Uppsala University
ITP1 Programkonstruktion, del 1
Periods 1 & 2 of Fall 2001
Exam 2
Friday 11 January 2002, from 9:00 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 concise and elegant answers, so if you get into some messy reasoning, you are probably on the wrong track and would 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 amply 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 11:00 and 12:00.

Provide a specification (with at least the names of the argument components, a signature, a pre-condition, and a post-condition involving **all** the names of the argument components) for **every** program you construct, such that this specification would be suitable for justifying your program or constructing another program. Provide a justification skeleton (the chosen induction parameter and well-founded relation) for **every** recursive program you construct. You often need not provide any other justifications, but the given ones must correspond to your program: for instance, each base case should **not** be redundant with the other cases. Failure to provide such a specification or justification skeleton 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 of the **standard** library of SML. For instance, the instructor's solutions to the questions below only involve `+`, `-`, `*`, `=`, `<>`, `::`, `@`, `div`, `hd`, `if`…`then`…`else`…, `mod`, and `orelse`. Layout is unimportant, but please be considerate.

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

The two 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]. Furthermore, a "med beröm godkänd" (5) grade is earned if this sum is in the interval [85,100], while an "icke utan beröm godkänd" (4) grade is earned if this sum is in the interval [70,84], and a "godkänd" (3) grade is earned if this sum is in the interval [55,69]. In all other cases, an "underkänd" (U) grade is earned.

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

Q1	Q2.a	Q2.b	Q2.c	Q2.d	Exam
/ 20	/ 9	/ 9	/ 18	/ 24	/ 80

Question 1 Type Inference, Reduction, Currying (20 pts)

Consider the following declarations:

```
fun strange x = ((fn y => y-1) x) * ((fn y => y+1) x) + 1
fun triple (g,z) = (1+1) * g z
fun max2 (x,y) = if x>y then x else y
fun max4 (v,w) = max2 (max2 v, max2 w)
fun max5 (u,t) = max2 (u, max4 t)
```

Answer the following sub-questions:

a. Complete the following specification for `strange`, simplifying the post-condition as much as possible:

```
function strange n :          →
pre: (none)
post:
```

b. Complete the following specification for `triple`:

```
function triple (f,x) :                  →
pre: f is defined on x
post:
```

c. **Step-by-step** reduce the expression `triple (strange,4)`.

To get **all** the points, remember that `strange` and `triple` are actually names for anonymous functions.

d. Give the signatures of `max2`, `max4`, and `max5`:

```
function max2 (a,b) :
function max4 (a,b) :
function max5 (a,b) :
```

Give the curried signature of `max2`, calling it `max2C` now:

```
function max2C
```

Give the curried signature of `max4`, calling it `max4C` now, assuming it uses `max2C` rather than `max2`:

```
function max4C
```

Question 2 Methodology and Recursion (60 points)

Definitions

The *factorial* $n!$ of a natural number n is 1 when $n = 0$, and $n * (n - 1)!$ otherwise.

For example, the factorial of 5 is $5 * 4 * 3 * 2 * 1 * 1$, that is 120.

The *pseudo-factorial* of two natural numbers a and b, such that $a \leq b$, is $b! / a!$.

For example, the pseudo-factorial of 3 and 5 is 20.

A *root* of a function f is an element r of the domain of f such that $f(r) = 0$.

For example, the roots of $x^2 - 8 * x + 15$ are 3 and 5.

A *segment* of a list is a prefix of a suffix of that list.

For example, the lists [], [4,5], and [2,1,4,5,3] are segments of [2,1,4,5,3].

A *plateau* of a list is a segment thereof with all-equal elements but different previous and next elements, if any.

For example, the list [3,3] is a plateau of [1,3,3,2,2,2,5], but its segments [3,3,2] and [3] are not plateaus thereof.

Questions

Using the definitions above, answer the following sub-questions, by programming in a **non-defensive** style and by **exploiting** the pre-conditions, if any:

a. Construct a **recursive** program computing the pseudo-factorial of two natural numbers.

Do **not** specify and construct any additional functions.

```
function pseudoFact (a,b) :              →
pre:
post:
fun pseudoFact
```

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

The recursive call respects the pre-condition because …

The recursion is terminating because …

b. Construct a **recursive** program computing a root lying within a closed integer interval [*a*,*b*] of the domain of a total integer-to-integer function *f*, **assuming** that such a root exists in [*a*,*b*] and that $a \leq b$.
Do **not** specify and construct any additional functions.

```
function root f (a,b) :                    →
pre:
post:
fun root
```

```
by simple induction on: the size of the interval [a,b], denoted #[a,b]
well-founded relation: <
```

The recursive call respects the pre-condition because …

The recursion is terminating because …

c. Construct a **recursive** program computing the number of plateaus as well as the length of the longest plateau of a list of elements of **any** type.

```
function plateaus L :          →
pre:
post:
Examples: plateaus [] = (0,0) ;  plateaus [1,3,3,2,2,2,5] = (4,3)
fun plateaus
```

```
by simple induction on: the number of plateaus of L
well-founded relation: <
```

If you need an additional **recursive** function, first specify and then construct it here:

function : →

pre:

post:

Examples:

fun

by induction on:

well-founded relation:

d. Construct **two** different **recursive** programs that, given two natural numbers *m* and *n* such that the decimal representation of *m* has at most *n* digits, compute the integer list of *n* elements that is the right-aligned decimal representation of *m*, with leading zeroes.

The **common** specification:

function decRepr (m,n) : →

pre:

post:

Examples: decRepr (0,3) = [0,0,0]; decRepr (522,6) = [0,0,0,5,2,2]

Counterexamples: decRepr (52,3) ≠ [5,2,0]; decRepr (52,3) ≠ [~1,5,2]

My **first** program:

val rec decRepr = (* **rec** means the value is recursively defined *)

by complete induction on: m

well-founded relation: <

This is **complete** induction because …

If you need an additional **recursive** function, first specify and then construct it here:

```
function              :          →
pre:

post:

Examples:

fun
```

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

My **second** program, for which I do **not** specify and construct any additional functions:

```
fun decRepr
```

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

This is **simple** induction because …