TECHNISCHE UNIVERSITÄT DRESDEN

FORMALE SYSTEME

3. Vorlesung: Endliche Automaten

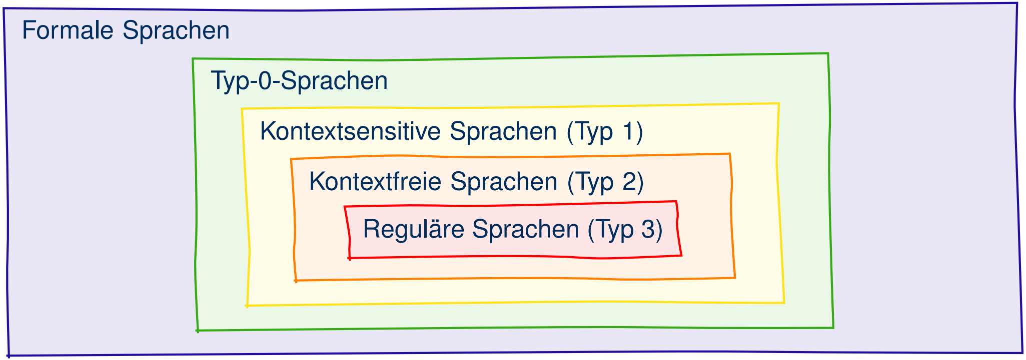
Markus Krötzsch

Professur für Wissensbasierte Systeme

TU Dresden, 16. Oktober 2023

Wiederholung

Mit Grammatiken können wir Sprachen beschreiben und sie grob in Typen unterteilen:



Grammatiken in der Praxis

oben: John Backus, Peter Naur
unten: Niklaus Wirth, Panini

Eine „ASCII-Syntax“ für Typ-2-Grammatiken ist die sogenannte Backus-Naur-Form (BNF):[1]

- statt $\to$ wird ::= verwendet
- | für Alternativen (wie bei uns)
- Markierung: "Terminalsymbole" und <Nichterminalsymbole>

Erweiterte BNF (EBNF, von Niklaus Wirth):

- allgemein wie BNF, aber Konkatenation mit Kommas
- Zusätzliche Abkürzungen `[X]` für „null oder ein X“ und `{X}` für „null oder mehr X“

In der Praxis sind unterschiedliche Notationen im Einsatz

(EBNF von Wirth; ISO EBNF; W3C EBNF; „Augmented BNF“ der IETF; verschiedene „EBNF“ aus Vorlesungen und Lehrbüchern . . .)

[1] Panini, 5./4. Jhd. v. Chr.; neu erfunden von John Backus in 1959

Sprachen Nutzen und Verstehen

Von Beschreiben zu Erkennen

Grammatiken beschreiben Sprachen als Mengen von Wörtern, die sie erzeugen können.

Praktisch wichtige Aufgabe: erkenne ob ein Wort in einer Sprache liegt

Das Wortproblem für eine Sprache **L** über Alphabet Σ besteht darin, die folgende Funktion zu berechnen:

Eingabe: ein Wort $w \in \Sigma^*$
Ausgabe: „ja“ wenn $w \in \mathbf{L}$ und „nein“ wenn $w \notin \mathbf{L}$

Beispiel: Für die Sprache $\{\mathtt{a}\} \circ \{\mathtt{b}\}^*$ wird das Wortproblem durch einen einfachen Algorithmus in linearer Zeit gelöst (teste, ob der erste Buchstabe $\mathtt{a}$ ist und ob alle folgenden Buchstaben $\mathtt{b}$ sind).

Andererseits kann das Wortproblem sehr schwer sein, selbst für Sprachen, die mit einer Grammatik formal beschrieben sind.

Ausblick

Wir werden sehen, dass jede Sprachklasse zu einem bestimmten Berechnungsmodell passt:

Sprachklasse	Berechnungsmodell	Wortproblem
Typ 0	Turingmaschine (TM)	semi-entscheidbar[1]
Kontextsensitiv	nichtdeterministische TM mit linearem Speicher	PSpace-vollständig[2]
Kontextfrei	nichtdeterministischer Kellerautomat	polynomiell
Regulär	endlicher Automat	polynomiell[3]

[1] Akzeptanz von Wörtern kann beliebig lange dauern, so dass man nie weiß, ob sie noch passiert; Wörter können nur rekursiv aufgezählt werden

[2] höchstwahrscheinlich schwerer als NP; jeder praktisch bekannte Algorithmus benötigt im Worst Case exponentiell viel Zeit

[3] tatsächlich sogar noch viel einfacher als Typ 2

Weitere Fragestellungen

Das Wortproblem ist nicht die einzige praktisch relevante Frage.

Darstellungen von Sprachen

- Welche verschiedenen Darstellungen gibt es (Grammatiken, Automaten, ...)?
  Wie kann man eine Darstellung in eine andere übersetzen?
- Beschreiben zwei Darstellungen die gleiche Sprache?
  Beschreibt eine Darstellung die leere Sprache?
- Wie kann eine Darstellung vereinfacht werden?

Eigenschaften von Sprachen

- Was passiert, wenn man Operationen anwendet, um neue Sprachen zu erzeugen?

Beispiel:
Wenn $\mathbf{L_1}$ und $\mathbf{L_2}$ regulär sind, wie ist es dann mit $\mathbf{L_1} \cap \mathbf{L_2}$?

$\leadsto$ sogenannte Abschlusseigenschaften

Vorschau

Plan der nächsten Vorlesungen:

Reguläre Sprachen

- endliche Automaten
- reguläre Ausdrücke
- Eigenschaften regulärer Sprachen

Kontextfreie Sprachen

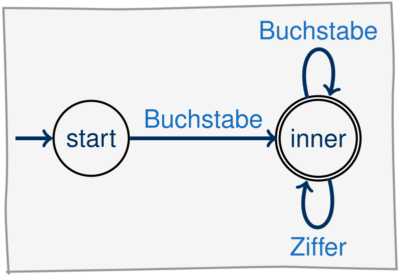
- Normalformen kontextfreier Grammatiken
- Kellerautomaten
- Eigenschaften kontextfreier Sprachen

Endliche Automaten

Beispiel

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

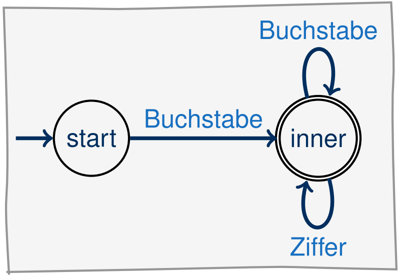
Darstellung als endlicher Automat:



- Automat beginnt im Startzustand → start
- Zustandswechsel gemäß Pfeilen
- Kein passender Pfeil für Symbol?
  „`return false`“
- Keine weiteren Symbole?
  „`return true`“ in Endzustand inner
  „`return false`“ falls in Zustand start

Worterkennung mit Automaten

Ein Automat kann eine Sprache erkennen, indem er Wörter akzeptiert oder ablehnt:



Wort	**Zustandsfolge**	**Ergebnis**
`utf8`	start inner inner inner inner	akzeptiert
`C++`	start inner ?	abgelehnt (fehlender Übergang)
ϵ	start	abgelehnt (kein Endzustand)

Deterministische Endliche Automaten

Die graphische Darstellung von Automaten ist anschaulich, aber nicht immer praktisch. Formal definieren wir Automaten wie folgt:

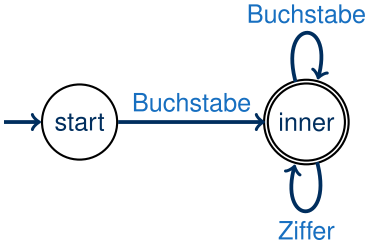
Ein deterministischer endlicher Automat (international: „DFA“) $\mathcal{M}$ ist ein Tupel $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F\rangle$ mit den folgenden Bestandteilen:

- Q: endliche Menge von Zuständen
- Σ: Alphabet
- δ: Übergangsfunktion, eine partielle* Funktion $Q \times \Sigma \to Q$
- q_0: Startzustand $q_0 \in Q$
- F: Menge von Endzuständen $F \subseteq Q$

(* d.h. manche Übergänge sind undefiniert, wie im Beispiel)

Notation: Wir schreiben statt $\delta(q, \mathtt{a}) = q'$ auch $q \xrightarrow{\mathtt{a}} q'$.

Beispiel



Endlicher Automat $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ mit

- $Q = \{\text{start}, \text{inner}\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}, \ldots\}$ (in Zeichnung unterspezifiziert)
- δ ist wie folgt definiert:
  $\delta(\text{start}, x) = \text{inner}$ für alle Buchstaben x
  $\delta(\text{inner}, y) = \text{inner}$ für alle Buchstaben und Ziffern y
  $\delta(q, x)$ undefiniert für alle anderen Fälle
- $q_0 = \text{start}$
- $F = \{\text{inner}\}$

Ist mein Computer ein DFA?

Beobachtungen:

- Computer haben endlich viel Speicher, können also nur endlich viele Zustände einnehmen
- Programme verändern Speicher nach festen Regeln, die man als Übergangsfunktion auffassen könnte

Sind alle Computer DFAs?

Jein.

- Zustandsmenge und Übergangsfunktion extrem groß
  $\rightsquigarrow$ keine praktisch nützliche Beschreibung für ganze Computer
- Computerprogramme verhalten sich systematisch, unabhängig von der Speichergröße
  $\rightsquigarrow$ DFA-Übergänge können diese Systematik nicht gut abbilden

Die Sprache eines DFA

Für einen DFA $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F\rangle$ erweitern wir $\delta : Q \times \Sigma \to Q$ zu einer Übergangsfunktion $\delta : Q \times \Sigma^* \to Q$ auf Wörtern:

Für einen Zustand $q \in Q$ und ein Wort $w \in \Sigma^*$ sei $\delta(q, w)$ der eindeutig bestimmte Zustand, den man erreicht, wenn man ausgehend von q das Wort w einliest:

- $\delta(q, \epsilon) = q$ (Fall $w = \epsilon$)
- $\delta(q, \mathtt{a}v) = \delta(\delta(q, \mathtt{a}), v)$ (Fall $w = \mathtt{a}v$)

$\delta(q, w)$ ist undefiniert wenn einer der nötigen Übergänge undefiniert ist.

Die Sprache eines DFA kann nun formal definiert werden:

Die Sprache eines DFA $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F\rangle$ ist die Menge

$$\mathbf{L}(\mathcal{M}) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}.$$

Alternative Definition: totale Übergänge

Wir können DFAs mit partieller Übergangsfunktion in DFAs mit totaler Übergangsfunktion umwandeln:

Eingabe: DFA $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$
Ausgabe: DFA $\mathcal{M}_{\text{total}} = \langle Q', \Sigma, \delta', q_0, F \rangle$

- füge einen neuen Fangzustand q_f ein: $Q' := Q \cup \{q_f\}$
- für alle Zustände $q \in Q'$ und Symbole $\mathtt{a} \in \Sigma$:

$$\delta'(q, \mathtt{a}) := \begin{cases} \delta(q, \mathtt{a}) & \text{falls } \delta(q, \mathtt{a}) \text{ definiert ist} \\ q_f & \text{falls } \delta(q, \mathtt{a}) \text{ nicht definiert ist} \end{cases}$$

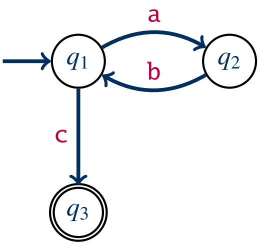
Anmerkung: laut dieser Definition gilt insbesondere $\delta'(q_f, \mathtt{a}) = q_f$ für alle $\mathtt{a} \in \Sigma$.
$\leadsto$ wird beim Lesen eines Wortes q_f erreicht, dann kann es nicht mehr akzeptiert werden (da $q_f \notin F$)

Totale Übergänge (2)

Satz: $\mathcal{M}_{\text{total}} = \langle Q', \Sigma, \delta', q_0, F\rangle$ hat eine totale Übergangsfunktion und akzeptiert die selbe Sprache wie $\mathcal{M}$, d.h. $\mathbf{L}(\mathcal{M}_{\text{total}}) = \mathbf{L}(\mathcal{M})$.

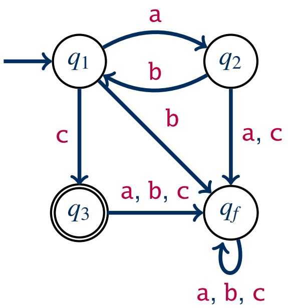
Beispiel: Wir betrachten das Alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$

DFA $\mathcal{M}$ mit partieller Übergangsfunktion:



$\mathbf{L}(\mathcal{M}) = (\mathtt{ab})^*\mathtt{c}$

DFA $\mathcal{M}_{\text{total}}$ mit totaler Übergangsfunktion:



Die Sprachen endlicher Automaten

Idee: Ein Typ von Automaten definiert eine Klasse von Sprachen
Bsp.: „Die Klasse aller Sprachen, die irgendein DFA erkennen kann“

Eine Alternative zur Chomsky-Hierarchie?

Es zeigt sich: mit DFAs erhält man keine neue Sprachklasse!

Satz: Die Klasse der Sprachen, die durch einen DFA erkannt werden können, ist genau die Klasse der regulären Sprachen.

Konsequenzen:

- DFAs können nur sehr einfache Sprachen akzeptieren
- Typ-3-Sprachen sind eine „natürliche“ Klasse: sowohl in der Welt der Grammatiken als auch in der Welt der Automaten

Von DFAs zu regulären Grammatiken

Satz: Die Klasse der Sprachen, die durch einen DFA erkannt werden können, ist genau die Klasse der regulären Sprachen.

Beweis: Eine Richtung dieser Behauptung ist leicht zu sehen:

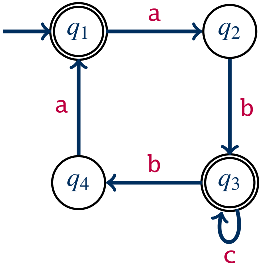
Für jeden DFA $\mathcal{M}$ gibt es eine reguläre Grammatik $G_{\mathcal{M}}$, welche die selbe Sprache erzeugt (d.h., $\mathbf{L}(\mathcal{M}) = \mathbf{L}(G_{\mathcal{M}})$).

Für $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F\rangle$ ergibt sich $G_{\mathcal{M}} = \langle V, \Sigma, P, S\rangle$ wie folgt:

- $V := Q$
- $S := q_0$
- P besteht aus den folgenden Produktionsregeln:

$$\begin{array}{lll} q & \to \mathtt{a}q' & \text{falls } \delta(q, \mathtt{a}) = q' \\ q & \to \mathtt{a} & \text{falls } \delta(q, \mathtt{a}) \in F \\ q_0 & \to \epsilon & \text{falls } q_0 \in F \end{array}$$

Beispiel



Für $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$ ergibt sich $G_{\mathcal{M}} = \langle V, \Sigma, P, S \rangle$ wie folgt:

- $V := Q$
- $S := q_0$
- P besteht aus den folgenden Produktionsregeln:

$$\begin{array}{lll} q & \to \mathtt{a}q' & \text{falls } \delta(q,\mathtt{a}) = q' \\ q & \to \mathtt{a} & \text{falls } \delta(q,\mathtt{a}) \in F \\ q_0 & \to \epsilon & \text{falls } q_0 \in F \end{array}$$

Produktionsregeln der entsprechenden Grammatik:

$q_1 \to \mathtt{a}q_2$ $\quad q_2 \to \mathtt{b}q_3$ $\quad q_3 \to \mathtt{c}q_3$ $\quad q_3 \to \mathtt{b}q_4$ $\quad q_4 \to \mathtt{a}q_1$

$q_2 \to \mathtt{b}$ $\quad q_3 \to \mathtt{c}$ $\quad q_4 \to \mathtt{a}$

$q_1 \to \epsilon$

DFAs akzeptieren reguläre Sprachen: Beweis

Satz: Die Klasse der Sprachen, die durch einen DFA erkannt werden können, ist genau die Klasse der regulären Sprachen.

Beweis (Fortsetzung): Wir müssen noch die Korrektheit des angegebenen Verfahrens zeigen, also dass $\mathbf{L}(\mathcal{M}) = \mathbf{L}(G_{\mathcal{M}})$.

Dazu beweisen wir beide Richtungen einzeln:

„$\subseteq$“ Jedes von $\mathcal{M}$ akzeptierte Wort kann von $G_{\mathcal{M}}$ erzeugt werden.

„$\supseteq$“ Jedes von $G_{\mathcal{M}}$ erzeugte Wort kann von $\mathcal{M}$ akzeptiert werden.

$\mathbf{L}(\mathcal{M}) \subseteq \mathbf{L}(G_\mathcal{M})$

Wir betrachten ein beliebiges Wort $w \in \mathbf{L}(\mathcal{M})$.

Falls $w = \epsilon$, dann ist $q_0 \in F$.

- Dann hat $G_\mathcal{M}$ eine Regel $q_0 \to \epsilon$
- Also ist $w \in \mathbf{L}(G_\mathcal{M})$

Falls $w = \mathtt{a}_1 \cdots \mathtt{a}_n$ mit $n \geq 1$, dann gilt $\delta(q_0, w) \in F$.

- Dann gibt es Übergänge $q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \ldots \xrightarrow{\mathtt{a}_n} q_n$ mit $q_n \in F$
- Dann hat $G_\mathcal{M}$ die Regeln:

$$q_0 \to \mathtt{a}_1 q_1 \qquad q_1 \to \mathtt{a}_2 q_2 \qquad \ldots \qquad q_{n-1} \to \mathtt{a}_n$$

- Durch Anwendung dieser Regeln kann man ableiten:

$$q_0 \Rightarrow \mathtt{a}_1 q_1 \Rightarrow \mathtt{a}_1 \mathtt{a}_2 q_2 \Rightarrow \ldots \Rightarrow \mathtt{a}_1 \mathtt{a}_2 \cdots \mathtt{a}_{n-1} q_{n-1} \Rightarrow \mathtt{a}_1 \mathtt{a}_2 \cdots \mathtt{a}_{n-1} \mathtt{a}_n$$

- Also ist $w \in \mathbf{L}(G_\mathcal{M})$.

Somit gilt $w \in \mathbf{L}(G_\mathcal{M})$ für beliebige $w \in \mathbf{L}(\mathcal{M})$, d.h. $\mathbf{L}(\mathcal{M}) \subseteq \mathbf{L}(G_\mathcal{M})$

$\mathbf{L}(\mathcal{M}) \supseteq \mathbf{L}(G_\mathcal{M})$

$G_\mathcal{M}$ hat drei Typen von Regeln: $q \to \mathtt{a}q'$, $q \to \mathtt{a}$ und $q_0 \to \epsilon$

Für ein Wort $w = \mathtt{a}_1 \cdots \mathtt{a}_n$ sind zwei Arten von Ableitungen denkbar:

(1) $q_0 \Rightarrow \mathtt{a}_1 q_1 \Rightarrow \ldots \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_{n-1} q_{n-1} \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_{n-1}\mathtt{a}_n$

(2) $q_0 \Rightarrow \mathtt{a}_1 q_1 \Rightarrow \ldots \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_{n-1} q_{n-1} \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_{n-1}\mathtt{a}_n q_n \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_n$

In Fall (1) wurden Regeln der folgenden Form angewendet:

$$q_0 \to \mathtt{a}_1 q_1 \qquad q_1 \to \mathtt{a}_2 q_2 \qquad \ldots \qquad q_{n-1} \to \mathtt{a}_n$$

Also hat $\mathcal{M}$ die folgenden Übergänge:

$$q_0 \xrightarrow{\mathtt{a}_1} q_1 \qquad q_1 \xrightarrow{\mathtt{a}_2} q_2 \qquad \ldots \qquad q_{n-1} \xrightarrow{\mathtt{a}_n} q_n$$

wobei $q_n \in F$ ein Endzustand ist.

Also gilt $\delta(q_0, \mathtt{a}_1 \cdots \mathtt{a}_n) = q_n \in F$ und $\mathcal{M}$ akzeptiert das Wort w.

$\mathbf{L}(\mathcal{M}) \supseteq \mathbf{L}(G_\mathcal{M})$

$G_\mathcal{M}$ hat drei Typen von Regeln: $q \to \mathtt{a}q'$, $q \to \mathtt{a}$ und $q_0 \to \epsilon$

Für ein Wort $w = \mathtt{a}_1 \cdots \mathtt{a}_n$ sind zwei Arten von Ableitungen denkbar:

(1) $q_0 \Rightarrow \mathtt{a}_1 q_1 \Rightarrow \ldots \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_{n-1} q_{n-1} \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_{n-1} \mathtt{a}_n$

(2) $q_0 \Rightarrow \mathtt{a}_1 q_1 \Rightarrow \ldots \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_{n-1} q_{n-1} \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_{n-1} \mathtt{a}_n q_n \Rightarrow \mathtt{a}_1 \cdots \mathtt{a}_n$

In Fall (2) wurden Regeln der folgenden Form angewendet:

$$q_0 \to \mathtt{a}_1 q_1 \qquad q_1 \to \mathtt{a}_2 q_2 \qquad \ldots \qquad q_{n-1} \to \mathtt{a}_n q_n \qquad q_n \to \epsilon$$

Also ist $q_n = q_0$ mit $q_0 \in F$ und $\mathcal{M}$ hat die folgenden Übergänge:

$$q_0 \xrightarrow{\mathtt{a}_1} q_1 \qquad q_1 \xrightarrow{\mathtt{a}_2} q_2 \qquad \ldots \qquad q_{n-1} \xrightarrow{\mathtt{a}_n} q_0$$

Also gilt $\delta(q_0, \mathtt{a}_1 \cdots \mathtt{a}_n) = q_0 \in F$ und $\mathcal{M}$ akzeptiert das Wort w.

Zusammengefasst gilt somit $w \in \mathbf{L}(\mathcal{M})$ für beliebige $w \in \mathbf{L}(G_\mathcal{M})$,
d.h. $\mathbf{L}(\mathcal{M}) \supseteq \mathbf{L}(G_\mathcal{M})$ $\square$

Reguläre Grammatiken und DFAs

Wir haben bisher gezeigt:

Jede von DFA erkannte Sprache ist regulär.

Für die Umkehrung müsste man reguläre Grammatiken in DFAs übersetzen.

Kann man die Übersetzung nicht einfach umdrehen?

- Für jede reguläre Regel $A \to \mathtt{a}B$ definieren wir $\delta(A, \mathtt{a}) = B$
- Für jede reguläre Regel $A \to \mathtt{a}$ definieren wir $\delta(A, \mathtt{a}) = C$ mit $C \in F$

Warum funktioniert das nicht?

Zusammenfassung und Ausblick

Das Wortproblem ist eine von mehreren wichtigen Fragestellungen zu formalen Sprachen

Deterministische endliche Automaten (DFAs) können mit partieller oder totaler Übergangsfunktion definiert werden

DFAs lösen das Wortproblem für reguläre Sprachen
(Bisher gezeigt: jede durch DFAs erkennbare Sprache ist regulär)

Nächste Themen:

- Noch zu zeigen: Alle regulären Sprachen sind durch DFAs erkennbar
- Nichtdeterministische Automaten
- Gibt es noch mehr Darstellungsformen für reguläre Sprachen?

Bildrechte

Folie 4:

- Portrait John Backus: Wikipedia-Nutzer **PIerre.Lescanne**, CC-By-SA 4.0
- Portrait Peter Naur: Wikipedia-Nutzer **Eriktj**, CC-By-SA 3.0
- Portrait Niklaus Wirth: Wikipedia-Nutzer **Tyomitch**, used with permission of copyright holder
- Foto Büste Panini: Wikipedia-Nutzer **Jameela P.**, CC-By-SA 4.0