

Wrap-up

Grundzüge digitaler Systeme

Vortrag von: Stefan Neumann, Wolfgang Dvořák

Zahlendarstellung

Natürliche Zahlen

Verschiedene **Stellenwertsysteme**:

- **Dezimalzahlen**: unser gewohntes Zahlensystem
- **Binärzahlen**: Ideal für die Informatik – können direkt durch elektrische Zustände (an/aus) dargestellt werden
- **Hexadezimalzahlen**: kompakte Darstellung von Binärzahlen/Binärcodes

binär	hexadezimal	dezimal
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9

binär	hexadezimal	dezimal
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15
10000	10	16
10001	11	17
10010	12	18
10011	13	19

Zahldarstellung

Negative Ganzzahlen

Darstellung negativer Zahlen

- **Vorzeichenbit**: simpelste Variante
- **Zweierkomplement**: ermöglicht die Subtraktion mittels Addition
- **Exzessdarstellung**: erhält die Ordnung der Zahlen
Verwendung bei der Codierung des Exponenten von Gleitkommazahlen

$$\begin{array}{lcl} -(502)_{10} = & -(111110110)_2 & \left. \begin{array}{l} \text{mit 0en auffüllen und} \\ \text{Vorzeichenbit 1 (weil neg. Z.)} \end{array} \right\} \\ \text{Vorzeichen \& Betrag: } & (100111110110)_2 & \left. \begin{array}{l} \text{Bits invertieren} \\ \text{1 dazu zählen} \end{array} \right\} \\ \text{Einerkomplement: } & (111000001001)_2 & \\ \text{Zweierkomplement: } & (111000001010)_2 & \left. \begin{array}{l} \text{VZ-Bit invertieren} \end{array} \right\} \\ \text{Exzessdarstellung: } & (011000001010)_2 & \end{array}$$

Zahlendarstellung

Überlange Zahlen

Bei manchen Anwendungen ist es erforderlich, mit extrem langen ganzen Zahlen zu rechnen. Der Zahlenbereich von den von der Hardware unterstützten 32bit / 64 bit Datentypen (Java: int, long) reicht nicht aus.

- Kryptographie
- Berechnung mathematischer Konstanten (z.B. Kreiszahl π)

Wir sprechen von **überlangen Zahlen** oder Langzahlen.

Überlange Zahlen

Wir speichern überlange Zahlen als Liste von Ziffern. Die Ziffern sind so gewählt, dass wir die von Hardware unterstützten Rechenoperationen nutzen können.

Bsp.: Python verwendet 30 Bit pro Ziffer. Also ein Stellenwertsystem mit Basis $b = 2^{30}$.

Überlange Zahlen in Java: `math.BigInteger` bzw. `math.BigDecimal`

Zahlendarstellung

Langzahlarithmetik: Rechnen mit überlangen Zahlen

- Manchmal ist es erforderlich, mit extrem langen ganzen Zahlen zu rechnen.
- Überlegungen in Richtung Rechenzeitabschätzung notwendig.

Wir verwenden der Einfachheit halber die üblichen dezimalen Ziffern $\{0, 1, \dots, 9\}$ und nehmen an, dass unsere Hardware diese Ziffern direkt addieren und multiplizieren kann.

Addition von überlangen Zahlen:

- Zeit direkt proportional zur Länge der Summanden.
- Sei k die Länge der Summanden.
- Rechenzeit:

$$C_1 \cdot k,$$

C_1 rechnerabhängige Konstante

Klassische Multiplikation:

- Sei k die Länge der Faktoren.
- Jede Ziffer des zweiten Faktors wird mit jeder Ziffer des ersten Faktors verknüpft.

⇒ Rechenzeit: $C_2 \cdot k^2$,

C_2 rechnerabhängige Konstante.

Multiplikationsalgorithmen, die weniger Zeit verbrauchen

- Unter Umständen erst dann schneller, wenn k relativ groß

Annahmen:

- k -stellige Zahlen sind im Rechner ohne Probleme darstellbar
- Rechner verwendet Dezimalzahlen
- Stellen Zahlen der Länge $2k$ als Paare von k -stelligen Zahlen dar.
- Seien (a, b) und (c, d) solche Paare. Definieren

$$u = (a, b) = a \cdot 10^k + b$$

und

$$v = (c, d) = c \cdot 10^k + d.$$

Zahlendarstellung

Langzahlarithmetik: Multiplikation

Produkt von u und v

$$\begin{aligned}w &= u \cdot v = (a \cdot 10^k + b) \cdot (c \cdot 10^k + d) \\&= a \cdot c \cdot 10^{2k} + (a \cdot d + b \cdot c) \cdot 10^k + b \cdot d.\end{aligned}$$

- Vier Multiplikationen von k -stelligen Zahlen
- Trickreiche Umformung \Rightarrow weniger k -stellige Multiplikationen

$$w = u \cdot v = a \cdot c \cdot (10^{2k} + 10^k) + (a - b) \cdot (d - c) \cdot 10^k + b \cdot d \cdot (10^k + 1).$$

- **Drei** Multiplikationen zweier k -stelliger Zahlen wegen

$$(a - b) \cdot (d - c) \cdot 10^k = (a \cdot d - b \cdot d - a \cdot c + b \cdot c) \cdot 10^k$$

Zahlendarstellung

Langzahlarithmetik: Multiplikation

Annahme k ist eine Zweierpotenz, d.h. $k = 2^i$.

- k -stellige Zahlen durch Paare von $k/2$ -stelligen Zahlen multiplizieren
- $k/2$ -stellige Zahlen durch Paare von $k/4$ -stelligen Zahlen multiplizieren
- Usw.
- **Rekursion:**
 - So lange fortführen, bis Anzahl der Stellen im Rechner darstellbar

Zahlendarstellung

Langzahlarithmetik: Multiplikation

Annahme: $T(k)$... Zeit, um zwei k -stellige Zahlen zu multiplizieren

$$T(2k) \leq 3 \cdot T(k) + c \cdot k,$$

- $3 \cdot T(k)$... Aufwand für drei Multiplikationen
- $c \cdot k$... Aufwand für die notwendigen Additionen
- Vollständige Induktion:

$$T(2^i) \leq c' \cdot (3^i - 2^i), \quad \text{für } i \geq 1.$$

(wählen c' so groß, dass die Ungleichung auch für $i = 1$ gilt)

$$T(k) = T(2^i) \leq c' \cdot (3^i - 2^i) < c' \cdot 3^i.$$

Zahlendarstellung

Langzahlarithmetik: Multiplikation

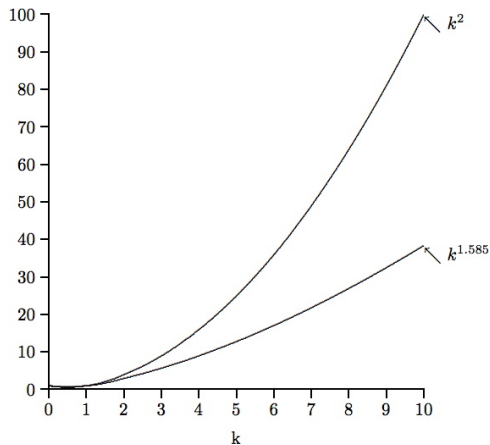
■ $i = \text{ld } k$ ($\text{ld} \dots$ Logarithmus zur Basis 2)

■ $3^{(\text{ld } k)} = 2^{(\text{ld } 3)(\text{ld } k)} = k^{(\text{ld } 3)}$

$$\Rightarrow T(k) < c' \cdot k^{(\text{ld } 3)} \approx c' \cdot k^{1.585}$$

Zahlendarstellung

Langzahlarithmetik: Multiplikation



$k = 10$ Verbesserung um einen Faktor von mehr als 2 ($10^{1.585} \approx 38.46 < 10^2 = 100$)

- Gewinn an Laufzeit kann durch die nicht genau angegebene Konstante c' verdeckt werden.
- Für große Werte von k jedoch beginnt sich früher oder später das geringere Wachstum von $k^{1.585}$ positiv bemerkbar zu machen.

Zahlendarstellung

Langzahlarithmetik: Multiplikation

- Aktuell schnellster Algorithmus zur Multiplikation ¹

$$T(k) \leq c \cdot n \cdot \log(n),$$

- Hypothese: Es gibt keinen Algorithmus, der eine bessere asymptotische Laufzeit als

$$d \cdot n \cdot \log(n)$$

hat.

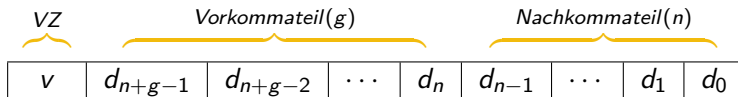
¹Harvey, David; van der Hoeven, Joris (2021). Integer multiplication in time $O(n \log n)$. Annals of Mathematics. Second Series. 193 (2): 563–617. doi: 10.4007/annals.2021.193.2.4.

Zahldarstellung

Kommazahlen

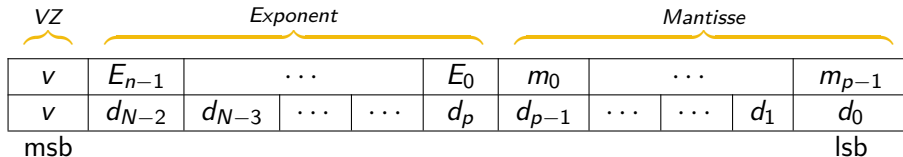
Wir müssen uns damit abfinden, dass reelle Zahlen im Rechner nur mit einer gewissen Genauigkeit dargestellt werden können.

■ Festpunkt-Darstellung



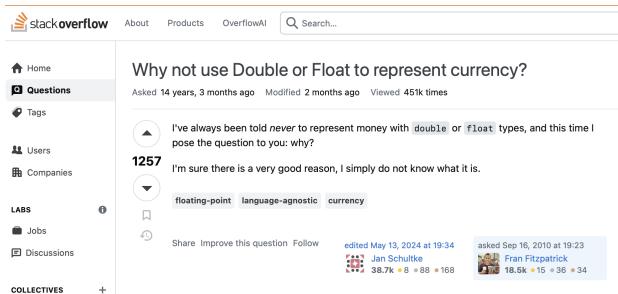
■ Gleitkomma

- Für Anwendungen mit einer große Zahlendynamik: Sehr kleine und sehr große Zahlen werden einheitlich dargestellt



Kann ich Zahlendarstellung nach der Prüfung wieder vergessen?

- Besser nicht
- Wichtig in allen Bereichen, in denen numerisch gearbeitet wird
 - Machine Learning / AI (mehr gleich)
 - Computerspiele und Visual Computing²
 - Banken und viele mehr³



²<https://www.gamedeveloper.com/programming/visualizing-floats>

³<https://stackoverflow.com/questions/3730019/why-not-use-double-or-float-to-represent-currency>

Zahlendarstellung und Large Language Models (LLMs)

- Um Large Language Models (LLMs) effizienter zu machen, gibt es viel Forschung zu Zahlendarstellung
- LLMs müssen große Matrizen speichern
- Klassisch wird jede Zahl der Matrix als Float mit 16 Bits gespeichert
 - Forschung, um mit weniger Genauigkeit auszukommen oder Genauigkeit gezielter einzusetzen
 - Reduziert Speicherplatz und erhöht Geschwindigkeit, verringert notwendige GPU-Ressourcen
 - Man spricht von sogenannter *Quantization*⁴

Quantized	Scalar	Vector	Matrix	Tensor
	1	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} [1 & 2] & [3 & 2] \\ [1 & 7] & [5 & 4] \end{bmatrix}$
High Precision	Scalar	Vector	Matrix	Tensor
	1.3	$\begin{bmatrix} 1.4 \\ 2.7 \end{bmatrix}$	$\begin{bmatrix} 1.8 & 2.2 \\ 3.7 & 4.2 \end{bmatrix}$	$\begin{bmatrix} [1.1 & 2.7] & [3.75 & 2.7] \\ [1.7 & 7.9] & [5.31 & 4.9] \end{bmatrix}$

⁴<https://www.tensorops.ai/post/what-are-quantized-llms>

Zahlendarstellung und Large Language Models (LLMs)

- Um Large Language Models (LLMs) effizienter zu machen, gibt es viel Forschung zu Zahlendarstellung
- LLMs müssen große Matrizen speichern
- Klassisch wird jede Zahl der Matrix als Float mit 16 Bits gespeichert
 - Forschung, um mit weniger Genauigkeit auszukommen oder Genauigkeit gezielter einzusetzen
 - Reduziert Speicherplatz und erhöht Geschwindigkeit, verringert notwendige GPU-Ressourcen
 - Man spricht von sogenannter *Quantization*⁴

Model	Original Size (FP16)	Quantized Size (INT4)
Llama2-7B	13.5 GB	3.9 GB
Llama2-13B	26.1 GB	7.3 GB
Llama2-70B	138 GB	40.7 GB

⁴<https://www.tensorops.ai/post/what-are-quantized-llms>

Zahlendarstellung und Large Language Models (LLMs)

- Um Large Language Models (LLMs) effizienter zu machen, gibt es viel Forschung zu Zahlendarstellung
- LLMs müssen große Matrizen speichern
- Klassisch wird jede Zahl der Matrix als Float mit 16 Bits gespeichert
 - Forschung, um mit weniger Genauigkeit auszukommen oder Genauigkeit gezielter einzusetzen
 - Reduziert Speicherplatz und erhöht Geschwindigkeit, verringert notwendige GPU-Ressourcen
 - Man spricht von sogenannter *Quantization*⁴

bf16: Brain Floating Point Format

Range: $\sim 1e^{-38}$ to $\sim 3e^{38}$



fp32: Single-precision IEEE Floating Point Format

Range: $\sim 1e^{-38}$ to $\sim 3e^{38}$

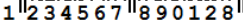


fp16: Half-precision IEEE Floating Point Format

Range: $\sim 5.96e^{-8}$ to 65504



⁴<https://www.tensorops.ai/post/what-are-quantized-llms>



Wir haben unterschiedliche Ziele betrachtet.

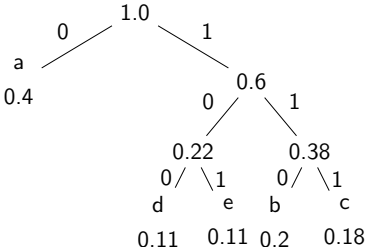
■ Fehlererkennung und Fehlerkorrektur

- Prüfstellencodes
- Polynomcodes
- Hamming-Code

■ Datenverdichtung

- ## ■ Huffman-Code

Quelle: Wikipedia



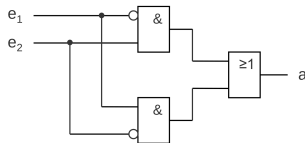
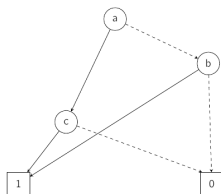
Darstellung Boolescher Funktionen

Unterschiedlichen Arten Boolesche Funktionen zu repräsentieren

- **Wahrheitstabellen:** bei vielen Variablen sehr groß
- **Boolesche Formeln:** Vereinfachung umständlich
- **If-Then-Else (ITE) Ausdrücke:** Geeignet zur Implementierung in Software
- **Binäre Entscheidungsdiagramme** (Binary Decision Diagrams, BDDs): Kompakte Darstellung, Operationen können direkt ausgeführt werden
- **Digitalschaltungen:** Abbildung Boolescher Algebra auf physikalische Systeme

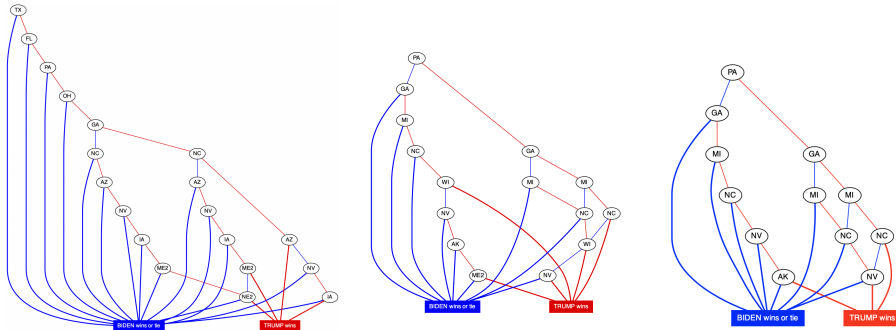
x	y	$\neg x$	$\neg y$	$\neg x \wedge \neg y$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

```
if (a = true) then
  if (b = true) then
    false
  else
    true
else
  true
```



Betrachtung von Wahlentscheidungen mittels OBDDs

- Stefan Szeider hat OBDDs genutzt, um mögliche Ausgänge von US-Wahlen darzustellen⁵
 - Das BDD vereinfacht sich, wenn einzelne US-Staaten ihre Ergebnisse bekanntgeben



⁵ <https://www.ac.tuwien.ac.at/people/szeider/2020-us-presidential-election-as-an-obdd/>

Modellierung:

- Abstrakte, vereinfachte Darstellung eines komplexen Sachverhalts,
- die nur die für relevant erachteten Aspekte enthält.

Formale Modellierung / Spezifikation:

- Übersetzung unpräziser natürlichsprachiger Problemstellungen in unmissverständliche Beschreibungen in einer formalen Sprache

Kernaspekte von Modellierungssprachen

- Syntax: Was ist eine zulässige Äußerung in dieser Sprache?
- Semantik: Was bedeutet jede der zulässigen Äußerungen?
- Ausdrucksstärke: Was kann ausgedrückt werden, was nicht?
- Verwendung: Wie setzt man die Sprache zur Modellierung ein?

Es gibt in der Informatik viele **verschiedene logische Systeme**.

Zwei haben wir in der Vorlesung kennengelernt:

■ Aussagenlogik

- Logische Verknüpfung atomarer Aussagen
- Keine Berücksichtigung der inneren Struktur der atomaren Aussagen
- Eher ausdruckschwach

$$\neg(a \wedge b) \Leftrightarrow (\neg a \vee \neg b)$$

■ Prädikatenlogik

- Erweiterung der Aussagenlogik
- Prädikatensymbole, Funktionsterme
- Quantoren \forall, \exists
- Spricht über Objekte eines Universums statt über Aussagen

$$\forall x (Mensch(x) \Rightarrow Sterblich(x))$$

Charakteristik:

- gut für deklarativ-statische Probleme: Welche Eigenschaften sollen gelten?
- modular: Neue Bedingungen werden durch zusätzliche Formeln berücksichtigt.

Eine Schwäche von Prädikatenlogik ist, dass sie nicht über Gruppen/Mengen von Objekten quantifizieren kann.

Beispiel

Aussage: Für jede Gruppe von Freunden, gibt es eine Sprache die alle in der Gruppe sprechen.

Sind die möglichen Gruppen durch Prädikate $Gruppe_1(\cdot), \dots, Gruppe_n(\cdot)$ gegeben, können wir das wie folgt formulieren:

$$\left(\bigwedge_{i=1}^n \forall x \forall y ((Gruppe_i(x) \wedge Gruppe_i(y)) \rightarrow Freunde(x, y)) \right) \wedge \\ \bigwedge_{i=1}^n \exists y \forall x (Gruppe_i(x) \rightarrow Spricht(y, x))$$

Wir können aber die Allaussage „Für jede (mögliche) Gruppe“ nicht mit unserer Prädikatenlogik formalisieren.

Unsere Prädikatenlogik heißt auch **Prädikatenlogik erster Stufe** (engl. first-order logic)

↪ Es gibt auch Prädikatenlogiken höherer Stufen.

Prädikatenlogik zweiter Stufe:

- ermöglicht **Quantifizierung über Prädikate und Funktionen** erster Stufe.
- $\forall P^1 \exists x P(x)$: Für alle 1-stelligen Prädikate P gibt es ein Objekt x , sodass $P(x)$ wahr ist.
- $\exists P^2 \forall x P(x, x)$: Es gibt ein 2-stelliges Prädikate P , das reflexiv ist.

Beispiel

Aussage: Für jede Gruppe von Freunden, gibt es eine Sprache die alle in der Gruppe sprechen.

Die Aussage kann jetzt formalisiert werden:

$$\forall \text{Gruppe}^1 \left(\forall x \forall y ((\text{Gruppe}(x) \wedge \text{Gruppe}(y)) \rightarrow \text{Freunde}(x, y)) \rightarrow \right. \\ \left. \exists y (\text{Sprache}(y) \wedge \forall x (\text{Gruppe}(x) \rightarrow \text{Spricht}(y, x))) \right)$$

Beispiel

Wir wollen die folgenden drei Aussagen formalisieren

- 1 Vögel können fliegen
- 2 Pinguine können nicht fliegen
- 3 Pinguine sind Vögel

Wir bekommen die folgenden prädikatenlogischen Regeln:

$$\forall x (Vogel(x) \Rightarrow KannFliegen(x))$$

$$\forall x (Pinguin(x) \Rightarrow \neg KannFliegen(x))$$

$$\forall x (Pinguin(x) \Rightarrow Vogel(x))$$

Wenn wir jetzt ein Objekt Tweety haben und $Pinguin(Tweety)$ gilt, bekommen wir

- sowohl $KannFliegen(Tweety)$
- als auch $\neg KannFliegen(Tweety)$.

Die vier Aussagen sind also nicht miteinander konsistent.

Beispiel

Die Aussage

1 Vögel können fliegen

würden wir eher als „Vögel können typischer Weise fliegen“ interpretieren. Die Formalisierung

$$\forall x (Vogel(x) \Rightarrow KannFliegen(x))$$

sagt aber, dass wirklich jeder Vogel fliegen kann.

„Typischer Weise“ können wir in Prädikatenlogik nicht ausdrücken.

Default Regeln:

Prämisse: Berechtigung
—————
Konklusion

Wenn die Prämisse gilt und die Berechtigung konsistent mit dem aktuellen Stand ist, dann folgern wir die Konklusion.

Beispiel

Wir bekommen die folgenden Regeln:

$$\frac{Vogel(x) : KannFliegen(x)}{KannFliegen(x)}$$

$$\frac{Pinguin(x)}{\neg KannFliegen(x)}$$

$$\frac{Pinguin(x)}{Vogel(x)}$$

Aus $Pinguin(Tweety)$ können wir

- mit der zweiten Regel ableiten, dass $\neg KannFliegen(Tweety)$
- mit der dritten Regel ableiten, dass $Vogel(Tweety)$

Die erste Regel können wir nicht verwenden, da wir $\neg KannFliegen(Tweety)$ abgeleitet haben und $KannFliegen(Tweety)$ nicht mehr kompatibel ist.

Aus $Vogel(Caruso)$ können wir (wenn keine weitere Information für $Caruso$ existiert)

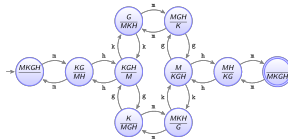
- mit der ersten Regel ableiten, dass $KannFliegen(Caruso)$

Endliche Automaten

Für das Modellieren dynamischer Probleme.

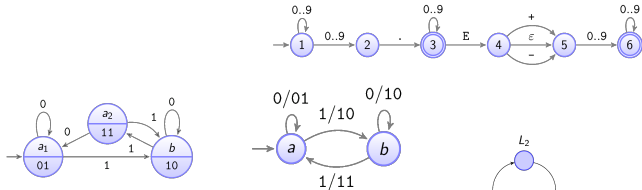
Automaten als Akzeptoren

- DEA: Deterministische Endliche Automaten
- NEA: Nichtdeterministische Endliche Automaten



Automaten mit Ein- und Ausgabe

- Transducer
- Moore Automat
- Mealy Automat

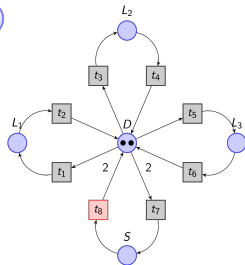


Automaten mit unendlichen Eingaben

- Büchi Automaten als DEA/NEA für unendliche Wörter
- Ähnlich können auch Moore & Mealy verallgemeinert werden

Grafische Modellierung verteilter Systeme

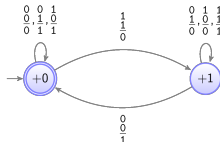
- Petri Netze



Sprachen

Unterschiedliche Arten formale Sprachen zu definieren:

- Induktive Definitionen
- Automaten
- Reguläre Ausdrücke
- Grammatiken
- Syntaxdiagramme



Arten **reguläre Sprachen** zu definieren:

- Reguläre Ausdrücke
- Endliche Automaten: DEA /NEA
- nicht-rekursive Syntaxdiagramme

Arten **kontextfreie Sprachen** zu definieren:

- Kontextfreie Grammatiken
- Nichtdeterministische Kellerautomaten
- (rekursive) Syntaxdiagramme

Reg. Sprache	Algebra	EBNF	Syntaxdiagramm	
A	A	A	$\rightarrow A \rightarrow$	Abkürzung
$\{\}$	\emptyset			Leersprache
$\{\varepsilon\}$	ε		\rightarrow	Leerwortsprache
$\{s\}$	s	"s"	$\rightarrow s \rightarrow$	Terminalsymbol
$X \cdot Y$	XY	XY	$\rightarrow X \rightarrow Y \rightarrow$	Aufeinanderfolge
$X \cup Y$	$X + Y$	$X Y$	$\rightarrow X \mid Y \rightarrow$	Alternativen
$\{\varepsilon\} \cup X$	$\varepsilon + X$	$[X]$	$\rightarrow X \mid \rightarrow$	Option
X^*	X^*	$\{X\}$	$\rightarrow X^* \rightarrow$	Wiederholung ≥ 0
X^+	XX^*, X^+	$X\{X\}$	$\rightarrow X^+ \rightarrow$	Wiederholung ≥ 1
(X)	(X)	(X)	$\rightarrow X \rightarrow$	Gruppierung

$G = \langle \{Fm, Op, Var\}, T, P, Fm \rangle$, wobei

$T = \{T, \perp, \neg, (,), \wedge, \uparrow, \vee, \downarrow, \Leftrightarrow, \Leftrightarrow, \Rightarrow, \Leftarrow\} \cup \mathcal{V}$

$P = \{ Fm \rightarrow Var \mid T \mid \perp \mid \neg Fm \mid (Fm Op Fm) ,$
 $Op \rightarrow \wedge \mid \uparrow \mid \vee \mid \downarrow \mid \Leftrightarrow \mid \Leftrightarrow \mid \Rightarrow \mid \Leftarrow ,$
 $Var \rightarrow A \mid B \mid C \mid \dots \}$

Ausdrucksstärke vs. Komplexität

Wir haben bei Logiken, Automaten, Sprachen jeweils unterschiedlich ausdrucksstarke Varianten gesehen.

- Wir streben einerseits eine **große Ausdrucksstärke** an, um viele interessante Problemstellungen modellieren zu können.
 - Mit Prädikatenlogik können wir Probleme modellieren, die sich mit Aussagenlogik nicht modellieren lassen.
 - Es gibt Sprachen, die nicht regulär sind, aber mit einer kontextfreien Grammatik beschrieben werden können.
- Aber aus **großer Ausdruckskraft** folgt immer **großer Berechnungsaufwand**.
 - Aussagenlogische Formeln können effizienter verarbeitet werden als prädikatenlogische Formeln.
 - Reguläre Sprachen können effizienter verarbeitet werden als kontextfreie Sprachen.

Automaten vs. LLMs

- Automaten und Grammatiken sind bei Compilern (Übersetzern) und in anderen Bereichen nicht mehr wegzudenken
 - Dienen zur Prüfung ob der Syntax des Programmcodes korrekt ist
- Bei der Übersetzung und Generierung “menschlicher” Sprache gibt es historisch zwei Ansätze:
 - Modellierung mittels Automaten und Grammatiken (weit verbreitet in der Linguistik)
 - Statistische Modelle basierend auf Verteilungen und Wahrscheinlichkeitstheorie (verwendet in LLMs)
 - ⇒ Aktuell haben LLMs die Nase vorn
 - ⇒ Aber: Schwierig, kausale oder logische Zusammenhänge in statistische Methoden/LLMs einzubauen
 - ⇒ Daher noch aktive Debatten⁶, was sich am Ende durchsetzen wird oder ob man beide Ansätze verbinden kann

⁶<https://www.nytimes.com/2023/03/08/opinion/noam-chomsky-chatgpt-ai.html>

Computersysteme

- Aufbau und Funktionsweise von Rechnerarchitekturen
- Computernetze, Netzwerkprotokolle

Theoretische Informatik

- Automatentheorie & Formale Sprachen
- Berechenbarkeit und Komplexität
- Formale Semantik von Programmiersprachen

Logic and Reasoning in Computer Science

- Vertiefung Prädikatenlogik
- Formal-mathematische Beschreibungen von Problemen in der Informatik
- Computergestützte Begründungen und Argumentationsmethoden
SAT solvers, SMT solvers, first-order theorem provers.

Tipps für Ihr weiteres Studium

- Bald haben Sie Ihr erstes Semester geschafft
- Nutzen Sie nach den Prüfungen eine ruhige Minute und reflektieren Sie über das vergangene Semester
 - Was hat gut und nicht so gut geklappt?
 - Was hat Ihnen Spaß gemacht und was nicht?
- Ziehen Sie Ihre Lehren daraus und setzen Sie sich konkrete Ziele fürs nächste Semester
 - Das hilft bei Ihrer persönlichen Entwicklung
 - Am besten machen Sie das nach jedem Semester
- Längerfristige Tipps:
 - Stellen Sie fest, welcher Bereich Ihnen am meisten Freude macht
 - Belegen Sie Kurse gezielt und nutzen Sie die Vielfalt von unserem Studienangebot
 - Je besser Sie Ihre Wahlmöglichkeiten nutzen, desto mehr Freude haben Sie an Ihrem Studium
 - **Trauen Sie sich was**
 - Tun Sie nicht nur das nötigste für Ihre Kurse
 - Informatik kann in so vielen Bereichen angewendet werden, entwickeln Sie eigene Projekte oder Forschung
 - Das hat oft auch positive Nebenwirkungen auf Ihre restlichen Kurse

Verschiedene Felder der Informatik

