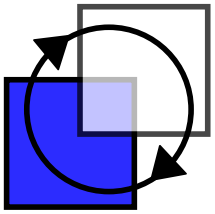
CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Prof. Bernd Finkbeiner, Ph.D.
Jana Hofmann, M.Sc.
Reactive Systems Group



Programmierung 1 (WS 2020/21)
Übungsblatt E

Lesen Sie im Buch Kapitel 5 - 6.4.1.

Hinweis: Über Aufgaben, die mit 🤔 markiert sind, müssen Sie eventuell etwas länger nachdenken. Falls Ihnen keine Lösung einfällt - kein Grund zur Sorge. Kommen Sie in die Office Hour, unsere Tutor:innen helfen gerne.

Listen

Aufgabe E.1
Schreiben Sie mit `foldr` eine Prozedur `seperate : ('a * 'b) list -> 'a list * 'b list`, welche aus einer Liste von Tupeln zwei Listen berechnet die jeweils die ersten und zweiten Positionen der Tupel in der selben Reihenfolge enthalten wie die ursprünglichen Liste.

So soll z.B. `seperate [(true, 1),(false,42)]` zu `([true,false],[1,42])` auswerten.

Aufgabe E.2
Implementieren Sie mittels Faltung eine Prozedur `sortEvenOdd : int list -> int list`, die eine Liste von natürlichen Zahlen so umsortiert, dass gilt:

(i) Die Liste enthält genau dieselben Elemente wie die Eingabe.

(ii) *Alle* geraden Zahlen stehen vor der ersten ungeraden Zahl.

(iii) Die geraden Zahlen haben die selbe Reihenfolge wie in der Eingabeliste, die ungeraden genau die umgekehrte.

Aufgabe E.3
Die Faltungsprozedur `foldl` kann mithilfe der Faltungsprozedur `foldr` deklariert werden, ohne dass dabei eine weitere rekursive Hilfsprozedur verwendet wird. Das können Sie sehen, wenn Sie wie folgt vorgehen: 🤔

(a) Deklarieren Sie `append` mithilfe von `foldr`.

(b) Deklarieren Sie `rev` mithilfe von `foldr` und `append`.

(c) Deklarieren Sie `foldl` mithilfe von `foldr` und `rev`.

(d) Deklarieren Sie `foldl` nur mithilfe von `foldr`.

Sortieren

Aufgabe E.4
Schreiben Sie eine Prozedur `isort'`, die eine Liste in absteigender Ordnung durch Einfügen sortiert.

Aufgabe E.5
Schreiben Sie eine Prozedur `tripleCompare (int*string*real) * (int*string*real) -> order`, welche die lexikalische Ordnung auf Tripeln des Typs `int*string*real` angibt.

Aufgabe E.6
In dieser Aufgabe implementieren Sie `smallsort : (`α `*` α $\rightarrow$ `order)` $\rightarrow$ α `list` $\rightarrow$ α `list`, ein neuer Sortieralgorithmus, der eine Liste sortiert, indem er immer das kleinste Element der unsortierten Liste sucht, dieses aus der ursprünglichen Liste löscht und vorne an die sortierte Restliste anhängt.

(a) Schreiben Sie eine Prozedur `drop: (` α `*` β $\rightarrow$ `order)` $\rightarrow$ α $\rightarrow$ β `list` $\rightarrow$ β `list`, die einen übergebenen Wert anhand einer `compare` Prozedur einmal aus einer Liste löscht.

(b) Schreiben Sie nun `smallsort :(`α `*` α $\rightarrow$ `order)` $\rightarrow$ α `list` $\rightarrow$ α `list`.

Aufgabe E.7
Schreiben Sie eine Prozedur `sorted : int list` $\rightarrow$ `bool`, die testet, ob eine Liste aufsteigend sortiert ist. Verwenden Sie dabei keine Hilfsprozedur.

Aufgabe E.8
Schreiben Sie eine Prozedur `polymsort :(`α `*` α $\rightarrow$ `order)` $\rightarrow$ α `list` $\rightarrow$ α `list`, welche eine beliebige Liste anhand einer compare-Prozedur durch *Mischen* sortiert.

Aufgabe E.9 *(Love-Generator 2.0)*
Das *Der Bachelor* Team von RTL ist noch nicht ganz zufrieden mit Ihrer Arbeit. Sie sollen den *Love-Generator* so erweitern, dass für einen Bachelor und mehrere Anwärter:innen eine nach Kompatibilität geordnete Liste von Paarungen zurückgegeben wird.

Schreiben Sie eine Prozedur `loveSort : string` $\rightarrow$ `string list` $\rightarrow$ `(string * string) list`, die für den Bachelor und eine Liste von Namen zunächst alle möglichen Paarungen berechnet und diese anschließend anhand ihrer LoveGen-Kompatibilität sortiert.

Aufgabe E.10
In dieser Aufgabe sollen Sie *Sortieren durch Filtern* implementieren. Die Idee ist die folgende: Angenommen, xs ist eine sortierte Teilliste. Nun wird x eingefügt, indem erst alle Zahlen $>= x$ aus xs entfernt werden, dann folgt x und dann die Liste xs, aus der alle Zahlen $< x$ entfernt wurden.

Für Freunde von one-linern: Implementieren Sie die Sortierprozedur mit nur einer Zeile, indem Sie Faltung nutzen. 🤔

Aufgabe E.11
Deklarieren Sie eine Prozedur

`lex : (`α `*` α $\rightarrow$ `order)` $\rightarrow$ `(`β `*` β $\rightarrow$ `order)` $\rightarrow$ `(`α `*` β`) * (`α `*` β`)` $\rightarrow$ `order`,

die zu zwei Ordnungen für die Typen α und β die lexikalische Ordnung für den Produkttyp α `*` β liefert.

Aufgabe E.12
Schreiben Sie eine Prozedur `perm : int list` $\rightarrow$ `int list` $\rightarrow$ `bool`, die testet, ob zwei Listen bis auf die Anordnung ihrer Elemente gleich sind. Verwenden Sie dabei die Prozedur `isort` aus dem Buch, die Sortieren durch Einfügen realisiert, und die Tatsache, dass `int list` ein Typ mit Gleichheit ist.

Aufgabe E.13 *(Bead Sort [1])*
In dieser Übungsaufgabe wird der Sortieralgorithmus bead sort vorgestellt, der eine `int list` absteigend sortiert. Zunächst sind einige Vorüberlegungen notwendig [2]. 🤔

(a) Machen Sie sich die Funktionsweise von bead sort klar: Gegeben sei die Liste `[3,3,1,4,2]` . Stellen Sie sich parallele Stangen vor, auf denen Bälle aufgespießt sind. Es gibt genauso viele Stangen wie der Wert des größten Elements, in unserem Fall vier. Reihen gibt es genauso viele wie es Elemente gibt, siehe Abbildung 1

[1] https://www.cs.auckland.ac.nz/~jaru003/research/publications/journals/beadsort.pdf
[2] Bilder: CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=826860
& CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=826890

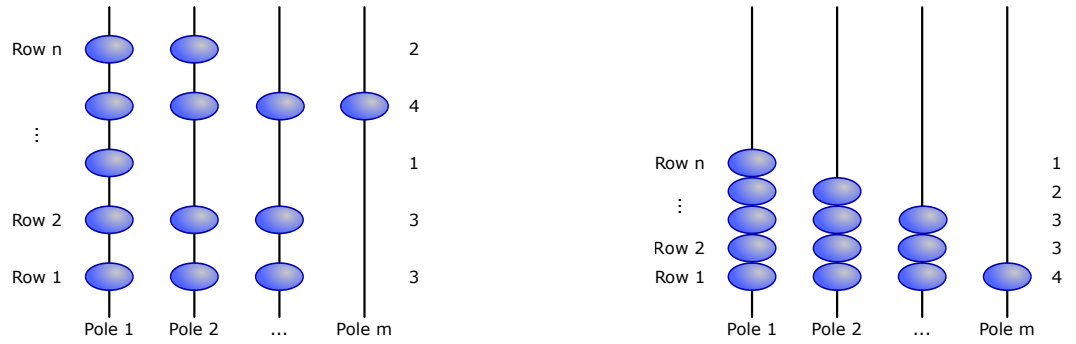


Abbildung 1: Bead Sort: Sortieren durch Fallenlassen.

links. Nun erlaubt man den Bällen *herunterzufallen*. Zählt man nun die Bälle in den Reihen ergibt sich die passende *absteigende* Sortierung, siehe Abbildung 1 rechts. Man erhält die Liste `[4,3,3,2,1]` .

(b) Schreiben Sie eine Prozedur `replicate : int` $\rightarrow$ α $\rightarrow$ α `list`, die gegeben eine natürliche Zahl n und ein beliebiges x, eine Liste ausgibt, die x genau n-mal enthält. So soll z.B. `replicate 3 5` zu `[5,5,5]` auswerten.

(c) Schreiben Sie mithilfe von `replicate` eine Prozedur `onlyOnes : int list` $\rightarrow$ `int list list`, die jede Zahl n einer Liste durch eine Liste repräsentiert, die die Zahl 1 n-mal enthält. Beispielsweise soll `onlyOnes [3,3,1,4,2]` zu der Liste `[[1,1,1],[1,1,1],[1],[1,1,1,1],[1,1]]` auswerten. Betrachten Sie das Beispiel in Verbindung mit Abbildung 1 links.

(d) Schreiben Sie eine Prozedur `columns :` α `list list` $\rightarrow$ α `list list`, welche die Elemente aus den Listen ihrer Position entsprechend den anderen zuordnet. Z.B. soll `columns [[1,2,3],[4,5,6],[7,8]]` zu `[[1,4,7],[2,5,8],[3,6]]` auswerten. Machen Sie sich die Funktionsweise von `columns` genau klar.

(e) Schreiben Sie die Prozedur `beadSort : int list` $\rightarrow$ `int list`. Hierzu muss zunächst die Prozedur `onlyOnes` auf die Argument-Liste angewendet werden. Wie (und wie oft?) müssen Sie nun die Prozeduren `columns` und `length` ins Spiel bringen um das gewünschte Ergebnis zu erzielen?

Konstruktoren

Aufgabe E.14

(a) Erweitern Sie den Datentyp `shape` aus Kapitel 6.1 um den Konstruktor `Polygon`, der ein gleichseitiges Polygon mit n Kanten und Kantenlänge x darstellt.

(b) Schreiben Sie nun eine Prozedur `edgeLength : shape` $\rightarrow$ `real` die die Kantenlänge (beziehungsweise Umfang bei `Circle`) der Form berechnet.

(c) Schreiben Sie eine Prozedur `scale : real` $\rightarrow$ `shape` $\rightarrow$ `shape`, die ein Objekt gemäß g einem Faktor skaliert. Beispielsweise soll *scale* 0.5 (*Square* 3.0) = *Square* 1.5 gelten.

Aufgabe E.15

In dieser Aufgabe stellen wir die natürlichen Zahlen mit den Werten des Typs

```
1 datatype nat = O | S of nat
```

wie folgt dar: $0 \mapsto O$, $1 \mapsto S\ O$, $2 \mapsto S\ (S\ O)$, $3 \mapsto S\ (S\ (S\ O))$, und so weiter.

(a) Deklarieren Sie eine Prozedur `rep : int` $\rightarrow$ `nat`, die die Darstellung einer natürlichen Zahl liefert.

(b) Deklarieren Sie eine Prozedur `num : nat` $\rightarrow$ `int`, die zu einer Darstellung die dargestellte Zahl liefert.

(c) Deklarieren Sie für `nat` kaskadierte Prozeduren `add`, `mul` und `less`, die den Operationen $+$, $*$ und $<$ für natürliche Zahlen entsprechen. Verwenden Sie dabei keine Operationen für `int`.

kNobelpreis

Nachdem Sie Ihren Freund mit der Erfüllbarkeit von DNFs und CNFs geholfen haben, müssen Sie sich nun schleunigst wieder dem Programmierung-1-Stoff widmen. Es bleibt weniger als zwei Wochen Zeit, um sich auf die Mittelklausur vorzubereiten! Doch schon bevor Sie das erste Übungsblatt noch einmal ganz durchgearbeitet haben reißt Sie ein Anruf aus Ihrer Konzentration. Am anderen Ende befindet sich niemand anderes als Dieter Schlau. Ohne Begrüßung kommt dieser direkt zum Punkt: "Es ist schrecklich! Ich konnte die Aufgabe 13 nicht lösen und habe sie dann einfach gegoogelt. Aber als ich den Code, den ich da gefunden habe, eingegeben habe, ist SOSML abgestürzt. Und jetzt kann ich keine `int`- oder `list`-Werte mehr benutzen! Wie soll ich mich denn so auf die Klausur vorbereiten?" Sie können Ihren Freund natürlich nicht im Stich lassen. Nach kurzem Grübeln stellen Sie Dieter die entscheidende Frage: "Funktionieren denn wenigstens die Prozeduren noch wie gehabt?" Als Dieter dies bejaht, erklären Sie ihm Ihre Idee.

Aufgabe E.16 *(Leben ohne int)*
Sie erinnern sich, schon einmal davon gehört zu haben, natürliche Zahlen als Abstraktionen darzustellen (negative Zahlen sind uns nicht so wichtig). Nach etwas Nachdenken kommt Ihnen die folgende Gleichung für die Darstellung natürlicher Zahlen in den Sinn:

$$\overline{n} \coloneqq \lambda f.\ \lambda a.\ f^n\ a$$

Das heißt, die Zahlen 0, 1 und 2 ließen sich folgendermaßen darstellen:

$$\overline{0} \coloneqq \lambda f.\ \lambda a.\ a$$
$$\overline{1} \coloneqq \lambda f.\ \lambda a.\ f\ a$$
$$\overline{2} \coloneqq \lambda f.\ \lambda a.\ f(f\ a)$$

(a) Machen Sie sich mit der gegebenen Kodierung der natürlichen Zahlen vertraut. Wie funktioniert sie? Deklarieren Sie anschließend die zwei Zahlen 0 und 3 mithilfe dieser Kodierung in SML.

(b) Entwerfen Sie also eine Prozedur `intToCode` und `codeToInt`, die zwischen `int` und Ihrer Kodierung konvertiert. Verwenden Sie in den folgenden Aufgaben keine noch nicht eingeführten Konstrukte.

(c) Erstellen Sie jeweils eine *nicht-rekursive* Prozedur,

(i) die den Nachfolger einer Zahl berechnet.

(ii) die zwei Zahlen addiert.

(iii) die zwei Zahlen multipliziert.

(iv) die für zwei Zahlen a, n die Potenz a^n bildet.

(v) die den Vorgänger einer Zahl berechnet. Der Vorgänger von 0 soll 0 sein.

Hinweis: Bedenken Sie, dass Sie die Prozeduren `intToCode` und `codeToInt` *nicht* benutzen dürfen, da diese den Typ `int` benutzen.

Aufgabe E.17 *(Leben ohne list)*
Nachdem Sie Dieter nun endlich wieder zu Zahlen verholfen haben, stehen nur noch die Listen aus. Doch da fällt Ihnen ein, dass man für diese eine ähnliche Darstellung wie für natürliche Zahlen verwenden kann.

(a) Beschreiben Sie, wie Sie Listen mit Abstraktionen darstellen können. Geben Sie dazu erneut die Prozeduren `listToCode` und `codeToList` an, die zwischen Listen und Ihrer Darstellung konvertieren. Ihre Darstellungsart soll für jeden Listentyp funktionieren, ausgenommen verschachtelte Listen. Der Einfachheit halber nehmen wir an, dass Ihr Interpreter normale `int`s nun wieder beherrscht.

(b) Geben Sie die folgenden nicht-rekursiven Prozeduren an, ohne Hilfsprozeduren zu verwenden (also auch nicht `listToCode` und `codeToList`):

(i) `app`, die für zwei Listen $\overline{xs}$ und $\overline{ys}$ die Konkatenation $\overline{xs@ys}$ liefert.

(ii) `forall`, die prüft, ob ein Prädikat p für alle Elemente einer Liste $\overline{xs}$ gilt.

Aufgabe E.18 *(Bonus: Wahrlich frugal)*
Nun sind Sie auf den Geschmack gekommen. Da sich `int` und `list` als Prozeduren darstellen lassen sind Sie überzeugt, dass dies auch für andere Datentypen gelten muss. Doch stimmt das wirklich? Versuchen Sie Encodings für weitere Typen, wie `bool`, ganze Zahlen und Paare, zu finden. Gibt es ein generisches Verfahren, wie man vorgehen kann, um dies zu bestimmen?