1. Klausur Programmierung 1 (WS 2008/2009)

Prof. Dr. Gert Smolka Mark Kaminski, M.Sc.

20. Dezember 2008

INTERN

Name Sitzplatz

Matrikelnummer Code

Bitte öffnen Sie das Klausurheft erst dann, wenn Sie dazu aufgefordert werden.

Sie können die Klausur nur auf dem für Sie vorgesehen Platz schreiben. Sie müssen das mit Ihrem Namen und Ihrer Matrikelnummer versehene Heft verwenden.

Hilfsmittel sind nicht zugelassen. Am Arbeitsplatz dürfen nur Schreibgeräte, Getränke, Speisen und Ausweise mitgeführt werden. Taschen und Jacken müssen an den Wänden des Klausursaals zurückgelassen werden.

Das Verlassen des Saals ohne Abgabe des Klausurhefts gilt als Täuschungsversuch.

Wenn Sie während der Bearbeitung zur Toilette müssen, geben Sie bitte Ihr Klausurheft bei der Aufsicht ab. Es kann immer nur eine Person zur 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 120 Minuten zur Verfügung. Insgesamt können 120 Punkte erreicht werden. 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 60 Punkte.

Bitte legen Sie zur Identifikation Ihren Personalausweis bzw. Reisepass sowie Ihren Studierendenausweis neben sich.

Viel Erfolg!

1	2	3	4	5	6
20	15	20	15	20	30

Summe
120

Note

Aufgabe 1. Striktes Sortieren durch Mischen (20 Punkte)

Schreiben Sie eine Prozedur

$$sort : (\alpha * \alpha \rightarrow order) \rightarrow \alpha\ list \rightarrow \alpha\ list$$

die Listen gemäß einer Vergleichsprozedur strikt sortiert (Doppelauftreten werden eliminiert). Sortieren Sie durch Mischen und halten Sie sich unbedingt an die vorgegebene Struktur.

```
fun split xs =




fun merge compare xs ys = case (xs,ys) of
```

```
fun sort compare nil =

  | sort compare [x] =

  | sort compare xs  = let
      val (ys,zs) =
    in



    end
```

Aufgabe 2. Primzerlegung (15 Punkte)

Schreiben Sie eine Prozedur

$$pf : int \to int \to int\ list$$

sodass $pf\ 2\ n$ für $n \geq 2$ die Primzerlegung von n liefert. Beispielsweise soll $pf\ 2\ 60 = [2, 2, 3, 5]$ gelten. Verwenden Sie keine Hilfsprozeduren.

```
fun pf k n =
```

Aufgabe 3. Bäume (4·5=20 Punkte)

a) Schreiben Sie eine Prozedur *pre* : α *tree* → α *list*, die die Marken eines markierten Baums gemäß der Präordnung liefert.

b) Schreiben Sie eine Prozedur *post* : *tree* → *tree list*, die die Teilbäume eines Baums gemäß der Postordnung liefert.

c) Schreiben Sie eine Prozedur *get* : *int* → *tree* → *tree*, die zu $n \geq 0$ und t den Teilbaum von t mit der Postnummer n liefert (wenn er existiert). Verwenden Sie dazu die Prozedur *post* aus (b) und *List.nth*.

d) Schreiben Sie eine Prozedur *frontier* : α *ltr* → α *list*, die die Marken der Blätter eines markierten Baums liefert (von links nach rechts, mit Doppelauftreten).

Aufgabe 4. Finitäre Mengen (15 Punkte)

Seien finitäre Mengen durch Bäume dargestellt.

a) Zeichnen Sie 2 verschiedene Bäume, die die Menge $\{\emptyset, \{\emptyset\}\}$ darstellen.

b) Schreiben Sie eine Prozedur $el : tree \to tree \to bool$, die für t und t' testet, ob die durch t dargestellte Menge ein Element der durch t' dargestellten Menge ist.

Aufgabe 5. Test auf Balanciertheit (20 Punkte)

Ein Baum ist genau dann balanciert, wenn seine Unterbäume alle balanciert sind und die gleiche Tiefe haben.

Schreiben Sie eine Prozedur *balanced* : *tree* → *bool*, die testet, ob ein Baum balanciert ist.

Aufgabe 6. Statische und dynamische Semantik (30 Punkte)

Sie sollen die statische und dynamische Semantik der folgenden Teilsprache von Standard ML implementieren:

$$t ::= unit \mid t * t$$
$$e ::= x \mid () \mid (e, e) \mid fst\ e \mid snd\ e \mid let\ x = e\ in\ e$$

Halten Sie sich unbedingt an die vorgegebene Struktur.

```
datatype ty = Unit | Pair of ty * ty
type id = string
datatype exp = Id of id | U | P of exp * exp
             | Fst of exp | Snd of exp | Let of id * exp * exp



fun update f x a y =



exception Error



fun elab f    (Id x)     =


  | elab f      U        =


  | elab f   (P(e,e'))   =


  | elab f    (Fst e)    =


  | elab f    (Snd e)    =


  | elab f (Let(x,e,e')) =
```

```
datatype value = UV | PV of value * value


fun eval f    (Id x)     =

  | eval f      U        =

  | eval f   (P(e,e'))   =

  | eval f    (Fst e)    =

  | eval f    (Snd e)    =

  | eval f (Let(x,e,e')) =
```

Sie können benutzen:

$datatype\ order = LESS \mid EQUAL \mid GREATER$

$datatype\ tree = T\ of\ tree\ list$

$datatype\ \alpha\ ltr = L\ of\ \alpha * \alpha\ ltr\ list$

$List.nth : \alpha\ list * int \to \alpha$

$List.exists : (\alpha \to bool) \to \alpha\ list \to bool$

$List.all : (\alpha \to bool) \to \alpha\ list \to bool$

$List.concat : \alpha\ list\ list \to \alpha\ list$

$map : (\alpha \to \beta) \to \alpha\ list \to \beta\ list$

$foldl : (\alpha * \beta \to \beta) \to \beta \to \alpha\ list \to \beta$