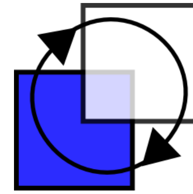
REACTIVE SYSTEMS GROUP

Universität des Saarlandes

Prof. Bernd Finkbeiner, Ph.D.
Markus Rabe, M.Sc.



Programmierung 1 (SS 2010) - 4. Übungsblatt

http://react.cs.uni-saarland.de/prog1/

Lesen Sie im Buch bis zum Ende von Kapitel 3.

Bei Problemen gehen Sie in die Office Hours: Mo-Mi von 13 bis 14 Uhr.

Aufgabe 4.1

Schreiben Sie eine zu der folgenden Prozedur `f` semantisch äquivalente Prozedur ohne dabei das Schlüsselwort `fun` zu verwenden.

```
fun f x = fn y => x*y
```

Aufgabe 4.2 (Baumdarstellung)

a) Gegeben sei eine Prozedur a mit dem Typ $int \to int \to int$. Geben Sie die Baumdarstellung des folgenden Ausdrucks an:

```
a x y + a x (y+2)*5
```

b) Geben Sie die Baumdarstellung der folgenden Typen an.

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

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

Aufgabe 4.3

Geben Sie zur folgenden Abstraktion einen semantisch äquivalenten Ausdruck an, der ohne die Verwendung einer Abstraktion gebildet ist:

```
fn (x:int) => (fn (y:int) => x + y)
```

Hilfe: Verwenden Sie Let-Ausdrücke und Prozedurdeklarationen.

Aufgabe 4.4

Deklarieren Sie eine Prozedur $prod : (int \to int) \to int \to int$, die für $n \geq 0$ die Gleichung

$$prod\ f\ n = 1 \cdot (f\ 1) \cdot\ \ldots\ \cdot (f\ n)$$

erfüllt. (Sie berechnet also das Produkt: $\prod_{i=1}^{n} f(i)$) Deklarieren Sie danach mit Hilfe von $prod$ eine Prozedur $fac : int \to int$, die für $n \geq 0$ die Fakultät $n! = n * (n-1) * \cdots * 2 * 1$ berechnet (beachten Sie, dass $0! = 1$ definiert ist). Ihre Prozedur fac selbst soll nicht rekursiv sein.

Aufgabe 4.5

a) Deklarieren Sie die Prozedur *add*, welche zwei Zahlen addiert, auf zwei Weisen: einmal mit kaskadiertem und einmal kartesischem Argumentenmuster.

b) Schreiben Sie zwei Prozeduren

$$cas : (int * int \to int) \to int \to int \to int$$

$$car : (int \to int \to int) \to int * int \to int$$

sodass *cas* zur kartesischen Darstellung einer zweistelligen Operation die kaskadierte Darstellung und *car* zur kaskadierten Darstellung die kartesische Darstellung liefert. Erproben Sie *cas* und *car* mit Prozeduren, die das Maximum zweier Zahlen liefern:

```
fun maxCas (x:int) (y:int) = if x<y then y else x
fun maxCar (x:int, y:int) = if x<y then y else x
val maxCas' = cas maxCar
val maxCar' = car maxCas
```

Wenn Sie *cas* und *car* richtig geschrieben haben, verhält sich $maxCas'$ genauso wie $maxCas$ und $maxCar'$ genauso wie $maxCar$.

Aufgabe 4.6 (Summen und Iteration)

a) 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* aus dem Buch formuliert werden.

b) Deklarieren Sie mithilfe von *iter* eine Prozedur fac, welche zu $n \geq 0$ die Fakultät $n!$ berechnet.

c) Deklarieren Sie mithilfe von *iter* eine zu *iterup* semantisch äquivalente Prozedur (siehe Buchkapitel 3.13).

d) Deklarieren Sie eine zu sum' semantisch äquivalente Prozedur mithilfe von *iter*. Hinweis: Machen Sie sich zunächst klar, wie man sum' mithilfe von *iterup* oder *iterdn* deklarieren könnte.

Aufgabe 4.7

Geben Sie geschlossene Ausdrücke an, die die folgenden Typen haben:

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

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

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

Aufgabe 4.8

Deklarieren Sie polymorphe Prozeduren, die die folgenden 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 4.9

Deklarieren Sie polymorphe Prozeduren für die folgenden Typschemata:

$$\begin{array}{ll} \forall\alpha. & \alpha \to \alpha \\ \forall\alpha\,\beta\,\gamma & (\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma \\ \forall\alpha\,\beta\,\gamma & (\alpha * \beta \to \gamma) \to \alpha \to \beta \to \gamma \\ \forall\alpha\,\beta\,\gamma & (\alpha \to \beta \to \gamma) \to \alpha * \beta \to \gamma \end{array}$$

Ihre Prozeduren sollen nicht rekursiv sein. Machen Sie sich klar, dass die angegebenen Typschemata das Verhalten der Prozeduren eindeutig festlegen. Sie haben diese Prozeduren bereits gesehen. Wie hießen sie? Ändert sich die Eindeutigkeit, wenn wir die Beschränkung auf nicht rekursive Prozeduren aufheben?

Aufgabe 4.10

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:

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

Was übersieht er? Denken Sie an Divergenz und Ausführungsreihenfolge. Konstruieren Sie ein Gegenbeispiel.

Aufgabe 4.11

Geben Sie die Typschemata an, mit denen die Bezeichner p und q typisiert werden.

```
fun p f (x,y) = f x y
fun q f g x = g (f x)
```

Tipp: Beginnen Sie damit die Typen der Argumente zu bestimmen.

Aufgabe 4.12

Geben Sie Deklarationen an, die monomorph getypte Bezeichner wie folgt deklarieren:

$$\begin{aligned} a &: \ \mathit{int} * \mathit{unit} * \mathit{bool} \\ b &: \ \mathit{unit} * (\mathit{int} * \mathit{unit}) * (\mathit{real} * \mathit{unit}) \\ c &: \ \mathit{int} \to \mathit{int} \\ d &: \ \mathit{int} * \mathit{bool} \to \mathit{int} \\ e &: \ \mathit{int} \to \mathit{real} \end{aligned}$$

Verzichten Sie dabei auf explizite Typangaben und verwenden Sie keine Operator- und Prozeduranwendungen.
Hinweis: Für einige der Deklarationen ist die Verwendung eines Konditionals essentiell. Die Typregel für Konditionale verlangt, dass die Konsequenz und die Alternative den gleichen Typ haben (siehe Abschnitt 2.6). Außerdem ist für einige der Deklarationen die Verwendung von Tupeln und Projektionen erforderlich, um Werte vergessen zu können, die nur zur Steuerung der Typrekonstruktion konstruiert wurden.

Aufgabe 4.13

Im Zusammenhang mit fehlenden Typangaben kann die Verwendung von Projektionen problematisch sein. Beispielsweise kann Typrekonstruktion das Programm $\mathit{fun}\ f\ x = \#1x$ nicht typisieren. Können Sie erklären, warum das so ist?

Aufgabe 4.14

Deklarieren Sie eine Identitätsprozedur $''a\ \mathit{ideq}\ :\ ''a\ \to\ ''a$, deren Typschema auf Typen mit Gleichheit eingeschränkt ist. Verzichten Sie dabei auf explizite Typangaben. Tipp: Schauen Sie sich die Beispiele im Buch in Kapitel 3.7 an.

Aufgabe 4.15

Betrachten Sie den Ausdruck

```
(fn x => (fn y => (fn x => y) x) y) x
```

a) Geben Sie die Baumdarstellung des Ausdrucks an.
b) Markieren Sie die definierenden Bezeichnerauftreten durch Überstreichen.
c) Stellen Sie die lexikalischen Bindungen durch Pfeile dar.
d) Geben Sie alle Bezeichner an, die in dem Ausdruck frei auftreten.
e) Bereinigen Sie den Ausdruck durch Indizieren der gebunden Bezeichnerauftreten.