Yannick Müller muelleya@student.ethz.ch
yajm.ch

ETH

MADE EASY

Computer Science Block 3

January 2020

Theoretische Informatik - Lecture

Prof. Dr. J. Hromkovič and Dr. H.-J. Böckenhauer
TA: David Wehner

Contents

Note 1. 8.11.19 ist der Midterm
Übungsblatt 2 ist sehr wichtig, eine Aufgabe wird an der Prüfung kommen.

The following was presented in the lecture on 20. September 2019.

2 Grundbegriffe

Definition 1. Man braucht:
ein Alphabet (Verschiedene Symbole)
Wort über ein Alphabet (Endliche Folgen von Symbolen aus dem Alphabet) dabei wird nicht zwischen Wort und Text (mit Leerzeichen) unterschieden

$$E^{\star}_{bin} = \{\lambda, 0, 1, 00, 01, 10, 11, 000, ...\} = \{x_1, x_2, ..., x_n | n \in \mathbb{N},\ \forall i \in \{1, ..., n\} : x_i \in \sigma\} \tag{1}$$

Konkatenation - Kon über σ

$$Kon\sigma^{\star} \times \sigma^{\star} \Rightarrow \sigma^{\star} \quad Kon(x, y) = x \cdot y = xy \tag{2}$$

$$(x_1 x_2 ... x_n)^R = x_n x_{n-1} ... x_1 \tag{3}$$

Es gibt auch Teilwörter, Präfix und Suffix

Definition 2. Kanonische Ordnung: Das kürzere Wort kommt immer zuerst und sonst nach den Buchstaben sortiert. Sprache: über $\sum$: $L \subseteq \sum^{\star}$
Leere Sprache : $L_{\varnothing}, L_x = \{\lambda\}$
Komplement von L: $L^C = \sum^{X} - L$

$$L^{\star} = \bigcup_{i \in \mathbb{N}} L^i \tag{4}$$

$$L^{+} = \bigcup_{i \in \mathbb{N} \setminus \{0\}} L^i \tag{5}$$

The following was presented in the lecture on 24. September 2019.

2.1 Algorithmus

Input: $x \in \sum_I^{\star}$
Ausgabe: $y \in \sum_O^{\star}$
Algorithmus A: $\sum_I^{\star} \to \sum_O^{\star}$

Note 2. Ein Entscheidungsproblem bekommt ein Wort und gibt 0 oder 1 aus. Ein Algorithmus löst ein Problem genau dann wenn $x \in L$ und gibt dann 1 aus.

Example 1. Gegeben ein Wort mit den Buchstaben a,b,c,d
Die Buchstaben haben unterschiedliche Häufigkeiten.
Dann macht es Sinn für Häufigere Buchstaben eine kürzere Kodierung zu wählen um Platz zu

sparen:

$$H_a \cdot log_2 \frac{|x|}{H_a} + H_b \cdot log_2 \frac{|x|}{H_b} + H_c \cdot log_2 \frac{|x|}{H_c} + H_d \cdot log_2 \frac{|x|}{H_d} \tag{6}$$

$$-H_a \cdot log_2 P_a - H_b \cdot log_2 P_b - H_c \cdot log_2 P_c - H_d \cdot log_2 P_d \tag{7}$$

Dennoch kann dies nicht verwendet werden um den Informationsgehalt eines Wortes anzugeben, denn es könnte eine kürzere Folge existieren. Mit speziellen Reihenfolge könnten noch viel kürzere Kodierungen entstehen.

Definition 3. Komogorovkomplexität des Wortes $x \in \{0,1\}^\star$ ist die binäre Länge des kürzesten Pascalprogramms, welches x generiert.

$$\exists c \in \mathbb{N}, \forall x \in \{0,1\}^\star \quad K(x) \leq |x| + c \tag{8}$$

Note 3. Ein Objekt ist zufällig, falls es nicht kürzer dargestellt werden kann. Sonst ist es nicht zufällig und es kann generiert werden.

The following was presented in the lecture on 27. September 2019.

Note 4. Für alle möglichen Programmiersprachen unterscheidet sich die Länge nur durch eine additive Konstante. Beweis: Code Sprache A + Interpreter Sprache A+B = Sprache A + additive Konstante

Note 5. Ein Wort heisst zufällig, falls die Komogorovkomplexität gösser als die Länge des Wortes ist.

Theorem 2.1

$\epsilon : L \subseteq \sum^\star$ Sei z_i das kanonisch n-te Wort in b. Falls das Programm A_B das Entscheidungsproblem $(\sum, L)$ löst gilt:

$$\forall n \in \mathbb{N} \backslash \{0\} \quad \mathcal{K}(z_n) \leq \lceil \log_2(n+1) \rceil + c \tag{9}$$

Theorem 2.2

$$\lim_{n \to \infty} \frac{Prim(n)}{\frac{n}{\ln(n)}} = 1 \quad \text{mit Prim Anzahl Primzahlen} \leq n \tag{10}$$

The following was presented in the lecture on 01. October 2019.

Proof.

$$n = P_m \cdot \frac{n}{P_m} \quad n \to (m, \frac{n}{P_m}) \tag{11}$$

$$n \to |BIN(m), m, \frac{n}{P_m}| : \tag{12}$$

$$\log_2(n-2) \leq \text{Wort}\left(n, \frac{n}{P_m}\right) \leq 2 \cdot \log_2 \log_2 \log_2 m + \log_2 \log_2 m + \log_2 m + \log_2 \left(\frac{n}{P_m}\right) + c \tag{13}$$

$$\Rightarrow P_m \leq 2^d \cdot (\log_2 \log_2 m)^2 \cdot \log_2 (m) \cdot m \tag{14}$$

$$\tag{15}$$

□

The following was presented in the lecture on 04. October 2019.

3 Endliche Automaten

$$A = (Q, \Sigma, \delta, q_0, F) \tag{16}$$

1. Q ist eine endliche Menge von Zuständen $Q = \{q_0, q_1, q_2, q_3\}$
2. Σ ist das Eingabealphabet $\sum = \{0, 1\}$
3. $\delta : \sum \to Q$ ist die Übergangsfunktion
4. $q_0 \in Q$ ist der Anfangszustand
5. $F \subseteq Q$ F ist die Menge der akzeptierten Zuständen.

Konfiguration:

$$Q \times \sum^{\star} \tag{17}$$

(q_0, x) ist die Anfangskonfiguration für x
(p, λ) ist die Endkonfiguration

Note 6. Zwei Zustände kann man zusammenführen, falls die Pfeile zur selben Destination gehen.

Note 7. Um zwei Automaten zu vereinigen also $L = L_1 \wedge L_2$ nimmt man das kartesische Produkt der beiden Automaten und erhählt dann $|L_1| \cdot |L_2|$ Zustände.

Note 8. "OR" funktioniert genau gleich, es gibt aber mehr zulässige Endzustände.

The following was presented in the lecture on 8. October 2018.

Lemma 3.2

Entweder

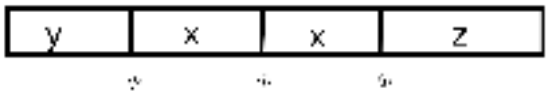
$$\{xz, yz\} \leq L(M) \text{ oder } \{x, yz\} \cap L(M) = \varnothing \tag{18}$$

Lemma 3.3

$$xz \in L(A) \Leftrightarrow yz \in L(A) \tag{19}$$

Note 9. Es ist nicht möglich ein Automat mit endlichen Anzahl von States zu bauen um zum Beispiel das Problem 0^n1^n zu lösen ohne dass n vorher bekannt ist. $\Rightarrow$ Nicht reguläre Sprache

Lemma 3.4: Pumping



Es können unendlich von diesen x eingefügt werden, ohne dass wir je den Status wissen.
$\forall L \in \mathcal{L}_{RE} \exists n_0 : \forall \omega \in \sum^\star, |\omega| \geq n_0$ $\exists$ eine Zerlegung $\omega = yxz$ sodass:

1. $|yx| \leq n_0$
2. $|x| > 0$
3. entweder $\{yx^iz \mid i \in \mathbb{N}\} \subseteq L$ oder $\{yx^iz \mid i \in \mathbb{N}\} \cap L = \varnothing$

Proof 3.1

Beweisidee: Man muss zeigen das ein Zustand q_i doppelt vorkommt. □

Theorem 3.1

Es existiert eine Konstante const, so dass für alle $x, y \in (\sum_{bool})^\star$ gilt:

$$K(y) \leq \lceil \log_2(n+1) \rceil + const \tag{20}$$

falls y das n-te Wort in der Sprache L_x ist.

The following was presented in the lecture on 15. October 2019.

Note 10. Man kann deterministische, aber auch nicht deterministische Automaten erstellen, welche mehrere gleiche Ausgaben enthalten.

Note 11. Sobald ein Wort unterbrochen wird, ist es kein gültiges Wort mehr, obwohl vielleicht die Bedingung schon erfüllt wurde.

The following was presented in the exercise on 15. October 2019.

3.1 Nichtregularitätsbeweise

1. Direkt über den Automaten
2. Pumpinglemma
3. Kolmogorov-Komplexität

Example 2. Endliche Automaten sind endlich (Lemma 3.3)

1. Beweisrahmen: Angenommen L sei regulär. Dann gibt es einen Automaten $A = (Q, \Sigma, \delta, q_0, F)$ mit $L(A) = L$
2. Wörter wählen: Betrachte $1^1 0, 1^2 0, ..., 1^{|Q|+1} 0$
3. Schubfacherprinzip: Da wir mehr Wörter als Zustände haben, gibt es nach dem Schubfacherprinzip $i < j$ sodass $\hat{\delta}(q_0,\ 1^i 0) = \hat{\delta}(q_0,\ 1^j 0)$
4. Suffix z wählen: Betrachte $z \stackrel{\text{def}}{=} 1^i 0$
5. Lemma anwenden: Dann gilt nach Lemma 3.3
$$1^i 0z \in L \Leftrightarrow 1^j 0z \in L \tag{21}$$
6. Widerspruch: Das ist ein Widerspruch, da $1^i 01^i 0 \in L$ aber $1^j 01^j 0 \notin L$, da $i < j$

Example 3. Pumping Lemma:

1. Beweisrahmen: Angenommen, L sei regulär. Dann existiert nach Pumpinglemma ein $n_0 \in \mathbb{N}$ mit den Eigenschaften des Lemmas.
2. Betrachte $w = 1^{n_0} 01^{n_0} 0$, offensichtlich $|w| > n_0$
3. Zerlegung anschauen: Nach dem Pumping Lemma existieren y, x, z mit $w = yxz$ und wegen erstem Punkt im Lemma ist $yx = 1^l 1^h$ mit $l \leq n_0$ und $l + h \leq n_0$.
   Wegen dem zweiten Punkt ist $h \geq 1$
   Da $w \in L$, muss auch $\{yx^k z \mid k \in \mathbb{N}\} \subseteq L$ sein.
   Das ist ein Widerspruch, da $yx^0 z = 1^{n_0 - h} 01^{n_0} 0$ ist und $h \geq 1$, also $yx^0 z \notin L$

Example 4. Kolmogorov-Komplexität
Besonders geeignet für $L = \{0^{f(n)} \mid n \in \mathbb{N}\}$

1. Beweisrahmen: Angenommen, L sei regulär
2. Folge von Sprachen wählen: Betrachte für $i \in \mathbb{N}'$die Sprachen $L_{1^i 0} = \{x \in \{0,1\}^\star \mid 1^i 0x \in L\}$
3. Erstes oder zweites Wort in Sprachen finden: Für alle $i \in \mathbb{N}$ ist das erste Wort in $L_{1^i 0} 1^i 0$
4. Satz 3.1 anwenden:
$$\forall i K(1^i 0) \leq \lceil \log(1+1) \rceil + c \tag{22}$$
5. Widerspruch: Das ist ein Widerspruch, weil es nur endlich viele Programme kürzer als $\log(1+1) + c$ gibt, aber unendlich viele Wörter $1^i 0$

The following was presented in the lecture on 17. October 2019.

4 Turing Maschine

Note 12. Die Turing Maschine darf im Gegensatz zum endlichen Automaten auch schreiben.

Definition 4. Eine Turingmaschine (TM) ist 7-Tubel $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

Q Endliche Zustandsmenge

Σ Eingabealphabet

Γ Arbeitsalphabet, $\Sigma \subseteq \Gamma$

δ $Q \backslash \{q_{acc}, q_{rej}\} \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ Übergangsfunktion

Es gibt jeweils nur einen Anfangszustand, einen akzeptierten und abgelehnten Zustand.
Die Turingmaschine kann deshalb unendlich lang laufen, halten und verwerfen oder halten und akzeptieren.

The following was presented in the lecture on 22. October 2019.

Definition 5. Ein Algorithmus (L_R) ist ein Programm (L_{RE}), welches ein Problem löst. Ein Algorithmus hält immer, aber ein Programm hält nicht immer

Definition 6. Eine Turing Maschine ist eine syntaktische Formulierung eines Algorithmus.

Note 13. Man kann jede Programmiersprache so stark vereinfachen, sodass man sie mit dem Mehrband-Turing Modell beschreiben kann.

Note 14. Ein Problem ist NP Hard, wenn der einzige Weg in einem Entscheidungsbaum die Lösung zu finden ist, alles mit einer Breitensuche zu durchsuchen ist.

$$Code(\delta(p, A_l) = (q, A_m, \alpha)) = \#Code(p)Code(A_l)Code(q)Code(A_m)Code(\alpha)\# \tag{23}$$

$$Code(M) = \#0^{m+3}\#0^v\#\# \tag{24}$$

The following was presented in the lecture on 25. October 2019. π oder auch $\sqrt{2}$ sind endlich darstellbar, da man ein Programm schreiben kann welches die Zahl auf n stellen generiert.

The following was presented in the lecture on 29. October 2019.

Note 15. Man kann mehr Sprachen schreiben, als man mit Algorithmen lösen kann.

Theorem 4.1: Rice

Jedes semantisch nicht-triviale Entscheidungsproblem über Turingmaschinen ist unentscheidbar.

The following was presented in the exercise on 29. October 2019.

4.1 Mögliche Prüfungsthemen

1 und 2 sind fast garantiert, aber 3 und 4 sind Vermutungen. (90min)

1. Automaten für Sprache. Mit Klassen,mit Beschreibung des Entwurfes und eventuell Mindestanzahl Zustände.
2. L_1 und L_2 und dann ist Irregulärität
3. Kolmogorov
4. Einfache Reduktion / Schwieriger Beweis

The following was presented in the lecture on 12. November 2019.

Definition 7. Sei M eine Mehrband-Turingmaschine die immer hält, mit Eingabealphabet Σ.
Sei $x \in \Sigma^\star$
Sei $D = C_1, C_2, ..., C_k$ die Berechnung von M auf x.
Zeitkomplexität von M auf x ist definiert durch:

$$Time_M(x) = k - 1 \tag{25}$$

$$Time_M(n) = \max\{Time_m(x) \mid x \in \Sigma^n\} \tag{26}$$

Definition 8. Sei M eine k-Band-Turingmaschine, die immer hält. Sei

$$C = (q, x, i, \alpha_1, i_1, \alpha_2, i_2, ..., \alpha_k, i_k) \tag{27}$$

eine Konfiguration von M. Die Speicherplatzkomplexität von C ist

$$Space_M(C) = \max\{|\alpha_i| \mid i = 1, ..., k\} \tag{28}$$

Lemma 4.1

Für jede k-Band-TM A, die immer hält existiert eine äquivalente 1-Band-TM B, so dass

$$Space_B(n) \leq Space_A(n) \tag{29}$$

Lemma 4.2

Für jedes k-Band-TM A existiert ein k-Band-TM so, dass L(A)=L(B) und

$$Space_B(n) \leq \frac{Spcac_A(n)}{2} + 2 \tag{30}$$

Note 16. $\Rightarrow$ Für jede k-Band-TM A existiert eine andere k-Band-TM mit Space 4

Note 17. Achtung: aber das Alphabet wird grösser

Lemma 4.3

Sei M eine Mehrband-Turingmaschine, die immer hält. Dann existiert eine zu M äquivalente Mehrband-Turingmaschine A mit

$$Time_A(n) \leq \frac{Time_M(n)}{2} + 2n \tag{31}$$

Theorem 4.2: Blumsches SpeedUp-Theorem

Es existiert ein Entscheidungsproblem L, so dass für jede Mehrband-TM A, die L entscheidet, eine Mehrband-TM B existiert, die auch L entscheidet und es gilt:

$$Time_B(n) \leq log_2(Time_A(n)) \tag{32}$$

für unendlich viele $n \in \mathbb{N}$

Definition 9. Eine Mehrband-TM C heisst **optimal** für L, falls eine obere Schranke $Time_C(n) \in \mathcal{O}(f(n))$ gilt und $Time_C(n) \in \Omega(f(n))$ eine untere Schranke für die Zeitkomplexität von L ist für **jede** MTM.

4.2 Komplexitätsklassen

Definition 10.

$$TIME(f) = \{L(B) \mid B \text{ ist eine MTM mit } Time_B(n) \in \mathcal{O}(f(n))\} \tag{33}$$
$$SPACE(f) = \{L(A) \mid A \text{ ist eine MTM mit } Space_A(n) \in \mathcal{O}(g(n))\} \tag{34}$$
$$DLOG = SPACE(log_2 n) \tag{35}$$
$$P = \bigcup_{c \in \mathbb{N}} TIME(n^c) \tag{36}$$
$$PSPACE = \bigcup_{c \in \mathbb{N}} SPACE(n^c) \tag{37}$$
$$EXPTIME = \bigcup_{d \in \mathbb{N}} TIME\left(2^{n^d}\right) \tag{38}$$

Lemma 4.4

$$TIME(t(n)) \subseteq SPACE(t(n)) \tag{39}$$

The following was presented in the exercise on 12. November 2019.

4.3 Methoden um zu beweisen wo eine Sprache liegt

$L \in \mathcal{L}_{EA}$ EA basteln

$L \notin \mathcal{L}_{EA}$ Lemma 3.3, Pumping-Lemma, Satz 3.1

$L \in P$ Algorithmus angeben der in Polynomieller Zeit entscheidet

$L \notin P$??? Grosses Problem der Informatik

$L \in NP$

L ist NP-vollständig

$L \in \mathcal{L}_R$ Algorithmus angeben, der L entscheidet

$L \notin \mathcal{L}_R$ R-Reduktion oder EE-Reduktion
$L_2 \leq_R L$
Satz von Rice

$L \in \mathcal{L}_{RE}$ TM, die L erkennt

$L \notin \mathcal{L}_{RE}$ EE-Reduktion oder Diagonalargument
Zeigen, dass Komplement $L^C \in \mathcal{L}_{RE} \backslash \mathcal{L}_R$

The following was presented in the lecture on 15. November 2019.

Theorem 4.3

$$SPACE(s(n)) \subseteq \bigcup_{c \in \mathbb{N}} TIME(c^{s(n)}) \quad \forall s(n) \geq \log_2(n) \tag{40}$$

Theorem 4.4

$$DLOG \subseteq P \subseteq PSPACE \subseteq EXPTIME \tag{41}$$

Theorem 4.5

1. t_2 ist Zeitkonstruierbar
2. $t_1(n) \geq n$
3. $t_1(n) \log_2(t_1(n)) = o(t_2(n))$

$$TIME(t_1) \subset_+ TIME(t_2) \tag{42}$$

The following was presented in the lecture on 19. November 2019.

Definition 11.

$$V(A) = \{w \in \sum^{\star} \mid \exists \lambda \in \{0,1\}^{\star} \text{ mit } |x| \subseteq (|w|) \text{ mit } (w,x) \in L(A)\} \tag{43}$$

$$VP \overset{\text{def}}{=} \{V(A) \mid A \text{ ist in Polynomieller Zeit verifizierbar}\} \tag{44}$$

Theorem 4.6

$$VP = NP \tag{45}$$

The following was presented in the exercise on 19. November 2019.

Theorem 4.7: 6.2

$$s_1(n) = o(s_2(n)) \tag{46}$$

$$SPACE(s_1) \subset SPACE(s_2) \tag{47}$$

Es ist eine wirkliche Teilmenge

Theorem 4.8: 6.3

$$t_1(n)\log_2(t_1(n)) = o(t_2(n)) \tag{48}$$

$$TIME(t_1) \subset TIME(t_2) \tag{49}$$

Es ist eine wirkliche Teilmenge

The following was presented in the lecture on 22. November 2019.

Definition 12. L ist NP-schwer, falls für alle $L' \in NP$ gilt $L' \leq_p L$.

Definition 13. L ist NP-vollständig, falls $L \in NP$ und L ist NP-schwer.

Lemma 4.5

$L \in P$ und L NP-schwer $\Rightarrow P = NP$.

Proof 4.1

$L' \in NP$, x Eingabe für L' $\underbrace{\Rightarrow}_{\text{Poly. Reduktion}}$ A(x) Eingabe für L $\underbrace{\Rightarrow}_{\text{Poly. Alg für L}}$ Lösung für L'. □

Definition 14.

$$SAT = \{\Phi \mid \Phi \text{ ist erfüllbare Formel in KNF}\} \tag{50}$$

Theorem 4.9: Satz von Cook

SAT ist NP-vollständig.

Lemma 4.6

$L_1 \leq_p L_2$ und L_1 ist NP-schwer $\Rightarrow$ L_2 ist NP-schwer.

Definition 15.

$$CLIQUE = \{(G,k) \mid G \text{ ist ungerichteter Graph, der k-Clique enthält.}\} \tag{51}$$

Theorem 4.10

$$\text{SAT} \leq_p \text{CLIQUE} \tag{52}$$

The following was presented in the lecture on 26. November 2019.

$$SAT \in P \tag{53}$$

$$SAT2 \in P \tag{54}$$

4.4 SAT3 $\in$ NP

$$SAT \leq_p SAT3 \tag{55}$$

The following was presented in the lecture on 29. November 2019.

Note 18. Man kann jedes Problem in NP beschreiben durch eine Formel.

The following was presented in the exercise on 3. December 2019.

4.5 Schema bei EE-Reduktion $L_1 \leq_{EE} L_2$

1. Überprüfen, ob es die richtige Richtung ist
2. Beweisrahmen: Wir beschreiben ein Algorithmus A, der ein $x \in \sum_{\star}$ in $A(x)$ transformiert, sodass $x \in L_1 \Rightarrow A(x) \in L_2$
3. Form prüfen (falsche Form $\Rightarrow A(x) = \lambda$)
4. Transition ändern / simulieren (Eigene Eingabe ignorieren andere TM Eingabe simulieren.)
5. Beweise $x \in L_1 \Rightarrow A(x) \in L_2$
6. Beweise $x \notin L_2 \Rightarrow A(x) \notin L_2$

4.6 Schema bei R-Reduktion $L_1 \leq_R L_2$

1. Überprüfen, ob es die richtige Richtung ist
2. Beweisrahmen: Angenommen $\exists$ Algorithmus A mit $L(A) = L_2$ dann konstruieren wir Algorithmus B mit $L(B) = L_1$
3. Form prüfen (x falsche Form $\Rightarrow$ Algorithmus B verwirft)
4. Transition ändern / simulieren
5. Resultat von Punkt 3 an Algorithmus A übergeben.
6. Ausgabe von A übernehmen (oder verdrehen)
7. Beweise $x \in L_1 \Rightarrow ... \Rightarrow x \in L(2)$
8. Beweise $x \notin L_2 \Rightarrow ... \Rightarrow x \notin L(B)$

4.7 NP-Vollständigkeit für L

1. In NP
2. Nimm NP-schweres Problem $\hat{L}$ und zeige $\hat{L} \leq_p L$
3. Überprüfen, ob es die richtige Richtung ist?
4. Beweisrahmen:
5. while (Beweis geht nicht && genug Zeit) mache die drei nächsten Schritte:
6. try $A(x)$ = irgendeine Umformung
7. Beweise $x \in \hat{L} \Rightarrow A(x) \in L$
8. Beweise $A(x) \in L \Rightarrow x \in \hat{L}$

4.8 Satz von Rice

1. Satz zeigt nur $\notin L_R$
2. Sprache über TM $L \subseteq$ Kod TM
3. Nicht trivial: $L \neq \varnothing$, $L \neq$ Kod TM
4. Semantisch: $Kod(A) \in L$ und TM B mit $L(A) = L(B) \Rightarrow Kod(B) \in L$

4.9 p-Reduktion

Wir beschreiben einen polynomiellen Algorithmus A, der eine Eingabe für SAT in eine Eingabe x für SAT in eine Eingabe $A(x)$ für VC umwandelt, sodass

$$x \in SAT \Leftrightarrow A(x) \in VC \tag{56}$$

Note 19. Typisch zu zeigen: SAT, 3SAT, CLIQUE, VC, DS, SCP (Wichtigkeit abfolgend)

Note 20. Atypisch zu zeigen: E3SAT, LARGE-CLIQUE

Example 5. Beschreiben Sie eine KNF-Formel, deren Belegungen die möglichen Platzierungen von Türmen auf einem $(n \times n)$-Schachbrett darstellen und die genau dann erfüllt ist, wenn genau n Türme platziert sind, die sich gegenseitig nicht bedrohen.

Jedes Feld (i, j) hat eine Variable x_{ij}
$x_{ij} = 1 \hat{=}$ "Turm ist auf (i, j)"
Für jede Zeile i definieren wir die Klausel Z, wie folgt:

$$Z_i = \bigwedge_{1 \le j_1 < j_2 \le n} (\overline{x_{i,j_1}} \vee \overline{x_{i,j_2}}) \tag{57}$$

Daraus folgt, dass keine zwei Variablen in Zeile i gleichzeitig 1 sind.

$$S_j = \bigwedge_{1 \le i_1 < i_2 \le n} (\overline{x_{i_1,j}} \vee \overline{x_{i_2,j}}) \tag{58}$$

Daraus folgt, dass keine zwei Variablen in Spalte j gleichzeitig 1 sind.

$$F_1 = \bigwedge_{1 \le i \le n} Z_i \wedge \bigwedge_{1 \le j \le n} S_j \tag{59}$$

In allen Zeilen und Spalten ist höchstens ein Turm.

$$\bigvee_{1 \le j \le n} x_{i,j} \hat{=} \text{ in Zeile i steht ein Turm} \tag{60}$$

Damit haben wir

$$F = F_1 \wedge \bigwedge_{1 \le i \le n} \bigvee_{1 \le j \le n} x_{i,j} \tag{61}$$

The following was presented in the lecture on 3. December 2019.

10 Grammatiken

Definition 16. Eine Grammatik G ist ein Quadrupel $G = \left(\sum_N, \sum_T, P, S\right)$ mit

$$\sum_N = \text{Nichtterminalalphabet} \tag{62}$$

$$\sum_T = \text{Terminalalphabet} \tag{63}$$

$$\sum = \sum_N \cup \sum_T, \sum_N \cap \sum_T = \varnothing \tag{64}$$

$$S \in \sum_N \quad \text{Startsymbol} \tag{65}$$

$$P \subseteq \overset{\star}{\sum} \sum_N \overset{\star}{\sum} x \overset{\star}{\sum} \text{Menge von Regeln oder Produktionen} \tag{66}$$

Definition 17.

$$\gamma \Rightarrow_G \delta \tag{67}$$

bedeutet δ ist in einem Schritt auf γ ableitbar.

Definition 18.

$$\gamma \Rightarrow_G^\star \delta \tag{68}$$

bedeutet es ist in beliebig vielen Schritten ableitbar oder die beiden sind gleich.

10.1 Chonsky-Hierarchie

G ist

Typ-0-Grammatik keine weiteren Bedingungen

Typ-1-Grammatik (kontextsensitiv) $\alpha \to \beta \in P: \quad |\alpha| \leq |\beta|$

Typ-2-Grammatik (kontextfrei) $\alpha \to \beta \in P: \quad \alpha \in \sum_N$ und $\beta \in \left(\sum\right)^\star$

Typ-3-Grammatik (regulär) $\alpha \to \beta \in P: \quad \alpha \in \sum_N$ und $\beta \in \left(\sum_T\right)^\star \cdot \sum_N \cup \left(\sum_T\right)^\star$

Theorem 10.1

Sei A ein Endlicher Automat. Dann existiert eine reguläre Grammatik G mit

$$L(G) = L(A) \tag{69}$$

Definition 19. Eine reguläre Grammatik $G = \left(\sum_N, \sum_T, P, S\right)$ heisst normiert, wenn alle Regeln die folgende Form haben:

- $S \to \lambda$
- $A \to a,\ A \in \sum_N, \quad a \in \sum_T$
- $B \to bC,\ B, C \in \sum_N, \quad b \in \sum_T$

Theorem 10.2

$$\mathcal{L}_3 = \mathcal{L}_{EA} \tag{70}$$

The following was presented in the lecture on 13. December 2019.

10.2 Kontextfreie Grammatiken

Definition 20. $G = \left(\sum_N, \sum_T, P, S\right)$ ist kontextfrei, falls alle Regeln der Form $A \to \alpha$ halten für $A \in \sum_N,\ \alpha \in \left(\sum_N \cup \sum_T\right)^\star$

Definition 21. Ein Ableitungsbaum/Syntaxbaum T für G ist ein markierter geordneter Baum mit folgenden Eigenschaften:

- T hat eine Wurzel die mit S markiert ist.
- Jeder Knoten ist mit einem Symbol aus $\sum_N \cup \sum_T \cup \{\lambda\}$ markiert
- Innere Knoten sind mit einem Symbol aus $\sum_N$ markiert.
- Alle Blätter sind mit einem Symbol aus $\sum_T \cup \{\lambda\}$ markiert.
- Ein innerer Knoten der mit A markiert ist hat Kinder α:

$$\to A \to \alpha_1\alpha_2...\alpha_k \in P \tag{71}$$

Definition 22. G ist in Chomsky-Normalform, falls alle Regeln in der Form

$$A \to BC \quad \text{oder} \quad A \to \alpha \tag{72}$$

G ist in Greibach-Normalform, falls alle Regeln in der Form

$$A \to a\alpha \tag{73}$$

Theorem 10.3

Für jede kontextfreie Grammatik G existieren zu G äquivalente Grammatiken in Chomsky-Normalform und in Greibach-Normalform.

The following was presented in the lecture on 17. December 2019.

Note 21. Ein nichtdeterministischer Kellerautomat nutzt die Datenstruktur des Kellers (Last-in-first-out-Speicherstrategie).

Definition 23. Ein nicht deterministischer Kellerautomat (NPdA) ist ein 6-Tubel:
$M = \left(Q, \sum, \Gamma, \delta, q_0, Z_0\right)$

Q Zustandsmenge

$\sum$ Eingabealphabet

Γ Kelleralphabet

$q_0 \in Q$ Anfangszustand

$Z_0 \in \Gamma$ Initialisierungssymbol des Kellers

δ ist eine Abbildung von $Q \times \left(\sum \cup \{\lambda\}\right) \times \Gamma$ in endliche Teilmengen von $Q \times \Gamma^\star$

Lemma 10.1

Sei L eine kontextfreie Sprache. Dann existiert ein NPdA M, so dass $L = L(M)$ gilt.

The following was presented in the exercise on 17. December 2019.

Example 6. HS18 2b)
Diese Sprache wird wie folgt erzeugt:

$$S \Rightarrow_G^\star SSSSS \Rightarrow_G^\star SS[S][S]S \Rightarrow_G^\star SS[[[[S]]]][[S][S]]S \tag{74}$$

$$L(G) = \{x \in \{[]\}^\star \mid \forall \text{Präfix } x_i \text{ mit } |x_i|_[\geq |x_i|_] \text{ und } |x|_[= |x|_]\} \tag{75}$$

$L(G) \notin \mathcal{L}_{EA}$: Sei $L(G)$ regulär, dann existiert ein Automat $A(E, Q, \delta, q_0)$ mit $L(A) = L(G)$
Betrachte die Wörter $[^1, [^2, ..., [^{|Q|+1}$. Nach Schubfacherprinzip gibt es $i < j$ sodass $\hat{\delta}\left(q_0, [^i\right) = \hat{\delta}\left(q_0, [^j\right)$
Nach Lemma 3.3 gilt $\forall z \in \{[,]\}^\star$

$$[^i z \in L(A) \Leftrightarrow [^j z \in L(A) \tag{76}$$

Aber für $z =]^i$ ist $[^i z = [^i]^i \in L(A)$, aber $[^j z = [^j z \notin L(A)$, da $|[^j z|_[> |[^j z|_]$
$\Rightarrow$ Widerspruch

Example 7. HS18 3b)
Angenommen L sei Kontextfrei. Dann existiert ein $n_L \in \mathbb{N}$ mit den Eigenschaften des Pumpinglemmas. Betrachte $z = a^{n_L} b^{2n_L} c^{3n_L}$ und es gilt $|z| \geq n_L$. Sei $z = uvwxy$ mit den Eigenschaften des Pumpinglemmas. Wegen (ii) besteht vwx aus höchstens 2 verschiedenen Buchstaben. Wegen (i) besteht vx aus mindestens einem Buchstaben. Beim pumpen können also nicht a,b und c gleichzeitig gepumpt werden und das entstandene Wort ist noch in L, was die Bedingung (iii) verletzt.

10.3 Wichtig für Prüfung

- Letzte beiden Grammatik Übungsserien
- Definitionen $L_0 < L_1 < L_2 < L_3$
- Pumping-Lemma kontextfreie Sprachen
- Regularitätsbeweise
- NP-Vollständigkeitsbeweis
- Kolmogorovaufgabe
- Beweisideen auswendig lernen / Schema

The following was presented in the lecture on 20. December 2019.

10.4 Randomisierte Algorithmen

Randomisierte Algorithmen sind die "Lösung" zu NP-Problemen.

Theoretische Informatik - Beweismuster

1 Pumping-Lemma

Sei L regulär. Dann existiert eine Konstante $n_0 \in \mathbb{N}$, so dass sich jedes Wort $w \in \sum^\star$ mit $|w| \geq n_0$ in drei Teile y,x und z zerlegen lässt, das heisst w = yxz, wobei

1. $|yx| \leq n_0$
2. $|x| \geq 1$
3. entweder $\{yx^kz \mid k \in \mathbb{N}\} \subseteq L$ oder $\{yx^kz \mid k \in \mathbb{N}\} \cap L = \varnothing$

Wir wählen nun das Wort $w = ...$ Offensichtlich gilt $|w| \geq n_0$.
~~Es muss also eine Zerlegung $y = ..., x = ..., z = ...$ existieren.~~
Zerlegung erzwingen: $\overset{\text{(i)}}{\Rightarrow} \ ... \ xy \ ... \ \overset{\text{(ii)}}{\Rightarrow} x$
x benutzen um Wort zu zerstören.
Offensichtlich ist (iii) nicht erfüllt $\Rightarrow$ L ist nicht regulär.

2 Kolmogorov-Komplexität

Angenommen L sei regulär. Betrachte für $i \in \mathbb{N}$ die unendliche Folge von Sprachen:

$$L_i = \{x \in \{0,1\}^\star \mid var1 \ x \in L\} \tag{1}$$

($var1$ von i abhängig)
Auch genannt Präfixsprache.
Für alle $i \in \mathbb{N}$ ist das erste Wort in L_i $var2$ (auch von i abhängig)
Nach Satz 3.1 gilt also:

$$\forall i K(var2) \leq \lceil \log_2(n+1) \rceil + const \quad \text{Konstant!} \tag{2}$$

Dies ist jedoch ein Widerspruch, weil es endlich viele Programme kürzer als $1 + c$ gibt, jedoch unendlich viele Wörter der Form $var2$ gibt.

3 Kolmogorov Theorem

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

$$K(x) \leq |x| + d \tag{3}$$

Sei $L \subseteq (\sum_{bool})^\star$ eine reguläre Sprache. Sei $L_x = \{y \in (\sum_{bool})^\star \mid xy \in L\}$ für jedes $x \in (\sum_{bool})^\star$. Dann existiert eine Konstante const, so dass für alle $x, y \in (\sum_{bool})^\star$:

$$K(y) \leq \lceil \log_2(n+1) \rceil + const \tag{4}$$

falls y das n-te Wort in der Sprache L_x ist.

4 Lemma 3.3

Angenommen L_1 sei regulär. Also existiert ein EA $A = (Q, \{0,1\}, \delta, q_0, F)$ mit $L(A) = L_1$
Wir betrachten folgende $|Q| + 1$ verschiedenen Wörter:
...
Da der EA A nur $|Q|$ verschiedene Zustände hat, bedeutet das, dass es $1 \leq i < j \leq |Q| + 1$ gibt, sodass $\hat{\delta}(q_0, ...^i) = \hat{\delta}(q_0, ...^j)$.
Laut Lemma 3.3 gilt für alle $z \in \{0,1\}^\star$, dass

$$...^i z \in L_1 \Leftrightarrow ...^j z \in L_1 \tag{5}$$

Wir wählen nun $z = ...$ Offensichtlich gilt $...^i... \notin L_1$ und $...^j... \in L_1$, also ergibt sich ein Widerspruch.
Daraus können wir schliessen, dass unsere ursprügnliche Annahme falsch ist. Somit ist L_1 nicht regulär.

5 EE-Reduktion

Wir beschreiben einen Algorithmus A, der eine Eingabe $x \in \{0,1\}^\star$ in $A(x)$ transformiert, sodass gilt:

$$x \in L_1 \Leftrightarrow A(x) \in L_2 \tag{6}$$

und zeigen somit $L_1 \leq_{EE} L_2$
b A arbeitet wie folgt: 1. 2. 3.
Beispielschritte:

1. (Überprüfe Eingabe)
2. Finde i sodass $x = w_i$
3. Generiere M_i und leite alle reject zu einer Endlosschleife
4. Setze $A(x) = ...$ (Eingabe für L_2)

Beweise nun $x \in L_1 \Leftrightarrow A(x) \in L_2$ (Sorgfälltig wenn man Rückrichtung in einem zeigt!)

$$\begin{aligned}
x \in L_1 &\Rightarrow x = w_i \text{ und } M_i \text{ ist die i-te TM} && (7)\\
&\Rightarrow \text{i-te TM } M_i \text{ akzeptiert } w_i && (8)\\
&\Rightarrow A(x) = ... && (9)\\
&\Rightarrow A(x) \in L_2 && (10)\\
A(x) \in L_2 &\Rightarrow ... && (11)\\
&\Rightarrow x \in L_1 && (12)\\
\text{Oder } x \notin L_1 &\Rightarrow ... && (13)\\
&\Rightarrow A(x) \notin L_2 && (14)
\end{aligned}$$

Somit ist $L_2 \notin L_{RE}$, weil $L_1 \notin L_{RE}$

6 R-Reduktion

Zeige $L_1 \leq_R L_2$
Sei L_2 regulär. Wir beschreiben einen Algorithmus A, der x als Eingabe nimmt und unter Verwendung eines Algorithmus B mit $L(B) = L_2$ die Sprache L_1 entscheidet.
A arbeitet wie folgt: 1. 2. 3.
Beispielsätze"

1. Wir nehmen x und berechnen das i sodass x das i-te Wort ist.
2. Wir generieren M_i sodass M_i die i-te TM ist.
3. Wir simulieren B auf ...
4. Falls B akzeptiert dann ... sonst ...

Nun beweisen wir $L(A) = L_1$

$$
\begin{aligned}
x \in L_1 &\Rightarrow w_i \text{ und } M_i \text{ hält auf } w_i \text{ nicht.} && (15)\\
&\Rightarrow B \text{ hält auf ... nicht} && (16)\\
&\Rightarrow \text{A akzeptiert} && (17)\\
&\Rightarrow x \in L(A) && (18)\\
& && (19)\\
x \notin L_1 &\Rightarrow w_i \text{ und } M_i \text{ hält auf } w_i. && (20)\\
&\Rightarrow B \text{ hält auf ...} && (21)\\
&\Rightarrow \text{A verwirft} && (22)\\
&\Rightarrow x \notin L(A) && (23)
\end{aligned}
$$

Nun haben wir einen Widerspruch, da L_1 nicht in L_R ist $\Rightarrow L_2$ nicht regulär.

7 P-Reduktion

Wir beschreiben einen Algorithmus A, der in polynomieller Zeit eine Eingabe $x \in \{0,1\}^\star$ in $A(x)$ transformiert, sodass gilt:

$$x \in L_1 \Leftrightarrow A(x) \in L_2 \tag{24}$$

und zeigen somit $L_1 \leq_p L_2$
A arbeitet wie folgt: 1. 2. 3.
Offensichtlich ist diese Umformung in $\mathcal{O}(n^k)$ / polynomieller Zeit möglich.
Wir zeigen nun x erfüllbar $\Leftrightarrow A(x)$ erfüllbar

8 Konstruktiver Beweis

Wir konstruieren eine TM A die L erkennt (L(A) = L). A arbeitet auf der Eingabe $x \in \{0.1\}^\star$ wie folgt:

Wir prüfen, ob die Eingabe x die Form ... hat, falls nicht verwerfen wir.
Falls die Eingabe die richtige Form, sei $M_1, ...$ sodass $x = ...$
Beispielsatz: $\varnothing \Rightarrow$ Wir simulieren für alle $i \in \mathbb{N}$ die ersten i Schritte von $M_1, ...$ auf den ersten i Wörtern.
Falls $M_1, ...$ akzeptieren, akzeptieren wir auch.
Nun beweisen wir $L(A) = L$:

$$
\begin{aligned}
x \in L : \Rightarrow\ & \text{Es existiert } w = ... && (25)\\
& \text{und... } \neq \varnothing && (26)\\
& \Rightarrow \exists w : ... \text{ akzeptiert} && (27)\\
& \Rightarrow \text{w ist das i-te Wort und das wird in einem Schritt der Simulation akzeptiert} && (28)\\
& \Rightarrow x \in L(A) && (29)
\end{aligned}
$$

$$
\begin{aligned}
x \notin L : \Rightarrow\ & \text{Falls Zerlegung } w = ... \text{sonst verwerfen} && (30)\\
& \Rightarrow \forall w : ... \text{ nicht akzeptiert} && (31)\\
& \Rightarrow \text{w ist das i-te Wort und das wird in keinem Schritt der Simulation akzeptiert} && (32)\\
& \Rightarrow x \notin L(A) && (33)
\end{aligned}
$$

9 Pumping Lemma für Kontextfreie Sprachen

Sei L kontextfrei. Dann existiert eine nur von L abhängige Konstante n_L, so dass für alle Wörter $z \in L$ mit $|z| \geq n_L$ eine Zerlegung:

$$z = uvwxy \tag{34}$$

von z existiert, so dass

1. $|vx| \geq 1$
2. $|vwx| \leq n_L$
3. $\{uv^iwx^iy \mid i \in \mathbb{N}\} \subseteq L$

10 Sprache $\notin L_R$

Um zu zeigen, dass $L_x \notin L_R$ gilt, geben wir eine konkrete Reduktion an. Wir wissen, dass $L_U \notin L_R$ und zeigen deshalb $L_U \leq_{EE} L_x$. Daraus folgt dann $L_x \notin L_R$.
Wir geben einen Algorithmus A an, der eine Eingabe x für L_U in eine Eingabe für L_x so umwandelt, dass $x \in L_U \Leftrightarrow A(x) \in L_x$. Zunächst prüft unser Algorithmus, ob die Eingabe die richtige Form hat. Falls nicht, gibt A das leere Wort weiter. Falls schon, konstruieren wir eine TM M', die für jede Eingabe die Arbeit von M auf w simuliert. A gibt $Kod(M')$ weiter.
Wir zeigen nun die Korrektheit unserer Reduktion:

$$x \in L_U \Leftrightarrow x = Kod(M)\#w \text{ und M akzeptiert w.} \quad \Leftrightarrow A(x) = ... \Leftrightarrow ... \quad \Leftrightarrow A(x) \in L_x \tag{35}$$

11 Mindestens k Zustände

Wir verwenden Lemma 3.3, um zu zeigen, dass jeder deterministische EA, der L akzeptiert, mindestens k Zustände haben muss. Hierfür betrachten wir die Wörter:
...
Wir zeigen nun, dass es für jedes Paar (w_i, w_j) von Wörtern aus $\{w_0, ..., w_s\}$ mit $i < j$ ein Wort $z_{i,j} \in \{0,1\}^\star$ gibt, sodass:

$$w_i z_{i,j} \in L \text{ und } w_j z_{i,j} \notin L \tag{36}$$

Damit folgt dann aus Lemma 3.3 dass w_i und w_j jeden EA für L in verschiedene Zustände führen müssen, für alle $i, j \in \{0, ..., k\}$. Entsprechende Wörter $z_{i,j}$ sind in der folgenden Tabelle gezeigt: **Fancy k*k Tabelle mit rechts oben ausfüllen.**

12 Monoton 3SAT

$$(x_a \vee x_b \vee x_c) = (x_a \vee a \vee \overline{b}) \wedge (x_b \vee \overline{a} \vee b) \wedge (x_c \vee \overline{a} \vee \overline{b}) \wedge (a \vee b) \tag{37}$$

13 Entwerfe Grammatik

Die Grammatik $G = (\{S, A, ...\}, \{a, b, ...\}, P, S\}$ mit
$P = \{S \to ..., ... \to \lambda\}$

14 NP-vollständig

Damit eine Sprache L NP-vollständig ist muss $L \in NP$ und L ist NP-schwer gelten.
Wir zeigen nun ... $\in NP$. Ein p-Verifizierer A kann wie folgt arbeiten:
...
Nun zeigen wir, dass ... NP-schwer ist. Wir wissen, dass ... NP-schwer ist. Deshalb gilt, falls ... $\leq_p$ ..., dann ist auch ... NP-schwer.
...
Da $L \in NP$ und L ist NP-schwer muss gelten, dass sie NP-vollständig ist.

15 Sprachen

$$L_U = \{Kod(M)\#w \mid w \in (\textstyle\sum_{bool})^\star \text{ und } M \text{ akzeptiert } w\} \tag{38}$$
$$L_H = \{Kod(M)\#w \mid w \in (\textstyle\sum_{bool})^\star \text{ und } M \text{ hält auf } w\} \tag{39}$$
$$L_{diag} = \{w \in (\textstyle\sum_{bool})^\star \mid w = w_i \text{ für ein } i \in \mathbb{N} - \{0\} \text{ und } M_i \text{ akzeptiert } w_i \text{ nicht}\} \tag{40}$$
$$= \{w \in (\textstyle\sum_{bool})^\star \mid w = w_i \text{ für ein } i \in \mathbb{N} - \{0\} \text{ und } d_{ii} = 0\} \tag{41}$$
$$L_{empty} = \{Kod(M) \mid L(M) = \varnothing\} \tag{42}$$
$$L_{H,\lambda} = \{Kod(M) \mid M \text{ hält auf } \lambda\} \tag{43}$$
$$L_{EQ} = \{Kod(M_1)\#Kod(M_2) \mid L(M_1) = L(M_2)\} \tag{44}$$

16 Klassen von Sprachen

$$\mathcal{L}_{RE} = \{L(M) \mid \text{M ist eine TM}\} \tag{45}$$

$$\mathcal{L}_{R} = \{L(M) \mid \text{M ist eine TM, die immer hält.}\} \tag{46}$$

ist die Klasse der rekursiven (algorithmisch erkennbaren) Sprachen.