Uppsala University
ITP1 Programkonstruktion, del 2 + DVP1 Programmeringsmetodik 1
Period 3, Spring 2002 + Fall 2001
Exam 2 + Exam 2'
Wednesday 5 June 2002, from 15:00 to 20:00

Global Instructions

Read these instructions, as well as the actual questions, very carefully **before** attempting to solve the problems. Especially pay attention to **stressed** words (in boldface). The questions have been engineered to have many short and elegant answers. If you get into some lengthy or difficult reasoning, you are probably on the wrong track and might benefit from re-reading the question.

This question set is double-sided. To the extent possible, write your answers into the gaps. The provided space is really sufficient each time. Write your name onto **every** sheet. This is an exam with **closed** books and notes. An English-Swedish dictionary is available at the front desk. Normally, the instructor will come and answer questions between 17:00 and 18:00.

Provide a specification (with at least the names of the argument components, a signature, a pre-condition, a post-condition involving **all** the names of the argument components, and **useful** examples) for **every** function you construct. Each specification **must** be suitable for justifying your function or for constructing another function. Provide a justification outline (with the chosen induction parameter and the chosen well-founded relation) for **every recursive** function you construct. You often need **not** provide any other justifications, but the given ones **must** correspond to your function. For instance, each clause should **not** be redundant with the other clauses. Failure to provide such a specification or justification outline for at least one function of a sub-question will result in zero points for that entire sub-question, even if the program is actually correct.

You may **only** use the functions and directives of the **standard** library of SML, as well as `Math.sqrt`. Do **not** use higher-order functions, **except** where requested. For instance, the instructor's solutions to the questions only involve `+`, `-`, `*`, `::`, `@`, `abstype`, `as`, `fn`, `foldl`, `foldr`, `fun`, `hd`, `infix`, `let…in…end`, `list`, `map`, `Math.sqrt`, `of`, `op`, `tl`, and `val`. Layout is unimportant, but please be considerate.

Unless otherwise posted, the instructor is **only** interested in correct SML functions. Any attempts at efficient functions are purely at your own risk, namely the risk of missing out on correctness or of losing time.

ITP1: The 2.2 credit points for this exam are awarded for VT02 if the sum of your exam points and bonus points is in the interval [55,100]. A "*med beröm godkänd*" (5) grade is earned if this sum is in [85,100], while an "*icke utan beröm godkänd*" (4) grade is earned if this sum is in [70,84], and a "*godkänd*" (3) grade is earned if this sum is in [55,69]. Otherwise, an "*underkänd*" (U) grade is earned.

DVP1: The 4 credit points for this exam are awarded for HT01 if the sum of your exam points and bonus points is in the interval [55,100]. A *very-good* grade is earned if this sum is in the interval [75,100], while a *good* grade is earned if this sum is in the interval [55,74]. Otherwise, an "*underkänd*" (U) grade is earned.

For official use (do not write below this line):

Q1	Q2	Exam
/ 20	/ 60	/ 80

Background

A *matrix* is a rectangular array of elements that is arranged in rows and columns. A matrix with m rows and n columns is called an $m \times n$ matrix. If A is a matrix, then a_{ij} denotes the element in its i^{th} row and j^{th} column. The rows of an $m \times n$ matrix are n-dimensional vectors, and its columns are m-dimensional vectors. The *dot product* of two vectors $(x_1,x_2,\ldots,x_n)$ and $(y_1,y_2,\ldots,y_n)$ is the scalar $x_1y_1 + x_2y_2 + \ldots + x_ny_n$. The *product* of an $m \times n$ matrix A by an $n \times p$ matrix B, denoted $A \cdot B$, is an $m \times p$ matrix C whose element c_{ij} is the dot product of the i^{th} row of A and the j^{th} column of B. The *transpose* of an $m \times n$ matrix A, denoted A', is the $n \times m$ matrix obtained by converting the rows of A into columns. The *norm* of an $m \times n$ matrix A, denoted $|A|$, is the scalar $\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^{2}}$. For example, $\begin{bmatrix}2 & 3\\ -1 & 4\end{bmatrix} \cdot \begin{bmatrix}5 & -2 & 1\\ 3 & 8 & -6\end{bmatrix} = \begin{bmatrix}19 & 20 & -16\\ 7 & 34 & -25\end{bmatrix}$, and $\begin{bmatrix}5 & -2 & 1\\ 3 & 8 & -6\end{bmatrix}' = \begin{bmatrix}5 & 3\\ -2 & 8\\ 1 & -6\end{bmatrix}$, and $\left|\begin{bmatrix}2 & 3\\ -1 & 4\end{bmatrix}\right| = \sqrt{30}$.

Question 1 Specification of an ADT (20 points)

Specify an SML abstract datatype (ADT) — called `'a mat` — for matrices, with the following functions:

(4 points) a. A function `mat2list`, which converts a matrix `A` into a list by appending its row vectors top-down.

function

pre:

post:

example:

(6 points) b. A curried function `list2mat`, which, given a pair `(m,n)` of positive integers and a list `L` of `m * n` elements, converts `L` into the `m` × `n` matrix `A` so that `mat2list A = L`.

function

pre:

post:

example:

(3 points) c. A function `norm`, which returns the norm of a real-number matrix `A`.

function

pre:

post:

(3 points) d. A function `transpose`, which returns the transpose of a matrix `A`.

function

pre:

post:

(4 points) e. An infix function `times`, which returns the product of two integer matrices `A`, `B`, assuming it is defined.

function

pre:

post:

Question 2 A Realisation of the ADT (60 points)

Realise the `mat` ADT, using a representation that is based on lists of lists. An $m \times n$ matrix A, with $m > 0$ and $n > 0$, is to be represented by `MAT` `[[`a_{11}`,`$\ldots$`,`a_{1n}`],...,[`a_{m1}`,`$\ldots$`,`a_{mn}`]]`. Protect the **representation invariant** that there are $m > 0$ element lists, all of the same length $n > 0$. Answer the following sub-questions:

(1 point) f. Declare the realisation of the `mat` ADT.

```
abstype 'a mat =
with (* here comes the code of the other sub-questions *) end
```

(3 points) g. Realise the `mat2list` function. Use **no** recursion. Use one or more of the **standard** higher-order functions `map`, `foldr`, and `foldl`. Introduce **no** new functions.

```
fun
```

(12 points) h. Realise the `list2mat` function. Use **recursion**. Introduce **at most one** new function.

```
fun
```

```
by simple induction on: m
well-founded relation: <
```

If you introduced a (recursive) new function, then give a **most general** specification and implement it here:

```
function
pre:
post:

example:
fun
```

```
by induction on:
well-founded relation:
```

(6 points) i. Realise the `norm` function. Use **no** recursion. Use one or more of the **standard** higher-order functions `map`, `foldr`, and `foldl`. Introduce **no** new functions. You **may** use other functions from the ADT.

```
fun
```

(14 points) j. Realise the `transpose` function. Build the transpose row by row, that is reduce the given matrix by traversing its columns. Use **recursion**. Introduce **at most one** new function.

```
example: transpose (MAT [[5,~2,1],[3,8,~6]])
                  = MAT [[5,3],[~2,8],[1,~6]]
fun
```

```
by induction on: A
well-founded relation: has one column less than
```

If you introduced a (recursive) new function, then give a **most general** specification and implement it here:

```
function
pre:
post:

example:
fun
```

```
by induction on:
well-founded relation:
```

In what sense is the given example of `transpose` **not** really an example?

(17 points) k. Specify and implement an **infix** function `dot` for the dot product of two integer vectors of the same length, represented as integer lists. Use **recursion**. Introduce **no** new functions.

function

pre:

post:

example:

fun

by induction on:

well-founded relation:

Is this function tail-recursive or not? Why?

If not, then specify a generalisation, called `dot''`, of `dot` and implement it using **tail-recursion**:

function

pre:

post:

example:

fun

by induction on:

well-founded relation:

Non-recursively re-realise the `dot` function, calling it `dot'` now but with the same specification as `dot`, using only `dot''`:

```
val dot' =
```

(7 points) l. To realise the `times` function, let us introduce a new, similar **infix** function `mult` that takes the transpose of the second matrix so as to get more convenient access to its columns.

```
example: (MAT [[2],[~1]]) times (MAT [[3,4]]) = (MAT [[6,8],[~3,~4]])
infix times
fun A times B = A mult (transpose B)
```

Give a **most general** specification of `mult` and implement it here. Use **recursion**. Use the `dot` function. Use the **standard** higher-order function `map` to avoid introducing another new function.

```
function A mult B :
pre:
post:
example: (MAT [[2],[~1]]) mult (MAT [[3],[4]])
        = (MAT [[6,8],[~3,~4]])

fun







by induction on: A
well-founded relation: has one row less than
```

You may draw pictures or take scratch notes below this line!