02157 Functional programming

Michael R. Hansen
DTU Compute
September 26, 2019

# Mandatory assignment 1

This is the first of four mandatory assignments in 02157 Functional programming. It is a requirement for exam participation that 3 of the 4 mandatory assignments are approved. The mandatory assignments can be solved individually or in groups of 2 or 3 students.

*Acceptance of a mandatory assignment from previous years does NOT apply this year.*

- Your solution should be handed in **no later than Thursday, October 3, 2019**. Submissions handed in after the deadline will face an *administrative rejection.*
- The assignment has three tasks:

  **Task 1:** This task has the form of a paper and pencil exercise. You may consider scanning in you hand-written solution to this task and submit it in the form of a pdf-file.

  **Tasks 2 and 3:** These task concern a programming problem. A solution to them should have the form of a single F# file (*file*`.fsx` or *file*`.fs`). In your solution you are allowed to introduce helper functions; but you must also provide a declaration for each of the required functions, so that it has exactly the type and effect asked for.

  You should make structural tests (that is, white-box tests) for the declared functions belonging to questions 1. and 2. for Task 2. Thus, for each declared function you must give test cases showing that every branch of the function is executed and works correctly. The expected result for each test must be stated explicitly.
- Do not use imperative features, like assignments, arrays and so on in your programs. Failure to comply with that will result in an *administrative rejection of your submission.*
- To submit you should upload two files to Inside under Assignment 1: a pdf-fil containing the solution to Task 1 and a single F# file (*file*`.fsx` or *file*`.fs`) containing the solutions to Task 2 and Task 3. The files should start with full names and study numbers for all members of the group. If the group members did not contribute equally to the solution, then the role of each student must be explicitly stated.
- Your F# solution must be a complete program that can be uploaded to F# Interactive without encountering compilation errors. Failure to comply with that will result in an *administrative rejection of your submission.*
- Be careful that you submit the right files. A submission of a wrong file will result in an *administrative rejection of your submission.*
- **DO NOT COPY solutions** from others and **DO NOT SHARE your solution** with others. Both cases are considered as fraud and will be reported.

## Task 1

The function `collect` from the `List` library could have the following declaration:

```
let rec collect f = function
                    | []    -> []
                    | x::xs -> f x @ collect f xs;;
val ('a -> 'b list) -> 'a list -> 'b list
```

Notice that the F# system automatically infers that `collect` has the type:

```
('a -> 'b list) -> 'a list -> 'b list
```

1. Give an argument showing that this type is indeed the most general type of `collect`. That is, any other type for `collect` is an instance of the most general type. You may support your argument by use of type assertions of the form $e : t$ and type constraints of the form $'a = .....$

An example use of `collect` is:

```
collect (fun (a,b) -> [a..b]) [(1,3); (4,7); (8,8)];;
val it : int list = [1; 2; 3; 4; 5; 6; 7; 8]
```

Notice that $[a..b]$ gives the list of integers from $a$ to $b$, when $a$ and $b$ are integers.

2. Give an evaluation showing that `[1; 2; 3; 4; 5; 6; 7; 8]` is the value of the expression `collect (fun (a,b) -> [a..b]) [(1,3); (4,7); (8,8)]`. Present your evaluation using $e_1 \rightsquigarrow e_2$ and include at least as many steps as there are recursive calls and applications of anonymous functions.
3. What is the type used for `collect` in the expression
   `collect (fun (a,b) -> [a..b]) [(1,3); (4,7); (8,8)] ?`
   Argue that this type for `collect` is an instance of the most general type shown above.

## Task 2

Make a solution to the Airport-Luggage problem, presented on the next two pages. You are NOT allowed to use functions from the `List` library in the first 4 questions.

## Task 3

You shall now solve questions 2. and 3. from the Airport-Luggage problem using higher-order functions from the list library. Attach in a comment to the programs justifications for your choices (of library functions).

## Luggage and Flights

Flight travellers may check-in the pieces of luggage, that should follow them on their journey, also when it contains multiple stops. A piece of luggage is marked with an *identification* (type `Lid`) by the start of the journey and that identification is associated with the *route* (type `Route`) of the journey. A route is a list of pairs identifying the *flights* (type `Flight`) and *airports* (type `Airport`) the luggage is passing on the journey.

Furthermore, a *luggage catalogue* (type `LuggageCatalogue`) is maintained, that uniquely identifies the routes of all pieces of luggage leaving some airport.

This is captured by the type declarations:

```
type Lid     = string
type Flight  = string
type Airport = string

type Route            = (Flight * Airport) list
type LuggageCatalogue = (Lid * Route) list
```

An example of a luggage catalogue is

```
[("DL 016-914", [("DL 189","ATL"); ("DL 124","BRU"); ("SN 733","CPH")]);
 ("SK 222-142", [("SK 208","ATL"); ("DL 124","BRU"); ("SK 122","JFK")])]
```

where first element in the list describes that the piece of luggage with identification "DL 016-914" is following a route, where it is first flown to Atlanta ("ATL") with flight "DL 189", then flown to Bruxelles "BRU" with flight "DL 124", and so on.

1. Declare a function `findRoute: Lid*LuggageCatalogue -> Route`, that finds the route for a given luggage identification in a luggage catalogue. A suitable exception should be raise if a route is not found.
2. Declare a function `inRoute: Flight -> Route -> bool`, that decides whether a given flight occurs in a route.
3. Declare a function `withFlight` *f lc*, where *f* is a flight and *lc* is a luggage catalogue. The value of the expression `withFlight` *f lc* is a list of luggage identifiers for the pieces of luggage that should travel with *f* according to *lc*. The sequence in which the identifiers occur in the list is of no concern.
   For the above example, both "DL 016-914" and "SK 222-142" should travel with the flight "DL 124".

An *arrival catalogue* associates with every airport, identifications of all pieces of luggage that should arrive at the airport. This is captured by the type declaration:

```
type ArrivalCatalogue = (Airport * Lid list) list
```

The following arrival catalogue is derived from the luggage catalogue appearing on the previous page:

```
[("ATL", ["DL 016-914"; "SK 222-142"]);
("BRU", ["DL 016-914"; "SK 222-142"]);
("JFK", ["SK 222-142"]);
("CPH", ["DL 016-914"])]
```

4. Declare a function `extend: Lid*Route*ArrivalCatalogue -> ArrivalCalalogue` so that `extend`$(lid, r, ac)$ is the arrival catalogue obtained by extending $ac$ with the information that *lid* will arrive at each airport contained in route $r$.
5. Declare a function `toArrivalCatalogue: LuggageCatalogue -> ArrivalCatalogue`, that creates an arrival catalogue from the information of a given luggage catalogue.
   You should solve this exercise using `extend` from the previous question in combination with either `List.fold` or `List.foldBack`. The types of these functions are:
   - `List.fold: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`
   - `List.foldBack: ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`