Written examination, 17 December 1999

Course number 49135

Name: Programming

Bemærk: Dansk version findes på de første to ark

Permitted aids: Written materials only

This examination paper contains four problems with the following weights:
Problem 1: 30 %, problem 2: 20 %, problem 3: 20 %, problem 4: 30 %.

The answers will be evaluated in the 13-scale.

The programming language SML should be used for solving the problems.
You are allowed to use functions from the SML standard library and examples from the text book. If you do that you must state a page reference for each function used. If you use a previous version of the book (i.e. prior to 1999), please state edition too.

Problem 1

Air travelers may bring *bagage* which is transported by the same *flight* and frequently has to be reloaded in *airports* during the journey. At the beginning of the journey the bagage is marked by an *identification* together with the *route*, which is a list of pairs stating the flights and airports to be passed by the bagage during the journey. The identification will be called *bagageId* in the following.

All bagage leaving an airport by air is listed in a *bagage catalog* associating a unique route with each bagageId.

These concepts are modelled by the following type declarations:

```
type bagageId      = string
type flight        = string
type airport       = string
type route         = (flight * airport) list
type bagageCatalog = (bagageId * route) list
```

The following is a sample value of type `bagageCatalog`:

```
[
 ("DL 016-914", [("DL 189","ATL"),("DL 124","BRU"),("SN 773","CPH")]),
 ("SK 222-142", [("SK 208","ATL"),("DL 124","BRU"),("SK 112","JFK")])
]
```

The first element of this list states the route for the bagage with bagageId `"DL 016-914"`, where the bagage is first carried to Atlanta (`"ATL"`) on the flight `"DL 189"`, then to Brussels (`"BRU"`) on the flight `"DL 124"`, and so on.

... Continued on the next page

1A: Declare a function `findRoute: bagageId * bagageCatalog -> route`, to find the route of a given bagageId in a bagageCatalog.

1B: Declare an infix function `inRoute: flight * route -> bool`, to test if a flight occurs in a route.

1C: Declare a function `onFlight: flight * bagageCatalog -> bagageId list`, to compute – for a given flight and bagageCatalog – a list of the bagageId's for the bagage which should be transported by the given flight.
For example, using the bagageCatalog on the previous page, both of `"DL 016-914"` and `"SK 222-142"` should be transported by the flight `"DL 124"`.

An *arrival catalog* associates lists of arriving bagage to airports. This is modelled by the following type:

```
type arrivalCatalog = (airport * bagageId list) list;
```

The following arrivalCatalog has been derived from the bagageCatalog on the previous page:

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

1D: Declare a function

```
update: bagageId * route * arrivalCatalog -> arrivalCatalog
```

to update a given arrivalCatalog with the information about a bagageId and its route.

Hint: Use an auxiliary function which inserts a bagageId and an associated airport in an arrivalCatalog.

... Continued on the next page

Problem 2

In this problem we use the term *account* for a list of the form:

$$[(m_1, y_1), (m_2, y_2), \ldots, (m_k, y_k)]$$

where the y_i's are mutually different and where the m_i's are positive integers. One says that the value y_i occurs m_i times in the account.

2A: Declare a function `f`, such that `f` *xs* is an account of the elements of the list *xs*, e.g.:

```
f ["a","b","a"] = [(1,"b"),(2,"a")]
```

Hint: Use an auxiliary function which adds one occurence of a value y to an account.

2B: Find the type of `f`.

2C: Find a function g and a value b such that the function `f` can be declared by:

```
fun f xs = foldr g b xs
```

Problem 3

The functions `h` and `k` are declared by:

```
fun h (x,[])       = []
  | h (x,(y::ys)) = if x = y then ys else y::h(x,ys) ;

fun k xs ys = foldr h xs ys
```

3A: Find the types of `h` and `k`.

3B: Find the values of the expressions `h(1,[2,1,3,1])` and `k [2,1,3,1] [1,2]`

3C: Describe the way in which the functions `h` and `k` work.

... Continued on the next page

Problem 4

In this problem we consider Boolean expressions of the form:

`VAR` s	where s is a character string (of type `string`)
`NOT` e	where e is a Boolean expression
e_1 `AND` e_2	where e_1 and e_2 are Boolean expressions
e_1 `OR` e_2	where e_1 and e_2 are Boolean expressions

They are modelled by the following `datatype` declaration:

```
infix 2 AND ;
infix 1 OR  ;

datatype bExpr = VAR of string
               | NOT of bExpr
               | AND of bExpr * bExpr
               | OR of bExpr * bExpr ;
```

4A: Declare a function:

```
vars: bExpr -> string list
```

such that `vars(`e`)` is a list of the strings s, such that `VAR` s occurs in the expression e.

4B: Declare a function `eval` where the value `eval(`e`,` $[s_1, \ldots, s_n]$`)` is computed according to the following rules, for Boolean expression e and list $[s_1, \ldots, s_n]$ of strings:

`VAR` s gives `true` if s is an element in the list $[s_1, \ldots, s_n]$, and `false` otherwise.

`NOT` e gives `true` exactly when e gives `false`.

e_1 `AND` e_2 gives `true` if e_1 as well as e_2 gives `true`, and `false` otherwise.

e_1 `OR` e_2 gives `true` if at least one of e_1 and e_2 gives `true`, and `false` otherwise.

Find the type of the function `eval`.

4C: Declare a function `toString` on Boolean expressions such that `toString(`e`)` is a textual representation of the Boolean expression e. The expression `VAR` s is represented by the string s, while the symbols `not`, `and` and `or` are used for the other constructors. The solution is allowed to produce strings containing superfluous brackets.

4D: Declare a function `noOfANDs` on Boolean expressions such that `noOfANDs(`e`)` is the number of occurrences of `AND` in the Boolean expression e.