Einführung in die Informatik 2

Prof. Dr. Seidl, J. Kranz, N. Hartmann, J. Brunner WS 2016/17

Übungsblatt 13 Abgabefrist: 6.2.2017

Aufgabe 13.1 (P) Verdrehungsanhängung

Gegeben seien folgende OCaml-Definitionen:

```
let rec app =
  fun l1 -> fun l2 ->
    match l1 with
    []     -> l2
  | x::xs -> x :: app xs l2

let rec rev =
  fun l ->
    match l with
    []     -> []
  | x::xs -> app (rev xs) [x]

let rec app_rev =
  fun l1 -> fun l2 ->
    match l1 with
    []     -> l2
  | x::xs -> app_rev xs (x::l2)
```

In der Vorlesung ist gezeigt worden, dass, sofern alle Aufrufe terminieren,

$$\texttt{app (app x y) z = app x (app y z)} \tag{1}$$

für alle Listen `x`,`y` und `z` gilt. Zeigen Sie unter Verwendung der Gleichung (1), dass, sofern alle Aufrufe terminieren,

$$\texttt{app (rev l1) l2 = app\_rev l1 l2}$$

für alle Listen `l1` und `l2` gilt.

Lösungsvorschlag 13.1

Der Beweis wird per Induktion über die Länge der Liste `l1` geführt.

Induktionsanfang: Es gilt `l1` = `[]`. Es folgt:

$$\begin{aligned}\texttt{app (rev []) l2} &= \texttt{app [] l2}\\ &= \texttt{l2}\\ &= \texttt{app\_rev [] l2}\end{aligned}$$

Induktionsschluss: Es gilt `l1` = `x::xs`. Es folgt:

```
app (rev (x::xs)) l2
 = app (app (rev xs) [x]) l2
 = app (rev xs) (app [x] l2)                (Identität (1))
 = app (rev xs) (x :: (app [] l2))
 = app (rev xs) (x :: l2)
 = app_rev xs (x :: l2)                     (Induktionsannahme)
 = app_rev (x::xs) l2
```

Aufgabe 13.2 (P) Funktionale Verifikationsfreude

In dieser Aufgabe werden Gleichheiten mittels `fact`, `map` und `comp` gebildet.

1. Es seien folgende, bekannte Definitionen gegeben:

```
let rec fact = fun n ->
  match n with
    0 -> 1
  | n -> n * fact (n-1)

let rec fact_aux = fun x n ->
  match n with
    0 -> x
  | n -> fact_aux (n*x) (n-1)

let fact_iter = fact_aux 1
```

Zeigen Sie, dass unter der Voraussetzung, dass alle Aufrufe terminieren,

`fact_iter n = fact n`

für alle nicht-negativen ganzen Zahlen $\texttt{n} \in \mathbb{N}_0$ gilt.

2. Es seien folgende, bekannte Definitionen gegeben:

```
let comp = fun f g x -> f (g x)

let rec map = fun op l ->
  match l with
    [] -> []
  | x::xs -> op x :: map op xs
```

Zeigen Sie, dass unter der Voraussetzung, dass alle Aufrufe terminieren,

$$\texttt{map (comp f g)} = \texttt{comp (map f) (map g)}$$

für alle `f` und `g` gilt.

Lösungsvorschlag 13.2

1. Per Induktion über `n` wird gezeigt, dass

$$\texttt{fact\_aux x n} = \texttt{x * fact n} \tag{2}$$

   für alle $\texttt{n} \in \mathbb{N}_0$ und alle $\texttt{x} \in \mathbb{Z}$ gilt.

   Induktionsverankerung: Es gilt $\texttt{n} = \texttt{0}$. Es folgt

$$\texttt{fact\_aux x n} = \texttt{fact\_aux x 0} = \texttt{x} = \texttt{x * 1} = \texttt{x * fact 0}.$$

   Induktionsschritt: Es gilt $\texttt{n} > \texttt{0}$. Es folgt:

$$\begin{aligned}
&\texttt{fact_aux x n} \\
&= \texttt{fact_aux (n * x) (n - 1)} && \text{(Def. }\texttt{fact_aux}\text{)} \\
&= \texttt{n * x * fact (n - 1)} && \text{(Induktionsannahme)} \\
&= \texttt{x * (n * fact (n - 1))} && \\
&= \texttt{x * fact n} && \text{(Def. }\texttt{fact}\text{)}
\end{aligned}$$

   Damit ist (2) gezeigt. Mithilfe von (2) folgt schließlich:

$$\texttt{fact\_iter n} = \texttt{fact\_aux 1 n} = \texttt{1 * fact n} = \texttt{fact n}$$

2. Es ist zu zeigen, dass

$$\texttt{map (comp f g) l} = \texttt{comp (map f) (map g) l}$$

   für alle Werte `l` gilt. Dies geschieht per Induktion.

   Induktionsverankerung: Es gilt `l = []`. Es folgt:

$$\begin{aligned}
&\texttt{map (comp f g) l} \\
&= \texttt{map (comp f g) []} && \\
&= \texttt{[]} && \text{(Def. }\texttt{map}\text{)} \\
&= \texttt{map f []} && \text{(Def. }\texttt{map}\text{)} \\
&= \texttt{map f (map g [])} && \text{(Def. }\texttt{map}\text{)} \\
&= \texttt{comp (map f) (map g) []} && \text{(Def. }\texttt{comp}\text{)} \\
&= \texttt{comp (map f) (map g) l} &&
\end{aligned}$$

Induktionsschritt: Es gilt `l = v::vs` und damit folgt:

$$\texttt{map (comp f g) vs} = \texttt{comp (map f) (map g) vs}$$

Es folgt:

$$
\begin{aligned}
&\texttt{map (comp f g) l} \\
&= \texttt{map (comp f g) (v::vs)} \\
&= \texttt{comp f g v :: map (comp f g) vs} && \text{(Def. \texttt{map})} \\
&= \texttt{comp f g v :: comp (map f) (map g) vs} && \text{(Induktionsann.)} \\
&= \texttt{f (g v) :: map f (map g vs)} && \text{(Def. \texttt{comp})} \\
&= \texttt{map f (g v :: map g vs)} && \text{(Def. \texttt{map})} \\
&= \texttt{map f (map g (v :: vs))} && \text{(Def. \texttt{map})} \\
&= \texttt{map f (map g l)} \\
&= \texttt{comp (map f) (map g) l} && \text{(Def. \texttt{comp})}
\end{aligned}
$$

Allgemeine Hinweise zur Hausaufgabenabgabe

Die Hausaufgabenabgabe bezüglich dieses Blattes erfolgt ausschließlich über Moodle. Die Abgabefrist finden Sie ebenfalls in Moodle. Bitte vergewissern Sie sich nach der Abgabe selbstständig, dass alle Dateien erfolgreich auf Moodle eingestellt wurden. Die Abgaben der Theorieaufgaben[1] müssen in Form einer einzigen PDF-Datei erfolgen. Nutzen Sie z.B. ImageMagick[2], um aus eingescannten Bildern ein PDF zu erzeugen. Achten Sie bei eingescannten Bildern darauf, dass diese im PDF-Dokument eine korrekte Orientierung haben. **Abgaben, die sich nicht in beschriebenen Formaten befinden, können nicht korrigiert oder bewertet werden!**

Aufgabe 13.3 (H) Weitere Wirrungen [10 Punkte]

Gegeben sind die Definitionen

```
let rec app = fun x -> fun y -> match x
    with [] -> y
    | x::xs -> x :: app xs y
let rec rev = fun x -> match x
    with [] -> []
    | x::xs -> app (rev xs) [x]
let rec rev1 = fun x -> fun y -> match x
    with [] -> y
    | x::xs -> rev1 xs (x::y)
```

sowie

Lemma 1 `app x [] = x`

Lemma 2 `app (rev x) y = rev1 x y`.

Beweisen Sie nun aufeinander aufbauend (das Ergebnis der jeweils vorherigen Aufgaben kann als gegeben betrachtet werden) folgende Aussagen:

1. `rev1 (rev1 x y) z = rev1 y (app x z)`
2. `rev (rev x) = x`
3. `rev (app (rev x) (rev y)) = app y x`

Geben Sie für jeden Schritt die verwendete Regel (z.B. Def. `APP`, IA, IS, Lemma 1 usw.) an!

---

[1] Theorieaufgaben sind Aufgaben, deren Lösung nicht programmiert wird

[2] `http://www.imagemagick.org/script/index.php`

Lösungsvorschlag 13.3

1. `rev1 (rev1 x y) z = rev1 y (app x z)`

   Induktionsverankerung: Es gilt `x = []` und damit folgt:

`rev1 (rev1 x y) z`	
`= rev1 (rev1 [] y) z`	
`= rev1 y z`	(Def. `rev1`)
`= rev1 y (app [] z)`	(Def. app rückwärts)
`= rev1 y (app x z)`	

   Induktionsschritt: Es gilt `x = h :: t` und damit folgt:

`rev1 (rev1 x y) z`	
`= rev1 (rev1 (h::t) y) z`	
`= rev1 (rev1 t (h::y)) z`	(Def. `rev1`)
`= rev1 (h::y) (app t z)`	(Induktionsannahme)
`= rev1 y (h::(app t z))`	(Def. `rev1`)
`= rev1 y (app (h::t) z)`	(Def. app rückwärts)
`= rev1 y (app x z)`	

2. `rev (rev x) = x`

`rev (rev x)`	
`= rev (app (rev x) [])`	(Lemma 1)
`= rev (rev1 x [])`	(Lemma 2)
`= app (rev (rev1 x [])) []`	(Lemma 1)
`= rev1 (rev1 x []) []`	(Lemma 2)
`= rev1 [] (app (x []))`	(Aufgabe 1)
`= app x []`	(Def. `rev1`)
`= x`	(Lemma 1)

3. `rev (app (rev x) (rev y)) = app y x`

$$
\begin{aligned}
&\texttt{rev (app (rev x) (rev y))} && \\
&= \texttt{rev (rev1 x (rev y))} && \text{(Lemma 2)}\\
&= \texttt{app (rev (rev1 x (rev y))) []} && \text{(Lemma 1)}\\
&= \texttt{rev1 (rev1 x (rev y)) []} && \text{(Lemma 2)}\\
&= \texttt{rev1 (rev y) (app x [])} && \text{(Aufgabe 1)}\\
&= \texttt{rev1 (rev y) x} && \text{(Lemma 1)}\\
&= \texttt{app (rev (rev y)) x} && \text{(Lemma 2)}\\
&= \texttt{app y x} && \text{(Aufgabe 2)}
\end{aligned}
$$

Punkteverteilung:

1. 4 Punkte
2. 3 Punkte
3. 3 Punkte

Aufgabe 13.4 (H) Beweise über kleine Kamele [10 Punkte]

1. Zeigen Sie mithilfe der Big-Step Semantik

$$\frac{\texttt{e} \Rightarrow \texttt{v}}{\texttt{map (fun x -> x) e} \Rightarrow \texttt{v}}$$

für

```
let rec map = fun f -> fun a -> match a with
      [] -> []
    | x::xs -> f x :: map f xs
```

Gehen Sie nur von wohlgetypten Ausdrücken aus. Benutzungen von $v \Rightarrow v$ können Sie weglassen. Desweiteren können Sie folgende Setzungen verwenden:

$$
\begin{aligned}
\pi_{\texttt{match}} &:= \texttt{match a with [] -> [] | x::xs -> f x :: map f xs}\\
\pi_{\texttt{map}} &:= \texttt{fun f -> fun a -> match a with}\\
&\qquad \texttt{[] -> [] | x::xs -> f x :: map f xs}
\end{aligned}
$$

2. Gegeben seien die *stets terminierenden* Funktionen

```
let rec length = fun x -> match x
    with [] -> 0
```

```
    | x::xs -> 1 + length xs
let rec app = fun x -> fun y -> match x
    with [] -> y
    | x::xs -> x :: app xs y
let rec rev = fun x -> match x
    with [] -> []
    | x::xs -> app (rev xs) [x]
```

Beweisen Sie nun aufeinander aufbauend (das Ergebnis der jeweils vorherigen Aufgabe kann als gegeben betrachtet werden) folgende Aussagen:

(a) `length (app x y) = length x + length y`

(b) `length (rev x) = length x`

Geben Sie für jeden Schritt die verwendete Regel an (z.B. Def. `APP`, IA, IS usw.)!

Lösungsvorschlag 13.4

1. **Induktionsverankerung:** es gilt $\texttt{e} \Rightarrow \texttt{[]}$ und damit folgt:

$$\text{App}\;\dfrac{\text{GD}\;\dfrac{\texttt{map} = \pi_{\texttt{map}}}{\texttt{map} \Rightarrow \pi_{\texttt{map}}} \qquad \texttt{e} \Rightarrow \texttt{[]} \qquad \text{PM}\;\dfrac{}{\pi_{\texttt{match}}[\texttt{fun x->x}/\texttt{f}, \texttt{[]}/\texttt{a}] \Rightarrow \texttt{[]}}}{\texttt{map (fun x->x) e} \Rightarrow \texttt{[]}}$$

Induktionsschritt: es gilt $\texttt{e} \Rightarrow \texttt{x::xs}$ und damit folgt:

$$\text{App}\;\dfrac{\text{GD}\;\dfrac{\texttt{map} = \pi_{\texttt{map}}}{\texttt{map} \Rightarrow \pi_{\texttt{map}}} \qquad \texttt{e} \Rightarrow \texttt{x::xs} \qquad \text{PM}\;\dfrac{\text{L}\;\dfrac{\text{App}\;\dfrac{}{\texttt{(fun x->x) x} \Rightarrow \texttt{x}} \qquad \text{IA}\;\dfrac{}{\texttt{map (fun x->x) xs} \Rightarrow \texttt{xs}}}{\texttt{(fun x->x) x::map (fun x->x) xs} \Rightarrow \texttt{x::xs}}}{\pi_{\texttt{match}}[\texttt{fun x->x}/\texttt{f}, \texttt{x::xs}/\texttt{a}] \Rightarrow \texttt{x::xs}}}{\texttt{map (fun x->x) e} \Rightarrow \texttt{x::xs}}$$

2. (a) `length (app x y) = length x + length y`

Induktionsverankerung: es gilt `x = []` und damit folgt:

$$\begin{aligned}
&\texttt{length (app x y)} & \\
&= \texttt{length (app [] y)} & \\
&= \texttt{length y} & (\text{Def. } \texttt{app}) \\
&= \texttt{0 + length y} & \\
&= \texttt{length x + length y} & (\text{Def. } \texttt{length} \text{ rückwärts})
\end{aligned}$$

Induktionsschritt: es gilt `x = h::t` und damit folgt:

```
length (app x y)
 = length (app (h::t) y)
 = length (h :: app t y)                  (Def. app)
 = 1 + length (app t y)                   (Def. length)
 = 1 + length t + length y                (Induktionsannahme)
 = length (h::t) + length y               (Def. length rückwärts)
 = length x + length y
```

(b) `length (rev x) = length x`

Induktionsverankerung: es gilt `x = []` und damit folgt:

```
length (rev x)
 = length (rev [])
 = length []                              (Def. rev)
 = length x
```

Induktionsschritt: es gilt `x = h::t` und damit folgt:

```
length (rev (h::t))
 = length (app (rev t) [h])               (Def. rev)
 = length (rev t) + length [h]            (Aufgabe1)
 = length t + length [h]                  (Induktionsannahme)
 = length t + (1 + length [])             (Def. length)
 = length t + (1 + 0)                     (Def. length)
 = 1 + length t
 = length (h::t)                          (Def. length rückwärts)
 = length x
```

Punkteverteilung:

1. 3 Punkte
2. (a) 3 Punkte

   (b) 3 Punkte

Ein weiterer Punkt wird zusätzlich bei einer der drei Teilaufgaben vergeben.