Prof. Ralf Hinze
A. Dinges, M.Sc.
S. Schloßer, M.Sc.

Rheinland-Pfälzische Technische Universität Kaiserslautern Landau (RPTU)
Standort Kaiserslautern · Fachbereich Informatik · AG Programmiersprachen

Probeklausur Grundlagen der Programmierung

Dienstag, 19.12.2023

Exemplar-ID: 4711

Nachname:	
Vorname:	
Matrikelnummer:	

Hinweise:

1. Schreiben Sie **direkt bei Beginn** der Klausur Ihren Namen, Vornamen und Matrikelnummer **auf dieses Deckblatt**.
2. Achten Sie darauf, dass Ihre Klausur vollständig ist (14 Seiten)!
3. Sie haben 90 Minuten Zeit, die Klausur zu bearbeiten.
4. Schreiben Sie Ihre Lösungen gut lesbar mit Kugelschreiber oder Füllfederhalter **(kein Bleistift, kein Rotstift, kein Grünstift)**! Unleserliche Lösungen werden nicht korrigiert!
5. Sie dürfen **keine** eigenen Blätter verwenden. Lassen Sie diese Klausur in Ihrem eigenen Interesse geheftet; lose Klausurblätter werden nicht korrigiert!
6. Die Aufgaben **müssen** auf den jeweiligen Blättern bearbeitet werden. Sollte der Platz nicht ausreichen, so benutzen Sie die Rückseite des betreffenden Blattes oder die Zusatzblätter am Ende der Klausur. Sollte auch dies nicht ausreichen, bekommen Sie weitere Blätter bei der Aufsicht. Verweisen Sie in jedem Fall deutlich auf die Fortsetzungen Ihrer Aufgaben!
7. **Als Hilfsmittel zur Klausur zugelassen sind zwei beidseitig handbeschriebene A4-Blätter sowie Sprachwörterbücher.** Darüber hinaus sind keine Hilfsmittel erlaubt. Die Benutzung von Handys, Smartwatches und anderen elektronischen Geräten ist nicht gestattet. Handys müssen ausgeschaltet sein! Auf Ihrem Platz darf sich kein Rucksack o. ä. befinden. **Bei Verstößen gegen diese Regeln sowie bei Täuschungsversuchen wird die Klausur mit 0 Punkten gewertet. Täuschungsversuche werden darüber hinaus dem Prüfungsamt gemeldet.**
8. Lesen Sie vor der Bearbeitung einer Aufgabe den gesamten Aufgabentext sorgfältig durch! Die Aufgabenteile jeder Aufgabe bauen in der Regel nicht aufeinander auf. Sie können also in den meisten Fällen die Bearbeitung einer Aufgabe fortsetzen, auch wenn Sie einen Aufgabenteil nicht gelöst haben.
9. Der Hinweis *"Verwenden Sie keine Bibliotheksfunktionen"*, mit dem einige Aufgaben versehen sind, bezieht sich auf F# Funktionen, die nach dem Schema `Modulname.funktionsname` benannt sind, also z.B. `List.map`. Die vordefinierten Funktionen `not`, `min`, `max` und `@` sind von diesem Verbot nicht betroffen.

Aufgabe:	**1**	**2**	**3**	**4**	**5**	**6**
Punkte:						
Maximum:	20	20	20	20	20	20

Gesamtpunktzahl:	
Maximum:	120

Aufgabe 1 Dynamische Semantik (__ /20 Punkte)

a) Geben Sie jeweils an, zu welchem Wert die folgenden Ausdrücke auswerten (ohne Rechnung, Endergebnis genügt):

1. `let x = 10 in let y = 3 in x % y` **__ /10**

2. `if 1 * 2 < 1 + 2 then 3 + 4 else 3 * 4`

3. `(fun x -> 3 * x + 4) 5`

4. `let x = 4 in (let x = 3 * x in 2 * x) + x`

5. `let a = 2 in let f (x: Nat): Nat = a * x in f 3`

b) Werten Sie den Ausdruck `f x` bezüglich der Umgebung **__ / 10**

$$\delta := \{\mathtt{x} \mapsto 2, \mathtt{a} \mapsto 3, \mathtt{f} \mapsto \langle \{\mathtt{a} \mapsto 4\}, \mathtt{x}, \mathtt{a + x} \rangle\}$$

aus. Geben Sie einen vollständigen Beweisbaum auf Grundlage der Auswertungsregeln aus der Vorlesung an.
Tipp: Legen Sie das Blatt quer und zeigen Sie $\delta \vdash \mathtt{f\ x} \Downarrow \ldots$

Aufgabe 2 Statische Semantik (__ /20 Punkte)

Füllen Sie die Lücken, sodass sich **gültige Aussagen der Statischen Semantik** ergeben. Alle vorkommenden Zahlen sind vom Typ `Nat`.

a) $\emptyset \vdash$ `4711` : _____

b) $\emptyset \vdash$ _____ : `Nat`

c) $\emptyset \vdash$ **`if true then`** _____ **`else`** _____ : Nat

d) { `f` $\mapsto$ `Bool` $\rightarrow$ `Nat` } $\vdash$ `f` _____ : _____

e) { `x` $\mapsto$ _____ } $\vdash$ `x % 5` : _____

f) { `x` $\mapsto$ `Bool` } $\vdash$ **`fun`** `(x: Nat) -> x` : _____

g) $\emptyset \vdash$ **`let`** `f (x: Bool): Bool =` **`if`** `x` **`then false else`** `x` : _____

Aufgabe 3 Entwurfsmuster (__ /20 Punkte)

Lösen Sie diese Aufgabe **funktional**, d. h. `mutable` und `ref` dürfen in Ihrer Lösung nicht vorkommen. Verwenden Sie **keine Bibliotheksfunktionen!**

a) Schreiben Sie die Funktion `interval: Nat -> List<Nat>`, welche für eine gegebene natürliche Zahl `n` die Liste `[n; ...; 1]` berechnet. Gehen Sie **strikt nach Peano Entwurfsmuster** vor.

Beispiele:

```
interval 0 = []          interval 1 = [1]          interval 2 = [2; 1]
```

```
let rec interval (n: Nat): List<Nat> =
```

__ /5

b) Schreiben Sie die Funktion `ntimes<'a>: ('a -> 'a) -> 'a -> Nat -> 'a`, welche eine Funktion `f` vom Typ `'a -> 'a`, einen Startwert `x` vom Typ `'a` sowie eine natürliche Zahl `n` nimmt und die Funktion `f` auf `x` `n`-mal anwendet. Für `n = 0` soll `x` unverändert zurückgegeben werden. Gehen Sie **strikt nach Peano Entwurfsmuster** vor.

Beispiele:

```
ntimes (fun x -> x * x) 3 0 = 3          ntimes (fun x -> x * 2) 1 5 = 32
```

```
let rec ntimes<'a> (f: 'a -> 'a) (x: 'a) (n: Nat): 'a =
```

__ /5

Für die nächsten beiden Teilaufgaben verwenden wir folgenden Typen für Binärbäume:

```
type Tree = | Leaf | Node of Tree * Tree
```

c) Schreiben Sie die Funktion `complete: Nat -> Tree`, die für eine gegebene natürliche Zahl `n` den Baum der Höhe `n` berechnet, der die maximal mögliche Anzahl an Blättern besitzt (einen sogenannten *vollständigen Binärbaum* der Höhe `n`). Gehen Sie **strikt nach Peano Entwurfsmuster** vor. Sie dürfen **nur einen rekursiven Aufruf** verwenden.

Beispiele:

```
complete 0 = Leaf
complete 1 = Node (Leaf, Leaf)
complete 2 = Node (Node (Leaf, Leaf), Node (Leaf, Leaf))
```

```
let rec complete (n: Nat): Tree =
```

__/5

d) Schreiben Sie die Funktion `nodes: Tree -> Nat`, welche die Anzahl der inneren Knoten im gegebenen Baum bestimmt. Gehen Sie **nach Struktur Entwurfsmuster des Datentyps** vor.

Beispiele:

```
nodes Leaf = 0
nodes Node (Leaf, Leaf) = 1
nodes Node (Leaf, Node (Leaf, Leaf)) = 2
```

```
let rec nodes (t: Tree): Nat =
```

__/5

Aufgabe 4 Mengen (__ /20 Punkte)

Lösen Sie diese Aufgabe **funktional**, d. h. `mutable` und `ref` dürfen in Ihrer Lösung nicht vorkommen. Verwenden Sie **keine Bibliotheksfunktionen!**

Wir betrachten folgenden Typen, um Mengen zu modellieren:

```
type Set<'a> = 'a -> Bool
```

Ist `m` vom Typ `Set<'a>` gegeben und `x` ein Element vom Typ `'a`, so ist `m x = true` genau dann, wenn $x \in m$ gilt.

Hinweis: Sie dürfen davon ausgehen, dass Werte vom Typ `'a` mit dem Gleichheitsoperator '=' vergleichbar sind.

a) Geben Sie die leere Menge $\emptyset$ an:

```
let empty<'a> : Set<'a> =
```

__ /5

b) Schreiben Sie eine Funktion `add<'a>: 'a -> Set<'a> -> Set<'a>`, die das gegebene Element in die gegebene Menge einfügt.

```
let add<'a> (elem: 'a) (m: Set<'a>): Set<'a> =
```

__ /5

c) Schreiben Sie eine Funktion `contains<'a>: 'a -> Set<'a> -> Bool`, die überprüft, ob das gegebene Element in der gegebenen Menge enthalten ist.

```
let contains<'a> (elem: 'a) (m: Set<'a>): Bool =
```

__/5

d) Schreiben Sie eine Funktion `difference<'a>: Set<'a> -> Set<'a> -> Set<'a>`, die die Differenzmenge zweier Mengen berechnet. Die Differenzmenge ist die Menge aller Elemente, die in der ersten Menge enthalten sind, aber nicht in der zweiten. Formal ausgedrückt bedeutet dies: $x \in \texttt{difference m1 m2} \Leftrightarrow x \in m1 \land x \notin m2$.

```
let difference<'a> (m1: Set<'a>) (m2: Set<'a>): Set<'a> =
```

__/5

Aufgabe 5 Bäume (__ /20 Punkte)

Lösen Sie diese Aufgabe **funktional**, d. h. `mutable` und `ref` dürfen in Ihrer Lösung nicht vorkommen. Verwenden Sie **keine Bibliotheksfunktionen!**

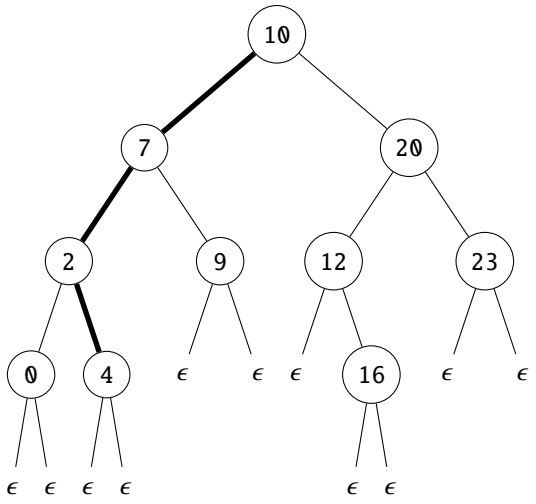
Wir betrachten folgende Typen:

```
type Tree<'a> = | Leaf | Node of Tree<'a> * 'a * Tree<'a>

type Path = | End | Left of Path | Right of Path
```

Bäume des Typs `Tree<'a>` speichern ihre Daten in den Knoten. Der Typ `Path` repräsentiert Pfade im Baum. Pfade beginnen immer in der Wurzel des Baumes.

Beispiel: Rechts sehen Sie den Code zu dem hier dargestellten Baum und dem hervorgehobenen Pfad mit den Knoten 10, 7, 2, 4.




```
let exTree: Tree<Nat> =
  Node (
    Node (
      Node (
        Node (Leaf, 0, Leaf),
        2,
        Node (Leaf, 4, Leaf)
      ),
      7,
      Node (Leaf, 9, Leaf)
    ),
    10,
    Node (
      Node (
        Leaf,
        12,
        Node (Leaf, 16, Leaf)
      ),
      20,
      Node (Leaf, 23, Leaf)
    )
  )

let exPath: Path = Left (Left (Right End))
```

a) Schreiben Sie eine Funktion `lookup: Tree<'a> -> Path -> Option<'a>`, die den Wert des Knotens bestimmt, zu dem der gegebenen Pfad führt. Wenn der Pfad zu keinem Knoten führt, soll `None` zurückgegeben werden.

Beispiele:

```
lookup exTree End = Some 10
lookup exTree exPath = Some 4
lookup exTree (Left (Right (Left End))) = None
lookup exTree (Left (Right (Left (Left End)))) = None
```

```
let rec lookup (t: Tree<'a>) (p: Path): Option<'a> =
```

__/6

b) Schreiben Sie eine Funktion `update: Tree<'a> -> Path -> 'a -> Tree<'a>`, die den Baum berechnet, der entsteht, wenn man im gegebenen Baum den Wert des Knotens, zu dem der Pfad führt, zum gegebenen Wert `x` ändert. Führt der Pfad zu keinem Knoten, soll der Baum unverändert zurückgegeben werden.

Beispiele:

```
let t = Node (Node (Leaf, 2, Leaf), 3, Leaf)
update t End 4 = Node (Node (Leaf, 2, Leaf), 4, Leaf)
update t (Left End) 4 = Node (Node (Leaf, 4, Leaf), 3, Leaf)
update t (Right End) 4 = Node (Node (Leaf, 2, Leaf), 3, Leaf)
```

```
let rec update (t: Tree<'a>) (p: Path) (x: 'a): Tree<'a> =
```

__/6

c) Schreiben Sie eine Funktion `search: Tree<Nat> -> Nat -> Option<Path>`, die im gegebenen *Suchbaum* einen Pfad zum gegebenen Wert `x` bestimmt. Befindet sich `x` nicht im Baum, soll `None` zurückgegeben werden.

Zur Erinnerung: Ein Suchbaum ist ein Baum, bei dem für jeden Knoten die Elemente im linken Teilbaum kleiner und im rechten Teilbaum größer sind als das Element im Knoten selbst. Der Baum `exTree` oben ist ein Suchbaum.

Nutzen Sie aus, dass es sich um einen Suchbaum handelt.

Beispiel:

```
search exTree 10 = Some End
search exTree 4 = Some (Left (Left (Right (End))))
search exTree 3 = None
```

```
let rec search (t: Tree<Nat>) (x: Nat): Option<Path> =
```

__/8

Aufgabe 6 Reguläre Ausdrücke (__ /20 Punkte)

a) Wir betrachten den regulären Ausdruck

$$\mathtt{b} \cdot \mathtt{a} \cdot \Big((\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{a}) \mid (\mathtt{b} \cdot \mathtt{a} \cdot \mathtt{b}) \mid (\mathtt{b} \cdot \mathtt{b})\Big)^* \cdot \mathtt{a}$$

über dem Alphabet $\{\mathtt{a}, \mathtt{b}\}$.

Kreuzen Sie an, ob die folgenden Wörter in der von dem Ausdruck beschriebenen Sprache enthalten sind oder nicht. Für richtige Antworten erhalten Sie einen Punkt, für falsche Antworten wird ein Punkt abgezogen. Nicht markierte Zeilen wirken sich nicht auf die Punktzahl aus. Diese Teilaufgabe wird mit mindestens 0 Punkten bewertet.

__ /6

Wort	enthalten	nicht enthalten
bababbbababba		
ababaaababaab		
baababababbbba		
babbbbbbbbbba		
bababaababbba		
bababababaaa		

b) Geben Sie für die folgenden Beschreibungen in natürlicher Sprache einen regulären Ausdruck über dem Alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ an:

1. Die Sprache, deren Wörter genau ein b enthalten. **__ /8**

2. Die Sprache, deren Wörter kein a enthalten.

3. Die Sprache, deren Wörter aus genau drei Buchstaben bestehen.

4. Die leere Sprache.

c) Bestimmen Sie die folgenden Rechtsfaktoren. Geben Sie in der Rechnung **jeweils den ersten Schritt explizit** an, nachfolgende Zwischenschritte dürfen Sie zusammenfassen.

Alphabet: $\{a, b, c\}$

__ /6

$a \setminus \Big(\big(a \cdot (b \mid c)\big)^* \cdot \big(b \cdot (c \mid a)\big)\Big) =$

$b \setminus \big(a \mid (b \cdot b)\big)^* =$

$c \setminus \Big((a \mid \epsilon) \cdot \big((c \mid b)^* \cdot (a \cdot c)^*\big)\Big) =$

Fortsetzung von Aufgabe ____