Alphabete, Wörter, Wortfunktionen

Montag, 30. September 2024 20:55

alphabete_woerter_...

Themen

1. **Alphabete, Wörter, Wortfunktionen**
2. Reguläre Ausdrücke
3. Endliche Automaten
4. Kellerautomaten
5. Grammatiken
6. Berechenbarkeit
7. Turing-Maschinen
8. Primitiv-rekursive und μ-rekursive Funktionen

Einführung

Wörter – in der Informatik spricht man von **Zeichenketten**, engl. **strings** – und darauf aufbauend Sprachen spielen (neben den Zahlen) in der Informatik eine zentrale Rolle.

Dabei sind nicht die Wörter oder Sätze in einem Schreibprogramm gemeint, sondern Zeichenfolgen incl. Leerzeichen zur Beschreibung von

- Programmier- und Skriptsprachen
- Auszeichnungssprachen
- Datenstrukturen
- Konfigurationsdateien
- Shell-Kommandos.

Ein Java-Programm oder ein JSON-Dokument beispielsweise verstehen wir als ein "Wort".

Typen von Sprachen

- Natürliche Sprache
  - Mündliches oder schriftliches Verständigungsmittel zwischen Menschen.
  - Funktioniert nach gewissen (manchmal willkürlichen) Regeln, die oft subjektiv sind.
  - Die Grammatik (Syntax) gibt die Regeln vor, die durch Semantik und Pragmatik ausgefüllt werden.
- Formale Sprache
  - Mathematisches Modell (algebraische Beschreibung) für Sprachen.
  - Kann nur Teilaspekte der natürlichen Sprache abbilden.
- Programmiersprache
  - Schriftliches Verständigungsmittel zwischen Mensch und Computer
  - Basiert auf den formalen Sprachen.
  - Es gibt nur eine eindeutige Interpretation.
  - Formal falsche "Sätze" (Programme, Befehle, ...) können nicht verarbeitet werden.

Bestandteile einer Sprache

- Syntax
  - Rein formale Betrachtung des Aufbaus wie Rechtschreibung und Grammatik.
  - Bei Programmiersprachen die Programmstruktur (Anordnung von Schlüsselwörtern, Bezeichnern, Befehle, Trennzeichen, ...) ohne Berücksichtigung der Bedeutung.
- Semantik
  - Bedeutung der Sätze, Befehle und Programme, d.h. die Lehre von der Beziehung der Zeichen/Wörter zum gemeinten Gegenstand.
  - Die Semantik natürlicher Sprache ist formal schwer zugänglich.
  - Bei Programmiersprachen ist jedoch auch die Semantik festgelegt, damit kein Interpretationsspielraum bestehen darf.
- Pragmatik
  - Die von den Umständen wie Zeit oder Gefühlen abhängige subjektive Bedeutung für den Benutzer.
  - Bei Programmiersprachen: Eignung, Effektivität, Erlernbarkeit, Fehlererkennung.

Wörter / Zeichenketten (1)

Intuitive Definition: Zeichen aus einem Alphabet (= Zeichenvorrat), die hintereinander geschrieben werden, bilden eine Zeichenkette.

Definition

Ein **Alphabet** ist eine endliche Menge Σ von wohldefinierten **Zeichen** (Symbolen, Buchstaben).

Die Zeichen werden in der Regel mit $a, b, c, \ldots$ oder mit $a_1, a_2, \ldots, a_n$ bezeichnet. D.h. $\Sigma = \{a_1, a_2, \ldots, a_n\}$ mit $n \in \mathbb{N}$.

Beispiele für Alphabete:

- $\{0, 1\}$
- $\{0, 1, \ldots, 9\}$
- $\{a, b, \ldots, z\}$
- $\{a, b, \ldots, z, A, B, \ldots, Z\}$
- $\{1, A\}$

- Alphabet: Menge der gültigen Zeichen
- wohldefiniert: Namen gegeben

Wörter / Zeichenketten (2)

Definition

Eine **Zeichenkette** (auch "string", Wort) w über Σ ist eine Folge von Zeichen $w := a_1 a_2 \ldots a_n$ mit $a_i \in \Sigma$ für alle $i \in \{1, \ldots, n\}$.

Ist $n = 0$, enthält w kein Zeichen, d.h. w ist das **leere Wort** und wird mit ϵ bezeichnet.

Zeichenketten werden meist mit Kleinbuchstaben wie p, q, r, s, t, u, v, w, x, y, z bezeichnet.

Hinweise:

- Eine Zeichenkette kann auch als Liste bzw. Array von Zeichen aufgefasst werden (vgl. z.B. Programmiersprache C, dort beginnt die Zählweise aber bei 0).
- Zeichenketten werden nicht mit Hochkommata umgeben (im Gegensatz zu Stringliteralen in Programmiersprachen).

- Wort: Folge von Zeichen über einem definierten Alphabet Σ
  - (Datentyp String)
  - Leere Wort ε
    - Wort, dass aus 0 Zeichen besteht
- Menge von Wörtern: Σ*
  - (Die Elemente sind vom Datentyp String)
- Menge: { }
  - (Datentyp Menge)
- { } ≠ {ε}
  - Leere Menge ≠ Menge mit einem Element, dem leeren Wort

- Die endlich langen Zeichenfolgen, die über einem Alphabet Σ gebildet werden können, heißen Wörter über Σ.
- Wörter entstehen, indem Symbole oder bereits erzeugte Wörter aneinandergereiht (miteinander verkettet, konkateniert) werden.

- Ein Wort schreibt man ohne { }.
- Die Menge alle Wörter über ein Alphabet schreibt man mit { }

Wortmengen

Definition

- Σ^* ist die Menge aller Wörter über Σ (insbesondere $\epsilon \in \Sigma^*$).
- $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$ ist die Menge aller nichtleeren Wörter über Σ.
- Σ^*_n ist die Menge aller Wörter der Länge $n \in \mathbb{N}$ über Σ.

Folgerungen:

- $\Sigma^*_0 = \{\epsilon\}$
- $\Sigma^*_1 = \Sigma$
- $\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^*_k$

Beachte: Ist ein Alphabet Σ nicht leer, dann besitzt Σ^* unendlich viele Wörter.

- Σ*, die Menge aller Wörter, die über dem Alphabet Σ gebildet werden kann, ist wie folgt definiert:
  - Jeder Buchstabe a ∈ Σ ist auch ein Wort über Σ, d.h. a ∈ Σ* .
  - Werden bereits konstruierte Wörter hintereinandergeschrieben, entstehen neue Wörter, d.h. sind v, w ∈ Σ* , dann ist auch ihre Verkettung (Konkatenation) vw ∈ Σ ∗ .
  - ε, das leere Wort, ist ein Wort über (jedem Alphabet) Σ
    - es gilt immer ε ∈ Σ*
    - ε ist ein definiertes Wort ohne „Ausdehnung". Es hat die Eigenschaft: εw = wε = w für alle w ∈ Σ ∗ .

Beispiele

Beispiel

Sei $\Sigma = \{a, b\}$.
Dann ist $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$.

Beispiel

Sei nun $\Sigma = \{0, 1\}$.
Dann ist:

- $\Sigma^*_0 \cup \Sigma^*_2 = \{\epsilon, 00, 01, 10, 11\}$
- $\Sigma^*_3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

Beispiel

Schließlich sei $\Sigma = \{a\}$. Dann

- $\Sigma^*_2 = \{aa\}$
- $\Sigma^* = \{\epsilon, a, aa, aaa, \ldots\}$

- Σ = {a}, Σ* = {ε, a, aa, aaa, …}
- Σ = {b}, Σ* = {b}* = {ε, b, bb, bbb, …}
- {a}* ∪ {b}* = {ε, a, b, aa, bb, …}
- {a, b}* = {ε, a, b, aa, ab, ba, bb, …}
- {a}* ∪ {b}* ⊆ {a, b}*
- {a, b}*$_2$ = {aa, ab, ba, bb}

- Wenn eine Sprache eine leere Menge als Alphabet hat dann ist das leere Wort trotzdem Teil der Sprache
  Σ = {} -> Σ* = {ε}

Konkatenation von Wörtern (1)

Definition

Werden zwei Wörter $p, q \in \Sigma^*$ hintereinandergesetzt, dann ist auch die **Konkatenation** (Verkettung) $pq \in \Sigma^*$.
Sei

- $p = a_1 a_2 \ldots a_n$ mit $n \in \mathbb{N}$, $a_k \in \Sigma$ und
- $q = b_1 b_2 \ldots b_m$ mit $m \in \mathbb{N}$, $b_k \in \Sigma$.

Dann ist $pq = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_m = p \cdot q$.

Theorem

1. *Die Konkatenation ist assoziativ, d.h.* $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.
2. *Das leere Wort* ϵ *ist das Neutralelement der Konkatenation, d.h.* $x \cdot \epsilon = x$.
3. *Die Konkatenation ist i.A. nicht kommutativ, d.h.* $\exists x, y$ *mit* $x \cdot y \neq y \cdot x$.

- pq = p · q

Konkatenation von Wörtern (2)

Hinweise:

- Es gilt $\epsilon w = w\epsilon = \epsilon w \epsilon = w$ für alle $w \in \Sigma^*$.
- Besteht ein Wort aus einer Folge von gleichen Zeichen bzw. gleichen Teilworten, so kann dies abkürzend mit einer Potenzschreibweise notiert werden: $w = \underbrace{a \cdot a \cdot a \cdot \ldots \cdot a}_{n\text{-mal}} = a^n$

Beispiel

- $a^4 = aaaa$
- $b^2 a^3 = bbaaa$
- $(ab)^2 c^3 (de)^3 = ababcccdedede$

Funktion zur Konkatenation: $concat : \Sigma^* \times \Sigma^* \to \Sigma^*$

- $concat(furt, wangen) = furtwangen$
- $concat(furt, concat(wan, gen)) = furtwangen$

- Konkatenation von Wörtern: concat String X String -> String
  $\underset{\Sigma^*}{\parallel}$ $\underset{\Sigma^*}{\parallel}$ $\underset{\Sigma^*}{\parallel}$

$v * w = \overbrace{w_1 \ldots v_2}^{v}\overbrace{w_1 \ldots w_2}^{w}$

- Σ* x Σ* -> Σ*
  Zwei Eingabewerte ergeben wieder ein Wort

Präfix, Infix und Postfix

Definition

Ist $w \in \Sigma^*$ ein Wort der Form $w = xyz$ mit $x, y, z \in \Sigma^*$, dann heißt

- x **Präfix** von w,
- y **Infix** oder **Teilwort** von w und
- z **Postfix** oder **Suffix** von w.

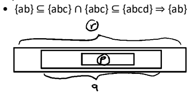
Ist y ein **Teilwort** von w, so schreiben wir auch $y \sqsubseteq w$. y heißt **echtes Teilwort** von w, wenn $y \sqsubseteq w$ und $y \neq w$.

Theorem

Die Teilwort-Relation auf Σ^* hat folgende Eigenschaften:

1. *reflexiv:* $p \sqsubseteq p$
2. *antisymmetrisch:* $p \sqsubseteq q \wedge q \sqsubseteq p \Rightarrow p = q$
3. *transitiv:* $p \sqsubseteq q \wedge q \sqsubseteq r \Rightarrow p \sqsubseteq r$

- x, y, z ∈ Σ*
  - x, y, z jeweils ein Wort
- Präfix, Infix und Postfix sind Teilwörter von w
- Beispiele Präfix, Infix, Postfix:
  - w = abcadε
    Präfix, Infix, Postfix
    w = abcadεε
    Präfix, Infix, Postfix
  - Beispiele sind laut Definition möglich, da Σ* das leer Wort enthält somit ist Infix = ε möglich
- Beispiel zu transitiv:
  - {ab} ⊆ {abc} ∩ {abc} ⊆ {abcd} ⇒ {ab} ⊆ {abcd}



Die Funktion Teilwort ("istTeilwort")

Definition

Das Vorkommen eines Wortes q in einem Wort p kann wie folgt als Funktion $istTeilwort : \Sigma^* \times \Sigma^* \to \mathbb{N}$ definiert werden:

- $istTeilwort(p, q) =$
  $\begin{cases} pos & pos \text{ ist der kleinste Index von } p \text{, ab dem } q \text{ Teilwort von } p \text{ ist} \\ 0 & \text{falls } q \text{ kein Teilwort von } p \text{ ist} \end{cases}$

Beispiel

- $istTeilwort(ababba, abb) = 3$
- $istTeilwort(ababba, aa) = 0$

- Man bekommt zwei Wörter des Alphabets als Eingabe
  - Einmal das Wort und einmal das Teilwort
- Als Rückgabe erhält man den Index ab dem Teilwort

Länge eines Wortes

Definition

Die **Länge eines Wortes** kann durch die Funktion $|\ | : \Sigma^* \to \mathbb{N}$ berechnet werden:

- $|\epsilon| = 0$.
- $|wa| = |w| + 1$ für $w \in \Sigma^*$ und $a \in \Sigma$.

Beispiel mit $\Sigma = \{a, b, c\}$

- $|b| = |\epsilon| + 1 = 0 + 1 = 1$
- $|abca| = |abc| + 1 = |ab| + 1 + 1 = |a| + 1 + 1 + 1 = |\epsilon| + 1 + 1 + 1 + 1 = 0 + 1 + 1 + 1 + 1 = 4$

Beachte: Es gilt für $p, q \in \Sigma^*$: $|pq| = |p| + |q|$.

Hinweis: Diese Längenfunktion heißt oft auch *len()* oder *length()*.

- |wa| = |w| + 1
  - Rekursiv definierte Funktion
  - w ist ein Wort des Alphabets und a ist ein Element des Alphabets (ein Zeichen)

Die Funktion Anzahl

Definition

Die Funktion $anzahl : \Sigma^* \times \Sigma \to \mathbb{N}$ zählt, wie oft ein Zeichen in einem Wort vorkommt:

- $anzahl(\epsilon, b) = 0$ für alle $b \in \Sigma$ und
- $anzahl(wa, b) = \begin{cases} anzahl(w, b) + 1 & \text{falls } a = b \\ anzahl(w, b) & \text{sonst} \end{cases}$

  für $a, b \in \Sigma$ und $w \in \Sigma^*$.

Beispiel mit $\Sigma = \{a, b, c\}$

- $anzahl(abca, a) = anzahl(abc, a) + 1 = anzahl(ab, a) + 1 = anzahl(a, a) + 1 = anzahl(\epsilon, a) + 1 + 1 = 0 + 1 + 1 = 2$

Hinweis: Eine andere Notation für $anzahl(w, a)$ ist auch $|w|_a$.

- Rekursiv definierte Funktion:
  - Wenn das letzte Zeichen des Wortes dem gesuchten Zeichen entspricht, wird die anzahl erhöht und der Rest vom Wort betrachtet. Ansonsten wird nur der Rest vom Wort betrachtet
  - Geht so lange, bis das Wort aus dem leeren Wort besteht

Die Funktion Tausche

Definition

Die Funktion $tausche : \Sigma^* \times \Sigma \times \Sigma \to \Sigma^*$ ersetzt jedes Vorkommen des Buchstabens a im Wort w durch den Buchstaben b:

- $tausche(\epsilon, a, b) = \epsilon$ und
- $tausche(cw, a, b) = \begin{cases} b \cdot tausche(w, a, b) & \text{falls } c = a \\ c \cdot tausche(w, a, b) & \text{sonst} \end{cases}$

  für $c \in \Sigma$ und $w \in \Sigma^*$

Beispiel mit $\Sigma = \{a, b, c, d\}$

- $tausche(abca, a, d) = d \cdot tausche(bca, a, d) = d \cdot b \cdot tausche(ca, a, d) = d \cdot b \cdot c \cdot tausche(a, a, d) = d \cdot b \cdot c \cdot d \cdot tausche(\epsilon) = dbcd$

- Funktionseingabe: Wort über das Alphabet, Zeichen des Alphabets, Zeichen des Alphabets
- Ausgabe: Wort über das Alphabet
- Rekursive Funktion:
  - Wenn das erste Zeichen dem gesuchten Zeichen entspricht, wird das erste Zeichen ausgetauscht und das restliche Wort betrachtet. Ansonsten bleibt das bisherige Zeichen und das restliche Wort wird betrachtet
  - Beim leeren Wort muss nichts getauscht werden

Die Funktion Teilwort

Definition

Die Funktion $teilwort : \Sigma^* \times \mathbb{N} \times \mathbb{N} \to \Sigma^*$ liefert das Teilwort eines Wortes beginnend ab dem Index k und der Länge l:

$$teilwort(a_1 a_2 \ldots a_n, k, l) = b_k b_{k+1} \ldots b_{k+l-1}.$$

Beispiel

Betrachte $w =$ Guten Morgen. Dann:

- $teilwort(w, 7, 6) =$ Morgen $= w_1$
- $teilwort(w_1, 2, 2) =$ or $= w_2$
- $teilwort(teilwort(w, 7, 6), 2, 2) = teilwort(w, 7 + 2 - 1, 2) = w_2$

Hinweis: Die Hintereinanderausführung der Funktion *teilwort* kann durch einen Aufruf realisiert werden:

- $teilwort(teilwort(w, m, n), k, l) = teilwort(w, m + k - 1, l)$.

- -1, weil ansonsten Buchstabe doppelt mitgezählt wird

Die Funktion Ersetze

Definition

Die Funktion *ersetze* : $\Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^*$ ersetzt in einem Wort ein Teilwort durch ein anderes Wort.
Sei $w, x, p, y \in \Sigma^*$ mit $w = x \cdot p \cdot y$ und p ist kein Teilwort von x. Dann ist:

- $ersetze(w, p, q) = x \cdot q \cdot y$

Beispiele

- $ersetze(Winter, in, et) = Wetter$
- $ersetze(Winter, W, \epsilon) = inter$
- $ersetze(Winter, ter, ner) = Winner$
- $ersetze(Winter, Winter, Sommer) = Sommer$

Zusammenfassung Wortfunktionen

Folgende Basisfunktionen sind typischerweise Teil einer Bibliothek zum Arbeiten mit Wörtern (d.h. Zeichenketten bzw. Strings):

- *position* : $\Sigma^* \times \mathbb{N} \to \Sigma$ gibt das i-te Zeichen eines Wortes zurück.
- *concat* : $\Sigma^* \times \Sigma^* \to \Sigma^*$ zur Verkettung von zwei Wörtern.
- *istTeilwort* : $\Sigma^* \times \Sigma^* \to \mathbb{N}$ zur Überprüfung, ob ein Wort in einem anderen Wort vorkommt.
- *length* : $\Sigma^* \to \mathbb{N}$ liefert die Anzahl der Zeichen in einem Wort.
- *anzahl* : $\Sigma^* \times \Sigma \to \mathbb{N}$ zählt, wie oft ein Zeichen in einem Wort vorkommt.
- *tausche* : $\Sigma^* \times \Sigma \times \Sigma \to \Sigma^*$ ersetzt jedes Vorkommen eines Zeichens in einem Wort durch ein anderes Zeichen.
- *teilwort* : $\Sigma^* \times \mathbb{N} \times \mathbb{N} \to \Sigma^*$ liefert ein Teilwort eines Wortes.
- *ersetze* : $\Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma^*$ ersetzt ein Teilwort in einem Wort durch ein anderen Wort.

Formale Sprachen (1)

Nachdem wir Sicherheit im Umgang mit Wörtern erworben haben, beschäftigen wir uns nun mit Mengen von Wörtern, d.h. Wortmengen.

Erste, einfache Wortmengen haben wir bereits zu Beginn des Kapitels kennengelernt wie etwa die Menge Σ^* für ein gegebenes Alphabet Σ.

Beispiel

- Sei $\Sigma = \{a, b\}$. Dann ist $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$.

Viel interessanter sind solche Wortmengen, in denen nicht alle Wörter aus Σ^* enthalten sind:

Beispiel

- Die Menge aller syntaktisch korrekten Programme einer gegebenen Programmiersprache.

Die führt uns direkt zu folgender Definition auf der nächsten Folie:

Formale Sprachen (2)

Definition

Es sei Σ ein Alphabet. Eine **formale Sprache** L (kurz: Sprache) ist eine Teilmenge von $L \subseteq \Sigma^*$, der Menge aller möglichen Wörter über Σ.

Formale Sprachen können

- endlich (z.B. $\{00, 01, 10, 11\}$) oder
- unendlich (z.B. $\{1, 11, 111, 1111, 11111, \ldots\}$) sein.

Hinweis: Eine formale Sprache kann auch leer sein sowie maximal alle möglichen Wörter über Σ enthalten.

Eine spannende Frage ist, wie eine konkrete formale Sprache L spezifiziert werden kann. Eine explizite Angabe aller Elemente von L ist nicht möglich, da L typischerweise unendlich viele Elemente beinhaltet.

Bevor wir uns in den folgenden Kapiteln näher mit dieser Frage auseinandersetzen, benötigen wir Funktionen für Wortmengen.

Sprachen sind Mengen von Wörtern

Sprachen sind also Mengen von Wörtern und können mit den üblichen Mengenoperationen wie

- Vereinigung
- Durchschnitt und
- Differenz

miteinander verknüpft werden.

Zusätzlich benötigen wir die

- Konkatenation sowie die
- Wiederholung (Iteration).

Vereinigung von Wortmengen

Definition

Seien L_1 und L_2 formale Sprachen. Unter der **Vereinigung** ("$\cup$" bzw. "$+$") von L_1 und L_2 versteht man die Menge der Wörter aus L_1 oder L_2:

$$L_1 \cup L_2 = L_1 + L_2 = \{w \mid w \in L_1 \text{ oder } w \in L_2\}.$$

Beispiel

Betrachte

- $L_1 = \{a, b, aa, ab, ba, bb\}$ über $\Sigma = \{a, b\}$ sowie
- $L_2 = \{b, c, bb, bc, cb, cc\}$ über $\Sigma = \{b, c\}$.

Dann:

- $L_1 \cup L_2 = \{a, b, aa, ab, ba, bb, c, bc, cb, cc\}$

Beispiel

Beachte: $\{a\}^* \cup \{b\}^* \neq \{a, b\}^*$. Warum?

Durchschnitt von Wortmengen

Definition

Seien L_1 und L_2 formale Sprachen. Unter dem **Durchschnitt** ("$\cap$") von L_1 und L_2 versteht man die Menge der Wörter, die sowohl in L_1 als auch L_2 sind:

$$L_1 \cap L_2 = \{w \mid w \in L_1 \text{ und } w \in L_2\}.$$

Beispiel

Betrachte

- $L_1 = \{a, b, aa, ab, ba, bb\}$ über $\Sigma = \{a, b\}$ sowie
- $L_2 = \{b, c, bb, bc, cb, cc\}$ über $\Sigma = \{b, c\}$.

Dann:

- $L_1 \cap L_2 = \{b, bb\}$

Beispiel

Beachte: $\{a\}^* \cap \{b\}^* = \{\epsilon\}$.

Differenz von Wortmengen

Definition

Unter der **Differenz** ("$\setminus$") von L_1 und L_2 versteht man die Menge der Wörter aus L_1, die nicht in L_2 sind:

$$L_1 \setminus L_2 = \{w \mid w \in L_1 \text{ und } w \notin L_2\}.$$

Beispiel

Betrachte

- $L_1 = \{a, b, aa, ab, ba, bb\}$ über $\Sigma = \{a, b\}$ sowie
- $L_2 = \{b, c, bb, bc, cb, cc\}$ über $\Sigma = \{b, c\}$.

Dann:

- $L_1 \setminus L_2 = \{a, aa, ab, ba\}$

Beispiel

Beachte: $\{a, b\}^* \setminus \{b\}^* = \{a, aa, ab, ba, \ldots\}$.

- Alle Elemente, die in der ersten Menge vorkommen, aber nicht in der zweiten

Konkatenation von Wortmengen

Definition

Unter der **Konkatenation** ("$\cdot$") von L_1 und L_2 versteht man die Verkettung der Wörter aus L_1 mit L_2:

$$L_1 \cdot L_2 = \{v \cdot w \mid v \in L_1 \text{ und } w \in L_2\}.$$

Beispiel

Seien $L_1 = \{\epsilon, ab, abb\}$ und $L_2 = \{b, ba\}$ zwei Sprachen über $\Sigma = \{a, b\}$. Dann ist

- $L_1 \cdot L_2 = \{b, ba, abb, abba, abbb, abbba\}$ sowie
- $L_2 \cdot L_1 = \{b, bab, babb, ba, baab, baabb\}$.

Beachte:

- $\{a\}^* \cdot \{a\}^* = \{a\}^*$.
- $\{a\}^* \cdot \{b\}^* \neq \{b\}^* \cdot \{a\}^*$.
- $\{a\}^* \cdot \{b\}^* \neq \{a, b\}^*$.

- Konkatenation von <u>Wortmengen</u>
  - Alle Kombinationen von Wörtern

Datentyp Menge

- Konkatenation von Wortmengen: concat Set<String> X Set<String> -> Set<String>

  Ψ L_1 Ψ L_2

  $L_1 * L_2 = \{ v * w \mid v \in L_1 \text{ und } w \in L_2 \}$
  $L_1 = \{ ab, c \}, L_2 = \{ ba, d \}, L_1 * L_2 = \{ abba, abd, cba, cd \}$
  $L_1 = \{ \varepsilon, a, ba \}, L_2 = \{ c \}, L_1 * L_2 = \{ c, ac, bac \}$
  => Alle Mengen aus der einen Menge werden mit allen aus der anderen kombiniert
  - Reihenfolge spielt eine Rolle

Iteration

Definition

Die **Iteration** (**Sternoperator** "$*$", **Kleene'scher Stern**) ist die beliebig oftmalige Aneinanderreihung von Wörtern einer Sprache L:

$$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^1 \cup L^2 \cup \ldots \text{ mit } L^0 = \{\epsilon\}, L^1 = L^0 \cdot L, L^2 = L^1 \cdot L, \ldots$$

Beispiel

Sei $L = \{a, ab\}$ eine Sprache über dem Alphabet $\Sigma = \{a, b\}$. Dann ist

- $L^0 = \{\epsilon\}$
- $L^1 = L^0 \cdot L = \{\epsilon\} \cdot \{a, ab\} = \{a, ab\} = L$.
- $L^2 = L^1 \cdot L = \{a, ab\} \cdot \{a, ab\} = \{aa, aab, aba, abab\}$
- $L^3 = L^2 \cdot L = \{aa, aab, aba, abab\} \cdot \{a, ab\} = \{aaa, aaab, aaba, \ldots\}$

- Kleensche Stern *
  - Man darf mehrmals in Wortmenge reingreifen und zusammenfügen

- $L^* \subseteq \Sigma^*$
  - L enthält nicht unbedingt alle Wörter über Σ^*
  - Somit enthält L^* auch nicht unbedingt alle Wörter über Σ^*

- $L^0 = \{\varepsilon\}$ weil so definiert

<u>Iteration</u>

- $v \in \Sigma^*$
  $v^n = \underbrace{v^* \ldots v}_{n}$
- Σ^*
- $L \subseteq \Sigma^*$, z.B: $L_1 = \{ a, ba, c \}$
  $L^* = \{ v_1 v_2 \ldots v_n \mid v_i \in L, i \geq 0 \}$
  "beliebig viele Wörter aus der L- Menge hintereinander setzen"
- $L_1^* = \{ abac, ac, a, aaa, caba, \ldots\}$

Zusammenfassung

- Wörter, d.h. Folgen von Zeichen, spielen in der Informatik eine zentrale Rolle für eine formalisierte Darstellung von Informationen.
- Für die Verarbeitung von Wörtern werden Funktionen benötigt wie etwa Konkatenation, Länge, Teilwort, Ersetzen von Zeichen und Teilwörtern.
- Eine formale Sprache beschreibt eine "interessante" Menge von Wörtern, d.h. Wörter, die eine bestimmte Eigenschaft haben, beispielsweise die Menge aller gültigen Datumsformate.
- Formalen Sprachen können definiert werden durch reguläre Ausdrücke, endliche Automaten und Grammatiken. Hierfür werden die am Ende dieses Kapitels eingeführten Funktionen auf Wortmengen benötigt.

Inhalt