Komplexität des Äquivalenzproblems für reguläre Ausdrücke

Philipp Zschoche
philipp.zschoche@uni-oldenburg.de

Proseminar ÄRA: Äquivalenz regulärer Ausdrücke
Sommersemester 2014
Modulverantwortlich: Dr. Hans Fleischhack, Prof. Dr. Eike Best

Carl von Ossietzky Universität Oldenburg
Department für Informatik
Ammerländer Heerstr. 114-118
26129 Oldenburg

Abstract: Diese Arbeit beschäftigt sich mit der Komplexität des Äquivalenzproblems für reguläre Ausdrücke mit den Operationen $\{\cup, ^*, \cdot\}$. In Kapitel 3 wird gezeigt, dass das Äquivalenzproblem für reguläre Sprachen *PSPACE-vollständig* ist.

1 Einführung

Reguläre Ausdrücke, definiert in Abschnitt 2.1, sind syntaktische Objekte für die Beschreibung einer regulären Sprache. Das Äquivalenzproblem für reguläre Ausdrücke stellt die Frage, ob zwei reguläre Ausdrücke die gleiche Sprache beschreiben. Jede reguläre Sprache kann von einem deterministischen endlichen Automat akzeptiert werden [Bes13, Satz 4.2.3]. Das Äquivalenzproblem für reguläre Ausdrücke wird auf das Äquivalenzproblem für deterministische endliche Automaten zurück geführt, indem für beliebige reguläre Ausdrücke re_1, re_2 deterministische endliche Automanten A_1, A_2 mit $L(A_1) = L(re_1)$ und $L(A_2) = L(re_2)$ mittels *Glushkov-Konstruktion* [CF01] erstellt werden. Reguläre Sprachen sind unter anderem abgeschlossen unter *Vereinigung*, *Schnitt* sowie *Komplement* und es gibt Algorithmen um diese Operationen durchzuführen [Bes13, Satz 4.3.1]. Daher ist die Sprache $L(A)$ regulär:

$$L(A) := (L(A_1) \cap \overline{L(A_2)}) \cup (L(A_2) \cap \overline{L(A_1)})$$

Offenbar ist $L(A) = \emptyset$ genau dann, wenn $L(A_1) = L(A_2)$ gilt, d.h., wenn re_1 und re_2 äquivalent sind. Aus der Entscheidbarkeit des Leerheitsproblems für reguläre Sprachen folgt direkt:

Satz 1 (Entscheidbarkeit)**.** *Das Äquivalenzproblem für reguläre Ausdrücke ist entscheidbar.*

Nachdem die Frage der Entscheidbarkeit des Äquivalenzproblems für reguläre Ausdrücke positiv beantwortet wurde, wird die Komplexität diskutiert. Abschnitt 2 führt ausschließlich Definitionen ein, die in Abschnitt 3 benötigt werden. In Abschnitt 3 wird gezeigt, dass das Äquivalenzproblem für reguläre Ausdrücke mit $\{\cup, ^*, \cdot\}$ *PSPACE-vollständig* ist.

2 Definitionen

In diesem Kapitel werden Definitionen eingeführt, die für die Betrachtung der Komplexität des Äquivalenzproblems für reguläre Ausdrücke notwendig sind. Falls nicht anders angegeben, handelt es sich bei Σ um ein endliches Alphabet und bei ε um das leere Wort.

2.1 Reguläre Ausdrücke

Reguläre Ausdrücke sind syntaktische Objekte, die eine reguläre Sprache (Chomsky-3-Sprache) beschreiben [Bes13, Kapitel 4.4].

Definition 1 (Syntax)**.** *Sei Σ ein endliches Alphabet. Induktive Definition:*

- *Beginn: $\emptyset$ und ε sind reguläre Ausdrücke über Σ. Für jedes $\alpha \in \Sigma$ ist α ein regulärer Ausdruck über Σ.*
- *Schritt: Wenn re_1, re_2 reguläre Ausdrücke über Σ sind, so auch $(re_1 \cup re_2), (re_1 \cdot re_2)$ und re_1^*.*
- *Abschluss: Sonst ist nichts ein regulärer Ausdruck über Σ.*

Definition 2 (Semantik)**.** *Seien re_1, re_2 reguläre Ausdrücke über Σ. Die Sprache $L(re)$ eines regulären Ausdrucks re ist foldendermaßen definiert:*

$$\begin{aligned} L(\emptyset) &= \emptyset \\ L(\varepsilon) &= \{\varepsilon\} \\ L(\alpha) &= \{\alpha\} \text{ wobei } \alpha \in \Sigma \\ L(re_1 \cup re_2) &= L(re_1) \cup L(re_2) \\ L(re_1 \cdot re_2) &= L(re_1) \cdot L(re_2) \\ L(re_1^*) &= (L(re_1))^* \end{aligned}$$

2.2 Turingmaschine

Bei einer Turiungmaschine handelt es sich um einen Leseschreibkopf auf einem unendlichen Band aus Speicherzellen, in denen Elemente aus Γ gespeichert werden können. Eine Turingmaschine befindet sich immer genau in einem Zustand. In jedem Schritt wird δ mit dem Inhalt der aktuellen Speicherzelle und dem aktuellen Zustand ausgeführt.
Die folgende Definition einer Turingmaschine (TM) entspricht sinngemäß der Definition in [Bes13, Kapitel 6.1] und wird hier nur auszugsweise aufgeführt.

Definition 3. *Eine Turingmaschine ist definiert als $M = (I, \Gamma, Q, \delta, q_0, q_a)$ wobei gilt:*

- Γ : *Das Arbeitsbandalphabet (endlich), wobei $\sqcup \in \Gamma$ das Leerheitssymbol ist*
- $\Gamma \supseteq I$: *Das Eingabealphabet*
- Q : *Die Menge der Zustände (endlich)*
- $q_0 \in Q$: *Der Startzustand*
- $q_a \in Q$: *Der Endzustand*
- *Transitionsrelation $\delta : Q \times \Gamma \to Q \times \Gamma \times \{ L, N, R \}$, wobei $\{ L, N, R \}$ Die Bewegung des Leseschreibkopfes angibt:*
  - *L: Leseschreibkopf eine Zelle nach links bewegen.*
  - *N: Leseschreibkopf nicht bewegen.*
  - *R: Leseschreibkopf eine Zelle nach rechts bewegen.*

M ist deterministisch, falls δ eine partielle Funktion ist, ansonsten ist M nichtdeterministisch.

Seien $q_1, q_2 \in Q$, $\alpha, \beta \in \Gamma$ und $m \in \{ L,\ N,\ R \}$. q_2 wird als Nachfolgezustand von q_1 bzgl. α, β und m bezeichnet, falls $\delta(q_1, \alpha) = (q_2, \beta, m)$. Anschließend ist q_2 der aktuelle Zustand.

Eine Turingmaschine akzeptiert, sobald sie den Zustand q_a erreicht. Die momentane Konfiguration von einer TM wird mit einem Wort beschrieben.

Definition 4. *Sei $M = (I, \Gamma, Q, \delta, q_0, q_a)$ eine TM und $d = yqsz$ ein momentane Konfiguration von M, wobei gilt: $y, z \in \Gamma^*, q \in Q$ und $s \in \Gamma$.*

$$\textit{PartComp}_M(d) := \{\ w \mid w = \$d_1\$...\$d_n\$ \textit{ wobei } d_1 = d \textit{ und } d_{i+1} \textit{ eine Nachfolgekonfiguration von } d_i \textit{ ist. für alle } i \in \{1, ..., n-1\}\ \}$$

$$\textit{Comp}_M(d) := \{\ w \in \textit{PartComp}_M(d) \mid w = \alpha q_a \beta \textit{ für } \alpha, \beta \in (Q \cup \Gamma \cup \{\$\})^*\ \}$$

Die Wörter in $\textit{PartComp}_M(d)$ sind alle möglichen Abschnitte aller Äste des Konfigurationsbaums [Bes13, Definition 6.4.1] von M, die mit d beginnen. Die Wörter in $\textit{Comp}_M(d)$ sind alle akzeptierenden Äste des Konfigurationsbaums von M.

Eine Mehrband-Turingmaschine (Mehrband-TM) ist eine Turingmaschine, die endlich viele Arbeitsbänder besitzt. Jede Mehrband-TM mit $n \in \mathbb{N}$ Bändern kann von einer TM mit nur einem Band simuliert werden [Bes13, Satz 8.1.1] . Ab hier ist mit Turingmaschine eine Mehrband-Turingmaschinen gemeint, falls es nicht anders angegeben wird.

2.3 Algorithmische Komplexität und Reduktion

Um die Komplexität des Äquivalenzproblems für reguläre Ausdrücke zu untersuchen, werden weitere Defintionen benötigt. Sie werden hier nur erwähnt und entsprechen der angegebenen Literatur.
Die Zeitkomplexität und Platzkomplexität einer Turingmaschine [Bes13, Definition 8.2.3]. Die Komplexitätsklassen *PSPACE*, *NSPACE* [Bes13, Definition 8.4.1] und *NLINSPACE* [vL94, Seite 100, Definition]. Die polynomielle Reduzierbarkeit von Sprachen [Bes13, Definition 8.6.1] und das polynomielle Reduktionlemma [Bes13, Lemma 8.6.3]. Den Begriff der Vollständigkeit für *PSPACE* [Win01, Definition 5.18].

3 Äquivalenzproblem von regulären Ausdrücken

In diesem Kapitel wird die Komplexität des Äquivalenzproblems für reguläre Ausdrücke betrachtet.

Definition 5 (Äquivalenzproblem)**.**

$$\textit{EQ}(\Sigma) := \{\ (re_1, ..., re_n) \mid re_1, ..., re_n \textit{ reguläre Ausdrücke über } \Sigma \textit{ und } L(re_i) = L(re_j) \textit{ wobei } 1 \leq i < j \leq n\ \}$$

Bewiesen wird, dass $\textit{EQ}(\Sigma)$ ein *PSPACE-vollständiges* Problem ist.
Dazu ist zu zeigen:

- $\textit{EQ}(\Sigma) \in \textit{PSPACE}$
- $A \leq_p \textit{EQ}(\Sigma)$, wobei A eine *PSPACE-vollständige* Sprache ist.

Die Ideen für diesen Beweis stammen von *Larry Joseph Stockmeyer* und *Albert Ronald da Silva Meyer* [Sto74][SM73]. Zuerst wird die Komplementbildung von Sprachen in *PSPACE* und *PSPACE-vollständigen* Sprachen untersucht.

Lemma 1 (PSPACE unter Komplementbildung)**.** *Sei A eine Sprache*

$$A \in \textit{PSPACE} \Leftrightarrow \overline{A} \in \textit{PSPACE}$$

Beweis. Sei $A \subseteq \Sigma^*$ eine Sprache, wobei $A \in$ *PSPACE*.
Aus $A \in$ *PSPACE* folgt: Es gibt eine deterministische Turingmaschine M mit polynomiell beschränkten Band, die das Wortproblem für A entscheidet. Also gilt für alle $w \in \Sigma^* : w \in A$ genau dann, wenn M angesetzt auf w akzeptiert. Da M deterministisch ist, gilt $w \in \overline{A}$ genau dann, wenn M nicht akzeptiert. Somit gibt es eine deterministische Turingmaschine $\overline{M}$ mit polynomiell beschränkten Band, die das Wortproblem für $\overline{M}$ entscheidet, indem sie M simuliert und das Ergebnis negiert. □

Lemma 2 (PSPACE-Vollständigkeit unter Komplementbildung)**.** *Sei A eine Sprache.*

$$A \textit{ ist PSPACE-vollständig} \Leftrightarrow \overline{A} \textit{ ist PSPACE-vollständig}$$

Beweis. Seien $A \subseteq \Sigma_1^*, B \subseteq \Sigma_2^*$ Sprachen, wobei $A, B \in$ *PSPACE* und A *PSPACE-vollständig* ist.
Es ist zu zeigen, dass $B \leq_p \overline{A}$ gilt. Wegen der Abgeschlossenheit unter Komplementbildung von *PSPACE* sind auch $\overline{A}, \overline{B} \in$ *PSPACE*. Es gibt somit ein f mit $\overline{B} \leq_p A$ mittels f. Also gilt:

$$\forall x \in \Sigma_2^* : x \in \overline{B} \Leftrightarrow f(x) \in A$$

logisch äquivalent dazu ist

$$\forall x \in \Sigma_2^* : x \in B \Leftrightarrow f(x) \in \overline{A}$$

$\overline{A}$ ist somit *PSPACE-vollständig*. □

Daher genügt es, die *PSPACE-vollständigkeit* von $\overline{EQ(\Sigma)}$ zu zeigen.

Definition 6 (Nicht Äquivalenzproblem)**.**

$$\textit{INEQ}(\Sigma) := \overline{EQ(\Sigma)} = \{\ (re_1, re_2) \mid re_1, re_2 \textit{ reguläre Ausdrücke über } \Sigma \textit{ und } L(re_1) \neq L(re_2)\ \}$$

Es bleibt zu zeigen:

- *INEQ*$(\Sigma) \in$ *PSPACE*
- $A \leq_p$ *INEQ*(Σ), wobei A eine *PSPACE-vollständige* Sprache ist.

Lemma 3.

$$\textit{INEQ}(\Sigma) \in \textit{PSPACE}$$

Beweis. Wegen dem Satz von Savitch [Bes13, Satz 8.4.2], *PSPACE* = *NSPACE*, genügt es eine nichtdeterministische Turingmaschine mit polynomiell beschränktem Band anzugeben. Seien re_1 und re_2 reguläre Ausdrücke über Σ. Wenn es ein Wort $w \in L(re_1) \triangle L(re_2) = (L(re_1) \setminus L(re_2)) \cup (L(re_2) \setminus L(re_1))$ gibt, sind re_1 und re_2 nicht äquivalent. Eine TM M versucht nichtdeterministisch ein Wort $w \in L(re_1) \triangle L(re_2)$ zu finden. Dabei rät M das Wort w Zeichen für Zeichen. re_1 und re_2 werden als nichtdeterministische endliche Automaten (NFA) [Bes13, Definition 4.1.1] dargestellt, die $L(re_1)$ und $L(re_2)$ akzeptieren. M simuliert diese NFA's so, dass sie w als Eingabe erhalten. Wenn genau ein NFA w akzeptiert, ist $w \in L(re_1) \triangle L(re_2)$.
Ein NFA, z.B. für re_1, wird wie folgt in M dargestellt: Für jede Klammer in re_1 gibt es einen Zustand in dem NFA. Sei re_{sub} ein regulärer Ausdruck innerhalb von re_1. Die erste öffnende Klammer von re_{sub} ist der Startzustand und die letzte schließende Klammer ist der Endzustand für einen NFA, der $L(re_{sub})$ akzeptiert. Dies impliziert die Zustandsübergänge aus Abbildung 1 in einem NFA.

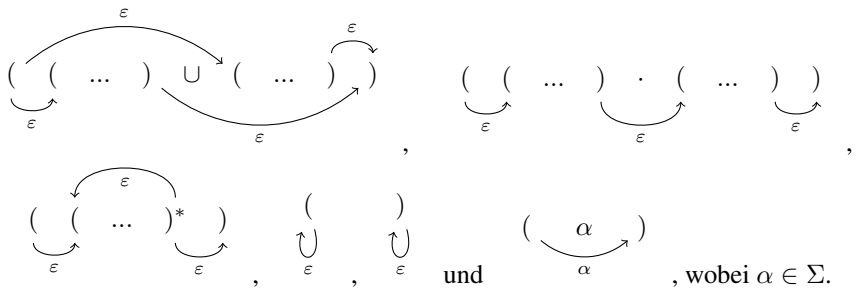

Abbildung 1: Zustandsübergänge

Die Zustandsübergänge in Abbildung 1 müssen nicht explizit auf dem Arbeitsband codiert werden, denn für zwei beliebige Klammern K_1 und K_2 kann M durch Zählen der Klammern zwischen K_1 und K_2 feststellen, ob K_1 und K_2 zu der gleichen Operation $\omega \in \{^*, \cup, \cdot\}$ gehören und ob es einen Übergang von K_1 zu K_2 gibt. Dieser Test kann mit einem Kellerautomaten [Bes13, Definition 5.3.1] mit linear begrenzten Keller durchgeführt werden, da lediglich für jede geöffnete Klammer ein Zeichen in den Keller geschrieben wird.
Wie M die NFA's von re_1 und re_2 für das Wort w simuliert, wird nun beschrieben. Die NFA's erhalten w Zeichen für Zeichen als Eingabe. Ein Zustand wird markiert, falls die bisherige Eingabe zu diesem Zustand führen könnte. Die folgende Methode *update*(σ) wird genutzt, um die Menge der markierten Zustände zu aktualisieren, wobei $\sigma \in \Sigma \cup \{\varepsilon\}$ ist. Nach der Ausführung von *update*(σ) ist ein Zustand P_2 markiert, falls es einen Zustand P_1 gibt ($P_1 = P_2$ möglich), so dass folgende Bedingungen zu treffen.

1. Es gibt eine Zustandsübergang von P_1 zu P_2 für σ.
2. P_1 war vor dem Aufruf von *update*(σ) markiert.

Da kein weiterer Platz für die Zustandsübergänge benötigt wird, kann *update*(σ) so programmiert werden, dass es deterministisch in polynomieller Zeit ausgeführt wird und ein linear beschränktes Band benötigt.
Die Funktionsweise von M ist die folgende:
Auf dem Eingabeband liegt x mit $|x| = n$.

1. Als erstes wird geprüft, ob die Eingabe x zwei reguläre Ausdrücke der Form (re_1, re_2) sind. Es muss also getestet werden, ob jede geöffnete Klammer auch wieder geschlossen wird und ob ausschließlich gültige Operationen vorkommen. Dies impliziert, dass es sich bei (re_1, re_2) um eine kontextfreie Sprache handelt. Dieser Test kann mit einem Kellerautomaten, der sich die Kammern merkt, durchgeführt werden. Dabei benötigt der Kellerautomat höchstens $log(n)^2$ Speicherzellen [LSH65]. Es ist also möglich diesem Kellerautomaten in einer TM mit einem logarithmisch beschränkten Arbeitsband zu simulieren. Falls x nicht die korrekte Form hat, stoppt M.
2. Schreibe re_1 und re_2 jeweils auf ein Arbeitsband.
3. Rufe n-mal *update*(ε) auf, damit alle durch ε-Transitionen erreichbaren Zustände makiert sind.
4. Falls nur von einem NFA ein Endzustand markiert ist, akzeptiert M.
5. Rate nichtdeterministisch ein Zeichen $\sigma \in \Sigma$ und rufe *update*(σ) auf.
6. Falls es Wörter in $L(re_1) \triangle L(re_2)$ gibt, muss es auch ein Wort der Länge $n' \leq max(|re_1|, |re_2|)$ in $L(re_1) \triangle L(re_2)$ geben. Daher stoppt M an dieser Stelle, falls schon mehr als n' Zeichen geraten wurden. Andernfalls springe zu 3.

Folglich benötigt M dauerhaft n Zellen auf den Arbeitsbändern für $x = (re_1, re_2)$ und weitere $log^2(n)$ Zellen für (1). Der Aufruf der *update*(σ) Methode benötigt cn Zellen. Somit benötigt M maximal eine

Bandgröße von $n + log^2(n) + cn$. Also gilt: *INEQ*$(\Sigma) \in$ NSPACE. Wegen dem Satz von Savitch [Bes13, Satz 8.4.2] gilt auch: *INEQ*$(\Sigma) \in$ *PSPACE*.
Somit ist die Behauptung gezeigt. □

Für die *PSPACE-vollständigkeit* von *INEQ*(Σ) ist noch zu zeigen, dass eine *PSPACE-vollständige* Sprache auf *INEQ*(Σ) reduzierbar ist. Das *Erfüllbarkeitsproblem für quantifizierte boolesche Formeln (QBF)* ist *PSPACE-vollständig* [SM73, Theorem 4.3].

Proposition 1.

$$QBF \in NLINSPACE$$

Beweis. Für *QBF* $\in$ *NLINSPACE* wird eine Turingmaschine angegeben. Sei M eine nichtdeterministische Turingmaschine, die wie folgt agiert:

1. Es wird geprüft, ob die Eingabe w mit $n = |w|$ eine quantifizierte boolesche Formel ist. Aufgrund der Klammerung, handelt es sich um eine kontextfreie Sprache. Dieser Test kann von einem Kellerautomaten mit höchstens $log^k(n)$ Speicherzellen durchgeführt werden [LSH65]. Falls w keine *QBF* ist, stoppt M.
2. Es werden Variablen $A := (a_1, ..., a_k)$ und $E := (e_1, ..., e_l)$ angelegt, wobei $a_1, ..., a_k$ alle mit $\forall$ und $e_1, ..., e_l$ alle nicht mit $\forall$ quantifizierten Variablen aus w sind. Zu Beginn wird jede Stelle von A mit 0 belegt, wobei $0 =$ *falsch* und $1 =$ *wahr*.
3. A wird Schritt für Schritt bis $A = 1...1$ binäre hochgezählt. Nach jeder Erhöhung von A wird die Methode *find()* aufgerufen. Falls *find()* für jedes A eine gültige Belegung gefunden hat, ist w erfüllbar. In diesem Fall akzeptiert M. Andernfalls wird gestoppt.

Die Methode *find()* versucht eine Belegung für E zu finden, so dass w, eingeschränkt auf die Belegung von A, zutrifft. Dafür wird nichtdeterministisch eine Belegung für E geraten. Anschließend wird $w' = w$ mit der Belegung von A und E auf ein weiteres Band geschrieben und Schrittweise überprüft, indem w' rekursiv ausgewertet wird. Die Auswertung wird direkt in w' vollzogen. Es wird somit kein weiterer Platz benötigt.
Insgesamt ergibt das eine Platzbegrenzung von:

$$\underbrace{|w|}_{=n} + \underbrace{|A| + |E|}_{\leq n} + \underbrace{|w'|}_{\leq n} + \underbrace{log^k(n)}_{\leq kn} \leq (4 + k)n$$

Somit ist die Behauptung gezeigt. □

Um *QBF* auf *INEQ*(Σ) zu reduzieren, wird der folgenden Spezialfall von *INEQ*(Σ) benutzt.

Definition 7.

$$NEC(\Sigma) := \{\ re \mid re \text{ ist reguläre Ausdrücke über } \Sigma \text{ und } L(re) \neq \Sigma^*\ \}$$

Korollar 1.

$$NEC(\Sigma) \leq_p INEQ(\Sigma)$$

Beweis. Sei re ein regulärer Ausdrück über Σ und $\Sigma_M = \Sigma \cup \{\ \cup,\ \cdot,\ ^*,\ (,\ )\ \}$ Dann gilt:

$$\begin{aligned}
& re \in NEC(\Sigma) \\
\Leftrightarrow\ & L(re) \neq \Sigma^* \\
\Leftrightarrow\ & L(re) \neq L((\sigma_1 \cup \sigma_2 \cup ... \cup \sigma_l)^*), \text{ wobei } \sigma_i \in \Sigma \text{ für } 1 \leq i \leq l = |\Sigma| \\
\Leftrightarrow\ & (re, (\sigma_1 \cup \sigma_2 \cup ... \cup \sigma_l)^*) \in INEQ(\Sigma)
\end{aligned}$$

Es reicht also, wenn eine totale Funktion $f : \Sigma_M \to \Sigma_M$ den regulären Ausdruck Σ^* hinter die Eingabe schreibt. $f(x) = (x, \Sigma^*)$, wobei $x \in \Sigma_M$ ist. Da die benötigten Zeichen für Σ^* von der Eingabe unabhängig sind, kann eine deterministische TM M_f in $|x|+1$ Schritten an das Ende der Eingabe springen und mit konstantem Aufwand Σ^* schreiben. Somit gilt $x \in$ *NEC*$(\Sigma) \Leftrightarrow f(x) \in$ *INEQ*(Σ) mittels f, wobei M_f deterministisch polynomiell zeitbeschränkt ist.
Damit ist das Korollar gezeigt. □

Statt *QBF* direkt auf *NEC*(Σ) zu reduzieren, wird gezeigt, dass jede Sprache in *NLINSPACE* auf *NEC*(Σ) reduzierbar ist.

Satz 2. *Sei A eine Sprache.*

$$A \in \textit{NLINSPACE} \Rightarrow \textit{Es existiert ein } \Sigma\textit{, so dass gilt: } A \leq_p \textit{NEC}(\Sigma)$$

Im folgenden Beweis werden Abkürzungen in regulären Ausdrücken benutzt:

- Abkürzung: $[\Sigma^k] = \Sigma \cdot ... \cdot \Sigma$ (k-mal)
  Bedeutet: k beliebige Zeichen aus Σ
  Beispiel: $\Sigma = \{a, b\}, [\Sigma^2] = (a \cup b) \cdot (a \cup b)$
- Abkürzung: $[\Sigma^{\leq k}] = (\Sigma \cup \varepsilon) \cdot ... \cdot (\Sigma \cup \varepsilon)$ (k-mal)
  Bedeutet: Maximal k beliebige Zeichen aus Σ
  Beispiel: $\Sigma = \{a, b\}, [\Sigma^{\leq 2}] = (a \cup b \cup \varepsilon) \cdot (a \cup b \cup \varepsilon)$
- Abkürzung: $\overline{\lambda} = \Sigma \setminus \{\lambda\}$, wobei $\lambda \in \Sigma$
  Bedeutet: Ein beliebiges Zeichen aus Σ außer λ
  Beispiel: $\Sigma = \{a, b, c\}, \overline{c} = \{a, b, c\} \setminus \{c\} = (a \cup b)$

Dies dient der Lesbarkeit und hat keine semantische Bedeutung.

Beweis. Sei $A \in$ *NLINSPACE* eine Sprache und $M = (I, \Gamma, Q, \delta, q_0, q_a)$ eine nichtdeterministische TM, die A mit einem $S(n) = cn$ beschränkten Band akzeptiert. Sei $x \in I^*, n = |x|$ und $\Sigma = Q \cup \Gamma \cup \{\$\}$ wobei $\$ \notin Q \cup \Gamma$. $E_M(x)$ definiert einen regulären Ausdruck über Σ, dass gilt:
$L(E_M(x)) = \Sigma^* \setminus \text{Comp}_M(q_0 x \sqcup^{S(n)-n})$. Also:

$$E_M(x) \in NEC(\Sigma) \Leftrightarrow L(E_M(x)) \neq \Sigma^* \Leftrightarrow \text{Comp}_M(q_0 x \sqcup^{S(n)-n}) \neq \emptyset \Leftrightarrow x \in A$$

Sei $f_M : I^* \to \Sigma^*, x \mapsto E_M(x)$, so gilt: $\forall x \in I^* : x \in A \Leftrightarrow f_M(x) \in$ *INEQ*(Σ).

Die Wörter in $L(E_M(x))$ können wie folgt charakterisiert werden: $w \in \Sigma^* \setminus \text{Comp}_M(q_0 x \sqcup^{S(n)-n})$, falls:

1. *Beginnt falsch*: w beginnt nicht mit $\$q_0 x \sqcup^{S(n)-n} \$$. Dies lässt sich mit einem regulären Ausdruck $re_1 := re_{1,1} \cup re_{1,2} \cup re_{1,3}$ testen. Sei $x = x_1 x_2 ... x_n$. Dann sei
   - $re_{1,1} := [\Sigma^{\leq S(n)+2}]$
   - $re_{1,2} := ([\Sigma^{n+2}] \cdot [\Sigma^{\leq S(n)-n-1}] \cdot \overline{\sqcup} \cdot \Sigma^*) \cup ([\Sigma^{S(n)+2}] \cdot \overline{\$} \cdot \Sigma^*)$
   - $re_{1,3} := (\overline{\$} \cup \$ \cdot (\overline{q_0} \cup q_0 \cdot (\overline{x_1} \cup x_1 \cdot (\overline{x_2} \cup ... \cdot (\overline{x_{n-1}} \cup x_{n-1} \cdot \overline{x_n})...)))) \cdot \Sigma^*$

   Es gilt also: $w \in re_{1,1}$, falls w zu kurz ist, $w \in re_{1,2}$, falls w zu lang ist und $w \in re_{1,3}$, falls w nicht der Form $\$q_0 x...$ entspricht.
2. *Bleibt stehen*: $w = \alpha_1\sigma_1\sigma_2\sigma_3\beta\sigma_4\sigma_5\sigma_6\alpha_2$, wobei $\alpha_1, \alpha_2 \in \Sigma^*, \beta \in \Sigma^{S(n)-1}, \sigma_i \in \Sigma$ für $i \in \{1, ..., 6\}$ und das Arbeitsband von M kann sich in einem Schritt nicht von $\sigma_1\sigma_2\sigma_3$ zu $\sigma_4\sigma_5\sigma_6$ verändern. Zur Überprüfung des Falls lässt sich wieder ein regulärer Ausdruck formulieren.

$$re_2 := \Sigma^* \cdot (\bigcup_{\sigma_1\sigma_2\sigma_3 \in \Sigma^3} \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdot [\Sigma^{S(n)-1}] \cdot (\Sigma^3 \setminus \mathrm{N}_M(\sigma_1\sigma_2\sigma_3))) \cdot \Sigma^*$$

wobei für $z, y \in \Gamma, q \in Q$:

$$\mathrm{Next}_M(z,q,y) = \{\ s \in \Sigma^3 \mid s = \begin{cases} zq'y' & \text{, falls } m = N \\ q'zy' & \text{, falls } m = L \\ zy'q' & \text{, falls } m = L \end{cases} \quad \text{wobei } \delta(q,y) = (q', y', m)\ \}$$

und $\sigma_1\sigma_2\sigma_3 \in \Sigma^3$:

$$N_M(\sigma_1\sigma_2\sigma_3) = \{\ \sigma_1'\sigma_2'\sigma_3' \in \Sigma^3 \mid \sigma_1', \sigma_2', \sigma_3' \text{ haben die folgenden 4 Bedingungen bzgl. } \sigma_1, \sigma_2, \sigma_3\ \}$$

1. $\sigma_i = \$ \Leftrightarrow \sigma_i' = \$$, für $i \in \{1,2,3\}$
2. $\sigma_1, \sigma_2, \sigma_3 \notin Q \Rightarrow \sigma_2 = \sigma_2'$
3. $\sigma_2 \in Q$ und $\sigma_3 \in \Gamma \Rightarrow \sigma_1'\sigma_2'\sigma_3' \in \mathrm{Next}_M(\sigma_1, \sigma_2, \sigma_3)$
4. $\sigma_2 \in Q$ und $\sigma_3 = \$ \Rightarrow N_M(\sigma_1\sigma_2\sigma_3) = \emptyset$

3. *Endet falsch*: w enthält nicht den q_a Zustand oder endet nicht mit \$. Auch hierfür kann ein regulärer Ausdruck konstruiert werden.

$$re_3 := (\overline{q_a})^* \cup (\Sigma^* \cdot \overline{\$})$$

Somit lässt sich $E_M(x)$ mit $re_1 \cup re_2 \cup re_3$ beschreiben.
Es bleibt zu zeigen, dass f_M von einer deterministische TM M_d in polynomieller Zeit berechnet werden kann, damit $A \leq_p NEC(\Sigma)$ mittels f_M gilt. Die Arbeitsweise von M_d ist wie folgt:

1. Eingabe x einlesen
2. $E_M(x)$ erstellen
3. $E_M(x)$ ausgeben

Die offensichtliche Frage ist, ob $E_M(x)$ in polynomieller Zeit erstellt werden kann. Dafür wird zuerst nur auf einen Teil von $E_M(x)$ eingegangen.

$$C_1 := \{\ \Sigma,\ \overline{\sqcup}\ ,\ \overline{\$},\ \overline{q_a}\ \}$$
$$C_2 := \{\ \Sigma^3 \setminus \mathrm{N}_M(x) \mid x \in \Sigma^3\ \}$$
$$C := C_1 \cup C_2 \cup \Sigma$$

Bei genauer Betrachtung der Elemente von C ist festzustellen, dass sie unabhängig von der Eingabe sind und somit bereits zur Konstruktionszeit von M_d eindeutig bestimmtbar sind. Also kann M_d so konstruiert werden, dass alle Elemente von C in konstanter Zeit auf das Arbeitsband geschrieben werden können. Da re_3 nur aus Elementen von C bestehen, ist der zeitliche Aufwand für re_3 konstant. Die Anzahl der für re_3 benötigten Schritte werden ab hier mit k_0 bezeichnet. Es bleibt zu zeigen, dass M_d die regulären Ausdrücke re_1 und re_2 in polynommieller Zeit erstellen kann.

Da $\Sigma \in C$ ist, kann Σ in k_1 Schritten auf das Arbeitsband geschrieben werden, wobei k_1 konstant ist. Für das Erstellen von $[\Sigma^{S(n)+2}]$ werden $k_1(S(n)+2)$ Schritten benötigt. Folglich kann $re_{1,1} = [\Sigma^{\leq S(n)+2}]$ in $k_2(S(n)+2)$ Schritten erstellt werden, wobei k_2 konstant ist und den zusätzlichen Aufwand für das ε berücksichtigt.

$$re_{1,2} = (\ \underbrace{[\Sigma^{n+2}]}_{k_1(n+2) \text{ Schritte}} \cdot \overbrace{[\Sigma^{\leq S(n)-n-1}]}^{k_2(S(n)-n-1) \text{ Schritte}} \underbrace{\overline{\sqcup} \cdot \Sigma^*) \cup (}_{k_3 \text{ Schritte}} \overbrace{[\Sigma^{\leq S(n)+2}]}^{k_2(S(n)+2) \text{ Schritte}} \underbrace{\cdot \overline{\$} \cdot \Sigma^*)}_{k_4 \text{ Schritte}}, \text{ wobei } k_3, k_4 \text{ konstant}$$

Für $re_{1,2}$ benötigt M_d also $p_1(n) := 1 + k_1(n+2) + k_2(S(n) - n - 1) + k_3 + k_2(S(n) + 2) + k_4$ Schritte. p_1 ist ein Polynom.

$$re_{1,3} = \underbrace{(\overline{\$} \cup \$ \cdot (\overline{q_0} \cup q_0}_{k_5 \text{ Schritte}} \overbrace{\cdot (\overline{x_1} \cup x_1}^{k_6 \text{ Schritte}} \underbrace{\cdot (\overline{x_2} \cup \ldots \cdot (\overline{x_{n-1}} \cup x_{n-1}}_{k_6(n-1) \text{ Schritte}} \overbrace{\cdot \overline{x_n}}^{k_7 \text{ Schritte}} \underbrace{)\ldots))))}_{n+1 \text{ Schritte}} \overbrace{\cdot \Sigma^*}^{k_8 \text{ Schritte}}$$

, wobei k_5, k_6, k_7, k_8 konstant sind. Für $re_{1,3}$ benötigt M_d also $p_2(n) := k_5 + k_6(n-1) + k_7 + k_8 + n + 1$ Schritte. p_2 ist ein Polynom.

$$re_2 = \overbrace{\Sigma^* \cdot (}^{k_9 \text{ Schritte}} \underbrace{\bigcup_{\sigma_1\sigma_2\sigma_3 \in \Sigma^3} \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdot \overbrace{[\Sigma^{S(n)-1}]}^{k_1(S(n)-1) \text{ Schritte}} \cdot \overbrace{(\Sigma^3 \setminus \mathrm{N}_M(\sigma_1\sigma_2\sigma_3))}^{k_{10} \text{ Schritte}}}_{3^{|\Sigma|}(k_1(S(n)-1)+k_{10}+k_{11}) \text{ Schritte}} \overbrace{) \cdot \Sigma^*}^{k_{12} \text{ Schritte}}$$

, wobei $k_9, k_{10}, k_{11}, k_{12}$ konstant sind. Für re_2 benötigt M_d also $p_3(n) := k_9 + 3^{|\Sigma|}(k_1(S(n)-1) + k_{10} + k_{11}) + k_{12}$ Schritte. p_3 ist ein Polynom.
Für das Erstellen von $E_M(x)$ werden $p(n) := p_1(n) + p_2(n) + p_3(n) + k_0$ Schritte benötigt. Also ist M_d polynomiell zeitbeschränkt.
Damit ist gezeigt:

$$A \leq_p \textit{NEC}(\Sigma) \text{ mittels } f_M$$

□

Nun sind alle Voraussetzungen für den Beweis der *PSPACE-Vollständigkeit* von $EQ(\Sigma)$ vorhanden.

Korollar 2.

$$\textit{EQ}(\Sigma) \textit{ ist PSPACE-vollständig}$$

Beweis. *INEQ*(Σ) ist *PSPACE-vollständig*, wegen:

1. *INEQ*$(\Sigma) \in$ *PSPACE*, gezeigt in Lemma 3
2. *QBF* $\in$ *NLINSPACE*, beschrieben in Proposition 1
3. *QBF* ist *PSPACE-vollständig*, zitiert aus [SM73, Theorem 4.3]
4. *QBF* $\leq_p$ *NEC*$(\Sigma) \leq_p$ *INEQ*(Σ), gezeigt in Satz 2 und Korollar 1

In Lemma 2 wurde bereits gezeigt, dass das für die *PSPACE-Vollständigkeit* von *EQ*(Σ) ausreicht. □

4 Zusammenfassung

In Kapitel 3 wird bewiesen, dass *EQ*(Σ) ein *PSPACE-vollständiges* Problem ist. Nach Lemma 2 reicht es aus, die *PSPACE-Vollständigkeit* von *INEQ*(Σ) zu zeigen. Lemma 3 besagt:

$$\textit{INEQ}(\Sigma) \in \textit{PSPACE}$$

Das *PSPACE-vollständige* Problem *QBF*, beschrieben in Proposition 1, ist durch Satz 2 auf *NEC*(Σ) reduzierbar. Mit Korollar 1 ist *NEC*(Σ) auf *INEQ*(Σ) reduzierbar. Durch die Transitivität der polynomiellen Reduktion gilt dann:

$$\textit{QBF} \leq_p \textit{INEQ}(\Sigma)$$

Insgesamt ist dadurch die *PSPACE-Vollständigkeit* von *INEQ*(Σ) und *EQ*(Σ) gezeigt.

Literatur

[Bes13] E. Best. Theoretische Informatik II, Vorlesungsskript, 2013.

[CF01] Pascal Caron und Marianne Flouret. Glushkov construction for multiplicities. In *Implementation and Application of Automata*, Seiten 67–79. Springer, 2001.

[LSH65] P. M. Lewis, R. E. Stearns und J. Hartmanis. Memory Bounds for Recognition of Context-free and Context-sensitive Languages. In *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, FOCS '65, Seiten 191–202, Washington, DC, USA, 1965. IEEE Computer Society.

[SM73] Larry J Stockmeyer und Albert R Meyer. Word problems requiring exponential time (Preliminary Report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, Seiten 1–9. ACM, 1973.

[Sto74] Larry Joseph Stockmeyer. The complexity of decision problems in automata theory and logic. 1974.

[vL94] Jan van Leeuwen. *Handbook of theoretical computer science (vol. A): algorithms and Complexity*. MIT Press Cambridge, MA, 1994.

[Win01] Renate Winter. *Theoretische Informatik: Grundlagen mit ÃIJbungsaufgaben und LÃűsungen*. Oldenbourg Wissenschaftsverlag, Berg am Laim, 2001.