Theoretische Informatik HS24

Nicolas Wehrli

Übungsstunde 01

25. September 2024

ETH Zürich
nwehrl@ethz.ch

Heute

1. Organisation
2. Grundbegriffe
   - Alphabet
   - Wort
   - Sprache
3. Algorithmische Probleme
4. Kolmogorov Komplexität

Organisation

Organisation

Kontakt

- In der Übungsstunde
- Per Mail an nwehrl@student.ethz.ch
- Discord: `.blackphoenyx`, bzw. **Nicolas[TI]**
- WhatsApp-Chat QR-Code und Link per Mail

Aufgaben

- 50% der Punkte reichen für Teilnahme an Midterms (zu empfehlen)
- Gruppeneinteilung heute
- LaTex Empfehlung, Overleaf, Template
- Abgaben per Moodle

Webseite: https://n.ethz.ch/~nwehrl/TheoInf

Grundbegriffe

Notation

Für eine Menge A bezeichnet $|A|$ die Kardinalität von A und $\mathcal{P}(A) = \{S \mid S \subseteq A\}$ die Potenzmenge von A.

In diesem Kurs definieren wir $\mathbb{N} = \{0, 1, 2, \ldots\}$.

Alphabet

Definition Alphabet

Eine endliche, nichtleere Menge Σ heisst **Alphabet**. Die Elemente eines Alphabets werden **Buchstaben (Zeichen, Symbole)** genannt.

Beispiele

$$\Sigma_{\text{bool}} = \{0, 1\}$$
$$\Sigma_{\text{lat}} = \{a, \ldots, z\}$$
$$\Sigma_{\text{Tastatur}} = \Sigma_{\text{lat}} \cup \{A, \ldots, Z, \text{␣}, >, <, (,), \ldots, !\}$$
$$\Sigma_{\text{logic}} = \{0, 1, (,), \wedge, \vee, \neg\}$$
$$\Sigma_{abc} = \{a, b, c\} \text{ (\textbf{unser Beispiel für weitere Definitionen})}$$

Definition Wort

- Sei Σ ein Alphabet. Ein **Wort** über Σ ist eine **endliche** (eventuell leere) Folge von Buchstaben aus Σ.
- Das **leere Wort** λ ist die leere Buchstabenfolge.
- Die **Länge** $|w|$ eines Wortes w ist die Länge des Wortes als Folge, i.e. die Anzahl der Vorkommen von Buchstaben in w.
- Σ^* ist die Menge aller Wörter über Σ. $\Sigma^+ := \Sigma^* \setminus \{\lambda\}$ ist Menge aller nichtleeren Wörter über Σ.
- Seien $x \in \Sigma^*$ und $a \in \Sigma$. Dann ist $|x|_a$ definiert als die Anzahl der Vorkommen von a in x.

Achtung Metavariablen! I.e. Das a steht hier für einen beliebigen Buchstaben aus Σ und **nicht** nur für den Buchstaben $'a'$, der in Σ sein könnte.

Bemerkungen

- Wir schreiben Wörter ohne Komma, i.e. eine Folge $x_1, x_2, ..., x_n$ schreiben wir $x_1 x_2 ... x_n$.
- $|\lambda| = 0$ aber $|\textvisiblespace| = 1$ von Σ_{Tastatur}.
- Der Begriff **Wort** als Fachbegriff der Informatik entspricht **nicht** der Bedeutung des Begriffs Wort in natürlichen Sprachen!
- E.g. Mit ␣ kann der Inhalt eines Buches oder ein Programm als ein Wort über Σ_{Tastatur} betrachtet werden.

Beispiel

Verschiedene Wörter über Σ_{abc}:

a, aa, aba, cba, $caaaab$ etc.

Konkatentation

Die **Verkettung (Konkatenation)** für ein Alphabet Σ ist eine Abbildung Kon: $\Sigma^* \times \Sigma^* \to \Sigma^*$, so dass

$$\mathrm{Kon}(x, y) = x \cdot y = xy$$

für alle $x, y \in \Sigma^*$.

- Die Verkettung Kon (i.e. Kon von einem Kon (über das gleiche Alphabet Σ)) ist eine assoziative Operation über Σ^*.

$$\mathrm{Kon}(u, \mathrm{Kon}(v, w)) = \mathrm{Kon}(\mathrm{Kon}(u, v), w),\ \forall u, v, w \in \Sigma^*$$

- $x \cdot \lambda = \lambda \cdot x = x,\ \forall x \in \Sigma^*$
- $\implies$ (Σ^*, Kon) ist ein Monoid mit neutralem Element λ.
- Kon nur kommutativ, falls $|\Sigma| = 1$.
- $|xy| = |x \cdot y| = |x| + |y|$. (Wir schreiben ab jetzt xy statt $\mathrm{Kon}(x, y)$)

Beispiel

Wir betrachten wieder Σ_{abc}. Sei $x = abba, y = cbcbc, z = aaac$.

- $\mathrm{Kon}(x, \mathrm{Kon}(y, z)) = \mathrm{Kon}(x, yz) = xyz = abbacbcbcaaac$
- $|xy| = |abbacbcbc| = 9 = 4 + 5 = |abba| + |cbcbc| = |x| + |y|$

Reversal und Iteration

Für eine Wort $a = a_1a_2...a_n$, wobei $\forall i \in \{1, 2, ..., n\}.\ a_i \in \Sigma$, bezeichnet $a^{\mathrm{R}} = a_na_{n-1}...a_1$ die **Umkehrung (Reversal)** von a.

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

$$x^0 = \lambda, x^1 = x \text{ und } x^i = xx^{i-1}.$$

Beispiel

Wir betrachten wieder Σ_{abc}. Sei $x = abba$, $y = cbcbc$, $z = aaac$.

- $z^{\mathrm{R}} = (aaac)^{\mathrm{R}} = caaa$
- $x^{\mathrm{R}} = (abba)^{\mathrm{R}} = abba$
- $x^0 = \lambda$
- $y^2 = yy^{2-1} = yy = cbcbccbcbc$
- $z^3 = zz^2 = zzz = aaacaaacaaac$
- $(x^{\mathrm{R}}z^{\mathrm{R}})^{\mathrm{R}} = ((abba)^{\mathrm{R}}(aaac)^{\mathrm{R}})^{\mathrm{R}} = (abbacaaa)^{\mathrm{R}} = aaacabba$

Teilwort, Präfix und Suffix

Seien $v, w \in \Sigma^*$ für ein Alphabet Σ.

- v heisst ein **Teilwort** von $w \iff \exists x, y \in \Sigma^* : \; w = xvy$
- v heisst ein **Präfix** von $w \iff \exists y \in \Sigma^* : \; w = vy$
- v heisst ein **Suffix** von $w \iff \exists x \in \Sigma^* : \; w = xv$
- $v \neq \lambda$ heisst ein **echtes** Teilwort (Präfix, Suffix) von $w \iff v \neq w$ und v Teilwort (Präfix, Suffix) von w

Teilwort, Präfix und Suffix - Beispiel

Beispiel

Wir betrachten wieder Σ_{abc}. Sei $x = abba$, $y = cbcbc$, $z = aaac$.

- bc ist ein echtes Suffix von y
- $abba$ ist kein echtes Teilwort von x.
- $cbcb$ ist ein echtes Teilwort und echtes Präfix von y.
- ac ist ein echtes Suffix.
- $abba$ ist ein Suffix, Präfix und Teilwort von x.

Teilwörter - Beispielaufgabe 1

> Sei Σ ein Alphabet und sei $w \in \Sigma^*$ ein Wort der Länge $n \in \mathbb{N} \setminus \{0\}$. Wie viele unterschiedliche Teilwörter kann w **höchstens** haben?

Wir haben $w = w_1 w_2 ... w_n$ mit $w_i \in \Sigma$ für $i = 1, ..., n$. Wie viele Teilwörter beginnen mit w_1? Wie viele Teilwörter beginnen mit w_2?

Wir haben also $n + (n-1) + ... + 1 = \frac{n(n+1)}{2}$ Teilwörter. Etwas fehlt aber in unserer Berechnung...

Das leere Wort λ ist auch ein Teilwort! Also haben wir $\frac{n(n+1)}{2} + 1$ Teilwörter.

Teilwörter - Beispielaufgabe 2

> Sei $\Sigma = \{a, b, c\}$ und $n \in \mathbb{N}$. Bestimme die Anzahl der Wörter aus Σ^n, die das Teilwort a enthalten.

In solchen Aufgaben ist es manchmal einfach, das Gegenteil zu berechnen und so auf die Lösung zu kommen. Wie viele Wörter aus Σ^n enthalten das Teilwort a **nicht**?

Da wir jetzt die Anzahl Wörter der Länge n wollen, die nur b und c enthalten, kommen wir auf $|\{b, c\}|^n = 2^n$.

Daraus folgt, dass genau $|\Sigma|^n - 2^n = 3^n - 2^n$ Wörter das Teilwort a enthalten.

Teilwörter - Beispielaufgabe 3

> Sei $\Sigma = \{a, b, c\}$ und $n \in \mathbb{N} \setminus \{0\}$. Bestimme die Anzahl der Wörter aus Σ^n, die das Teilwort aa nicht enthalten.

Wir bezeichnen die Menge aller Wörter mit Länge n über Σ, die aa nicht enthalten als L_n.

Schauen wir mal die ersten zwei Fälle an:

$$L_1 = \{a, b, c\} \implies |L_1| = 3$$
$$L_2 = \{ab, ac, ba, bb, bc, ca, cb, cc\} \implies |L_2| = 8$$

Teilwörter - Beispielaufgabe 3

Nun können wir für $m \geq 3$ jedes Wort $w \in L_m$ als Konkatination $w = x \cdot y \cdot z, |y| = |z| = 1$ schreiben, wobei wir zwei Fälle unterscheiden:

(a) $\boldsymbol{z \neq a}$

In diesem Fall kann $y \in \{a, b, c\}$ sein, ohne dass die Teilfolge aa entsteht und somit ist xy ein beliebiges Wort aus L_{m-1}.
Dann könnten wir alle Wörter in diesem Case durch $L_{m-1} \cdot \{b, c\}$ beschreiben, was uns die Kardinalität $2 \cdot |L_{m-1}|$ gibt.

(b) $\boldsymbol{z = a}$

In diesem Fall muss $y \neq a$ sein, da sonst aa entstehen würde.
Somit kann xy nur in b oder c enden. x kann aber ein beliebiges Wort der Länge $m - 2$ sein.
Deshalb können wir alle Wörter in diesem Case durch $L_{m-2} \cdot \{b, c\} \cdot \{a\}$ beschreiben. Kardinalität: $2 \cdot |L_{m-2}|$.

Daraus folgt

$$|L_n| = \begin{cases} 3 & n = 1 \\ 8 & n = 2 \\ 2|L_{n-1}| + 2|L_{n-2}| & n \geq 3 \end{cases}$$

Kanonische Ordnung

Sei $\Sigma = \{s_1, s_2, ..., s_m\}, m \geq 1$, ein Alphabet und sei $s_1 < s_2 < ... < s_m$ eine Ordnung auf Σ. Wir definieren die **kanonische Ordnung** auf Σ^* für $u, v \in \Sigma^*$ wie folgt:

$$u < v \iff |u| < |v| \vee (|u| = |v| \wedge u = x \cdot s_i \cdot u' \wedge x \cdot s_j \cdot v')$$

$$\text{für irgendwelche } x, u', v' \in \Sigma^* \text{ und } i < j.$$

Kanonische Ordnung - Beispiel

Sei $\Sigma_{abc} = \{a, b, c\}$ und wir betrachten folgende Ordnung auf Σ_{abc}: $c < a < b$.
Was wäre die kanonische Ordnung folgender Wörter?

- c, abc, $aaac$, $aaab$, $bacc$, a, λ

λ, c, a, abc, $aaac$, $aaab$, $bacc$

Sprache

Eine **Sprache** L über einem Alphabet Σ ist eine Teilmenge von Σ^*.

- Das Komplement $L^{\complement}$ der Sprache L bezüglich Σ ist die Sprache $\Sigma^* \setminus L$.
- $L_\emptyset = \emptyset$ ist die **leere Sprache**.
- $L_\lambda = \{\lambda\}$ ist die einelementige Sprache, die nur aus dem leeren Wort besteht.

Konkatenation von Sprachen

Sind L_1 und L_2 Sprachen über Σ, so ist

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

die **Konkatenation** von L_1 und L_2.

Iteration von Sprachen

Ist L eine Sprache über Σ, so definieren wir

$$L^0 := L_\lambda \text{ und } L^{i+1} := L^i \cdot L \text{ für alle } i \in \mathbb{N},$$

$$L^* = \bigcup_{i \in \mathbb{N}} L^i \text{ und } L^+ = \bigcup_{i \in \mathbb{N} \setminus \{0\}} L^i = L \cdot L^*.$$

L^* nennt man den **Kleene'schen Stern** von L.

Man bemerke, dass $\Sigma^i = \{x \in \Sigma^* \mid |x| = i\}$, $L_\emptyset L = L_\emptyset = \emptyset$ und $L_\lambda \cdot L = L$.

Sprachen - Beispiel

Mögliche Sprachen über Σ_{abc}

- $L_1 = \emptyset$
- $L_2 = \{\lambda\}$
- $L_3 = \{\lambda, ab, baca\}$
- $L_4 = \Sigma_{abc}^*$, $L_5 = \Sigma_{abc}^+$, $L_6 = \Sigma_{abc}$ oder $L_7 = \Sigma_{abc}^{27}$
- $L_8 = \{c\}^* = \{c^i \mid i \in \mathbb{N}\}$
- $L_9 = \{a^p \mid p \text{ ist prim.}\}$
- $L_{10} = \{c^i a^{3i^2} b a^i c \mid i \in \mathbb{N}\}$

λ ist ein Wort über jedes Alphabet. Aber es muss nicht in jeder Sprache enthalten sein!

Lemmas über Sprachen

Seien L_1, L_2 und L_3 Sprachen über einem Alphabet Σ. Dann gilt

$$L_1L_2 \cup L_1L_3 = L_1(L_2 \cup L_3) \tag{1}$$

$$L_1(L_2 \cap L_3) \subseteq L_1L_2 \cap L_1L_3 \tag{2}$$

Weshalb nicht '=' bei (2)?

Sei $\Sigma = \Sigma_{\text{bool}} = \{0, 1\}$, $L_1 = \{\lambda, 1\}$, $L_2 = \{0\}$ und $L_3 = \{10\}$.

Dann haben wir $L_1(L_2 \cap L_3) = \emptyset \neq \{10\} = L_1L_2 \cap L_1L_3$.

Beweise im Buch/Vorlesung

Aufgabe 2.10

Seien L_1, L_2 und L_3 Sprachen über dem Alphabet $\{0\}$. Gilt

$$L_1(L_2 \cap L_3) = L_1L_2 \cap L_1L_3?$$

Aufgabe 2.11

Seien $L_1 \subseteq \Sigma_1^*$ und $L_2, L_3 \subsetneq \Sigma_2^*$ für zwei Alphabete Σ_1 und Σ_2 mit $\Sigma_1 \cap \Sigma_2 = \emptyset$. Gilt

$$L_1(L_2 \cap L_3) = L_1L_2 \cap L_1L_3?$$

Von einem Alphabet zum anderen

Seien Σ_1 und Σ_2 zwei beliebige Alphabete. Ein Homomorphismus von Σ_1^* nach Σ_2^* ist jede Funktion $h : \Sigma_1^* \to \Sigma_2^*$ mit den folgenden Eigenschaften:

(i) $h(\lambda) = \lambda$ und

(ii) $h(uv) = h(u) \cdot h(v)$ für alle $u, v \in \Sigma_1^*$.

Wir können Probleme etc. in anderen Alphabeten kodieren. So wie wir verschiedenste Konzepte, die wir auf Computer übertragen in Σ_{bool} kodieren.

Algorithmische Probleme

Vorläufige Definition des Begriffs Algorithmus

Mathematische Definition folgt in Kapitel 4 (Turingmaschinen).

Vorerst betrachten wir Programme, die **für jede zulässige Eingabe halten und eine Ausgabe liefern**, als Algorithmen.
Wir betrachten ein Programm (Algorithmus) A als Abbildung $A : \Sigma_1^* \to \Sigma_2^*$ für beliebige Alphabete Σ_1 und Σ_2. Dies bedeutet, dass

(i) die Eingaben als Wörter über Σ_1 kodiert sind,
(ii) die Ausgaben als Wörter über Σ_2 kodiert sind und
(iii) A für jede Eingabe eine eindeutige Ausgabe bestimmt.

A und B äquivalent $\iff$ Eingabealphabet Σ gleich, $A(x) = B(x), \forall x \in \Sigma^*$

Ie. diese Notion von "Äquivalenz" bezieht sich nur auf die Ein und Ausgabe.

Entscheidungsprobleme

Das **Entscheidungsproblem** (Σ, L) für ein gegebenes Alphabet Σ und eine gegebene Sprache $L \subseteq \Sigma^*$ ist, für jedes $x \in \Sigma^*$ zu entscheiden, ob

$$x \in L \text{ oder } x \notin L.$$

Ein Algorithmus A **löst** das Entscheidungsproblem (Σ, L), falls für alle $x \in \Sigma^*$ gilt:

$$A(x) = \begin{cases} 1, & \text{falls } x \in L, \\ 0, & \text{falls } x \notin L. \end{cases}$$

Wir sagen auch, dass A die Sprache L erkennt.

Why do we care

> Wenn für eine Sprache L ein Algorithmus existiert, der L erkennt, sagen wir, dass L **rekursiv** ist.

Wir sind oft an spezifischen Eigenschaften von Wörtern aus Σ^* interessiert, die wir mit einer Sprache $L \subseteq \Sigma^*$ beschreiben können.

Dabei sind dann L die Wörter, die die Eigenschaft haben und $L^{\complement} = \Sigma^* \setminus L$ die Wörter, die diese Eigenschaft nicht haben.

Jetzt ist die allgemeine Formulierung von Vorteil!

Why do we care - Beispiele

i. **Primzahlen finden**:
   Entscheidungsproblem $(\Sigma_{\text{bool}}, L_p)$ wobei
   $L_p = \{x \in (\Sigma_{\text{bool}})^* \mid \text{Nummer}(x) \text{ ist prim}\}$.

ii. **Syntaktisch korrekte Programme**:
   Entscheidungsproblem $(\Sigma_{\text{Tastatur}}, L_{C++})$ wobei
   $L_{C++} = \{x \in (\Sigma_{\text{Tastatur}})^* \mid x \text{ ist ein syntaktisch korrektes C++ Programm}\}$.

iii. **Hamiltonkreise finden:**
   Entscheidungsproblem (Σ, HK) wobei $\Sigma = \{0, 1, \#\}$ und
   HK$= \{x \in \Sigma^* \mid x \text{ kodiert einen Graphen, der einen Hamiltonkreis enthält.}\}$

Äquivalenzprobleme $\subset$ Entscheidungsprobleme

Funktion, Relation

Seien Σ und Γ zwei Alphabete.

- Wir sagen, dass ein Algorithmus A eine **Funktion (Transformation)** $f : \Sigma^* \to \Gamma^*$ **berechnet (realisiert)**, falls

$$A(x) = f(x) \text{ für alle } x \in \Sigma^*$$

- Sei $R \subseteq \Sigma^* \times \Gamma^*$ eine Relation in Σ^* und Γ^*. Ein Algorithmus A **berechnet** R (bzw. **löst das Relationsproblem** R), falls für jedes $x \in \Sigma^*$, für das ein $y \in \Gamma^*$ mit $(x, y) \in R$ existiert, gilt:

$$(x, A(x)) \in R$$

Optimierungsprobleme

Ein **Optimierungsproblem** ist ein 6-Tupel $\mathcal{U} = (\Sigma_I, \Sigma_O, L, M, \text{cost}, \text{goal})$, wobei:

(i) Σ_I ist ein Alphabet (genannt **Eingabealphabet**),

(ii) Σ_O ist ein Alphabet (genannt **Ausgabealphabet**),

(iii) $L \subseteq \Sigma_I^*$ ist die Sprache der **zulässigen Eingaben** (als Eingaben kommen nur Wörter in Frage, die eine sinnvolle Bedeutung haben). Ein $x \in L$ wird ein **Problemfall (Instanz) von** $\mathcal{U}$ genannt.

(iv) M ist eine Funktion von L nach $\mathcal{P}(\Sigma_O^*)$, und für jedes $x \in L$ ist $M(x)$ die **Menge der zulässigen Lösungen für** x,

(v) **cost** ist eine Funktion, **cost**: $\bigcup_{x \in L}(\mathcal{M}(x) \times \{x\}) \to \mathbb{R}^+$, genannt **Kostenfunktion**,

(vi) **goal** $\in \{\text{Minimum}, \text{Maximum}\}$ ist das **Optimierungsziel**.

Optimierungsprobleme

Eine zulässige Lösung $\alpha \in \mathcal{M}(x)$ heisst **optimal** für den Problemfall x des Optimierungsproblems $\mathcal{U}$, falls

$$\mathrm{cost}(\alpha, x) = \mathbf{Opt}_{\mathcal{U}}(x) = \mathrm{goal}\{\mathrm{cost}(\beta, x) \mid \beta\mathcal{M}(x)\}.$$

Ein Algorithmus A **löst** $\mathcal{U}$, falls für jedes $x \in L$

(i) $A(x) \in \mathcal{M}(x)$

(ii) $\mathrm{cost}(A(x), x) = \mathrm{goal}\{\mathrm{cost}(\beta, x) \mid \beta \in \mathcal{M}(x)\}$.

Kolmogorov Komplexität

Algorithmen generieren Wörter

Sei Σ ein Alphabet und $x \in \Sigma^*$. Wir sagen, dass ein Algorithmus A das Wort x **generiert**, falls A für die Eingabe λ die Ausgabe x liefert.

Beispiel:

A_n:
```
begin
    for i = 1 to n;
        write(01);
end
```

A_n generiert $(01)^n$.

Aufzählungsalgorithmus

Sei Σ ein Alphabet und sei $L \subseteq \Sigma^*$. A ist ein **Aufzählungsalgorithmus für** L, falls A für jede Eingabe $n \in \mathbb{N} \setminus \{0\}$ die Wortfolge $x_1, \ldots, x_n$ ausgibt, wobei $x_1, \ldots, x_n$ die kanonisch n ersten Wörter in L sind.

Information messen

Kolmogorov-Komplexität

Für jedes Wort $x \in (\Sigma_{\text{bool}})^*$ ist die **Kolmogorov-Komplexität** $K(x)$ **des Wortes** x das Minimum der binären Längen, der Pascal-Programme, die x generieren.

$K(x)$ ist die kürzestmögliche Länge einer Beschreibung von x.

Die einfachste (und triviale) Beschreibung von x, ist wenn man x direkt angibt.

x kann aber eine Struktur oder Regelmässigkeit haben, die eine Komprimierung erlaubt.

Beispiel

Sei $w = 0101010101010101010101010101010101010101$. Die Länge von w ist $|w| = 40$ und die triviale Beschreibunglänge wäre wie gegeben 40.

Aber durch die Regelmässigkeit von einer 20-fachen Wiederholung der Sequenz 01, können w auch durch $(01)^{20}$ beschreiben. Hierbei ist die Beschreibungslänge ein wenig mehr als 4 Zeichen.

Grundlegende Resultate

Es existiert eine Konstante d, so dass für jedes $x \in (\Sigma_{\text{bool}})^*$

$$K(x) \leq |x| + d$$

Die **Kolmogorov-Komplexität einer natürlichen Zahl** n ist $K(n) = K(\text{Bin}(n))$.

Für jede Zahl $n \in \mathbb{N} \setminus \{0\}$ existiert ein Wort $w_n \in (\Sigma_{\text{bool}})^n$, so dass

$$K(w_n) \geq |w_n| = n$$