

## Übungsblatt 6: Grundlagen der Programmierung (WS 2023/24)

Ausgabe: 05. Dezember 2023

Abgabe: 11./12./13. Dezember 2023, siehe [Homepage](#)

### Aufgabe 1 Funktionen höherer Ordnung (Präsenzaufgabe)

*Motivation:* In dieser Aufgabe sollen Sie sich mit Funktionen höherer Ordnung auf Listen beschäftigen. Sie können sich an den Vorlesungsfolien 394 bis 404 bzw. am Skript Kapitel 4.3.2 orientieren.

*Schreiben Sie Ihre Lösungen in die Datei `Lists.fs` aus der Vorlage `Aufgabe-6-1.zip`.*

*Hinweis:* Verwenden Sie in Ihrer Lösung **nicht** das `List`-Modul aus der Standardbibliothek. Die hier zu definierenden Funktionen gibt es alle so im `List`-Modul.

- a) Schreiben Sie eine Funktion `map`, die eine Funktion `f: 'a -> 'b` auf alle Elemente einer Liste `xs` vom Typ `List<'a>` anwendet und die Ergebnisliste vom Typ `List<'b>` zurückgibt.

Verwenden Sie `map`, um...

- eine Funktion `double` zu schreiben, die alle Zahlen einer Liste natürlicher Zahlen verdoppelt,
- eine Funktion `firstComponents` zu schreiben, die aus einer Liste von Tupeln die Liste der ersten Tupelkomponenten zurückgibt.

- b) Schreiben Sie eine Funktion `collect`, die eine Funktion `f: 'a -> List<'b>` auf alle Elemente einer Liste `xs` anwendet, die Einzelergebnisse konkateniert und eine Ergebnisliste vom Typ `List<'b>` zurückgibt.

Verwenden Sie `collect`, um eine Funktion `cloneElements` zu schreiben, die jedes Element einer Liste noch einmal in die Liste aufnimmt, Beispiel `cloneElements [1N; 2N; 3N] = [1N; 1N; 2N; 2N; 3N; 3N]`.



### Aufgabe 3 Heaps (Einreichaufgabe, 14 Punkte)

*Motivation:* In dieser Aufgabe sollen Sie üben mit komplexeren rekursiven Varianten zu programmieren. Sie können sich an den Vorlesungsfolien 512 bis 525 sowie am Skript Kapitel 5.2.3 orientieren.

Schreiben Sie Ihre Lösungen in die Datei `Heaps.fs` aus der Vorlage `Aufgabe-6-3.zip`.

Ein Heap ist eine Datenstruktur in Form eines Binärbaums: Ein Heap ist entweder leer oder ein Knoten mit einem Eintrag, einem linken Teilbaum und einem rechten Teilbaum. Die Datenstruktur dient dazu, Elemente partiell geordnet abzuspeichern. Daher muss ein gültiger Heap die *Heap-Bedingung* erfüllen: Der Eintrag jedes Knotens muss kleiner oder gleich der Einträge seiner beiden Teilbäume sein.

```
type Heap<'a> =  
  | Empty  
  | Node of Heap<'a> * 'a * Heap<'a>
```

Beispiele für gültige Heaps:

```
let ex1 = Node(Node(Empty,6N,Empty), 2N, Node(Empty,4N,Empty))  
let ex2 = Node(Node(Empty,7N,Empty), 3N, Node(Empty,5N,Empty))  
let ex3 = Node(Node(Empty,1N,Empty), 1N, Empty)
```

Beispiele für ungültige Heaps:

```
let inv1 = Node(Node(Empty,2N,Empty), 3N, Empty)  
let inv2 = Node(Node(Node(Empty,4N,Empty), 5N, Empty), 3N, Empty)
```

- a) Schreiben Sie Funktionen `size` und `height` jeweils vom Typ `Heap<'a> -> Nat`, die die Größe bzw. Höhe eines Heaps berechnen. Die Größe entspricht der Anzahl an Einträgen. Die Höhe ist die Länge des längsten Pfades von der Wurzel des Heaps bis zu einem leeren Teilbaum.

Beispiele mit den oben definierten Heaps:

```
size Empty = 0N           size ex3 = 2N           height ex1 = 2N  
size ex1 = 3N           height Empty = 0N           height ex3 = 2N
```

- b) Schreiben Sie eine Funktion `isHeap: Heap<'a> -> bool`, die überprüft ob der gegebene Heap die Heap-Bedingung erfüllt.

Beispiele:

```
isHeap<Nat> Empty = true           isHeap ex2 = true           isHeap inv1 = false  
isHeap ex1 = true           isHeap ex3 = true           isHeap inv2 = false
```

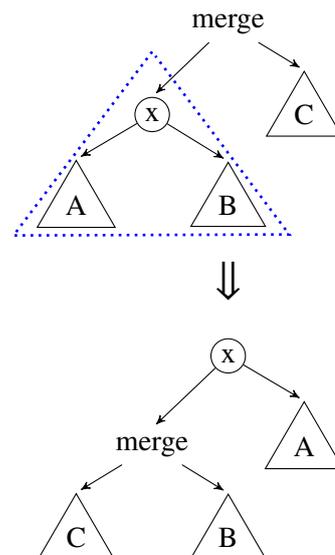
- c) Schreiben Sie eine Funktion `head: Heap<'a> -> Option<'a>`, die aus einem Heap das kleinste Element bestimmt. Da der Heap leer sein kann, verwenden wir wieder den `Option`-Typ. Das heißt, wenn der Heap leer ist, soll `None` zurückgegeben werden.

Sie können davon ausgehen, dass der gegebene Heap die Heap-Bedingung erfüllt ist.

Beispiel: `head ex1 = Some 2N`

d) Schreiben Sie eine Funktion `merge: Heap<'a> -> Heap<'a> -> Heap<'a>`, die zwei gegebene Heaps in einen Heap zusammenführt. Nutzen Sie für Ihre Implementierung den folgenden Algorithmus<sup>1</sup>:

- Wenn einer der beiden gegebenen Heaps leer ist, dann ist das Ergebnis der jeweils andere Heap.
- Ansonsten wähle als `p` den gegebenen Heap mit dem kleineren `head` und als `q` den anderen gegebenen Heap. Konstruiere den Ergebnis-Heap `r` als `Node` wie folgt:
  - Die erste Komponente (der Eintrag) von `r` ist `head p`.
  - Die zweite Komponente (linker Teilbaum) von `r` wird berechnet, indem `q` und der rechte Teilbaum von `p` rekursiv zusammengeführt werden.
  - Die dritte Komponente (rechter Teilbaum) von `r` ist der linke Teilbaum von `p`.



Die Abbildung rechts verdeutlicht den Algorithmus nochmals. Kreise sind Knoten und Dreiecke sind (Teil-)Bäume. Der blau umrandete Teil ist `p`. Der Baum `C` ist `q`.

Beispiel: `merge ex1 ex2 = Node( Node( Node(Empty, 5N, Empty), 4N, Empty), 3N, Node(Empty, 7N, Empty)), 2N, Node(Empty, 6N, Empty))`

*Tip:* Die `merge` Funktion können Sie gut in den weiteren Teilaufgaben verwenden.

e) Schreiben Sie eine Funktion `tail: Heap<'a> -> Heap<'a>`, die aus einem Heap das kleinste Element entfernt und einen gültigen Heap zurückgibt, der aus den restlichen Elementen besteht. Ist der Heap leer, soll der leere Heap wieder zurückgegeben werden. Sie können davon ausgehen, dass der gegebene Heap die Heap-Bedingung erfüllt ist.

Beispiel: `tail ex3 = Node(Empty, 1N, Empty)`

f) Schreiben Sie eine Funktion `insert: Heap<'a> -> 'a -> Heap<'a>`, die in den gegebenen Heap das gegebene Element einfügt. Sie können davon ausgehen, dass der gegebene Heap die Heap-Bedingung erfüllt. Der Ergebnis-Heap muss die Heap-Bedingung erfüllen. *Tip:* Verwenden Sie Ihre `merge` Funktion!

Beispiel: `insert Empty 3N = Node(Empty, 3N, Empty)`

g) Schreiben Sie Funktionen `ofList: List<'a> -> Heap<'a>` und `toList: Heap<'a> -> List<'a>`. Erstere nimmt eine Liste und erstellt einen gültigen Heap, der die Elemente der Liste enthält. Die Funktion `toList` nimmt einen gültigen Heap und erstellt daraus eine **sortierte** Liste der Elemente des Heaps.

Beispiele:

<code>ofList&lt;Nat&gt; [] = Empty</code>	<code>toList&lt;Nat&gt; Empty = []</code>
<code>ofList [3N] = Node(Empty, 3N, Empty)</code>	<code>toList ex1 = [2N; 4N; 6N]</code>
	<code>toList ex2 = [3N; 5N; 7N]</code>

h) Schreiben Sie eine Funktion `heapsort: List<'a> -> List<'a>`, die eine Liste von Elementen nimmt und diese sortiert. Verwenden Sie dazu die beiden Funktionen aus der vorherigen Teilaufgabe.

Beispiele: `heapsort<Nat> [] = []`      `heapsort [2N; 3N; 1N; 2N] = [1N; 2N; 2N; 3N]`

Wenn Sie die Funktion `merge` wie vorgesehen implementiert haben und in den anderen Funktionen sinnvoll verwenden, dann sollten Sie Listen der Länge `n` mit höchstens  $2 * n * \log_2(n)$  Vergleichen sortieren können. Dies wird durch den Testfall `heapsort Zufall` Effizienz abgeprüft.

<sup>1</sup>Skew Heap Verschmelzung, siehe [https://de.wikipedia.org/wiki/Skew\\_Heap](https://de.wikipedia.org/wiki/Skew_Heap)

## Aufgabe 4 Turtle-Grafik (Trainingsaufgabe)

*Motivation:* In dieser Aufgabe sollen Sie noch einmal den Umgang mit Listen einüben. Sie können sich an den Vorlesungsfolien 379 bis 404 sowie am Skript Kapitel 4.3 orientieren.

Schreiben Sie Ihre Lösungen in die Datei `Turtle.fs` aus der Vorlage `Aufgabe-6-4.zip`.

Als Turtle-Grafik wird eine Bildbeschreibungssprache verstanden, bei der man sich vorstellt, dass eine mit einem Stift ausgestattete Schildkröte (oder ein Roboter) sich über eine Zeichenebene bewegt. Die Schildkröte versteht verschiedene Kommandos, mit deren Hilfe sich ganze Programme zusammensetzen lassen, um ein Bild zu erstellen.

Dazu verwenden wir folgende Typen:

```
type Command =
  | D           // Drop:   Stift absetzen/anfangen zu zeichnen
  | F of Double // Forward: Vorwärts bewegen
  | L of Double // Left:   Nach links/gegen den Uhrzeigersinn drehen

type Program = List<Command>
```

*Hinweis:* Anders als sonst, arbeiten wir in dieser Aufgabe nicht mit natürlichen Zahlen. Stattdessen verwenden wir den Typ `Double`, um mit Fließkommazahlen zu arbeiten. Wenn Sie eine Zahl vom Typ `Double` angeben, müssen Sie darauf achten, den Dezimaltrenner mit anzugeben. Zum Beispiel wäre `2` eine Ganzzahl vom Typ `Int`, jedoch `2.0` eine Fließkommazahl.

Damit wir die Turtle-Grafiken auch tatsächlich betrachten können, ist in der Programmvorlage das Modul `Draw` enthalten, das Turtle-Grafiken in SVG-Bilder umwandeln kann. Sie können SVG-Dateien mit allen gängigen Webbrowsern öffnen. Im `Draw`-Modul gibt es eine Funktion `draw`, die ein Turtle-Programm als Argument erwartet und daraus eine SVG-Datei mit dem Namen `image.svg` im aktuellen Verzeichnis generiert. Das zu konvertierende Programm können Sie in der `main` Funktion auswählen und das gesamte Programm mit dem Befehl `dotnet run` ausführen.

Folgendes Turtle-Programm wird damit wie in Abbildung 1 dargestellt, in eine Grafik überführt.

```
let ex = [D; F 50.0; L 45.0; F 50.0]
```

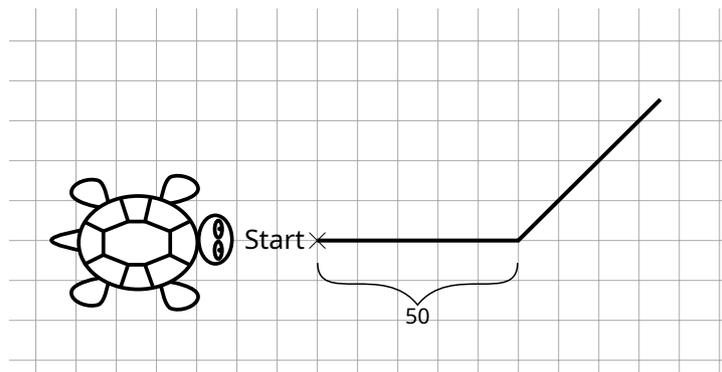


Abbildung 1: Darstellung des Programms `Turtle.ex`. Initial ist unsere Schildkröte nach rechts ausgerichtet. Sie setzt den Stift ab, bewegt sich um 50 Längeneinheiten vorwärts, dreht sich um 45 Grad nach links (gegen den Uhrzeigersinn) und bewegt sich erneut um 50 Längeneinheiten vorwärts.

Die Schildkröte startet ohne abgesetzten Stift und ist nach rechts ausgerichtet.

- a) Implementieren Sie einen „smarten Konstruktor“ (eine Funktion, die den Konstruktor eines bestimmten Typs aufruft und dabei ggf. noch zusätzliche Prüfungen durchführt, die das Typsystem nicht durchführen kann - das tun wir hier jedoch nicht), der ein Element des Typs `Command` zurückgibt, das eine Drehung um den Winkel `angle` nach rechts modelliert.

*Tipp:* Eine Drehung nach rechts entspricht einer Drehung nach links um einen Winkel mit negativem Vorzeichen.

- b) Unsere Turtle-Programme eröffnen uns eine spannende Möglichkeit: Wir können Teile eines gegebenen Programms `p` anhand bestimmter Regeln ersetzen. Wir betrachten hier die Funktion `substF`, die alle Vorkommen der Vorwärtsbewegung `F` ersetzt. Implementieren Sie die Funktion `substF` und rufen Sie die Funktion `transformF: Double -> Program` mit der Länge des jeweiligen `F` Konstruktors auf, um die Substitution durchzuführen. Die Transformationsfunktion `transformF` arbeitet mit der Länge des bisherigen `F` Segments. So ist es möglich, Längenverhältnisse in der Transformationsfunktion zu berücksichtigen.

- c) **Lévy-C-Kurve** Wir starten mit einer geraden Linie der Länge `s`. Implementieren Sie dazu die Funktion `levyStart`, die den Stift absetzt und diesen um die Länge `len` vorwärts bewegt.

Schreiben Sie nun eine Transformation `levyTransform`, welche eine Vorwärtsbewegung um die Länge `len` ersetzt durch

1. eine Drehung nach links um  $45^\circ$
2. eine Vorwärtsbewegung der Länge  $len / \sqrt{2}$
3. eine Drehung nach rechts um  $90^\circ$
4. eine Vorwärtsbewegung der Länge  $len / \sqrt{2}$
5. und noch eine Drehung nach links um  $45^\circ$ .

In den folgenden Aufgabenteilen verwenden wir dafür Abkürzungen:

- Den Buchstaben `F` für eine Vorwärtsbewegung (Skalierungsfaktor beachten)
- `+` für eine Drehung nach links, also gegen den Uhrzeigersinn (Winkel beachten)
- `-` für eine Drehung im Uhrzeigersinn (Winkel beachten)
- `->` gibt an, dass das links vom Pfeil ersetzt wird durch das, was rechts davon steht

Im vorliegenden Fall der Lévy-C-Kurve wäre die Kurzschreibweise für die Transformation `F -> +F--F+`, wobei das Längenverhältnis (vor Transformation vs nach Transformation)  $1 : 1/\sqrt{2}$  beträgt. `+` und `-` ändern den Winkel jeweils um  $45^\circ$  gegen den bzw. im Uhrzeigersinn.

*Hinweis:* In der `main` Funktion des `Draw` Moduls wird mit Hilfe der Funktion `iterate` die Transformation `n` mal angewendet.

*Hinweis:* Sie können die Funktion `sqrt` zur Berechnung der Quadratwurzel verwenden.

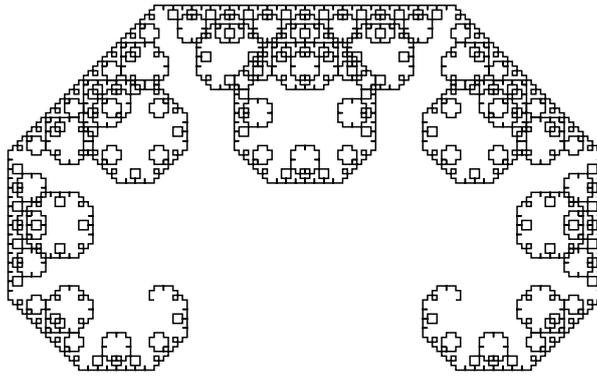


Abbildung 2: Lévy-C-Kurve, 12 Iterationen

- d) Implementieren Sie die Funktionen `kochflockeStart` und `kochflockeTransform`. Die Symbole `+` und `-` ändern den Winkel um  $60^\circ$ . `kochflockeStart` soll mit der Sequenz `F--F--F` ein Dreieck zeichnen. `kochflockeTransform` soll die Transformationsregel `F -> F+F--F+F` implementieren. Das Längenverhältnis beträgt dabei  $1 : 1/3$ .

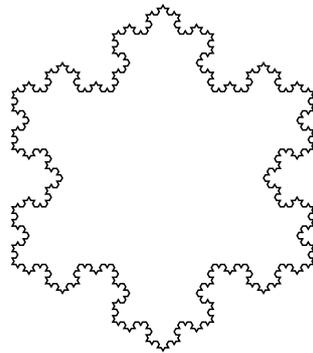


Abbildung 3: Koch-Flocke, 4 Iterationen

- e) Implementieren Sie die Funktionen `pentaplexityStart` und `pentaplexityTransform`. Die Symbole `+` und `-` ändern den Winkel um  $36^\circ$ . `pentaplexityStart` soll mit der Sequenz `F++F++F++F++F` ein Pentagon zeichnen. `pentaplexityTransform` soll die Transformationsregel `F -> F++F++F|F-F++F` implementieren. Das Symbol `|` repräsentiert eine Drehung um  $180^\circ$ . Das Längenverhältnis beträgt  $1 : 1/\phi^2$ , wobei  $\phi := \frac{1+\sqrt{5}}{2}$ .

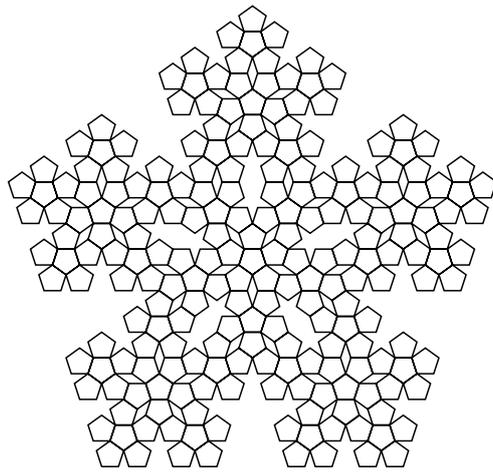


Abbildung 4: Penta Plexity, 3 Iterationen

## Aufgabe 5 Statische und dynamische Semantik (Trainingsaufgabe)

- a) Bestimmen Sie den Typ des Ausdrucks  $f$  ( $f \in \mathbb{N}$ ) bezüglich der Signatur  $\Sigma := \{f \mapsto (\text{Nat} \rightarrow \text{Nat})\}$ . Geben Sie einen vollständigen Beweisbaum auf Grundlage der Regeln der statischen Semantik aus der Vorlesung an.
- b) Bestimmen Sie den Typ des Ausdrucks  $f$  ( $a + b$ ) bezüglich der folgenden Signatur.

$$\Sigma := \{a \mapsto \text{Nat}, b \mapsto \text{Nat}, f \mapsto (\text{Nat} \rightarrow \text{Bool})\}$$

Geben Sie einen vollständigen Beweisbaum auf Grundlage der Regeln der statischen Semantik aus der Vorlesung an.

- c) Werten Sie den Ausdruck  $f$  ( $a + b$ ) bezüglich der folgenden Umgebung aus.

$$\delta := \{a \mapsto 1, b \mapsto 3, f \mapsto \langle \emptyset, x, x > 10\mathbb{N} \rangle\}$$

Geben Sie einen vollständigen Beweisbaum auf Grundlage der dynamischen Semantik aus der Vorlesung an.

- d) Bestimmen Sie den Typ des Ausdrucks `if f true then 2N else 1N` bezüglich der folgenden Signatur.

$$\Sigma := \{f \mapsto (\text{Bool} \rightarrow \text{Bool})\}$$

Geben Sie einen vollständigen Beweisbaum auf Grundlage der Regeln der statischen Semantik aus der Vorlesung an.

- e) Werten Sie den Ausdruck `if f true then 2N else 1N` bezüglich der folgenden Umgebung aus.

$$\delta := \{f \mapsto \langle \emptyset, x, \text{false} \rangle\}$$

Geben Sie einen vollständigen Beweisbaum auf Grundlage der dynamischen Semantik aus der Vorlesung an.