

---

# *Einführung in die Informatik*

---

Formale Sprachen

Meik Teßmer

---

Inhalte  
●○○

Interaktion  
○○○

Produktion  
○○○○○○○

Bedeutung  
○○○○○○○

Anwendungen  
○○○○○○○○○○○○○

# Inhalte

# Lernziele

- Formulierung korrekter Fragen (und Antworten)
- Theorie der formalen Sprachen
- Entwicklung einer domänenspezifischen Sprache

Allgemein formuliert: Wie „kommuniziere“ ich mit einem Computer?

# Rückblick und Überblick

in den letzten beiden Vorlesungen gesehen:

- Mittel, um über Probleme und Lösungen reden zu können

Organisationsform *Schichten*:

- Anwender
- Problemlösungsschicht
- OS
- Hardware

Fokus liegt heute auf dem Übergang Anwender-Problemlöseschicht

Inhalte  
○○○

**Interaktion**  
●○○

Produktion  
○○○○○○○

Bedeutung  
○○○○○○○

Anwendungen  
○○○○○○○○○○○○○

# Interaktion

# Einstieg

- Anwender an Computer: Was ergibt 2 und 3?
- Computer: ??
- Anwender an Computer: Wo ist die Bank?
- Computer: ??

→ Wo liegt das Problem?

## Formulierung des Problems

- anders formuliert:  
*Berechne die Summe der Zahlen 2 und 3.*
- eine mögliche Übersetzung:  $2 + 3$

Was sehen wir?

ASCII → Zeichen

Inhalte  
○○○

Interaktion  
○○○

Produktion  
●○○○○○

Bedeutung  
○○○○○

Anwendungen  
○○○○○○○○○○○○

# Produktion

## Zwei Seiten von Sprachen

jede Aussage in einer Sprache hat zwei Aspekte:

Syntax *Satzlehre*: Grammatik einer Aussage

Semantik *Bedeutungslehre*: Bedeutung einer Aussage

Problem: Semantik oft nicht eindeutig, erschließt sich aus dem Kontext der Aussage (*Pragmatik*)

# Grundlagen formaler Sprachen

Zu einer formalen Sprache gehören:

- $\Sigma$ : eine endliche, nichtleere Menge von Zeichen
- Verknüpfungsoperation (*Konkatenation*): verknüpft Zeichen zu *Wörtern*

→ *formale Sprache* über  $\Sigma$ : Menge aller Wörter, die so gebildet werden können

Beispiel:

- $\Sigma = \{b, e, i, r\}$
- Verknüpfung: einfach hintereinander schreiben

# Produktionsregeln

## Backus-Naur-Form (BNF)

- benannt nach John Backus und Peter Naur
- kam beim Compiler von Algol 60 zum Einsatz

### Beispiel:

```
<Ziffer ohne Null> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<Ziffer>           ::= 0 | <Ziffer ohne Null>
<Zweistellige Zahl> ::= <Ziffer ohne Null><Ziffer>
```

# Backus-Naur-Form

Eine *Backus-Naur-Form* besteht aus:

- Terminalsymbole: ! hallo, welt
- Nichtterminalsymbole: <name\_des\_symbols>
- Metazeichen: ::=, |
- Menge von Regeln:
  - <name\_1> ::= <name\_2> <name\_3> !
  - <name\_2> ::= hallo
  - <name\_3> ::= welt

(einzige) mögliche Produktion: hallo welt !

## Beispiel: Satzbau

*Aufbau eines Satzes?*

## Beispiel: Satzbau

<Satz> ::= <Subjekt> <Prädikat> <Objekt> .  
<Subjekt> ::= <Artikel> <Substantiv>  
<Prädikat> ::= <Verb>  
<Objekt> ::= <Artikel> <Substantiv>  
<Artikel> ::= der | die | das  
<Substantiv> ::= Mann | Frau | Auto | Buch  
<Verb> ::= fährt | liest

mögliche Produktionen:

- der Mann liest das Buch .
- die Frau fährt das Auto .

Inhalte  
○○○

Interaktion  
○○○

Produktion  
○○○○○○○

**Bedeutung**  
●○○○○○

Anwendungen  
○○○○○○○○○○○○○○

# Bedeutung

## Zurück zur Einstiegsfrage

- Problem: „Was ergibt  $2 + 3$ “?
- übersetzt in BNF:

```
<formel> ::= <zahl> <operator> <zahl>
<zahl>   ::= "2" | "3"
<operator> ::= "+"
```

- Code:

```
from pyarsing import Literal, Or
# Terminalsymbole
zahl = Or( [ Literal("2"), Literal("3") ] )
operator = Literal("+")
# Produktionsregel mit Nichtterminalsymbolen
formel = zahl + operator + zahl
```

## Zurück zur Einstiegsfrage

- Test einer Zeichenkette auf syntaktische Korrektheit:

```
test_satz = "2 + 3"  
print(formel.parseString(test_satz))
```

- Test mit einer falschen Zeichenkette:

```
test_satz = "4 + 3"  
print(formel.parseString(test_satz))
```

# Semantik

- Ergebnis der Syntaxanalyse:

tokens = ['2', '+', '3']

- Bedeutung:

- 2, 3 sind Zahlen
- + weist auf Addition hin

- Übertragung:

- '2' → 2
- '3' → 3
- '+' → sum()

# Semantik

```
# Semantik definieren
zahlen = { "2": 2, "3": 3 }
operatoren = { "+": sum }
# Tokens analysieren
liste_der_zahlen = []
operator = None
for token in tokens:
    if token in zahlen:
        # Zahlen suchen
        echte_zahl = zahlen[token]
        liste_der_zahlen.append(echte_zahl)
    elif token in operatoren:
        # Operator suchen
        operator = operatoren[token]
ergebnis = operator(liste_der_zahlen)
```

## Erstes Zwischenergebnis

- Computer brauchen präzise Angaben, die sie *verstehen*
- Aussagen haben eine Syntax und eine Semantik
- Aussagen können mit Hilfe einer formale Sprachdefinition auf Korrektheit geprüft werden
- Semantik muss „von uns“ vorgegeben werden

Frage: Anwendungsgebiete formaler Sprachen?

Inhalte  
○○○

Interaktion  
○○○

Produktion  
○○○○○○○

Bedeutung  
○○○○○○○

Anwendungen  
●○○○○○○○○○○○

# Anwendungen

# Einsatzgebiete

- formale Sprachen allg.: Compiler-/Interpreterbau → *Chomsky-Hierarchie*: Hierarchie von Klassen formaler Grammatiken
- verwandt mit BNF: *reguläre Ausdrücke*
- Verwendung:
  - Suchen/Ersetzen in Editoren/Textverarbeitung
  - Textanalyse
  - Datenbereinigung (Filter)

# Beispiel für die Verwendung reguläre Ausdrücke: Projektliste

Datei mit einer Liste von Projekten:

= Projekt A =

...

= Projekt B =

...

Was sehen wir?

## Beispiel: Projektliste

charakteristisch sind:

- bestimmte Wörter: Projekt
- =- und Leerzeichen
- Position von Zeichen (Zeilenanfang, -ende)

→ wir brauchen ein *Muster*, das diese Eigenschaften berücksichtigt

# Formulierung regulärer Ausdrücke

Position in einer Zeile:

- $\wedge$  entspricht Zeilenanfang
- $\$$  entspricht Zeilenende

Zeichen:

- $.$  steht für ein beliebiges Zeichen
- $[abc]$  definiert Menge von Zeichen (hier a, b und c)

Auftreten:

- $*$ : 0 oder mehrfaches Auftreten
- $+$ : 1 oder mehrfaches Auftreten
- $?$ : optionales Auftreten

# Formulierung regulärer Ausdrücke

Beispiele:

- hallo: passt genau auf das Wort hallo
- ^c: Buchstabe c am Zeilenanfang
- a\$: Buchstabe a am Zeilenende
- x\*\$: 0 oder beliebig viele x am Zeilenende
- [a-z]+ly: passt auf quickly und carefully

in Python:

```
text = "... carefully disguised but captured quickly."  
matches = re.findall("[a-z]+ly", text)  
print(matches)  
>>> ['carefully', 'quickly']
```

## Beispiel: Projektliste

= Projekt A =

...

= Projekt B =

...

Wie könnte ein passendes Muster aussehen, um alle Projekte aufzulisten?

# Umsetzung

mit grep:

```
grep '^= Projekt .* =$' projekte.txt
```

## Beispiel aus der Praxis

Logdatei eines Webservers:

```
84.19.190.98 - - [29/Sep/2023:03:30:24 +0100] "GET \
/cgi-bin/a.pl?a=login HTTP/1.0" 200 10574 \
"http://www.wellho.net/cgi-bin/a.pl?a=login" \
"Mozilla/4.0 (compatible; MSIE 6.0; Windows XP)"
```

Frage: Wie oft wurde eine bestimmte Webseite aufgerufen?

# Umsetzung

```
import re
counter = {}
# Muster
muster = "(GET|POST)\s+([\^\s\?]+).*\s+([2345]\d\d)\s+(\S+)"
url = re.compile(muster)

for line in lines:
    matched = url.findall(line)
    if matched:
        # [('GET', '/cgi-bin/a.pl', '200', '10574')]
        page = matched[0][1]
        if not page in counter:
            counter[page] = 0
        counter[page] += 1
```

# Zusammenfassung

- wir können mit dem Computer *reden*:
  - syntaktisch korrekte Aussagen
  - Semantik: was dem Computer „einprogrammiert“ wurde
- Einblick in die Theorie formaler Sprachen (BNF, reguläre Ausdrücke)
- praktische Anwendungen

Inhalte  
○○○

Interaktion  
○○○

Produktion  
○○○○○○○

Bedeutung  
○○○○○○○

Anwendungen  
○○○○○○○○○○○●

Ende