Algorithmen und Datenstrukturen

Theoretische Grundlagen Informatik

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Algorithmen und Datenstrukturen

Alphabete, Wörter und Sprachen

Prof. Dr. rer. nat. Jörg Striegnitz

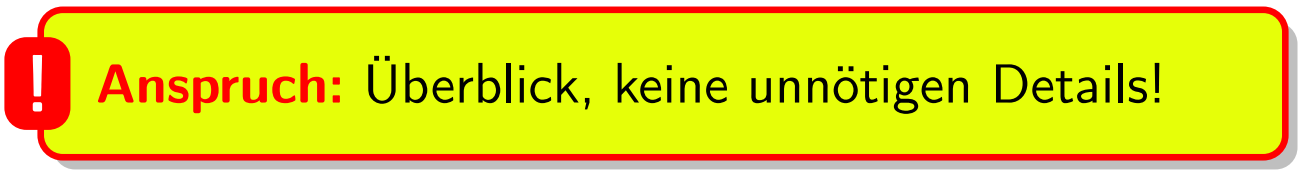
Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Was und wozu?

- Kleiner, "praktischer" Exkurs in die Theoretische Informatik
  - Formale Sprachen, Automaten
  - Berechenbarkeit und Komplexität

- Warum?
  - Klärung zentraler Begriffe (z.B. Determinismus, Nichtdeterminismus)
  - Endliche Automaten in Algorithmen (z.B. Textsuche)
  - Einführung von Standard-Problemstellungen
  - **Erste Entwurfstechniken für Datenstrukturen und Algorithmen!**



Ziel

- Computer verarbeiten Bitfolgen
  Daher: Computerprogramm realisiert Funktion $\mathbb{B} \to \mathbb{B}$, wobei $\mathbb{B}$ die Menge aller Bitfolgen ist.

- Aus Anwendersicht sind Ein- und Ausgaben **komplexere Objekte**; z.B.
  - Musikdaten
  - Videos und Bilder
  - Graphen

- **Ziel**: Formale Darstellung der Ein- / Ausgaben eines Computers - Repräsentation von (komplexen) Daten

- **Beachte**: Computerprogramme sind ebenfalls Daten!

Alphabete

Definition 1.12 (Alphabet)

Eine endliche, nicht leere Menge Σ heißt **Alphabet**. Die Elemente von Σ heißen **Buchstaben** (Zeichen, Symbole).

Beachte:

- Ein Alphabet ist **endlich**!
- **Konsequenz**: Die Menge der natürlichen Zahlen $\mathbb{N}$ ist als Alphabet nicht zugelassen!

Alphabete - Beispiele

Beispiel 1.5 (Alphabete)

- $\Sigma_{Bool} = \{0, 1\}$ - das Boole'sche Alphabet
  auf diesem Alphabet operiert ein Computer
- $\Sigma_{RGB} = \{\underline{\text{rot}}, \underline{\text{grün}}, \underline{\text{blau}}\}$ - das Alphabet der Grundfarben
  Beachte: ein Symbol besteht hier aus mehreren Zeichen. Um dies zu verdeutlichen unterstreicht man solche Symbole
- $\Sigma_{Card} = \{\diamondsuit, \heartsuit, \clubsuit, \spadesuit\}$ - Spielkartensymbole
- $\Sigma_{lat} = \{a, b, \ldots, z\}$ - Kleinbuchstaben des lateinischen Alphabets
- $\Sigma_{Tastatur} = \{A, B, \ldots, Z, \sqcup, <, >, (,)\} \cup \Sigma_{lat}$ - Zeichen einer Computertastatur
  $\sqcup$ ist das Leerzeichen - auch „*Space*" genannt. $\Sigma_{Tastatur}$ benutzen wir zur Darstellung von Computerprogrammen.
- $\Sigma_{logic} = \{0, 1, x, (,), \vee, \wedge, \neg\}$ - Darstellung von Formeln der Aussagenlogik

Wörter

Ein **Wort** ist eine Folge von Buchstaben.

Aus gutem Grund wählen wir aber eine **induktive Definition**:

Definition 1.13 (Wort)

Sei Σ ein Alphabet. Die Menge Σ^* aller **Wörter über** Σ ist induktiv definiert durch

- $\epsilon \in \Sigma^*$ - das **leere Wort** ist ein Wort (* Induktionsanfang *)
- falls $w \in \Sigma^*$ und $a \in \Sigma$, so ist $w \cdot a \in \Sigma^*$- die **Konkatenation** von w und a ist ein Wort (* Induktionsschritt *)

Konvention und **Anmerkungen**:

- Konkatenation ist **links-assoziativ**: $a_1 \cdot a_2 \cdot a_3 = (a_1 \cdot a_2) \cdot a_3$
- ε ist **linksneutrales Element**: $\varepsilon \cdot a = a$
- Anstelle von $a_1 \cdot a_2 \cdot a_3 \cdot \cdots \cdot a_n$ schreiben wir kurz $a_1 a_2 \cdots a_n$

  war a_1 das leere Wort, so können wir es weglassen

! Um Verwechslungen vorzubeugen, gehen wir fortan davon aus, dass das Symbol ε in keinem Alphabet Σ verwendet wird.

Wörter - Beispiele

Beispiel 1.6 (Wörter)

- Das Wort

  $$011001$$

  ist ein Wort über Σ_{Bool}, Σ_{logic} und $\Sigma_{Tastatur}$.
- Das Wort

  $$\texttt{while}\, a < b\ a = a - b;$$

  ist ein Wort über $\Sigma_{Tastatur}$
- ε ist ein Wort über einem beliebigem Alphabet.

Vorsicht!

In der Informatik hat der Begriff „Wort" eine andere Bedeutung als in natürlichen Sprachen. In der Informatik wird der Inhalt eines Buches als ein Wort über dem Alphabet $\Sigma_{Tastatur}$ angesehen - genauso, wie ein JAVA-Programm in der Theorie der formalen Sprachen als ein Wort über dem Alphabet $\Sigma_{Tastatur}$ behandelt wird.

Wörter - Wirkung der induktiven Definition

Mithilfe der induktiven Definition wird ganz Σ^* beschrieben:

Beispiel 1.7 (Erzeugen von Σ^*_{Bool} mithilfe der induktiven Definition)

Betrachte $\Sigma_{Bool} = \{0, 1\}$ - was enthält Σ^*_{Bool}?

- Gemäß **Induktionsanfang** gilt $\{\varepsilon\} \subseteq \Sigma^*_{Bool}$
- Durch Anwendung des **Induktionsschritts** können wir unser Wissen über Σ^*_{Bool} sukzessive erweitern:
  - $\varepsilon \in \Sigma^*_{Bool}$ und $0 \in \Sigma_{Bool} \Rightarrow \varepsilon \bullet 0 \in \Sigma^*_{Bool}$
  - $\varepsilon \in \Sigma^*_{Bool}$ und $1 \in \Sigma_{Bool} \Rightarrow \varepsilon \bullet 1 \in \Sigma^*_{Bool}$

  also gilt $\{\varepsilon, \varepsilon \cdot 0, \varepsilon \cdot 1\} \subseteq \Sigma^*_{Bool}$ und wir können fortfahren:
  - $\varepsilon \bullet 0 \in \Sigma^*_{Bool}$ und $0 \in \Sigma_{Bool} \Rightarrow \varepsilon \bullet 0 \bullet 0 \in \Sigma^*_{Bool}$
  - $\varepsilon \bullet 1 \in \Sigma^*_{Bool}$ und $0 \in \Sigma_{Bool} \Rightarrow \varepsilon \bullet 1 \bullet 0 \in \Sigma^*_{Bool}$
  - usf.

  es folgt $\{\varepsilon, \varepsilon \cdot 0, \varepsilon \cdot 1, \varepsilon \cdot 0 \cdot 0, \varepsilon \cdot 1 \cdot 0, \varepsilon \cdot 0 \cdot 1, \varepsilon \cdot 1 \cdot 1\} \subseteq \Sigma^*_{Bool}$ usw. ....
- **Offenbar gilt**: Ist ein Wort in Σ^*_{Bool} enthalten, so wird es irgendwann erzeugt!

- **Systematisches Vorgehen (Algorithmus) zum Aufzählen aller Wörter?**

Wörter - Analyse mit Hilfe der induktiven Definition

Die induktive Definition kann man auch zum Nachweis verwenden, dass ein Wort in Σ^* enthalten ist:

Beispiel 1.8 (Nachweis von $w \in \Sigma^*$)

Betrachte Σ^*_{Bool} und $v = 01$. Gilt $v \in \Sigma^*_{Bool}$?

1. Da $01 \neq \varepsilon$ kann 01 nur durch Anwendung des Induktionsschritts entstanden sein; konkret also durch $0 \cdot 1 = 01$. Diese Rechnung ist nur erlaubt, wenn $0 \in \Sigma^*_{Bool}$ und $1 \in \Sigma_{Bool}$. Zweiteres gilt offenbar, wir müssen also noch $0 \in \Sigma^*_{Bool}$ sicherstellen.
2. Da $0 \neq \varepsilon$ kann auch 0 nur durch Anwendung des Induktionsschritts entstanden sein; konkret also durch $\varepsilon \cdot 0 = 0$. Diese Rechnung ist nur erlaubt, wenn $\varepsilon \in \Sigma^*_{Bool}$ und $0 \in \Sigma_{Bool}$. Ersteres gilt gem. Induktionsanfang und $0 \in \Sigma_{Bool}$ ist ebenfalls korrekt.
3. **Fazit**: $0 \in \Sigma^*_{Bool}$, damit ist $0 \cdot 1$ eine erlaubte Operation und insgesamt gilt $01 \in \Sigma^*_{Bool}$.

Beachte:

- In den ersten beiden Schritten wird w gewissermaßen zerlegt (in w und a, so dass $w \cdot a = v$), bis wir mit $w = \varepsilon$ am Induktionsanfang angelangt sind - die Zerlegung ist jeweils eindeutig!
- Dann läuft die Argumentation »Rückwärts« (Schritt 3)

Induktive Definitionen

- zur endlichen Beschreibungen von unendlichen Mengen (z.B. enthält Σ^*_{Bool} unendlich viele Objekte)
- erleichtern die Definition von Funktionen über diesen Mengen (meist rekursiv)
- öffnen Zugang zur Beweistechnik der vollständigen Induktion (dazu später mehr)
- helfen bei der systematischen Dekomposition von komplexen Objekten in weniger komplexe
- findet man in der Informatik an unterschiedlichsten Stellen, etwa bei der Definition von Programmiersprachen oder Datenstrukturen (z.B. Listen und Bäume)

Wortlänge

Definition 1.14 (Wortlänge)

Seien Σ ein Alphabet und $w \in \Sigma^*$. Die **Wortlänge** $|w|$ von w ist induktiv definiert durch

1. $|\varepsilon| = 0$
2. Falls $w = v \cdot a$, so $|w| = |v| + 1$ (für $v \in \Sigma^*$ und $a \in \Sigma$)

Mit $|w|_a$ bezeichnen wir die Anzahl der Vorkommen des Buchstabens a im Wort w.

Beachte:

- Die Definition von $|\cdot|$ geschieht »entlang« der induktiven Definition von Σ^*

Beispiel: Berechnung der Wortlänge

Beispiel 1.9 (Anwendung einer induktiv definierten Operation)

Betrachte $w = 010 \in \Sigma^*_{Bool}$. Für $|010|$ erhalten wir:

- $|010| = |01| + 1$, denn $\underbrace{01}_{v} \cdot \underbrace{0}_{a} = 010$
- $|01| = |0| + 1$, denn $0 \cdot 1 = 01$
- $|0| = |\varepsilon| + 1$, denn $\varepsilon \cdot 0 = 0$
- $|\varepsilon| = 0$, per Definition

Es ergibt sich also:

$$|010| = |01| + 1 = |0| + 1 + 1 = |\varepsilon| + 1 + 1 + 1 = 0 + 1 + 1 + 1 = 3$$

Offenbar ist $|010|_1 = 1$ und $|010|_0 = 2$.

Beachte:

- Auch hier wird das zu untersuchende Wort sukzessive in seine eindeutigen Bestandteile zerlegt

Die Funktionen `head` und `tail`

Definieren wir ein wenig komplexere Operationen:

- Für das leere Wort liefern head und tail jeweils das leere Wort.
- Ansonsten liefert head das erste Symbol eines Wortes, während tail das erste Symbol entfernt.

Definition 1.15 (`head` und `tail`)

Seien Σ ein Alphabet und $w \in \Sigma^*$. Dann ist die Funktion $\texttt{head} : \Sigma^* \to \Sigma \cup \{\varepsilon\}$ definiert durch

- $\texttt{head}(\varepsilon) = \varepsilon$
- $\texttt{head}(\varepsilon \cdot a) = a$
- $\texttt{head}(v \cdot a) = \texttt{head}(v)$

ferner ist die Funktion $\texttt{tail} : \Sigma^* \to \Sigma^*$ definiert durch

- $\texttt{tail}(\varepsilon) = \varepsilon$
- $\texttt{tail}(\varepsilon \cdot a) = \varepsilon$
- $\texttt{tail}(v \cdot a) = \texttt{tail}(v) \cdot a$

Für jeweils geeignete $v \in \Sigma^* - \{\varepsilon\}$ und $a \in \Sigma$.

Konkatenation von Wörtern

- **Bisher**: '$\cdot$' verbindet (konkateniert) ein Wort mit einem Buchstaben
- **Jetzt**: Konkatenation von Wörtern
  festzulegen: was soll passieren, wenn rechter Operand Wort ist - für Symbole greift Standard-Definition

Definition 1.16 (Konkatenation)

Sei Σ ein Alphabet, dann erweitern wir $\cdot$ zu $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ wie folgt:

- $w \cdot \varepsilon = w$
- $w \cdot (v \cdot a) = (w \cdot v) \cdot a$

für $\{v, w\} \subseteq \Sigma^*$ und $a \in \Sigma$.

Beobachtung 1.1

- Konkatenation ist offenbar **assoziativ**: $\forall u, v, w \in \Sigma^*$ gilt:

$$(u \cdot v) \cdot w = u \cdot (v \cdot w)$$

- aber nur für einelementige Alphabete kommutativ.
- $(\Sigma^*, \cdot, \varepsilon)$ ist ein **Monoid** (eine Halbgruppe mit neutralem Element - im Vergleich zur Gruppe fehlt es an einem inversen Element)

Iteration

Definition 1.17 (Iteration)

Sei Σ ein Alphabet. Für alle $w \in \Sigma^*$ und $i \in \mathbb{N}$ definieren wir die ***i*-te Iteration** w^i von w als

1. $w^0 = \varepsilon$
2. $w^i = w \cdot w^{i-1}$

- Zur Erinnerung: $(\Sigma, \cdot, \varepsilon)$ ist ein Monoid
- **Beachte**: Analogie der Definition der Iteration und der des Potenzierens über den natürlichen Zahlen mit Null!
  - n^0 liefert ebenfalls neutrales Element, die 1
  - $n^k = n \cdot n^{k-1}$
- Iteration erlaubt manchmal kürzere Schreibweisen; z.B.
  - 0^3 anstelle von 000
  - $(abc)^2 b^3 cda^5$ anstelle von $abcabcbbbcdaaaaa$

Präfix, Suffix und Teilwort

Definition 1.18 (Präfix, Suffix und Teilwort)

Sei Σ ein Alphabet und seien $v, w \in \Sigma^*$ Wörter.

- v heißt **Teilwort** von w gdw. $\exists s, t \in \Sigma^* : w = svt$.
- v heißt **Suffix** von w gdw. $\exists s \in \Sigma^* : w = sv$ - wir schreiben $v \sqsupseteq w$.
- v heißt **Präfix** von w gdw. $\exists t \in \Sigma^* : w = vt$ - wir schreiben $v \sqsubseteq w$
- v heißt echtes Teilwort (Präfix/Suffix) von w gdw. $v \neq \varepsilon \wedge v \neq w$ und v Teilwort (Präfix/Suffix) von w ist. Wir schreiben dann $v \sqsubset w$ bzw. $v \sqsupset w$.

Beachte:

- Jedes Wort ist Präfix, Suffix und Teilwort seiner selbst
- Das leere Wort ε ist Präfix, Suffix und Teilwort eines jeden Wortes

Aufzählungsalgorithmus für Σ^*

Aufzählunsalgorithmus für Σ^*

procedure printAllWords(Σ : Alphabet, $maxLen : \mathbb{N}$)
    $\Sigma^*_{alt} = \{\varepsilon\}$; $i = 0$
    print ε
    while $(i < maxLen)$
        $\Sigma_{help} = \Sigma$; $\Sigma^*_{neu} = \varnothing$
        while$\left(\Sigma_{help} \neq \varnothing\right)$ /* Alle Symbole durchgehen */
            wähle **beliebiges** $a \in \Sigma_{help}$ und setze $\Sigma_{help} = \Sigma_{help} - \{a\}$
            $\Sigma^*_{help} = \Sigma^*_{alt}$
            while $\left(\Sigma^*_{help} \neq \varnothing\right)$ /* Wörter der vorherigen Iteration durchgehen */
                wähle **beliebiges** $v \in \Sigma^*_{help}$ und setze $\Sigma^*_{help} = \Sigma^*_{help} - \{v\}$
                print $v{\cdot}a$ und setze $\Sigma^*_{neu} = \Sigma^*_{neu} \cup \{v \cdot a\}$
            end while
        end while
        $\Sigma^*_{alt} = \Sigma^*_{neu}$; $i = i + 1$
    end while

Beobachtungen:

- Algorithmus ist korrekt (listet alle Wörter $w \in \Sigma^*$ mit $|w| \leq maxLen$)
- Wörter werden in aufsteigender Länge ausgegeben
- Reihenfolge von Wörtern gleicher Länge nicht festgelegt (freie Wahl von a und v)
  $\Rightarrow$ Verhalten des Algorithmus ist **nicht deterministisch**!

Kanonische Ordnung

- In der Praxis möchte man **deterministische** Algorithmen!
- Mengen sind nicht geordnet (z.B. $\{A, B, C\} = \{A, C, B\} = \{B, A, C\}$)
- **Folge**: Reihenfolge der Wahl eines Elements unterliegt Zufall
- **Lösung**: $\Rightarrow$ Sortiere Buchstaben und leite Ordnung auf Wörtern ab:

Definition 1.19 (Kanonische Ordnung)

Sei $\Sigma = \{a_1, \cdots, a_n\}$ ein Alphabet mit $n \geq 1$ und $<\subseteq \Sigma \times \Sigma$ eine Ordnung auf Σ mit $a_1 < a_2 < \cdots < a_n$. Wir definieren die **kanonische Ordnung** auf Σ^* wie folgt:

$$\begin{aligned} u < v \quad \text{gdw.} \quad & |u| < |v| \\ \vee \quad & |u| = |v| \ \wedge \ u = w \cdot a_i \cdot u' \ \wedge \ v = w \cdot a_j \cdot v' \\ & \text{für beliebige } w, u, u', v, v' \in \Sigma^* \text{und } i < j \end{aligned}$$

Sprachen

Definition 1.20 (Sprache)

Sei Σ ein Alphabet. Eine **Sprache** L über Σ ist eine Teilmenge von Σ^* ($L \subseteq \Sigma^*$). $\mathcal{P}(\Sigma^*) = \{L \mid L \subseteq \Sigma^*\}$, die Potenzmenge von Σ^*, ist die Menge aller Sprachen über Σ. Ferner sind $L_\varnothing = \varnothing$ und ist $L_\varepsilon = \{\varepsilon\}$.

Beachte:

- Die Mathematik stellt sehr mächtige Mechanismen zur Definition von Mengen zur Verfügung!

Beispiele für Sprachen

Beispiel 1.10 (Sprachen)

- **Endliche Sprachen**
  - $L_1 = \varnothing$
  - $L_2 = \{\varepsilon\}$
  - $L_3 = \{\varepsilon, a, b, ab, ba\}$
- **Unendliche Sprachen**
  - $L_4 = \{0^p \mid p \text{ ist Primzahl}\}$
  - $L_5 = \left\{0^i 1^{2i} 0^i \mid i \in \mathbb{N}\right\}$
- **Beispiele aus der Informatik (unendliche Sprachen)**
  - $L_{Matrix} = \{w \in \Sigma^*_{Bool} \mid w \text{ codiert positiv definite Matrix}\}$
  - $L_{JavaSyn} = \{w \in \Sigma^*_{Tastatur} \mid w \text{ ist syntaktisch korrektes JAVA-Programm}\}$
  - $L_{JavaType} = \{w \in \Sigma^*_{Tastatur} \mid w \text{ ist typ-korrektes JAVA-Programm}\}$
  - $L_{JavaTerm} = \{w \in \Sigma^*_{Tastatur} \mid w \text{ ist terminierendes JAVA-Programm}\}$

Vorschau: Einsatzgebiete für Sprachen

- Sprachen zur Darstellung von **Daten**; z.B.:

$$L_{Music} = \{w \in \Sigma^*_{Bool} \mid w \text{ codiert Musikstück der Band } xy\}$$

  Fokus: **Codierung von Information**.

- Sprachen zur Darstellung von **algorithmischen Problemen**; z.B.:

$$L_{SAT} = \left\{w \in \Sigma^*_{Logic} \mid w \text{ codiert erfüllbare Formel}\right\}$$

  Fokus: **Berechnung der Sprache**.

- Beide Aspekte werden wir später genauer untersuchen.