# Klausur Programmierung WS 2002/03

Prof. Dr. Gert Smolka, Dipl. Inf. Thorsten Brunklaus

14. Dezember 2002

| Leo Schlau | 45 |
|---|---|
| Vor- und Nachname | Sitz-Nr. |
| 4711 | 007 |
| Matrikelnummer | Code |

- Bitte öffnen Sie das Klausurheft erst dann, wenn Sie dazu aufgefordert werden (gleiche Bearbeitungszeit für alle).
- Sie können die Klausur nur auf dem für Sie vorgesehen Platz mitschreiben. Sie müssen das mit Ihrem Namen und Ihrer Matrikelnummer versehene Klausurheft verwenden.
- Hilfsmittel sind nicht zugelassen. Am Arbeitsplatz dürfen nur Schreibgeräte, Getränke, Speisen sowie Ausweise mitgeführt werden. Taschen, Jacken und Mäntel müssen an den Wänden des Klausursaals zurückgelassen werden.
- Verlassen des Saals ohne vorherige Abgabe des Klausurhefts gilt als Täuschungsversuch.
- Wenn Sie während der Bearbeitung auf die Toilette müssen, geben Sie bitte Ihr Klausurheft bei der Aufsicht ab. Es kann immer nur eine Person auf die Toilette.
- Alle Lösungen müssen auf den bedruckten rechten Seiten des Klausurhefts notiert werden. Die leeren linken Seiten dienen als Platz für Skizzen und werden **nicht korrigiert**. Notizpapier ist nicht zugelassen. Sie können mit Bleistift schreiben.
- Für die Bearbeitung der Klausur stehen 150 Minuten zur Verfügung. Insgesamt sind 150 Punkte erreichbar. Die für jede Aufgabe angebene Punktzahl gibt Ihnen also einen Anhaltspunkt, wieviel Zeit Sie auf die Bearbeitung der Aufgabe verwenden sollten. Zum Bestehen der Klausur genügen 75 Punkte.
- Bitte legen Sie Ihren Personalausweis oder Reisepass sowie Ihren Studierendenausweis zur Identifikation neben sich.
- Viel Erfolg!

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | $\sum$ | Note |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| 14 | 12 | 22 | 40 | 12 | 24 | 26 | 150 | |

**Aufgabe 1: Bindungs- und Typanalyse** (14 = 4 + 4 + 6)

(a) Geben Sie alle Bezeichner an, die in dem folgenden Ausdruck frei vorkommen:

```
(fn x => fn y => h (f x) (g y)) x
```

(b) Seien die folgenden Deklarationen gegeben:

```
fun p x y f g h = h (f x) (g y)

val x = p 3 4 (fn x => x-1) (fn x => x+1) (fn x => fn y => x*y)
```

(i) An welchen Wert wird `x` gebunden?

(ii) Geben Sie das Typschema an, das der Interpreter für den Bezeichner `p` bestimmt.

**Aufgabe 2: Falten von Listen** ($12 = 2 * 6$) Seien die folgenden Prozeduren gegeben:

```
fun p f s g xs = foldr f s (map g xs)

fun q f g xs = map f (map g (rev xs))
```

Wir bezeichnen zwei Prozeduren als gleichwertig, wenn sie denselben Typ haben und die gleichen Ergebnisse liefern.

(a) Geben Sie eine zu `p` gleichwertige Prozedur `p'` an, die nur `foldr` verwendet:

```
fun p' f s g xs = foldr
```

(b) Geben Sie eine zu `q` gleichwertige Prozedur `q'` an, die nur `foldl` verwendet:

```
fun q' f g xs = foldl
```

**Aufgabe 3: Sortieren durch Mischen** $(22 = 3 * 6 + 4)$ Sie sollen eine polymorphe Prozedur schreiben, die Listen gemäß einer linearen Ordnung sortiert. Die lineare Ordnung wird dabei durch eine Prozedur $p\colon \alpha * \alpha \to \texttt{order}$ dargestellt, für die $p(x, y) = \texttt{LESS}$ genau dann gilt, wenn $x$ in einer sortierten Liste vor $y$ stehen muss.

(a) Geben Sie eine Prozedur

```
split : 'a list → 'a list * 'a list
```

an, die eine Liste in zwei etwa gleich große Listen zerlegt.

(b) Geben Sie eine Prozedur

```
merge : ('a * 'a → order) → 'a list → 'a list → 'a list
```

an, die zwei sortierte Liste gemäß einer linearen Ordnung zu einer sortierten Liste verschmilzt.

(c) Geben Sie eine Prozedur

```
sort : ('a *'a → order) → 'a list → 'a list
```

an, die eine Liste gemäß einer linearen Ordnung sortiert.

(d) Geben Sie die asymptotische Laufzeit Ihrer Prozedur `sort` mit $\Theta$ an.

**Aufgabe 4: Arithmetische Ausdrücke** ($40 = 2 * 5 + 8 + 2 * 6 + 10$) Seien die folgenden Deklarationen gegeben:

```
type var = int
datatype exp = C of int | V of var | A of exp * exp | M of exp * exp
exception Var
```

Die Werte des Typs `exp` stellen arithmetische Ausdrücke über $\mathbb{Z}$ dar, die mit Konstanten, Variablen, Addition und Multiplikation gebildet sind.

(a) Schreiben Sie eine Prozedur

```
eval : exp → int  (* Var *)
```

die den Wert eines Ausdrucks liefert, falls er keine Variable enthält. Wenn der Ausdruck eine Variable enthält, soll die Ausnahme `Var` geworfen werden.

(b) Schreiben Sie eine Prozedur

```
var : exp → bool
```

die testet, ob ein Ausdruck Variablen enthält. Falls der Ausdruck Variablen enthält, soll `true`, sonst `false` geliefert werden. Realisieren Sie `var` mithilfe der obigen Prozeduren `eval`, indem Sie die Ausnahme `Var` fangen.

(c) Schreiben Sie eine Prozedur

```
vars : exp → var list
```

die zu einem Ausdruck eine Liste liefert, die alle Variablen enthält, die in dem Ausdruck vorkommen.

(d) Unter einer Umgebung wollen wir eine Prozedur verstehen, die zu jeder Variable eine ganze Zahl liefert:

```
type env = var → int
```

Geben Sie eine Prozedur

```
adjoin : env → var → int → env
```

an, die zu einer Umgebung *env*, einer Variablen $x$ und einer Zahl $z$ eine Umgebung liefert, die $x$ den Wert $z$ und allen anderen Variablen denselben Wert wie *env* gibt.

(e) Schreiben Sie eine Prozedur

*eval′ : env → exp → int*

die den Wert eines Ausdrucks in einer Umgebung liefert.

(f) Schreiben Sie eine Prozedur

*subexp : exp → int list → exp  (* Subscript *)*

die zu einem Ausdruck und einer Adresse den entsprechenden Teilausdruck liefert. Die Adressierung soll wie bei Bäumen erfolgen. Beispielsweise soll

```
subexp (A(V 1, M(C 1, C 2))) [2,2] = C 2
```

gelten. Wenn die Adresse keinen Teilausdruck bezeichnet, soll die Ausnahme `Subscript` geworfen werden.

**Aufgabe 5: Grenze von Bäumen mit Akkumulatorargument** ($12 = 2 * 6$) Die Prozedur

```
fun frontier (T(x,[])) = [x]
  | frontier (T(_,ts)) = foldr
   (fn (t,xs) => frontier t @ xs) nil ts
```

```
val frontier : 'a tree -> 'a list
```

liefert zu einem Baum eine Liste, die die Marken der Blätter des Baums enthält. Sie sollen eine effizientere Version dieser Prozedur angeben, die mithilfe eines Akkumulatorarguments die Konkatenation `frontier t @ xs` einspart.

(a) Schreiben Sie eine Prozedur

```
frontier': 'a tree -> 'a list -> 'a list
```

sodass für alle Bäume `t` und alle Listen `ys` gilt:

```
frontier' t ys = (frontier t) @ ys
```

Verwenden Sie dabei nur eine Faltungsprozedur und eine Abstraktion.

```
fun frontier' (T(x,[])) ys =

  | frontier' (T(_,ts)) ys =
```

(b) Schreiben Sie eine Prozedur

```
frontier'': 'a tree -> 'a list -> 'a list
```

sodass für alle Bäume `t` und alle Listen `ys` gilt:

```
frontier'' t ys = (rev (frontier t)) @ ys
```

Verwenden Sie dabei nur eine Faltungsprozedur und eine Abstraktion.

```
fun frontier'' (T(x,[])) ys =

  | frontier'' (T(_,ts)) ys =
```

**Aufgabe 6: Iteration und Induktion** $(24 = 4 + 9 + 7 + 4)$ Sei $X$ eine Menge. Die Prozedur

$$\begin{aligned}
&\mathit{count}\colon\ \mathcal{L}(X) \times X \to \mathbb{N} \\
&\mathit{count}(\mathit{nil}, y) = 0 \\
&\mathit{count}(x :: xr, y) = \mathit{count}(xr, y) + (\text{if } x = y \text{ then } 1 \text{ else } 0)
\end{aligned}$$

liefert zu einer Liste $xs$ und einem Wert $y$ die Anzahl der Auftreten von $y$ in $xs$.

(a) Geben Sie die definierenden Gleichungen für eine iterative Prozedur

$$\mathit{count}'\colon\ \mathcal{L}(X) \times X \times \mathbb{Z} \to \mathbb{Z}$$

an, für die gilt:

$$\forall xs \in \mathcal{L}(X)\ \forall y \in X\ \forall a \in \mathbb{Z}\colon\ \mathit{count}'(xs, y, a) = \mathit{count}(xs, y) + a$$

$$\mathit{count}'(\mathit{nil}, y, a) =$$

$$\mathit{count}'(x :: xr, y, a) =$$

(b) Beweisen Sie, dass Ihre Prozedur $\mathit{count}'$ die verlangte Eigenschaft hat. Verwenden Sie dazu strukturelle Induktion über $xs \in \mathcal{L}(X)$ und unterscheiden Sie drei Fälle.

(c) Geben Sie für Ihren Beweis die folgenden Dinge an:

Grundmenge:

Aussagemenge:

Induktionsrelation:

Induktionsannahme für $zs \in \mathcal{L}(X)$:

(d) Geben Sie in Standard ML eine nicht rekursive Prozedur

```
count'' : ''a list -> ''a -> int
```

an, die zu einer Liste *xs* und zu einem Wert *x* die Anzahl der Auftreten von *x* in *xs* liefert. Verwenden Sie dabei `foldl`.

**Aufgabe 7: Laufzeiten** $(26 = 3 * 2 + 2 * (4 + 4 + 2))$ Seien die folgenden Prozeduren gegeben:

$f : \mathbb{N} \to \mathbb{N}, \quad f(n) = \text{if } n = 0 \text{ then } 0 \text{ else } f(n-1) + n$

$g : \mathbb{N} \to \mathbb{N}, \quad g(n) = \text{if } n = 0 \text{ then } 0 \text{ else } (f(n)\ ;\ 2n + g(n-1))$

$h : \mathbb{N} \to \mathbb{N}, \quad h(n) = \text{if } n = 0 \text{ then } 1 \text{ else } h(n-1) + h(n-1)$

(a) Geben Sie $f(n)$ für alle $n \in \mathbb{N}$ ohne Rekursion an.

$f(n) =$

(b) Geben Sie die exakte Laufzeit der rekursiven Prozedur $f$ an. Verwenden Sie dabei keine Rekursion.

$\phi_f(n) =$

(c) Geben Sie die asymptotische Laufzeit der rekursiven Prozedur $f$ mit $\Theta$ an.

(d) Geben Sie $g(n)$ für alle $n \in \mathbb{N}$ ohne Rekursion an.

$g(n) =$

(e) Geben Sie die exakte Laufzeit der rekursiven Prozedur $g$ an. Verwenden Sie dabei keine Rekursion.

$\phi_g(n) =$

(f) Geben Sie die asymptotische Laufzeit der rekursiven Prozedur $g$ mit $\Theta$ an.

(g) Geben Sie $h(n)$ für alle $n \in \mathbb{N}$ ohne Rekursion an.

$h(n) =$

(h) Geben Sie die exakte Laufzeit der rekursiven Prozedur $h$ an. Verwenden Sie dabei keine Rekursion.

$\phi_h(n) =$

(i) Geben Sie die asymptotische Laufzeit der rekursiven Prozedur $h$ mit $\Theta$ an.