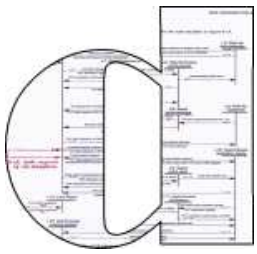
DEPENDABLE SYSTEMS AND SOFTWARE

Fachrichtung 6.2 — Informatik
Prof. Dr.-Ing. Holger Hermanns
Dipl.-Inform. Lijun Zhang



Übungsblatt 3 (Programmierung I)

Lesen Sie im Skript: Kapitel 3.1-3.5

Aufgabe 3.1: (Fac ohne Rekursion)
Deklarieren Sie eine Prozedur $prod : (int \to int) \to int \to int$, die für $n \geq 0$ die Gleichung

$$\text{prod } f\ n = 1 \cdot (f\ 1) \cdot \ \ldots\ \cdot (f\ n)$$

erfüllt. Deklarieren Sie danach mit Hilfe von *prod* eine Prozedur $fac : int \to int$, die für $n \geq 0$ die Fakultät $n!$ berechnet. Ihre Prozedur *fac* soll nicht rekursiv sein.

Aufgabe 3.2: (Sum')
Deklarieren Sie eine Prozedur $sum' : (int \to int) \to int \to int \to int$ die für $k \geq 0$ die Gleichung

$$sum'\ f\ m\ k = 0 + f(m+1) + \ldots + f(m+k)$$

erfüllt. Die Prozedur sum' soll mit Hilfe der Prozedur *sum* formuliert werden und nicht rekursiv sein.

Aufgabe 3.3: (Ganzzahlige Division)

Deklarieren Sie mit *first* (siehe Kapitel 3) eine Prozedur $divi : int \to int \to int$, die zu $x \geq 0$ und $m > 0$ dasselbe Ergebnis liefert wie Division mit dem Operator *div*. Hinweis: Für $x \geq 0$ und $m > 0$ soll *divi* die größte ganze Zahl k mit $k \cdot m \leq x$ liefern.

Aufgabe 3.4: (Typen)
Geben Sie geschlossene Ausdrücke an, die die folgenden Typen haben:

(a) $(int \to bool) \to (bool \to real) \to int \to real$

(b) $(int * int \to bool) \to int \to bool$

(c) $(int \to bool \to real) \to int \to bool \to real$

(d) $(int \to real) \to (bool \to int) \to int * bool \to real * int$

Ihre Ausdrücke sollen nur mit Abstraktionen, Prozeduranwendungen, Tupeln und Bezeichnern gebildet sein. Sie sollen also keine Konstanten oder Operatoren verwenden.

Aufgabe 3.5: (Polymorphe Prozeduren)
Deklarieren Sie polymorphe Prozeduren wie folgt:

$$\begin{aligned}
&\alpha\ id : \alpha \to \alpha \\
&\alpha\ \beta\ \gamma\ com : (\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma \\
&\alpha\ \beta\ \gamma\ cas : (\alpha * \beta \to \gamma) \to \alpha \to \beta \to \gamma \\
&\alpha\ \beta\ \gamma\ car : (\alpha \to \beta \to \gamma) \to \alpha * \beta \to \gamma
\end{aligned}$$

Ihre Prozeduren sollen nicht rekursiv sein. Machen Sie sich klar, dass die angegebenen Typschemata das Verhalten der Prozeduren eindeutig festlegen.

Aufgabe 3.6: (Polymorphe Prozeduren)
Deklarieren Sie polymorphe Prozeduren, die die folgen Typschemata besitzen:

(a) $\forall\ \alpha\ \beta$. $(\alpha * \alpha \to \beta) \to \alpha \to \beta$

(b) $\forall\ \alpha\ \beta\ \gamma$. $\alpha * \beta * \gamma \to \beta$

(c) $\forall\ \alpha\ \beta\ \gamma$. $\alpha * \beta * \gamma \to \alpha * \gamma$

(d) $\forall\ \alpha\ \beta\ \gamma$. $(\alpha \to \beta \to \gamma) \to \alpha \to \beta \to \gamma$

Aufgabe 3.7: (Pif)
Dieter Schlau ist ganz begeistert von polymorphen Prozeduren. Er deklariert die Prozedur

```
fun 'a pif (x:bool, y:'a, z:'a) = if x then y else z
```

und behauptet, dass man statt eines Konditionals stets die Prozedur pif verwenden kann:

$$if\ e_1\ then\ e_2\ else\ e_3 \quad \rightsquigarrow \quad pif(e_1, e_2, e_3)$$

Was übersieht Dieter? Denken Sie an rekursive Prozeduren. Und daran, dass der Ausdruck, der das Argument einer Prozeduranwendung beschreibt, vor dem Aufruf der Prozedur ausgeführt wird.

Aufgabe 3.8: (Typschema)
Geben Sie ein Typschema für f an, sodass der Ausdruck $(f\ 1,\ f\ 1.0)$ wohlgetypt ist.

Aufgabe 3.9: (Typen)
Warum gibt es keinen Typen t, sodass der Ausdruck $fn\ f : t \Rightarrow (f\ 1, f\ 1.0)$ wohlgetypt ist?

Aufgabe 3.10: (Let und Abstraktionen)
Dieter Schlau ist begeistert. Scheinbar kann man Let-Ausdrücke mit val-Deklarationen auf Abstraktion und Applikation zurückführen:

$$let\ val\ x = e\ in\ e'\ end\ \sim\ (fn\ x : t \Rightarrow e')\ e$$

Auf der Heimfahrt von der Uni kommen ihm jedoch Zweifel an seiner Entdeckung, da ihm plötzlich einfällt, dass Argumentvariablen nicht polymorph getypt werden können. Können Sie ein Beispiel angeben, das zeigt, dass Dieters Zweifel berechtigt sind? Hilfe: Sie sollen einen semantisch zulässigen Let-Ausdruck angeben, dessen Übersetzung gemäß des obigen Schemas scheitert, da Sie keinen passenden Typ t für die Argumentvariable x finden können.

Aufgabe 3.11: (Monomorph getypte Bezeichner)
Geben Sie Deklarationen an, die monomorph getypte Bezeichner wie folgt deklarieren:

(a) : $int * unit * bool$
(b) : $unit * (int * unit) * (real * unit)$
(c) : $int \to int$
(d) : $int * bool \to int$
(e) : $int \to real$
(f) : $int \to real \to real$
(g) : $(int \to int) \to bool$

Verzichten Sie dabei auf explizite Typangaben und verwenden Sie keine Operator- und Prozeduranwendungen.

Aufgabe 3.12: (Typrekonstruktion und Projektionen)
Im Zusammenhang mit fehlenden Typangaben kann die Verwendung von Projektionen problematisch sein. Beispielsweise kann Typrekonstruktion das Programm $fun\ f\ x = \#1x$ nicht typisieren. Können Sie erklären, warum das so ist?

Aufgabe 3.13: (Ideq)
Deklarieren Sie eine Identittsprozedur $''a\ ideq\ :\ ''a\ \to\ ''a$, deren Typschema auf Typen mit Gleichheit eingeschrnkt ist. Verzichten Sie dabei auf explizite Typangaben.