Prof. Dr. Ralf Hinze
Alexander Dinges, M.Sc.
Sebastian Schloßer, M.Sc.

RPTU
Fachbereich Informatik
AG Programmiersprachen

Übungsblatt 4: Grundlagen der Programmierung (WS 2023/24)

Ausgabe: 21. November 2023
Abgabe: 27./28./29. November 2023, siehe Homepage

Klausuranmeldung Denken Sie daran, die Klausur und die Vorleistung im Prüfungsamt anzumelden!

Aufgabe 1 Datentypen (Präsenzaufgabe)

Motivation: Um Programmieraufgaben im ExClaim-System testen zu können, müssen wir die darin verwendeten Typdefinitionen vorgeben. In dieser Aufgabe sollen Sie (rekursive) Varianten und Records wiederholen und selbst entsprechende Typen definieren, um Sachverhalte zu modellieren. Sie können sich an den Vorlesungsfolien 281 bis 373 sowie am Skript Kapitel 4.1 und 4.2 orientieren.

a) Varianten

1. Definieren Sie einen Variantentyp zur Modellierung von Tierarten, der die Ausprägungen `Hund`, `Katze` und `Maus` annehmen kann.

   Zusätzlich soll ein Variantentyp für Lebewesen definiert werden: Bei einem Lebewesen kann es sich entweder um ein Tier einer der oben genannten Tierarten handeln, oder um ein Lebewesen, das kein Tier ist.

2. Schreiben Sie eine Funktion `eineMaus: Tierart -> Bool`, die prüft, ob es sich bei der übergebenen Tierart um eine Maus handelt.

3. Schreiben Sie eine Funktion `eineKatze: Lebewesen -> Bool`, die prüft, ob es sich beim übergenenen Lebewesen um eine Katze handelt.

4. Schreiben Sie eine Funktion `mindestensEinTier: Lebewesen -> Lebewesen -> Bool`, die prüft, ob es sich bei mindestens einem der beiden Argumente um ein Tier handelt.

b) Rekursive Varianten

1. Wiederholen Sie den Typ `Nats` für Listen natürlicher Zahlen

2. Schreiben Sie eine Funktion `findMax`, welche die größte Zahl in einer Liste natürlicher Zahlen zurückgibt.

3. Schreiben Sie eine Funktion `plusOne`, die alle Zahlen in der Liste um eins erhöht.

c) Records

1. Schreiben Sie einen Record-Typ, um Eigenschaften von Büchern zu speichern. Gesichert werden soll der Titel, der Name der Autorin/des Autors, das Veröffentlichungsjahr sowie die ISBN.

   Verwenden Sie den Record, um das Buch „Harry Potter and the Philosopher's Stone“ von „J. K. Rowling“ aus dem Jahr 1997 mit der ISBN 9780747532743 zu erfassen.

2. Wo liegt der Vorteil gegenüber einem entsprechenden Quadrupel?

Aufgabe 2 Kalenderdaten (Einreichaufgabe, 7 Punkte)

Motivation: In dieser Aufgabe sollen Sie sich mit Variantentypen und Records beschäftigen. Sie können sich an den Vorlesungsfolien 281 bis 327 sowie am Skript Kapitel 4.1 und 4.2 orientieren.

Schreiben Sie Ihre Lösungen in die Datei `Dates.fs` *aus der Vorlage* `Aufgabe-4-2.zip`.

Wir repräsentieren ein Kalenderdatum durch folgende Datentypen:

```
type Weekday = | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday
type Date = { year: Nat; month: Nat; day: Nat; weekday: Weekday }
```

Ein Datum besteht also aus je drei Zahlen für das Jahr, den Monat und den Tag, sowie dem Wochentag.

Hinweis: Einige Teilaufgaben können Sie mit Hilfe der Funktionen aus vorherigen Teilaufgaben lösen.

a) Schreiben Sie eine Funktion `nextWeekday: Weekday -> Weekday`, die einen Wochentag nimmt und den nächsten Wochentag zurückgibt.

   Beispiel:
   ```
   nextWeekday Monday = Tuesday
   ```

b) Schreiben Sie eine Funktion `isLeapYear: Nat -> Bool`, die eine natürliche Zahl nimmt, die eine Jahreszahl darstellen soll, und prüft, ob diese ein Schaltjahr[1] ist.

   Beispiele:
   ```
   isLeapYear 2023 = false
   isLeapYear 2024 = true
   ```

c) Schreiben Sie eine Funktion `daysInMonth: Nat -> Nat -> Nat`, die eine Jahreszahl sowie eine Monatszahl nimmmt und zurückgibt, wie viele Tage es im Monat gibt. Ist der Monatswert nicht sinnvoll, ist es egal, was zurückgegeben wird.

   Beispiel:
   ```
   daysInMonth 2021 2 = 28
   daysInMonth 2020 2 = 29
   daysInMonth 2023 11 = 30
   ```

d) Schreiben Sie eine Funktion `nextDate: Date -> Date`, die ein Datum nimmt und das nächste Datum zurückgibt. Sie müssen nicht prüfen, ob die Eingabe ein gültiges Datum ist.

   Beispiel:
   ```
   nextDate { year = 2023N; month = 11N; day = 21N; weekday = Tuesday } =
            { year = 2023N; month = 11N; day = 22N; weekday = Wednesday }
   nextDate { year = 2023N; month = 12N; day = 31N; weekday = Sunday } =
            { year = 2024N; month = 1N;  day = 1N;  weekday = Monday }
   ```

e) Schreiben Sie eine Funktion `nextDateN: Date -> Nat -> Date`, die ein Datum und eine natürliche Zahl nimmt und das Datum zurückgibt, das nach der angegebenen Anzahl von Tagen folgt. Sie müssen nicht prüfen, ob die Eingabe ein gültiges Datum ist.

   Tipp: Nutzen Sie das Peano-Entwurfsmuster und Ihre Funktion aus der vorherigen Teilaufgabe.

f) *Freiwillige Zusatzaufgabe:* Schreiben Sie eine Funktion `validateWeekday: Date -> Bool option`, die ein Datum nimmt und prüft, ob der Wochentag mit dem Datum übereinstimmt. Sie soll darüber hinaus nichts prüfen.

   Beispiel:
   ```
   validateWeekday { year = 2023N; month = 11N; day = 21N; weekday = Tuesday } = true
   validateWeekday { year = 2023N; month = 11N; day = 21N; weekday = Wednesday } = false
   ```

[1] https://de.wikipedia.org/wiki/Schaltjahr#Gregorianischer_Kalender

Aufgabe 3 Listen natürlicher Zahlen (Einreichaufgabe, 6 Punkte)

Motivation: In dieser Aufgabe sollen Sie sich mit rekursiven Variantentypen beschäftigen. Sie können sich an den Vorlesungsfolien 332 bis 373 sowie am Skript Kapitel 4.2.2 orientieren.

Schreiben Sie Ihre Lösungen in die Datei `Nats.fs` *aus der Vorlage* `Aufgabe-4-3.zip`.

Bevor wir nächste Woche den in F# eingebauten Typ für Listen kennenlernen, arbeiten wir diese Woche zunächst mit einem selbst definierten Typ für Listen natürlicher Zahlen. Alles was wir dazu brauchen, sind rekursive Varianten und Tupel. Sie kennen den Typ bereits aus der Vorlesung:

```
type Nats = | Nil | Cons of Nat * Nats
```

Hinweis: Wenn Sie im Internet nach Teilen der Aufgabenstellung suchen, werden Sie vielleicht auf das `List` *F#-Modul stoßen, das allerdings mit dem in F# eingebauten Typ für Listen arbeitet. Dieses Modul werden wir auf einem späteren Übungsblatt vorstellen, hier dürfen Sie es in Ihrer Lösung jedoch* ***nicht*** *verwenden.*

Wir verwenden bei einigen Teilaufgaben die Beispielliste:

```
let ex = Cons (2N, Cons (4N, Cons (3N, Cons(4N, Cons(2N, Cons (1N, Nil))))))
```

a) Schreiben Sie eine Funktion `double: Nats -> Nats`, die jeden Eintrag der gegebenen Liste verdoppelt. Beispiele:

```
double Nil = Nil
double (Cons (1N, Cons (2N, Nil))) = Cons (2N,Cons (4N,Nil))
double ex = Cons (4N, Cons (8N, Cons (6N, Cons (8N, Cons (4N, Cons (2N, Nil))))))
```

b) Schreiben Sie eine Funktion `isSorted: Nats -> Bool`, die prüft, ob die gegebene List *aufsteigend* sortiert ist.

Beispiele:

```
isSorted Nil = true
isSorted (Cons (1N, Cons (2N, Nil))) = true
isSorted (Cons (1N, Cons (1N, Nil))) = true
isSorted (Cons (2N, Cons (1N, Nil))) = false
isSorted ex = false
```

c) Schreiben Sie eine Funktion `filter: (Nat -> Bool) -> Nats -> Nats`, die ein Prädikat auf den natürlichen Zahlen (also eine Funktion, die eine natürliche Zahl nimmt und prüft, ob diese eine bestimmte Bedingung erfüllt) und eine Liste von natürlichen Zahlen nimmt und und die Zahlen zurückgibt, die das Prädikat erfüllen.

Beispiele:

```
filter p Nil = Nil
filter (fun n -> n%2=0) (Cons (2N, Cons (4N, Nil))) = Cons (2N, Cons (4N, Nil))
filter (fun n -> n%2=0) (Cons (1N, Cons (6N, Nil))) = Cons (6N, Nil)
filter (fun n -> n>3) ex = Cons (4N, (Cons (4N, Nil)))
```

Aufgabe 4 Statische Semantik von rekursiven Funktionen (Einreichaufgabe, 3 Punkte)

Wir betrachten die rekursive Funktionsdefinition

```
let rec f  (x: Bool): Bool = (if x then x else f (not x))
```

bezüglich der Signatur $\Sigma := \{\texttt{not} \mapsto \text{Bool} \to \text{Bool}\}$. Geben Sie einen vollständigen Beweisbaum an.

Aufgabe 5 Statische und dynamische Semantik (Trainingsaufgabe)

Prüfen Sie, ob die folgenden Mini-F#-Ausdrücke gültige Ausdrücke bezüglich der statischen Semantik sind. Geben Sie für die gültigen Ausdrücke deren Typ und einen entsprechenden Beweisbaum mit den Regeln der statischen Semantik an. Werten Sie dann gültige Ausdrücke mit den Regeln der dynamischen Semantik aus und geben Sie einen entsprechenden Beweisbaum an. Erklären Sie für die ungültigen Ausdrücke, wo der Fehler liegt.

a) `let a = 5 in (fun (x: Nat) -> x + a) 2`

b) `true in 4711`

c) `let a = 2 in (fun (x: Nat) -> let a = 3 in a * x) a`

d) `(fun (x: Nat) -> x * b) (let b = 3 in 2 * b)`

e) `((fun (c: Nat) -> (fun (x: Nat) -> c)) 42) 815`