Probeklausur
Theoretische Informatik 2
4. Juli 2023

Prof. Dr. Roland Meyer
René Maseli
TU Braunschweig
Sommersemester 2023

1. **Bitte am Anfang ausfüllen:**

   Vorname:

   Nachname:

   Matrikelnummer:

   Nummer des Sitzplatzes:

   Unterschrift:

2. Die Nummer Ihrer Klausur ist `# 0`. Bitte merken Sie sich die Nummer. Wir werden Ihr Klausurergebnis anonymisiert unter Verwendung dieser Nummer bekanntgeben.
3. Achten Sie darauf, dass Ihre Klausur **vollständig ist** und **getackert bleibt** (20 Blätter).
4. Benutzen Sie **nur das an dieses Blatt angeheftete Papier**. Bei Bedarf können wir weitere Leerblätter austeilen. Wenn der Platz auf der Vorderseite des jeweiligen Aufgabenblatts nicht ausreicht, **machen Sie kenntlich**, wo Sie die Bearbeitung der Aufgabe fortsetzen.
5. Als Hilfsmittel sind **ausschließlich** Sprachwörterbücher sowie ein beidseitig **handschriftlich beschriebenes DIN A4-Blatt** erlaubt. Elektronische Geräte müssen während der Klausur ausgeschaltet bleiben. Täuschungsversuche werden als nicht bestanden gewertet und dem Prüfungsamt gemeldet.
6. Schreiben Sie leserlich und bearbeiten Sie Ihre Klausur mit einem **dokumentenechten Stift** (nicht mit Bleistift, kein Tipp-Ex, kein Tintenkiller) und **nicht in roter oder grüner Farbe**.
7. Wir werden die **Klausurergebnisse** auf unserer Website bekanntgeben: `tcs.cs.tu-bs.de/teaching/TheoInf2_SS_2023.html`.
8. Die **Bearbeitungszeit** beträgt **120 Minuten** (+ ggf. Zeit zum Lüften).
9. Mit **28 Punkten** ist die Klausur **sicher bestanden.**

Bepunktung: (wird von den Korrektoren ausgefüllt)

Aufgabe	1	2	3	4	5	6	7	Σ
Max.	10	10	10	10	10	10	10	**70**
Punkte								

1. Konstruktion einer DTM

10 Punkte

Konstruieren Sie eine **deterministische** Turingmaschine M, welche die Sprache

$$L = \left\{ a^m b^n \;\middle|\; m, n > 0 \text{ UND } m^2 < 3n \right\}$$

entscheidet. Beispielsweise sind $ab, aabb \in L$, aber $aab, aaabbb \notin L$.

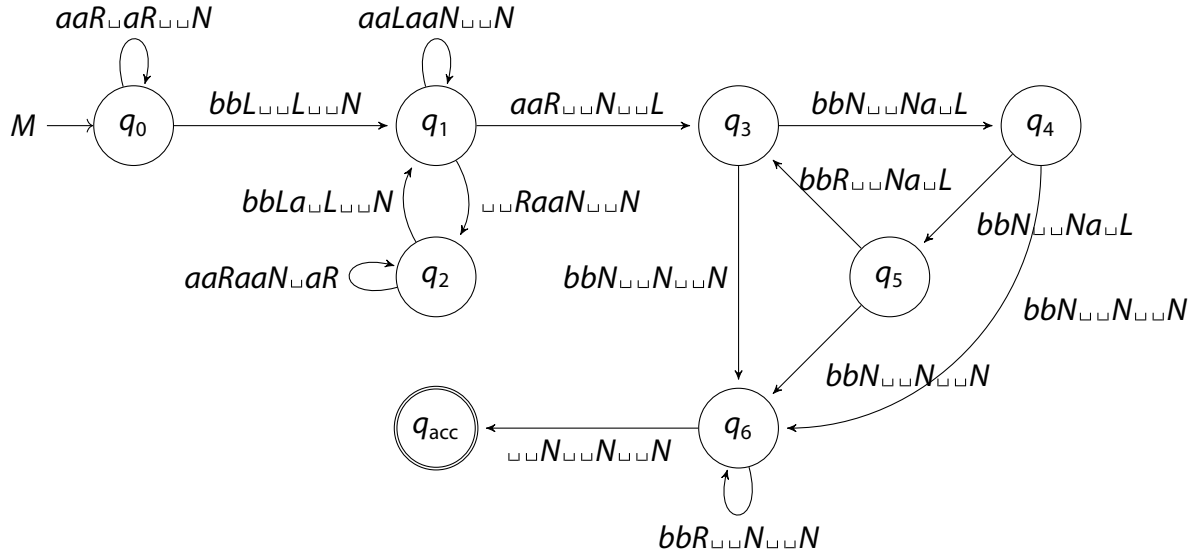
- Erklären Sie die Arbeitsweise der Maschine ausführlich. Geben Sie insbesondere die Aufgabe jedes Kontrollzustands der Maschine an.
- Geben Sie die Transitionen der Maschine explizit an, z.B. in Form einer Tabelle oder als Zustandsgraph. Im Zustandsgraphen brauchen Sie Transitionen nach q_{rej} nicht zu zeichnen.
- Sie können wahlweise annehmen, dass das Band auf beiden Seiten der Eingabe mit ␣-Symbolen gefüllt ist, oder dass das Band auf der linken Seite durch ein \$-Symbol beschränkt ist. Geben Sie an, wofür Sie sich entschieden haben und geben Sie an, auf welches Symbol der Lese-/Schreibkopf initial zeigt.

Hinweis: Die Turingmaschine darf mehrere Bänder verwenden.

Vorschlag:

Zustand q_0 kopiert a^m in ein Arbeitsband. Für jedes dieser m Symbole, hängen q_1 und q_2 zusammen a^m in ein weiteres Arbeitsband an. Für jedes der n b-Symbole, versuchen q_3 bis q_5 ein a^3 aus dem zweiten Band zu entfernen. Zustand q_6 wird nur dann erreicht, wenn es mindestens ein a zu wenig gab. Akzeptiere nur dann und wenn die Form a^*b^* passt.

Die Turing-Maschine benutzt also drei Bänder und jede Transition hat dementsprechend drei Symbol-Symbol-Richtung-Tripel. Im folgenden Graphen sind alle Kanten in den ablehnenden Zustand ausgeblendet.



2. NL-Vollständigkeit

6 + 4 Punkte

Sei Σ eine endliche Menge. Ein Σ-gelabelter Graph $G = \langle V, E, \ell \rangle$ besteht aus einem gerichteten Graphen $\langle V, E \rangle$ mit $E \subseteq V \times V$ und einer Kantenlabel-Funktion $\ell : E \to \Sigma$, welche adjazente Knotenpaare $\langle u, v \rangle \in E$ auf ein Label $\ell(u, v) \in \Sigma$ abbildet.

Jeder Pfad p in G, bestehend aus einer Knotenfolge $p = \langle s, v_1, v_2, \ldots, v_k, t \rangle$, beschreibt ein Wort über dem Alphabet Σ, genauer $\ell^*(p) := \ell(s, v_1)\ell(v_1, v_2)\ldots\ell(v_k, t) \in \Sigma^*$.

Betrachten Sie das folgende Problem.

Pfadproblem mit Endlichem Automat (DFA-PATH)

Gegeben: Ein $\{0, 1\}$-gelabelter Graph $G = \langle V, E, \ell \rangle$, Knoten $s, t \in V$ und einen DFA A über dem Alphabet $\{0, 1\}$.

Entscheide: Gibt es in G einen s-t-Weg p mit $\ell^*(p) \in \mathcal{L}(A)$?

Zeigen Sie, dass DFA-PATH NL-vollständig (bzgl. logspace-many-one-Reduktionen) ist:

a) „Membership“: DFA-PATH $\in$ NL.

b) „Hardness“: DFA-PATH ist NL-schwer (bzgl. logspace-many-one-Reduktionen).

Vorschlag:

a) Idee: Bilde das Produkt vom Graphen und dem Zustandsgraphen der Maschine. Kombinierte Kanten müssen dabei das gleiche Label haben. Finde einen Pfad nach $\{t\} \times Q_F$. Der folgende Algorithmus entscheidet DFA-PATH:

input: $\langle V, E, \ell \rangle, s, t, \langle Q, q_0, \to, Q_F \rangle$
$v \leftarrow s \quad q \leftarrow q_0 \quad i \leftarrow |V| \cdot |Q|$
while $i > 0$ und $\langle v, q \rangle \notin \{t\} \times Q_F$ **do**
  Rate $\langle v, w \rangle \in E$ und $q \xrightarrow{\ell(v,w)} q'$
  $v \leftarrow w \quad q \leftarrow q' \quad i \leftarrow i - 1$
end while
if $i > 0$, **accept**

b) PATH $\leq_m^{\log}$ DFA-PATH: Idee: Nutze einen DFA A_{Σ^*} für $\{0, 1\}^*$.
Definiere $f(V, E, s, t) = \langle V, E, \ell, s, t, A_{\Sigma^*} \rangle$ mit z.B. für alle $e \in E$ sei $\ell(e) = 0$. Das ist problemlos LogSpace-berechenbar.

Falls $\langle G, s, t \rangle \in$ PATH, dann gibt es einen s-t-Pfad p in G. Dieser ist auch ein s-t-Weg in G' und erfüllt $\ell^*(p) = 0^{|p|} \in \mathcal{L}(A_{\Sigma^*})$. Darum folgt $\langle G', s, t, A_{\Sigma^*} \rangle \in$ DFA-PATH.

Falls andersherum $\langle G', s, t, A_{\Sigma^*} \rangle \in$ DFA-PATH. Dann gibt es einen s-t-Weg p in G (mit $\ell^*(p) \in \mathcal{L}(A_{\Sigma^*})$). Dieser ist auch ein s-t-Weg in G. Nach Entfernen aller Schleifen aus p kann man einen s-t-Pfad in G erhalten. Darum folgt $\langle G, s, t \rangle \in$ PATH.

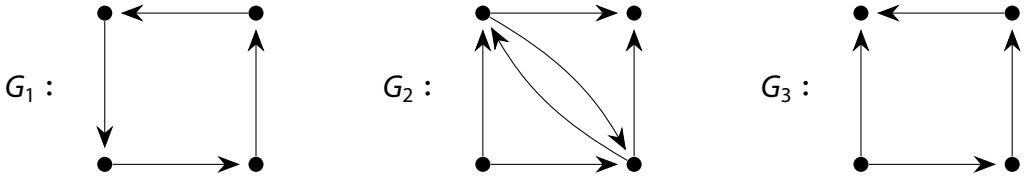
3. NP-Vollständigkeit

4 + 6 = 10 Punkte

Betrachten Sie das folgende Problem.

2-Hamilton-Pfade (2HP)
Gegeben: Ein gerichteter Graph $G = \langle V, \rightarrow \rangle$.
Entscheide: Gibt es zwei unterschiedliche Pfade in G, die jeweils alle Knoten besuchen?

Dabei sind zum Beispiel $G_1 \in$ 2HP und $G_2 \in$ 2HP, aber $G_3 \notin$ 2HP.



Zeigen Sie, dass 2HP NP-vollständig (bzgl. Polynomialzeit-Reduktionen) ist:

a) „Membership“: 2HP $\in$ NP.

b) „Hardness“: 2HP ist NP-schwer (bzgl. Polynomialzeit-Reduktionen).

Vorschlag:

a) Idee: Rate zwei Pfade und prüfe, ob sie den Anforderungen genügen.

input: $G = \langle V, \rightarrow \rangle$
Rate zwei Knoten-Sequenzen p_1, p_2 jeweils der Länge $|V|$.
if $p_1 \neq p_2$ und sowohl p_1 und p_2 hamiltonsche Pfade sind, **accept**

Sei M eine NTM, die diesen Algorithmus implementiert. M benötigt linear viel Zeit, um zu raten und die Ungleichheit zu prüfen, und weiterhin quadratisch viel Zeit, um zu testen, dass sich kein Knoten wiederholt, sowie dass die Kanten existieren.

Falls $G \in \mathcal{L}(M)$, dann gibt es mindestens eine akzeptierende Berechnung in M. Diese musste per Konstruktion zwei Knotenfolgen erraten haben und erfolgreich getestet haben, dass sie unterschiedliche Hamiltonsche Pfade sind. Also gibt es beide Pfade und $G \in$ 2HP.

Falls $G \in$ 2HP, dann gibt es mindestens zwei unterschiedliche hamiltonsche Pfade in G. Dann kann wird M diese erraten können. Die Berechnung, die das tat, wird dies anschließend feststellen und akzeptieren. Also $G \in \mathcal{L}(M)$.

b) HC $\leq_m^{\log}$ 2HP: Gesucht ist $f : \Sigma^* \to \Gamma^*$ mit $w \in \text{HC} \iff f(w) \in \text{2HP}$.

Idee: Erzwinge wo beide Pfade beginnen und enden, sollten sie existieren. Verdopple die Anzahl der Möglichkeiten.

Sei $G = \langle V, \to \rangle$ ein gerichteter Graph. Das Bild $\langle V', \to' \rangle$ kann wiefolgt konstruiert werden:
Wähle einen beliebigen Knoten $v \in V$.
Ersetze ihn mit 4 Knoten x, y, s, t.
s erhält die ausgehenden Kanten von v.
t erhält die eingehenden Kanten nach v.
Dazu gibt es die Kanten $x \to y$, $y \to x$, $x \to s$ und $y \to s$.
Alle weiteren Knoten und Kanten bleiben unverändert wie in G.

Eine solche Funktion f ist sogar LogSpace-berechenbar.

Falls $G \in$ HC, dann gibt es einen Kreis, der alle Knoten genau einmal durchläuft. Seine Folge vwv sei so gewählt, dass sie mit v beginnt. Nun besuchen beide Pfade $xyswt \neq yxswt$ jeweils alle Knoten. Es folgt $f(G) \in$ 2HP.

Falls $f(G) \in$ 2HP, dann gibt es zwei Hamiltonsche Pfade p_1 und p_2 in $f(G)$. Betrachte p_1. Dieser muss nach Konstruktion von der Form $p_1 = xyswt$ oder $p_1 = yxswt$ sein. Nun ist vwv ein hamiltonscher Kreis in G. Also gilt $G \in$ HC.

4. Entscheidbarkeit II

10 Punkte

Betrachten Sie das folgende Problem.

Strict-Regular-Upward-Boundedness (SRUB)	
Gegeben:	DTM M und DFA A mit Eingabealphabet $\{0,1\}$.
Entscheide:	Ist $\mathcal{L}(M) \neq \mathcal{L}(A)$, aber liegt jedes $x \in \mathcal{L}(M)$ auch in $\mathcal{L}(A)$?

Beweisen Sie, dass SRUB weder semi-entscheidbar, noch co-semi-entscheidbar ist.

Vorschlag:

Zeige dazu $\overline{\text{UNIVERSALITY}} \leq \text{SRUB}$. Da $\overline{\text{UNIVERSALITY}}$ weder semi- noch cosemientscheidbar ist, folgen dann beide Aussagen auch für SRUB. Gesucht ist eine berechenbare Funktion, die eine TM M auf ein TM-DFA-Paar $\langle M', A\rangle$ abbildet, sodass $\mathcal{L}(M) = \{0,1\}^* \iff \mathcal{L}(M') \subset \mathcal{L}(A)$ gilt.

Lass dazu $M' = M$ unverändert und wähle ein A mit $\mathcal{L}(A) = \{0,1\}^*$. Diese Funktion ist sogar ganz einfach berechenbar.

Korrektheit / Soundness:

$$\begin{aligned} M \notin \text{UNIVERSALITY} &\iff \mathcal{L}(M) \neq \{0,1\}^* \\ &\iff \mathcal{L}(M) \subset \{0,1\}^* \\ &\iff \mathcal{L}(M') \subset \mathcal{L}(A) \\ &\iff \langle M', A\rangle \in \text{SRUB}\,. \end{aligned}$$

Alternativ: Zeige $\text{HP} \leq \text{SRUB}$ und $\overline{\text{HP}} \leq \text{SRUB}$ einzeln. Für Letzeres konstruiere ein A_x mit $\mathcal{L}(A_x) = \{x\}$…Für Ersteres konstruiere den **inversen** Verifizierer für akzeptierende M-Berechnungen auf x: $\mathcal{L}(V_{Mx}) = \{\langle r\rangle \mid r \text{ ist keine akzeptierende Berechnung von } M \text{ auf } x\}$ (entscheidbar und berechenbar). Dazu nehme A_{Σ^*}…

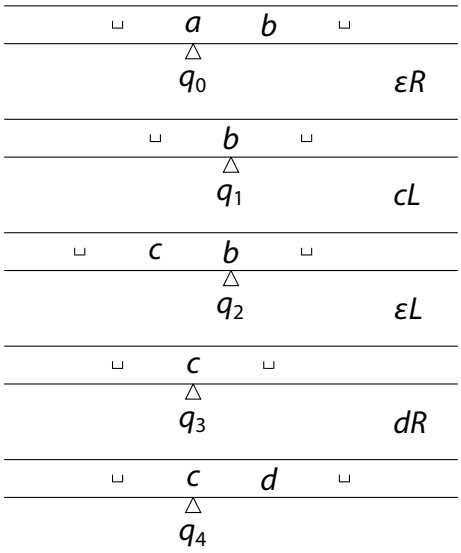
Hinweis: Falls die Sprache (Co-)Semi-Entscheidbar ist, muss ein passender Algorithmus konstruiert werden. Dazu zeige, dass dieser **genau dann** akzeptiert, wenn die Eingabe in der Sprache ist.

5. Insert-Delete-Maschinen

5+5=10 Punkte

Wir definieren eine alternative Turingmaschine, die statt Band-Zellen zu überschreiben und sich zu benachbarten Zellen zu bewegen, mittels vier Operationen Zellen einfügen und entfernen kann. Hierzu benutzen wir ε als Markierung für Lösch-Transitionen.

Zum Beispiel entfernt $\delta(q_0, a) = \langle \varepsilon, R, q_1 \rangle$ die aktuelle Zelle und geht nach rechts, $\delta(q_1, b) = \langle c, L, q_2 \rangle$ fügt eine Zelle mit c links neben der aktuellen Zelle ein, $\delta(q_2, b) = \langle \varepsilon, L, q_3 \rangle$ löscht die aktuelle Zelle und geht nach links und $\delta(q_3, c) = \langle d, R, q_4 \rangle$ fügt d in eine neue Zelle rechts daneben hinzu.



Auch hier gilt, Insert-Delete-Maschinen definieren eine Sprache aus genau den Wörtern, auf die eine Berechnung in einen akzeptieren Zustand möglich ist.

Beweisen Sie:

a) Jede durch einen Insert-Delete-Maschine akzeptierte Sprache ist semientscheidbar.

b) Jede semientscheidbare Sprache wird durch eine Insert-Delete-Maschine akzeptiert.

Vorschlag:

a) Sei M eine Insert-Delete-Maschine (IDM). Es lässt sich wiefolgt eine deterministische Turing-Maschine M' mit $\mathcal{L}(M') = \mathcal{L}(M)$ konstruieren, wonach dann $\mathcal{L}(M)$ semientscheidbar ist. Die DTM M' simuliert M, indem sie bei einer Insert-Transition $\delta(p, a) = \langle b, D, q \rangle$ alle Symbole in Richtung D um eine Zelle verschiebt, und b in die neue Lücke einfügt. Bei einer Delete-Transition $\delta(p, a) = \langle \varepsilon, D, q \rangle$ verschiebt sie alle Symbole D vom Kopf zurück, um das Symbol zu überschreiben. Die Simulation startet im initialen Zustand von M. M' akzeptiert, wenn die Simulation einen akzeptierenden Zustand von M erreicht.

(Details: Dazu sollte M' ein weiteres Band-Symbol nutzen. Für jede Transition von M reichen endlich viele Zustände um diese zu simulieren.)

(Alternative: Die TM nutzt ein zweites Band, um wie bei einem 2-Pushdown z.B. den linken Inhalt der IDM zu speichern.)

b) Sei L semientscheidbar. Es gibt nun eine DTM M, sodass $\mathcal{L}(M) = L$ gilt. Es lässt sich eine IDM M' mit $\mathcal{L}(M') = L$ konstruieren. Die IDM M' simuliert nicht-bewegende TM-Transitionen von M mit einer Sequenz aus vier IDM-Transitionen, wie im Beispiel dargestellt. Dabei muss nur $d = b$ gesetzt werden. Sie simuliert bewegende TM-Transitionen mit zwei IDM-Transitionen. Rechts-Bewegungen wie die ersten beiden Transitionen des Beispiels, Links-Bewegungen wie die letzten beiden. Gestartet wird im initialen Zustand von M. Akzeptiert wird, sobald ein akzeptierender Zustand von M erreicht wird.

6. Quiz

2+2+3+3=10 Punkte

Beantworten Sie die folgenden Fragen. Begründen Sie Ihre Entscheidung mit einem kurzen Beweis oder einem Gegenbeispiel.

a) Ist die Menge der unentscheidbaren Sprachen abzählbar?

b) Falls das Problem A in NP liegt, aber B unentscheidbar ist, muss $A \cap B$ dann auch unentscheidbar sein?

c) Ist die folgende Schlussfolgerung korrekt? Weil das Circuit Value Problem P-hart bzgl. logspace-many-one-Reduktionen ist, aber mittels eines Linear-Platz-beschränkten Algorithmus gelöst werden kann, gibt es für jedes Problem in P einen LBA, der das Problem entscheidet.

d) Sei $f\colon \Sigma^* \to \Gamma^*$ eine Funktion mit $f(x) \in$ SELF-ACCEPT genau dann, wenn $x \in$ UNIVERSALITY für jedes $x \in \Sigma^*$. Muss f unberechenbar sein?

Vorschlag:

a) Nein, denn die Menge aller Sprachen ist bekannterweise überabzählbar und die Differenz, die Menge entscheidbarer Sprachen, ist bekannterweise abzählbar. Anderenfalls wäre die Menge aller Sprachen als disjunkte Vereinigung zweier abzählbarer Sprachen auch abzählbar.

b) Nein, nehme dazu $A = \varnothing$ und $B =$ ACCEPT. Der Schnitt $A \cup B = \varnothing$ ist entscheidbar.

c) Nein, nehme dazu den CYK-Algorithmus, oder den Kleene-Stern-Automaten aus dem letzten Übungsblatt. Der bekannte Algorithmus hat eine quadratische Platzkomplexität. Die Reduktion auf das CVP erzeugt einen Schaltkreis mit mindestens quadratischer Größe. Der kombinierte Algorithmus mit CVP hat nun mindestens quadratische Platzkomplexität.

d) Ja, denn SELF-ACCEPT ist semientscheidbar und UNIVERSALITY ist es nicht. Falls f berechenbar wäre, ließe sich aus einem Berechner für f und einem Erkenner für SELF-ACCEPT ein Erkenner für UNIVERSALITY konstruieren, was zum Widerspruch führt.

7. Berechenbarkeit

10 Punkte

Es sei $\Sigma = \{0, 1\}$. Betrachten Sie die Funktion *shallowWords* : $\Sigma^* \times \mathbb{N} \to (\mathbb{N} \cup \{\infty\})$, die für die Kodierung $w \in \Sigma^*$ einer deterministischen Turingmaschine M_w und eine Schranke n, die Anzahl der Wörter ausgibt, die von M_w in höchstens n Schritten akzeptiert werden. Dabei soll ∞ ausgegeben werden, falls es unendlich viele solcher Eingaben gibt.

Beweisen Sie, dass die Funktion *shallowWords* berechenbar ist.

Geben Sie hierzu einen Algorithmus (als Pseudo-Code) an.

Hinweis: Die Kodierung der Zahlen in der Ein- und Ausgabe der Funktion (z.B. unär oder binär) ist für die Bearbeitung der Aufgabe unerheblich.

Vorschlag:

Idee: Es gibt genau dann unendlich viele Wörter, die in n Schritten akzeptiert werden, wenn es ein solches Wort mit genau n Buchstaben gibt. Anderenfalls zählt man einfach auf.

```
Input: TM M_w, n ∈ ℕ
for x ∈ Σ^n do
    if M_w accepts x in at most n steps then
        return ∞
    end if
end for
num ← 0
for x ∈ Σ^≤n do
    if M_w accepts x in at most n steps then
        num++
    end if
end for
return num
```

Church-Turing-These: Es gibt eine TM M_{sW}, die den Algorithmus implementiert.

Seien $w \in \Sigma^*$ und $n \in \mathbb{N}$. Falls $\mathit{shallowWords}(w, n) = \infty$ ist, gibt es unendlich viele Wörter, die von M_w nach höchstens n Schritten akzeptiert werden. Insbesondere muss es Wörter der Länge $\geq n$ geben. Da M_w keine Möglichkeit hat, Buchstaben der Stelle $n + 1$ zu lesen, muss es auch ein Wort der Länge n geben, dass nach n Schritten akzeptiert wird. M_{sW} wird dann dieses Wort finden und ∞ zurückgeben.
Falls $\mathit{shallowWords}(w, n) = k \in \mathbb{N}$, gibt es nur k Wörter, die nach n Schritten akzeptiert werden. Das Längste davon und damit alle diese Wörter müssen kürzer als n sein, da M_w sonst die unendlich vielen Verlängerungen auch akzeptiert hätte. Also wird M_{sW} in der ersten Schleife nichts finden, die zweite Schleife erreichen, num = k aufzählen und zurückgeben.

Da M_{sW} *shallowWords* berechnet, ist *shallowWords* berechenbar.