Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: ______________ Kennwort: ______________ Arbeitsplatz-Nr.: ________	**Herbst 2003**	**66112**

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Automatentheorie, Komplexität, Algorith.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 6

Bitte wenden!

Thema Nr. 1

Sämtliche Teilaufgaben sind zu bearbeiten!

Für das Alphabet $\Sigma = \{0,1\}$ und $n,m \in \mathbb{N}_0$ sei $L(n,m) \subseteq \Sigma^*$ wie folgt definiert:

$$L(n,m) = \{w \in \Sigma^* \mid w \text{ enthält genau } n \text{ Zeichen 0 und } m \text{ Zeichen 1}\}.$$

1. Geben Sie alle Elemente von $L(3,2)$ an!

2. Beweisen Sie: Für alle $n,m \in \mathbb{N}_0$ mit $n > 0$ und $m > 0$ gilt:

$$L(n,m) = \{0w \mid w \in L(n-1,m)\} \cup \{1w \mid w \in L(n,m-1)\}.$$

3. In dieser Teilaufgabe soll ein Algorithmus entwickelt werden, der zu gegebenen $n,m \in \mathbb{N}_0$ die Menge $L(n,m)$ bestimmt. Verwenden Sie dazu eine gängige höhere Programmiersprache oder einen entsprechenden Pseudocode. Nehmen Sie dabei an, dass die gewählte Sprache die Datentypen **char** (für Zeichen), **nat** (für $\mathbb{N}_0$) sowie für jeden Datentyp **d** einen Datentyp **sequ d** der Listen (Sequenzen) von Elementen vom Typ d zur Verfügung stellt. Für Listen seien die Konstante *empty* (leere Liste) und folgende Funktionen verfügbar:

isempty(x)	(Test, ob die Liste *x* leer ist),
first(x)	(erstes Element der Liste *x*),
rest(x)	(Liste *x* ohne ihr erstes Element),
prefix (a, x)	(Anfügen des Elements *a* als neues erstes Element an die Liste *x*).

Zeichenketten seien als Listen von Zeichen und Mengen von Zeichenketten als Listen von Zeichenketten repräsentiert. (Dabei sollen die Elemente einer Menge in der Liste nicht mehrfach vorkommen.)

a) Geben Sie einen Algorithmus an, der für ein Zeichen z und $n \in \mathbb{N}_0$ die Zeichenkette z^k (d. i. $zz... z$ mit k Zeichen z) als Ergebnis hat!

b) Geben Sie einen Algorithmus an, der für **zwei** Mengen von Zeichenketten die Vereinigung dieser Mengen als Ergebnis hat!

c) Geben Sie einen Algorithmus an, der für eine Menge M von Zeichenketten und ein Zeichen z die Menge $\{zw \mid w \in M\}$ als Ergebnis hat!

d) Geben Sie (unter Verwendung von Teilaufgabe 2 und der Algorithmen unter a), b) und c)) einen Algorithmus an, der für beliebige $n,m \in \mathbb{N}_0$ die Menge $L(n, m)$ als Ergebnis hat!

Geben Sie ausführliche Erläuterungen zu Ihren Lösungen an!

Fortsetzung nächste Seite!

4. Gegeben sei die Grammatik $G=(V,\Sigma,P,S)$ mit $V=\{S,A,B\}$ und der Menge P der Produktionen

$$S \to 0 \mid 0A \mid 1S \mid 1B$$
$$A \to 1 \mid 1A$$
$$B \to 1S \mid 1B$$

L(G) sei die von *G* erzeugte Sprache.

a) Beweisen oder widerlegen Sie: $11011 \in L(G)$.

b) Beweisen oder widerlegen Sie: $11010 \in L(G)$.

c) Konstruieren Sie direkt aus *G* zunächst einen nicht-deterministischen endlichen Automaten, der die Sprache *L(G)* akzeptiert, und daraus einen deterministischen endlichen Automaten, der ebenfalls *L(G)* akzeptiert!

d) Beweisen Sie: $L(G)=\bigcup_{m\in\mathbb{N}_0} L(1,m)$.

5. Beweisen Sie: Für jedes $m\in\mathbb{N}_0$ ist die Sprache $L_m=\bigcup_{n\in\mathbb{N}_0} L(n,m)$ regulär (vom Typ 3).

6. Beweisen Sie, dass für beliebige $n,m,\in\mathbb{N}_0$ mit $n\le m$ gilt:

a) Zu jedem $w=0w'\in L(n,m)$ gibt es $k\in\mathbb{N}_0, v\in L(k,k)$ und $v'\in\Sigma^*$ mit $w=0v1v'$.

b) Zu jedem $w=0w'\in L(n,n)$ gibt es $k,j\in\mathbb{N}_0, v\in L(k,k)$ und $v'\in L(j,j)$ mit $w=0v1v'$.

7. Die Sprache *L* sei definiert als $L=\bigcup_{n\in\mathbb{N}_0} L(n,n)$.

a) Beweisen Sie: *L* ist nicht regulär.

b) Geben Sie eine kontextfreie (d.h. Typ-2-) Grammatik an, die *L* erzeugt.

8. Geben Sie eine deterministische Turingmaschine *T* an mit folgenden Eigenschaften:

a) *T* berechnet die Funktion
$f:\mathbb{N}_0\times\Sigma^*\to\{0,1\}$,
$$f(n,w)=\begin{cases}1, & \text{falls } w\in L(n,1)\\ 0, & \text{falls } w\notin L(n,1)\end{cases}$$

in folgendem Sinne:
Angesetzt auf das Wort $1^n\#w$ (mit $n\in\mathbb{N}_0, w\in\Sigma^*$ und Trennzeichen #) hält *T* nach endlicher Zeit in einer Konfiguration an, in der $f(n,w)$ als Ergebnis auf dem Arbeitsfeld steht.
Geben Sie ausführliche Erläuterungen zur Wirkungsweise Ihrer Lösung!

b) Die Anzahl der Rechenschritte von *T* für eine Eingabe der unter a) genannten Art ist $0(n^2)$. Begründen Sie diese Aussage für Ihre Lösung!

Thema Nr. 2

Sämtliche Teilaufgaben sind zu bearbeiten!

Automatentheorie, formale Sprachen, Berechenbarkeit

1. Seien die regulären Ausdrücke $\alpha = (a\,b^2)^*\,a\,b$ und $\beta = a\,b\,(\,b\,a\,b)^*$ gegeben.

   a) Zeigen Sie die Äquivalenz der beiden regulären Ausdrücke, also $L(\alpha) = L(\beta)$.

   b) Geben Sie eine $L(\alpha)$ erzeugende Grammatik an.

   c) Geben Sie einen deterministischen erkennenden Automaten an, der $L(\beta)$ akzeptiert.

2. Sei L die Sprache aller Wörter über $\{a,b\}$ die „aba" als Teilwort enthalten. Konstruieren Sie einen deterministischen erkennenden Automaten für L!

3. Konstruieren Sie eine Turing-Maschine, die die Präfixrelation *präfix* auf Σ^* mit $\Sigma = \{0,1\}$ entscheidet, d. h. die charakteristische Funktion $\text{char}_{\textit{präfix}}$ berechnet (x *präfix* y $:\leftrightarrow$ x ist Anfangsstück von y).

4. Zeigen Sie die Korrektheit der (aussagenlogischen) Regel
   „Aus $A \rightarrow \neg B$ kann man auf $B \rightarrow \neg A$ schließen"!

5. a) Zeigen Sie mit Hilfe vollständiger Induktion, dass das folgende Programm bzgl. der Vorbedingung $x \geq 0$ und der Nachbedingung (drei_hoch x) = 3^x partiell korrekt ist!

```
(define (drei_hoch x)
   (cond ((= x 0) 1)
         (else (* 3 (drei_hoch (- x 1)))))
) )
```

   b) Zeigen Sie, dass die gegebene Funktion (drei_hoch x) unter der Vorbedingung $x \in \mathbb{N}_0$ terminiert, indem Sie eine geeignete Terminierungsfunktion angeben und begründen Sie, warum die von Ihnen angegebene Funktion das Gewünschte leistet!

6. Gegeben sind die drei folgenden Prozeduren zur Multiplikation zweier natürlicher Zahlen.

```
(define (mult n m)
   (cond ((= n 0) 0)
         (else + n (mult (- n 1) m)))
) )

(define (mult2 n m)
   (cond ((= n 0) 0)
         ((even? n) (* 2 (mult2 (/ n 2) m)))
         (else (+ m (mult2 (- n 1) m)))
) )

(define (mult3 n m)
   (define (mult-iter counter m sum)
      (cond ((= counter 0) sum)
            (else (mult-iter (- counter 1) m (+ m sum)))
   ) )
   (mult-iter n m 0)
)
```

Geben Sie für jede dieser Prozeduren an, welchen Speicherbedarf und welchen Zeitbedarf die jeweils erzeugten Prozesse haben! Formulieren Sie die Ergebnisse in der O(f(n))-Notation!

7. Schreiben Sie in einer funktionalen Programmiersprache folgende Prozeduren:

a) Die Prozedur `mod(x, y)` berechnet den Rest bei der Division natürlicher Zahlen. Falls die Division aufgeht, wird der Divisor ausgegeben.
Die Parameter genügen der Vorbedingung $x > 0$ und $y > 0$.

`mod : (zahl, zahl) -> zahl`
Beispiel: mod(34, 7) = 6 wegen 34 : 7 = 4 Rest 6
mod(12, 4) = 4 wegen 12 : 4 = 3 Rest 0

b) Die Prozedur `len` berechnet die Anzahl der Elemente in einer Liste.

`len : (liste) -> zahl`
Beispiel: len([2; 1; 1; 1; 4]) = 5

c) In einer Liste werden Kettenbrüche implementiert. Sie haben folgende Eigenschaften:

- Sie sind periodisch. Die Periode beginnt stets beim zweiten Eintrag der Liste und reicht bis zu deren Ende: $[2;\overline{1;5;4}] = 2;1;5;4;1;5;4;1;5;4;1;\ldots$
- Falls n ≤ len(liste) ist die n. Stelle des Kettenbruchs das n. Element in der Liste.
Beispiel: 3. Stelle ([2; 1; 5; 4]) = 5
- Falls n > len(liste) wird die Zählung der Stellen stets am Beginn der Periode fortgesetzt.
Beispiel: 5. Stelle ([2; 1; 5; 4]) = 1;
7. Stelle ([2; 1; 5; 4]) = 4;
8. Stelle ([2; 1; 5; 4]) = 1;
usw.

Die Prozedur `n-te-stelle` berechnet die n. Stelle des Kettenbruchs.

`n-te-stelle (liste, zahl) -> zahl`
Beispiel: n-te-stelle([2; 1; 5; 4], 3) = 5

Fortsetzung nächste Seite!

d) Der Wert eines Kettenbruchs lässt sich näherungsweise durch die Einbeziehung seiner ersten Stellen berechnen. Sein Wert wird umso genauer, je mehr Stellen für die Berechnung verwendet werden.
Sei s(n) die n. Stelle des Kettenbruchs und k(n) der Näherungswert des Kettenbruchs unter Einbeziehung der ersten n Stellen des Kettenbruchs, dann gilt folgende rekursive Definition:

$$A(n) = \begin{cases} 0 & \text{falls } n = -1 \\ 1 & \text{falls } n = 0 \\ s(n) * A(n-1) + A(n-2) & \text{falls } n \geqslant 1 \end{cases}$$

$$B(n) = \begin{cases} 1 & \text{falls } n = -1 \\ 0 & \text{falls } n = 0 \\ s(n) * B(n-1) + B(n-2) & \text{falls } n \geqslant 1 \end{cases}$$

$$k(n) = A(n)/B(n)$$

Schreiben Sie eine Prozedur, die den Näherungswert k(n) nach obigen Formeln berechnet!

8. a) Implementieren Sie in einer objektorientierten Sprache einen binären Suchbaum für ganze Zahlen! Dazu gehören Methoden zum Setzen und Ausgeben der Attribute zahl, linker_teilbaum und rechter_teilbaum.

b) Schreiben Sie die Methode `fuege_ein (...)`, die eine Zahl in den Baum einfügt!

c) Schreiben Sie die Methode `post_order ()`, die die Zahlen in der Reihenfolge postorder ausgibt!

d) Ergänzen Sie Ihr Programm um die rekursiv implementierte Methode `summe (...)`, die die Summe der Zahlen des Unterbaums, dessen Wurzel der Knoten x ist, zurückgibt! Falls der Unterbaum leer ist, ist der Rückgabewert 0!
`int summe (Knoten x) {...}`

e) Schreiben Sie ein Folge von Anweisungen, die einen Baum mit Namen BinBaum erzeugt und nacheinander die Zahlen 5 und 7 einfügt!
In den binären Suchbaum werden noch die Zahlen 4, 11, 6 und 2 eingefügt. Zeichnen Sie den Baum, den Sie danach erhalten haben, und schreiben Sie die eingefügten Zahlen in der Reihenfolge der Traversierungsmöglichkeit postorder auf!