INSTITUT FÜR INFORMATIK
UND PRAKTISCHE MATHEMATIK
DER UNIVERSITÄT KIEL

KIEL, 13.01.2000

ÜBUNGEN ZU INFORMATIK I
WS 1999/2000

SERIE 9

Aufgabe T11 *(nicht für Physiker)* (6 Punkte)

Transformieren Sie den in der Vorlesung eingeführten Y-Kombinator

$$Y \ =_s \ (P \ \ P) \qquad , \qquad P \ =_s \ \lambda u.\lambda v.(v \ \ ((u \ \ u) \ \ v))$$

in SKI-Kombinator-Schreibweise.
Zur Erinnerung

- $S \ =_s \ \lambda x.\lambda y.\lambda z.((x \ \ z) \ \ (y \ \ z))$, wobei $S \ f \ g \ a \rightsquigarrow ((f \ \ a) \ \ (g \ \ a))$,
- $K \ =_s \ \lambda x.\lambda y.x$, wobei $K \ c \ a \rightsquigarrow c$,
- $I \ =_s \ \lambda x.x$, wobei $I \ a \rightsquigarrow a$.

Verwenden Sie das durch die folgenden Regeln definierte systematische Transformationsverfahren

- $\lambda x.(e_1 \ \ e_2) \quad \rightarrow \quad S \ \lambda x.e_1 \ \lambda x.e_2$,
- $\lambda x.c \quad \rightarrow \quad K \ c$,
- $\lambda x.x \quad \rightarrow \quad I$.

Beginnen Sie mit der innersten Abstraktion und arbeiten Sie sich systematisch nach außen vor. Üben Sie das Verfahren zunächst mit dem einfacheren λ-Term

$$\lambda x.((x \ \ v) \ \ x) \quad .$$

Aufgabe P13 (14 Punkte)

Erweitern Sie den im Rahmen der Aufgaben P11 und P12 erstellten Interpretierer für Terme des reinen λ-Kalküls zu einem Interpretierer für Terme eines angewandten λ-Kalküls, der den reinen λ-Kalkül u. a. um arithmetische Operationen, sowie Operationen auf Listen ergänzt. Die Menge der Konstruktorterme wird dazu wie folgt erweitert:

$$\begin{array}{lll} \textit{term} & \Rightarrow & \ldots \\ & | & \texttt{Num \{ } \textit{zahl} \texttt{ \}} \quad | \quad \texttt{Bool \{ } b \texttt{ \}} \quad | \quad \texttt{Op \{ } \textit{fun} \texttt{ \}} \\ & | & \texttt{List \{ } [\ \textit{term}\ ]^* \texttt{ \}} \quad | \quad \texttt{Cond \{ } \textit{term} \texttt{ , } \textit{term} \texttt{ \}} \\ & | & \texttt{Clos \{ } \textit{fun} \texttt{ , } \textit{term} \texttt{ \}} \quad | \quad \texttt{Alpha \{ } \textit{var} \texttt{ , } \textit{term} \texttt{ \}} \end{array}$$

wobei gilt:

$$\begin{array}{rcl} \textit{zahl} & \in & \mathbb{Z} \\ b & \in & \{\texttt{true}, \texttt{false}\} \\ \textit{fun} & \in & \{\texttt{'+'}, \texttt{'-'}, \texttt{'*'}, \texttt{'/'}, \texttt{'head'}, \texttt{'tail'}, \texttt{'cons'}, \texttt{'empty'}, \\ & & \texttt{'le'}, \texttt{'lt'}, \texttt{'ge'}, \texttt{'gt'}, \texttt{'eq'}, \texttt{'ne'}, \texttt{'and'}, \texttt{'not'}, \texttt{'or'}\} \end{array}$$

Dabei repräsentiert

- `Num` { *zahl* } eine ganze Zahl,
- `Op` { *fun* } eine primitive Operation,
- `List` { [*term*]* } ein n-stellige Liste,
- `Clos` { *fun* , *term* } eine einstellige Closure als Resultat der Applikation einer zweistelligen primitiven Operation auf ein Argument,
- `Bool` { *b* } einen Wahrheitswert,
- `Cond` { *then* , *else* } eine einstellige Funktion, die
  - dem Term *then* liefert, falls sie auf den Wahrheitswert *true* angewendet wird,
  - den Term *else* liefert, falls sie auf *false* angewendet wird,
  - andernfalls nicht reduziert wird.
- `Alpha` { *var* , *term* } den primitiven Rekursionsoperator α aus der Vorlesung:

$$\texttt{Alpha} \{ \textit{var} , \textit{term} \} \quad \rightsquigarrow \quad \textit{term} \; [\; \textit{var} \leftarrow \texttt{Alpha} \{ \textit{var} , \textit{term} \} \;] \quad .$$

Werten Sie mit Hilfe Ihres Interpreters die folgenden Terme des angewandten λ-Kalküls beispielhaft aus:

- Fakultät von 6 (bereits in Konstruktur-Darstellung):

```
Ap{ Alpha{ 'fac',
           Lam{ 'n',
                Ap{ Cond{ Num{1},
                          Ap{ Ap{ Op{'*'}, Var{'n'}},
                              Ap{ Var{'fac'},
                                  Ap{ Ap{ Op{'-'}, Var{'n'}},
                                      Num{1}
                                    }
                                }
                          }
                        },
                    Ap{ Ap{ Op{'le'}, Var{'n'}},
                        Num{1}
                      }
                  }
              }
         },
    Num{6}
  }
```

- Summation einer Liste (muß von Ihnen noch in die Konstruktur-Darstellung überführt werden):

```
def
  Sum [ L ] = if empty( L )
              then 0
              else ( ( 1 | L ) + Sum [ lcut( 1, L ) ] )
in Sum [ < 1, ~2, 3, ~4, 5 > ]
```

Abgabe: Bis Do., 27.01.2000, 11 Uhr s.t. im Schrein (Institut für Informatik, Haus I, Erdgeschoß).