Uppsala University
MN Functional Programming
Period 2, Autumn 2004
Exam 1
Saturday 18 December 2004, from 09:00 to 14:00

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 may be available at the front desk. Normally, the instructor will come and answer questions between 11:00 and 12:00.

To save time, program in a **non-defensive** style. Provide a specification (at least the names of the argument components, the type, the pre-condition, a post-condition involving **all** the names of the argument components, and **useful** examples) for **every** SML function you construct. Each specification **must** be suitable for justifying your SML function or for constructing another one. Provide a justification outline (the chosen variant) for **every recursive** SML function you construct. You need **not** provide any other justifications, but the given ones **must** correspond to your SML function. For instance, each clause should **not** be redundant with previous clauses. Failure to provide such a specification or justification outline for at least one SML function of a sub-question will result in zero points for that entire sub-question, even if the program is actually correct. If you cannot comply with a requirement of a sub-question, such as the presence or absence of recursion, the indicated variant, or the number of help functions, then **explicitly** lift that requirement and proceed without it.

You may **only** use the directives and functions of the **standard** library of SML. Do **not** use higher-order functions, **except** where explicitly requested. Exact SML syntax is not required. Layout is unimportant, but please be considerate.

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

The 4 credit points for this exam are awarded if your exam points are in the interval [50%,100%] of the maximum score. Furthermore, a *very-good* (VG) pass grade is earned for the interval [85%,100%] of the maximum score, while a *good* (G) pass grade is earned for the interval [50%,84%] of the maximum score. Otherwise, an "*underkänd*" (U) fail grade is earned.

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

Q1	Q2	Q3	Exam
/ 34	/ 28	/ 18	/ 80

Question 1 Methodology and Recursion (34 points)

Given two natural numbers *n* and *m*, such that the decimal representation of *n* has at most *m* digits (by convention, the decimal representation of 0 is empty), the function *decRepr* returns the list of *m* digits with the right-aligned decimal representation of *n*, with leading zeroes if necessary. From Question 17 of the *Google Labs Aptitude Test* (GLAT): given a natural number *n*, the function f returns the number of ones required when writing out [the decimal representations of] all [natural] numbers between 0 and *n* [inclusive].

Using the concept names above, answer the following sub-questions.

(12 points) a. Construct a **recursive** curried SML function for *decRepr*. Use *n* as **variant**. Use **at most one** help function. (**Hint**: Recall that 123 *div* 10 = 12 and 123 *mod* 10 = 3.)

```
function decRepr n m
type:
pre:
post:
ex: decRepr 522 7 = [0,0,0,0,5,2,2]
fun decRepr
```

```
variant: n
```

If you needed a help function, then give its **most general** specification and construct it here.

```
function
type:
pre:
post:
ex:
fun
```

```
variant:
```

(9 points) b. Is this SML function for *decRepr* tail-recursive or not? Why / Why not?

If not, then specify a **descending** generalisation (which introduces an accumulator), called *decRepr'*, of *decRepr* and construct a **tail-recursive** SML function for it.

```
function decRepr'
type:
pre:
post:
ex:
fun decRepr'
```

```
variant:
```

Non-recursively re-realise the *decRepr* function. Use **only** *decRepr'*.

```
fun decRepr n m =
```

(13 points) c. Construct a **recursive** SML function for the Google Labs function *f*. Use **at most one** help function. (**Hint**: Recall that 123 *div* 10 = 12 and 123 *mod* 10 = 3.)

```
function f n
type:
pre:
post:
ex: f 13 = 6
fun f
```

```
variant:
```

If you needed a help function, then give its **most general** specification and construct it here.

function

type:

pre:

post:

ex:

fun

variant:

Question 2 Trees (28 points)

A *binary tree* is either empty, or non-empty with a root node as well as left and right binary trees as subtrees. The *prefix walk* of a binary tree *B* is a list where the root of *B* appears just before the prefix walk of its left subtree followed by the prefix walk of its right subtree. A *variadic tree* is non-empty with a root node as well as any number of variadic trees as subtrees; this number may vary from node to node. The *prefix walk* of a variadic tree is defined analogously.

Using the concept names above, answer the following sub-questions.

(3 points) d. Declare an SML type for binary trees of elements of **any** type, for use in other sub-questions.

convention:

invariant:

datatype

(7 points)

e. Construct a **recursive** SML function computing the prefix walk of a binary tree. Use **no** help functions.

```
function prefix b
type:
pre:
post:
val rec prefix =
```

```
variant:
```

Is this SML function for *prefix* tail-recursive or not? Why / Why not?

(10 points)

f. Specify the generalisation, called *prefix*', of *prefix* that computes the concatenation of the prefix walks of the elements of a **list** of binary trees (this is called a **tupling** generalisation). Construct a **recursive** SML function for *prefix*'. Use **no** help functions. Use **at most one** recursive call per clause.

```
function prefix'
type:
pre:
post:
fun prefix'
```

```
variant:
```

Is this SML function for *prefix*' tail-recursive or not? Why / Why not?

Non-recursively re-realise the *prefix* function. Use **only** *prefix*'.

```
fun prefix b =
```

Is this second SML function for *prefix* more or less efficient than the first one for *prefix*? Why?

(3 points) g. Declare an SML type for variadic trees of elements of **any** type, for use in the last sub-question.

```
convention:


invariant:


datatype
```

(5 points) h. Using **standard** higher-order functions (*map*, *foldl*, and/or *foldr*), construct an SML function called *prefix''* that computes the prefix walk of a variadic tree. Use **no** help functions.

```
function prefix'' v
type:
pre:
post:
fun prefix''
```

```
variant (if any):
```

Question 3 Integer Sets (18 points)

Let us start developing an SML abstract datatype, called *intSet*, for integer sets. Given an integer i and an integer set S that may or may not contain i, the function *delete* returns the integer set $S - \{i\}$.

Answer the following sub-questions.

(4 points) i. **Without** reading sub-question j, specify the *delete* function.

```
function delete S i
type:
pre:
post:
ex:
```

(6 points)

j. Declare an SML abstract datatype for integer sets, for use in the last sub-question. It is based on a representation using intervals: for example, the set {1,3,5,6,7,8,9,10,11,12,13,14,15,17} will be represented by the term IS [*Interval*(1,1), *Interval*(3,3), *Interval*(5,15), *Interval*(17,17)]. Ensure that the representation of a set is **unique**.

convention:

invariant:

```
abstype intSet =

with (* here comes the code of the last sub-question *) end
```

(8 points)

k. Construct a **recursive** SML function for *delete*. Use **no** help functions.

```
fun delete
```

variant: