Technical University of Munich
Chair for Logic and Verification
Prof. Tobias Nipkow, Ph.D.
J. Rädle, L. Stevens, K. Kappelmann, MC M. Eberl

WS 2019/20
08.11.2019
Deadline: 18.11.2019, 23:59

Functional Programming and Verification

Sheet 4

Tutorial Exercises

Exercise T4.1 Oxford Comma

Write a function `andList :: [String] -> String` that takes a list of words $[w_1, w_2, \ldots, w_{n-1}, w_n]$ and returns the text „w_1, w_2, …, w_{n-1}, and w_n“ (using a comma in front of the „and“[1]). There are, however, exceptions for $n < 3$:

```
andList [] = ""
andList ["Ayize"] = "Ayize"
andList ["Bhekizizwe", "Gugu"] = "Bhekizizwe and Gugu"
andList ["Jabu", "Lwazi", "Nolwazi"] =
  "Jabu, Lwazi, and Nolwazi"
andList ["Phila", "Sihle", "Sizwe", "Zama"] =
  "Phila, Sihle, Sizwe, and Zama"
```

Exercise T4.2 Hello Type Constraints

Without using any list library functions (except the cons operator `(:)`), implement the following functions:

1. `lSub :: Num a => [a] -> [a]` takes a list $[x_1, \ldots, x_n]$ and returns the list $[x_1 - x_2, x_2 - x_3, \ldots, x_{n-1} - x_n, x_n]$.
2. `noDupSnoc :: Eq a => [a] -> a -> [a]` takes a list $[x_1, \ldots, x_n]$ and an element y and returns the list $[x_1, \ldots, x_n, y]$ if y is not already contained in the list; otherwise, it returns the input list.
3. `addAbsLt :: Num a => Ord a => [a] -> a -> a` that takes a list $[x_1, \ldots, x_n]$ and an element y and returns $\sum_{i \in \{i | x_i < y\}} |x_i|$.

   Hint: use `abs` from the typeclass `Num` to compute the absolute value of a number.

Exercise T4.3 Accumulators

Again, *without* using any list library functions (except the cons operator `(:)`), implement the following functions using an accumulator based approach:

1. Rewrite the function `addAbsLt` from the previous exercise.
2. `maxAbs` takes a list xs and returns $\max\{|x| \mid x \in xs\}$ if xs is not empty and 0 otherwise.

[1] This is known as the „Oxford comma“ or „serial comma“ and is beloved by the MC Jr.

3. `countSigns` returns the number of negative, zero, and positive elements in a given list. For example,

   ```
   countSigns [-1,0,0,-5,2,0] = (2,3,1)
   ```

   Hint: use `signum` from the typeclass `Num` to compute the sign (-1, 0, or 1) of a number.

4. `ltAndGt` returns `True` for a given list `xs` and element `y` if and only if `xs` contains both an element less than and greater than `y` while iterating the list only once.

5. `uniqCount` takes a list, collapses all adjacent elements that are equal, and counts the number of collapsed elements. For example,

   ```
   uniqCount []          = []
   uniqCount [1,2,3]     = [(1,1),(2,1),(3,1)]
   uniqCount [1,2,2,1]   = [(1,1),(2,2),(1,1)]
   uniqCount [1,1,4,3,3] = [(1,2),(4,1),(3,2)]
   ```

Homework

You need to collect 4 out of 5 points (P) to pass this sheet.

Exercise H4.0

Get yourself a good cup of tea and check out the Data.List library for your favourite list functions, and the references for Eq, Num, and Ord. You might also want to change your default search engine to Hoogle – a search engine for Haskell libraries.

Exercise H4.1 Kowalski, Text Analysis! [1–3: 1P, 4+5: 1P, 6–9: 1P, 10: 1P, 11: 1P]

The subsequent vocabulary analysis makes use of multisets, which, in contrast to sets, may contain the same element multiple times, e.g. the multiset $\{a, b, a\}$ contains a two times (we say that the *multiplicity* of a is two). A multiset of elements of type `a` is represented as a list `[(a, Int)]` of tuples where the second component of each tuple is the multiplicity of the element. Implement the following multiset operations:

1. `isMultiSet :: Eq a => [(a, Int)] -> Bool` is a predicate that checks whether its argument is a proper multiset in the sense that all multiplicities are greater than 0 and that there are no duplicates.

2. `toList :: [(a, Int)] -> [a]` converts a multiset into a list that contains n copies of each element e with multiplicity n.

3. `toSet :: Eq a => [(a, Int)] -> [a]` is similar to `toList` but also eliminates duplicates.

4. `toMultiSet :: Eq a => [a] -> [(a, Int)]` converts a list to a multiset.

5. `multiplicity :: Eq a => a -> [(a, Int)] -> Int` determines the multiplicity of an element in a multiset.

We can consider a multiset as an infinite vector $(m_1, m_2, \ldots)$ by assigning each `e :: a` a unique index i and letting m_i be the multiplicity of `e`. A possible measure for the similarity of two vectors $\vec{a}$ and $\vec{b}$ is the cosine of the angle θ between them, which can be calculated as follows

$$cos\ \theta = \frac{\vec{a} \circ \vec{b}}{||\vec{a}|| \cdot ||\vec{b}||},$$

where $\vec{a} \circ \vec{b}$ is the dot product of $\vec{a}$ and $\vec{b}$ and $||\vec{a}||$ is the euclidean norm of $\vec{a}$ defined as $\sqrt{a_1^2 + a_2^2 + \cdots}$ for $\vec{a} = (a_1, a_2, \ldots)$. Implement the following functions in order to calculate the similarity of two multisets:

6. `dotProduct :: Eq a => [(a, Int)] -> [(a, Int)] -> Int` computes the dot product between the vector representation of two multisets.
7. `euclidean :: Eq a => [(a, Int)] -> Float` calculates the euclidean norm of the vector representation of a multiset.
8. `cosine :: Eq a => [(a, Int)] -> [(a, Int)] -> Float` returns the cosine of the angle between two multisets. *Hint:* you can convert from `Int` to `Float` with the function `fromIntegral`.
9. Finally, implement the function `vocabSimilarity :: String -> String -> Float` that uses the multisets of the words occuring in two different texts to compare them with `cosine`:

```
vocabSimilarity "i am hungry" "hello hungry this is patrick"
>>> 0.2581989

vocabSimilarity "ask not what your country can do for you"
                "ask what you can do for your country"
>>> 0.9428091

vocabSimilarity "lorem ipsum dolor sit amet"
                "consectetur adipiscing elit"
>>> 0.0
```

We now turn our attention from text analysis to word analysis. In particular, we consider the edit distance between two words v and w which is the minimum number of character removals, insertions, or substitutions that have to be performed on v such that we arrive at w. In addition to the above operations, we also allow for the transposition of two adjacent characters, e.g. changing "Kowalksi" to "Kowalski" is one operation. Formally, the edit distance $d_{a,b}(|a|, |b|)$ between two strings a and b, where $|a|, |b|$ is the length of a, b, respectively, is defined recursively as

$$d_{a,b}(i,j) = \min \begin{cases} 0 & \text{if } i = j = 0, \\ d_{a,b}(i-1,j)+1 & \text{if } i > 0, \\ d_{a,b}(i,j-1)+1 & \text{if } j > 0, \\ d_{a,b}(i-1,j-1) & \text{if } i,j > 0 \text{ and } a[i-1] = b[j-1] \\ d_{a,b}(i-1,j-1)+1 & \text{if } i,j > 0 \text{ and } a[i-1] \neq b[j-1] \\ d_{a,b}(i-2,j-2)+1 & \text{if } i,j > 1 \text{ and } a[i-1] = b[j-2] \text{ and } a[i-2] = b[j-1]. \end{cases}$$

The edit distance is useful for word processing as the permissible operations correspond to typical errors made by human typists. Your task is to implement a simple spell correction scheme using the following functions:

10. `editDistance :: Eq a => [a] -> [a] -> Int` computes the edit distance between two lists.
11. **(Competition)** Write a function

    ```
    spellCorrect :: [String] -> [String] -> [[String]]
    ```

    such that `spellCorrect d xs` returns a list of lists of those strings in the dictionary `d` whose `editDistance` is minimal for each `x` in `xs`. For example, using the `frequentWords` dictionary provided by the template, we have:

    ```
    spellCorrect frequentWords ["iq", "fpv"]
    >>> [["in","is","it","if"],["up","for"]]
    ```

    For this week's competition, we are asking you to optimize your `spellCorrect` function for performance. As a first step, you might consider implementing `editDistance` using dynamic programming in order to avoid recomputing values of $d_{a,b}$.

Important: If you submit a competition exercise, you agree that we are allowed to publish your name as part of the competition on our website. If you just want to submit a competition exercise as part of your homework without taking part in the competition, you can just remove the {`-WETT-`}...{`-TTEW-`} comments of your submission.

Haskell is concise
Functional well-typed and neat
It is like Haiku

— Haskell's Haiku webpage