit von konkreter Probleminstanz
  - Unabhängigkeit von Programmiersprache
  - Unabhängig von Prozessor

### 16.1.2 Genauigkeit

- Ignorieren konstanter Faktoren (genaue Prozessorgeschwindigkeit gibt nur einen konstanten Vorfaktor)
- Nur obere (bzw. untere) Schranke angeben (z.B. nur schlechtesten Fall analysieren)

### 16.1.3 Asymptotisch gleichschnelles Wachstum

Notation:

- $\mathbb{R}_+$: Menge der positiven reellen Zahlen (ohne 0)
- $\mathbb{R}_0^+$: Menge der nichtnegativen reellen Zahlen $[\mathbb{R}_0^+ = \mathbb{R}_+ \cup \{0\}]$
- es werden nur Funktionen $f : \mathbb{N}_0 \to \mathbb{R}_0^+$ betrachtet

Definition:

- Die Funktion $g : \mathbb{N}_0 \to \mathbb{R}_0^+$ wächst asymptotisch genauso schnell wie $f : \mathbb{N}_0 \to \mathbb{R}_0^+$, wenn gilt:
  $$\exists c, c' \in \mathbb{R}_+ \ \exists n_0 \in \mathbb{N}_0 \ \forall n \geq n_0 \ : \ cf(n) \leq g(n) \leq c'f(n)$$
- schreibe: $f \asymp g$ oder $f(n) \asymp g(n)$
- *Beispiel:*
  - $f : n \mapsto 3n^2$ und $g : n \mapsto 10^{-2}n^2$
  - Behauptung: $f \asymp g$
  - Setze $c = 10^{-3}$ und $c' = 1$, sowie $m_0 = 0$:
    $cf(n) \leq g(n) \leq c'f(n)$
    $10^{-3} \cdot 3n^2 \leq 10^{-2}n^2 \leq 3n^2$
- Regel: $\forall f : \mathbb{N}_0 \to \mathbb{R}_0^+ \ \forall a, b \in \mathbb{R}_+ : \ af \asymp bf$

[12] Wobei es diese ja eigentlich bei einem deterministischen Algorithmus nach unserer Definition gar nicht geben darf

### 16.1.4 Äquivalenzrelation $\asymp$

:

- $\asymp$ ist eine Äquivalenzrelation, d.h.
  $\asymp$ ist
  - reflexiv ($f \asymp f$ gilt immer)
  - transitiv ($(f \asymp g \land g \asymp h) \iff f \asymp h$)
  - symmetrisch ($f \asymp g \iff g \asymp f$)

### 16.1.5 Groß $\Theta$

- $\Theta(f)$: Menge aller Funktionen, die (im Sinne von $\asymp$) zu $f$ äquivalent sind.
- $\Theta(f) = \{g \mid f \asymp g\}$
- Rechenregel: $\forall\ f : \mathbb{N}_0 \to \mathbb{R}_0^+\ \forall a, b \in \mathbb{R}_+ :\ \Theta(af) = \Theta(bf)$

### 16.1.6 Obere Schranke O

- $O(f) = \{g \mid \exists c \in \mathbb{R}_+\ \exists n_0 \in \mathbb{N}_0\ \forall n \geq n_0 :\ g(n) \leq cf(n)\}$
- Schreibweisen:
  $g \preceq f \iff g \in O(f)$

### 16.1.7 Untere Schranke $\Omega$

- $\Omega(f) = \{g \mid \exists c \in \mathbb{R}_+\ \exists n_0 \in \mathbb{N}_0\ \forall n \geq n_0 :\ g(n) \geq cf(n)\}$
- Schreibweisen:
  $g \succeq f \iff g \in \Omega(f)$

### 16.1.8 Rechenregeln für obere und untere Schranken

- Für alle Funktionen $f, g : \mathbb{N}_0 \to \mathbb{R}_0^+$ gilt:
  $g \in O(f) \iff f \in \Omega(g)$, also $g \preceq f \iff f \succeq g$
- $\Theta(f) = O(f) \cap \Omega(f)$, also:
  $g \asymp f \iff (g \preceq f \land g \succeq f)$
- Es gilt:
  Wenn $g_1 \preceq f_1$ und $g_2 \preceq f_2$, dann $g_1 + g_2 \preceq f_1 + f_2$
- $\forall f_1, f_2 : \mathbb{N}_0 \to \mathbb{R}_0^+$ gilt:
  $O(f_1) + O(f_2) = O(f1 + f2)$
- *Beispiel:* $O(n^3 + n^2) = O(n^3)$

### 16.1.9 Komplexoperationen

- Sind $M_1$ und $M_{;2}$ Mengen, deren Elemente man addieren bzw. multiplizieren kann, dann sei:

$$M_1 + M_2 = \{g_1 + g_2 \mid g_1 \in M_1 \land g_2 \in M_2\}$$
$$M_1 \cdot M_2 = \{g_1\ \cdot\ g_2 \mid g_1 \in M_1 \land g_2 \in M_2\}$$

- Diese Definition ist analog zum Produkt formaler Sprachen
- Es gilt außerdem:
  statt $\{n^3\} + O(n^2)$ schreibt man $n^3 + O(n^2)$

## 16.2 Teile und herrsche / divide and conquer

### 16.2.1 Definition divide and conquer

- Probleminstanz wird in kleine Teile zerlegt
- Diese Teile werden Rekursiv nach gleichem Verfahren bearbeitet
- Gesamtergebnis wird aus den Teilergebnissen konstruiert

### 16.2.2 Laufzeit von Teile-und-Herrsche-Algorithmen

- Problem der Größe $n$ wird zerhackt in
  konstante Anzahl a von Teilproblem gleicher Größe $\frac{n}{b}$
- Aus diesen Teilergebnissen wird Gesamtergebnis zusammengesetzt
- Aufteilen und Zusammensetzen «kostet» $f(n)$
- Abschätzen der Laufzeit $T(n)$ liefert ungefähr:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

### 16.2.3 Master-Theorem

- Fall 1: Wenn $f(n) \in O(n^{(log_b a)-\varepsilon})$ $(\varepsilon > 0)$, dann $T(n) \in \Theta(n^{log_b a})$
- Fall 2: Wenn $f(n) \in O(n^{(log_b a)})$, dann $T(n) \in \Theta(n^{log_b a} log\, n)$
- Fall 3: Wenn $f(n) \in O(n^{(log_b a)+\varepsilon})$ $(\varepsilon > 0)$, dann $T(n) \in \Theta(f(n))$,
  falls zusätzlich auch $\exists\, 0 < d < 1 :$ $af(\frac{n}{b}) \leq df(n)$ gilt.
- Diese Fallunterscheidung ist nicht vollständig!

# 17 Kapitel 18 - Endliche Automaten

## 17.1 Moore-Automat

Im WS2020/21 werden nur Moore-Automaten betrachtet. Mealy-Automaten wurden weggelassen.

### 17.1.1 Bestandteile/Definition Moore-Automat

- endliche Zustandsmenge $Z$
  Alle Zustände des Automaten
- Anfangszustand $z_0 \in Z$
  Der Zustand, indem die «Ausführung» beginnt.
- Eingabealphabet $X$
  Das Alphabet der eingegebenen Wörter
- Zustandüberführungsfunktion $f : Z \times X \to Z$
  Gibt an, welcher Zustand kommt, nachdem in dem gegebenen Zustand das Eingegebene Wort eingegeben wurde.
- Ausgabealphabet Y
  Alphabet der ausgegebenen Wörter.
- Ausgabefunktion $h : Z \to Y^*$
  Gibt an, welches Wort der angegebene Zustand ausgibt.

### 17.1.2 Erweiterung Zustandsüberführungsfunktion

- $f_*$ für den am Ende der Ausführung erreichten Zustand
  $f_* : Z \times X^* \to Z$ mit
  $f_*(z, \varepsilon) = z$ und $f_*(z, wx) = f(f_*(z, w), x) \quad (w \in X^*, x \in X)$
- $f_{**}$ für alle durchlaufenen Zustände
  $f_{**} : Z \times X^* \to Z^*$ mit
  $f_{**}(z, \varepsilon) = z$ und $f_{**}(z, wx) = f_{**}(z, w) \cdot f_*(z, wx) \quad (w \in X^*, x \in X)$

### 17.1.3 Erweiterung Ausgabefunktion

- «letzte Ausgabe»: $g_* = h \circ f_*$:
  $g_*(z, w) = h(f_*(z, w))$
- «alle Ausgaben»: $g_{**} = h^{**} \circ f_{**}$ (induzierter Homomorphismus):
  $g_{**}(z, w) = h^{**}(f_{**}(z, w))$
  *Es werden alle Ausgaben von Beginn bis Ende hintereinander konkatiniert*

## 17.2 Endliche Akzeptoren

- Immer genau ein Bit als Ausgabe:
  $\forall z \in Z : h(z) \in Y = \{\mathtt{0}, \mathtt{1}\}$
  Interpretiert wird `1` als gut/korrekt und `0` als schlecht/falsch

### 17.2.1 Formalisierung

- Spezifikation einer Menge $F$: Akzeptierende Zustände
- $F = \{z \in Z \mid h(z) = \mathtt{1}\}$
- Alle $z \in (Z \setminus F)$ sind ablehnende Zustände
- ablehnende Zustände werden im Schaubild mit einem Kreis gezeichnet, akzeptierende Zustände bekommen einen «Doppelkringel»

### 17.2.2 Akzeptiert und Ablehnen

- Ein Wort $w \in X^*$ wird akzeptiert, falls $f_*(z_0, w) \in F$
- Ein Wort $w \in X^*$ wird abgelehnt, falls $f_*(z_0, w) \notin F$
- Die von einem Akzeptor $A = (Z, z_0, X, f, F)$ akzeptierte oder erkannte formale Sprache ist $L(A) = \{w \in X^* \mid f_*(z_0, w) \in F\}$
- Es handelt sich um ganz einfache Syntaxanalyse.

### 17.2.3 Nicht erkennbare Formale Sprachen

*Beispiel:* Die formale Sprache $L = \{\mathtt{a}^k\mathtt{b}^k \mid k \in \mathbb{N}_0\}$ kann nicht von einem endlichen Akzeptor erkannt werden.
Beweisansatz: Wähle $k =$Anzahl der Zustände des Akzeptors.
Dann **muss** der Akzeptor irgendwo «im Kreis laufen», kann also nicht mehr mitzählen, wie oft er bereits im Kreis gelaufen ist. Dadurch lassen sich Worte konstruieren, die fälschlicher Weise als korrekt angenommen werden.

## 18 Kapitel 20 - Turing-Maschine

TODO: kommt, sobald das Thema in der Vorlesung vollständig behandelt wurde.