Klausur Programmierung WS2003/04

Prof. Dr. Andreas Podelski und Dipl. Inf. Andrey Rybalchenko

23. Februar 2004

Vor- und Nachname

Matrikelnummer Sitz-Nr.

- 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 dürfen nur SML zum Lösen von Programmieraufgaben 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 angegebene Punktzahl gibt Ihnen also einen Anhaltspunkt, wieviel Zeit Sie auf die Bearbeitung der Aufgabe verwenden sollten. Zum Bestehen der Klausur genügen 75 Punkte.
- Keine Abgaben während der letzten 30 Min. der Bearbeitungszeit.
- Bitte legen Sie Ihren Personalausweis oder Reisepaß sowie Ihren Studierendenausweis zur Identifikation neben sich.
- Viel Erfolg!

1	2	3	4	5	6	7	8	9	*	$\sum$	Note
14	30	18	10	20	13	25	10	10	10	150	

Aufgabe 1 (Rekursive Prozeduren). $(14 = 4 + 4 + 2 + 4)$
Sei die folgende Prozedur gegeben:

$$f : \mathbb{N} * \mathbb{N} \to \mathbb{N}$$
$$f(x, y) = \text{if } x < y \text{ then } f(x + 1, y) \text{ else } y$$

a) Geben Sie die Rekursionsfunktion und die Rekursionsrelation von f an.
b) Geben Sie eine natürliche Terminierungsfunktion für f an.
c) Wie würden Sie vorgehen, falls die Prozedur f kaskadiert deklariert wäre:

$$f\ x\ y = \text{if } x < y \text{ then } f\ (x + 1)\ y \text{ else } y$$

Sei die folgende Prozedur gegeben:

$$\begin{aligned}
exp :&\mathbb{Z} \to \mathbb{N} \to \mathbb{N} \\
exp(x, 0) &= 1 \\
exp(x, n) &= \text{if } (n \bmod 2) = 0 \text{ then } exp(x * x, n \text{ div } 2) \\
&\qquad \text{else } x * exp(x * x, (n - 1) \text{ div } 2)
\end{aligned}$$

d) Bringen Sie die Prozedur exp in die endrekursive Form.

Aufgabe 2 (Korrektheit und Laufzeit). $(30 = 4 + 10 + 9 + 7)$
Seien die folgenden Prozeduren gegeben:

$$\begin{aligned}
insert &: (\text{int } * \text{ int list}) \to \text{int list} \\
sort &: \text{int list} \to \text{int list} \\
insert(x, [\,]) &= [x] \\
insert(x, y :: ys) &= \text{if } x \leq y \text{ then } x :: y :: ys \text{ else } y :: insert(x, ys) \\
sort([\,]) &= [\,] \\
sort(x :: xs) &= insert(x, sort(xs))
\end{aligned}$$

a) Geben Sie eine natürliche Terminierungsfunktion für $insert$ an.
b) Beweisen Sie, dass $insert(x, ys)$ eine aufsteigend sortierte Liste liefert, falls ys eine aufsteigend sortierte Liste ist.
c) Beweisen Sie, dass $sort(xs)$ eine aufsteigend sortierte Liste liefert.
d) Welche Laufzeit hat die Prozedur $sort$? (mit Begründung)

Aufgabe 3 (Laufzeit). $(18 = 6 + 12)$
a) Seien $f, g \in \mathbb{Z} \to \mathbb{R}$. Wir definieren die Relation $\preceq$:

$$f \preceq g \text{ genau dann wenn } \exists n_0 \in \mathbb{N}\ \exists c \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c\ g(n).$$

Beweisen Sie, dass die Relation $\preceq$ transitiv ist, d.h. dass für alle $f, g, h \in \mathbb{N} \to \mathbb{R}$ gilt:

$$f \preceq g \land g \preceq h \implies f \preceq h$$

b) Geben Sie möglichst einfache Prozeduren int $\rightarrow$ unit, die für $n \in \mathbb{N}$ die folgenden Laufzeiten haben: (Sie können dabei die Ergebnisfunktionen von im Rahmen der Aufgabe angegebenen Prozeduren mit der entsprechenden Laufzeitannahme verwenden. Neben den Ergebnisfunktionen der im Rahmen der Aufgabe angegebenen Prozeduren dürfen Sie nur Funktionen verwenden, die mit konstanter Laufzeit in die Analyse eingehen.)

$\Theta(\log n)$ `fun f_logn n =`

$\Theta(n)$ `fun f_n n =`

$\Theta(n \log n)$ `fun f_nlogn n =`

$\Theta(2^n)$ `fun f_exp n =`

Aufgabe 4 (Direkt ausführbare Programme). (10)
Schreiben Sie ein Programm, das die Anzahl der Zeilen in einer Datei auf die Benutzeroberfläche schreibt. Das Startargument beim Programmaufruf ist der Dateiname. Sie können die folgenden Prozeduren der Struktur `TextIO` verwenden:

```
type instream
val openIn    : string   -> instream
val inputLine : instream -> string
val closeIn   : instream -> unit
```

Die Prozedur `CommandLine.arguments: unit` $\rightarrow$ `string list` liefert die Argumentenliste.

Aufgabe 5 (Große natürliche Zahlen). $(20 = 5 + (6 + 6) + 3)$
Wir realisieren große natürliche Zahlen mithilfe von Listen und den Zahlen in `int`. Große Zahlen sollen den Typ `bignat` haben. Wir wollen große Zahlen aus Zeichenketten lesen (`fromString`), als Zeichenketten ausgeben (`toString`), arithmetische Operationen anwenden (`add`, `mult`), auf Gleichheit (=) und kleiner-als (`less`) testen.
a) Geben Sie eine adäquate Signatur für eine Struktur, die große Zahlen zusammen mit den oben genannten Operationen zur Verfügung stellt.

```
signature BIGNAT = sig




end
```

b) Geben Sie eine Realisierung der Multiplikation und des `less`-Vergleichs an. Wir nehmen an, dass bei der Multiplikation der zweite Operand eingeschränkt ist: er ist kleiner als die Basis. Sie dürfen keine Hilfsprozeduren über die großen Zahlen verwenden.

```
type bignat =

val Basis =

fun mult a b =





fun less a b =
```

c) Für welche maximale Basisgröße ist Ihre Realisierung korrekt? (eine Zeile Begründung)

Aufgabe 6 (Kontextfreie Grammatik). $(13 = 4 + 3 + 6)$ Gegeben sei die folgende kontextfreie Grammatik (mit dem Startsymbol *exp*):

$$\begin{aligned} c &= 0 \mid 1 \mid \ldots \mid 9 \\ num &= c \mid num\ c \\ exp &= num \mid exp + exp \end{aligned}$$

a) Geben Sie Syntaxbäume für die folgenden Sätze an.

$123 \qquad\qquad 12 + 3$

b) Ist die oben angegebene kontextfreie Grammatik eindeutig? (mit Begründung).

c) Falls die Grammatik nicht eindeutig ist, geben Sie eine eindeutige Grammatik an, die dieselben Sätze darstellt.

Aufgabe 7 (Statische und dynamische Semantik von F). $(25 = 8+5+12)$
a) Wir erweitern F auf heterogene Paare (bzgl. Typen der Elemente), z.B. `(2, true)`. Geben Sie die statische und dynamische Semantik von heterogenen Paaren an.

$$\overline{T \vdash (e_1, e_2) \implies t_1 * t_2}$$

$$\overline{V \vdash (e_1, e_2) \implies (v_1, v_2)}$$

b) Geben Sie die statische Semantik von Abstraktion, die ein Paar als Argument bekommt, an.

$$\overline{T \vdash \texttt{fn}\,(x, y) : t_1 * t_2 \texttt{ => } e \implies t_1 * t_2 \to t}$$

c) Durch welches Tupel stellen Sie die Prozedur, die durch die Auswertung einer Abstraktion erhalten wird, dar? Geben Sie die dynamische Semantik von Abstraktion, die ein Paar als Argument bekommt, und deren Anwendung an.

$$\overline{V \vdash \texttt{fn}\,(x, y) : t_1 * t_2 \texttt{ => } e \implies \qquad\qquad\qquad}$$

$$\overline{V \vdash e\,(e_1, e_2) \implies v}$$

Aufgabe 8 (Imperative Objekte). (10) Wir realisieren ein Notizbuch, das über nur einen Notizeintrag (eine Zeichenkette) verfügt, mit der folgenden Funktionalität:

- Am Anfang ist der Eintrag eine leere Zeichenkette.
- Der Eintrag kann durch einen neuen Eintrag ersetzt werden.
- Der Eintrag kann auf die Benutzeroberfläche ausgegeben werden.

Ein Notizbuch wird durch eine Prozedur *memo* : `action` $\to$ `unit` realisiert. Der Typ `action` ist wie folgt definiert:

```
datatype action = Write of string | Read
```

Hier ist ein Beispiel für die Notizbuchbenutzung:

```
> memo Read;

> memo (Write "AAA");
> memo Read;
AAA
> memo (Write "BBB");
> memo Read;
BBB
```

Geben Sie die Prozedur `memo` an.

Aufgabe 9 (Übersetzung). (10)
Gegeben sei der folgende Datentyp für arithmetische Ausdrücke.

```
datatype exp = Const of int | Sum of exp * exp |
               Sub of exp * exp | Mult of exp * exp
```

Schreiben Sie einen Übersetzer von arithmetischen Ausdrücken in ein Programm in der folgende Maschinensprache (die Bedeutung der einzelnen Befehle entspricht den Befehlsnamen, z.B. `mul` ersetzt zwei obere Stackelemente durch deren Produkt):

```
datatype instruction = con of int | sum | sub | mul | halt
datatype program = instruction list
```

Der Übersetzer soll durch die Prozedur *compile* : `exp` → `program` realisiert werden.

Aufgabe 10 (Bonus). (10)
Wir erweitern das Notizbuch aus der Aufgabe 8. Wir realisieren ein TODO-Verwaltungssystem in dem es einen TODO-Geber (den "Chef") und eine fixierte Anzahl von TODO-Ausführern ("TODOers") gibt (die Referenz auf diese Liste wird `people` genannt).
Ein TODO-Auftrag wird vom Chef für einen bestimmten TODOer erteilt. Der Auftrag ist mit einem Wichtigkeitsgrad (`HIGH, LOW, NORMAL`) versehen. Bei den Aufträgen mit dem gleichen Wichtigkeitsgrad ist der zuerst erteilte Auftrag wichtiger.
Ein TODOer kann den wichtigsten seiner Aufträge lesen. Außerdem kann er diesen Auftrag bearbeiten und somit aus seiner TODO-Liste entfernen. Die TODO-Listen sollen mithilfe von Referenzen realisiert werden.
Hier sind ein Paar Ausgangspunkte für die einzelnen Komponenten unseres Systems:

```
datatype grad = HIGH | LOW | NORMAL
datatype todo = Empty | Todo of string * grad * todo ref
type todolist = todo ref
datatype todoer = Todoer of string * todolist

(* >= Ordnung auf den Priorit"aten *)
fun geq_grads(x, y) = y = LOW orelse x = HIGH orelse x = y
(* Priorit"aten ausgeben *)
fun print_grad LOW = print "LOW"
  | print_grad NORMAL = print "NORMAL"
  | print_grad HIGH = print "HIGH"

val people = ref [Todoer ("Mr. 1",  ref Empty),
                  Todoer ("Mrs. 2", ref Empty),
                  Todoer ("Mr. 3",  ref Empty)]
```

Realisieren Sie die folgenden Prozeduren:
a) Finde die TODOs von dem TODOer `name`.

$$find\_todos : \texttt{string} \to \texttt{todolist}$$

```
fun find_todos name =
```

b) Füge einen neuen Auftrag in die TODOs ein (Prioritäten nicht vergessen!!!).

$$add\_todo : \texttt{todolist} \rightarrow \texttt{string} \rightarrow \texttt{grad} \rightarrow \texttt{unit}$$

```
fun add_todo todos what prior =
```

c) Gebe einen neuen Auftrag `what` mit der Priorität `prior` für den TODOer `name`:

$$assign\_todo : \texttt{string} \rightarrow \texttt{string} \rightarrow \texttt{string} \rightarrow \texttt{grad} \rightarrow \texttt{unit}$$

```
fun assign_todo name what prior =
```

d) Gebe den wichtigsten Auftrag von dem TODOer `name` auf die Benutzeroberfläche aus.

$$next\_todo : \texttt{string} \rightarrow \texttt{unit}$$

```
fun next_todo name =
```

e) Lösche den wichtigsten Auftrag von dem TODOer `name` aus seiner TODOs.

$$process\_todo : \texttt{string} \rightarrow \texttt{unit}$$

```
fun process_todo name =
```

f) Teste, ob noch etwas bei jemandem zu tun gibt.

$$is\_all\_done : \texttt{unit} \rightarrow \texttt{bool}$$

```
fun is_all_done =
```