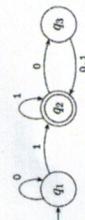


Teil 1

1 Reguläre Sprachen

1.1 DFA



1.1.1 Formale Definition

Ein deterministischer endlicher Automat ist ein 5-Tupel $(Q, \Sigma, \delta, q_0, F)$ mit

- Q ist die Menge der Zustände,
- Σ ist das Alphabet,
- $\delta : Q \times \Sigma \mapsto P(Q)$ ist die Übergangsfunktion,
- $q_0 \in Q$ ist der Startzustand,
- $F \subseteq Q$ ist die Menge der akzeptierenden Zustände.

Bemerkungen

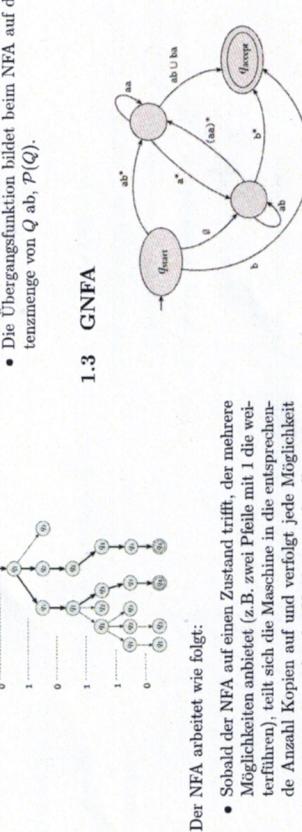
- Σ_ϵ bedeutet $\Sigma \cup \epsilon$. Der NFA kann also auch den Zustand wechseln, ohne ein Zeichen gelesen zu haben.
- Jeder NFA verfügt über einen äquivalenten DFA.

1.2 NFA

- Ein DFA kann 0 akzeptierende Zustände besitzen.
- Für jedes Symbol muss von jedem Zustand ein Pfeil wegführen.

1.2 Äquivalenz mit endlichen Automaten

- Sie müssen nicht pro Zustand und Zeichen des Alphabets einen ausgehenden Pfeil besitzen, vielmehr dürfen sie keinen, einen oder mehrere Pfeile für das gleiche Symbol besitzen.
- Die Übergangsfunktion bildet beim NFA auf die Potenzmenge von Q ab, $P(Q)$.



- Der NFA arbeitet wie folgt:
- Sobald der NFA auf einen Zustand trifft, der mehrere Möglichkeiten anbietet (z.B. zwei Pfeile mit 1 die wegführen), teilt sich die Maschine in die entsprechende Anzahl Kopien auf und verzweigt jede Möglichkeit mit einer entsprechenden Kopie von sich selbst weiter.

- Wenn die Maschine in einen Zustand gelangt, der eine ϵ -Transformation anbietet, wird für diese eben falls eine Kopie erstellt. Danach wird mit der NFA-Berechnung weitergefahren.
- Landet eine Kopie in einem Zustand in dem sie nicht mehr weiterkommt, stirbt sie.
- Sobald eine der Kopien in einem akzeptierenden Zustand landet, akzeptiert der ganze NFA.

1.3.1 Formale Definition

Ein generalisierter nichtdeterministischer endlicher Automat ist ein 5-Tupel $(Q, \Sigma, \delta, q_{start}, q_{accept})$ mit

- Q ist die endliche Menge der Zustände,
- Σ ist das Alphabet,
- $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow \mathcal{P}(Q)$ ist die Übergangsfunktion,
- q_{start} ist der Startzustand,
- q_{accept} ist der Endzustand.

Bemerkungen

- Der Startzustand hat Transitionen zu allen anderen Zuständen, aber es gibt keine, die von anderen Zuständen auf den Startzustand führen.
- Es gibt nur einen akzeptierenden Zustand. Von jedem anderen Zustand führt eine Transition auf den akzeptierenden Zustand.
- $\forall q \in Q \setminus \{q_{accept}, q_{start}\} :$
 - Es führt eine Transition zu allen anderen Zuständen.
 - Es führt eine Transition auf sich selbst
 - Es gilt $q_{start} \neq q_{accept}$.
- Wird für die Umwandlung eines DFAs in einen regulären Ausdruck angewendet, erhält die resultierende Menge immer den leeren String!

Bemerkungen

- Ein GNFAs liest die Eingabesymbole blockweise anstatt nur zeichenweise ein.
- Übergangszeichen können zusätzlich mit regulären Ausdrücken beschriftet sein.
- Ein GNFAs liefert $\Sigma \cup \epsilon$. Der NFA kann also auch den Zustand wechseln, ohne ein Zeichen gelesen zu haben.

Bemerkungen

- Wird für die Umwandlung eines DFAs in einen regulären Ausdruck benötigt.
- Ein DFA verfügt über einen äquivalenten DFA.
- Ein DFA kann 0 akzeptierende Zustände besitzen.
- Für jedes Symbol muss von jedem Zustand ein Pfeil wegführen.

1.3.2 Unterschied zum NFA

- Ein GNFAs liest die Eingabesymbole blockweise anstatt nur zeichenweise ein.

1.4 Äquivalenz mit endlichen Automaten

1.4.1 Formale Definition

Definition 1.1. R ist ein regulärer Ausdruck, falls

1. Stern *
2. Konkatenation ◦
3. Vereinigung ∪

1.4.2 Beispiele

- $0^*10^* = \{w|w \text{ enthält genau eine } 1\}$
- $\Sigma^*1\Sigma^* = \{w|w \text{ enthält mindestens eine } 1\}$
- $\Sigma^*001\Sigma^* = \{w|w \text{ enthält } 001 \text{ als Substring}\}$
- $1^*(01^+)^* = \{w|w \text{ ist gefolgt von mind. einer } 1\}$
- $(\Sigma\Sigma)^* = \{w|w \text{ ist ein String mit gerader Länge}\}$
- $(\Sigma\Sigma\Sigma)^* = \{w|w \text{ die Länge von } w \text{ ist ein Vielf. von } 3\}$
- $01 \cup 10 = \{01, 10\}$

1.4.3 Vereinigung ∪

Zwei reguläre Ausdrücke können vereinigt werden:

$$R_1 \cup R_2$$

Bemerkungen

- $R \cup \emptyset = R$

1.4.4 Konkatenation ◦

$R_1 \circ R_2$ konkateniert die beiden regulären Ausdrücke R_1 und R_2 . Die Kurzschreibweise hierfür ist R_1R_2 .

Bemerkungen

- $R \circ \epsilon = R$
- $R \circ \emptyset = \emptyset$

1.4.5 Stern *

R^* enthält alle Strings, welche 0 oder mehrere Konkatenationen von R sind. R^+ ist eine Kurzschreibweise für RR^* und enthält alle Strings, welche 1 oder mehrere Konkatenationen von R sind. R^k ist die k-fache Konkatenation von R .

Beispiele

$$\begin{aligned} \text{Gegeben sei } R = aa^*b: \\ R^* &= (aa^*b)^* = \{\epsilon, ab, aab, \dots, aabab, aabaaa, \dots\} \\ R^+ &= (aa^*)^+ = \{ab, aab, \dots, aabab, aabaaa, \dots\} \\ R^3 &= (aa^*)^3 = \{aab, aabab, aabaaa, \dots\} \end{aligned}$$

Gegeben sei $R = aa^*b:$

$$R^* = 1^*$$

$$R^+ = 1^+$$

$$R^3 = 1^3 = \{111\}$$

Gegeben sei $R = aab^*b:$

$$R^* = (aab^*)^* = \{ab, aab, aabab, aabaaa, \dots\}$$

$$R^+ = (aab^*)^+ = \{aab, aabab, aabaaa, \dots\}$$

$$R^2 = (aab^*)^2 = \{aabab, aabaaa, \dots\}$$

Bemerkungen

- Wird der Stern-Operator auf einen regulären Ausdruck angewendet, erhält die resultierende Menge immer den leeren String!
- $R^+ \cup \epsilon = R^*$

1.4.6 Operator-Reihenfolge

Reguläre Ausdrücke werden in folgender Reihenfolge ausgewertet:

1.4.7 Beispiele

1. Stern *

2. Konkatenation ◦

3. Vereinigung ∪

1.4.8 Äquivalenz mit endlichen Automaten

Definition 1.2. Eine Sprache ist regulär wenn es einen regulären Ausdruck gibt, der die Sprache beschreibt.

Ein endlicher Automat kann in einen regulären Ausdruck überführt werden und umgekehrt.

1.4.9 Äquivalenz mit endlichen Automaten

Definition 1.3. GNFAs

Zwei reguläre Ausdrücke können vereinigt werden:

$$R_1 \cup R_2$$

Bemerkungen

- $R \cup \emptyset = R$

- $(0 \cup \epsilon)^* = 01^* \cup 1^*$
- $(0 \cup \epsilon)(1 \cup \epsilon) = \{\epsilon, 0, 1, 01\}$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \{\epsilon\}$

2.1.3 Chomsky-Normalform

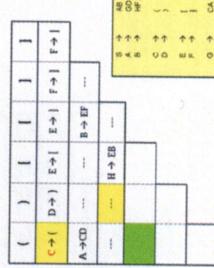
Definition 2.2. Eine Grammatik liegt in Chomsky-Normalform vor, wenn jede Regel in einer der beiden Formen vorliegt:

$$\begin{array}{l} A \rightarrow BC \\ A \rightarrow a \end{array}$$

a ist ein beliebiges Terminal und A, B, C sind Nonstermine. Zusätzlich ist die Produktion $S \rightarrow \epsilon$ erlaubt, wobei S das Startsymbol ist.

2.1.4 CYK-Algorithmus

Damit lässt sich herausfinden, ob ein bestimmtes Wort von einer Grammatik produziert werden kann bzw. ob es du der kontextfreien Sprache gehört. Das Wort wird von der Grammatik erzeugt, wenn das Startsymbol im untersten Kasten des Schemas enthalten ist.



Bemerkungen

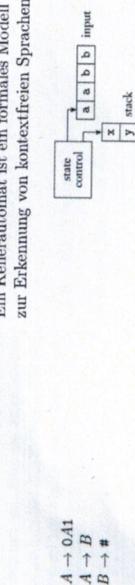
- Σ ist eine endliche Menge von Terminalen, diskunkt zu V ,
- R ist eine endliche Menge von Produktionsregeln bestehend aus einem Nonterminal und einem String bestehend aus Nonterminalen und Terminalen, und
- $S \in V$ ist das Startsymbol.

Definition 2.1. Wendet man ausgehend vom Startsymbol, eine Sequenz von Produktionen solange an, bis nur noch Terminals übrigbleiben, spricht man von einer Ableitung.

2.1.2 Ableitung und Parsebaum

$w \Rightarrow^* w'$ bedeutet die Anwendung einer einzelnen Produktionsregel. $w \Rightarrow^* w'$ bedeutet die Anwendung mehrerer Produktionsregeln.

Grammatik:



Beispielableitung:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111$$

2.2.1 Formale Definition

Ein Kellerautomat ist ein 6-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, F)$ mit

- Q ist die Menge von Zuständen,
- Σ ist das Eingabealphabet,
- Γ ist das Kelleralphabet,
- $\delta : Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)$ ist die Übergangsfunktion,
- $q_0 \in Q$ ist der Startzustand,

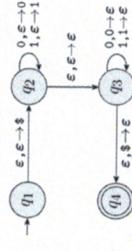
- $F \subseteq Q$ ist die Menge der akzeptierenden Zustände.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ ist die Übergangsfunktion.
- $q_0 \in Q$ ist der Startzustand,
- $q_{accept} \in Q$ ist der akzeptierende Zustand, und
- $q_{reject} \in Q$ ist der verworfene Zustand, wobei $q_{reject} \neq q_{accept}$.

Bemerkungen

- Ein Kellerautomat entspricht einem NFA mit zusätzlichem Stack.
- Der Stack kann eine unendliche Anzahl von Symbolen speichern.
- Kellerautomaten sind gleich mächtig wie kontextfreie Grammatiken.

2.2.2 Beispiel

Der nachfolgende Kellerautomat erkennt die Sprache $\{ww^R | w \in \{0, 1\}^*\}$ (Palindrome).



Bemerkungen

- $a, b \rightarrow c$ bedeutet: Wenn die Maschine auf dem Eingabeband ein Symbol a gelesen hat, wird auf dem Stack das oberste Symbol b durch c ersetzt. Falls a leer ist, heisst das, dass der Kellerautomat den Überhang machen kann, ohne ein Zeichen vom Eingabeband zu lesen. Falls b leer ist, heisst das, dass c auf den Stack gelegt wird. Falls c leer ist, heisst das, dass b vom Stack gelöscht wird.

- Laufzeit $O(n^3)$, Speicherplatz $O(n^2)$ für ein Wort der Länge n .

3 Turingmaschinen

3.1 Einband-TM

Eine TM besteht aus:

- einem unendlichen Band
 - einer endlichen Kontrolle (das Programm)
 - einem Lese- und Schreibkopf, der sich in beide Richtungen auf dem Band bewegen kann
- $M_2 = \text{"On input string } w: \text{1. Sweep left to right across the tape, crossing off every other 0. 2. If in stage 1 the tape contained a single 0, accept. 3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject. 4. Return the head to the left-hand end of the tape. 5. Go to stage 1."}$

3.2 Turingmaschinen

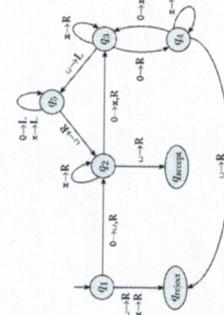
3.1.4 Beispiel

Beispiel einer Turingmaschine, die prüft, ob das Eingabewort zur Sprache $\{0^n | n \geq 0\}$ gehört.

Beschreibung der TM:

- δ : On input string w :
 - sweep left to right across the tape, crossing off every other 0.
 - If in stage 1 the tape contained a single 0, accept.
 - If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject.
 - Return the head to the left-hand end of the tape.
 - Go to stage 1.

Graph:



3.1.1 Formale Definition

Eine Turingmaschine ist ein 7-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ mit

- Q ist die Menge der Zustände,
- Σ ist das Eingabealphabet (ohne Blank-Symbol),
- Γ ist das Bandalphabet mit $\cdot \in \Gamma$ und $\Sigma \subseteq \Gamma$,
- $\delta : Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)$ ist die Übergangsfunktion,
- $q_0 \in Q$ ist der Startzustand,

Berechnung für die Eingabe 0000:

q_0000	$\text{xx0x}0x0xu$
ω_{0000}	$q_0x0x0xu$
$\text{xx}0x00$	$\omega_{q_0}0x0xu$
$\text{xx}0x_0$	$\omega_{q_0}0x0xu$
$\text{xx}0x_0q_0$	$\omega_{q_0}0x0xu$
$\text{xx}0x_0xu$	$\omega_{q_0}0x0xu$

Bemerkungen

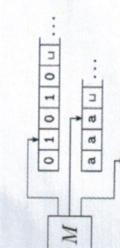
- Die TM hält, wenn q_{accept} oder q_{reject} erreicht wird.
- Im anderen Fall rechnet die TM unendlich lange weiter.

3.2 Mehrband-TM

Eine k-Band-TM besteht aus:

- einer endlichen Kontrolle (das Programm)
- einem endlichen Eingabeband mit Lesekopf
- k Arbeitsbändern mit je einem Lese-/Schreibkopf

Schema:



Übergangsfunktion:

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$$

3.2.1 Simulation

Eine k-Band-TM kann auf einer Einband-TM simuliert werden, indem der Inhalt aller k Bänder auf das eine Band geschrieben wird (getrennt mit #). Um die Kopfposition zu speichern, wird das Baudalphabet der Einband-TM verdoppelt (Symbole mit Punkt als Markierung. Der Punkt repräsentiert die Kopfposition).

Aus dem Satz von Rice folgt

- dass es keinen Algorithmus gibt, der für jede TM entscheidet, ob sie für jede Eingabe hält.
- dass es nicht entscheidbar ist, ob eine TM eine bestimmte Funktion berechnet.
- dass es nicht entscheidbar ist, ob zwei Programme äquivalent sind (für die gleiche Eingabe die gleiche Ausgabe produzieren).

3.3 Nichtdeterministische TM

Der wesentliche Unterschied zur normalen Turingmaschine liegt in der Übergangsfunktion. Diese lautet:

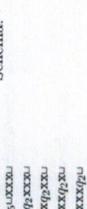
$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Bemerkungen

- Jede nichtdeterministische TM hat eine äquivalente deterministische TM.
- Sobald eine Berechnung im Berechnungsbaum den akzeptierenden Zustand erreicht, hält die TM.

3.4 Enumerator

Ein Enumerator drückt alle Wörter einer Sprache.



4 Asymptotische Notation

5.4 Implikation \rightarrow

a	b	$a \rightarrow b$
f	f	w
w	w	w

- $\mathcal{O}(1)$: Laufzeit immer gleich, unabhängig von der Inputgröße n .
- $\mathcal{O}(n)$: Laufzeit nimmt linear zur Inputgröße n zu.
- $\mathcal{O}(n^2)$: Laufzeit nimmt quadratisch zur Inputgröße n zu.

Nachfolgende Schreibweisen sind äquivalent:
 $a \rightarrow b \quad \neg a \vee b \quad \neg(a \wedge \neg b) \quad \neg b \rightarrow \neg a$

- $\mathcal{O}(2^n)$: Laufzeit verdoppelt sich mit jedem zusätzlichen Element in der Inputgröße n .

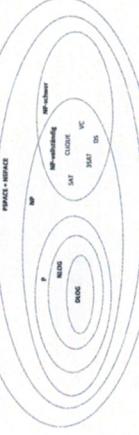
\dots

- $\mathcal{O}(2^{2^n})$
- $\mathcal{O}(2^{3^n})$
- $\mathcal{O}(2^{4^n})$

\dots

\dots

\dots



5 Aussagenlogik

5.1 Begriffe

Tautologie: Eine aussageologische Formel, die immer wahr ist.

6 Komplexitätstheorie

Definition 6.1. $TIME(t(n))$ ist die Menge aller Sprachen, die von einer deterministischen TM in $O(t(n))$ Zeit entschieden werden kann.

Kontraposition: Eine aussagenlogische Formel, die immer falsch ist.

Konjunktive Normalform: $\bigwedge_i V_j(\neg)x_{ij}$

Disjunktive Normalform: $V_i \bigwedge_j (\neg)x_{ij}$

6.2 Deterministische Klassen

Definition 6.2. $NTIME(t(n))$ ist die Menge aller Sprachen, die von einer nichtdeterministischen TM in $O(t(n))$ Zeit entschieden werden kann.

NP-Vollständigkeit
 $\neg(a \wedge b) = \neg a \vee \neg b$
 $\neg(a \vee b) = \neg a \wedge \neg b$

Bemerkungen

- Die Regeln lassen sich auch für Verknüpfungen beliebig vieler Elemente erweitern.

6.2.3 NP-Vollständigkeit

Definition 6.3. Eine Sprache ist NP-schwer, falls für alle Sprachen L' in NP gilt $L' \leq_p L$. Eine Sprache ist NP-vollständig, falls

- $L \in NP$, und
- L ist NP-schwer.

6.3 Reduktionen

$A \leq_p B$ bedeutet:

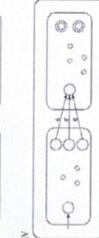
- Problem A lässt sich in polynomiellem Zeit (effizient) auf Problem B reduzieren.
- Falls eine Lösung zu Problem B gefunden wird, kann auch Problem A gelöst werden.

7 Beweise

7.1 Reguläre Sprache ist geschlossen unter Vereinigung

Definition 7.1. Die Klasse der regulären Sprachen ist geschlossen unter Vereinigung. Wenn A_1 und A_2 zwei reguläre Sprachen sind, so ist auch $A_1 \cup A_2$ eine reguläre Sprache.

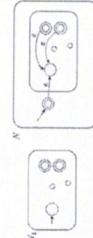
Beweis. Weil A_1 und A_2 regulär sind, gibt es einen endlichen Automaten M_1 der A_1 akzeptiert und einen anderen Automaten M_2 der A_2 akzeptiert. Wir bauen einen neuen Automaten M der die Eingabe akzeptiert, wenn entweder M_1 oder M_2 die Eingabe akzeptiert. Hierzu simuliert M die beiden Automaten schrittweise. M muss die Zustände beider Automaten M_1 und M_2 verwalten. Wenn M_1 k_1 Zustände und M_2 k_2 Zustände besitzt, muss M insgesamt $k_1 \times k_2$ Zustände besitzen, um alle Kombinationen abdecken zu können. Die akzeptierenden Zustände von M sind all die Zustände, in denen ein akzeptierender Zustand von M_1 oder M_2 enthalten ist. \square



7.3 Reguläre Sprache ist geschlossen unter Stern

Definition 7.3. Die Klasse der regulären Sprachen ist geschlossen unter Stern. Wenn A_1 eine reguläre Sprache ist, so ist auch A_1^* eine reguläre Sprache.

Beweis. Weil A_1 regulär ist, gibt es einen endlichen Automaten M_1 der A_1 akzeptiert. Wir modifizieren M_1 so, dass eine ϵ -Transformation von allen akzeptierenden Zuständen auf den Startzustand führen. Die neue Maschine N besitzt einen akzeptierenden Startzustand (ϵ ist auch Teil des Startzustands und von diesem führt eine ϵ -Transformation auf den Startzustand von N_1). \square



Bemerkungen:

- Das leere Wort ϵ ist immer Teil von A^* , ganz egal was A ist.

7.4 Sprache ist regulär

Eine Sprache ist regulär, wenn

- ein DFA existiert, der die Sprache akzeptiert.
- ein NFA existiert, der die Sprache akzeptiert.
- ein regulärer Ausdruck existiert, der die Sprache akzeptiert.
- eine reguläre Grammatik existiert, welche die Sprache erzeugt.

- Bemerkungen
- Sequentialles Simulieren (zuerst M_1 dann M_2) ist nicht möglich, da das Band nicht zurückgespult werden kann.
 - Ein Problem liegt in NP, wenn einer der folgenden Punkte zutrifft:
 - Eine (womöglich geratene) Lösung kann ich polynomieller Zeit verifiziert werden.
 - ...

- Bemerkungen
- Zuerst wird immer das Zeichen gelesen und erst dann wird den ϵ -Pfeilen gefolgt!
 - Kontrolle: Jeder Zustand muss für jedes Symbol einen Übergangspfeil besitzen.
 - Man reduziert es auf ein Problem, das ebenfalls NP-schwer ist (z.B. auf SAT oder 3SAT).
 - Der kleinste DFA für einen NFA mit n Zuständen beträgt 2^n .

Der Startzustand vom M wird mit einer ϵ -Transformation auf den Startzustand von M_1 geführt. Die akzeptierenden Zustände von M_1 werden auf den Startzustand von M_2 geführt. M akzeptiert die Eingabe, wenn si ein einem der akzeptierenden Zustände von M_2 endet. \square

• Problem A lässt sich in polynomiellem Zeit (effizient) auf Problem B reduzieren.

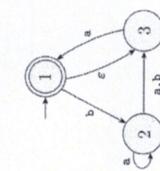
• Falls eine Lösung zu Problem B gefunden wird, kann auch Problem A gelöst werden.

1. Annahme: Die Sprache L sei regulär. Somit existiert ein EA $A = (Q, \{a, b\}, \delta, q_0, F)$ mit $L(A) = L$. Beweis mittels Pumping-Lemma:

- Es existiert eine Konstante $p \in \mathbb{N}$, so dass sich jedes Wort $w \in \{0, 1\}^*$ mit $|w| \geq p$ in drei Teile $w = xyz$ zerlegen lässt, wobei

- für alle $i \geq 0$, $xy^iz \in L$
- $|y| > 0$
- $|xy| \leq p$

Wir wählen das Wort $w = 0^p1^p$.



7.8 Problem ist NP-vollständig

Ein Problem ist NP-vollständig, wenn einer der folgenden Punkte gezeigt werden kann:

- das Problem lässt sich in polynomieller Zeit auf SAT reduzieren.

1. Bestimme Anzahl Zustände von N .
Vorgehen

- y enthält nur Nullen (0): In diesem Fall verfügt das Wort $xyyz$ über mehr Nullen als Einsen und ist somit nicht Teil der Sprache L .
- y enthält nur Einsen (1): Dieser Fall führt zum selben Widerspruch wie im 1. Fall.
- y enthält Nullen (0) und Einsen (1): In diesem Fall verfügt das Wort $xyyz$ zwar über die gleiche Anzahl Nullen wie Einsen. Jedoch ist die Ordnung der Symbole nicht mehr gegeben, da einige Einsen vor Nullen stehen. Somit sind auch diese Wörter nicht Teil von L .

- Also ist die Annahme falsch und die Sprache L ist nicht regulär. \square

7.5.1 Minimal Pumping Length

Bezeichnet den kleinstmöglichen Wert für $p \in \mathbb{N}$, so dass das Wort beim Autopumpen immer noch Teil der Sprache ist (Bedingungen Bedingungen des Pumping Lemmas sind erfüllt).

7.6 Problem liegt in NP

Ein Problem liegt in NP, wenn einer der folgenden Punkte zutrifft:

- Eine (womöglich geratene) Lösung kann ich polynomieller Zeit verifiziert werden.
- ...

Bemerkungen

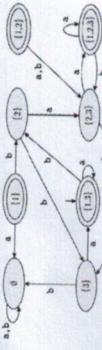
- Zuerst wird immer das Zeichen gelesen und erst dann wird den ϵ -Pfeilen gefolgt!
- Kontrolle: Jeder Zustand muss für jedes Symbol einen Übergangspfeil besitzen.
- Der kleinste DFA für einen NFA mit n Zuständen beträgt 2^n .

7.7 Problem ist NP-schwer

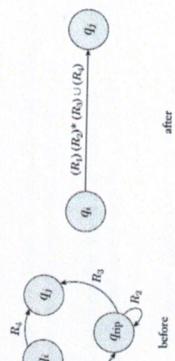
Ein Problem ist NP-schwer, wenn einer der folgenden Punkte zutrifft:

- Man reduziert es auf ein Problem, das ebenfalls NP-schwer ist (z.B. auf SAT oder 3SAT).
- Das Pumping Lemma wird verwendet, um zu beweisen, dass eine Sprache nicht regulär ist.
- Beweis: Für die Sprache $L = \{0^n1^n | n \geq 0\}$ soll bewiesen werden, dass sie nicht regulär ist. Beweis mittels Widerspruch.

Als Resultat erhalten wir:

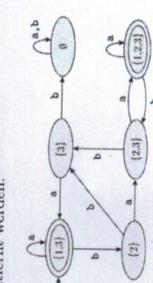
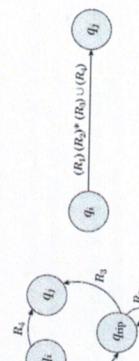


Da keine Pfeile auf die Zustände $\{1\}$ und $\{1, 2\}$ zeigen, kann der DFA weiter vereinfacht werden, indem diese Zustände entfernt werden.



3. Zeichne die neuen Zustände ein.

4. Solange $k > 2$, gehe zurück zu Schritt 1. Wenn $k = 2$ steht der reguläre Ausdruck auf dem Pfeil.

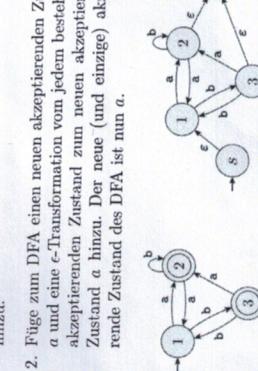


8.2 DFA → Regulärer Ausdruck

Gegeben sei ein DFA. Die Umwandlung läuft in zwei Schritten ab:

DFA → GNFA

1. Füge zum DFA einen neuen Startzustand s und eine ϵ -Transformation zum bestehenden Startzustand hinzu.
2. Füge zum DFA einen neuen akzeptierenden Zustand a und eine ϵ -Transformation vom jedem bestehenden akzeptierenden Zustand zum neuen akzeptierenden Zustand a hinzu. Der neue (und einzige) akzeptierende Zustand des DFA ist nun a .



GNFA → Regulärer Ausdruck

1. Bestimme einen Zustand q_{rip} des GNFA mit $q_{rip} \in Q - \{q_{start}, q_{accept}\}$. Es gilt $Q' = Q - \{q_{rip}\}$.
2. Für jedes $q_i \in Q' - \{q_{accept}\}$ und jedes $q_j \in Q' - \{q_{start}\}$ ermittle die Übergangsfunktionen

$$\delta(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$$

auf. Es gilt:

- $R_1 = \delta(q_i, q_{rip})$
- $R_2 = \delta(q_{rip}, q_{rip})$
- $R_3 = \delta(q_{rip}, q_j)$
- $R_4 = \delta(q_i, q_j)$

Sprachen beschreiben

1. Aufzählen aller Wörter
2. Generieren der Wörter → Regeln → Grammatik
3. Fähigkeit Wörter erkennen → Maschinen, Algo. → EP
4. (mathematische) Konstruktionsvorschrift von Wörtern

Grammatiken

- $G := (N, T, R, S)$
 $N, T : \text{nicht leere Menge, endlich}$
 $N : \text{nicht terminal Symbole (Variablen), } T : \text{Terminal Sym.}$
 $R : \text{endl. Menge von Regeln}$
 $R \subset (N \cup T)^+ \times (N \cup T)^*$
 $S : \text{Startsymbol } S \in N$

CNF-Algorithmen	
Übertragungsalgorithmen für kontextfreie Grammatiken	
$S \rightarrow \dots \rightarrow \alpha \beta \gamma$	$\alpha \beta \gamma \in L(G)$
$S \rightarrow AB \mid CD \mid AD$	$A \rightarrow \alpha$
$C \rightarrow AB \mid SC$	$B \rightarrow \beta$
$D \rightarrow BC \mid AD \mid BS$	$C \rightarrow \gamma$
$A \rightarrow \alpha$	$\alpha \in L(G)$
$B \rightarrow \beta$	$\beta \in L(G)$
$C \rightarrow \gamma$	$\gamma \in L(G)$
$D \rightarrow \delta$	$\delta \in L(G)$

Turing-Tabelle:

q_0	b	a	b	H
q_0	0	q_1	b	R
$...$	-	-	-	-
q_a	-	-	-	-

- ## 8.5 Minimieren eines DFA
- Die Minimierung eines DFA kann mit dem schnellen Markierungsalgorithmus wie folgt geschehen:
1. Neues Startsymbol einfügen.
 2. Eliminieren aller ϵ -Regeln der Form $A \rightarrow \epsilon$.
 3. Eliminieren aller Regeln der Form $A \rightarrow B$.
 4. Verbleibende Regeln durch Einfügen neuer Nonterminals in Chomsky-Normalform bringen.

8.6 Minimierung

- Bsp.
 $L^0 = \{\epsilon\}, \emptyset^0 = \{\epsilon\}, \emptyset \cdot X = \emptyset, L^1 = L \cdot L^0 = L \cdot \{\epsilon\} = L$

Potenzmenge

- $P(M) = 2^M = 2^{|q_1, q_2, q_3|} = \{\emptyset, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_1, q_3\}, \{q_2, q_3\}, \{q_1, q_2, q_3\}\}$
 $|2^M| = 2^{|M|}$ (Bsp.: $= 8$)

Regulärer Ausdruck → NFA

- $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$.

$$R = \emptyset$$

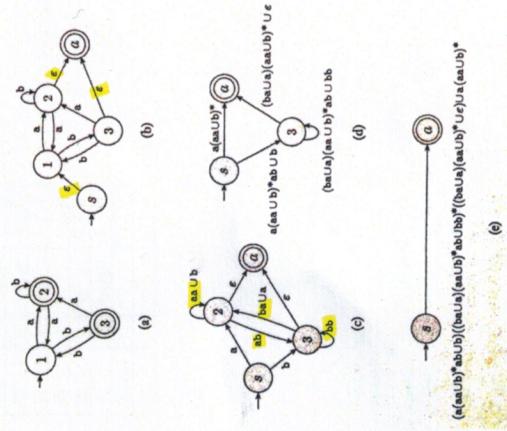
$$(a,b)*;$$

$$ab;$$

$$(a,b)*ab;$$

Teil 2

sich selber), ausgenommen sind Start- und Endzustand. Um einen GNFA zu erzeugen wird in einem ersten Schritt ein neuer Start- bzw. Endzustand, mit ϵ -Übergängen kreiert (siehe Figur unten, Schritt (a) zu (b)). Dann beginnt die Reduktion: Jeder Knoten (außer Start-, End-Knoten) wird entfernt und die verbleibenden Übergänge angepasst (siehe Figur unten).



wir pumpen" (XYZ) gilt $|0| \neq |1|$ oder die Reihenfolge stimmt nicht mehr. Daraus folgt ein Widerspruch mit der Annahme und der Beweis, dass A_1 nicht regulär sein kann.

Beispiel $A_2 = \{0^n 1^n 2^n | n \geq 0\}$

Annahme: die Sprache A_2 ist regulär. Sei p die pumping-length von s der String $0^p 1^p 2^p$. Da $s \in A_2$ und $|s| > p$, kann s aufteilt werden: $s = xyz$. Nach dem Pumping Lemma muss $s = xy^i z$ für jedes $i \geq 0$ in A_2 sein. Dem ist nicht so! Denn ob man $y = 0, y = 1$ oder $y = 2$ wählt, $xy^i z$ wird in jedem Fall nicht mehr in A_2 sein. Auch wenn für y mehr als eine Ziffer gewählt wird, ist s nicht in A_1 , da die Reihenfolge nicht mehr stimmt.

Beispiel $A_3 = \{0^n 1^n 2^n | n \geq 0\}$

Annahme: die Sprache A_3 ist regulär. Sei p die pumping-length von s der String $a^p 2^p$. Da $s \in A_3$ und $|s| > p$, kann s aufteilt werden: $s = xyz$. Die 3. Bedingung des Pumping Lemmas besagt, dass $|xy| \leq p$. Weil $p < 2^p$ kann xy nicht das ganze Wort umfassen (xy ist ja genau 2^p lang). Daraum gilt sicher, dass $|y| < 2^p$. Nach dem Pumping Lemma muss $|xy|z = 2^{p+k}, k \in \mathbb{N}$. Das kann aber nicht stimmen, denn $|xyz| = |xy|z + |y| < 2^p + 2^p = 2^{p+1}$, d.h. $|xyz|$ liegt zwischen 2^p und 2^{p+1} .

Beispiel $A_4 = \{0^m 1^n m | n \neq m\}$

Man weiß: $\neg A_4 \wedge 0^* 1^* = \{0^k 1^k | k \geq 0\}$. Wenn A_4 regulär wäre, so auch $\neg A_4 \wedge 0^* 1^*$. Wir haben bereits gezeigt, dass $\{0^k 1^k | k \geq 0\}$ nicht regulär ist. Daraum kann auch A_4 nicht regulär sein.

2 Kontextfreie Sprachen

Mit Kontext-freien Sprachen werden rekursive Strukturen beschrieben. Kontext-freie Grammatiken (CFG) beschreiben Kontext-freie Sprachen (CFL). Beispielsweise zeigt die Grammatik $G_1: A \rightarrow 0A1, A \rightarrow B, A \rightarrow c$. G_1 generiert unter anderem den String 000111. Die Sequenz von Ersetzungen wird eine Ableitung genannt: $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000B111 \Rightarrow 000111$, oder kurz: $A \Rightarrow^*$ 000111.

Beispiele: Sprache $L(G) \leftrightarrow$ Grammatik G

– $\{w | w$ contains at least three 1s $\}$

$S \rightarrow 0|R1|R1R, R \rightarrow 0|R1|e$

after

before

– $\{w |$ length of w is odd and its middle symbol is a 0

$S \rightarrow 0|0S0|0S1|1S0|1S1$

– $\{w \# x | w^R$ is a substring of x for $w, x \in 0, 1^*$

$S \rightarrow TX, T \rightarrow 0T|0|T1\#X, X \rightarrow 0X|1|X|e$

2.1 Parser

Ein Parser ist ein Programm, das testet, ob ein Wort w zur einer CFG gehört. Ein typisches Beispiel ist das Testen korrekter Klammererstellung.

Beispiel $A_1 = \{0^n 1^n | n \geq 0\}$

Annahme: die Sprache A_1 ist regulär. Dann ist $s = 0^p 1^p \in L_R$ und $|s| \geq p \leq |XYZ|$. Zu jeder regulären Sprache L_R gibt es eine Zahl p pumping length, so dass $|L_R| \geq p$. Außerdem muss für jede reguläre Sprache folgendes gelten:

- $XY^i Z \in L_R, i \geq 0$,
- $|Y| \geq 1$
- $|XY| \leq p \leq |XYZ|$

CKY-Algorithmus Der Cocke-Younger-Kasami Algorithmus ist ein Parse-Algorithmus für Kontext-freie Sprachen. Damit kann entschieden werden, ob das Wort w

zur Sprache $L(G)$ gehört. Der CYK-Algorithmus verlangt als Input eine Kontext-freie Sprache in Chomsky-Normalform.

Beispiel: Gegeben sei die folgende Grammatik $G := (N, T, R, S)$ mit $N = \{S, A, B\}, T = \{a, b\}$ und den folgenden Regeln (die gleich hier in die Chomsky-Normalform umgewandelt werden):

$$\begin{array}{l} S \rightarrow AA|Y_BD|a \\ S \rightarrow AA|aBa|b \\ A \rightarrow aA|a|ab \\ B \rightarrow aBaa|b \\ Y_a \rightarrow a \end{array}$$

$$S \rightarrow AA|Y_A|Y_a|b$$

$$\begin{array}{l} A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

$$\begin{array}{l} S \rightarrow AA|Y_A|Y_a|C|b \\ A \rightarrow Y_a A|AY_a|b \\ B \rightarrow Y_a BD|b \\ C \rightarrow BD \\ D \rightarrow Y_a Y_a \\ Y_a \rightarrow a \end{array}$$

– schiebe jedes 0 auf den Stack (push!)

– für jede 1, die danach kommt: nehme eine 0 vom Stack (pop!)

– wenn nach den 0-en nur noch 1-en kommen und gleichzeitig das Wortende + Stack leer, dann akzeptiere das Wort. Verweigere es andernfalls.

Beispiel PDA für die Sprache $C = \{ww^R | w \in \{0, 1\}^*\}$

– schiebe alle Zeichen auf den Stack

– nach jedem Symbol nehme ich nicht-deterministisch an, dass die Menge erreicht ist (paralleler Prozess statt "then")

– teste, ob w auf Stack und w^R vom Input-Band identisch

– falls Wortende + Stack leer + auch sonst alles ok, dann akzeptiere Sprache!

3 Turing Maschinen

Eine Turing Maschine besteht aus einem endlichen Automaten mit einem Lesekopf, der auf ein ∞ -grosses Band zugriff hat. Bei jedem Rechenzyklus (bei jedem Übergang des endlichen Automaten) kann sich dieser Lesekopf nach links oder rechts bewegen, das aktuelle Zeichen lesen und es allenfalls überschreiben. Neu gibt es die beiden Zustände q_{accept} und q_{reject} . Während man beim PDA oder bei einem endlichen Automaten sicher sein konnte, dass diese nach endliche vielen Schritten halten, kann es bei der Turing Maschine sein, dass sie nie hält! Ein Wort gilt als akzeptierend, wenn die TM im Zustand q_{accept} hält.

Was ist $L(G)$? Für diese Sprache gibt es keine geschlossene, einheitliche Beschreibung. Man kann die Sprachbeschreibung jedoch genäss der ersten Zeile $S \rightarrow AA|B$ (2x A oder 1x B) aufteilen. Es gilt $L = (L_1 \cup L_2)$ mit $L_1 = \{a^k b^k | k \in \mathbb{N}\}$ und $L_2 = \{a^k b^{\neq k} | k \in \mathbb{N}\}$.

Gehört das Wort $aabbba$ zur Sprache $L(G)$? Wir betrachten die Tabelle unten:

Zeile 1, Spalte 1: Welches non-terminal Symbol erzeugt Y_a ? Zeile 1, Spalte 2: Welches non-terminal Symbol erzeugt bY_a ? Zeile 2, Spalte 1: Welches Symbol erzeugt $Y_a Y_a$? Zeile 2, Spalte 2: Welches Symbol erzeugt Y_a ? Zeile 3, Spalte 1: Welches Symbol erzeugt Y_A ? Achtung: Ab Zeile 3 ändert das Verfahren! Betrachte nun die Spalte 1, so werden zuerst die beiden Zellen $[1, j]$ (hier Y_a) und $[2, j+1]$ (hier A) augeschaut. Unabhängig davon, ob eine entsprechende Produktionseule (hier $a \rightarrow Y_a$) gefunden wird, müssen weitere Überprüfungen vorgenommen und alle entsprechenden non-terminalen Symbole notiert werden.

Die nächsten beiden Zellen sind $[2, j]$ (hier D) und $[1, j+2]$ (hier B,A,S) angesehen.

Zeil 1, Spalte 2: man betrachtet der Reihe nach: $[1, j] \wedge [i-1, j+1], [2, j] \wedge [i-2, j+2], \dots, [1, j+2] \wedge [i, j+1]$. Ein Wort gilt als nicht akzeptierend, wenn es im Zustand q_{reject} oder nie hält.

Definition: A Turing machine is a 7-tupel, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and $Q \subseteq \Gamma$ is the set of states.

1. Q is the tape alphabet not containing the blank symbol b .

2. Σ is the tape alphabet, where $b \in \Gamma$ and $\Sigma \subseteq \Gamma$.

3. $\delta : Q \times \Gamma \rightarrow \Gamma \times \{L, R\}$ is the transition function,

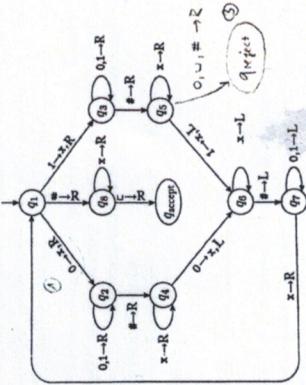
4. $q_0 \in Q$ is the start state,

5. $q_{accept} \in Q$ is the accept state, and

6. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

Beispiel: $M_2 = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $Q = \{q_0, q_1, q_2, q_{reject}\}, \Sigma = \{0, 1\}, \Gamma = \{0, 1, b\}, F = \{q_a\}$ und $\delta : Q \times \Gamma \rightarrow Q \times \{L, H, R\}$.

Die TM beschreibt die Sprache $L(M_2) = \{0^n10^n | k, m, n \geq 0\}$ und ist durch die folgende Turingtabelle gegeben.



Beispiel: $\{w|w \text{ contains an equal number of 0s and 1s}\}$

Wir geben die Beschreibung auf Implementationslevel an:

- i. Band scannen, die erste 0 markieren und den Lesekopf zurück an den Anfang setzen. Ist keine unmarkierte 0 zu finden, geht zu Schritt iv.
- ii. Band scannen und die erste 1 markieren. Ist keine unmarkierte 1 zu finden, reject.
- iii. Lesekopf an den Anfang zurücksetzen und zum Schrift i gelien.
- iv. Lesekopf an den Anfang zurücksetzen. Band scannen, um zu sehen, ob es noch unmarkierte 1s gibt. Falls es keine mehr gibt: accept, sonst: reject.

4 Entscheidbarkeit

Semi-Entscheidbar: Eine Sprache heisst semi-entscheidbar, oder rekursiv aufzählbar, falls es eine TM gibt, die diese Sprache akzeptiert, aber bei einem Wort, das nicht zur Sprache gehört eventuell nie hält.

Entscheidbar: Eine Sprache heisst entscheidbar, oder rekursiv, falls es eine TM gibt, die diese Sprache akzeptiert/ablehnt und immer hält.

Varianten von Turing Maschinen Es gibt viele Varianten von Turing Maschinen, die aber trotz ihrer Unterschiede alle gleich mächtig sind. Beispielsweise gibt es TM mit ≥ 2 Bändern. Sie sind oft bequemer zu definieren. Zwar kann jede TM-Variante mit jeder anderen TM-Variante simuliert werden, die Rechenzeit kann dabei aber erheblich unterschiedlich sein!

4.1 These von Church

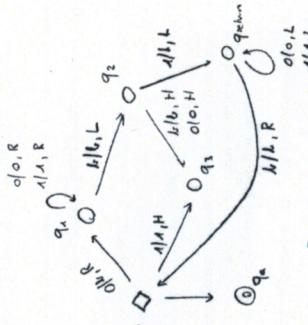
Die These von Church besagt, dass alle Computer dasselbe können. Oder anders formuliert: Lässt sich eine Problemklasse mit einem dieser Computer lösen, dann auch mit allen anderen.

Beispiel: Gegeben sei die deterministische TM $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $Q = \{q_0, q_1, q_2, q_{reject}, q_{accept}\}$ und $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, H, R\}$ gegeben durch die Turingtabelle unten. M_1 beschreibt die Sprache $\{0^k1^k | k \in \mathbb{N}\}$.

Beispiel: Gegeben sei die deterministische TM $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $Q = \{q_0, q_1, q_2, q_{reject}, q_{accept}\}$ und $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, H, R\}$ gegeben durch die Turingtabelle unten. M_1 beschreibt die Sprache $\{0^k1^k | k \in \mathbb{N}\}$.

Für das Wort $w_1 = 01$ werden $t_M(w_1) = 6$ Schritte benötigt und $s_M(w_1) = 3 \geq |w_1|$ Zeilen des Input-Bandes angesprochen.

q_0	b	q_a	b	H
q_0	0	q_0	b	R
q_0	1	q_1	1	R
q_1	b	q_{reject}	b	H
q_1	1	q_2	1	R
q_2	b	q_a	b	H
q_2	0	q_2	b	R
q_2	1	q_{reject}	1	H
q_{reject}	-	-	-	-
q_a	-	-	-	-



Beispiele Komplexitätsschranken Sei $\Sigma := \{0, 1\}$. Die folgenden Sprachen über Σ können wie folgt klassifiziert werden:

1. Berechnungsmodell (TM 1-Band, TM k-Band, RAM) genau 1x übers ganze Band fahren muss.
2. Berechnungsmodus (deterministisch oder nicht-deterministisch)
3. betrachtete Ressource (Zeit und Speicherplatz)
4. Berechnungsschranke ($O(n^k), O(\log(n))$)

Beispiel: $L = \{0^k1^k | k \geq 0\}$ gehört zur Klasse der in quadratischer Zeit (3) mit einer deterministischen (2) 1-Band TM (1) entscheidbaren Sprachen.

Anzahl Knoten: $\leq 2k^{t(n)}$

P wird definiert als die Klasse aller Sprachen, die in polynomiellem Laufzeit auf einer deterministischen 1-Band TM entscheidbar sind: $P = \bigcup_k NTIME(n^k)$.

Beispiele: Alle regulären Sprachen, alle Kontext-freien Sprachen, Berechnung des größten gemeinsamen Teilers, lineare Programmierung und die Verifikation, ob eine Nummer Prim ist.

Bemerkung: Wenn $L_1, L_2 \in \mathbb{P}$, so sind auch $L_1 \vee L_2, L_1 \circ L_2, \sum^* - L_1 \in \mathbb{P}$.

5.5 Die Klasse NP

Für viele Probleme könnte man bis anhin noch kein Algorithmus mit einer polynomiellem Laufzeit finden. Hat man für ein Problem eine Lösung gefunden, lässt sich diese oft schnell (in polynomieller Zeit) verifizieren (z.B. Hamiltonpfad)! Zu zeigen, dass ein Problem keine Lösung hat ist hingegen oft schwierig (z.B. Hamiltonpfad).

Theorem 1. Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multtape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Durch den nicht-Determinismus erhält man anstelle eines Berechnungspfades eine Berechnungsbaum. Theorem 7.11 aus Sipser:

Theorem 2. Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

5.3 Formatierung des Inputs

Da die Komplexität in Abhängigkeit der Länge des Inputs genauso wird, ist sie auch abhängig von deren Codierung. Oder mit anderen Worten: Die Laufzeit von Algorithmen ist meistens proportional zur Eingabellänge und können daher nur sinvoll verglichen werden, wenn deren Eingabe gleich codiert ist.

Eingaben können beispielsweise unär oder binär codiert werden, z.B. $5 = 11111 = 101_2$. Zur Darstellung der Zahl n in Basis b (binär: $b = 2$) sind $\lceil \log_b(n) \rceil + 1$ Zeichen notwendig, während in unärer Codierung n Zeichen notwendig sind.

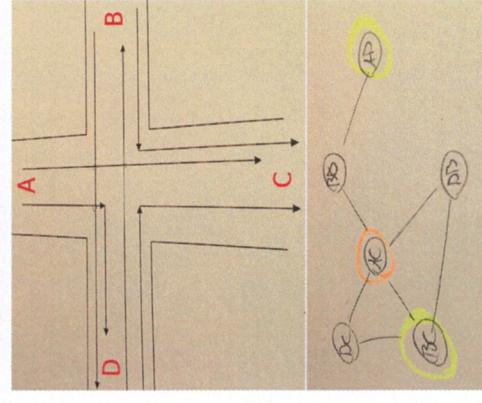
Die Verwendung binärer Codierung haben wir im Untericht als natürliches Kostenmass bezeichnet.

5.4 Die Klasse P

Polynomielles Unterschiede in der Laufzeit werden als klein angesehen, während exponentielle Unterschiede als gross betrachtet werden. Berechnungsmodelle, deren Laufzeiten sich höchstens polynomiel unterscheiden, werden in einer Klasse P vereinigt. Sie werden auch als polynomial äquivalent bezeichnet und umfassen alle deterministischen Berechnungsmodelle (vgl. Theorem 7.8 aus Sipser).

Beispielsweise konnten wir im Unterricht das Ampelproblem auf das Graph-Coloring Problem reduziert, indem wir jede Fahrt Richtungen als Karten und jede mögliche Kolisation zwischen zwei Fahrtrichtungen als Karte modelliert haben. Das Problem konnte schliesslich mit 3 Farben bzw. 3 Ampeln gelöst werden.

Beispiel: Wenn $L_1, L_2 \in \mathbb{P}$, so sind auch $L_1 \vee L_2, L_1 \circ L_2, \sum^* - L_1 \in \mathbb{P}$.



6.3 NP-complete

Es gibt eine Reihe von Problemen in NP mit einer wichtigen Eigenschaft: Wenn für eines dieser Probleme ein polynomialer Algorithmus gefunden werden kann, so wären alle Probleme in NP polynomial lösbar. Diese speziellen NP-Probleme werden als NP-complete oder NPC bezeichnet.

Zusätzlich wird zwischen NP-hard und NP-complete (NPC) unterschieden. Wir betrachten eine Sprache L :
 L heißt NP-hard $\Leftrightarrow \forall L' \in NP : L' \leq_p L$

L heißt NP-complete $\Leftrightarrow L \in NP\text{-hard} \wedge L \in NP$

6.4 SAT

Problembeschreibung: Das 'satisfiability problem' SAT ist das erste Problem, von dem bewiesen wurde, dass es in NP-complete liegt. Dabei geht es um die Belegung von Booleschen Ausdrücken. Notation:

- $p_1, \neg p_2$ Wahrheitswerte (TRUE bzw. FALSE)
- l_1, l_2, \dots, l_n Literale, jeder Literal ist ein Wahrheitswert
- $k_1 = (l_1 \vee l_2 \vee \dots \vee l_n)$ Klauseln, bestehend aus Litralen
- $k_1 \wedge k_2 \wedge \dots \wedge k_n$ Konjunktive Normalform KNF.

6.1 Turing-Äquivalenz

Die Beziehung \leq_T ist reflexiv und translativ. Gilt $A \leq_T B$ und $B \leq_T A$, so sind A und B Turing-äquivalent. Man schreibt $A =_T B$.

Es gilt immer $L \leq_T \bar{L}$: Das Komplement eines Problems ist immer gleich schwierig zu lösen wie das Problem an sich. Dies ist sehr schön am Beispiel der 3 Probleme: Clinique, Independent Set und Vertex Cover zu sehen.

Beispiel: Gegeben sei die aussagenlogische Formel $\neg p_1 \vee x_2 \vee v \wedge p_1 \wedge \neg x_3 \vee \neg x_4 \vee \neg x_5 \wedge \neg x_7$.
 $\phi_{SAR} = (\neg x_1 \vee x_2 \vee v \wedge p_1 \wedge \neg x_3 \vee \neg x_4 \vee \neg x_5 \wedge \neg x_7) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_5 \vee \neg x_7)$.
 ϕ_{SAR} wird für die folgende Belegung wahr: $w(x_1) = 1, w(x_2) = 0, w(x_3) = 1, w(x_4) = 0, w(x_5) = 0, w(x_7) = 0$.

6.5 SAT \leq_p 3SAT

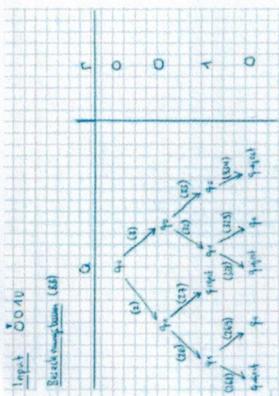
Problembeschreibung: Das 3SAT Problem ist eine Probleminstanz von SAT mit höchstens 3 Literalen pro Klausel.

Beispiel Probleminstanz 3SAT: $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_4) \wedge (x_3 \vee \neg x_4 \vee x_5)$

Vorgehen

Definition 3. A function $f : \sum^* \rightarrow \sum^*$ is a polynomial time computation function if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w .

- Klauseln mit genau 3 Literalen sind bereits ok.
- Klauseln mit < 3 Literalen werden mit einem der Literale in der Klausel aufgefüllt.



5.2 Vergleich von TM-Varianten

Das folgende Theorem aus Sipser (7.8) bestätigt die These von Church-Turing.

Theorem 1. Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multtape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Durch den nicht-Determinismus erhält man anstelle eines Berechnungspfades eine Berechnungsbaum. Theorem 7.11 aus Sipser:

Theorem 2. Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

6 Reduktion

Motivation: Man sucht ein allgemeines Konzept, um ausdrücklich, dass ein Problem nicht viel schwieriger als ein anderes ist.

Bemerkung: Wenn $L_1, L_2 \in \mathbb{P}$, so sind auch $L_1 \vee L_2, L_1 \circ L_2 \in \mathbb{P}$. Ob die Sprache $\sum^* - L_1$ in NP liegt, ist im Allgemeinen unbekannt.

6 Reduktion

Motivation: Man sucht ein allgemeines Konzept, um ausdrücklich, dass ein Problem nicht viel schwieriger als ein anderes ist.

Definition Ein Problem A heisst Turing-reduzierbar auf ein Problem B, $A \leq_T B$, wenn es einen Algorithmus für A gibt, der als Unterprogramm einen Algorithmus für B aufruft, so dass $T_A(n) \leq p(n) * T_B(r(n))$. Dabei sind

- p(n) die Rechenzeit ohne die Aufrufe des Unterprogramms,
- q(n) die Anzahl der Aufrufe,
- r(n) die Eingabellänge für jeden Aufruf.

Wenn nun $T_A(n)$ polynomial beschränkt ist, so auch $T_B(n)$.
Bemerkung: In diesem Kontext wird das Unterprogramm traditionell Orakel genannt, da wir oft Probleme mit einem Algorithmus umsetzen wollen, ohne die entsprechenden Algorithmen zu kennen.

6.2 Polynomielle (Karp-) Reduktion

Werden nur Reduktionen betrachtet, die effizient, bzw. in polynomieller Zeit berechenbar sind, so ist die Reduktion gemäss Sipser wie Folgt definiert (Definition 7.28):

Definition 3. A function $f : \sum^* \rightarrow \sum^*$ is a polynomial time computation function if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w .

- Klauseln mit genau 3 Literalen sind bereits ok.
- Klauseln mit < 3 Literalen werden mit einem der Literale in der Klausel aufgefüllt.

- Klauseln mit > 3 Literalen werden nach dem folgenden Schema in 3-er Gruppen aufgeteilt und mit neuen Literalen $\neg z_1, \neg z_2$ verbunden:

$$(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \dots \wedge (\overline{z_{k-3}} \vee a_{k-1} \vee a_k)$$

6.6 3SAT \leq_p SUBSET-SUM

Problembeschreibung Das Subset-Sum Problem ist ein Spezialfall des Rucksackproblems: Gegeben sind $n+1$ positive Zahlen a_1, a_2, \dots, a_n, t . Gesucht sind $x_1, \dots, x_n \in \{0, 1\}$, so dass $t = \sum_{i=1}^n x_i a_i$ gilt.

Beispiel: Sei $a_1 = 3, a_2 = 6, a_3 = 11, a_4 = 9, t = 20$. Dann ist $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 0$ eine Lösung. $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$ eine andere. Manchmal gibt es gar keine Lösung.

Vorgehen: Für eine 3SAT-Formel mit Variablen x_1, \dots, x_t und Klauseln $c_1 \dots c_k$ wird folgendes Schema aufgestellt:

6.7 3SAT \leq_p CLIQUE

Problembeschreibung Eine Clique bezeichnet eine Teilmenge von Knoten in einem ungerichteten Graphen, bei der jedes Knotenpaar durch eine Kante verbunden ist.

Eine 3SAT-Formel ϕ mit k Klauseln und max. drei Literalen pro Klausel kann in polynomialer Zeit in einem Graphen mit einer k -Clique umgewandelt werden. Der Graph besitzt genau dann eine k -Clique, wenn die Formel erfüllbar ist und umgekehrt.

Vorgehen Für die Reduktion auf das Cliquen Problem wird ϕ in einen Graphen umgewandelt werden:

1. Der Graph ist in $|k|$ (Anzahl Klauseln) Gruppen aufgeteilt. In jeder dieser Gruppe hat es 3 Knoten. Jeder Knoten korrespondiert zu einem Literal. Zwischen allen Knoten gibt es eine Verbindung, ausser sie befinden sich in der selben Gruppe, oder sie bilden einen Gegensatz (z.B. x_1 und $\neg x_1$).
2. Nun sucht man im Graphen nach k -Cliques (wobei jede Gruppe genau mit 1 Knoten involviert ist).
3. Die gewählten Knoten entsprechen genau einer gültigen Belegung der Literale.

1. Füllte Grundschemata gemäß Bild aus (oben rechts lassen).
2. Setze in $(y_{1..t}, c_{1..k}) = 1$ (rechts oben), wenn Variable $x_{1..t}$ in der entsprechenden Klausel positiv vorkommt.
3. Setze das Feld $(z_{1..t}, c_{1..k}) = 1$, wenn Variable $x_{1..t}$ in der entsprechenden Klausel positiv vorkommt.
4. t ist das Soll der Summe pro Spalte. Wähle die Zeilen y_1, z_1, \dots, y_k so aus (je entweder y_i oder z_i), dass die Summe pro Spalte t ergibt. Beachte: die Werte $g_1, h_1, \dots, g_k, h_k$ müssen mitgezählt werden.
5. Alle markierten Zeilen entsprechen nun dem Subset des SUBSET-SUM-Problems mit Summe t (aus Schema). Alle Zahlen sind decimal zu interpretieren.

Beispiel: Gegeben sei die aussagenlogische Formel $\phi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$.

Vorgehen

1. Zeichne für jede Klausel ein Knotentripel, welche nach den Literalen in der Klausel benannt sind. Die Knoten des Tripels sind alle untereinander verbunden.

2. Zeichne für jeden Literal zwei zusätzliche Knoten x_i und $\neg x_i$.
3. Verbinde nun alle Knoten der Tripel mit identischen Knoten aus Schritt 2.
4. $k = m + 2t$, wobei m die Anzahl Variablen und t die Anzahl der Klauseln in ϕ ist.

Algorithmus: MaxMatching(E)

```

1:  $C \leftarrow \emptyset$ 
2: while  $E \neq \emptyset$  do
3:   pick at random  $e = \{v, w\}$ 
4:   pick both nodes  $C \leftarrow C \wedge \{v, w\}$ 
5:    $E \leftarrow E \setminus \{e \in E | v \in e \wedge w \in e\}$ 
6: end while
7: return C

```

8 Anhang

8.1 Chomsky Hierarchy

Die Chomsky Hierarchy beschreibt die Mächtigkeit formaler Sprachen und besteht aus 4 Ebenen. Typo-1 wurde im Unterricht nicht behandelt (darum die ...).

Sprache	Automat	Produktionsregel
Rek. enumerierbar	Turing Maschine	keine Restriktionen
Kontext-sensitiv	Kellerautomaten	$A \rightarrow \gamma$
Regulär	State Machines	$A \rightarrow \beta$ und $A \rightarrow \beta$
Kontext-frei

8.2 Chomsky Normalform

Eine Grammatik ist in Chomsky-Normalform, wenn all ihre Regeln von der folgenden Form sind:

```

A → BC
A → b

```

Wie bringt man eine Grammatik in Chomsky Normalform? Es gibt eine Reihe von Tricks, die angewendet werden können. Abschliessend werden für die vorliegenden Regeln neue Variablen eingeführt, bis die Chomsky Normalform erreicht ist. Generell sind die folgenden Schritte zu befolgen:

1. Neue Startvariable hinzufügen
 2. ϵ -Regeln entfernen
 3. 'unit rules' von der Art $A \rightarrow B$ entfernen
- 7 Approximation**
- Oft reicht es in der Praxis, für Optimierungsprobleme nur annähernd die beste Lösung zu finden. Indem man für einen Algorithmus bestimmt, um wie viel Mal schlechter die Lösung im schlechtesten Fall ist (vergleichen mit dem Optimum), lassen sich die Algorithmen vergleichen. Ein k Mal schlechterer Algorithmus wird k-optimal genannt. Im

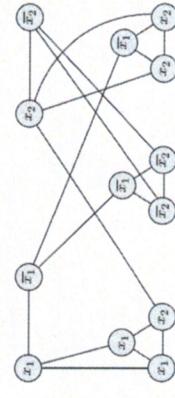
Unterricht haben wir 3 Varianten von Algorithmen ange schaut, die das Vertex-Cover Problem approximieren. Der beste von ihnen ist 2-optimal:

Algorithmus: MaxMatching(E)

```

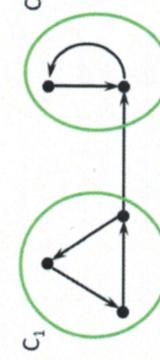
1:  $C \leftarrow \emptyset$ 
2: while  $E \neq \emptyset$  do
3:   pick at random  $e = \{v, w\}$ 
4:   pick both nodes  $C \leftarrow C \wedge \{v, w\}$ 
5:    $E \leftarrow E \setminus \{e \in E | v \in e \wedge w \in e\}$ 
6: end while
7: return C

```



6.9 2SAT \leq_p Graph mit SZK

Problembeschreibung Eine 2SAT-Formel ϕ kann auf einen Graphen mit starken Zusammenhangskomponenten (SZK) reduziert werden. Eine starke Zusammenhangskomponente ist ein Teil eines Graphen, worin von jedem Knoten aus jeder andere Knoten erreicht werden kann. In der folgenden Graphik bilden C_1 und C_2 je eine SZK. Sie dürfen miteinander verbunden sein, aber nur in eine Richtung. Wären C_1, C_2 in beide Richtungen verbanden, würden sie zusammen eine gemeinsame Komponente bilden.



Vorgehen

1. Literale der 2SAT-Formel aufzufrieren (positive links untereinander, negative rechts untereinander).
2. Für jede Klausel werden nun 2 Kanten eingelegt:
 1. Literal negiert nach 2. Literal
 2. Literal negiert nach 1. Literal

6.8 3SAT \leq_p VERTEX-COVER

Problembeschreibung Eine 3SAT-Formel ϕ kann auf einen Graphen G und eine Zahl k in polynomieller Zeit reduziert werden. Die Formel ist nur dann erfüllbar, wenn der Graph G ein Knotenüberdeckung mit max. k Knoten hat (die k Knoten müssen alle Kanten des Graphen abdecken).

Vorgehen

1. Zeichne für jede Klausel ein Knotentripel, welche nach den Literalen in der Klausel benannt sind. Die Knoten des Tripels sind alle untereinander verbunden.
2. Zeichne für jede aussagenlogische Formel $\phi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$.

	1	2	3	4	c_1	c_2	c_3
y_1	1	0	0	0	0	1	0
z_1	1	0	0	0	0	0	1
y_2	1	0	0	0	0	1	0
z_2	0	1	0	0	0	0	1
y_3	0	0	1	0	0	0	1
z_3	0	0	1	0	0	0	1
y_4	0	0	0	1	0	0	1
z_4	0	0	0	1	0	0	1
g_1	0	0	0	0	0	0	0
h_1	0	0	0	0	0	0	0
g_2	0	0	0	0	0	0	0
h_2	0	0	0	0	0	0	0
g_3	0	0	0	0	0	0	0
h_3	0	0	0	0	0	0	0
t	1	1	1	1	1	3	3

	1	2	3	4	c_1	c_2	c_3
y_1	1	0	0	0	0	1	0
z_1	1	0	0	0	0	0	1
y_2	1	0	0	0	0	1	0
z_2	0	1	0	0	0	0	1
y_3	0	0	1	0	0	0	1
z_3	0	0	1	0	0	0	1
y_4	0	0	0	1	0	0	1
z_4	0	0	0	1	0	0	1
g_1	1	0	0	0	0	0	0
h_1	0	1	0	0	0	0	0
g_2	0	0	1	0	0	0	0
h_2	0	0	0	1	0	0	0
g_3	0	0	0	0	1	0	0
h_3	0	0	0	0	0	1	0
t	1	1	1	1	1	3	3

Erst Sprache mit $L \neq L(\Sigma)$ nicht!

wenn die Sprache $L \in \Sigma$ regulär ist, dann ist auch ihre komplement regulär
Mit dem Pumping-Lemma lässt sich nachweisen, dass eine Sprache L regulär ist
wenn P die folgende reguläre Sprache ist dann ist $L \supseteq P$ sicher ebenfalls
dafür kann man alle vor Sprache setzen \square

H

Ein Entscheidbarkeitstest für keine Erweiterung in der Endlichkeit sehr
Ein Ergebnis der Automatentheorie ist das es jeder DFA D ein endlich langer
binärer Zeichenketten existiert und ein Minimalkriterium existiert
1) REG \rightarrow NFA 2) NFA \rightarrow DFA 3) DFA minimalkriterium
 $S \rightarrow bAc$ konse. | $SAt \rightarrow aCatb$ konse. | $bAb \rightarrow bAcc$ konse. | $bAbc \rightarrow bAcc$ konse. | $c \rightarrow bAc$ konse.

Th 2.1:

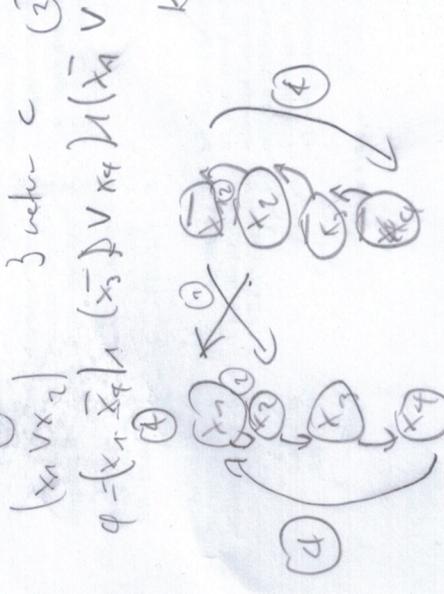
Ein Entscheidbarkeitstest für keine Erweiterung in der Endlichkeit sehr
Ein Ergebnis der Automatentheorie ist das es jeder DFA D ein endlich langer
binärer Zeichenketten existiert und ein Minimalkriterium existiert
1) REG \rightarrow NFA 2) NFA \rightarrow DFA 3) DFA minimalkriterium
 $S \rightarrow bAc$ konse. | $SAt \rightarrow aCatb$ konse. | $bAb \rightarrow bAcc$ konse. | $bAbc \rightarrow bAcc$ konse. | $c \rightarrow bAc$ konse.

Alg. Matchen h_1
Algo Matchen h_1
1. pick $e \in E$
2. pick $x_1 \in \Sigma$
3. pick $x_2 \in \Sigma$
4. pick $x_3 \in \Sigma$
5. pick $x_4 \in \Sigma$
6. pick $x_5 \in \Sigma$

Rechts
 $L_1 = \{0^k 1110^l \mid k \in \mathbb{N}\}$
 $L_2 = \{111, 0110\}$
a) Def von $L_1 \leq_p L_2$
b) Karp und

$$S2K \quad q = (x_1 \bar{x}_4) \wedge (\bar{x}_3 \bar{x}_4 \vee x_1 x_2) \wedge (x_3 \vee \bar{x}_2)$$

k Klausur



6) $f: \Sigma^* \rightarrow \Sigma^*$

$w \mapsto f(w) := \{111, 0110\}$
samt $w \in L_1$ ist in polynomialer Zeit T.
berechenbar