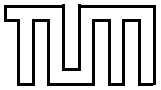
TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



**Lehrstuhl für Sprachen und Beschreibungsstrukturen** WS 2008/09
**Einführung in die Informatik 2** **Klausur**
Prof. Dr. Helmut Seidl, T. M. Gawlitza, S. Pott, M. Schwarz 03.02.2009

| Name | Vorname | Studiengang | Matrikelnummer |
|---|---|---|---|
| Hörsaal | Reihe | Sitzplatz | Unterschrift |

# Allgemeine Hinweise:

- **Bitte füllen Sie die oben angegebenen Felder vollständig aus und unterschreiben Sie!**
- **Schreiben Sie nicht mit Bleistift oder in roter/grüner Farbe!**
- **Die Arbeitszeit beträgt 120 Minuten.**
- **Prüfen Sie, ob Sie alle 9 Seiten erhalten haben.**
- **In dieser Klausur können Sie insgesamt 50 Punkte erreichen. Zum Bestehen werden 20 Punkte benötigt.**
- **Im Anhang finden Sie die im Modul List definierten OCaml-Funktionen, die Sie verwenden dürfen.**

**Aufgabe 1** [6 Punkte] **Multiple-Choice**

Kreuzen Sie zutreffende Antworten an bzw. geben Sie die richtige Antwort.

Punkte werden nach folgendem Schema vergeben:

- Falsche Antwort: $-\frac{1}{2}$ Punkt
- Keine Antwort: $0$ Punkte
- Richtige Antwort: $\frac{1}{2}$ Punkt

Eine negative Gesamtpunktzahl wird zu $0$ aufgerundet.

a) Verifikation

i) **false** ist die schwächste Zusicherung. ☐ Richtig ☐ Falsch

ii) Wenn in einem lokal konsistent annotierten Kontrollfluß-Diagramm der Start-Knoten mit **false** annotiert ist, dann ist bei Erreichen eines Stop-Knotens die Zusicherung dort ebenfalls nicht erfüllt. ☐ Richtig ☐ Falsch

iii) Wenn in einem lokal konsistent annotierten Kontrollfluß-Diagramm der Start-Knoten mit **true** und jeder Stop-Knoten mit **false** annotiert ist, dann terminiert das Programm nie. ☐ Richtig ☐ Falsch

iv) Geben Sie die stärkste Bedingung `A` an, so dass

$$\mathbf{WP}[\![\mathtt{x} = \mathtt{y} + 5;]\!](\mathtt{A}) \equiv \mathtt{y} = \mathtt{z} - 10$$

gilt. [ ]

b) OCaml

i) `f (g x)` wertet sich zu dem gleichen Wert aus wie `f g x`. ☐ Richtig ☐ Falsch

ii) Die durch

```
let f x = let p a b = a + b in p x
```

definierte Funktion `f` ist vom Typ `int -> int`. ☐ Richtig ☐ Falsch

iii) `fold_left` im Modul `List` ist endrekursiv. ☐ Richtig ☐ Falsch

iv) Die folgende OCaml-Zeile ist fehlerfrei:

```
match 3 with (x,y) -> x + y | x -> x
```

☐ Richtig ☐ Falsch

v) Die Auswertung des Ausdrucks

```
map (fold_left (fun x y -> x + 1) 0) [["a";"b"];["c"]]
```

liefert [ ]

vi) Die Auswertung des Ausdrucks

```
let x = 5 in let f y = x + y in let x = 37 in f 5
```

liefert [ ]

c) Verifikation funktionaler Programme

i) Jeder Beweis einer Aussage $\mathtt{e} \Rightarrow \mathtt{v}$ zeigt insbesondere, dass die Auswertung des Ausdrucks $\mathtt{e}$ terminiert. ☐ Richtig ☐ Falsch

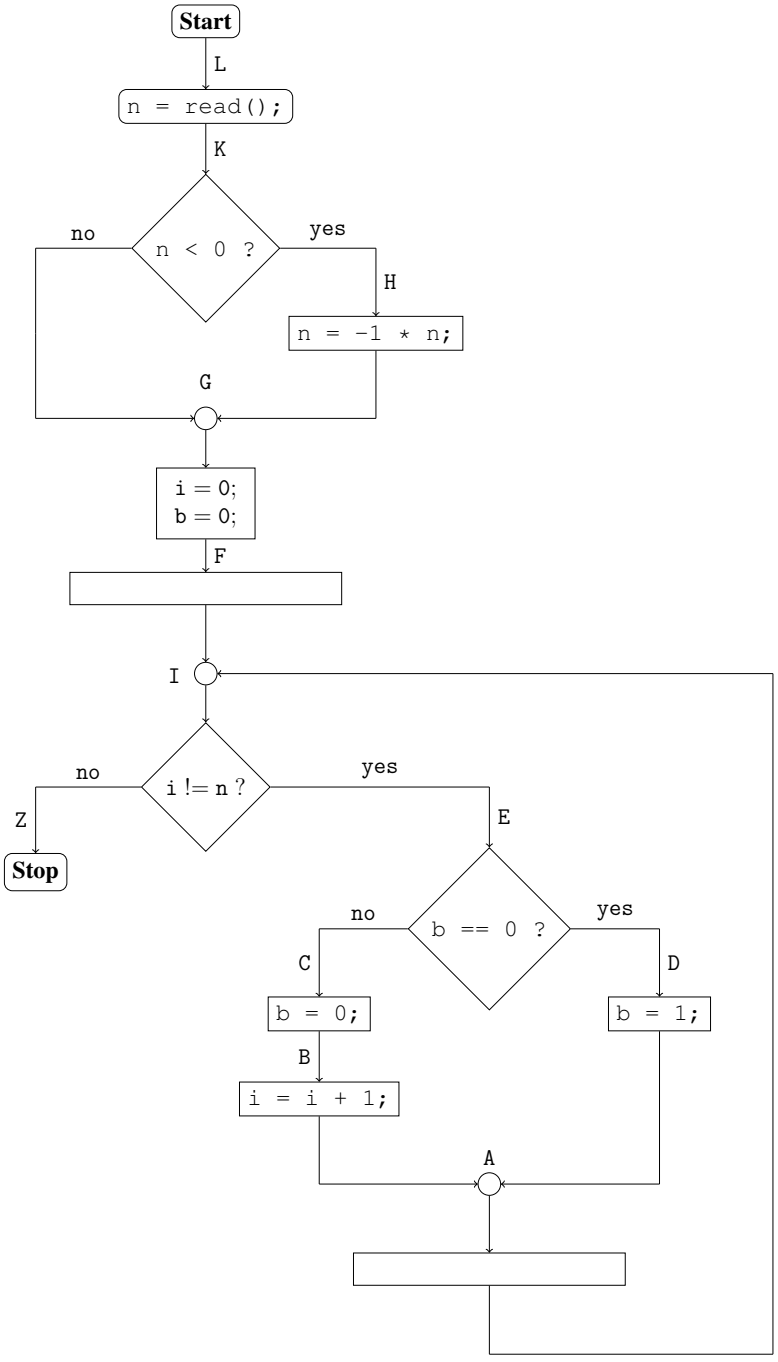
ii) Die folgende abgeleitete Regel ist gültig:

$$\frac{\mathtt{e}' \text{ und } \mathtt{e}'' \text{ terminieren} \qquad \mathtt{e} = \mathtt{e}' :: \mathtt{e}''}{(\mathtt{match\ e\ with}\ [] \rightarrow \mathtt{e}_1 \mid \mathtt{x} :: \mathtt{xs} \rightarrow \mathtt{e}_2) = \mathtt{e}_2[\mathtt{e}'/\mathtt{x}, \mathtt{e}''/\mathtt{xs}]}$$

☐ Richtig ☐ Falsch

**Aufgabe 2** [11 Punkte] **Terminierung**

In dieser Aufgabe soll gezeigt werden, dass das durch folgendes Kontrollfluß-Diagramm gegebene MiniJava-Programm terminiert.




Gehen Sie wie folgt vor:

a) Ergänzen Sie obiges Kontrollfluß-Diagramm um eine geeignete Variable `r`, die in jedem Schleifendurchlauf kleiner wird und bei jedem Betreten der Schleife positiv ist.

b) Geben Sie eine geeignete Schleifen-Invariante an.

c) Zeigen Sie, dass das gegebene Programm terminiert.

**Aufgabe 3** [6 Punkte] **Verifikation in Anwesenheit von Prozeduren**

Es sei angenommen, dass für eine Prozedur g() folgendes Tripel gültig ist:

$$\{\mathtt{x} = m \wedge \mathtt{y} = n\}\ \mathtt{g}();\ \{\mathtt{x} = m + n\}$$

Dabei sind $m$ und $n$ logische Variablen. Ferner seien folgende Definitionen gegeben:

```
void f1() {
  y = x;
  g();
  y = x;
  g();
}

void f2() {
  y = x;
  g();
  g();
}
```

Zeigen oder widerlegen Sie, dass unter den gegebenen Voraussetzungen folgende Tripel notwendigerweise gültig sein müssen:

a) $\{\mathtt{x} = (m-2) \cdot n \wedge \mathtt{y} = 2 \cdot n \wedge n > 0\}\ \mathtt{g}();\ \{\mathtt{y} \leq 0 \vee \mathtt{x} = m \cdot n\}$

b) $\{\mathtt{x} = m\}\ \mathtt{f1}();\ \{\mathtt{x} = 4m\}$

c) $\{\mathtt{x} = m\}\ \mathtt{f2}();\ \{\mathtt{x} = 3m\}$

**Aufgabe 4** [10 Punkte] **OCaml 1**

a) Schreiben Sie eine Funktion

**cycle** : 'a list −> 'a list ,

die das erste Element einer Liste ans Ende stellt.

**Beispiel:** cycle [3;4;2;1;6;7] = [4;2;1;6;7;3]

b) Schreiben Sie eine Funktion

**ncycle** : int −> 'a list −> 'a list .

Der Aufruf **ncycle** n 1 liefert das Ergebnis nach n-fachem Aufruf der Funktion cycle zurück.

**Beispiel:** **ncycle** 3 [3;4;2;1;6;7] = [1;6;7;3;4;2]

c) Schreiben Sie eine Funktion

**bubble** : 'a list −> 'a list .

Die Funktion **bubble** soll eine Liste $[\mathtt{x}_1; \ldots; \mathtt{x}_n]$ von vorne nach hinten durchlaufen und jeweils benachbarte Elemente $\mathtt{x}_i$ und $\mathtt{x}_{i+1}$ vertauschen, sofern sie nicht in aufsteigender Reihenfolge sind, d.h., sie werden vertauscht, wenn $\mathtt{x}_i > \mathtt{x}_{i+1}$ gilt.

**Beispiele:**
**bubble** [3;4;2;1;6;7] = [3;2;1;4;6;7]
**bubble** [100;2;2;4;6;7] = [2;2;4;6;7;100]

d) Verwenden Sie die Funktion **bubble**, um eine Funktion

**sort** : 'a list −> 'a list

zu implementieren, die eine Liste sortiert. Beachten Sie: Wenn man die Funktion **bubble** $k$ mal auf eine Liste anwendet, dann sind danach die letzten $k$ Elemente der Liste sortiert.

**Aufgabe 5** [12 Punkte] **OCaml 2**

Gegeben sei:

```
module type AddMul = sig
  type t
  val add : t -> t -> t
  val mul : t -> t -> t
end
```

a) Definieren Sie ein Modul Integer der Signatur AddMul für ganze Zahlen.

b) Ergänzen Sie die OCaml-Definition

```
module AddMulExpr(X : AddMul) = struct
  open X
  type expr =
    Const of t
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr

  (* Ihr Code hier *)
end
```

an der markierten Stelle wie folgt:

i) Definieren Sie eine Funktion

**const** : expr -> expr ,

die einen Ausdruck e vereinfacht, indem alle Ausdrücke, die keine Variablen enthalten, durch entsprechende Konstanten ersetzt werden.

**Beispiel:** **const** (Mul(Const x , Const y)) und Const (mul x y) werten sich zu dem gleichen Wert aus.

ii) Definieren Sie eine Funktion

**subst** : string -> expr -> expr -> expr ,

die in einem Ausdruck e alle Vorkommen einer Variablen x durch e' ersetzt, d.h., der Aufruf **subst** x e' e liefert den Ausdruck, der aus e entsteht, indem alle Vorkommen der Variablen x durch e' ersetzt werden.

**Beispiel:** Der Aufruf **subst** "x" (Var("y")) (Add(Var("x"),Var("y"))) wertet sich zu Add(Var("y"),Var("y")) aus.

iii) Schreiben Sie eine Funktion

**simp** : expr -> expr ,

die in einem Ausdruck e, falls vorhanden, mindestens einen Teilausdruck der Form Mul(e1,Add(e2,e3)) oder Mul(Add(e2,e3),e1) findet und ihn durch den Teilausdruck Add(Mul(e1,e2),Mul(e1,e3)) ersetzt, d.h., es soll an mindestens einer Stelle das Distributivgesetz angewendet werden.

iv) Schreiben Sie eine Funktion

**norm** : expr -> expr ,

die einen Ausdruck in einen gleichbedeutenden Ausdruck überführt, der lediglich eine Summe von Produkten ist, d.h., es soll solange das Distributivgesetz angewendet werden, bis diese gewünschte Form erreicht ist.

**Aufgabe 6** [5 Punkte] **Verifikation funktionaler Programme**

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,

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

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

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

für alle Listen l1 und l2 gilt.

# Anhang

Im Modul List definierte Funktionen, die verwendet werden dürfen:

```
val length     : 'a list -> int = <fun>
val nth        : 'a list -> int -> 'a = <fun>
val rev        : 'a list -> 'a list = <fun>
val append     : 'a list -> 'a list -> 'a list = <fun>
val map        : ('a -> 'b) -> 'a list -> 'b list = <fun>
val fold_left  : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
val fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b = <fun>
val for_all    : ('a -> bool) -> 'a list -> bool = <fun>
val exists     : ('a -> bool) -> 'a list -> bool = <fun>
val mem        : 'a -> 'a list -> bool = <fun>
val find       : ('a -> bool) -> 'a list -> 'a = <fun>
val filter     : ('a -> bool) -> 'a list -> 'a list = <fun>
val find_all   : ('a -> bool) -> 'a list -> 'a list = <fun>
```