Korrekturexemplar

Fachbereich
Informatik

# Klausur Informatik 1
# 22. März 2000

Mahr / Lemke / Weisweber
Tschirley / Hofstedt / Wieland

**Name:** ................................

**Vorname:** ................................

**Matr.-Nr.:** ................................

Bearbeitungszeit: 120 Minuten

**Wichtige Hinweise**:

- Benutzen Sie für die Lösung der Aufgaben **nur** das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier geschrieben werden, können nicht gewertet werden.** Schreiben Sie Ihre Lösung auch auf die Rückseiten der Blätter! Weiteres Papier kann bei den Tutoren angefordert werden.
- Tragen Sie jetzt (vor Beginn der eigentlichen Bearbeitungszeit!) auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein.
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen können nicht gewertet werden.
- Schreiben Sie **nicht** mit Bleistift!
- Schreiben Sie nur in **blau** oder **schwarz**!
- Bei der Implementation von OPAL-Funktionen brauchen *keine* Importe mit angegeben zu werden, wenn Funktionen aus dem gegebenen Teil der Bibliotheca Opalica (oder aus der Struktur `Nat`) oder sonstigen (in den Aufgaben) gegebenen Strukturen benutzt werden!
- Die OPAL-Implementierungen müssen nicht dokumentiert werden.

Viel Erfolg!

| Aufgabe | Punkte | erreicht |
|---|---|---|
| 1 | 12 | |
| 2 | 21 | |
| 3 | 7 | |
| 4 | 9 | |

Korrekturexemplar

Zur Benutzung für die OPAL-Aufgaben, hier einige Strukturen aus der Bibliotheca Opalica (nicht vollständig!):

```
SIGNATURE Bool
IMPORT BOOL ONLY bool
FUN true false :                bool
FUN not        : bool        -> bool
FUN and or     : bool ** bool -> bool

SIGNATURE Seq[data]
SORT data
TYPE seq == <>
            ::(ft: data, rt: seq)

SIGNATURE SeqMap[from,to]
SORT from to
IMPORT Seq[from] ONLY seq
       Seq[to]   ONLY seq
-- map(f)(x1 :: ... :: xn :: <>) == f(x1) :: ... :: f(xn) :: <>
FUN map : (from -> to) -> seq[from] -> seq[to]

SIGNATURE SeqReduce[from,to]
SORT from to
IMPORT Seq[from] ONLY seq
-- reduce(op,y)(x1 :: ... :: xn :: <>) ==
-- (x1 op ... op (xn op y)...) ==
-- op(x1,op(...,op(xn,y)...))
FUN reduce : (from ** to -> to) ** to -> seq[from] -> to

SIGNATURE SeqFilter[data]
SORT data
IMPORT Seq[data] ONLY seq
       Bool      ONLY bool
-- filter(p) liefert eine Funktion, die aus einer Sequenz alle die Elemente entfernt,
-- fuer die p nicht true liefert.
FUN filter : (data -> bool) -> seq[data] -> seq[data]

SIGNATURE SeqZip[from1,from2,to]
SORT from1 from2 to
IMPORT Seq[from1] ONLY seq
       Seq[from2] ONLY seq
       Seq[to]    ONLY seq
-- zip(f)(x1 :: x2 :: ... :: xn :: <>, y1 :: y2 :: ... :: yn :: <>) ==
-- f(x1,y1) :: f(x2,y2) :: ... :: f(xn,yn) :: <>
-- zip(f) ist nur auf Paaren von gleich langen Sequenzen definiert!
FUN zip : (from1 ** from2 -> to) -> seq[from1] ** seq[from2] -> seq[to]

SIGNATURE Compose[from,middle,to]
SORT from middle to
-- g o f ist die Verkettung zweier Funktionen und liefert eine Funktion,
-- die auf ein Argument erst f und auf das Ergebnis dann g anwendet.
FUN o : (middle -> to) ** (from -> middle) -> from -> to
-- f ; g == g o f
FUN ; : (from -> middle) ** (middle -> to) -> from -> to

SIGNATURE Pair[from1,from2]
SORT from1 from2
-- pair ist eine Sorte von Paaren. Jedes Paar hat einen Wert der Sorte from1 an erster Stelle (1st)
-- und einen Wert der Sorte from2 an zweiter Stelle (2nd). Der Konstruktor &, angewandt auf zwei
-- solche Werte, liefert ein Element der Sorte pair[from1,from2].
TYPE pair == &(1st: from1, 2nd: from2)

IMPLEMENTATION SeqMap[from,to]
IMPORT Seq[from] COMPLETELY
       Seq[to]   COMPLETELY
DEF map(f)(<>) == <>
DEF map(f)(x :: xs) == f(x) :: map(f)(xs)

IMPLEMENTATION SeqReduce[from,to]
IMPORT Seq[from] COMPLETELY
DEF reduce(f,y)(<>) == y
DEF reduce(f,y)(x :: xs) == f(x,reduce(f,y)(xs))

IMPLEMENTATION SeqFilter[data]
DEF filter(p)(<>) == <>
DEF filter(p)(x :: xs) == IF p(x) THEN x :: rest ELSE rest FI
                          WHERE rest == filter(p)(xs)

IMPLEMENTATION SeqZip[from1,from2,to]
IMPORT Seq[from1] COMPLETELY
       Seq[from2] COMPLETELY
       Seq[to]    COMPLETELY
DEF zip(f)(<>,<>) == <>
DEF zip(f)(x :: xs, y :: ys) == f(x,y) :: zip(f)(xs,ys)

IMPLEMENTATION Compose[from,middle,to]
DEF (g o f)(x) == g(f(x))
DEF (f ; g) == (g o f)
```

Korrekturexemplar

# Aufgabe 1 (12 Punkte): Funktionen höherer Ordnung

Gegeben sei folgende Signatur:

```
SIGNATURE Predicate
IMPORT Bool       ONLY bool
       Seq  ONLY seq

-- Negation
FUN !  : (nat -> bool) -> (nat -> bool)

-- Konjunktion, Disjunktion
FUN && || : (nat -> bool) ** (nat -> bool) -> (nat -> bool)

-- Abgeleitete Funktionen: Implikation, Exklusive Disjunktion (Ungleichheit)
FUN => != : (nat -> bool) ** (nat -> bool) -> (nat -> bool)

-- Sequenz-Praedikate: Universelle und Existentielle Quantifikation
FUN forall exists : (nat -> bool) -> (seq[nat] -> bool)
```

Eine totale Funktion mit Funktionalität `nat -> bool` kann als Prädikat auf natürlichen Zahlen angesehen werden. Eine totale Funktion mit der Funktionalität `seq[nat] -> bool` ist demnach ein Prädikat auf Sequenzen von natürlichen Zahlen. Eine totale Funktion der Funktionalität `bool -> bool` oder `bool ** bool -> bool` (siehe Struktur `Bool`) ist ein Basis-Prädikat.

1. **Negation, Konjunktion, Disjunktion (4 Punkte)** Die Funktionen höherer Ordnung `!`, `&&` und `||` bekommen Prädikate als Argumente und liefern Prädikate als Resultate zurück. Definieren Sie die Funktionen `!`, `&&` und `||` nach gegebenen Funktionalitäten so, dass `!` die Negation eines Prädikats, `&&` die Konjunktion und `||` die Disjunktion zweier Prädikate liefern!

Negation:

| p(x) | !(p)(x) |
|---|---|
| true | false |
| false | true |

Konjunktion:

| p(x) | q(x) | (p && q)(x) |
|---|---|---|
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

Disjunktion:

| p(x) | q(x) | (p \|\| q)(x) |
|---|---|---|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

2. **Abgeleitete Funktionen höherer Ordnung (4 Punkte)** Definieren Sie die Funktion `=>` so, dass sie zwei Prädikate als Argumente bekommt und ein Prädikat liefert, das die Implikation der Argumente ist! Die Implikation liefert für einen Wert genau nur dann `false`, wenn das erste Prädikat für diesen Wert `true` liefert und das zweite `false`.

   Definieren Sie weiterhin eine Funktion `!=` so, dass sie zwei Prädikate als Argumente bekommt und ein Prädikat liefert, das ein logisches XOR (Ungleichheit) der Argumente ist! Nur, wenn die zwei Prädikate für einen Wert unterschiedliche Werte liefern, liefert das Ergebnis-Prädikat für diesen Wert `true`.

   Benutzen Sie hierfür nur andere Funktionen höherer Ordnung!

Implikation:

| p(x) | q(x) | (p => q)(x) |
|---|---|---|
| false | false | true |
| false | true | true |
| true | false | false |
| true | true | true |

XOR:

| p(x) | q(x) | (p != q)(x) |
|---|---|---|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | false |

3. **Funktionen höherer Ordnung und Sequenz-Prädikate (4 Punkte)** Definieren Sie die Funktionen `forall` und `exists`, die jeweils ein Prädikat auf natürlichen Zahlen als Argument bekommen, und ein Prädikat auf Sequenzen von natürlichen Zahlen liefern!

   Das Sequenzen-Prädikat `forall(p)` soll genau dann `true` liefern, wenn `p` für alle Elemente der Sequenz, auf die das Prädikat angewendet wird, `true` liefert.

   Das Sequenzen-Prädikat `exists(p)` soll genau dann `true` liefern, wenn es mindestens ein Element in der Sequenz gibt, für das `p` `true` liefert.

   Benutzen Sie auch hierfür keine Basis-Prädikate (`and`/`or`/`not`/`IF`)!

Korrekturexemplar

Korrekturexemplar

## Aufgabe 2 (21 Punkte): Datentypen und Funktionale

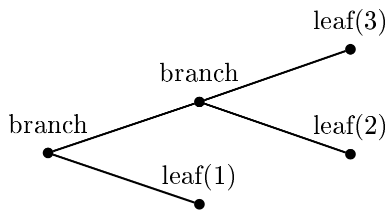
Gegeben seien folgende Signaturen:

```
SIGNATURE Tree
IMPORT Nat ONLY nat

TYPE tree == empty
             leaf(value: nat)
             branch(left: tree, right: tree)

SIGNATURE TreeMap
IMPORT Tree ONLY tree

FUN map : (nat -> nat) -> tree -> tree
```

Damit ist ein Tree der Sorte `tree` aus der Struktur `Tree` eine rekursive Datenstruktur, die bestehen kann aus:

- einem leeren Tree (`empty`) ohne 'Blätter',
- einem 'Blatt' (`leaf`) mit einem Wert (`value`) der Sorte `nat`, oder
- einer 'Verzweigung' (`branch`), die einen linken (`left`) und einen rechten (`right`) Tree enthält.

*Beispiel:* `branch(leaf(1),branch(leaf(2),leaf(3))) ==`




**1. Induzierte Signatur (7 Punkte)** Welche Signatur wird von der Typdeklaration für `tree` in der Signatur `Tree` induziert? Wie nennt man die einzelnen induzierten Elemente?

**2. `map` auf Bäumen (4 Punkte)** Implementieren Sie die Funktion `map` der Struktur `TreeMap`, wobei sich die Funktion analog zu der Funktion `map` der Struktur `SeqMap` verhalten soll!

Die an `map` übergebene Funktion soll also auf jedes Blatt des Tree angewendet werden. Damit erhält man einen Tree gleicher Form (siehe Beispiel), der an den Blättern die Resultate der Funktionsanwendungen auf die entsprechenden Werte im Ausgangs-Tree enthält.

*Beispiel:*

```
map(_ + 1)(branch(leaf(1),branch(leaf(2),leaf(3)))) ==
           branch(leaf(2),branch(leaf(3),leaf(4))))
```

**3. Umwandlung von Bäumen in Sequenzen (5 Punkte)** Definieren Sie eine Funktion `toSeq`, die aus einem Tree eine Sequenz der Daten des Tree macht! Die Daten der Blätter des Tree sollen sich in der Ergebnis-Sequenz in der gleichen links-nach-rechts Reihenfolge wiederfinden. Geben Sie die Funktionalität der Funktion `toSeq` mit an!

*Beispiel:* `toSeq(branch(leaf(1),branch(leaf(2),leaf(3)))) == 1 :: 2 :: 3 :: <>`

**4. Anwenden von Funktionalen (5 Punkte)** Gegeben sei eine Struktur mit folgender Signatur:

```
SIGNATURE Sort[data]
SORT data
IMPORT Seq[data] ONLY seq
       Bool      ONLY bool
-- Die Funktion sort bekommt eine Ordnungsrelation (wie beispielsweise < fuer
-- die natuerlichen Zahlen) auf der Sorte data, also eine Boolsche Funktion
-- auf Paaren dieser Sorte, als Argument und liefert eine Funktion, die eine
-- Sequenz von Daten dieser Sorte nach dieser Ordnungsrelation sortiert.
-- Beispiel: sort(<)(3 :: 2 :: 1 :: <>) == 1 :: 2 :: 3 :: <>
FUN sort : (data ** data -> bool) -> seq[data] -> seq[data]
```

Korrekturexemplar

Gegeben sei weiterhin folgende Struktur:

```
SIGNATURE F
IMPORT Nat      ONLY nat
       Seq[nat] ONLY seq

FUN f : seq[nat] ** seq[nat] -> seq[nat]

IMPLEMENTATION F
IMPORT Nat                 ONLY nat <
       Pair[nat,nat]       ONLY pair & 1st 2nd
       SeqZip[nat,nat,pair] ONLY zip
       SeqMap[pair,nat]    ONLY map
       Sort[nat]           ONLY sort

DEF f(xs,ys) ==
    LET
      r1 == zip(&)(xs,ys)
      r2 == sort(\\ x, y. 2nd(x) < 2nd(y))(r1)
    IN
      map(1st)(r2)
```

- Geben Sie geeignete Anwendungsbedinungen an, bei denen die Funktion `f` total ist!

- Geben Sie das Ergebnis des folgenden Aufrufs an:

  `f(3 :: 5 :: 1 :: 4 :: 7 :: 4 :: <>, 2 :: 5 :: 1 :: 4 :: 3 :: 6 :: <>) ==`

- Wie müßte das Argument `x` lauten, damit folgende Gleichung gilt:

  `f(1 :: 2 :: 3 :: 4 :: 5 :: <>, x) == 5 :: 3 :: 1 :: 4 :: 2 :: <>`

Korrekturexemplar

Korrekturexemplar

# Aufgabe 3 (7 Punkte): Schaltnetzentwurf

Folgendes Problem ist mit einem Schaltnetz zu lösen:
Ein Bauer arbeitet normalerweise jeden Tag. Allerdings arbeitet er am Liebsten nicht bei Regen, nicht bei Kälte, nicht an Tagen, an denen seine Lieblingsfußballmannschaft spielt und nicht an Sonntagen. Doch da das zu wenig Arbeitstage ergeben würde, muß er zum Teil auch an solchen Tagen arbeiten, an denen er eigentlich lieber nicht arbeiten möchte. Dabei richtet er sich nach folgenden Grundsätzen:

1. An Tagen, an denen es regnet, arbeitet er, falls es keine Sonntage sind und es nicht kalt ist.
2. Fußballspiele verpaßt er, wenn es nicht regnet und es kein Sonntag ist.
3. An folgenden Tagen ist es ihm egal, ob er arbeiten muß oder nicht:
   - An Sonntagen, an denen es nicht regnet und nicht kalt ist und kein Fußballspiel ausgetragen wird
   - und an solchen Tagen, an denen es kalt ist, die aber keine Sonntage sind und an denen nicht außerdem ein Fußballspiel läuft.

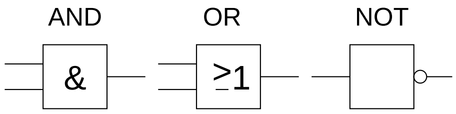
**1. Wertetabelle (2 Punkte)** Tragen Sie in die folgende Wertetabelle die oben gegebenen Aussagen ein. Es gelten die Bezeichnungen R = Regen, K = Kälte, F = Fußball, S = Sonntag, Y = Arbeiten.

**Falls Sie sich verschreiben**, nutzen Sie bitte die weiteren Spalten. Markieren Sie deutlich, welche Ergebnisspalte gewertet werden soll.

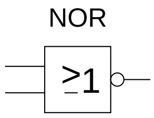
| R | K | F | S | Y | Y | Y | Y |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | | | |
| 0 | 0 | 0 | 1 | | | | |
| 0 | 0 | 1 | 0 | | | | |
| 0 | 0 | 1 | 1 | | | | |
| 0 | 1 | 0 | 0 | | | | |
| 0 | 1 | 0 | 1 | | | | |
| 0 | 1 | 1 | 0 | | | | |
| 0 | 1 | 1 | 1 | | | | |
| 1 | 0 | 0 | 0 | | | | |
| 1 | 0 | 0 | 1 | | | | |
| 1 | 0 | 1 | 0 | | | | |
| 1 | 0 | 1 | 1 | | | | |
| 1 | 1 | 0 | 0 | | | | |
| 1 | 1 | 0 | 1 | | | | |
| 1 | 1 | 1 | 0 | | | | |
| 1 | 1 | 1 | 1 | | | | |

**2. KV-Diagramm (2 Punkte)** Zeichnen Sie das KV-Diagramm zu der Wertetabelle aus Teil 1. Ermitteln Sie aus dem KV-Diagramm die disjunktive Normalform der die Wertetabelle erfüllenden Funktion.

**3. Schaltnetz (1 Punkte)** Zeichnen Sie das minimierte Schaltnetz zu der in Teil 2 ermittelten Funktion. Benutzen Sie dabei so wenig Gatter wie möglich. Verwenden Sie ausschließlich die nachfolgend abgebildeten Gatter.




**4. Schaltnetz mit NOR-Gattern (2 Punkte)** Leiten sie aus der Funktion, die Sie in Teil 3 erhalten haben, eine Form ab, die lediglich NOR-Verknüpfungen verwendet. Zeichnen Sie das entsprechende Schaltnetz. Benutzen Sie dabei so wenig Gatter wie möglich. Verwenden Sie ausschließlich die abgebildeten NOR-Gatter mit 2 Eingängen.



Korrekturexemplar

Korrekturexemplar

## Aufgabe 4 (9 Punkte): Zahlendarstellungen

**Kommentieren Sie Ihren Lösungsweg – Lösungen, bei denen der Lösungsweg nicht erkennbar ist, können nicht gewertet werden!**

1. **(2 Punkte)** Berechnen Sie die Lösung von $435_{(7)} + 44_{(5)}$ und geben Sie das Ergebnis im Binärsystem (Basis 2) an.

2. **(4 Punkte)** Geben Sie die beiden Dezimalzahlen $a = -50,4375$ und $b = 2,1875$ als binäre Festkommazahlen mit 8 Vor- und 4 Nachkommastellen im Zweierkomplement an.

   Berechnen Sie dann *binär*

   - $a + b$ und
   - $a \cdot b$

   und konvertieren Sie das Ergebnis jeweils in das Dezimalsystem. Rechnen Sie ausschließlich mit vorzeichenbehafteten Zahlen!

3. **(3 Punkte)** Gegeben ist die mit 16 Bit gespeicherte Gleitkommazahl 1100 1011 0011 1001, die nach dem in der LV vorgestellten, an dem Gleitkommazahlenformat der IEEE orientierten Schema codiert ist. Berechnen Sie den dargestellten Dezimalbruch *ohne Verlust an Genauigkeit*.

   **Hinweis:** Das Format sieht 1 Vorzeichenbit, 4 Bit für den mit dem Bias 7 versehenen Exponenten und 11 Bit für den Nachkommaanteil der Mantisse in dieser Folge vor.

Korrekturexemplar