How to Derive an Electronic Functional Programming Exam from a Paper Exam with Proofs and Programming Tasks

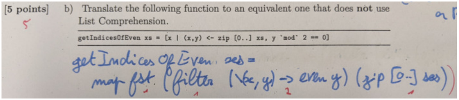
Ole Lübke[1]
ole.luebke@tuhh.de
Konrad Fuger[2] Fin Hendrik Bahnsen[3]
Katrin Billerbeck[4] Sibylle Schupp[1]

[1]Institute for Software Systems
[2]Institute for Communication Networks
[4]Center for Teaching and Learning
Hamburg University of Technology
Hamburg, Germany

[3]Institute for Artificial Intelligence in Medicine
University Medicine Essen
Essen, Germany

January 12, 2023

Motivation



[5 points] b) Translate the following function to an equivalent one that does **not** use List Comprehension.

```
getIndicesOfEven xs = [x | (x,y) <- zip [0..] xs, y `mod` 2 == 0]
```

5

getIndicesOfEven xs =
map fst (filter (\(x,y) -> even y) (zip [0..] xs))

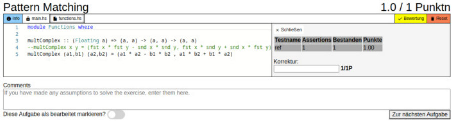
1 1 2 1

or F

Figure 1: Paper Exam

- 👍 Complex, constructively aligned tasks
- 👎 High grading effort
- 👎 Handwritten code . . .

↓



```
module Functions where

multComplex :: (Floating a) => (a, a) -> (a, a) -> (a, a)
--multComplex x y = (fst x * fst y - snd x * snd y, fst x * snd y + snd x * fst y)
multComplex (a1,b1) (a2,b2) = (a1 * a2 - b1 * b2 , a1 * b2 + b1 * a2)
```

Testname	Assertions	Bestanden	Punkte
ref	1	1	1.00

Korrektur: 1/1P

Comments

If you have made any assumptions to solve the exercise, enter them here.

Diese Aufgabe als bearbeitet markieren? ○

Figure 2: E-Exam

- 👍👍 Complex, more constructively aligned tasks
- 👍 Automated grading
- 👍 Familiar working environment

Outline

1 Analysis of our Pre-E-Exam FP Course

2 Realization

   Programming Tasks

   New Algorithm to Evaluate Proof Puzzles

3 Summary & Future Work

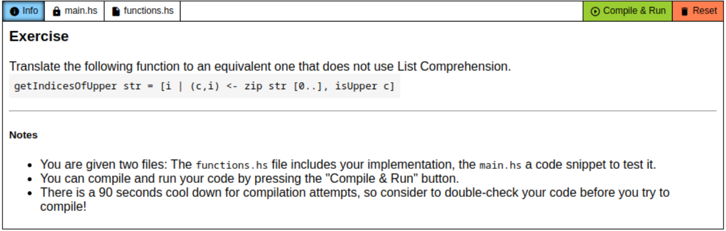
Analysis of our Pre-E-Exam FP Course

Table 1: Excerpt from analysis results

Task	Learning Objective	Type
1a	K_1, K_2, K_4	snippet
1b	K_1, K_2, K_4	multiple choice
2a	K_1, K_2, K_3	single choice, snippet
2b	K_1, K_2, S_2, S_3	code
3a	K_1, K_2, K_3, K_4	single choice, snippet
3b	K_1, K_2, K_4, S_3	code
4a	K_1, K_2	text
...	...	...

Realization

Programming Tasks



Exercise

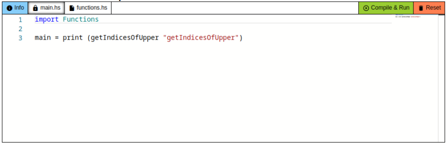
Translate the following function to an equivalent one that does not use List Comprehension.

```
getIndicesOfUpper str = [i | (c,i) <- zip str [0..], isUpper c]
```

Notes

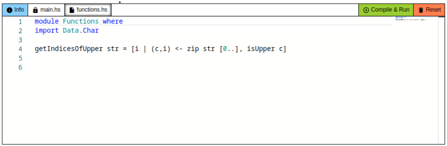
- You are given two files: The `functions.hs` file includes your implementation, the `main.hs` a code snippet to test it.
- You can compile and run your code by pressing the "Compile & Run" button.
- There is a 90 seconds cool down for compilation attempts, so consider to double-check your code before you try to compile!

Task description



```
import Functions

main = print (getIndicesOfUpper "getIndicesOfUpper")
```

main.hs



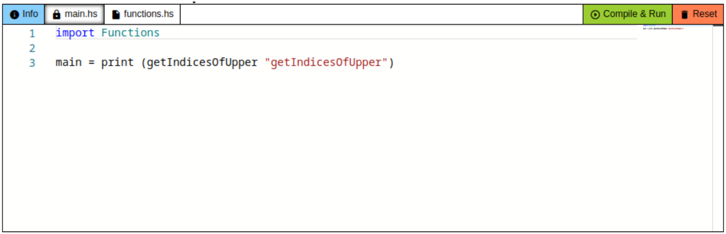
```
module Functions where
import Data.Char

getIndicesOfUpper str = [i | (c,i) <- zip str [0..], isUpper c]
```

functions.hs

Figure 3: Programming task

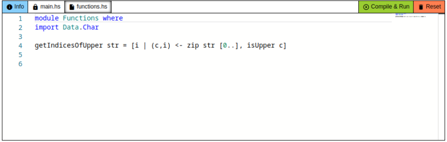
Realization

Programming Tasks



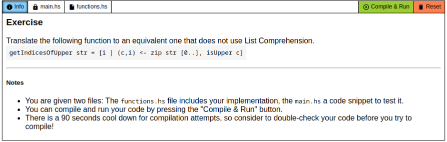
```haskell
import Functions

main = print (getIndicesOfUpper "getIndicesOfUpper")
```

main.hs



```haskell
module Functions where
import Data.Char

getIndicesOfUpper str = [i | (c,i) <- zip str [0..], isUpper c]
```

functions.hs



Exercise

Translate the following function to an equivalent one that does not use List Comprehension.

```haskell
getIndicesOfUpper str = [i | (c,i) <- zip str [0..], isUpper c]
```

Notes

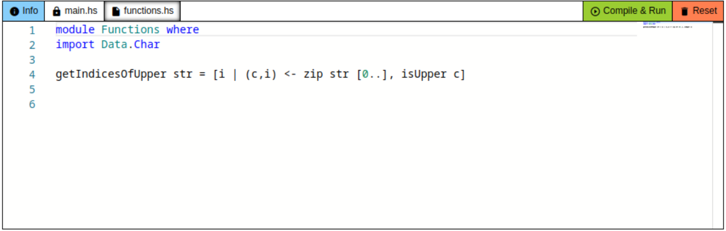
- You are given two files: The `functions.hs` file includes your implementation, the `main.hs` a code snippet to test it.
- You can compile and run your code by pressing the "Compile & Run" button.
- There is a 90 seconds cool down for compilation attempts, so consider to double-check your code before you try to compile!

Task description

Figure 3: Programming task

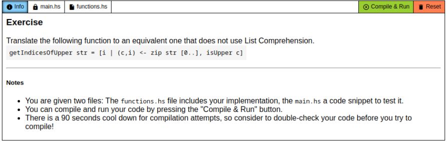
Realization

Programming Tasks



```haskell
module Functions where
import Data.Char

getIndicesOfUpper str = [i | (c,i) <- zip str [0..], isUpper c]
```

functions.hs



Exercise

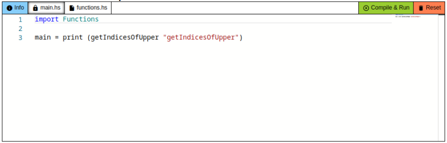
Translate the following function to an equivalent one that does not use List Comprehension.

```haskell
getIndicesOfUpper str = [i | (c,i) <- zip str [0..], isUpper c]
```

Notes

- You are given two files: The `functions.hs` file includes your implementation, the `main.hs` a code snippet to test it.
- You can compile and run your code by pressing the "Compile & Run" button.
- There is a 90 seconds cool down for compilation attempts, so consider to double-check your code before you try to compile!

Task description



```haskell
import Functions

main = print (getIndicesOfUpper "getIndicesOfUpper")
```

main.hs

Figure 3: Programming task

Realization

Programming Tasks – Static Analysis of Student Code

Listing 1: Example of quicksort in Haskell[1]

```haskell
quicksort :: Ord a => [a] -> [a]
quicksort [] = []
quicksort (p:xs) = (quicksort ls) ++ [p] ++ (quicksort gs)
  where ls = filter (< p) xs
        gs = filter (>= p) xs
```

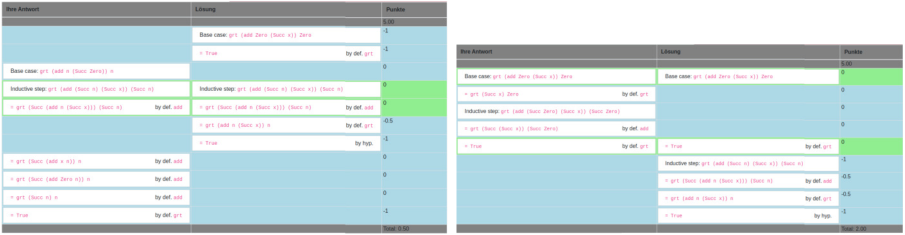
Listing 2: Corresponding output of analyzer

```json
{ "functions": [{
  "name": "quicksort",
  "patMatch": true,
  "guards": false,
  "listComprehension": false,
  "hasIf": false,
  "hasCase": false,
  "args": [ "p", "xs" ],
  "calledFns": [ "quicksort", "++", "filter", "<", ">=" ],
  "declaredFns": [ ]
}]}
```

[1] https://wiki.haskell.org/Introduction#Quicksort_in_Haskell

Realization

New Algorithm to Evaluate Proof Puzzles



Ihre Antwort	Lösung	Punkte
		5.00
	Base case: grt (add Zero (Succ x)) Zero	-1
	= True by def. grt	-1
Base case: grt (add n (Succ Zero)) n		0
Inductive step: grt (add (Succ n) (Succ x)) (Succ n)	Inductive step: grt (add (Succ n) (Succ x)) (Succ n)	0
= grt (Succ (add n (Succ x))) (Succ n) by def. add	= grt (Succ (add n (Succ x))) (Succ n) by def. add	0
	= grt (add n (Succ x)) n by def. grt	-0.5
	= True by hyp.	-1
= grt (Succ (add x n)) n by def. add		0
= grt (Succ (add Zero n)) n by def. add		0
= grt (Succ n) n by def. add		0
= True by def. grt		-1
		Total: 0.50

(a) Student A

Ihre Antwort	Lösung	Punkte
		5.00
Base case: grt (add Zero (Succ x)) Zero	Base case: grt (add Zero (Succ x)) Zero	0
= grt (Succ x) Zero by def. grt		0
Inductive step: grt (add (Succ Zero) (Succ x)) (Succ Zero)		0
= grt (Succ (Succ x)) (Succ Zero) by def. add		0
= True by def. grt	= True by def. grt	0
	Inductive step: grt (add (Succ n) (Succ x)) (Succ n)	-1
	= grt (Succ (add n (Succ x))) (Succ n) by def. add	-0.5
	= grt (add n (Succ x)) n by def. grt	-0.5
	= True by hyp.	-1
		Total: 2.00

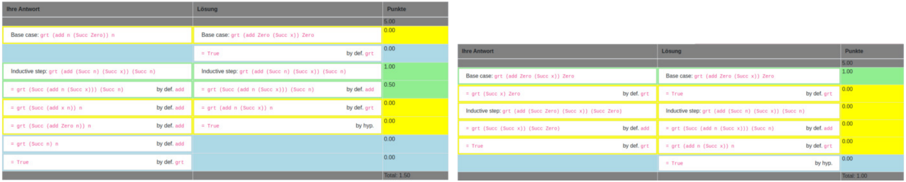
(b) Student B

Figure 4: Example of unfair grading with old algorithm

- Evaluation algorithm based on **edit-distance** between given and correct solution

Realization

New Algorithm to Evaluate Proof Puzzles



Ihre Antwort	Lösung	Punkte
		5.00
Base case: `grt (add n (Succ Zero)) n`	Base case: `grt (add Zero (Succ x)) Zero`	0.00
	`= True` by def. `grt`	0.00
Inductive step: `grt (add (Succ n) (Succ x)) (Succ n)`	Inductive step: `grt (add (Succ n) (Succ x)) (Succ n)`	1.00
`= grt (Succ (add n (Succ x))) (Succ n)` by def. `add`	`= grt (Succ (add n (Succ x))) (Succ n)` by def. `add`	0.50
`= grt (Succ (add x n)) n` by def. `add`	`= grt (add n (Succ x)) n` by def. `grt`	0.00
`= grt (Succ (add Zero n)) n` by def. `add`	`= True` by hyp.	0.00
`= grt (Succ n) n` by def. `add`		0.00
`= True` by def. `grt`		0.00
		Total: 1.50

(a) Student A

Ihre Antwort	Lösung	Punkte
		5.00
Base case: `grt (add Zero (Succ x)) Zero`	Base case: `grt (add Zero (Succ x)) Zero`	1.00
`= grt (Succ x) Zero` by def. `grt`	`= True` by def. `grt`	0.00
Inductive step: `grt (add (Succ Zero) (Succ x)) (Succ Zero)`	Inductive step: `grt (add (Succ n) (Succ x)) (Succ n)`	0.00
`= grt (Succ (Succ x)) (Succ Zero)` by def. `add`	`= grt (Succ (add n (Succ x))) (Succ n)` by def. `add`	0.00
`= True` by def. `grt`	`= grt (add n (Succ x)) n` by def. `grt`	0.00
	`= True` by hyp.	0.00
		Total: 1.00

(b) Student B

Figure 5: No unfair grading with new algorithm

- Evaluation algorithm based on **correct sequences with predefined entry points**

Summary & Future Work

- Today: Glimpse of preliminary analysis, proof & programming tasks
- What else?
  - Checking code snippets with regular expressions
  - Flexibly generating suitable regular expressions
  - General-purpose comment field for each task
  - Evaluation
    - Degree of automation
    - Student view
    - Examiner view
- Future:
  - Improve awarding partial points
  - Automatically generate most of the exam from a single literate Haskell file with Markdown