

# FORMALE SYSTEME

## 1. Vorlesung: Willkommen/Einleitung formale Sprachen

Hannes Straß

Folien: © Markus Krötzsch, <https://iccl.inf.tu-dresden.de/web/FS2020>, CC BY 3.0 DE

TU Dresden, 11. Oktober 2021

## Raum, Zeit, URL

- **Vorlesungen:** virtuell (über Zoom), werden aufgezeichnet
- **Vorlesungstermine:**  
Montag, DS3 (11:10–12:40)  
Donnerstag, DS4 (13:00–14:30)
- **Keine Vorlesungen:**  
Jahreswechsel: Do 23.12., Mo 27.12., Do 30.12., Mo 03.01.
- **Vorlesungswebseite:**  
[https://iccl.inf.tu-dresden.de/web/Formale\\_Systeme\\_\(WS2021\)](https://iccl.inf.tu-dresden.de/web/Formale_Systeme_(WS2021))  
(Folien, Übungsblätter, Termine, etc.)
- **OPAL-Kurs**  
<https://tud.link/kef0>  
(Im Forum: Support bei allen Fragen zur Vorlesung)
- **Chatkanal (Matrix)**  
<https://matrix.to/#/#fs2021:tu-dresden.de>

# Willkommen zur Vorlesung Formale Systeme

## Übungen

- **Anmeldung** zu den Übungen über OPAL (Link siehe Vorlesungswebseite)  
**Einschreibungsbeginn: Heute, 21. Oktober 2021, 15:00 Uhr**
- **Übungsblätter** jeweils donnerstags „nach der Vorlesung“  
(erstes Übungsblatt am 14. Oktober 2021)
- **Beginn der Übungen:** 18. Oktober 2021
- **Übungsablauf**, vereinfacht, idealisiert:
  - Aufgaben werden zu Hause bearbeitet, so gut es geht;
  - in der Übung helfen Gruppenleiter:innen bei Fragen und Problemen und zeigen Beispiellösungen.

## Prüfung und Prüfungsvorbereitung

- schriftliche Prüfung (90min?) am Ende des Wintersemesters
- prüfungsrelevant:
  - kompletter Stoff, der in der Vorlesung behandelt wird
  - wiedergeben (definieren) **und** anwenden (rechnen)
- Modulnote ergibt sich je nach Studiengang
- zur zusätzlichen Vorbereitung wird es eine Probeklausur geben

## Übersicht und Motivation

## Formale Systeme bestehen

Tipps:

- **Übungsaufgaben selbst lösen – vor der Übung**  
Die Prüfung besteht im Lösen von Rechenaufgaben. Die Konzepte „verstanden“ zu haben hilft allein nicht weiter.
- **Schnell sein**  
Prüfungszeit ist meistens knapp. Es reicht nicht, Aufgaben „im Prinzip“ lösen zu können. Man muss sie schnell lösen.
- **Während des Semesters lernen**  
Die Zeit vor der Prüfung sollte der Wiederholung und Festigung dienen, nicht der erstmaligen Beschäftigung mit den Themen.
- **Sich selbst realistisch einschätzen**  
Man sollte selbst wissen, ob man genug gelernt hat.<sup>1</sup>

---

<sup>1</sup>Man beachte aber auch den Dunning-Kruger-Effekt.

## Grundlegende Fragen der Informatik

Was ist ein Computer?

Eine Maschine, die rechnet.

Was ist „Rechnen“?

Die systematische Überführung von Eingaben in Ausgaben.

Was sind „Eingaben“ und „Ausgaben“?

Folgen von Zeichen, zum Beispiel Dateien oder Textausgaben.

So viele Computer, Programme, ... – das passt doch in kein Studium!

Nein, man muss sich auf das Wesentliche konzentrieren.

Was ist das Wesentliche?

Vereinfachte Modelle für Computer und Rechenverfahren.

Was sagen uns Modelle über echte Computer und Software?

Was man berechnen kann, wie aufwändig es ist, wie man es implementieren kann, ob es „stimmt“, ...

## Zielstellung, Kernthemen

Ziel dieser Vorlesung ist es, wichtige Grundlagen zur **Modellierung von Berechnung** in der Informatik einzuführen, **konkrete Modelle** vorzustellen und ihre **Eigenschaften verständlich zu machen**.

- Ein- und Ausgaben sind Zeichenfolgen  
~ Wir beginnen mit Wörtern und **formalen Sprachen**
- Wir wollen Berechnungsaufgaben beschreiben  
~ Spezifikation von Sprachen  
(direkt, mit **Grammatiken**, mit **regulären Ausdrücken**, ...)
- Fokus auf einfache Berechnungsmodelle  
~ **Automaten** (in vielen Versionen ...)
- Man kann Berechnungsaufgaben auch logisch spezifizieren  
~ **Aussagenlogik** als einfacher Einstieg
- Lösung logischer Probleme  
~ Berechnungsverfahren zum **logischen Schließen**

## Gliederung „Formale Systeme“

### Teil 1: Sprachen und Automaten

- Formale Sprachen und Grammatiken
- Reguläre Sprachen und endliche Automaten
- Kontextfreie Sprachen und Kellerautomaten
- Kontextsensitive Sprachen, Typ-0-Sprachen und Turingmaschinen

### Teil 2: Aussagenlogik

- Syntax und Semantik
- logisches Schließen, Backtracking und andere Verfahren
- Horn-Logik als Vereinfachung

## Literatur: Lehrbücher

Der Vorlesungsstoff gehört zu fast jeder Informatikausbildung. Es gibt viele Lehrmaterialien und eine weitgehend einheitliche Notation.

Lehrbücher zum ersten Teil der Vorlesung (formale Sprachen):

- Uwe Schöning: **Theoretische Informatik – kurz gefasst**. Spektrum Akademischer Verlag  
deutschsprachiger Standardtext; in der Tat ziemlich kurz gefasst
- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman: **Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit**. Pearson Studium  
aus dem Englischen übertragenes Standardwerk; Original evtl. besser
- Michael Sipser: **Introduction to the Theory of Computation**. Cengage Learning  
Standardtext zu Sprachen und Berechnungskomplexität; Englisch

## Literatur: Frei zugängliche Skripte

Es existieren mehrere Skripte zu dieser Vorlesung:

- Franz Baader: **Skript Formale Systeme, Teil 1 – Automaten und formale Sprachen**. TU Dresden.  
Siehe [https://lat.inf.tu-dresden.de/teaching/ws2013-2014/FS/script\\_2016-02.pdf](https://lat.inf.tu-dresden.de/teaching/ws2013-2014/FS/script_2016-02.pdf).
- Christel Baier, Manuela Berg, Walter Nauber: **Formale Systeme WS 2011/2012: Skript zur Vorlesung**. TU Dresden.  
Siehe [https://www.researchgate.net/profile/Christel-Baier/publication/267409804\\_Formale\\_Systeme\\_WS\\_20112012\\_Skript\\_zur\\_Vorlesung/links/54e24b020cf2966637965e77/Formale-Systeme-WS-2011-2012-Skript-zur-Vorlesung.pdf](https://www.researchgate.net/profile/Christel-Baier/publication/267409804_Formale_Systeme_WS_20112012_Skript_zur_Vorlesung/links/54e24b020cf2966637965e77/Formale-Systeme-WS-2011-2012-Skript-zur-Vorlesung.pdf)

Die Vorlesungen haben kleine Abweichungen, stimmen aber in vielen wichtigen Punkten überein.

# Fragen, Ideen, Anmerkungen

# Sprachen in der Informatik

## Wozu Sprachen?

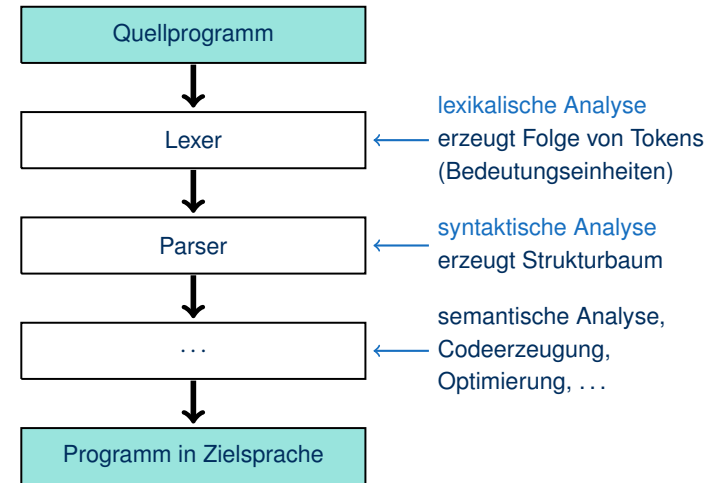


# Wozu Sprachen?

Formale Sprachen und die zugehörigen Automaten und Grammatiken haben sehr viele Anwendungen:

- **Compilerbau**  
Programmiersprachen sind typische formale Sprachen
- **Interpretation natürlicher Sprachen**  
viele Anwendungen des Sprachverstehens nutzen Grammatiken
- **Datenaustausch**  
Textformate (HTML, CSV, JSON, XML, ...) bilden formale Sprachen
- **Formatierung/Validierung/Spezifikation**  
z.B. um die Gültigkeit von Formulareingaben zu prüfen
- **Berechenbarkeit und Komplexität**  
mächtigere Sprachdefinitionen verlangen teurere Algorithmen
- **Informationsextraktion**  
Formale Sprachen helfen bei der Mustersuche in Textdokumenten
- **Datenbanken**  
Datenbankanfragen können Muster suchen, z.B. in Graphdatenbanken

# Beispiel Compiler



# Lexikalische Analyse

Eingabe: Zeichenkette eines Programms

```
Bsp.: lengthCm = lengthInch * 2.54;  
Kette von 29 Zeichen
```

Ausgabe: Kette von Grundsymbolen (Tokens)

```
Bsp.: IDENTIFIER EQUALS IDENTIFIER STAR NUMBER SEMICOLON  
Kette von 6 Tokens
```

- Zur Weiterverarbeitung wird Tokens oft weitere Information mitgegeben, z.B. IDENTIFIER("lengthCm") und IDENTIFIER("lengthInch").
- Manche Zeichen werden nicht zu Tokens (z.B. Leerzeichen, Kommentare).

# Lexikalische Analyse (2)

Verschiedene Arten von Grundsymbolen:

- Schlüsselwörter (if, while, class, ...)
- Operatoren (=, +, >>, ...)
- Bezeichner (lengthCm, getLength, InventoryItem, ...)
- Literale (2.54, true, "HelloWorld!", ...)

Für einige Grundsymbole gibt es unendlich viele Möglichkeiten:

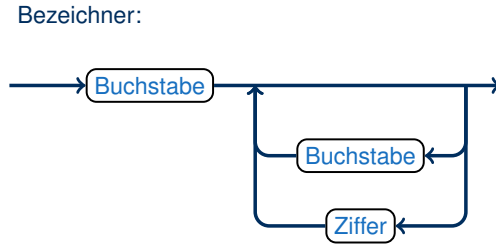
```
Bsp.: „Ein Bezeichner ist ein String, der mit einem Buchstaben beginnt und danach nur Buchstaben oder Ziffern enthält.“
```

Wie soll ein Lexer das korrekt erkennen?

## Bezeichner erkennen

Bsp.: „Ein Bezeichner ist ein String, der mit einem Buchstaben beginnt und danach nur Buchstaben oder Ziffern enthält.“

Schematische Darstellung als Syntaxdiagramm:



Hierbei stehen **Buchstabe** und **Ziffer** jeweils für ein beliebiges Zeichen dieses Typs.

## Bezeichner erkennen (2)

Wie setzt man das praktisch um?

Code eines unerfahrenen Programmierers:

```
function isIdentifier(input) {
  let state = "start", symbol;
  while input.hasNextSymbol() {
    symbol = input.getNextSymbol();
    if ( state == "start" && isLetter(symbol) ) { state = "inner"; }
    else if ( state == "start" && !isLetter(symbol) ) { return false; }
    else if ( state == "inner" && isLetter(symbol) ) {
      /* ok, wir lesen einfach weiter */
    }
    else if ( state == "inner" && isNumber(symbol) ) {
      /* ok, wir lesen einfach weiter */
    }
    else if ( state == "inner" &&
      !isLetter(symbol) && !isNumber(symbol) ) { return false; }
  }
  if (state == "inner") return true;
  return false;
}
```

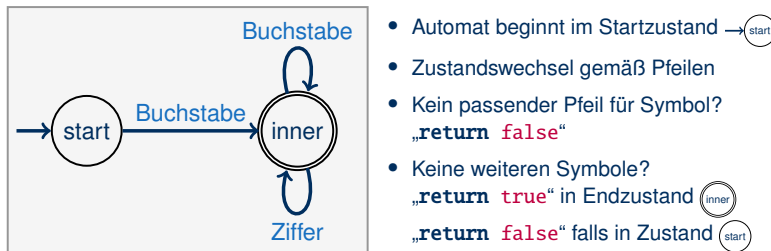
## Bezeichner erkennen (3)

Der (schlechte) Programmcode zeigt eine wichtige Eigenschaft:

Der **Lexer** muss nur einen „Zustand“ speichern.

(Im Beispiel ist dies der Wert der Variablen state.)

Darstellung als **endlicher Automat**:



## Bezeichner erkennen (4)

Wie kann man jemandem am besten erklären, was ein „Bezeichner“ ist?

- **Sprachliche Umschreibung**: ungenau und mehrdeutig
- **Syntaxdiagramm, endlicher Automat**: graphische Darstellung anschaulich, aber schnell unübersichtlich
- **Programmcode**: Kernidee geht in Implementierungsdetails verloren

~ Spezifikationen verwenden meist **Grammatiken**:

```
Bezeichner ::= Buchstabe | Buchstabe InBezeichner
InBezeichner ::= BuchOderZiff | BuchOderZiff InBezeichner
BuchOderZiff ::= Buchstabe | Ziffer
Buchstabe ::= "a" | "b" | ... | "z"
Ziffer ::= "0" | "1" | ... | "9"
```

# Formale Sprachen

Wir haben gesehen:

- In der Praxis interessieren wir uns für (oftmals unendliche) Mengen von Strings, z.B. **Bezeichner** oder **Java-Programme**.  
→ Solche Mengen nennt man **formale Sprachen**.
- Man kann formale Sprachen auf viele Arten beschreiben, z.B. mit Automaten, Grammatiken oder Syntaxdiagrammen.  
→ Beschreibungsmethoden haben unterschiedliche Stärken und Schwächen.
- Kein Ansatz kann alle Sprachen beschreiben, z.B. gibt es keinen Automaten, der Java-Programme parsen kann.  
→ Man unterscheidet **Typen von Sprachen**.

Erster Teil der Vorlesung:

Formale Sprachen und ihre Darstellungsarten

# Formale Sprachen

## Grundbegriffe: Alphabete und Wörter

Ein **Alphabet** ist eine endliche, nichtleere Menge.  
Elemente des Alphabets heißen **Symbole**.

Meistens verwendet man die Buchstaben  $\Sigma$  (Sigma) oder  $\Gamma$  (Gamma) für Alphabete.

Ein endliches **Wort** über einem Alphabet  $\Sigma$  ist eine endliche Folge von Symbolen aus  $\Sigma$ . Wenn  $w$  ein endliches Wort ist, dann ist  $|w|$  seine **Länge** (die Anzahl seiner Symbole).

Alle Wörter in dieser Vorlesung sind endlich. Wir sagen das ab jetzt nicht immer dazu.

Beispiel:  $\Sigma = \{a, b\}$  ist ein Alphabet.  
Wörter über diesem Alphabet sind z.B. **abba** oder **bbb**.  
Die Längen dieser Wörter sind  $|abba| = 4$  und  $|bbb| = 3$ .

## Grundbegriffe: Konkatenation, Leeres Wort

Die wichtigste Operation auf Wörtern ist Konkatenation („Hintereinanderhängung“):

Die **Konkatenation** von zwei Wörtern  $w = a_1 \dots a_n$  und  $v = b_1 \dots b_m$  ist das Wort  $wv = a_1 \dots a_n b_1 \dots b_m$ .

Wir schreiben also konkatenierte Wörter einfach nebeneinander.

Das **leere Wort**  $\epsilon$  (epsilon) ist das Wort der Länge 0, also  $|\epsilon| = 0$ .

Es gibt genau ein leeres Wort und man kann es über jedem Alphabet bilden.

Beispiel: Für die Wörter  $w = \text{tuben}$  und  $v = \text{wachs}$  gilt  $wv = \text{tubenwachs}$  und  $vw = \text{wachstuben}$ .

Beispiel: Für jedes Wort  $w$  gilt:  $w\epsilon = \epsilon w = w$ .

## Grundbegriffe: \*-fixe

Manchmal ist es praktisch, Teile von Wörtern zu bezeichnen:

Sei  $w = a_1 \dots a_n$  ein Wort der Länge  $n$ .

- Ein **Präfix** von  $w$  ist ein Wort  $a_1 \dots a_i$  mit  $0 \leq i \leq n$ . **oder**  $\epsilon$ .
- Ein **Suffix** von  $w$  ist ein Wort  $a_j \dots a_n$  mit  $1 \leq j \leq n+1$ . **oder**  $\epsilon$ .
- Ein **Infix** von  $w$  ist ein Wort  $a_i \dots a_j$  mit  $1 \leq i \leq j \leq n$  **oder**  $\epsilon$ .

Beispiel: Das Wort **Staubecken** hat ein Präfix **Staub**, ein Suffix **ecken** und ein Infix **taube** (und viele andere mehr).

Beispiel: Das leere Wort  $\epsilon$  ist Präfix, Suffix und Infix von jedem Wort (sogar von  $\epsilon$  selbst).

## Die kleinste und die größte Sprache

Die kleinste Sprache über dem Alphabet  $\Sigma$  ist die **leere Sprache**  $\emptyset$ .

Beispiel: Die Sprache  $\{\epsilon\}$ , welche nur das leere Wort enthält, ist nicht leer!

Die leere Sprache ist also Teilmenge jeder anderen Sprache.

Die größte Sprache über einem Alphabet  $\Sigma$  ist die **Sprache aller Wörter** über  $\Sigma$ . Man bezeichnet sie mit  $\Sigma^*$ .

Beispiel:  $\{a, b\}^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$ .

Jede Sprache über  $\Sigma$  ist also eine Teilmenge von  $\Sigma^*$ .

## Grundbegriffe: formale Sprache

Sei  $\Sigma$  ein Alphabet. Eine Menge von Wörtern über  $\Sigma$  wird **formale Sprache** über  $\Sigma$  genannt.

Die Zusätze „formal“ und „über  $\Sigma$ “ werden meist weggelassen, wenn dadurch keine Missverständnisse auftreten können.

Sprachen werden typischerweise mit dem Buchstaben **L** bezeichnet.

Beispiel: Die Sprache **Ziffer** =  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$  ist eine endliche Sprache (über jedem Alphabet, das zumindest die Symbole **0, 1, 2, 3, 4, 5, 6, 7, 8, 9** enthält).

Beispiel: Die zuvor genannte Sprache **Bezeichner** ist unendlich.

Beispiel: Die Menge aller Tweets ist eine endliche Sprache.

Beispiel: Die Menge aller gültigen HTML-5-Dokumente ist eine unendliche Sprache über dem Alphabet aller Unicode Code Points.

## Operationen auf Sprachen

Man kann mit Sprachen „rechnen“, um neue Sprachen zu bilden:

- **Vereinigung:**  $L_1 \cup L_2$
- **Schnitt:**  $L_1 \cap L_2$
- **Komplement:**  $\bar{L} = \Sigma^* \setminus L$  (nur sinnvoll, wenn das Alphabet  $\Sigma$  klar festgelegt ist!)
- **Produkt:**  $L_1 \circ L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$
- **Potenz:**  $L^0 = \{\epsilon\}$  und  $L^{n+1} = L \circ L^n$
- **Kleene-Abschluss:**  $L^* = L^0 \cup L^1 \cup L^2 \cup \dots = \bigcup_{i \geq 0} L^i$

Beispiel: Die Sprache **Bezeichner** kann aus den Sprachen **Buchstabe** und **Ziffer** konstruiert werden:

$$\text{Bezeichner} = \text{Buchstabe} \circ (\text{Buchstabe} \cup \text{Ziffer})^*$$

Beispiel: Tweets sind Ketten aus bis zu 140 Zeichen:<sup>1)</sup>

$$\text{Tweet} = \bigcup_{1 \leq i \leq 140} \text{Buchstabe}^i$$



## Operationen auf Sprachen: Kurzschreibweisen

- Der Produktoperator wird oft eingespart:  
Statt  $L_1 \circ L_2$  schreiben wir  $L_1 L_2$ .

Hinweis: Manche Autor:innen verwenden  $\cdot$  statt  $\circ$ .

- Ein **Plus-Operator** ist manchmal praktisch:  $L^+ = \bigcup_{i \geq 1} L^i$
- Die **Differenz** ist indirekt ausdrückbar:  $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$
- Klammern einsparen: Kleene-Stern  $*$  und Plus  $^+$  binden immer am stärksten, gefolgt **in Reihenfolge absteigender Bindung** von  $\circ$ ,  $\cap$ ,  $\cup$  und  $\setminus$ .

Manchmal sind Klammern und  $\circ$  aber zur Lesbarkeit trotzdem hilfreich.

Beispiel: Die Menge aller gültigen Schreibweisen für Dezimalzahlen kann wie folgt definiert werden:

$$\text{Dezimalzahl} = (\{\epsilon\} \cup \{+, -\}) \circ (\{\emptyset\} \cup (\mathbb{Z} \setminus \{\emptyset\}) \circ \mathbb{Z}^*) \circ (\{\epsilon\} \cup \{.\} \mathbb{Z}^+)$$

mit  $\mathbb{Z} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ .

Ausdrücke der Form  $(\{\epsilon\} \cup L)$  beschreiben optionale Bestandteile.

## Rechenregeln (1)

Es gelten viele Rechenregeln für Operationen auf Sprachen.

Sprachen sind Mengen – es gelten die Gesetze der Mengenlehre.

Für Sprachen  $L_1, L_2, K$  über einem gemeinsamen Alphabet  $\Sigma$  gilt:

- $L_1 \cap L_2 = L_2 \cap L_1$  (Kommutativgesetz  $\cap$ )
- $L_1 \cup L_2 = L_2 \cup L_1$  (Kommutativgesetz  $\cup$ )
- $K \cup (L_1 \cap L_2) = (K \cup L_1) \cap (K \cup L_2)$  (Distributivgesetz  $\cup$  über  $\cap$ )
- $K \cap (L_1 \cup L_2) = (K \cap L_1) \cup (K \cap L_2)$  (Distributivgesetz  $\cap$  über  $\cup$ )
- $L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$  (De Morgansches Gesetz 1)
- $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$  (De Morgansches Gesetz 2)

## Rechenregeln (2)

Es gelten weitere Gesetze für die zusätzlichen Sprachoperationen.

Für Sprachen  $L_1, L_2, K$  über einem gemeinsamen Alphabet  $\Sigma$  gilt:

- $K \circ (L_1 \cup L_2) = (K \circ L_1) \cup (K \circ L_2)$   
 $(L_1 \cup L_2) \circ K = (L_1 \circ K) \cup (L_2 \circ K)$  } Distributivgesetze  $\circ/\cup$
- Achtung: Distributivgesetze  $\circ/\cap$  gelten nicht!**
- $K = K \circ \{\epsilon\} = \{\epsilon\} \circ K$  ( $\{\epsilon\}$  neutrales Element zu  $\circ$ )
- $K \circ \emptyset = \emptyset \circ K = \emptyset$  ( $\emptyset$  absorbierendes Element zu  $\circ$ )
- $K^+ = K^* \circ K = K \circ K^*$
- $K^* = K^+ \cup \{\epsilon\} = (K \setminus \{\epsilon\})^*$

Für  $K = \{\epsilon, a\}$ ,  $L_1 = \{a\}$ ,  $L_2 = \{aa\}$  über  $\Sigma = \{a\}$  gilt:

$$K \circ (L_1 \cap L_2) = \{a, \epsilon\} \circ (\{a\} \cap \{aa\}) = \{a, \epsilon\} \circ \emptyset = \emptyset$$

$$(K \circ L_1) \cap (K \circ L_2) = (\{a, \epsilon\} \circ \{a\}) \cap (\{a, \epsilon\} \circ \{aa\}) = \{aa, a\} \cap \{aaa, aa\} = \{aa\}$$

## Quiz: Rechnen mit Sprachen

- Vereinigung:**  $L_1 \cup L_2$
- Schnitt:**  $L_1 \cap L_2$
- Komplement:**  $\overline{L} = \Sigma^* \setminus L$  (nur sinnvoll, wenn das Alphabet  $\Sigma$  klar festgelegt ist!)
- Produkt:**  $L_1 \circ L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$
- Potenz:**  $L^0 = \{\epsilon\}$  und  $L^{n+1} = L \circ L^n$
- Kleene-Abschluss:**  $L^* = L^0 \cup L^1 \cup L^2 \cup \dots = \bigcup_{i \geq 0} L^i$

**Quiz:** Seien  $K = \{\epsilon, a\}$ ,  $L_1 = \{a\}$ ,  $L_2 = \{aa\}$  über  $\Sigma = \{a\}$ .

Welche der folgenden Aussagen sind wahr?

- $K \cap (L_1 \cup L_2) = \{a\}$  ✓
- $K^* \cap (L_1 \cup L_2) = K$  ✗
- $L_1^* = K^+$  ✓
- $L_1^* \subseteq L_2^*$  ✗

## Rechenregeln beweisen?

**Behauptung:**  $\mathbf{K} \circ (\mathbf{L}_1 \cup \mathbf{L}_2) = (\mathbf{K} \circ \mathbf{L}_1) \cup (\mathbf{K} \circ \mathbf{L}_2)$

**Beweis:** Wir schauen uns einfach die Definitionen der jeweiligen Operationen an.

$$\begin{aligned}\mathbf{K} \circ (\mathbf{L}_1 \cup \mathbf{L}_2) &= \{wv \mid w \in \mathbf{K}; v \in (\mathbf{L}_1 \cup \mathbf{L}_2)\} \\ &= \{wv \mid w \in \mathbf{K}; v \in \mathbf{L}_1 \text{ oder } v \in \mathbf{L}_2\} \\ &= \{wv \mid w \in \mathbf{K}; v \in \mathbf{L}_1\} \cup \{wv \mid w \in \mathbf{K}; v \in \mathbf{L}_2\} \\ &= (\mathbf{K} \circ \mathbf{L}_1) \cup (\mathbf{K} \circ \mathbf{L}_2)\end{aligned}$$

(Allgemein gilt: Jede Operation, die „punktweise“ auf einer Menge arbeitet, distribuiert über  $\cup$ .) □

## Zusammenfassung und Ausblick

Formale Sprachen sind für die Informatik sehr wichtig.

Grundbegriffe: Alphabet, Wort, Sprache, Konkatenation, Präfix/Suffix/Infix,  $\epsilon$ ,  $\Sigma^*$ .

Es gibt eine Reihe von Operationen auf Sprachen, mit denen man rechnen kann.

Offene Fragen:

- Wie viele Wörter und wie viele Sprachen gibt es eigentlich?
- Wie kann man Sprachen beschreiben oder implementieren?
- Wie kommt Berechnung ins Spiel?