The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, SPRING SEMESTER 2013-2014

G54FOP FOUNDATIONS OF PROGRAMMING

Time allowed 2 hours

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced

Answer all 4 questions
Marks available for sections of questions are shown in square brackets.

No calculators are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn your examination paper over until instructed to do so

Question 1: [25 pts]

Representation of lists in the untyped λ-calculus

Lists are constructed by using two combinators, nil for the empty list and cons for the construction of a new list from a head and a tail. At first we represent a list by repeated pairing:

$$\begin{aligned}\mathsf{nil} &= \mathsf{false} = \lambda x.\lambda y.y\\ \mathsf{cons} &= \lambda h.\lambda t.(h,t) = \lambda h.\lambda t.\lambda x.x\,h\,t\end{aligned}$$

(a) [9 pts]

Define combinators (pure λ-terms) head and tail for the head and tail of a list, respectively. They must satisfy these reduction relations (it doesn't matter what they do on empty lists):

$$\mathsf{head}\,(\mathsf{cons}\,h\,t) \rightsquigarrow^* h \qquad \mathsf{tail}\,(\mathsf{cons}\,h\,t) \rightsquigarrow^* t$$

Define a combinator isnil that checks if a list is empty. It must satisfy these reduction relations:

$$\mathsf{isnil}\,\mathsf{nil} \rightsquigarrow^* \mathsf{true} \qquad \mathsf{isnil}\,(\mathsf{cons}\,h\,t) \rightsquigarrow^* \mathsf{false}$$

(b) [9 pts]

Implement a recursor for lists: for every pair of terms c and g, define a combinator $(\mathsf{lrec}\,c\,g)$ satisfying these reduction relations:

$$\begin{aligned}&(\mathsf{lrec}\,c\,g)\,\mathsf{nil} \rightsquigarrow^* c\\ &(\mathsf{lrec}\,c\,g)\,(\mathsf{cons}\,h\,t) \rightsquigarrow^* g\,h\,t\,((\mathsf{lrec}\,c\,g)\,t)\end{aligned}$$

In your solution, you can use the fixpoint combinator:

$$\mathsf{Y} = \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x)).$$

(c) [7 pts]

Now write an alternative implementation of lists as iterators. Define the new list constructors $\mathsf{nil}_{\mathsf{IT}}$ and $\mathsf{cons}_{\mathsf{IT}}$ so that they have the following reduction behaviour (for all terms c and g):

$$\begin{aligned}&\mathsf{nil}_{\mathsf{IT}}\,c\,g \rightsquigarrow^* c\\ &(\mathsf{cons}_{\mathsf{IT}}\,h\,t)\,c\,g \rightsquigarrow^* g\,h\,(t\,c\,g)\end{aligned}$$

Continued on next page

Question 2: [25 pts]

SIMPLY TYPED λ-CALCULUS AND SYSTEM T

(a) [10 pts]

Give a short precise definition of the following properties that a type system may or may not satisfy:

- Confluence
- Weak Normalization
- Strong Normalization
- Progress
- Subject Reduction (Preservation)

(b) [6 pts]

The following are incomplete terms in the simply typed λ-calculus with a base type o. Rewrite the terms, replacing each question mark with the type of the corresponding abstracted variable; also state what type each resulting term has:

$$\lambda x :?.\ \lambda y :?.\ \lambda z :?.\ x\ (z\ y)\ y$$
$$\lambda x :?.\ \lambda y :?.\ y\,(x\,y)$$
$$\lambda x :?.\ \lambda y :?.\ \lambda z :?.\ z\,(y\,z\,(x\,z))$$

(c) [9 pts]

The aim of this part is to program the Fibonacci sequence in system T. The common recursive implementation uses pairs, but system T doesn't have Cartesian products. Instead we use higher-order recursion to define an auxiliary function that takes the initial values as arguments:

$\mathsf{fib} : \mathsf{Nat} \to \mathsf{Nat}$	$\mathsf{fib}_{\mathsf{HO}} : \mathsf{Nat} \to (\mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat})$
$\mathsf{fib}\,0 = 0$	$\mathsf{fib}_{\mathsf{HO}}\,0\,a\,b = a$
$\mathsf{fib}\,1 = 1$	$\mathsf{fib}_{\mathsf{HO}}\,(\mathsf{S}\,n)\,a\,b = \mathsf{fib}_{\mathsf{HO}}\,n\,b\,(a+b)$
$\mathsf{fib}\,(\mathsf{S}\,(\mathsf{S}\,n)) = (\mathsf{fib}\,n) + (\mathsf{fib}\,(\mathsf{S}\,n))$	then $\mathsf{fib} = \lambda n.\mathsf{fib}_{\mathsf{HO}}\,n\,0\,1$

Give the definition of $\mathsf{fib}_{\mathsf{HO}}$ in system T.

Question 3: [25 pts]

INDUCTIVE TYPES AND SYSTEM F

We define a type of games with two players. When the game finishes, each player will receive a certain payoff. On each player's turn, they can choose between two new positions, the *left* or the *right* move.

We implement game positions by the inductive type with the following introduction rules:

$$\frac{x, y : \mathsf{Nat}}{(\mathsf{end}\, x\, y) : \mathsf{Game}} \qquad \frac{r, l : \mathsf{Game}}{(\mathsf{play}\, r\, l) : \mathsf{Game}}$$

(a) [8 pts]

Write the elimination and reduction rules for the type Game.

(b) [9 pts]

Define the type Game in system F and implement the recursor (from the elimination rule). [Remember the method to define a recursor from an iterator: define an auxilliary function that gives as result a pair whose first element is a copy of the input.]

(c) [8 pts]

A strategy is a function that tells a player how to choose between two game positions:

$$\mathsf{Strategy} = \mathsf{Game} \to \mathsf{Game} \to \mathsf{Bool}.$$

(If the result is true the first position is chosen, if it is false the second position is chosen.)

Write (using the recursor from the elimination rule) a function that, given strategies for the two players, computes the outcome of a game:

$$\mathsf{outcome} : \mathsf{Game} \to \mathsf{Strategy} \to \mathsf{Strategy} \to \mathsf{Nat} \times \mathsf{Nat}.$$

Continued on next page

Question 4: [25 pts]

Coinductive Types

Consider the two coinductive types of streams and of infinite trees on a base type A, defined by:

$$\mathbb{S}_A = \nu X.A \times X \qquad \mathbb{T}_A = \nu X.A \times X \times X.$$

(a) [8 pts]

For each of the two types write down its formal rules (elimination, introduction, reduction).

[For the rest of the Question, you can use the functions head, tail and $\prec$ for streams; and label, lchild, rchild and node for trees.]

(b) [8 pts]

Using the rules from part (a), implement the interleave function on streams:

$$\mathsf{interleave} : \mathbb{S}_A \to \mathbb{S}_A \to \mathbb{S}_A$$

That takes two streams $a_0 \prec a_1 \prec a_2 \prec \cdots$, $b_0 \prec b_1 \prec b_2 \prec \cdots$ and returns the stream $a_0 \prec b_0 \prec a_1 \prec b_1 \prec a_2 \prec b_2 \prec \cdots$.

Also write two functions that select the elements on even and odd positions:

$$\mathsf{evens} : \mathbb{S}_A \to \mathbb{S}_A \qquad \mathsf{odds} : \mathbb{S}_A \to \mathbb{S}_A$$

(c) [9 pts]

Write two functions that convert a tree to a stream and vice-versa [the label is the head of the stream, the left subtree contains the odd elements, the right subtree contains the (nonzero) even elements]:

$$\mathsf{stream\_tree} : \mathbb{S}_A \to \mathbb{T}_A \qquad \mathsf{tree\_stream} : \mathbb{T}_A \to \mathbb{S}_A$$

[Hint: For $\mathsf{tree\_stream}$ you should extend the function to operate on the type $\mathsf{Tree}_{\mathbb{T}_A}$ of finite trees with infinite trees as leaves. You can use informal pattern-matching notation to define a coalgebra on it.]