Final examination — given Tuesday 14th December

General instructions

Closed-book, closed-notes, closed-computer, in-class exam.

Time allowed: 120 minutes.

Total points available: 200 pts.

Answer in the spaces provided.

Your name (print):

I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual.

Please sign and date:

3.1 List functionals in Haskell (30 pts)

1. (10 pts) Define a Haskell function

   ```
   flatten :: [[a]] -> [a]
   ```

   that combines the elements from a list of nested lists into a single list containing the same elements in the same order.

   For example, `flatten [[1,2,3],[],[4,5]]` should evaluate to `[1,2,3,4,5]`.

2. (20 pts) Write down the result of evaluating the following Haskell expressions, which might be **returning a calculated value** (in which case you must **state** that value), **raising a runtime exception** (in which case you must **explain** the error), or **an infinite loop** (in which case you must **explain** why the expression fails to terminate).

   - `foldr (-) 0 [1,2,3]`

   - `foldl (-) 0 [1,2,3]`

   - `sum (drop 3 [2,4..])`

   - `sum (take 3 [2,4..])`

3.2 Proving properties of Haskell list functionals (20 pts)

Prove, by induction on the structure of lists, that

```
foldr (\x -> \acc -> (f x):acc) [] xs = map f xs
```

for all lists `xs :: [a]` and functions `f :: a -> b`.

You must **clearly state** your inductive hypothesis in the inductive case of your proof.

3.3 Programming in the untyped lambda-calculus (30 pts)

1. (10 pts) Write down definitions for the untyped lambda-terms corresponding to the Church numerals c_0, c_1, c_2, and c_3, which represent the natural numbers 0, 1, 2, and 3, respectively.

2. (20 pts) The "predecessor" function (*pred*) over Church numerals (c_i) in the untyped lambda-calculus was defined in class as follows:

$$pred = \lambda n.\, fst\ (n\ ss\ zz)$$

where

$$zz = pair\ \mathsf{c}_0\ \mathsf{c}_0$$
$$ss = \lambda p.\, pair\ (snd\ p)\ (succ\ (snd\ p))$$

and where *pair*, *fst*, and *snd* implement pairing, projection of the first element of a pair, and projection of the second element of a pair, respectively.

Briefly explain how and why this implementation of the predecessor function works as expected, using the *pred* c_0 and *pred* c_2 examples to **illustrate your answer**.

3.4 Typing and evaluation in the simply-typed lambda-calculus (40 pts)

1. (15 pts) Using the typing rules for the simply-typed lambda calculus with numbers and booleans (presented in Appendix A), **attempt to construct a typing derivation for the following term**, starting in the empty typing environment:

$$(\lambda p : \mathsf{Nat} \to \mathsf{Bool}.\ \mathsf{if}\ p\ 0\ \mathsf{then\ false\ else\ true})\ (\lambda n : \mathsf{Nat}.\ \mathsf{iszero}\ (\mathsf{pred}\ n))$$

   If your derivation is successful, clearly state this and identify the resulting type.

   If your derivation fails, construct as much of the derivation as possible and explain where and why the derivation fails to assign a type.

2. (25 pts) Using the call-by-value evaluation rules for the simply-typed lambda calculus with numbers and booleans (presented in Appendix A), **construct a full derivation for *each* step of reduction of the following term**:

$$\left(\lambda p : \mathsf{Nat} \to \mathsf{Bool}.\ \mathsf{if}\ p\ 0\ \mathsf{then\ false\ else\ true}\right)\ \left(\lambda n : \mathsf{Nat}.\ \mathsf{iszero}\ (\mathsf{pred}\ n)\right)$$

   Identify when no more reduction steps can be taken and whether the resulting term is a **value** or whether the term is **stuck** at a non-value term.

   You should attempt to reduce this term **regardless** of whether you found it to be well-typed or not in part 1.

3.5 General recursion in the simply-typed lambda-calculus (40 pts)

1. (15 pts) Compare and contrast the broad approaches to implementing recursive functions in the untyped lambda-calculus and the simply-typed lambda-calculus, as discussed during the course.

   Note that you **do not** need to provide explicit definitions or examples here.

2. (25 pts) Let g stand for the following function definition in the simply-typed lambda calculus extended with general recursion (as defined in Appendix A and Appendix B):

$$\lambda f : \mathsf{Nat} \to \mathsf{Bool}.\ \lambda n : \mathsf{Nat}.\ \mathsf{if}\ (\mathsf{iszero}\ n)\ \mathsf{then\ false\ else}\ (\mathsf{if}\ f\ (\mathsf{pred}\ n)\ \mathsf{then\ false\ else\ true}).$$

Then, **sketch the overall structure of the reduction sequence** for the following term:

$$(\mathsf{fix}\ g)\ (\mathsf{succ}\ 0)$$

You **do not** need to present every single step or construct full derivations. However, you **must explain the key steps** involved in the operation of fix. You should conclude by **determining what predicate is computed by fix g.**

3.6 Concepts (40 pts)

Briefly explain the following concepts. **Illustrate each answer with an explanation of a simple example.**

1. (10 pts) lazy evaluation in Haskell.

2. (10 pts) higher-order functions.

3. (10 pts) full beta-reduction.

4. (10 pts) Church numerals in the untyped lambda calculus.

END OF EXAM

A Reference: simply typed lambda calculus with booleans and numbers

Syntax

Terms, t	$::=$	x	*variable*
	$\mid$	$\lambda x : T.\, t$	*abstraction*
	$\mid$	$t\ t$	*application*
	$\mid$	true	*constant true*
	$\mid$	false	*constant false*
	$\mid$	if t then t else t	*conditional*
	$\mid$	0	*constant zero*
	$\mid$	succ t	*successor*
	$\mid$	pred t	*predecessor*
	$\mid$	iszero t	*zero test*
Values, v	$::=$	$\lambda x : T.\, t$	*abstraction value*
	$\mid$	true	*true value*
	$\mid$	false	*false value*
	$\mid$	nv	*numeric value*
Numeric values, nv	$::=$	0	*zero value*
	$\mid$	succ nv	*successor value*
Types, T	$::=$	$T \to T$	*type of functions*
	$\mid$	Bool	*type of booleans*
	$\mid$	Nat	*type of natural numbers*
Typing contexts, Γ	$::=$	$\varnothing$	*empty context*
	$\mid$	$\Gamma, x : T$	*variable type assumption*

Capture-Avoiding Substitution

$$
\begin{array}{rcl}
[x \mapsto t]x & = & t \\
[x \mapsto t]y & = & y \qquad \text{if } x \neq y \\
[x \mapsto t](t_1\ t_2) & = & ([x \mapsto t]t_1)\ ([x \mapsto t]t_2) \\
[x \mapsto t](\lambda x{:}T.\, t') & = & \lambda x{:}T.\, t' \\
[x \mapsto t](\lambda y{:}T.\, t') & = & \lambda y{:}T.\, ([x \mapsto t]t') \qquad \text{if } x \neq y \\
[x \mapsto t]\mathsf{true} & = & \mathsf{true} \\
[x \mapsto t]\mathsf{false} & = & \mathsf{false} \\
[x \mapsto t](\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3) & = & \mathsf{if}\ ([x \mapsto t]t_1)\ \mathsf{then}\ ([x \mapsto t]t_2)\ \mathsf{else}\ ([x \mapsto t]t_3) \\
[x \mapsto t]0 & = & 0 \\
[x \mapsto t](\mathsf{succ}\ t') & = & \mathsf{succ}\ ([x \mapsto t]t') \\
[x \mapsto t](\mathsf{pred}\ t') & = & \mathsf{pred}\ ([x \mapsto t]t') \\
[x \mapsto t](\mathsf{iszero}\ t') & = & \mathsf{iszero}\ ([x \mapsto t]t')
\end{array}
$$

NB: this definition is capture avoiding if we assume that t is a closed term.

Evaluation Rules

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2}\ (\text{E-App1})$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'}\ (\text{E-App2})$$

$$\frac{}{(\lambda x{:}T_{11}.\, t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12}}\ (\text{E-AppAbs})$$

$$\frac{}{\mathsf{if\ true\ then}\ t_2\ \mathsf{else}\ t_3 \longrightarrow t_2}\ (\text{E-IfTrue})$$

$$\frac{}{\mathsf{if\ false\ then}\ t_2\ \mathsf{else}\ t_3 \longrightarrow t_3}\ \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \longrightarrow \mathsf{if}\ t_1'\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3}\ \text{(E-If)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{succ}\ t_1 \longrightarrow \mathsf{succ}\ t_1'}\ \text{(E-Succ)}$$

$$\frac{}{\mathsf{pred}\ 0 \longrightarrow 0}\ \text{(E-PredZero)}$$

$$\frac{}{\mathsf{pred}\ (\mathsf{succ}\ nv_1) \longrightarrow nv_1}\ \text{(E-PredSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{pred}\ t_1 \longrightarrow \mathsf{pred}\ t_1'}\ \text{(E-Pred)}$$

$$\frac{}{\mathsf{iszero}\ 0 \longrightarrow \mathsf{true}}\ \text{(E-IsZeroZero)}$$

$$\frac{}{\mathsf{iszero}\ (\mathsf{succ}\ nv_1) \longrightarrow \mathsf{false}}\ \text{(E-IsZeroSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{iszero}\ t_1 \longrightarrow \mathsf{iszero}\ t_1'}\ \text{(E-IsZero)}$$

Typing Rules

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-VAR)}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2 \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \lambda x{:}T_1.\, t_2 : T_1 \to T_2} \text{ (T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \text{ (T-APP)}$$

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}} \text{ (T-TRUE)}$$

$$\frac{}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}} \text{ (T-FALSE)}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 : T} \text{ (T-IF)}$$

$$\frac{}{\Gamma \vdash 0 : \mathsf{Nat}} \text{ (T-ZERO)}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\, t : \mathsf{Nat}} \text{ (T-SUCC)}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{pred}\, t : \mathsf{Nat}} \text{ (T-PRED)}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{iszero}\, t : \mathsf{Bool}} \text{ (T-ISZERO)}$$

B Reference: extending the simply typed lambda-calculus with general recursion

Extended Syntax

$$\text{Extended terms, } t \quad ::= \quad \cdots \mid \mathsf{fix}\, t \qquad\qquad \textit{fixed point of } t$$

Extended Capture-Avoiding Substitution

$$[x \mapsto t](\mathsf{fix}\, t') \quad = \quad \mathsf{fix}\, ([x \mapsto t]t')$$

New Evaluation Rules

$$\frac{}{\mathsf{fix}\, (\lambda f : T_1.\, t_2) \longrightarrow [f \mapsto (\mathsf{fix}\, (\lambda f : T_1.\, t_2))]t_2}\ \text{(E-FixBeta)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{fix}\, t_1 \longrightarrow \mathsf{fix}\, t_1'}\ \text{(E-Fix)}$$

New Typing Rules

$$\frac{\Gamma \vdash t_1 : T_1 \to T_1}{\Gamma \vdash \mathsf{fix}\, t_1 : T_1}\ \text{(T-Fix)}$$