Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer
Kennzahl: __________	**Herbst 1997**	**66110**
Kennwort: __________		
Arbeitsplatz-Nr.: _____		

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

\- Prüfungsaufgaben -

Fach: Informatik (vertieft studiert)

Einzelprüfung: Automatentheorie, Algorithm. Sprachen

Anzahl der gestellten Themen (Aufgaben): 1

Anzahl der Druckseiten dieser Vorlage: 4

Bitte wenden!

Sämtliche Teilaufgaben sind zu bearbeiten!

1. Aufgabe (Rechenstrukturen und Termersetzungssysteme)

Gegeben sei eine Rechenstruktur NAT mit der Signatur:

$$\Sigma = (S_{\text{NAT}}, F_{\text{NAT}}) \text{ mit } S_{\text{NAT}} = \{\textbf{bool}, \textbf{nat}\} \text{ und } F_{\text{NAT}} = \{\texttt{zero}, \texttt{succ}, \texttt{pred}\}$$

Den beiden Sorten seien die folgenden Trägermengen zugeordnet:

$$\textbf{bool}^{\text{NAT}} = B^{\perp} := B \cup \{\perp\}, \quad \textbf{nat}^{\text{NAT}} = N^{\perp} := N \cup \{\perp\}$$

Dabei ist $N = \{0, 1, 2, \ldots\}$ die Menge der natürlichen Zahlen und $B = \{\text{TRUE}, \text{FALSE}\}$ die Menge der Booleschen Werte. Das Symbol $\perp$ steht für „undefiniert".

Die Spezifikationen der den Funktionssymbolen zero, succ und pred zugeordneten strikten Funktionen lauten:

$$\begin{array}{llll} \texttt{zero}^{\text{NAT}}: & \rightarrow N^{\perp}; & \texttt{zero}^{\text{NAT}} & = 0; \\ \texttt{succ}^{\text{NAT}}: & N^{\perp} \rightarrow N^{\perp}; & \texttt{succ}^{\text{NAT}}(x) & = x+1; \\ \texttt{pred}^{\text{NAT}}: & N^{\perp} \rightarrow N^{\perp}; & \texttt{pred}^{\text{NAT}}(x) & = x-1; \text{ falls } x \geq 1 \text{ und } \texttt{pred}^{\text{NAT}}(0) = \perp \end{array}$$

Im folgenden nehmen wir an, daß jede natürliche Zahl durch einen Grundterm dargestellt ist, der nur aus den Konstruktoren zero und succ besteht.

a) Geben Sie ein Termersetzungssystem an, das die Funktion pred auf die Konstruktoren zero und succ zurückführt!

b) Nun sollen die strikten Funktionen add und sub mit den Spezifikationen

$$\begin{array}{llll} \texttt{add}^{\text{NAT}}: & N^{\perp} \times N^{\perp} \rightarrow N^{\perp}; & \texttt{add}^{\text{NAT}}(x,y) & = x+y \\ \texttt{sub}^{\text{NAT}}: & N^{\perp} \times N^{\perp} \rightarrow N^{\perp}; & \texttt{sub}^{\text{NAT}}(x,y) & = x-y \text{ falls } x \geq y \\ & & \texttt{sub}^{\text{NAT}}(x,y) & = \perp \text{ falls } x < y \end{array}$$

zu F^{NAT} hinzugefügt werden $(x, y \neq \perp)$. Geben Sie Termersetzungssysteme an, die add bzw. sub durch zero, succ und pred darstellen.

c) Stellen Sie analog zur Teilaufgabe b) je ein Termersetzungssystem für die folgenden, ebenfalls strikten Funktionen auf (mit $x, y \neq \perp$):

$$\begin{array}{llll} \texttt{mult}^{\text{NAT}}: & N^{\perp} \times N^{\perp} \rightarrow N^{\perp}; & \texttt{mult}^{\text{NAT}}(x,y) & = x \cdot y \\ \texttt{div}^{\text{NAT}}: & N^{\perp} \times N^{\perp} \rightarrow N^{\perp}; & \texttt{div}^{\text{NAT}}(x,y) & = x \div y \text{ falls } y > 0 \\ & & \texttt{div}^{\text{NAT}}(x,0) & = \perp \end{array}$$

Sie dürfen dabei auch die Funktionen add, sub aus b) verwenden.

Wir betrachten nun den seit dem Altertum bekannten Euklidischen Algorithmus in der Notation einer imperativen Programmiersprache (a, b seien von der Sorte **nat**)

```
(*)   while a ≠ b do
          while a < b do b := b − a od
          (a, b) := (b, a)
          od
```

Fortsetzung nächste Seite!

d) Stellen Sie einen exemplarischen Ablauf des Algorithmus für die Variablenbelegung $a = 32, b = 18$ als Folge von Zuständen des Variablenraumes dar.

e) Der Algorithmus soll nun mit Hilfe der Zusicherungsmethode nach Floyd und Hoare verifiziert werden. Geben Sie für beide `while`–Schleifen geeignete Invarianten an, und beweisen Sie damit die Korrektheit des Algorithmus.
Hinweis: Gehen Sie von der Zusicherung $\{P \wedge a = mg \wedge b = ng \wedge n, m$ teilerfremd$\}$ mit $P = a > 0 \wedge b > 0$ vor Beginn der ersten `while`–Anweisung aus.

f) Formulieren Sie den Euklidischen Algorithmus rekursiv in einer beliebigen Notation.

g) Geben Sie ein Termersetzungssystem auf der Rechenstruktur NAT für den Euklidischen Algorithmus an. Sie dürfen dabei alle in den Teilaufgaben a) mit c) auf NAT eingeführten Funktionen und Hilfsfunktionen verwenden.

h) Beweisen Sie die Terminierung der beiden `while`–Schleifen in unserer ursprünglichen Formulierung (*) des Euklidischen Algorithmus.

2. Aufgabe (Rekursive Rechenstrukturen)

Gegeben sei die rekursive Rechenstruktur

$Liste = Leer \mid (Float, Liste)$

der Listen über reellen Gleitpunktzahlen mit den Funktionen

- $head: \quad Liste \rightarrow Float$
- $tail: \quad Liste \rightarrow Liste$
- $mklist: \quad Float \times Liste \rightarrow Liste$

Für diese Funktionen gelten folgende Spezifikationen:

- $head(x)$ und $tail(x)$ sind partiell nur für nicht–leere Listen definiert, und
  $head(x)$ ist das erste Element der Liste x,
  $tail(x)$ ist die um das erste Element verkürzte Liste x.
- $mklist(r, x)$ ist total und liefert die Liste, die durch Voransetzen des Elements r vor die Liste x entsteht.

Die Rechenstruktur *Liste* soll nun um die folgenden Funktionen erweitert werden:

- $length : Liste \rightarrow Int$
  mit der Spezifikation: $length(x)$ ist total und liefert die Anzahl der Elemente in der Liste x.

Fortsetzung nächste Seite!

- *proj* : $Int \times Liste \to Float$
  mit der Spezifikation: $proj(n, x)$ ist partiell nur für nicht-leere Listen x sowie für n mit $1 \leq n \leq length(x)$ definiert und liefert das n-te Element der Liste x.

- *part* : $Int \times Int \times Liste \to Liste$
  mit der Spezifikation: $part(m, n, x)$ ist partiell nur für nicht-leere Listen x sowie für m und n mit $1 \leq m \leq n \leq length(x)$ definiert und liefert die Teilliste von x vom m-ten bis zum n-ten Element einschließlich.

1. Programmieren Sie die 3 Funktionen *length*, *proj* und *part* rekursiv unter Abstützung auf die primitiven Rechenstrukturen *Int* und *Bool* sowie auf die oben angegebene Rechenstruktur *Liste*.

2. Beweisen Sie für das von Ihnen für die Funktion *part* angegebene Programm,

   (a) daß es für alle zulässigen Parameter terminiert und

   (b) daß es die Spezifikation für die Funktion *part* erfüllt.

3. Aufgabe (Endliche Automaten und reguläre Mengen)

Gegeben sei das Alphabet $\mathcal{A} = (A, B, C)$. In $\mathcal{A}^*$ zeichnen wir die Teilmenge $\mathbf{T}$ der Wörter aus, die weder ACC noch BCC als Teilzeichenreihe enthalten. Dabei ist $x \in \mathcal{A}^*$ genau dann eine Teilzeichenreihe von $y \in \mathcal{A}^*$, wenn es ein $y' \in \mathcal{A}^*$ und ein $y'' \in \mathcal{A}^*$ gibt, so daß $y = y'xy''$ ist.

1. Konstruieren Sie den (bis auf die Bezeichnungen der Zustände eindeutigen) minimalen deterministischen endlichen Automaten $\mathbf{A} = (S, I, \delta, s_0, F)$ mit der Zustandsmenge S, dem Eingabealphabet $I = \mathcal{A}$, der Zustandsübergangsfunktion $\delta : S \times I \to S$, dem Anfangszustand s_0 und der Endzustandsmenge F, der genau $\mathbf{T}$ akzeptiert! Stellen Sie hierzu den Automaten $\mathbf{A}$ durch seinen Zustandsübergangsgraphen dar.

2. Beweisen Sie, daß der von Ihnen in der Antwort zu 1. angegebene Automat $\mathbf{A}$

   (a) genau $\mathbf{T}$ akzeptiert und

   (b) minimal ist.

3. Stellen Sie $\mathbf{T}$ als eine reguläre Menge über $\mathcal{A}$ dar.